Databases

Theoretical Introduction

Contents

Chapter 1

Databases

1.1 Database

"Database Software" redirects here. For the computer program, see Europress.

A **database** is an organized collection of data.[1] It is the collection of schemas, tables, queries, reports, views and other objects. The data is typically organized to model aspects of reality in a way that supports processes requiring information, such as modelling the availability of rooms in hotels in a way that supports finding a hotel with vacancies.

A **database management system** (**DBMS**) is a computer software application that interacts with the user, other applications, and the database itself to capture and analyze data. A general-purpose DBMS is designed to allow the definition, creation, querying, update, and administration of databases. Well-known DBMSs include MySQL, PostgreSQL, Microsoft SQL Server, Oracle, Sybase and IBM DB2. A database is not generally portable across different DBMSs, but different DBMS can interoperate by using standards such as SQL and ODBC or JDBC to allow a single application to work with more than one DBMS. Database management systems are often classified according to the database model that they support; the most popular database systems since the 1980s have all supported the relational model as represented by the SQL language. Sometimes a DBMS is loosely referred to as a 'database'.

1.1.1 Terminology and overview

Formally, a "database" refers to a set of related data and the way it is organized. Access to this data is usually provided by a "database management system" (DBMS) consisting of an integrated set of computer software that allows users to interact with one or more databases and provides access to all of the data contained in the database (although restrictions may exist that limit access to particular data). The DBMS provides various functions that allow entry, storage and retrieval of large quantities of information and provides ways to manage how that information is organized.

Because of the close relationship between them, the term "database" is often used casually to refer to both a database and the DBMS used to manipulate it.

Outside the world of professional information technology, the term *database* is often used to refer to any collection of related data (such as a spreadsheet or a card index). This article is concerned only with databases where the size and usage requirements necessitate use of a database management system.[2]

Existing DBMSs provide various functions that allow management of a database and its data which can be classified into four main functional groups:

- Data definition – Creation, modification and removal of definitions that define the organization of the data.
- Update – Insertion, modification, and deletion of the actual data.[3]
- Retrieval – Providing information in a form directly usable or for further processing by other applications. The retrieved data may be made available in a form basically the same as it is stored in the database or in a new form obtained by altering or combining existing data from the database.[4]
- Administration – Registering and monitoring users, enforcing data security, monitoring performance, maintaining data integrity, dealing with concurrency control, and recovering information that has been corrupted by some event such as an unexpected system failure.[5]

Both a database and its DBMS conform to the principles of a particular database model.[6] "Database system" refers collectively to the database model, database management system, and database.[7]

Physically, database servers are dedicated computers that hold the actual databases and run only the DBMS and related software. Database servers are usually multiprocessor computers, with generous memory and RAID disk arrays used for stable storage. RAID is used for recovery of data if any of the disks fail. Hardware database accelerators, connected to one or more servers via a high-speed channel, are also used in large volume

transaction processing environments. DBMSs are found at the heart of most database applications. DBMSs may be built around a custom multitasking kernel with built-in networking support, but modern DBMSs typically rely on a standard operating system to provide these functions. Since DBMSs comprise a significant economical market, computer and storage vendors often take into account DBMS requirements in their own development plans.

Databases and DBMSs can be categorized according to the database model(s) that they support (such as relational or XML), the type(s) of computer they run on (from a server cluster to a mobile phone), the query language(s) used to access the database (such as SQL or XQuery), and their internal engineering, which affects performance, scalability, resilience, and security.

1.1.2 Applications

Databases are used to support internal operations of organizations and to underpin online interactions with customers and suppliers (see Enterprise software).

Databases are used to hold administrative information and more specialized data, such as engineering data or economic models. Examples of database applications include computerized library systems, flight reservation systems, computerized parts inventory systems, and many content management systems that store websites as collections of webpages in a database.

1.1.3 General-purpose and special-purpose DBMSs

A DBMS has evolved into a complex software system and its development typically requires thousands of person-years of development effort.[8] Some general-purpose DBMSs such as Adabas, Oracle and DB2 have been undergoing upgrades since the 1970s. General-purpose DBMSs aim to meet the needs of as many applications as possible, which adds to the complexity. However, the fact that their development cost can be spread over a large number of users means that they are often the most cost-effective approach. However, a general-purpose DBMS is not always the optimal solution: in some cases a general-purpose DBMS may introduce unnecessary overhead. Therefore, there are many examples of systems that use special-purpose databases. A common example is an email system that performs many of the functions of a general-purpose DBMS such as the insertion and deletion of messages composed of various items of data or associating messages with a particular email address; but these functions are limited to what is required to handle email and don't provide the user with the all of the functionality that would be available using a general-purpose DBMS.

Many other databases have application software that accesses the database on behalf of end-users, without exposing the DBMS interface directly. Application programmers may use a wire protocol directly, or more likely through an application programming interface. Database designers and database administrators interact with the DBMS through dedicated interfaces to build and maintain the applications' databases, and thus need some more knowledge and understanding about how DBMSs operate and the DBMSs' external interfaces and tuning parameters.

1.1.4 History

Following the technology progress in the areas of processors, computer memory, computer storage and computer networks, the sizes, capabilities, and performance of databases and their respective DBMSs have grown in orders of magnitude. The development of database technology can be divided into three eras based on data model or structure: navigational,[9] SQL/relational, and post-relational.

The two main early navigational data models were the hierarchical model, epitomized by IBM's IMS system, and the CODASYL model (network model), implemented in a number of products such as IDMS.

The relational model, first proposed in 1970 by Edgar F. Codd, departed from this tradition by insisting that applications should search for data by content, rather than by following links. The relational model employs sets of ledger-style tables, each used for a different type of entity. Only in the mid-1980s did computing hardware become powerful enough to allow the wide deployment of relational systems (DBMSs plus applications). By the early 1990s, however, relational systems dominated in all large-scale data processing applications, and as of 2015 they remain dominant : IBM DB2, Oracle, MySQL and Microsoft SQL Server are the top DBMS.[10] The dominant database language, standardised SQL for the relational model, has influenced database languages for other data models.

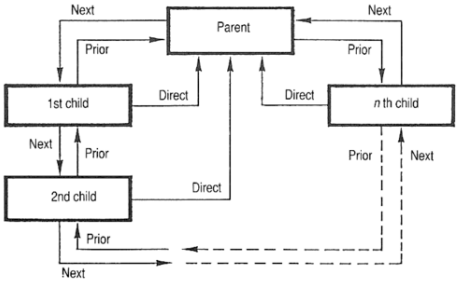
Object databases were developed in the 1980s to overcome the inconvenience of object-relational impedance mismatch, which led to the coining of the term "post-relational" and also the development of hybrid object-relational databases.

The next generation of post-relational databases in the late 2000s became known as NoSQL databases, introducing fast key-value stores and document-oriented databases. A competing "next generation" known as NewSQL databases attempted new implementations that retained the relational/SQL model while aiming to match the high performance of NoSQL compared to commercially available relational DBMSs.

1960s, navigational DBMS

Further information: Navigational database

The introduction of the term *database* coincided with the availability of direct-access storage (disks and drums) from the mid-1960s onwards. The term represented a contrast with the tape-based systems of the past, allowing shared interactive use rather than daily batch processing. The Oxford English Dictionary cites[11] a 1962 report by the System Development Corporation of California as the first to use the term "data-base" in a specific technical sense.



A closed chain of records in a navigational database model (e.g. CODASYL), with ***next pointers***, ***prior pointers*** and ***direct pointers*** provided by keys in the various records.

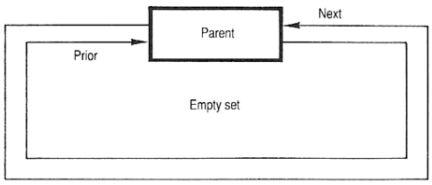


Illustration of an ***empty set***

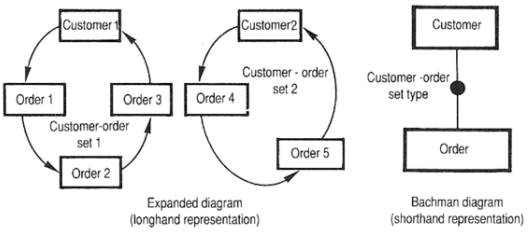


Illustration of a set type using a ***Bachman diagram***

The record set, basic structure of navigational (e.g. CODASYL) databse model. A set consists of one parent record (also called "the owner"), and n child records (also called members records)

Basic structure of navigational CODASYL database model

As computers grew in speed and capability, a number of general-purpose database systems emerged; by the mid-1960s a number of such systems had come into commercial use. Interest in a standard began to grow, and Charles Bachman, author of one such product, the Integrated Data Store (IDS), founded the "Database Task Group" within CODASYL, the group responsible for the creation and standardization of COBOL. In 1971 the Database Task Group delivered their standard, which generally became known as the "CODASYL approach", and soon a number of commercial products based on this approach entered the market.

The CODASYL approach relied on the "manual" navigation of a linked data set which was formed into a large network. Applications could find records by one of three methods:

1. Use of a primary key (known as a CALC key, typically implemented by hashing)
2. Navigating relationships (called sets) from one record to another
3. Scanning all the records in a sequential order

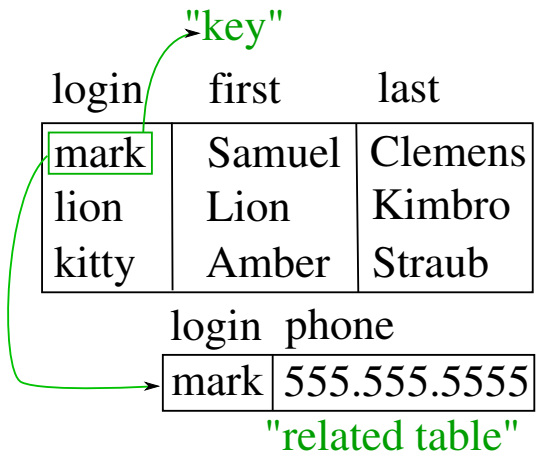
Later systems added B-trees to provide alternate access paths. Many CODASYL databases also added a very straightforward query language. However, in the final tally, CODASYL was very complex and required significant training and effort to produce useful applications.

IBM also had their own DBMS in 1968, known as Information Management System (IMS). IMS was a development of software written for the Apollo program on the System/360. IMS was generally similar in concept to CODASYL, but used a strict hierarchy for its model of data navigation instead of CODASYL's network model. Both concepts later became known as navigational databases due to the way data was accessed, and Bachman's 1973 Turing Award presentation was *The Programmer as Navigator*. IMS is classified as a hierarchical database. IDMS and Cincom Systems' TOTAL database are classified as network databases. IMS remains in use as of 2014.[12]

1970s, relational DBMS

Edgar Codd worked at IBM in San Jose, California, in one of their offshoot offices that was primarily involved in the development of hard disk systems. He was unhappy with the navigational model of the CODASYL approach, notably the lack of a "search" facility. In 1970, he wrote a number of papers that outlined a new approach to database construction that eventually culminated in the groundbreaking *A Relational Model of Data for Large Shared Data Banks*.[13]

In this paper, he described a new system for storing and working with large databases. Instead of records being stored in some sort of linked list of free-form records as in CODASYL, Codd's idea was to use a "table" of fixed-length records, with each table used for a different type of entity. A linked-list system would be very inefficient when storing "sparse" databases where some of the data for any one record could be left empty. The relational model solved this by splitting the data into a series of normalized tables (or *relations*), with optional elements being moved out of the main table to where they would take up room only if needed. Data may be freely inserted, deleted and edited in these tables, with the DBMS doing whatever maintenance needed to present a table view to the application/user.



In the relational model, records are "linked" using virtual keys not stored in the database but defined as needed between the data contained in the records.

The relational model also allowed the content of the database to evolve without constant rewriting of links and pointers. The relational part comes from entities referencing other entities in what is known as one-to-many relationship, like a traditional hierarchical model, and many-to-many relationship, like a navigational (network) model. Thus, a relational model can express both hierarchical and navigational models, as well as its native tabular model, allowing for pure or combined modeling in terms of these three models, as the application requires.

For instance, a common use of a database system is to track information about users, their name, login information, various addresses and phone numbers. In the navigational approach all of these data would be placed in a single record, and unused items would simply not be placed in the database. In the relational approach, the data would be *normalized* into a user table, an address table and a phone number table (for instance). Records would be created in these optional tables only if the address or phone numbers were actually provided.

Linking the information back together is the key to this system. In the relational model, some bit of information was used as a "key", uniquely defining a particular record. When information was being collected about a user, information stored in the optional tables would be found by searching for this key. For instance, if the login name of a user is unique, addresses and phone numbers for that user would be recorded with the login name as its key. This simple "re-linking" of related data back into a single collection is something that traditional computer languages are not designed for.

Just as the navigational approach would require programs to loop in order to collect records, the relational approach would require loops to collect information about any *one* record. Codd's solution to the necessary looping was a set-oriented language, a suggestion that would later spawn the ubiquitous SQL. Using a branch of mathematics known as tuple calculus, he demonstrated that such a system could support all the operations of normal databases (inserting, updating etc.) as well as providing a simple system for finding and returning *sets* of data in a single operation.

Codd's paper was picked up by two people at Berkeley, Eugene Wong and Michael Stonebraker. They started a project known as INGRES using funding that had already been allocated for a geographical database project and student programmers to produce code. Beginning in 1973, INGRES delivered its first test products which were generally ready for widespread use in 1979. INGRES was similar to System R in a number of ways, including the use of a "language" for data access, known as QUEL. Over time, INGRES moved to the emerging SQL standard.

IBM itself did one test implementation of the relational model, PRTV, and a production one, Business System 12, both now discontinued. Honeywell wrote MRDS for Multics, and now there are two new implementations: Alphora Dataphor and Rel. Most other DBMS implementations usually called *relational* are actually SQL DBMSs.

In 1970, the University of Michigan began development of the MICRO Information Management System[14] based on D.L. Childs' Set-Theoretic Data model.[15][16][17] Micro was used to manage very large data sets by the US Department of Labor, the U.S. Environmental Protection Agency, and researchers from the University of Alberta, the University of Michigan, and Wayne State University. It ran on IBM mainframe computers using the Michigan Terminal System.[18] The system remained in production until 1998.

Integrated approach

Main article: Database machine

In the 1970s and 1980s attempts were made to build database systems with integrated hardware and software. The underlying philosophy was that such integration would provide higher performance at lower cost. Examples were IBM System/38, the early offering of Teradata, and the Britton Lee, Inc. database machine.

Another approach to hardware support for database management was ICL's CAFS accelerator, a hardware disk controller with programmable search capabilities. In the long term, these efforts were generally unsuccessful because specialized database machines could not keep pace with the rapid development and progress of general-purpose computers. Thus most database systems nowadays are software systems running on general-purpose hardware, using general-purpose computer data storage. However this idea is still pursued for certain applications

by some companies like Netezza and Oracle (Exadata).

Late 1970s, SQL DBMS

IBM started working on a prototype system loosely based on Codd's concepts as *System R* in the early 1970s. The first version was ready in 1974/5, and work then started on multi-table systems in which the data could be split so that all of the data for a record (some of which is optional) did not have to be stored in a single large "chunk". Subsequent multi-user versions were tested by customers in 1978 and 1979, by which time a standardized query language – SQL – had been added. Codd's ideas were establishing themselves as both workable and superior to CODASYL, pushing IBM to develop a true production version of System R, known as *SQL/DS*, and, later, *Database 2* (DB2).

Larry Ellison's Oracle started from a different chain, based on IBM's papers on System R, and beat IBM to market when the first version was released in 1978.

Stonebraker went on to apply the lessons from INGRES to develop a new database, Postgres, which is now known as PostgreSQL. PostgreSQL is often used for global mission critical applications (the .org and .info domain name registries use it as their primary data store, as do many large companies and financial institutions).

In Sweden, Codd's paper was also read and Mimer SQL was developed from the mid-1970s at Uppsala University. In 1984, this project was consolidated into an independent enterprise. In the early 1980s, Mimer introduced transaction handling for high robustness in applications, an idea that was subsequently implemented on most other DBMSs.

Another data model, the entity–relationship model, emerged in 1976 and gained popularity for database design as it emphasized a more familiar description than the earlier relational model. Later on, entity–relationship constructs were retrofitted as a data modeling construct for the relational model, and the difference between the two have become irrelevant.

1980s, on the desktop

The 1980s ushered in the age of desktop computing. The new computers empowered their users with spreadsheets like Lotus 1-2-3 and database software like dBASE. The dBASE product was lightweight and easy for any computer user to understand out of the box. C. Wayne Ratliff the creator of dBASE stated: "dBASE was different from programs like BASIC, C, FORTRAN, and COBOL in that a lot of the dirty work had already been done. The data manipulation is done by dBASE instead of by the user, so the user can concentrate on what he is doing, rather than having to mess with the dirty details of opening, reading, and closing files, and managing space allocation."[19] dBASE was one of the top selling software titles in the 1980s and early 1990s.

1990s, object-oriented

The 1990s, along with a rise in object-oriented programming, saw a growth in how data in various databases were handled. Programmers and designers began to treat the data in their databases as objects. That is to say that if a person's data were in a database, that person's attributes, such as their address, phone number, and age, were now considered to belong to that person instead of being extraneous data. This allows for relations between data to be relations to objects and their attributes and not to individual fields.[20] The term "object-relational impedance mismatch" described the inconvenience of translating between programmed objects and database tables. Object databases and object-relational databases attempt to solve this problem by providing an object-oriented language (sometimes as extensions to SQL) that programmers can use as alternative to purely relational SQL. On the programming side, libraries known as object-relational mappings (ORMs) attempt to solve the same problem.

2000s, NoSQL and NewSQL

Main articles: NoSQL and NewSQL

The next generation of post-relational databases in the 2000s became known as NoSQL databases, including fast key-value stores and document-oriented databases.

XML databases are a type of structured document-oriented database that allows querying based on XML document attributes. XML databases are mostly used in enterprise database management, where XML is being used as the machine-to-machine data interoperability standard. XML database management systems include commercial software MarkLogic and Oracle Berkeley DB XML, and a free use software Clusterpoint Distributed XML/JSON Database. All are enterprise software database platforms and support industry standard ACID-compliant transaction processing with strong database consistency characteristics and high level of database security.[21][22][23]

NoSQL databases are often very fast, do not require fixed table schemas, avoid join operations by storing denormalized data, and are designed to scale horizontally. The most popular NoSQL systems include MongoDB, Couchbase, Riak, Memcached, Redis, CouchDB, Hazelcast, Apache Cassandra and HBase,[24] which are all open-source software products.

In recent years there was a high demand for massively distributed databases with high partition tolerance but according to the CAP theorem it is impossible for a distributed system to simultaneously provide consistency,

availability and partition tolerance guarantees. A distributed system can satisfy any two of these guarantees at the same time, but not all three. For that reason many NoSQL databases are using what is called eventual consistency to provide both availability and partition tolerance guarantees with a reduced level of data consistency.

NewSQL is a class of modern relational databases that aims to provide the same scalable performance of NoSQL systems for online transaction processing (read-write) workloads while still using SQL and maintaining the ACID guarantees of a traditional database system. Such databases include ScaleBase, Clustrix, EnterpriseDB, MemSQL, NuoDB[25] and VoltDB.

1.1.5 Research

Database technology has been an active research topic since the 1960s, both in academia and in the research and development groups of companies (for example IBM Research). Research activity includes theory and development of prototypes. Notable research topics have included models, the atomic transaction concept and related concurrency control techniques, query languages and query optimization methods, RAID, and more.

The database research area has several dedicated academic journals (for example, *ACM Transactions on Database Systems*-TODS, *Data and Knowledge Engineering*-DKE) and annual conferences (e.g., ACM SIGMOD, ACM PODS, VLDB, IEEE ICDE).

1.1.6 Examples

One way to classify databases involves the type of their contents, for example: bibliographic, document-text, statistical, or multimedia objects. Another way is by their application area, for example: accounting, music compositions, movies, banking, manufacturing, or insurance. A third way is by some technical aspect, such as the database structure or interface type. This section lists a few of the adjectives used to characterize different kinds of databases.

- An in-memory database is a database that primarily resides in main memory, but is typically backed-up by non-volatile computer data storage. Main memory databases are faster than disk databases, and so are often used where response time is critical, such as in telecommunications network equipment.[26] SAP HANA platform is a very hot topic for in-memory database. By May 2012, HANA was able to run on servers with 100TB main memory powered by IBM. The co founder of the company claimed that the system was big enough to run the 8 largest SAP customers.
- An active database includes an event-driven architecture which can respond to conditions both inside and outside the database. Possible uses include security monitoring, alerting, statistics gathering and authorization. Many databases provide active database features in the form of database triggers.
- A cloud database relies on cloud technology. Both the database and most of its DBMS reside remotely, "in the cloud", while its applications are both developed by programmers and later maintained and utilized by (application's) end-users through a web browser and Open APIs.
- Data warehouses archive data from operational databases and often from external sources such as market research firms. The warehouse becomes the central source of data for use by managers and other end-users who may not have access to operational data. For example, sales data might be aggregated to weekly totals and converted from internal product codes to use UPCs so that they can be compared with ACNielsen data. Some basic and essential components of data warehousing include extracting, analyzing, and mining data, transforming, loading and managing data so as to make them available for further use.
- A deductive database combines logic programming with a relational database, for example by using the Datalog language.
- A distributed database is one in which both the data and the DBMS span multiple computers.
- A document-oriented database is designed for storing, retrieving, and managing document-oriented, or semi structured data, information. Document-oriented databases are one of the main categories of NoSQL databases.
- An embedded database system is a DBMS which is tightly integrated with an application software that requires access to stored data in such a way that the DBMS is hidden from the application's end-users and requires little or no ongoing maintenance.[27]
- **End-user databases** consist of data developed by individual end-users. Examples of these are collections of documents, spreadsheets, presentations, multimedia, and other files. Several products exist to support such databases. Some of them are much simpler than full-fledged DBMSs, with more elementary DBMS functionality.
- A federated database system comprises several distinct databases, each with its own DBMS. It is handled as a single database by a federated database management system (FDBMS), which transparently integrates multiple autonomous DBMSs, possibly of different types (in which case it would also be a heterogeneous database system), and provides them with an integrated conceptual view.

- Sometimes the term *multi-database* is used as a synonym to federated database, though it may refer to a less integrated (e.g., without an FDBMS and a managed integrated schema) group of databases that cooperate in a single application. In this case typically middleware is used for distribution, which typically includes an atomic commit protocol (ACP), e.g., the two-phase commit protocol, to allow distributed (global) transactions across the participating databases.

- A graph database is a kind of NoSQL database that uses graph structures with nodes, edges, and properties to represent and store information. General graph databases that can store any graph are distinct from specialized graph databases such as triplestores and network databases.

- An array DBMS is a kind of NoSQL DBMS that allows to model, store, and retrieve (usually large) multi-dimensional arrays such as satellite images and climate simulation output.

- In a hypertext or hypermedia database, any word or a piece of text representing an object, e.g., another piece of text, an article, a picture, or a film, can be hyperlinked to that object. Hypertext databases are particularly useful for organizing large amounts of disparate information. For example, they are useful for organizing online encyclopedias, where users can conveniently jump around the text. The World Wide Web is thus a large distributed hypertext database.

- A knowledge base (abbreviated **KB**, **kb** or Δ[28][29]) is a special kind of database for knowledge management, providing the means for the computerized collection, organization, and retrieval of knowledge. Also a collection of data representing problems with their solutions and related experiences.

- A mobile database can be carried on or synchronized from a mobile computing device.

- Operational databases store detailed data about the operations of an organization. They typically process relatively high volumes of updates using transactions. Examples include customer databases that record contact, credit, and demographic information about a business' customers, personnel databases that hold information such as salary, benefits, skills data about employees, enterprise resource planning systems that record details about product components, parts inventory, and financial databases that keep track of the organization's money, accounting and financial dealings.

- A parallel database seeks to improve performance through parallelization for tasks such as loading data, building indexes and evaluating queries.

  The major parallel DBMS architectures which are induced by the underlying hardware architecture are:

  - **Shared memory architecture**, where multiple processors share the main memory space, as well as other data storage.
  - **Shared disk architecture**, where each processing unit (typically consisting of multiple processors) has its own main memory, but all units share the other storage.
  - **Shared nothing architecture**, where each processing unit has its own main memory and other storage.

- Probabilistic databases employ fuzzy logic to draw inferences from imprecise data.

- Real-time databases process transactions fast enough for the result to come back and be acted on right away.

- A spatial database can store the data with multidimensional features. The queries on such data include location based queries, like "Where is the closest hotel in my area?".

- A temporal database has built-in time aspects, for example a temporal data model and a temporal version of SQL. More specifically the temporal aspects usually include valid-time and transaction-time.

- A terminology-oriented database builds upon an object-oriented database, often customized for a specific field.

- An unstructured data database is intended to store in a manageable and protected way diverse objects that do not fit naturally and conveniently in common databases. It may include email messages, documents, journals, multimedia objects, etc. The name may be misleading since some objects can be highly structured. However, the entire possible object collection does not fit into a predefined structured framework. Most established DBMSs now support unstructured data in various ways, and new dedicated DBMSs are emerging.

1.1.7 Design and modeling

Main article: Database design

The first task of a database designer is to produce a conceptual data model that reflects the structure of the

information to be held in the database. A common approach to this is to develop an entity-relationship model, often with the aid of drawing tools. Another popular approach is the Unified Modeling Language. A successful data model will accurately reflect the possible state of the external world being modeled: for example, if people can have more than one phone number, it will allow this information to be captured. Designing a good conceptual data model requires a good understanding of the application domain; it typically involves asking deep questions about the things of interest to an organisation, like "can a customer also be a supplier?", or "if a product is sold with two different forms of packaging, are those the same product or different products?", or "if a plane flies from New York to Dubai via Frankfurt, is that one flight or two (or maybe even three)?". The answers to these questions establish definitions of the terminology used for entities (customers, products, flights, flight segments) and their relationships and attributes.

Producing the conceptual data model sometimes involves input from business processes, or the analysis of workflow in the organization. This can help to establish what information is needed in the database, and what can be left out. For example, it can help when deciding whether the database needs to hold historic data as well as current data.

Having produced a conceptual data model that users are happy with, the next stage is to translate this into a schema that implements the relevant data structures within the database. This process is often called logical database design, and the output is a logical data model expressed in the form of a schema. Whereas the conceptual data model is (in theory at least) independent of the choice of database technology, the logical data model will be expressed in terms of a particular database model supported by the chosen DBMS. (The terms *data model* and *database model* are often used interchangeably, but in this article we use *data model* for the design of a specific database, and *database model* for the modelling notation used to express that design.)

The most popular database model for general-purpose databases is the relational model, or more precisely, the relational model as represented by the SQL language. The process of creating a logical database design using this model uses a methodical approach known as normalization. The goal of normalization is to ensure that each elementary "fact" is only recorded in one place, so that insertions, updates, and deletions automatically maintain consistency.

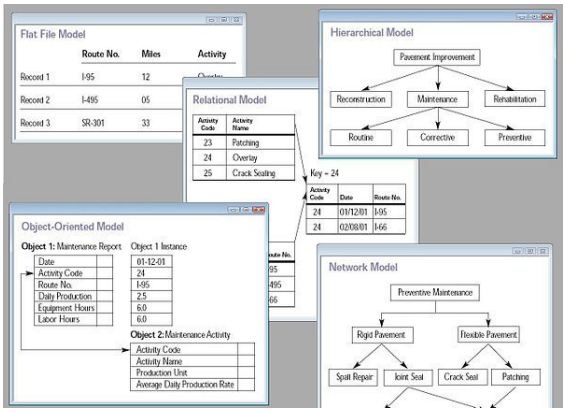
The final stage of database design is to make the decisions that affect performance, scalability, recovery, security, and the like. This is often called *physical database design*. A key goal during this stage is data independence, meaning that the decisions made for performance optimization purposes should be invisible to end-users and applications. Physical design is driven mainly by performance requirements, and requires a good knowledge of the expected workload and access patterns, and a deep understanding of the features offered by the chosen DBMS.

Another aspect of physical database design is security. It involves both defining access control to database objects as well as defining security levels and methods for the data itself.

Models

Main article: Database model

A database model is a type of data model that determines the logical structure of a database and fundamentally determines in which manner data can be stored, organized, and manipulated. The most popular example of a database model is the relational model (or the SQL approximation of relational), which uses a table-based format.



Collage of five types of database models

Common logical data models for databases include:

- Navigational databases
  - Hierarchical database model
  - Network model
  - Graph database
- Relational model
- Entity–relationship model
  - Enhanced entity–relationship model
- Object model
- Document model
- Entity–attribute–value model
- Star schema

An object-relational database combines the two related structures.

Physical data models include:

- Inverted index
- Flat file

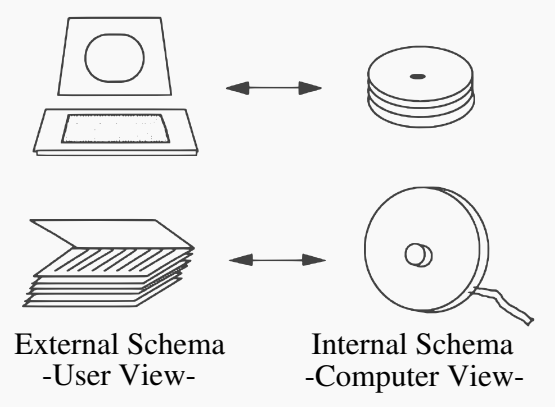
Other models include:

- Associative model
- Multidimensional model
- Array model
- Multivalue model

Specialized models are optimized for particular types of data:

- XML database
- Semantic model
- Content store
- Event store
- Time series model

External, conceptual, and internal views



Traditional view of data[30]

A database management system provides three views of the database data:

- The **external level** defines how each group of end-users sees the organization of data in the database. A single database can have any number of views at the external level.
- The **conceptual level** unifies the various external views into a compatible global view.[31] It provides the synthesis of all the external views. It is out of the scope of the various database end-users, and is rather of interest to database application developers and database administrators.
- The **internal level** (or *physical level*) is the internal organization of data inside a DBMS. It is concerned with cost, performance, scalability and other operational matters. It deals with storage layout of the data, using storage structures such as indexes to enhance performance. Occasionally it stores data of individual views (materialized views), computed from generic data, if performance justification exists for such redundancy. It balances all the external views' performance requirements, possibly conflicting, in an attempt to optimize overall performance across all activities.

While there is typically only one conceptual (or logical) and physical (or internal) view of the data, there can be any number of different external views. This allows users to see database information in a more business-related way rather than from a technical, processing viewpoint. For example, a financial department of a company needs the payment details of all employees as part of the company's expenses, but does not need details about employees that are the interest of the human resources department. Thus different departments need different *views* of the company's database.

The three-level database architecture relates to the concept of *data independence* which was one of the major initial driving forces of the relational model. The idea is that changes made at a certain level do not affect the view at a higher level. For example, changes in the internal level do not affect application programs written using conceptual level interfaces, which reduces the impact of making physical changes to improve performance.

The conceptual view provides a level of indirection between internal and external. On one hand it provides a common view of the database, independent of different external view structures, and on the other hand it abstracts away details of how the data is stored or managed (internal level). In principle every level, and even every external view, can be presented by a different data model. In practice usually a given DBMS uses the same data model for both the external and the conceptual levels (e.g., relational model). The internal level, which is hidden inside the DBMS and depends on its implementation, requires a different level of detail and uses its own types of data structure types.

Separating the *external*, *conceptual* and *internal* levels was a major feature of the relational database model implementations that dominate 21st century databases.[31]

1.1.8 Languages

Database languages are special-purpose languages, which do one or more of the following:

- Data definition language – defines data types and the relationships among them

- Data manipulation language – performs tasks such as inserting, updating, or deleting data occurrences
- Query language – allows searching for information and computing derived information

Database languages are specific to a particular data model. Notable examples include:

- SQL combines the roles of data definition, data manipulation, and query in a single language. It was one of the first commercial languages for the relational model, although it departs in some respects from the relational model as described by Codd (for example, the rows and columns of a table can be ordered). SQL became a standard of the American National Standards Institute (ANSI) in 1986, and of the International Organization for Standardization (ISO) in 1987. The standards have been regularly enhanced since and is supported (with varying degrees of conformance) by all mainstream commercial relational DBMSs.[32][33]
- OQL is an object model language standard (from the Object Data Management Group). It has influenced the design of some of the newer query languages like JDOQL and EJB QL.
- XQuery is a standard XML query language implemented by XML database systems such as MarkLogic and eXist, by relational databases with XML capability such as Oracle and DB2, and also by in-memory XML processors such as Saxon.
- SQL/XML combines XQuery with SQL.[34]

A database language may also incorporate features like:

- DBMS-specific Configuration and storage engine management
- Computations to modify query results, like counting, summing, averaging, sorting, grouping, and cross-referencing
- Constraint enforcement (e.g. in an automotive database, only allowing one engine type per car)
- Application programming interface version of the query language, for programmer convenience

1.1.9 Performance, security, and availability

Because of the critical importance of database technology to the smooth running of an enterprise, database systems include complex mechanisms to deliver the required performance, security, and availability, and allow database administrators to control the use of these features.

Storage

Main articles: Computer data storage and Database engine

Database storage is the container of the physical materialization of a database. It comprises the *internal* (physical) *level* in the database architecture. It also contains all the information needed (e.g., metadata, "data about the data", and internal data structures) to reconstruct the *conceptual level* and *external level* from the internal level when needed. Putting data into permanent storage is generally the responsibility of the database engine a.k.a. "storage engine". Though typically accessed by a DBMS through the underlying operating system (and often utilizing the operating systems' file systems as intermediates for storage layout), storage properties and configuration setting are extremely important for the efficient operation of the DBMS, and thus are closely maintained by database administrators. A DBMS, while in operation, always has its database residing in several types of storage (e.g., memory and external storage). The database data and the additional needed information, possibly in very large amounts, are coded into bits. Data typically reside in the storage in structures that look completely different from the way the data look in the conceptual and external levels, but in ways that attempt to optimize (the best possible) these levels' reconstruction when needed by users and programs, as well as for computing additional types of needed information from the data (e.g., when querying the database).

Some DBMSs support specifying which character encoding was used to store data, so multiple encodings can be used in the same database.

Various low-level database storage structures are used by the storage engine to serialize the data model so it can be written to the medium of choice. Techniques such as indexing may be used to improve performance. Conventional storage is row-oriented, but there are also column-oriented and correlation databases.

Materialized views Main article: Materialized view

Often storage redundancy is employed to increase performance. A common example is storing *materialized views*, which consist of frequently needed *external views* or query results. Storing such views saves the expensive computing of them each time they are needed. The downsides of materialized views are the overhead incurred when updating them to keep them synchronized with their original updated database data, and the cost of storage redundancy.

Replication Main article: Database replication

Occasionally a database employs storage redundancy by database objects replication (with one or more copies) to increase data availability (both to improve performance of simultaneous multiple end-user accesses to a same database object, and to provide resiliency in a case of partial failure of a distributed database). Updates of a replicated object need to be synchronized across the object copies. In many cases the entire database is replicated.

Security

Main article: Database security

Database security deals with all various aspects of protecting the database content, its owners, and its users. It ranges from protection from intentional unauthorized database uses to unintentional database accesses by unauthorized entities (e.g., a person or a computer program).

Database access control deals with controlling who (a person or a certain computer program) is allowed to access what information in the database. The information may comprise specific database objects (e.g., record types, specific records, data structures), certain computations over certain objects (e.g., query types, or specific queries), or utilizing specific access paths to the former (e.g., using specific indexes or other data structures to access information). Database access controls are set by special authorized (by the database owner) personnel that uses dedicated protected security DBMS interfaces.

This may be managed directly on an individual basis, or by the assignment of individuals and privileges to groups, or (in the most elaborate models) through the assignment of individuals and groups to roles which are then granted entitlements. Data security prevents unauthorized users from viewing or updating the database. Using passwords, users are allowed access to the entire database or subsets of it called "subschemas". For example, an employee database can contain all the data about an individual employee, but one group of users may be authorized to view only payroll data, while others are allowed access to only work history and medical data. If the DBMS provides a way to interactively enter and update the database, as well as interrogate it, this capability allows for managing personal databases.

Data security in general deals with protecting specific chunks of data, both physically (i.e., from corruption, or destruction, or removal; e.g., see physical security), or the interpretation of them, or parts of them to meaningful information (e.g., by looking at the strings of bits that they comprise, concluding specific valid credit-card numbers; e.g., see data encryption).

Change and access logging records who accessed which attributes, what was changed, and when it was changed. Logging services allow for a forensic database audit later by keeping a record of access occurrences and changes. Sometimes application-level code is used to record changes rather than leaving this to the database. Monitoring can be set up to attempt to detect security breaches.

Transactions and concurrency

Further information: Concurrency control

Database transactions can be used to introduce some level of fault tolerance and data integrity after recovery from a crash. A database transaction is a unit of work, typically encapsulating a number of operations over a database (e.g., reading a database object, writing, acquiring lock, etc.), an abstraction supported in database and also other systems. Each transaction has well defined boundaries in terms of which program/code executions are included in that transaction (determined by the transaction's programmer via special transaction commands).

The acronym ACID describes some ideal properties of a database transaction: Atomicity, Consistency, Isolation, and Durability.

Migration

> *See also section Database migration in article Data migration*

A database built with one DBMS is not portable to another DBMS (i.e., the other DBMS cannot run it). However, in some situations it is desirable to move, migrate a database from one DBMS to another. The reasons are primarily economical (different DBMSs may have different total costs of ownership or TCOs), functional, and operational (different DBMSs may have different capabilities). The migration involves the database's transformation from one DBMS type to another. The transformation should maintain (if possible) the database related application (i.e., all related application programs) intact. Thus, the database's conceptual and external architectural levels should be maintained in the transformation. It may be desired that also some aspects of the architecture internal level are maintained. A complex or large database migration may be a complicated and costly (one-time) project by itself, which should be factored into the decision to migrate. This in spite of the fact that tools may exist to help migration between specific DBMSs. Typically a DBMS vendor provides tools to help importing databases from other popular DBMSs.

Building, maintaining, and tuning

Main article: Database tuning

After designing a database for an application, the next stage is building the database. Typically an appropriate general-purpose DBMS can be selected to be utilized for this purpose. A DBMS provides the needed user interfaces to be utilized by database administrators to define the needed application's data structures within the DBMS's respective data model. Other user interfaces are used to select needed DBMS parameters (like security related, storage allocation parameters, etc.).

When the database is ready (all its data structures and other needed components are defined) it is typically populated with initial application's data (database initialization, which is typically a distinct project; in many cases using specialized DBMS interfaces that support bulk insertion) before making it operational. In some cases the database becomes operational while empty of application data, and data is accumulated during its operation.

After the database is created, initialised and populated it needs to be maintained. Various database parameters may need changing and the database may need to be tuned (tuning) for better performance; application's data structures may be changed or added, new related application programs may be written to add to the application's functionality, etc.

Backup and restore

Main article: Backup

Sometimes it is desired to bring a database back to a previous state (for many reasons, e.g., cases when the database is found corrupted due to a software error, or if it has been updated with erroneous data). To achieve this a **backup** operation is done occasionally or continuously, where each desired database state (i.e., the values of its data and their embedding in database's data structures) is kept within dedicated backup files (many techniques exist to do this effectively). When this state is needed, i.e., when it is decided by a database administrator to bring the database back to this state (e.g., by specifying this state by a desired point in time when the database was in this state), these files are utilized to **restore** that state.

Static Analysis

Static analysis techniques for software verification can be applied also in the scenario of query languages. In particular, the *Abstract interpretation framework has been extended to the field of query languages for relational databases as a way to support sound approximation techniques.[35] The semantics of query languages can be tuned according to suitable abstractions of the concrete domain of data. The abstraction of relational database system has many interesting applications, in particular, for security purposes, such as fine grained access control, watermarking, etc.

Other

Other DBMS features might include:

- Database logs
- Graphics component for producing graphs and charts, especially in a data warehouse system
- **Query optimizer** – Performs query optimization on every query to choose for it the most efficient *query plan* (a partial order (tree) of operations) to be executed to compute the query result. May be specific to a particular storage engine.
- Tools or hooks for database design, application programming, application program maintenance, database performance analysis and monitoring, database configuration monitoring, DBMS hardware configuration (a DBMS and related database may span computers, networks, and storage units) and related database mapping (especially for a distributed DBMS), storage allocation and database layout monitoring, storage migration, etc.

1.1.10 See also

Main article: Outline of databases

- Comparison of database tools
- Comparison of object database management systems
- Comparison of object-relational database management systems
- Comparison of relational database management systems
- Data hierarchy
- Data bank
- Data store
- Database theory
- Database testing
- Database-centric architecture
- Question-focused dataset

1.1.11 References

[1] "Database - Definition of database by Merriam-Webster". *merriam-webster.com*.

[2] Jeffrey Ullman 1997: *First course in database systems*, Prentice–Hall Inc., Simon & Schuster, Page 1, ISBN 0-13-861337-0.

[3] "Update - Definition of update by Merriam-Webster". *merriam-webster.com*.

[4] "Retrieval - Definition of retrieval by Merriam-Webster". *merriam-webster.com*.

[5] "Administration - Definition of administration by Merriam-Webster". *merriam-webster.com*.

[6] Tsitchizris, D. C. and F. H. Lochovsky (1982). *Data Models.* Englewood-Cliffs, Prentice–Hall.

[7] Beynon-Davies P. (2004). *Database Systems* 3rd Edition. Palgrave, Basingstoke, UK. ISBN 1-4039-1601-2

[8] Raul F. Chong, Michael Dang, Dwaine R. Snow, Xiaomei Wang (3 July 2008). "Introduction to DB2". Retrieved 17 March 2013.. This article quotes a development time of 5 years involving 750 people for DB2 release 9 alone

[9] C. W. Bachmann (November 1973), "The Programmer as Navigator" (PDF), *CACM* (Turing Award Lecture 1973)

[10] "TOPDB Top Database index". *pypl.github.io*.

[11] "database, n". *OED Online*. Oxford University Press. June 2013. Retrieved July 12, 2013.

[12] IBM Corporation. "IBM Information Management System (IMS) 13 Transaction and Database Servers delivers high performance and low total cost of ownership". Retrieved Feb 20, 2014.

[13] Codd, E.F. (1970)."A Relational Model of Data for Large Shared Data Banks". In: *Communications of the ACM* 13 (6): 377–387.

[14] William Hershey and Carol Easthope, "A set theoretic data structure and retrieval language", Spring Joint Computer Conference, May 1972 in *ACM SIGIR Forum*, Volume 7, Issue 4 (December 1972), pp. 45–55, DOI=10.1145/1095495.1095500

[15] Ken North, "Sets, Data Models and Data Independence", *Dr. Dobb's*, 10 March 2010

[16] *Description of a set-theoretic data structure*, D. L. Childs, 1968, Technical Report 3 of the CONCOMP (Research in Conversational Use of Computers) Project, University of Michigan, Ann Arbor, Michigan, USA

[17] *Feasibility of a Set-Theoretic Data Structure : A General Structure Based on a Reconstituted Definition of Relation*, D. L. Childs, 1968, Technical Report 6 of the CONCOMP (Research in Conversational Use of Computers) Project, University of Michigan, Ann Arbor, Michigan, USA

[18] *MICRO Information Management System (Version 5.0) Reference Manual*, M.A. Kahn, D.L. Rumelhart, and B.L. Bronson, October 1977, Institute of Labor and Industrial Relations (ILIR), University of Michigan and Wayne State University

[19] Interview with Wayne Ratliff. The FoxPro History. Retrieved on 2013-07-12.

[20] Development of an object-oriented DBMS; Portland, Oregon, United States; Pages: 472 – 482; 1986; ISBN 0-89791-204-7

[21] "Oracle Berkeley DB XML" (PDF). Retrieved 10 March 2015.

[22] "ACID Transactions, MarkLogic". Retrieved 10 March 2015.

[23] "Clusterpoint Database at a Glance". Retrieved 10 March 2015.

[24] "DB-Engines Ranking". January 2013. Retrieved 22 January 2013.

[25] Proctor, Seth (2013). "Exploring the Architecture of the NuoDB Database, Part 1". Retrieved 2013-07-12.

[26] "TeleCommunication Systems Signs up as a Reseller of TimesTen; Mobile Operators and Carriers Gain Real-Time Platform for Location-Based Services". *Business Wire*. 2002-06-24.

[27] Graves, Steve. "COTS Databases For Embedded Systems", *Embedded Computing Design* magazine, January 2007. Retrieved on August 13, 2008.

[28] Argumentation in Artificial Intelligence by Iyad Rahwan, Guillermo R. Simari

[29] "OWL DL Semantics". Retrieved 10 December 2010.

[30] itl.nist.gov (1993) *Integration Definition for Information Modeling (IDEFIX)*. 21 December 1993.

[31] Date 1990, pp. 31–32

[32] Chapple, Mike. "SQL Fundamentals". *Databases*. About.com. Retrieved 2009-01-28.

[33] "Structured Query Language (SQL)". International Business Machines. October 27, 2006. Retrieved 2007-06-10.

[34] Wagner, Michael (2010), "1. Auflage", *SQL/XML:2006 – Evaluierung der Standardkonformität ausgewählter Datenbanksysteme*, Diplomica Verlag, ISBN 3-8366-9609-6

[35] R.Halder and A.Cortesi, Abstract Interpretation of Database Query Languages. COMPUTER LANGUAGES, SYSTEMS & STRUCTURES, vol. 38(2), pp. 123-−157, Elsevier Ed. (ISSN 1477-8424)

1.1.12 Further reading

- Ling Liu and Tamer M. Özsu (Eds.) (2009). "Encyclopedia of Database Systems, 4100 p. 60 illus. ISBN 978-0-387-49616-0.
- Beynon-Davies, P. (2004). Database Systems. 3rd Edition. Palgrave, Houndmills, Basingstoke.
- Connolly, Thomas and Carolyn Begg. *Database Systems.* New York: Harlow, 2002.

- Date, C. J. (2003). *An Introduction to Database Systems, Fifth Edition*. Addison Wesley. ISBN 0-201-51381-1.
- Gray, J. and Reuter, A. *Transaction Processing: Concepts and Techniques*, 1st edition, Morgan Kaufmann Publishers, 1992.
- Kroenke, David M. and David J. Auer. *Database Concepts.* 3rd ed. New York: Prentice, 2007.
- Raghu Ramakrishnan and Johannes Gehrke, *Database Management Systems*
- Abraham Silberschatz, Henry F. Korth, S. Sudarshan, *Database System Concepts*
- Discussion on database systems,
- Lightstone, S.; Teorey, T.; Nadeau, T. (2007). *Physical Database Design: the database professional's guide to exploiting indexes, views, storage, and more*. Morgan Kaufmann Press. ISBN 0-12-369389-6.
- Teorey, T.; Lightstone, S. and Nadeau, T. *Database Modeling & Design: Logical Design*, 4th edition, Morgan Kaufmann Press, 2005. ISBN 0-12-685352-5

1.1.13 External links

- Database at DMOZ
- DB File extension – informations about files with DB extension

1.2 Schema migration

Not to be confused with Data migration.

In software engineering, **schema migration** (also **database migration**, **database change management**[1][2]) refers to the management of incremental, reversible changes to relational database schemas. A schema migration is performed on a database whenever it is necessary to update or revert that database's schema to some newer or older version.

Migrations are performed programmatically by using a *schema migration tool*. When invoked with a specified desired schema version, the tool automates the successive application or reversal of an appropriate sequence of schema changes until it is brought to the desired state.

Most schema migration tools aim to minimise the impact of schema changes on any existing data in the database. Despite this, preservation of data in general is not guaranteed because schema changes such as the deletion of a database column can destroy data (i.e. all values stored under that column for all rows in that table are deleted). Instead, the tools help to preserve the meaning of the data or to reorganize existing data to meet new requirements. Since meaning of the data often cannot be encoded, the configuration of the tools usually needs manual intervention.

1.2.1 Risks and Benefits

Schema migration allows to fix mistakes and adapt the data as requirements change. They are an essential part of software evolution, especially in agile environments (see below).

Applying a schema migration to a production database is always a risk. Development and test databases tend to be smaller and cleaner. The data in them is better understood or, if everything else fails, the amount of data is small enough for a human to process. Production databases are usually huge, old and full of surprises. The surprises can come from many sources:

- Corrupt data that was written by old versions of the software and not cleaned properly
- Implied dependencies in the data which no one knows about anymore
- People directly changing the database without using the designated tools
- Bugs in the schema migration tools
- Mistakes in assumptions how data should be migrated

For these reasons, the migration process needs a high level of discipline, thorough testing and a sound backup strategy.

1.2.2 Schema migration in agile software development

When developing software applications backed by a database, developers typically develop the application source code in tandem with an evolving database schema. The code typically has rigid expectations of what columns, tables and constraints are present in the database schema whenever it needs to interact with one, so only the version of database schema against which the code was developed is considered fully compatible with that version of source code.

In software testing, while developers may mock the presence of a compatible database system for unit testing, any level of testing higher than this (e.g. integration testing or system testing) it is common for developers to test their application against a local or remote test database schematically compatible with the version of source code

under test. In advanced applications, the migration itself can be subject to migration testing.

With schema migration technology, data models no longer need to be fully designed up-front, and is more capable of being adapted with changing project requirements throughout the software development lifecycle.

Relation to revision control systems

Teams of software developers usually use version control systems to manage and collaborate on changes made to versions of source code. Different developers can develop on divergent, relatively older or newer branches of the same source code to make changes and additions during development.

Supposing that the software under development interacts with a database, every version of the source code can be associated with at least one database schema with which it is compatible.

Under good software testing practise, schema migrations can be performed on test databases to ensure that their schema is compatible to the source code. To streamline this process, a schema migration tool is usually invoked as a part of an automated software build as a prerequisite of the automated testing phase.

Schema migration tools can be said to solve versioning problems for database schemas just as version control systems solve versioning problems for source code. In practice, many schema migration tools actually rely on a textual representation of schema changes (such as files containing SQL statements) such that the version history of schema changes can effectively be stored alongside program source code within VCS. This approach ensures that the information necessary to recover a compatible database schema for a particular code branch is recoverable from the source tree itself. Another benefit of this approach is the handling of concurrent conflicting schema changes; developers may simply use their usual text-based conflict resolution tools to reconcile differences.

Relation to schema evolution

Schema migration tooling could be seen as a facility to track the history of an evolving schema.

Advantages

Developers no longer need to remove the entire test database in order to create a new test database from scratch (e.g. using schema creation scripts from DDL generation tools). Further, if generation of test data costs a lot of time, developers can avoid regenerating test data for small, non-destructive changes to the schema.

1.2.3 Available Tools

- Flyway - database migration tool (for Windows, OSX, Linux, Android and the JVM) where migrations are written in SQL or Java
- LiquiBase - cross platform tool where migrations are written in XML, YAML, JSON or SQL.
- Datical - Enterprise commercial version of Liquibase.
- Redgate SQL Compare - a schema comparison and deployment tool for SQL Server and Oracle.
- ReadyRoll - a migrations-based Visual Studio extension for SQL Server development and deployment.
- Active Record (Migrations) - schema migration tool for Ruby on Rails projects based on Active Record.
- Ruckusing-migrations - schema migration tool for PHP projects.
- Phinx - another framework-independent PHP migration tool.
- MyBatis Migrations - seeks to be the best migration tool of its kind.
- Ragtime - a SQL database schema migration library written in Clojure
- Lobos - a SQL database schema manipulation and migration library written in Clojure.
- Alembic - a lightweight database migration tool for usage with the SQLAlchemy Database Toolkit for Python.
- RoundhousE - a SQL database versioning and change management tool written in C#.
- XMigra - a SQL database evolution management tool written in Ruby that generates scripts without communicating with the database.
- DBmaestro - a database version control and schema migration solution for SQL Server and Oracle.
- DB Change Manager - Commercial Change Management Software by Embarcadero.
- Sqitch - Sqitch by Theory.

1.2.4 References

[1] http://www.liquibase.org/ Liquibase Database Refactoring

[2] http://flywaydb.org/ Flyway: The agile database migration framework for Java

1.3 Star schema

In computing, the **Star Schema** is the simplest style of data mart schema. The star schema consists of one or more fact tables referencing any number of dimension tables. The star schema is an important special case of the snowflake schema, and is more effective for handling simpler queries.[1]

The star schema gets its name from the physical model's[2] resemblance to a star shape with a fact table at its center and the dimension tables surrounding it representing the star's points.

1.3.1 Model

The star schema separates business process data into facts, which hold the measurable, quantitative data about a business, and dimensions which are descriptive attributes related to fact data. Examples of fact data include sales price, sale quantity, and time, distance, speed, and weight measurements. Related dimension attribute examples include product models, product colors, product sizes, geographic locations, and salesperson names.

A star schema that has many dimensions is sometimes called a *centipede schema*.[3] Having dimensions of only a few attributes, while simpler to maintain, results in queries with many table joins and makes the star schema less easy to use.

Fact tables

Fact tables record measurements or metrics for a specific event. Fact tables generally consist of numeric values, and foreign keys to dimensional data where descriptive information is kept.[3] Fact tables are designed to a low level of uniform detail (referred to as "granularity" or "grain"), meaning facts can record events at a very atomic level. This can result in the accumulation of a large number of records in a fact table over time. Fact tables are defined as one of three types:

- Transaction fact tables record facts about a specific event (e.g., sales events)
- Snapshot fact tables record facts at a given point in time (e.g., account details at month end)
- Accumulating snapshot tables record aggregate facts at a given point in time (e.g., total month-to-date sales for a product)

Fact tables are generally assigned a surrogate key to ensure each row can be uniquely identified.

Dimension tables

Dimension tables usually have a relatively small number of records compared to fact tables, but each record may have a very large number of attributes to describe the fact data. Dimensions can define a wide variety of characteristics, but some of the most common attributes defined by dimension tables include:

- Time dimension tables describe time at the lowest level of time granularity for which events are recorded in the star schema
- Geography dimension tables describe location data, such as country, state, or city
- Product dimension tables describe products
- Employee dimension tables describe employees, such as sales people
- Range dimension tables describe ranges of time, dollar values, or other measurable quantities to simplify reporting

Dimension tables are generally assigned a surrogate primary key, usually a single-column integer data type, mapped to the combination of dimension attributes that form the natural key.

1.3.2 Benefits

Star schemas are denormalized, meaning the normal rules of normalization applied to transactional relational databases are relaxed during star schema design and implementation. The benefits of star schema denormalization are:

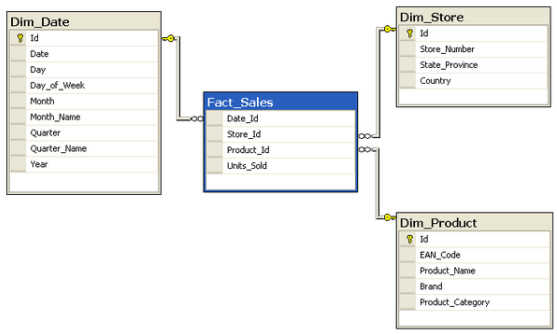
- Simpler queries - star schema join logic is generally simpler than the join logic required to retrieve data from a highly normalized transactional schemas.
- Simplified business reporting logic - when compared to highly normalized schemas, the star schema simplifies common business reporting logic, such as period-over-period and as-of reporting.
- Query performance gains - star schemas can provide performance enhancements for read-only reporting applications when compared to highly normalized schemas.
- Fast aggregations - the simpler queries against a star schema can result in improved performance for aggregation operations.
- Feeding cubes - star schemas are used by all OLAP systems to build proprietary OLAP cubes efficiently; in fact, most major OLAP systems provide a ROLAP mode of operation which can use a star schema directly as a source without building a proprietary cube structure.

1.3.3 Disadvantages

The main disadvantage of the star schema is that data integrity is not enforced as well as it is in a highly normalized database. One-off inserts and updates can result in data anomalies which normalized schemas are designed to avoid. Generally speaking, star schemas are loaded in a highly controlled fashion via batch processing or near-real time "trickle feeds", to compensate for the lack of protection afforded by normalization.

Star schema is also not as flexible in terms of analytical needs as a normalized data model. Normalized models allow any kind of analytical queries to be executed as long as they follow the business logic defined in the model. Star schemas tend to be more purpose-built for a particular view of the data, thus not really allowing more complex analytics. Star schemas don't support many-to-many relationships between business entities - at least not very naturally. Typically these relationships are simplified in star schema to conform to the simple dimensional model.

1.3.4 Example



Star schema used by example query.

Consider a database of sales, perhaps from a store chain, classified by date, store and product. The image of the schema to the right is a star schema version of the sample schema provided in the snowflake schema article.

Fact_Sales is the fact table and there are three dimension tables Dim_Date, Dim_Store and Dim_Product.

Each dimension table has a primary key on its Id column, relating to one of the columns (viewed as rows in the example schema) of the Fact_Sales table's three-column (compound) primary key (Date_Id, Store_Id, Product_Id). The non-primary key Units_Sold column of the fact table in this example represents a measure or metric that can be used in calculations and analysis. The non-primary key columns of the dimension tables represent additional attributes of the dimensions (such as the Year of the Dim_Date dimension).

For example, the following query answers how many TV sets have been sold, for each brand and country, in 1997:

```
SELECT P.Brand, S.Country AS Countries, SUM(F.Units_Sold) FROM Fact_Sales F INNER JOIN Dim_Date D ON (F.Date_Id = D.Id) INNER JOIN Dim_Store S ON (F.Store_Id = S.Id) INNER JOIN Dim_Product P ON (F.Product_Id = P.Id) WHERE D.Year = 1997 AND P.Product_Category = 'tv' GROUP BY P.Brand, S.Country
```

1.3.5 See also

- Online analytical processing
- Reverse star schema
- Snowflake schema
- Fact constellation

1.3.6 References

[1] "DWH Schemas". 2009.

[2] C J Date, "An Introduction to Database Systems (Eighth Edition)", p. 708

[3] Ralph Kimball and Margy Ross, *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling (Second Edition)*, p. 393

1.3.7 External links

- Designing the Star Schema Database by Craig Utley
- Stars: A Pattern Language for Query Optimized Schema
- Fact constellation schema
- Data Warehouses, Schemas and Decision Support Basics by Dan Power

Chapter 2

Not Only SQL

2.1 CAP

CAP may refer to:

2.1.1 Science and medicine

- CaP, prostate cancer
- CAP (protein), cyclase-associated protein
- Carrierless amplitude phase modulation
- Catabolite activator protein, a regulatory protein for mRNA transcription in prokaryotes that binds cyclic AMP
- Cellulose acetate phthalate, a cellulose-based polymer
- Community-acquired pneumonia

2.1.2 Computing

- CAP computer, an experimental machine built in Cambridge, UK
- CAP theorem, Consistency, Availability, Partition-tolerance theorem in computer science
- Camel Application Part, a protocol used in CAMEL servers
- Common Alerting Protocol, an XML based data format for exchanging public warnings between different alerting technologies

2.1.3 Organisations

- Canadian Action Party
- Canadian Association of Physicists
- Center for Adoption Policy
- Center for American Progress, a left-of-centre think tank
- Central Atlanta Progress
- Chicago Area Project, a juvenile delinquency project
- Christian Appalachian Project, a program to assist disadvantaged persons in Kentucky and West Virginia
- Christians Against Poverty, the UK charity
- Church Action on Poverty, UK national ecumenical social justice charity established in 1982
- College of American Pathologists
- Committee of Advertising Practice
- Committee for Another Policy (*Comité voor een Andere Politiek* / *Comité pour une Autre Politique)*, a Belgian political movement
- Concerned Alumni of Princeton
- Congress of Aboriginal Peoples, Canadian aboriginal organization

2.1.4 Companies

- Companhia Aeronáutica Paulista, a 1940s Brazilian aircraft manufacturer
- CAP Group (Computer Analysts and Programmers), a UK software company
- CAP S.A. (*Compañía de Acero del Pacífico*), a Chilean mining and steel sector holding company
- CAP Scientific, a British defence software company (1979-1988)
- Constructions Aéronautiques Parisiennes, Apex Aircraft training and aerobatic aircraft

2.1.5 Projects, programs, policies

- Common Agricultural Policy, the European Union's agricultural subsidy system
- Community Access Program, a government of Canada initiative to provide access to the Internet in remote areas
- Capital Assistance Program
- Community Action Program, Lyndon Johnson's anti-poverty programs
- Community Action Programme, United Kingdom workfare scheme

2.1.6 Military

- Combat air patrol
- Combined Action Program (AKA Combined Action Platoon), a United States Marine Corps Vietnam era special operation
- Civil Air Patrol, the official US Air Force Auxiliary

2.1.7 Certifications

- Certified Automation Professional, certification from the International Society of Automation
- Certified Administrative Professional, certification from the International Association of Administrative Professionals

2.1.8 Other

- Carlos Andrés Pérez (1922-2010), twice President of Venezuela
- CAP Markets, social franchise and supermarket chain in Germany
- Capital Airlines, the ICAO airline designator for this airline
- Causal adequacy principle, a philosophical claim made by René Descartes
- Central Arizona Project, the Colorado River diversion canal in Arizona
- Chip Authentication Program, using EMV smartcards to authenticate online banking transactions
- Coded Anti-Piracy, an anti-piracy system for motion picture prints exhibited theatrically
- Consolidated Appeals Process, a funding mechanism used by humanitarian aid organisations
- Codice di Avviamento Postale, literally *Postal Expedition Code*, Italy's postal code system
- Estadio CAP (*Compañía de Acero del Pacífico*), a football stadium in Talcahuano, Chile

2.1.9 See also

- CAP code (disambiguation)
- Cap (disambiguation)

2.2 Eventual consistency

Eventual consistency is a consistency model used in distributed computing to achieve high availability that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value.[1] Eventual consistency is widely deployed in distributed systems, often under the moniker of optimistic replication,[2] and has origins in early mobile computing projects.[3] A system that has achieved eventual consistency is often said to have **converged**, or achieved **replica convergence**.[4] Eventual consistency is a weak guarantee - most stronger models, like linearizability are trivially eventually consistent, but a system that is merely eventually consistent doesn't usually fulfill these stronger constraints.

Eventually consistent services are often classified as providing BASE (**B**asically **A**vailable, **S**oft state, **E**ventual consistency) semantics, in contrast to traditional ACID (**A**tomicity, **C**onsistency, **I**solation, **D**urability) guarantees.[5][6] Eventual consistency is sometimes criticized[7] as increasing the complexity of distributed software applications. This is partly because eventual consistency is purely a liveness guarantee (reads eventually return the same value) and does not make safety guarantees: an eventually consistent system can return any value before it converges.

2.2.1 Conflict resolution

In order to ensure replica convergence, a system must reconcile differences between multiple copies of distributed data. This consists of two parts:

- exchanging versions or updates of data between servers (often known as **anti-entropy**);[8] and
- choosing an appropriate final state when concurrent updates have occurred, called **reconciliation**.

The most appropriate approach to reconciliation depends on the application. A widespread approach is *"last writer wins"*.[1] Another is to invoke a user-specified conflict

handler.[4] Timestamps and vector clocks are often used to detect concurrency between updates.

Reconciliation of concurrent writes must occur sometime before the next read, and can be scheduled at different instants:[3][9]

- Read repair: The correction is done when a read finds an inconsistency. This slows down the read operation.
- Write repair: The correction takes place during a write operation, if an inconsistency has been found, slowing down the write operation.
- Asynchronous repair: The correction is not part of a read or write operation.

2.2.2 Strong eventual consistency

Whereas EC is only a liveness guarantee (updates will be observed eventually), Strong Eventual Consistency (SEC) adds the safety guarantee that any two nodes that have received the same (unordered) set of updates will be in the same state. If, furthermore, the system is monotonic, the application will never suffer rollbacks. Conflict-free replicated data types are a common approach to ensuring SEC.[10]

2.2.3 See also

- CAP theorem

2.2.4 References

[1] Vogels, W. (2009). "Eventually consistent". *Communications of the ACM* **52**: 40. doi:10.1145/1435417.1435432.

[2] Vogels, W. (2008). "Eventually Consistent". *Queue* **6** (6): 14. doi:10.1145/1466443.1466448.

[3] Terry, D. B.; Theimer, M. M.; Petersen, K.; Demers, A. J.; Spreitzer, M. J.; Hauser, C. H. (1995). "Managing update conflicts in Bayou, a weakly connected replicated storage system". *Proceedings of the fifteenth ACM symposium on Operating systems principles - SOSP '95*. p. 172. doi:10.1145/224056.224070. ISBN 0897917154.

[4] Petersen, K.; Spreitzer, M. J.; Terry, D. B.; Theimer, M. M.; Demers, A. J. (1997). "Flexible update propagation for weakly consistent replication". *ACM SIGOPS Operating Systems Review* **31** (5): 288. doi:10.1145/269005.266711.

[5] Pritchett, D. (2008). "Base: An Acid Alternative". *Queue* **6** (3): 48. doi:10.1145/1394127.1394128.

[6] Bailis, P.; Ghodsi, A. (2013). "Eventual Consistency Today: Limitations, Extensions, and Beyond". *Queue* **11** (3): 20. doi:10.1145/2460276.2462076.

[7] Yaniv Pessach (2013), *Distributed Storage* (Distributed Storage: Concepts, Algorithms, and Implementations ed.), Amazon, Systems using Eventual Consistency result in decreased system load and increased system availability but result in increased cognitive complexity for users and developers

[8] Demers, A.; Greene, D.; Hauser, C.; Irish, W.; Larson, J. (1987). "Epidemic algorithms for replicated database maintenance". *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing - PODC '87*. p. 1. doi:10.1145/41840.41841. ISBN 978-0-89791-239-6.

[9] Olivier Mallassi (2010-06-09). "Let's play with Cassandra… (Part 1/3)". http://blog.octo.com/en/: OCTO Talks!. Retrieved 2011-03-23. Of course, at a given time, chances are high that each node has its own version of the data. Conflict resolution is made during the read requests (called read-repair) and the current version of Cassandra does not provide a Vector Clock conflict resolution mechanisms [sic] (should be available in the version 0.7). Conflict resolution is so based on timestamp (the one set when you insert the row or the column): the higher timestamp win[s] and the node you are reading the data [from] is responsible for that. This is an important point because the timestamp is specified by the client, at the moment the column is inserted. Thus, all Cassandra clients' [sic] need to be synchronized...

[10] Shapiro, Marc; Preguiça, Nuno; Baquero, Carlos; Zawirski, Marek (2011-10-10). "Conflict-free replicated data types". *SSS'11 Proceedings of the 13th international conference on Stabilization, safety, and the security of distributed systems* (Springer-Verlag Berlin, Heidelberg): 386–400.

2.3 Object-relational impedance mismatch

The **object-relational impedance mismatch** is a set of conceptual and technical difficulties that are often encountered when a relational database management system (RDBMS) is being used by a program written in an object-oriented programming language or style, particularly when objects or class definitions are mapped in a straightforward way to database tables or relational schemata.

The term *object-relational impedance mismatch* is derived from the electrical engineering term *impedance matching*.

2.3.1 Mismatches

Object-oriented concepts

Encapsulation Object-oriented programs are designed with techniques that result in encapsulated objects whose representation is hidden. In an object-oriented framework, the underlying properties of a given object are ex-

pected to be unexposed to any interface outside of the one implemented alongside the object. However, object-relational mapping necessarily exposes the underlying content of an object to interaction with an interface that the object implementation cannot specify. Hence, object-relational mapping violates the encapsulation of the object.

Accessibility In relational thinking, "private" versus "public" access is relative to need rather than being an absolute characteristic of the data's state, as in the object-oriented (OO) model. The relational and OO models often have conflicts over relativity versus absolutism of classifications and characteristics.

Interface, class, inheritance and polymorphism Under an object-oriented paradigm, objects have interfaces that together provide the only access to the internals of that object. The relational model, on the other hand, utilizes derived relation variables (views) to provide varying perspectives and constraints to ensure integrity. Similarly, essential OOP concepts for classes of objects, inheritance and polymorphism, are not supported by relational database systems.

Mapping to relational concepts A proper mapping between relational concepts and object-oriented concepts can be made if relational database tables are linked to associations found in object-oriented analysis.

Data type differences

A major mismatch between existing relational and OO languages is the type system differences. The relational model strictly prohibits by-reference attributes (or pointers), whereas OO languages embrace and expect by-reference behavior. Scalar types and their operator semantics can be vastly different between the models, causing problems in mapping.

For example, most SQL systems support string types with varying collations and constrained maximum lengths (open-ended text types tend to hinder performance), while most OO languages consider collation only as an argument to sort routines and strings are intrinsically sized to available memory. A more subtle, but related example is that SQL systems often ignore trailing white space in a string for the purposes of comparison, whereas OO string libraries do not. It is typically not possible to construct new data types as a matter of constraining the possible values of other primitive types in an OO language.

Structural and integrity differences

Another mismatch has to do with the differences in the structural and integrity aspects of the contrasted models. In OO languages, objects can be composed of other objects—often to a high degree—or specialize from a more general definition. This may make the mapping to relational schemas less straightforward. This is because relational data tends to be represented in a named set of global, unnested relation variables. Relations themselves, being sets of tuples all conforming to the same header do not have an ideal counterpart in OO languages. Constraints in OO languages are generally not declared as such, but are manifested as exception raising protection logic surrounding code that operates on encapsulated internal data. The relational model, on the other hand, calls for declarative constraints on scalar types, attributes, relation variables, and the database as a whole.

Manipulative differences

The semantic differences are especially apparent in the manipulative aspects of the contrasted models, however. The relational model has an intrinsic, relatively small and well-defined set of primitive operators for usage in the query and manipulation of data, whereas OO languages generally handle query and manipulation through custom-built or lower-level, case- and physical-access-path-specific imperative operations. Some OO languages do have support for declarative query sublanguages, but because OO languages typically deal with lists and perhaps hash tables, the manipulative primitives are necessarily distinct from the set-based operations of the relational model.

Transactional differences

The concurrency and transaction aspects are significantly different also. In particular, transactions, the smallest unit of work performed by databases, are much larger in relational databases than are any operations performed by classes in OO languages. Transactions in relational databases are dynamically bounded sets of arbitrary data manipulations, whereas the granularity of transactions in an OO language is typically on the level of individual assignments to primitive-typed fields. In general, OO languages have no analogue of isolation or durability, so atomicity and consistency are only ensured when writing to fields of those primitive types.

2.3.2 Solving impedance mismatch

Solving the impedance mismatch problem for object-oriented programs starts with recognition of the differences in the specific logic systems being employed, then either the minimization or compensation of the mismatch.

Minimization

There have been some attempts at building object-oriented database management systems (OODBMS) that would avoid the impedance mismatch problem. They have been less successful in practice than relational databases however, partly due to the limitations of OO principles as a basis for a data model.[1] There has been research performed in extending the database-like capabilities of OO languages through such notions as transactional memory.

One common solution to the impedance mismatch problem is to layer the domain and framework logic. In this scheme, the OO language is used to model certain relational aspects at runtime rather than attempt the more static mapping. Frameworks which employ this method will typically have an analogue for a tuple, usually as a "row" in a "dataset" component or as a generic "entity instance" class, as well as an analogue for a relation. Advantages of this approach may include:

- Straightforward paths to build frameworks and automation around transport, presentation, and validation of domain data.
- Smaller code size; faster compile and load times.
- Ability for the schema to change dynamically.
- Avoids the name-space and semantic mismatch issues.
- Expressive constraint checking
- No complex mapping necessary

Disadvantages may include:

- Lack of static type "safety" checks. Typed accessors are sometimes utilized as one way to mitigate this.
- Possible performance cost of runtime construction and access.
- Inability to natively utilize uniquely OO aspects, such as polymorphism.

Alternative architectures

The rise of XML databases and XML client structures has motivated other alternative architectures to get around the impedance mismatch challenges. These architectures use XML technology in the client (such as XForms) and native XML databases on the server that use the XQuery language for data selection. This allows a single data model and a single data selection language (XPath) to be used in the client, in the rules engines and on the persistence server.[2]

Compensation

The mixing of levels of discourse within OO application code presents problems, but there are some common mechanisms used to compensate. The biggest challenge is to provide framework support, automation of data manipulation and presentation patterns, within the level of discourse in which the domain data is being modeled. To address this, reflection and/or code generation are utilized. Reflection allows code (classes) to be addressed as data and thus provide automation of the transport, presentation, integrity, etc. of the data. Generation addresses the problem through addressing the entity structures as data inputs for code generation tools or meta-programming languages, which produce the classes and supporting infrastructure en masse. Both of these schemes may still be subject to certain anomalies where these levels of discourse merge. For instance, generated entity classes will typically have properties which map to the domain (e. g. Name, Address) as well as properties which provide state management and other framework infrastructure (e. g. IsModified).

2.3.3 Contention

It has been argued, by Christopher J. Date and others, that a truly relational DBMS would pose no such problem,[3][4][5] as domains and classes are essentially one and the same thing. A naïve mapping between classes and relational schemata is a fundamental design mistake ; and that individual tuples within a database table (relation) ought to be viewed as establishing relationships between entities; not as representations for complex entities themselves. However, this view tends to diminish the influence and role of object-oriented programming, using it as little more than a field type management system.

The impedance mismatch is in programming between the domain objects and the user interface. Sophisticated user interfaces, to allow operators, managers, and other non-programmers to access and manipulate the records in the database, often require intimate knowledge about the nature of the various database attributes (beyond name and type). In particular, it's considered a good practice (from an end-user productivity point of view) to design user interfaces such that the UI prevents illegal transactions (those which cause a database constraint to be violated) from being entered; to do so requires much of the logic present in the relational schemata to be duplicated in the code.

Certain code-development frameworks can leverage certain forms of logic that are represented in the database's schema (such as referential integrity constraints), so that such issues are handled in a generic and standard fashion through library routines rather than ad hoc code written on a case-by-case basis.

It has been argued that SQL, due to a very limited set of

domain types (and other alleged flaws) makes proper object and domain-modelling difficult; and that SQL constitutes a very lossy and inefficient interface between a DBMS and an application program (whether written in an object-oriented style or not). However, SQL is currently the only widely accepted common database language in the marketplace; use of vendor-specific query languages is seen as a bad practice when avoidable. Other database languages such as Business System 12 and Tutorial D have been proposed; but none of these has been widely adopted by DBMS vendors.

In current versions of mainstream "object-relational" DBMSs like Oracle and Microsoft SQL Server, the above point may be a non-issue. With these engines, the functionality of a given database can be arbitrarily extended through stored code (functions and procedures) written in a modern OO language (Java for Oracle, and a Microsoft .NET language for SQL Server), and these functions can be invoked in-turn in SQL statements in a transparent fashion: that is, the user neither knows nor cares that these functions/procedures were not originally part of the database engine. Modern software-development paradigms are fully supported: thus, one can create a set of library routines that can be re-used across multiple database schemas.

These vendors decided to support OO-language integration at the DBMS back-end because they realized that, despite the attempts of the ISO SQL-99 committee to add procedural constructs to SQL, SQL will never have the rich set of libraries and data structures that today's application programmers take for granted, and it is reasonable to leverage these as directly as possible rather than attempting to extend the core SQL language. Consequently, the difference between "application programming" and "database administration" is now blurred: robust implementation of features such as constraints and triggers may often require an individual with dual DBA/OO-programming skills, or a partnership between individuals who combine these skills. This fact also bears on the "division of responsibility" issue below.

Some, however, would point out that this contention is moot due to the fact that: (1) RDBMSes were never intended to facilitate object modelling, and (2) SQL generally should only be seen as a "lossy" or "inefficient" interface language when one is trying to achieve a solution for which RDBMSes were not designed. SQL is very efficient at doing what it was designed to do, namely, to query, sort, filter, and store large sets of data. Some would additionally point out that the inclusion of OO language functionality in the back-end simply facilitates bad architectural practice, as it admits high-level application logic into the data tier, antithetical to the RDBMS.

Here the "canonical" copy of state is located. The database model generally assumes that the database management system is the only authoritative repository of state concerning the enterprise; any copies of such state held by an application program are just that — temporary copies (which may be out of date, if the underlying database record was subsequently modified by a transaction). Many object-oriented programmers prefer to view the in-memory representations of objects themselves as the canonical data, and view the database as a backing store and persistence mechanism.

Another point of contention is the proper division of responsibility between application programmers and database administrators (DBA). It is often the case that needed changes to application code (in order to implement a requested new feature or functionality) require corresponding changes in the database definition; in most organizations, the database definition is the responsibility of the DBA. Due to the need to maintain a production database system 24 hours a day many DBAs are reluctant to make changes to database schemata that they deem gratuitous or superfluous and in some cases outright refuse to do so. Use of developmental databases (apart from production systems) can help somewhat; but when the newly developed application "goes live" the DBA will need to approve any changes. Some programmers view this as intransigence; however the DBA is frequently held responsible if any changes to the database definition cause a loss of service in a production system—as a result, many DBAs prefer to contain design changes to application code, where design defects are far less likely to have catastrophic consequences.

In organizations with a non-dysfunctional relationship between DBAs and developers, though, the above issue should not present itself, as the decision to change a database schema or not would only be driven by business needs: a new requirement to persist additional data or a performance boost of a critical application would both trigger a schema modification, for example.

2.3.4 Philosophical differences

Key philosophical differences between the OO and relational models can be summarized as follows:

- **Declarative vs. imperative interfaces** — Relational thinking tends to use data as interfaces, not behavior as interfaces. It thus has a declarative tilt in design philosophy in contrast to OO's behavioral tilt. (Some relational proponents propose using triggers, stored procedures, etc. to provide complex behavior, but this is not a common viewpoint.)

- **Schema bound** — Objects do not have to follow a "parent schema" for which attributes or accessors an object has, while table rows must follow the entity's schema. A given row must belong to one and only one entity. The closest thing in OO is inheritance, but it is generally tree-shaped and optional. A dynamic form of relational tools that allows ad

hoc columns may relax schema bound-ness, but such tools are currently rare.

- **Access rules** — In relational databases, attributes are accessed and altered through predefined relational operators, while OO allows each class to create its own state alteration interface and practices. The "self-handling noun" viewpoint of OO gives independence to each object that the relational model does not permit. This is a "standards versus local freedom" debate. OO tends to argue that relational standards limit expressiveness, while relational proponents suggest the rule adherence allows more abstract math-like reasoning, integrity, and design consistency.

- **Relationship between nouns and verbs** — OO encourages a tight association between verbs (actions) and the nouns (entities) that the operations operate on. The resulting tightly bound entity containing both nouns and the verbs is usually called a class, or in OO analysis, a concept. Relational designs generally do not assume there is anything natural or logical about such tight associations (outside of relational operators).

- **Object identity** — Objects (other than immutable ones) are generally considered to have a unique identity; two objects which happen to have the same state at a given point in time are not considered to be identical. Relations, on the other hand, have no inherent concept of this kind of identity. That said, it is a common practice to fabricate "identity" for records in a database through use of globally unique candidate keys; though many consider this a poor practice for any database record which does not have a one-to-one correspondence with a real world entity. (Relational, like objects, can use domain keys if they exist in the external world for identification purposes). Relational systems in practice strive for and support "permanent" and inspectable identification techniques, whereas object identification techniques tend to be transient or situational.

- **Normalization** — Relational normalization practices are often ignored by OO designs. However, this may just be a bad habit instead of a native feature of OO. An alternate view is that a collection of objects, interlinked via pointers of some sort, is equivalent to a network database; which in turn can be viewed as an extremely denormalized relational database.

- **Schema inheritance** — Most relational databases do not support schema inheritance. Although such a feature could be added in theory to reduce the conflict with OOP, relational proponents are less likely to believe in the utility of hierarchical taxonomies and sub-typing because they tend to view set-based taxonomies or classification systems as more powerful and flexible than trees. OO advocates point out that inheritance/subtyping models need not be limited to trees (though this is a limitation in many popular OO languages such as Java), but non-tree OO solutions are seen as more difficult to formulate than set-based variation-on-a-theme management techniques preferred by relational. At the least, they differ from techniques commonly used in relational algebra.

- **Structure vs. behaviour** — OO primarily focuses on ensuring that the structure of the program is reasonable (maintainable, understandable, extensible, reusable, safe), whereas relational systems focus on what kind of behaviour the resulting run-time system has (efficiency, adaptability, fault-tolerance, liveness, logical integrity, etc.). Object-oriented methods generally assume that the primary user of the object-oriented code and its interfaces are the application developers. In relational systems, the end-users' view of the behaviour of the system is sometimes considered to be more important. However, relational queries and "views" are common techniques to present information in application- or task-specific configurations. Further, relational does not prohibit local or application-specific structures or tables from being created, although many common development tools do not directly provide such a feature, assuming objects will be used instead. This makes it difficult to know whether the stated non-developer perspective of relational is inherent to relational, or merely a product of current practice and tool implementation assumptions.

- **Set vs. graph relationships** — The relationship between different items (objects or records) tend to be handled differently between the paradigms. Relational relationships are usually based on idioms taken from set theory, while object relationships lean toward idioms adopted from graph theory (including trees). While each can represent the same information as the other, the approaches they provide to access and manage information differ.

As a result of the object-relational impedance mismatch, it is often argued by partisans on both sides of the debate that the other technology ought to be abandoned or reduced in scope.[6] Some database advocates view traditional "procedural" languages as more compatible with an RDBMS than many OO languages; or suggest that a less OO-style ought to be used. (In particular, it is argued that long-lived domain objects in application code ought not to exist; any such objects that do exist should be created when a query is made and disposed of when a transaction or task is complete). On the other hand, many OO advocates argue that more OO-friendly persistence mechanisms, such as OODBMS, ought to be developed and used, and that relational technology ought

to be phased out. Of course, it should be pointed out that many (if not most) programmers and DBAs do not hold either of these viewpoints; and view the object-relational impedance mismatch as a mere fact of life that information technology has to deal with.

It is also argued that the O/R mapping is paying off in some situations, but is probably oversold: it has advantages besides drawbacks. Skeptics point out that it is worth to think carefully before using it, as it will add little value in some cases.[7]

2.3.5 References

[1] C. J. Date, Relational Database Writings

[2] Dan McCreary, XRX: Simple, Elegant, Disruptive on XML.com

[3] Date, Christopher 'Chris' J; Pascal, Fabian (2012-08-12) [2005], "Type vs. Domain and Class", *Database debunkings* (World Wide Web log), Google, retrieved 12 September 2012.

[4] ——— (2006), "4. On the notion of logical difference", *Date on Database: writings 2000–2006*, The expert's voice in database; Relational database select writings, USA: Apress, p. 39, ISBN 978-1-59059-746-0, Class seems to be indistinguishable from type, as that term is classically understood.

[5] ——— (2004), "26. Object/Relational databases", *An introduction to database systems* (8th ed.), Pearson Addison Wesley, p. 859, ISBN 978-0-321-19784-9, ...any such *rapprochement* should be firmly based on the relational model.

[6] Neward, Ted (2006-06-26). "The Vietnam of Computer Science". Interoperability Happens. Retrieved 2010-06-02.

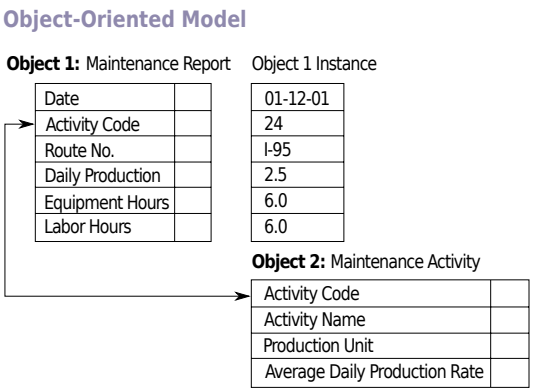
[7] *J2EE Design and Development* by Rod Johnson, © 2002 Wrox Press, p. 256.

2.3.6 External links

- The Object-Relational Impedance Mismatch - Agile Data Essay
- The Vietnam of Computer Science - Examples of mismatch problems

2.4 Object database

An **object database** (also **object-oriented database management system**) is a database management system in which information is represented in the form of objects as used in object-oriented programming. Object databases are different from relational databases which are table-oriented. Object-relational databases are a hybrid of both approaches.



Example of an object-oriented model[1]

Object databases have been considered since the early 1980s.[2]

2.4.1 Overview

Object-oriented database management systems (OODBMSs) combine database capabilities with object-oriented programming language capabilities. OODBMSs allow object-oriented programmers to develop the product, store them as objects, and replicate or modify existing objects to make new objects within the OODBMS. Because the database is integrated with the programming language, the programmer can maintain consistency within one environment, in that both the OODBMS and the programming language will use the same model of representation. Relational DBMS projects, by way of contrast, maintain a clearer division between the database model and the application.

As the usage of web-based technology increases with the implementation of Intranets and extranets, companies have a vested interest in OODBMSs to display their complex data. Using a DBMS that has been specifically designed to store data as objects gives an advantage to those companies that are geared towards multimedia presentation or organizations that utilize computer-aided design (CAD).[3]

Some object-oriented databases are designed to work well with object-oriented programming languages such as Delphi, Ruby, Python, Perl, Java, C#, Visual Basic .NET, C++, Objective-C and Smalltalk; others have their own programming languages. OODBMSs use exactly the same model as object-oriented programming languages.

2.4.2 History

Object database management systems grew out of research during the early to mid-1970s into having intrinsic database management support for graph-structured objects. The term "object-oriented database system"

first appeared around 1985.[4] Notable research projects included Encore-Ob/Server (Brown University), EXODUS (University of Wisconsin–Madison), IRIS (Hewlett-Packard), ODE (Bell Labs), ORION (Microelectronics and Computer Technology Corporation or MCC), Vodak (GMD-IPSI), and Zeitgeist (Texas Instruments). The ORION project had more published papers than any of the other efforts. Won Kim of MCC compiled the best of those papers in a book published by The MIT Press.[5]

Early commercial products included Gemstone (Servio Logic, name changed to GemStone Systems), Gbase (Graphael), and Vbase (Ontologic). The early to mid-1990s saw additional commercial products enter the market. These included ITASCA (Itasca Systems), Jasmine (Fujitsu, marketed by Computer Associates), Matisse (Matisse Software), Objectivity/DB (Objectivity, Inc.), ObjectStore (Progress Software, acquired from eXcelon which was originally Object Design), ONTOS (Ontos, Inc., name changed from Ontologic), O_2[6] (O_2 Technology, merged with several companies, acquired by Informix, which was in turn acquired by IBM), POET (now FastObjects from Versant which acquired Poet Software), Versant Object Database (Versant Corporation), VOSS (Logic Arts) and JADE (Jade Software Corporation). Some of these products remain on the market and have been joined by new open source and commercial products such as InterSystems Caché.

Object database management systems added the concept of persistence to object programming languages. The early commercial products were integrated with various languages: GemStone (Smalltalk), Gbase (LISP), Vbase (COP) and VOSS (Virtual Object Storage System for Smalltalk). For much of the 1990s, C++ dominated the commercial object database management market. Vendors added Java in the late 1990s and more recently, C#.

Starting in 2004, object databases have seen a second growth period when open source object databases emerged that were widely affordable and easy to use, because they are entirely written in OOP languages like Smalltalk, Java, or C#, such as Versant's db4o (db4objects), DTS/S1 from Obsidian Dynamics and Perst (McObject), available under dual open source and commercial licensing.

2.4.3 Timeline

- 1966
  - MUMPS
- 1979
  - InterSystems M
- 1980
  - TORNADO – an object database for CAD/CAM[7]
- 1982
  - Gemstone started (as Servio Logic) to build a set theoretic model data base machine.
- 1985 – Term Object Database first introduced
- 1986
  - Servio Logic (Gemstone Systems) Ships Gemstone 1.0
- 1988
  - Versant Corporation started (as Object Sciences Corp)
  - Objectivity, Inc. founded
- Early 1990s
  - Servio Logic changes name to Gemstone Systems
  - Gemstone (Smalltalk)-(C++)-(Java)
  - GBase (LISP)
  - VBase (O2- ONTOS – INFORMIX)
  - Objectivity/DB
- Mid 1990's
  - InterSystems Caché
  - Versant Object Database
  - ObjectStore
  - ODABA
  - ZODB
  - Poet
  - Jade
  - Matisse
  - Illustra Informix
  - Webcrossing
- 2000's
  - db4o project started by Carl Rosenberger
  - ObjectDB
- 2001 IBM acquires Informix
- 2003 odbpp public release
- 2004 db4o's commercial launch as db4objects, Inc.
- 2008 db4o acquired by Versant Corporation
- 2010 VMware acquires GemStone[8]
- 2012 Wakanda first production versions with open source and commercial licenses
- 2013 GemTalk Systems acquires GemStone products from VMware[9]
- 2014 Realm

2.4.4 Adoption of object databases

Object databases based on persistent programming acquired a niche in application areas such as engineering and spatial databases, telecommunications, and scientific areas such as high energy physics and molecular biology.

Another group of object databases focuses on embedded use in devices, packaged software, and real-time systems.

2.4.5 Technical features

Most object databases also offer some kind of query language, allowing objects to be found using a declarative programming approach. It is in the area of object query languages, and the integration of the query and navigational interfaces, that the biggest differences between products are found. An attempt at standardization was made by the ODMG with the Object Query Language, OQL.

Access to data can be faster because joins are often not needed (as in a tabular implementation of a relational database). This is because an object can be retrieved directly without a search, by following pointers.

Another area of variation between products is in the way that the schema of a database is defined. A general characteristic, however, is that the programming language and the database schema use the same type definitions.

Multimedia applications are facilitated because the class methods associated with the data are responsible for its correct interpretation.

Many object databases, for example Gemstone or VOSS, offer support for versioning. An object can be viewed as the set of all its versions. Also, object versions can be treated as objects in their own right. Some object databases also provide systematic support for triggers and constraints which are the basis of active databases.

The efficiency of such a database is also greatly improved in areas which demand massive amounts of data about one item. For example, a banking institution could get the user's account information and provide them efficiently with extensive information such as transactions, account information entries etc. The Big O Notation for such a database paradigm drops from O(n) to O(1), greatly increasing efficiency in these specific cases.

2.4.6 Standards

The Object Data Management Group was a consortium of object database and object-relational mapping vendors, members of the academic community, and interested parties. Its goal was to create a set of specifications that would allow for portable applications that store objects in database management systems. It published several versions of its specification. The last release was ODMG 3.0. By 2001, most of the major object database and object-relational mapping vendors claimed conformance to the ODMG Java Language Binding. Compliance to the other components of the specification was mixed. In 2001, the ODMG Java Language Binding was submitted to the Java Community Process as a basis for the Java Data Objects specification. The ODMG member companies then decided to concentrate their efforts on the Java Data Objects specification. As a result, the ODMG disbanded in 2001.

Many object database ideas were also absorbed into SQL:1999 and have been implemented in varying degrees in object-relational database products.

In 2005 Cook, Rai, and Rosenberger proposed to drop all standardization efforts to introduce additional object-oriented query APIs but rather use the OO programming language itself, i.e., Java and .NET, to express queries. As a result, Native Queries emerged. Similarly, Microsoft announced Language Integrated Query (LINQ) and DLINQ, an implementation of LINQ, in September 2005, to provide close, language-integrated database query capabilities with its programming languages C# and VB.NET 9.

In February 2006, the Object Management Group (OMG) announced that they had been granted the right to develop new specifications based on the ODMG 3.0 specification and the formation of the Object Database Technology Working Group (ODBT WG). The ODBT WG planned to create a set of standards that would incorporate advances in object database technology (e.g., replication), data management (e.g., spatial indexing), and data formats (e.g., XML) and to include new features into these standards that support domains where object databases are being adopted (e.g., real-time systems). The work of the ODBT WG was suspended in March 2009 when, subsequent to the economic turmoil in late 2008, the ODB vendors involved in this effort decided to focus their resources elsewhere.

In January 2007 the World Wide Web Consortium gave final recommendation status to the XQuery language. XQuery uses XML as its data model. Some of the ideas developed originally for object databases found their way into XQuery, but XQuery is not intrinsically object-oriented. Because of the popularity of XML, XQuery engines compete with object databases as a vehicle for storage of data that is too complex or variable to hold conveniently in a relational database. XQuery also allows modules to be written to provide encapsulation features that have been provided by Object-Oriented systems.

2.4.7 Comparison with RDBMSs

An object database stores complex data and relationships between data directly, without mapping to relational rows and columns, and this makes them suitable for applications dealing with very complex data.[10] Objects have a

many to many relationship and are accessed by the use of pointers. Pointers are linked to objects to establish relationships. Another benefit of an OODBMS is that it can be programmed with small procedural differences without affecting the entire system.[11]

2.4.8 See also

- Comparison of object database management systems
- Component-oriented database
- EDA database
- Enterprise Objects Framework
- NoSQL
- Object Data Management Group
- Object-relational database
- Persistence (computer science)
- Relational model

2.4.9 References

[1] Data Integration Glossary, U.S. Department of Transportation, August 2001.

[2] ODBMS.ORG :: Object Database (ODBMS) | Object-Oriented Database (OODBMS) | Free Resource Portal. ODBMS (2013-08-31). Retrieved on 2013-09-18. Archived July 25, 2014 at the Wayback Machine

[3] O'Brien, J. A., & Marakas, G. M. (2009). Management Information Systems (9th ed.). New York, NY: McGraw-Hill/Irwin

[4] Three example references from 1985 that use the term: T. Atwood, "An Object-Oriented DBMS for Design Support Applications," *Proceedings of the IEEE COMPINT 85*, pp. 299-307, September 1985; N. Derrett, W. Kent, and P. Lyngbaek, "Some Aspects of Operations in an Object-Oriented Database," *Database Engineering*, vol. 8, no. 4, IEEE Computer Society, December 1985; D. Maier, A. Otis, and A. Purdy, "Object-Oriented Database Development at Servio Logic," *Database Engineering*, vol. 18, no.4, December 1985.

[5] Kim, Won. *Introduction to Object-Oriented Databases.* The MIT Press, 1990. ISBN 0-262-11124-1

[6] Bancilhon, Francois; Delobel,Claude; and Kanellakis, Paris. *Building an Object-Oriented Database System: The Story of O_2*. Morgan Kaufmann Publishers, 1992. ISBN 1-55860-169-4.

[7] Ulfsby; et al. (July 1981). "TORNADO: a DBMS for CAD/CAM systems". *Computer-Aided Design* **13** (4): 193–197.

[8] "SpringSource to Acquire Gemstone Systems Data Management Technology". WMware. May 6, 2010. Retrieved August 5, 2014.

[9] GemTalk Systems (May 2, 2013). "GemTalk Systems Acquires GemStone/S Products from VMware". PRWeb. Retrieved August 5, 2014.

[10] Radding, Alan (1995). "So what the Hell is ODBMS?". *Computerworld* **29** (45): 121–122, 129.

[11] Burleson, Donald. (1994). OODBMSs gaining MIS ground but RDBMSs still own the road. Software Magazine, 14(11), 63

2.4.10 External links

- Object DBMS resource portal
- Object-Oriented Databases – From CompTechDoc.org
- DB-Engines Ranking of Object Oriented DBMS by popularity, updated monthly

2.5 NoSQL

"Structured storage" redirects here. For the Microsoft technology also known as structured storage, see COM Structured Storage.

A **NoSQL** (originally referring to "non SQL" or "non relational" [1]) database provides a mechanism for storage and retrieval of data that is modeled in means other than the tabular relations used in relational databases. Such databases have existed since the late 1960s, but did not obtain the "NoSQL" moniker until a surge of popularity in the early twenty-first century,[2] triggered by the needs of Web 2.0 companies such as Facebook, Google and Amazon.com.[3][4][5]

Motivations for this approach include: simplicity of design, simpler "horizontal" scaling to clusters of machines, which is a problem for relational databases,[2] and finer control over availability. The data structures used by NoSQL databases (e.g. key-value, wide column, graph, or document) differ slightly from those used by default in relational databases, making some operations faster in NoSQL and others faster in relational databases. The particular suitability of a given NoSQL database depends on the problem it must solve. Sometimes the data structures used by NoSQL databases are also viewed as "more flexible" than relational database tables.[6]

NoSQL databases are increasingly used in big data and real-time web applications.[7] NoSQL systems are also sometimes called "Not only SQL" to emphasize that they may support SQL-like query languages.[8][9]

Many NoSQL stores compromise consistency (in the sense of the CAP theorem) in favor of availability, partition tolerance, and speed. Barriers to the greater adoption of NoSQL stores include the use of low-level query languages (instead of SQL, for instance the lack of ability to perform ad-hoc JOIN's across tables), lack of standardized interfaces, and huge previous investments in existing relational databases.[10] Most NoSQL stores lack true ACID transactions, although a few databases, such as MarkLogic, Aerospike, FairCom c-treeACE, Google Spanner (though technically a NewSQL database), Symas LMDB and OrientDB have made them central to their designs. (See ACID and JOIN Support.)

Instead, most NoSQL databases offer a concept of "eventual consistency" in which database changes are propagated to all nodes "eventually" (typically within milliseconds) so queries for data might not return updated data immediately or might result in reading data that is not accurate, a problem known as stale reads.[11] Additionally, some NoSQL systems may exhibit lost writes and other forms of data loss.[12] Fortunately, some NoSQL systems provide concepts such as write-ahead logging to avoid data loss.[13] For distributed transaction processing across multiple databases, data consistency is an even bigger challenge that is difficult for both NoSQL and relational databases. Even current relational databases "do not allow referential integrity constraints to span databases."[14] There are few systems that maintain both ACID transactions and X/Open XA standards for distributed transaction processing.

2.5.1 History

The term *NoSQL* was used by Carlo Strozzi in 1998 to name his lightweight, Strozzi NoSQL open-source relational database that did not expose the standard SQL interface, but was still relational.[15] His NoSQL RDBMS is distinct from the circa-2009 general concept of NoSQL databases. Strozzi suggests that, as the current NoSQL movement "departs from the relational model altogether; it should therefore have been called more appropriately 'NoREL'",[16] referring to 'No Relational'.

Johan Oskarsson of Last.fm reintroduced the term *NoSQL* in early 2009 when he organized an event to discuss "open source distributed, non relational databases".[17] The name attempted to label the emergence of an increasing number of non-relational, distributed data stores, including open source clones of Google's BigTable/MapReduce and Amazon's Dynamo. Most of the early NoSQL systems did not attempt to provide atomicity, consistency, isolation and durability guarantees, contrary to the prevailing practice among relational database systems.[18]

Based on 2014 revenue, the NoSQL market leaders are MarkLogic, MongoDB, and Datastax.[19] Based on 2015 popularity rankings, the most popular NoSQL databases are MongoDB, Apache Cassandra, and Redis.[20]

2.5.2 Types and examples of NoSQL databases

There have been various approaches to classify NoSQL databases, each with different categories and subcategories, some of which overlap. A basic classification based on data model, with examples:

- **Column**: Accumulo, Cassandra, Druid, HBase, Vertica
- **Document**: Apache CouchDB, Clusterpoint, Couchbase, DocumentDB, HyperDex, Lotus Notes, MarkLogic, MongoDB, OrientDB, Qizx
- **Key-value**: Aerospike, CouchDB, Dynamo, FairCom c-treeACE, FoundationDB, HyperDex, MemcacheDB, MUMPS, Oracle NoSQL Database, OrientDB, Redis, Riak
- **Graph**: Allegro, InfiniteGraph, MarkLogic, Neo4J, OrientDB, Virtuoso, Stardog
- **Multi-model**: Alchemy Database, ArangoDB, CortexDB, FoundationDB, MarkLogic, OrientDB

A more detailed classification is the following, based on one from Stephen Yen:[21]

Correlation databases are model-independent, and instead of row-based or column-based storage, use value-based storage.

Key-value stores

Main article: Key-value database

Key-value (KV) stores use the associative array (also known as a map or dictionary) as their fundamental data model. In this model, data is represented as a collection of key-value pairs, such that each possible key appears at most once in the collection.[22][23]

The key-value model is one of the simplest non-trivial data models, and richer data models are often implemented on top of it. The key-value model can be extended to an *ordered* model that maintains keys in lexicographic order. This extension is powerful, in that it can efficiently process key *ranges*.[24]

Key-value stores 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.

Examples include Oracle NoSQL Database, redis, and dbm.

Document store

Main articles: Document-oriented database and XML database

The central concept of a document store is the notion of a "document". While each document-oriented database implementation differs on the details of this definition, in general, they all assume that documents encapsulate and encode data (or information) in some standard formats or encodings. Encodings in use include XML, YAML, and JSON as well as binary forms like BSON. Documents are addressed in the database via a unique *key* that represents that document. One of the other defining characteristics of a document-oriented database is that in addition to the key lookup performed by a key-value store, the database offers an API or query language that retrieves documents based on their contents

Different implementations offer different ways of organizing and/or grouping documents:

- Collections
- Tags
- Non-visible metadata
- Directory hierarchies

Compared to relational databases, for example, collections could be considered analogous to tables and documents analogous to records. But they are different: every record in a table has the same sequence of fields, while documents in a collection may have fields that are completely different.

Graph

Main article: Graph database

This kind of database is designed for data whose relations are well represented as a graph (elements interconnected with an undetermined number of relations between them). The kind of data could be social relations, public transport links, road maps or network topologies, for example.

Graph databases and their query language

Object database

Main article: Object database

- db4o
- GemStone/S
- InterSystems Caché
- JADE
- NeoDatis ODB
- ObjectDatabase++
- ObjectDB
- Objectivity/DB
- ObjectStore
- ODABA
- Perst
- OpenLink Virtuoso
- Versant Object Database
- ZODB

Tabular

- Apache Accumulo
- BigTable
- Apache Hbase
- Hypertable
- Mnesia
- OpenLink Virtuoso

Tuple store

- Apache River
- GigaSpaces
- Tarantool
- TIBCO ActiveSpaces
- OpenLink Virtuoso

Triple/quad store (RDF) database

Main articles: Triplestore and Named graph

- AllegroGraph
- Apache JENA (It's a framework, not a database)
- MarkLogic
- Ontotext-OWLIM
- Oracle NoSQL database
- SparkleDB
- Virtuoso Universal Server
- Stardog

Hosted

- Amazon DynamoDB
- Amazon SimpleDB
- Datastore on Google Appengine
- Clusterpoint database
- Cloudant Data Layer (CouchDB)
- Freebase
- Microsoft Azure Tables [25]
- Microsoft Azure DocumentDB [26]
- OpenLink Virtuoso

Multivalue databases

- D3 Pick database
- Extensible Storage Engine (ESE/NT)
- InfinityDB
- InterSystems Caché
- Northgate Information Solutions Reality, the original Pick/MV Database
- OpenQM
- Revelation Software's OpenInsight
- Rocket U2

Multimodel database

- OrientDB
- FoundationDB
- ArangoDB
- MarkLogic

2.5.3 Performance

Ben Scofield rated different categories of NoSQL databases as follows: [27]

Performance and scalability comparisons are sometimes done with the YCSB benchmark.

See also: Comparison of structured storage software

2.5.4 Handling relational data

Since most NoSQL databases lack ability for joins in queries, the database schema generally needs to be designed differently. There are three main techniques for handling relational data in a NoSQL database. (See table Join and ACID Support for NoSQL databases that support joins.)

Multiple queries

Instead of retrieving all the data with one query, it's common to do several queries to get the desired data. NoSQL queries are often faster than traditional SQL queries so the cost of having to do additional queries may be acceptable. If an excessive number of queries would be necessary, one of the other two approaches is more appropriate.

Caching/replication/non-normalized data

Instead of only storing foreign keys, it's common to store actual foreign values along with the model's data. For example, each blog comment might include the username in addition to a user id, thus providing easy access to the username without requiring another lookup. When a username changes however, this will now need to be changed in many places in the database. Thus this approach works better when reads are much more common than writes.[28]

Nesting data

With document databases like MongoDB it's common to put more data in a smaller number of collections. For example, in a blogging application, one might choose to store comments within the blog post document so that with a single retrieval one gets all the comments. Thus in this approach a single document contains all the data you need for a specific task.

2.5.5 ACID and JOIN Support

If a database is marked as supporting ACID or joins, then the documentation for the database makes that claim. The degree to which the capability is fully supported in a manner similar to most SQL databases or the degree to which it meets the needs of a specific application is left up to the reader to assess.

(*) HyperDex currently offers ACID support via its Warp extension, which is a commercial add-on. (**) Joins do not necessarily apply to document databases, but MarkLogic can do joins using semantics [29]

2.5.6 See also

- CAP theorem
- Comparison of object database management systems
- Comparison of structured storage software
- Correlation database
- Distributed cache
- Faceted search
- MultiValue database
- Multi-model database
- Triplestore

2.5.7 References

[1] http://nosql-database.org/ "NoSQL DEFINITION: Next Generation Databases mostly addressing some of the points: being non-relational, distributed, open-source and horizontally scalable"

[2] Leavitt, Neal (2010). "Will NoSQL Databases Live Up to Their Promise?" (PDF). *IEEE Computer*.

[3] Mohan, C. (2013). *History Repeats Itself: Sensible and NonsenSQL Aspects of the NoSQL Hoopla* (PDF). Proc. 16th Int'l Conf. on Extending Database Technology.

[4] http://www.eventbrite.com/e/nosql-meetup-tickets-341739151 "Dynamo clones and BigTables"

[5] http://www.wired.com/2012/01/amazon-dynamodb/ "Amazon helped start the "NoSQL" movement."

[6] http://www.allthingsdistributed.com/2012/01/amazon-dynamodb.html "Customers like SimpleDB's table interface and its flexible data model. Not having to update their schemas when their systems evolve makes life much easier"

[7] "RDBMS dominate the database market, but NoSQL systems are catching up". DB-Engines.com. 21 Nov 2013. Retrieved 24 Nov 2013.

[8] "NoSQL (Not Only SQL)". NoSQL database, also called Not Only SQL

[9] Fowler, Martin. "NosqlDefinition". many advocates of NoSQL say that it does not mean a "no" to SQL, rather it means Not Only SQL

[10] Grolinger, K.; Higashino, W. A.; Tiwari, A.; Capretz, M. A. M. (2013). "Data management in cloud environments: NoSQL and NewSQL data stores" (PDF). JoCCASA, Springer. Retrieved 8 Jan 2014.

[11] https://aphyr.com/posts/322-call-me-maybe-mongodb-stale-reads

[12] Martin Zapletal: Large volume data analysis on the Typesafe Reactive Platform, ScalaDays 2015, Slides

[13] http://www.dummies.com/how-to/content/10-nosql-misconceptions.html "NoSQL databases lose data" section

[14] https://iggyfernandez.wordpress.com/2013/07/28/no-to-sql-and-no-to-nosql/

[15] Lith, Adam; Mattson, Jakob (2010). "Investigating storage solutions for large data: A comparison of well performing and scalable data storage solutions for real time extraction and batch insertion of data" (PDF). Göteborg: Department of Computer Science and Engineering, Chalmers University of Technology. p. 70. Retrieved 12 May 2011. Carlo Strozzi first used the term NoSQL in 1998 as a name for his open source relational database that did not offer a SQL interface[...]

[16] "NoSQL Relational Database Management System: Home Page". Strozzi.it. 2 October 2007. Retrieved 29 March 2010.

[17] "NoSQL 2009". Blog.sym-link.com. 12 May 2009. Retrieved 29 March 2010.

[18] Chapple, Mike. "The ACID Model".

[19] "Hadoop-NoSQL-rankings". Retrieved 2015-11-17.

[20] "DB-Engines Ranking". Retrieved 2015-07-31.

[21] Yen, Stephen. "NoSQL is a Horseless Carriage" (PDF). NorthScale. Retrieved 2014-06-26..

[22] Sandy (14 January 2011). "Key Value stores and the NoSQL movement". http://dba.stackexchange.com/questions/607/what-is-a-key-value-store-database: Stackexchange. Retrieved 1 January 2012. Key-value stores allow the application developer to store schema-less data. This data usually consists of a string that represents the key, and the actual data that is considered the value in the "key-value" relationship. The data itself is usually some kind of primitive of the programming language (a string, an integer, or an array) or an object that is being marshaled by the programming language's bindings to the key-value store. This structure replaces the need for a fixed data model and allows proper formatting.

[23] Seeger, Marc (21 September 2009). "Key-Value Stores: a practical overview" (PDF). http://blog.marc-seeger.de/2009/09/21/key-value-stores-a-practical-overview/: Marc Seeger. Retrieved 1 January 2012. Key-value stores provide a high-performance alternative to relational database systems with respect to storing and accessing data. This paper provides a short overview of some of the currently available key-value stores and their interface to the Ruby programming language.

[24] Katsov, Ilya (1 March 2012). "NoSQL Data Modeling Techniques". Ilya Katsov. Retrieved 8 May 2014.

[25] http://azure.microsoft.com/en-gb/services/storage/tables/

[26] http://azure.microsoft.com/en-gb/services/documentdb/

[27] Scofield, Ben (2010-01-14). "NoSQL - Death to Relational Databases(?)". Retrieved 2014-06-26.

[28] "Making the Shift from Relational to NoSQL" (PDF). *Couchbase.com*. Retrieved December 5, 2014.

[29] http://www.gennet.com/big-data/cant-joins-marklogic-just-matter-semantics/

2.5.8 Further reading

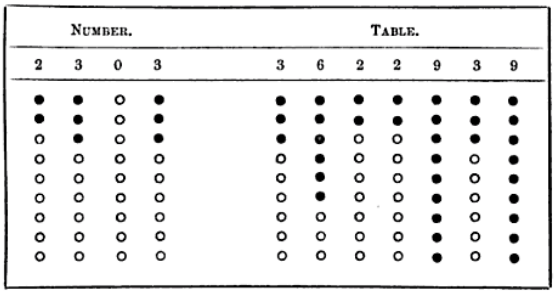
- Sadalage, Pramod; Fowler, Martin (2012). *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence*. Addison-Wesley. ISBN 0-321-82662-0.
- McCreary, Dan; Kelly, Ann (2013). *Making Sense of NoSQL: A guide for managers and the rest of us*. ISBN 9781617291074.
- Strauch, Christof (2012). "NoSQL Databases" (PDF).
- Moniruzzaman, A. B.; Hossain, S. A. (2013). "NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison". arXiv:1307.0191.
- Orend, Kai (2013). "Analysis and Classification of NoSQL Databases and Evaluation of their Ability to Replace an Object-relational Persistence Layer". CiteSeerX: 10.1.1.184.483.
- Krishnan, Ganesh; Kulkarni, Sarang; Dadbhawala, Dharmesh Kirit. "Method and system for versioned sharing, consolidating and reporting information".

2.5.9 External links

- Strauch, Christoph. "NoSQL whitepaper" (PDF). Stuttgart: Hochschule der Medien.
- Edlich, Stefan. "NoSQL database List".
- Neubauer, Peter (2010). "Graph Databases, NOSQL and Neo4j".
- Bushik, Sergey (2012). "A vendor-independent comparison of NoSQL databases: Cassandra, HBase, MongoDB, Riak". NetworkWorld.
- Zicari, Roberto V. (2014). "NoSQL Data Stores – Articles, Papers, Presentations". *odbms.org*.

2.6 Key-value database

A **key-value store**, or **key-value database**, is a data storage paradigm designed for storing, retrieving, and managing associative arrays, a data structure more commonly known today as a *dictionary* or *hash*. 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 quickly find the data within the database.



A tabular data card proposed for Babbage's Analytical Engine showing a key-value pair, in this instance a number and its base-ten logarithm.

Key-value stores work in a very different fashion than 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 as in most RDBs, key-value stores often use far less memory to store the same database, which can lead to large performance gains in certain workloads.

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. A subclass of the key-value store is the document-oriented database, which offers additional tools that use the metadata in the data to provide a richer key-value database that more closely matches the use patterns of RDBM systems. Some graph databases are also key-value stores internally, adding the concept of the relationships (*pointers*) between records as a first class data type.

2.6.1 Types and notable examples

Key-value stores 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.

Redis was the most popular implementation of a key-value database as of August 2015, according to DB-

Engines Ranking.[1]

Another example of key-value database is Oracle NoSQL Database. Oracle NoSQL Database provides a key-value paradigm to the application developer. 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 entity 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]

The Unix system provides dbm (DataBase Manager) which is a library originally written by Ken Thompson. The dbm manages associative arrays of arbitrary data by use of a single key (a primary key). Modern implementations include ndbm, sdbm and GNU dbm.

KV - eventually consistent

- Apache Cassandra
- Dynamo
- Oracle NoSQL Database
- Project Voldemort
- Riak[3]
- OpenLink Virtuoso

KV - ordered

- Berkeley DB
- FairCom c-treeACE/c-treeRTG
- FoundationDB
- HyperDex
- IBM Informix C-ISAM
- InfinityDB
- LMDB
- MemcacheDB
- NDBM

KV - RAM

- Aerospike
- Coherence
- FairCom c-treeACE
- Hazelcast
- memcached
- OpenLink Virtuoso
- Redis
- XAP
- Gemfire

KV - solid-state drive or rotating disk

- Aerospike
- BigTable
- CDB
- Clusterpoint Database Server
- Couchbase Server
- FairCom c-treeACE
- GT.M[4]
- Hibari
- Keyspace
- LevelDB
- LMDB
- MemcacheDB (using Berkeley DB or LMDB)
- MongoDB
- NoSQLz
- Coherence
- Oracle NoSQL Database
- OpenLink Virtuoso
- Tarantool
- Tokyo Cabinet
- Tuple space

2.6.2 References

[1] http://db-engines.com/en/ranking

[2] "Oracle NoSQL Database"

[3] "Riak: An Open Source Scalable Data Store". 28 November 2010. Retrieved 28 November 2010.

[4] Tweed, Rob; James, George (2010). "A Universal NoSQL Engine, Using a Tried and Tested Technology" (PDF). p. 25. Without exception, the most successful and well-known of the NoSQL databases have been developed from scratch, all within just the last few years. Strangely, it seems that nobody looked around to see whether there were any existing, successfully implemented database technologies that could have provided a sound foundation for meeting Web-scale demands. Had they done so, they might have discovered two products, GT.M and Caché.....*

2.6.3 External links

- Ranking of key-value databases by popularity

2.7 Document-oriented database

This article is about the software type. For usage/deployment instances, see Full text database.

A **document-oriented database** or **document store** is a computer program designed for storing, retrieving, and managing document-oriented information, also known as semi-structured data. Document-oriented databases are one of the main categories of NoSQL databases and the popularity of the term "document-oriented database" has grown[1] with the use of the term NoSQL itself.

Document-oriented databases are inherently a subclass of the key-value store, another NoSQL database concept. The difference lies in the way the data is processed; in a key-value store the data is considered to be inherently opaque to the database, whereas a document-oriented system relies on internal structure in the *document* order to extract metadata that the database engine uses for further optimization. Although the difference is often moot due to tools in the systems,[lower-alpha 1] conceptually the document-store is designed to offer a richer experience with modern programming techniques. XML databases are a specific subclass of document-oriented databases that are optimized to extract their metadata from XML documents.

Document databases[lower-alpha 2] contrast strongly with the traditional relational database (RDB). Relational databases are strongly typed during database creation, and store repeated data in separate *table*s that are defined by the programmer. In an RDB, every instance of data has the same format as every other, and changing that format is generally difficult. Document databases get their type information from the data itself, normally store all related information together, and allow every instance of data to be different from any other. This makes them more flexible in dealing with change and optional values, maps more easily into program objects, and often reduces database size. This makes them attractive for programming modern web applications, which are subject to continual change in place, and speed of deployment is an important issue.

2.7.1 Documents

The central concept of a document-oriented database are the *documents*, which is used in usual English sense of a group of data that encodes some sort of user-readable information. This contrasts with the *value* in the key-value store, which is assumed to be opaque data. The basic concept that makes a database document-oriented as opposed to key-value is the idea that the documents include internal structure, or metadata, that the database engine can use to further automate the storage and provide more value.

To understand the difference, consider this text document:

Bob Smith 123 Back St. Boys, AR, 32225 US

Although it is clear to the reader that this document contains the address for a contact, there is no information within the document that indicates that, nor information on what the individual fields represent. This file could be stored in a key-value store, but the semantic content that this is an address may be lost, and the database has no way to know how to optimize or index this data by itself. For instance, there is no way for the database to know that "AR" is the state and add it to an index of states, it is simply a piece of data in a string that also includes the city and zip code. It is possible to add additional logic to deconstruct the string into fields, to extract the state by looking for the middle item of three comma separated values in the 3rd line, but this is not a simple task. For instance, if another line is added to the address, adding a PO Box or suite number for instance, the state information is in the 4th line instead of 3rd. Without additional information, parsing free form data of this sort can be complex.

Now consider the same document marked up in pseudo-XML:

<contact> <firstname>Bob</firstname> <lastname>Smith</lastname> <street1>123 Back St.</street1> <city>Boys</city> <state>AR</state> <zip>32225</zip> <country>US</country> </contact>

In this case, the document includes both data and the metadata explaining each of the fields. A key-value store receiving this document would simply store it. In the case of a document-store, the system understands that contact documents may have a state field, allowing the programmer to "find all the <contact>s where the <state> is 'AR'". Additionally, the programmer can provide hints based on the document type or fields within it, for instance, they may tell the engine to place all <contact> documents in a separate physical store, or to make an index on the state field for performance reasons. All of this can be done in a key-value store as well, and the difference lies pri-

marily in how much programming effort is needed to add these indexes and other features; in a document-store this is normally almost entirely automated.

Now consider a slightly more complex example:

<contact> <firstname>Bob</firstname> <lastname>Smith</lastname> <email type="Home">bob.smith@example.com</email> <phone type="Cell">(123) 555-0178</phone> <phone type="Work">(890) 555-0133</phone> <address> <type>Home</type> <street1>123 Back St.</street1> <city>Boys</city> <state>AR</state> <zip>32225</zip> <country>US</country> </address> </contact>

In this case a number of the fields are either repeated or split out into separate containers in the case of <address>. With similar hints, the document store will allow searches for things like "find all my <contact>s with a <phone> of type <work> but does not have an <email> of type <work>". This is not unlike other database systems in terms of retrieval. What is different is that these fields are defined by the metadata in the document itself. There is no need to pre-define these fields in the database.

This is another major advantage of the document-oriented concept; a single database can contain both of these <contact> objects in the same store, and more generally, every document in the database can have a different format. It is very common for a particular type of document to differ from instance to instance; one <contact> might have a work email, another might not, one might have a single address, another might have several. More widely, the database can store completely unrelated documents, yet still understand that parts of the data within them are the same. For instance, one could construct a query that would look for any document that has the <state> 'AR', it doesn't matter that the documents might be <contact>s or <business>es, or if the <state> is within an <address> or not.

In addition to making it easier to handle different types of data, the metadata also allows the document format to be changed at any time without affecting the existing records. If one wishes to add an <image> field to their contact book application some time in the future, they simply add it. Existing documents will still work fine without being changed in the database, they simply won't have an image. Fields can be added at any time, anywhere, with no need to change the physical storage.

The usefulness of this sort of introspection of the data is not lost on the designers of other database systems. Many key-value stores include some or all of the functionality of dedicated from the start document stores, and a number of relational databases, notably PostgreSQL and Informix, have added functionality to make these sorts of operations possible. It is not the ability to provide these functions that define the document-orientation, but the ease with which these functions can be implemented and used; a document-oriented database is designed from the start to work with complex documents, and will (hopefully) make it easier to access this functionality than a system where this was added after the fact.

Practically any "document" containing metadata can be managed in this fashion, and common examples include XML, YAML, JSON, and BSON. Some document-oriented databases include functionality to help map data lacking clearly defined metadata. For instance, many engines include functionality to index PDF or TeX documents, or may include predefined document formats that are in turn based on XML, like MathML, JATS or DocBook. Some allow documents to be mapped onto a more suitable format using a schema language such as DTD, XSD, Relax NG, or Schematron. Others may include tools to map enterprise data, like column-delimited text files, into formats that can be read more easily by the database engine. Still others take the opposite route, and are dedicated to one type of data format, JSON. JSON is widely used in online programming for interactive web pages and mobile apps, and a niche has appeared for document stores dedicated to efficiently handling them.

Some of the most popular Web sites are document databases, including the many collections of articles at pubmed.gov or major journal publishers; Wikipedia and its kin; and even search engines (though many of those store links to indexed documents, rather than the full documents themselves).

Keys and retrieval

Documents may be addressed in the database via a unique *key* that represents that document. This key is often a simple string, a URI, or a path. The key can be used to retrieve the document from the database. Typically, the database retains an index on the key to speed up document retrieval. The most primitive document databases may do little more than that. However, modern document-oriented databases provide far more, because they extract and index all kinds of metadata, and usually also the entire data content, of the documents. Such databases offer a query language that allows the user to retrieve documents based on their content. For example, you may want to retrieve all the documents whose date falls within some range, that contains a citation to another document, etc.. The set of query APIs or query language features available, as well as the expected performance of the queries, varies significantly from one implementation to the next.

Organization

Implementations offer a variety of ways of organizing documents, including notions of:

- Collections

- Tags
- Non-visible Metadata
- Directory hierarchies
- Buckets

2.7.2 Comparison with relational databases

In a relational database, data is first categorized into a number of predefined types, and *tables* are created to hold individual entries, or *records*, of each type. The tables define the data within each record's *fields*, meaning that every record in the table has the same overall form. The administrator also defines the *relations* between the tables, and selects certain fields that they believe will be most commonly used for searching and defines *indexes* on them. A key concept in the relational design is that any data that may be repeated is placed in its own table, and if these instances are related to each other, a field is selected to group them together, the *foreign key*.

For example, an address book application will generally need to store the contact name, an optional image, one or more phone numbers, one or more mailing addresses, and one or more email addresses. In a canonical relational database solution, tables would be created for each of these records with predefined fields for each bit of data: the CONTACT table might include FIRST_NAME, LAST_NAME and IMAGE fields, while the PHONE_NUMBER table might include COUNTRY_CODE, AREA_CODE, PHONE_NUMBER and TYPE (home, work, etc). The PHONE_NUMBER table also contains a foreign key field, "CONTACT_ID", which holds the unique ID number assigned to the contact when it was created. In order to recreate the original contact, the system has to search through all of the tables and collect the information back together using *joins*.

In contrast, in a document-oriented database there may be no internal structure that maps directly onto the concept of a table, and the fields and relations generally don't exist as predefined concepts. Instead, all of the data for an object is placed in a single *document*, and stored in the database as a single entry. In the address book example, the document would contain the contact's name, image, and any contact info, all in a single record. That entry is accessed through a *key*, some unique bit of data, which allows the database to retrieve and return the document to the application. No additional work is needed to retrieve the related data; all of this is returned in a single object.

A key difference between the document-oriented and relational models is that the data formats are not predefined in the document case. In most cases, any sort of document can be stored in any database, and those documents can change in type and form at any time. If one wishes to add a COUNTRY_FLAG to a CONTACT, simply add this field to new documents as they are inserted, this will have no effect on the database or the existing documents already stored, they simply won't have this field. This indicates an advantage of the document-based model: optional fields are truly optional, so a contact that does not include a mailing address simply does not have a mailing address, and there is no need to check another table to see if there are entries.

To aid retrieval of information from the database, document-oriented systems generally allow the administrator to provide *hints* to the database to look for certain types of information. In the address book example, the design might add hints for the first and last name fields. When the document is inserted into the database (or later modified), the database engine looks for these bits of information and indexes them, in the same fashion as the relational model. Additionally, most document-oriented databases allow documents to have a *type* associated with them, like "address book entry", which allows the programmer to retrieve related types of information, like "all the address book entries". This provides functionality similar to a table, but separates the concept (categories of data) from its physical implementation (tables).

All of this is predicated on the ability of the database engine to examine the data in the document and extract fields from the formatting, its *metadata*. This is easy in the case of, for example, an XML document or HTML page, where *markup tags* clearly identify various bits of data. Document-oriented databases may include functionality to automatically extract this sort of information from a variety of document types, even those that were not originally designed for easy access in this manner. In other cases the programmer has to provide this information using their own code. In contrast, a relational database relies on the programmer to handle all of these tasks, breaking down the document into fields and providing those to the database engine, which may require separate instructions if the data spans tables.

Document-oriented databases normally map more cleanly onto existing programming concepts, like object-oriented programming (OOP). OOP systems have a structure somewhere between the relational and document models; they have predefined fields but they may be empty, they have a defined structure but that may change, they have related data store in other objects, but they may be optional, and collections of other data are directly linked to the "master" object; there is no need to look in other collections to gather up related information. Generally, any object that can be *archived* to a document can be stored directly in the database and directly retrieved. Most modern OOP systems include archiving systems as a basic feature.

The relational model stores each part of the object as a separate concept and has to split out this information

on storage and recombine it on retrieval. This leads to a problem known as object-relational impedance mismatch, which requires considerable effort to overcome. Object-relational mapping systems, which solve these problems, are often complex and have a considerable performance overhead. This problem simply doesn't exist in a document-oriented system, and more generally, in NoSQL systems as a whole.

2.7.3 Implementations

Main category: Document-oriented databases

XML database implementations

Further information: XML database

Most XML databases are document-oriented databases.

2.7.4 See also

- Database theory
- Data hierarchy
- Full text search
- In-memory database
- Internet Message Access Protocol (IMAP)
- NoSQL
- Object database
- Online database
- Real time database
- Relational database

2.7.5 Notes

[1] To the point that document-oriented and key-value systems can often be interchanged in operation.

[2] And key-value stores in general.

2.7.6 References

[1] DB-Engines Ranking per database model category

[2] Document-oriented Database. Clusterpoint. Retrieved on 2015-10-08.

[3] Documentation. Couchbase. Retrieved on 2013-09-18.

[4] CouchDB Overview

[5] CouchDB Document API

[6]

[7] eXist-db Open Source Native XML Database. Exist-db.org. Retrieved on 2013-09-18.

[8] http://www.ibm.com/developerworks/data/library/techarticle/dm-0801doe/

[9] http://developer.marklogic.com/licensing

[10] MongoDB Licensing

[11] Additional 30+ community MongoDB supported drivers

[12] MongoDB REST Interfaces

[13] GTM MUMPS FOSS on SourceForge

2.7.7 Further reading

- Assaf Arkin. (2007, September 20). Read Consistency: Dumb Databases, Smart Services. Labnotes: Don't let the bubble go to your head!

2.7.8 External links

- DB-Engines Ranking of Document Stores by popularity, updated monthly

2.8 NewSQL

NewSQL is a class of modern relational database management systems that seek to provide the same scalable performance of NoSQL systems for online transaction processing (OLTP) read-write workloads while still maintaining the ACID guarantees of a traditional database system.[1][2][3]

2.8.1 History

The term was first used by 451 Group analyst Matthew Aslett in a 2011 research paper discussing the rise of new database systems as challengers to established vendors.[4] Many enterprise systems that handle high-profile data (e.g., financial and order processing systems) also need to be able to scale but are unable to use NoSQL solutions because they cannot give up strong transactional and consistency requirements.[4][5] The only options previously available for these organizations were to either purchase a more powerful single-node machine or develop custom middleware that distributes queries over traditional DBMS nodes. Both approaches are prohibitively expensive and thus are not an option for many. Thus, in this paper, Aslett discusses how NewSQL upstarts are poised to challenge the supremacy of commercial vendors, in particular Oracle.

2.8.2 Systems

Although NewSQL systems vary greatly in their internal architectures, the two distinguishing features common amongst them is that they all support the relational data model and use SQL as their primary interface.[6] The applications targeted by these NewSQL systems are characterized as having a large number of transactions that (1) are short-lived (i.e., no user stalls), (2) touch a small subset of data using index lookups (i.e., no full table scans or large distributed joins), and (3) are repetitive (i.e. executing the same queries with different inputs).[7] These NewSQL systems achieve high performance and scalability by eschewing much of the legacy architecture of the original IBM System R design, such as heavyweight recovery or concurrency control algorithms.[8] One of the first known NewSQL systems is the H-Store parallel database system.[9][10]

NewSQL systems can be loosely grouped into three categories: [11][12]

New architectures

The first type of NewSQL systems are completely new database platforms. These are designed to operate in a distributed cluster of shared-nothing nodes, in which each node owns a subset of the data. These databases are often written from scratch with a distributed architecture in mind, and include components such as distributed concurrency control, flow control, and distributed query processing. Example systems in this category are Google Spanner, Clustrix, VoltDB, MemSQL, Pivotal's GemFire XD, SAP HANA,[13] NuoDB, and Trafodion.[14]

SQL engines

The second category are highly optimized storage engines for SQL. These systems provide the same programming interface as SQL, but scale better than built-in engines, such as InnoDB. Examples of these new storage engines include MySQL Cluster, Infobright, TokuDB and the now defunct InfiniDB.

Transparent sharding

These systems provide a sharding middleware layer to automatically split databases across multiple nodes. Examples of this type of system includes dbShards and ScaleBase.

2.8.3 See also

- Transaction processing
- Partition (database)

2.8.4 References

[1] Aslett, Matthew (2011). "How Will The Database Incumbents Respond To NoSQL And NewSQL?" (PDF). 451 Group (published 2011-04-04). Retrieved 2012-07-06.

[2] Stonebraker, Michael (2011-06-16). "NewSQL: An Alternative to NoSQL and Old SQL for New OLTP Apps". Communications of the ACM Blog. Retrieved 2012-07-06.

[3] Hoff, Todd (2012-09-24). "Google Spanner's Most Surprising Revelation: NoSQL is Out and NewSQL is In". Retrieved 2012-10-07.

[4] Aslett, Matthew (2010). "What we talk about when we talk about NewSQL". 451 Group (published 2011-04-06). Retrieved 2012-10-07.

[5] Lloyd, Alex (2012). "Building Spanner". Berlin Buzzwords (published 2012-06-05). Retrieved 2012-10-07.

[6] Cattell, R. (2011). "Scalable SQL and NoSQL data stores" (PDF). *ACM SIGMOD Record* **39** (4): 12. doi:10.1145/1978915.1978919.

[7] Stonebraker, Mike; et al. (2007). "The end of an architectural era: (it's time for a complete rewrite" (PDF). *VLDB '07: Proceedings of the 33rd international conference on Very large data bases*. Vienna, Austria.

[8] Stonebraker, M.; Cattell, R. (2011). "10 rules for scalable performance in 'simple operation' datastores". *Communications of the ACM* **54** (6): 72. doi:10.1145/1953122.1953144.

[9] Aslett, Matthew (2008). "Is H-Store the future of database management systems?" (published 2008-03-04). Retrieved 2012-07-05.

[10] Dignan, Larry (2008). "H-Store: Complete destruction of the old DBMS order?". Retrieved 2012-07-05.

[11] Venkatesh, Prasanna (2012). "NewSQL - The New Way to Handle Big Data" (published 2012-01-30). Retrieved 2012-10-07.

[12] Levari, Doron (2011). "The NewSQL Market Breakdown". Retrieved 2012-04-08.

[13] "SAP HANA". SAP. Retrieved 17 July 2014.

[14] "Trafodion: Transactional SQL-on-HBase". 2014.

Chapter 3

ACID

3.1 ACID

For other uses, see Acid (disambiguation).

In computer science, **ACID** (*Atomicity, Consistency, Isolation, Durability*) is a set of properties that guarantee that database transactions are processed reliably. In the context of databases, a single logical operation on the data is called a transaction. For example, a transfer of funds from one bank account to another, even involving multiple changes such as debiting one account and crediting another, is a single transaction.

Jim Gray defined these properties of a reliable transaction system in the late 1970s and developed technologies to achieve them automatically.[1][2][3]

In 1983, Andreas Reuter and Theo Härder coined the acronym *ACID* to describe them.[4]

3.1.1 Characteristics

The characteristics of these four properties as defined by Reuter and Härder:

Atomicity

Main article: Atomicity (database systems)

Atomicity requires that each transaction be “all or nothing": if one part of the transaction fails, the entire transaction fails, and the database state is left unchanged. An atomic system must guarantee atomicity in each and every situation, including power failures, errors, and crashes. To the outside world, a committed transaction appears (by its effects on the database) to be indivisible (“atomic”), and an aborted transaction does not happen.

Consistency

Main article: Consistency (database systems)

The consistency property ensures that any transaction will bring the database from one valid state to another. Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This does not guarantee correctness of the transaction in all ways the application programmer might have wanted (that is the responsibility of application-level code) but merely that any programming errors cannot result in the violation of any defined rules.

Isolation

Main article: Isolation (database systems)

The isolation property ensures that the concurrent execution of transactions results in a system state that would be obtained if transactions were executed serially, i.e., one after the other. Providing isolation is the main goal of concurrency control. Depending on concurrency control method (i.e. if it uses strict - as opposed to relaxed - serializability), the effects of an incomplete transaction might not even be visible to another transaction.

Durability

Main article: Durability (database systems)

The durability property ensures that once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors. In a relational database, for instance, once a group of SQL statements execute, the results need to be stored permanently (even if the database crashes immediately thereafter). To defend against power loss, transactions (or their effects) must be recorded in a non-volatile memory.

3.1.2 Examples

The following examples further illustrate the ACID properties. In these examples, the database table has two columns, A and B. An integrity constraint requires that the value in A and the value in B must sum to 100. The following SQL code creates a table as described above:

CREATE TABLE acidtest (A INTEGER, B INTEGER, CHECK (A + B = 100));

Atomicity failure

In database systems, atomicity (or atomicness; from Greek a-tomos, undividable) is one of the ACID transaction properties. In an atomic transaction, a series of database operations either all occur, or nothing occurs. The series of operations cannot be divided apart and executed partially from each other, which makes the series of operations "indivisible", hence the name. A guarantee of atomicity prevents updates to the database occurring only partially, which can cause greater problems than rejecting the whole series outright. In other words, atomicity means indivisibility and irreducibility.

Consistency failure

Consistency is a very general term, which demands that the data must meet all validation rules. In the previous example, the validation is a requirement that A + B = 100. Also, it may be inferred that both A and B must be integers. A valid range for A and B may also be inferred. All validation rules must be checked to ensure consistency. Assume that a transaction attempts to subtract 10 from A without altering B. Because consistency is checked after each transaction, it is known that A + B = 100 before the transaction begins. If the transaction removes 10 from A successfully, atomicity will be achieved. However, a validation check will show that A + B = 90, which is inconsistent with the rules of the database. The entire transaction must be cancelled and the affected rows rolled back to their pre-transaction state. If there had been other constraints, triggers, or cascades, every single change operation would have been checked in the same way as above before the transaction was committed.

Isolation failure

To demonstrate isolation, we assume two transactions execute at the same time, each attempting to modify the same data. One of the two must wait until the other completes in order to maintain isolation.

Consider two transactions. T_1 transfers 10 from A to B. T_2 transfers 10 from B to A. Combined, there are four actions:

- T_1 subtracts 10 from A.
- T_1 adds 10 to B.
- T_2 subtracts 10 from B.
- T_2 adds 10 to A.

If these operations are performed in order, isolation is maintained, although T_2 must wait. Consider what happens if T_1 fails half-way through. The database eliminates T_1's effects, and T_2 sees only valid data.

By interleaving the transactions, the actual order of actions might be:

- T_1 subtracts 10 from A.
- T_2 subtracts 10 from B.
- T_2 adds 10 to A.
- T_1 adds 10 to B.

Again, consider what happens if T_1 fails halfway through. By the time T_1 fails, T_2 has already modified A; it cannot be restored to the value it had before T_1 without leaving an invalid database. This is known as a write-write failure, because two transactions attempted to write to the same data field. In a typical system, the problem would be resolved by reverting to the last known good state, canceling the failed transaction T_1, and restarting the interrupted transaction T_2 from the good state.

Durability failure

Consider a transaction that transfers 10 from A to B. First it removes 10 from A, then it adds 10 to B. At this point, the user is told the transaction was a success, however the changes are still queued in the disk buffer waiting to be committed to disk. Power fails and the changes are lost. The user assumes (understandably) that the changes have been persisted.

3.1.3 Implementation

Processing a transaction often requires a sequence of operations that is subject to failure for a number of reasons. For instance, the system may have no room left on its disk drives, or it may have used up its allocated CPU time. There are two popular families of techniques: write-ahead logging and shadow paging. In both cases, locks must be acquired on all information to be updated, and depending on the level of isolation, possibly on all data that be read as well. In write ahead logging, atomicity is guaranteed by copying the original (unchanged) data to a log before changing the database. That allows the database to return to a consistent state in the event of a crash. In shadowing, updates are applied to a partial copy of the database, and the new copy is activated when the transaction commits.

Locking vs multiversioning

Many databases rely upon locking to provide ACID capabilities. Locking means that the transaction marks the

data that it accesses so that the DBMS knows not to allow other transactions to modify it until the first transaction succeeds or fails. The lock must always be acquired before processing data, including data that is read but not modified. Non-trivial transactions typically require a large number of locks, resulting in substantial overhead as well as blocking other transactions. For example, if user A is running a transaction that has to read a row of data that user B wants to modify, user B must wait until user A's transaction completes. Two phase locking is often applied to guarantee full isolation.

An alternative to locking is multiversion concurrency control, in which the database provides each reading transaction the prior, unmodified version of data that is being modified by another active transaction. This allows readers to operate without acquiring locks, i.e. writing transactions do not block reading transactions, and readers do not block writers. Going back to the example, when user A's transaction requests data that user B is modifying, the database provides A with the version of that data that existed when user B started his transaction. User A gets a consistent view of the database even if other users are changing data. One implementation, namely snapshot isolation, relaxes the isolation property.

Distributed transactions

Main article: Distributed transaction

Guaranteeing ACID properties in a distributed transaction across a distributed database, where no single node is responsible for all data affecting a transaction, presents additional complications. Network connections might fail, or one node might successfully complete its part of the transaction and then be required to roll back its changes because of a failure on another node. The two-phase commit protocol (not to be confused with two-phase locking) provides atomicity for distributed transactions to ensure that each participant in the transaction agrees on whether the transaction should be committed or not. Briefly, in the first phase, one node (the coordinator) interrogates the other nodes (the participants) and only when all reply that they are prepared does the coordinator, in the second phase, formalize the transaction.

3.1.4 See also

- Basically Available, Soft state, Eventual consistency (BASE)
- CAP theorem
- Concurrency control
- Java Transaction API
- Open Systems Interconnection
- Transactional NTFS
-

3.1.5 References

[1] "Gray to be Honored With A.M. Turing Award This Spring". Microsoft PressPass. Archived from the original on February 6, 2009. Retrieved March 27, 2015.

[2] Gray, Jim (September 1981). "The Transaction Concept: Virtues and Limitations" (PDF). *Proceedings of the 7th International Conference on Very Large Databases*. Cupertino, CA: Tandem Computers. pp. 144–154. Retrieved March 27, 2015.

[3] Gray, Jim & Andreas Reuter. *Distributed Transaction Processing: Concepts and Techniques*. Morgan Kaufmann, 1993; ISBN 1-55860-190-2.

[4] Haerder, T.; Reuter, A. (1983). "Principles of transaction-oriented database recovery". *ACM Computing Surveys* **15** (4): 287. doi:10.1145/289.291. These four properties, atomicity, consistency, isolation, and durability (ACID), describe the major highlights of the transaction paradigm, which has influenced many aspects of development in database systems.

3.2 Consistency (database systems)

Consistency in database systems refers to the requirement that any given database transaction must change affected data only in allowed ways. Any data written to the database must be valid according to all defined rules, including constraints, cascades, triggers, and any combination thereof. This does not guarantee correctness of the transaction in all ways the application programmer might have wanted (that is the responsibility of application-level code) but merely that any programming errors cannot result in the violation of any defined rules.

3.2.1 As an ACID guarantee

Consistency is one of the four guarantees that define ACID transactions; however, significant ambiguity exists about the nature of this guarantee. It is defined variously as:

- The guarantee that any transactions started in the future necessarily see the effects of other transactions committed in the past[1][2]
- The guarantee that database constraints are not violated, particularly once a transaction commits[3][4][5][6]
- The guarantee that operations in transactions are performed accurately, correctly, and with validity, with respect to application semantics[7]

As these various definitions are not mutually exclusive, it is possible to design a system that guarantees "consistency" in every sense of the word, as most relational database management systems in common use today arguably do.

3.2.2 As a CAP trade-off

The CAP Theorem is based on three trade-offs, one of which is "atomic consistency" (shortened to "consistency" for the acronym), about which the authors note, "Discussing atomic consistency is somewhat different than talking about an ACID database, as database consistency refers to transactions, while atomic consistency refers only to a property of a single request/response operation sequence. And it has a different meaning than the Atomic in ACID, as it subsumes the database notions of both Atomic and Consistent."[1]

3.2.3 See also

- Consistency model
- CAP Theorem
- Eventual consistency

3.2.4 References

[1] http://webpages.cs.luc.edu/~{}pld/353/gilbert_lynch_brewer_proof.pdf "Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services"

[2] Ports, D.R.K, Clements, A.T, Zhang, I, Madden, S, Liskov, B. "Transactional Consistency and Automatic Management in an Application Data Cache" (PDF). *MIT CSAIL*.

[3] Haerder, T, Reuter, A. (December 1983). "Principles of Transaction-Oriented Database Recovery" (PDF). *Computing Surveys* **15** (4): 287–317.

[4] Mike Chapple. "The ACID Model". *About*.

[5] "ACID properties".

[6] Cory Janssen. "What is ACID in Databases? - Definition from Techopedia". *Techopedia.com*.

[7] "ISO/IEC 10026-1:1998 - Information technology -- Open Systems Interconnection -- Distributed Transaction Processing -- Part 1: OSI TP Model".

3.3 Durability (database systems)

In database systems, **durability** is the ACID property which guarantees that transactions that have committed will survive permanently. For example, if a flight booking reports that a seat has successfully been booked, then the seat will remain booked even if the system crashes.

Durability can be achieved by flushing the transaction's log records to non-volatile storage before acknowledging commitment.

In distributed transactions, all participating servers must coordinate before commit can be acknowledged. This is usually done by a two-phase commit protocol.

Many DBMSs implement durability by writing transactions into a transaction log that can be reprocessed to recreate the system state right before any later failure. A transaction is deemed committed only after it is entered in the log.

3.3.1 See also

- Atomicity
- Consistency
- Isolation
- Relational database management system

Chapter 4

Isolation

4.1 Serializability

In concurrency control of databases,[1][2] transaction processing (transaction management), and various transactional applications (e.g., transactional memory[3] and software transactional memory), both centralized and distributed, a transaction schedule is **serializable** if its outcome (e.g., the resulting database state) is equal to the outcome of its transactions executed serially, i.e., sequentially without overlapping in time. Transactions are normally executed concurrently (they overlap), since this is the most efficient way. Serializability is the major correctness criterion for concurrent transactions' executions. It is considered the highest level of isolation between transactions, and plays an essential role in concurrency control. As such it is supported in all general purpose database systems. *Strong strict two-phase locking* (SS2PL) is a popular serializability mechanism utilized in most of the database systems (in various variants) since their early days in the 1970s.

Serializability theory provides the formal framework to reason about and analyze serializability and its techniques. Though it is mathematical in nature, its fundamentals are informally (without mathematics notation) introduced below.

4.1.1 Database transaction

Main article: Database transaction

For this is a specific intended run (with specific parameters, e.g., with transaction identification, at least) of a computer program (or programs) that accesses a database (or databases). Such a program is written with the assumption that it is running in *isolation* from other executing programs, i.e., when running, its accessed data (after the access) are not changed by other running programs. Without this assumption the transaction's results are unpredictable and can be wrong. The same transaction can be executed in different situations, e.g., in different times and locations, in parallel with different programs. A *live* transaction (i.e., exists in a computing environment with already allocated computing resources; to distinguish from a *transaction request*, waiting to get execution resources) can be in one of three states, or phases:

1. *Running* - Its program(s) is (are) executing.
2. *Ready* - Its program's execution has ended, and it is waiting to be *Ended (Completed)*.
3. *Ended* (or *Completed*) - It is either *Committed* or *Aborted (Rolled-back)*, depending whether the execution is considered a success or not, respectively . When committed, all its *recoverable* (i.e., with states that can be controlled for this purpose), *durable* resources (typically *database data*) are put in their *final* states, states after running. When aborted, all its recoverable resources are put back in their *initial* states, as before running.

A failure in transaction's computing environment before ending typically results in its abort. However, a transaction may be aborted also for other reasons as well (e.g., see below).

Upon being ended (completed), transaction's allocated computing resources are released and the transaction disappears from the computing environment. However, the effects of a committed transaction remain in the database, while the effects of an aborted (rolled-back) transaction disappear from the database. The concept of *atomic transaction* ("all or nothing" semantics) was designed to exactly achieve this behavior, in order to control correctness in complex faulty systems.

4.1.2 Correctness

Correctness - serializability

Serializability is a property of a transaction schedule (history). It relates to the *isolation* property of a database transaction.

> **Serializability** of a schedule means equivalence (in the outcome, the database state, data values) to a *serial schedule* (i.e., sequential with no transaction overlap in time) with the same

transactions. It is the major criterion for the correctness of concurrent transactions' schedule, and thus supported in all general purpose database systems.

The rationale behind serializability is the following:

If each transaction is correct by itself, i.e., meets certain integrity conditions, then a schedule that comprises any *serial* execution of these transactions is correct (its transactions still meet their conditions): "Serial" means that transactions do not overlap in time and cannot interfere with each other, i.e, complete *isolation* between each other exists. Any order of the transactions is legitimate, if no dependencies among them exist, which is assumed (see comment below). As a result, a schedule that comprises any execution (not necessarily serial) that is equivalent (in its outcome) to any serial execution of these transactions, is correct.

Schedules that are not serializable are likely to generate erroneous outcomes. Well known examples are with transactions that debit and credit accounts with money: If the related schedules are not serializable, then the total sum of money may not be preserved. Money could disappear, or be generated from nowhere. This and violations of possibly needed other invariant preservations are caused by one transaction writing, and "stepping on" and erasing what has been written by another transaction before it has become permanent in the database. It does not happen if serializability is maintained.

If any specific order between some transactions is requested by an application, then it is enforced independently of the underlying serializability mechanisms. These mechanisms are typically indifferent to any specific order, and generate some unpredictable partial order that is typically compatible with multiple serial orders of these transactions. This partial order results from the scheduling orders of concurrent transactions' data access operations, which depend on many factors.

A major characteristic of a database transaction is *atomicity*, which means that it either *commits*, i.e., all its operations' results take effect in the database, or *aborts* (rolled-back), all its operations' results do not have any effect on the database ("all or nothing" semantics of a transaction). In all real systems transactions can abort for many reasons, and serializability by itself is not sufficient for correctness. Schedules also need to possess the *recoverability* (from abort) property. **Recoverability** means that committed transactions have not read data written by aborted transactions (whose effects do not exist in the resulting database states). While serializability is currently compromised on purpose in many applications for better performance (only in cases when application's correctness is not harmed), compromising recoverability would quickly violate the database's integrity, as well as that of transactions' results external to the database. A schedule with the recoverability property (a *recoverable* schedule) "recovers" from aborts by itself, i.e., aborts do not harm the integrity of its committed transactions and resulting database. This is false without recoverability, where the likely integrity violations (resulting incorrect database data) need special, typically manual, corrective actions in the database.

Implementing recoverability in its general form may result in *cascading aborts*: Aborting one transaction may result in a need to abort a second transaction, and then a third, and so on. This results in a waste of already partially executed transactions, and may result also in a performance penalty. **Avoiding cascading aborts** (ACA, or Cascadelessness) is a special case of recoverability that exactly prevents such phenomenon. Often in practice a special case of ACA is utilized: **Strictness**. Strictness allows an efficient database recovery from failure.

Note that the *recoverability* property is needed even if no database failure occurs and no database *recovery* from failure is needed. It is rather needed to correctly automatically handle aborts, which may be unrelated to database failure and recovery from failure.

Relaxing serializability

In many applications, unlike with finances, absolute correctness is not needed. For example, when retrieving a list of products according to specification, in most cases it does not matter much if a product, whose data was updated a short time ago, does not appear in the list, even if it meets the specification. It will typically appear in such a list when tried again a short time later. Commercial databases provide concurrency control with a whole range of isolation levels which are in fact (controlled) serializability violations in order to achieve higher performance. Higher performance means better transaction execution rate and shorter average transaction response time (transaction duration). *Snapshot isolation* is an example of a popular, widely utilized efficient relaxed serializability method with many characteristics of full serializability, but still short of some, and unfit in many situations.

Another common reason nowadays for distributed serializability relaxation (see below) is the requirement of availability of internet products and services. This requirement is typically answered by large-scale data replication. The straightforward solution for synchronizing replicas' updates of a same database object is including all these updates in a single atomic distributed transaction. However, with many replicas such a transaction is very large, and may span several computers and networks that some of them are likely to be unavailable. Thus such a transaction is likely to end with abort and miss its purpose.[4] Consequently, Optimistic replication (Lazy

replication) is often utilized (e.g., in many products and services by Google, Amazon, Yahoo, and alike), while serializability is relaxed and compromised for eventual consistency. Again in this case, relaxation is done only for applications that are not expected to be harmed by this technique.

Classes of schedules defined by *relaxed serializability* properties either contain the serializability class, or are incomparable with it.

4.1.3 View and conflict serializability

Mechanisms that enforce serializability need to execute in real time, or almost in real time, while transactions are running at high rates. In order to meet this requirement special cases of serializability, sufficient conditions for serializability which can be enforced effectively, are utilized.

Two major types of serializability exist: *view-serializability*, and *conflict-serializability*. View-serializability matches the general definition of serializability given above. Conflict-serializability is a broad special case, i.e., any schedule that is conflict-serializable is also view-serializable, but not necessarily the opposite. Conflict-serializability is widely utilized because it is easier to determine and covers a substantial portion of the view-serializable schedules. Determining view-serializability of a schedule is an NP-complete problem (a class of problems with only difficult-to-compute, excessively time-consuming known solutions).

> **View-serializability** of a schedule is defined by equivalence to a serial schedule (no overlapping transactions) with the same transactions, such that respective transactions in the two schedules read and write the same data values ("view" the same data values).
>
> **Conflict-serializability** is defined by equivalence to a serial schedule (no overlapping transactions) with the same transactions, such that both schedules have the same sets of respective chronologically ordered pairs of conflicting operations (same precedence relations of respective conflicting operations).

Operations upon data are *read* or *write* (a write: either *insert* or *modify* or *delete*). Two operations are *conflicting*, if they are of different transactions, upon the same datum (data item), and at least one of them is *write*. Each such pair of conflicting operations has a *conflict type*: It is either a *read-write*, or *write-read*, or a *write-write* conflict. The transaction of the second operation in the pair is said to be *in conflict* with the transaction of the first operation. A more general definition of conflicting operations (also for complex operations, which may consist each of several "simple" read/write operations) requires that they are noncommutative (changing their order also changes their combined result). Each such operation needs to be atomic by itself (by proper system support) in order to be considered an operation for a commutativity check. For example, read-read operations are commutative (unlike read-write and the other possibilities) and thus read-read is not a conflict. Another more complex example: the operations *increment* and *decrement* of a *counter* are both *write* operations (both modify the counter), but do not need to be considered conflicting (write-write conflict type) since they are commutative (thus increment-decrement is not a conflict; e.g., already has been supported in the old IBM's IMS "fast path"). Only precedence (time order) in pairs of conflicting (non-commutative) operations is important when checking equivalence to a serial schedule, since different schedules consisting of the same transactions can be transformed from one to another by changing orders between different transactions' operations (different transactions' interleaving), and since changing orders of commutative operations (non-conflicting) does not change an overall operation sequence result, i.e., a schedule outcome (the outcome is preserved through order change between non-conflicting operations, but typically not when conflicting operations change order). This means that if a schedule can be transformed to any serial schedule without changing orders of conflicting operations (but changing orders of non-conflicting, while preserving operation order inside each transaction), then the outcome of both schedules is the same, and the schedule is conflict-serializable by definition.

Conflicts are the reason for blocking transactions and delays (non-materialized conflicts), or for aborting transactions due to serializability violations prevention. Both possibilities reduce performance. Thus reducing the number of conflicts, e.g., by commutativity (when possible), is a way to increase performance.

A transaction can issue/request a conflicting operation and be *in conflict* with another transaction while its conflicting operation is delayed and not executed (e.g., blocked by a lock). Only executed (*materialized*) conflicting operations are relevant to *conflict serializability* (see more below).

4.1.4 Enforcing conflict serializability

Testing conflict serializability

Schedule compliance with conflict serializability can be tested with the precedence graph (*serializability graph*, *serialization graph*, *conflict graph*) for committed transactions of the schedule. It is the directed graph representing precedence of transactions in the schedule, as reflected by precedence of conflicting operations in the transactions.

In the **precedence graph** transactions are

nodes and precedence relations are directed edges. There exists an edge from a first transaction to a second transaction, if the second transaction is *in conflict* with the first (see Conflict serializability above), and the conflict is **materialized** (i.e., if the requested conflicting operation is actually executed: in many cases a requested/issued conflicting operation by a transaction is delayed and even never executed, typically by a lock on the operation's object, held by another transaction, or when writing to a transaction's temporary private workspace and materializing, copying to the database itself, upon commit; as long as a requested/issued conflicting operation is not executed upon the database itself, the conflict is **non-materialized**; non-materialized conflicts are not represented by an edge in the precedence graph).

Comment: In many text books only *committed transactions* are included in the precedence graph. Here all transactions are included for convenience in later discussions.

The following observation is a **key characterization of conflict serializability**:

A schedule is *conflict-serializable* if and only if its precedence graph of *committed transactions* (when only *committed* transactions are considered) is *acyclic*. This means that a cycle consisting of committed transactions only is generated in the (general) precedence graph, if and only if conflict-serializability is violated.

Cycles of committed transactions can be prevented by aborting an *undecided* (neither committed, nor aborted) transaction on each cycle in the precedence graph of all the transactions, which can otherwise turn into a cycle of committed transactions (and a committed transaction cannot be aborted). One transaction aborted per cycle is both required and sufficient number to break and eliminate the cycle (more aborts are possible, and can happen in some mechanisms, but unnecessary for serializability). The probability of cycle generation is typically low, but nevertheless, such a situation is carefully handled, typically with a considerable overhead, since correctness is involved. Transactions aborted due to serializability violation prevention are *restarted* and executed again immediately.

Serializability enforcing mechanisms typically do not maintain a precedence graph as a data structure, but rather prevent or break cycles implicitly (e.g., SS2PL below).

Common mechanism - SS2PL

Main article: Two-phase locking

Strong strict two phase locking (SS2PL) is a common mechanism utilized in database systems since their early days in the 1970s (the "SS" in the name SS2PL is newer though) to enforce both conflict serializability and *strictness* (a special case of recoverability which allows effective database recovery from failure) of a schedule. In this mechanism each datum is locked by a transaction before accessing it (any read or write operation): The item is marked by, associated with a *lock* of a certain type, depending on operation (and the specific implementation; various models with different lock types exist; in some models locks may change type during the transaction's life). As a result, access by another transaction may be blocked, typically upon a conflict (the lock delays or completely prevents the conflict from being materialized and be reflected in the precedence graph by blocking the conflicting operation), depending on lock type and the other transaction's access operation type. Employing an SS2PL mechanism means that all locks on data on behalf of a transaction are released only after the transaction has ended (either committed or aborted).

SS2PL is the name of the resulting schedule property as well, which is also called *rigorousness*. SS2PL is a special case (proper subset) of Two-phase locking (2PL)

Mutual blocking between transactions results in a *deadlock*, where execution of these transactions is stalled, and no completion can be reached. Thus deadlocks need to be resolved to complete these transactions' execution and release related computing resources. A deadlock is a reflection of a potential cycle in the precedence graph, that would occur without the blocking when conflicts are materialized. A deadlock is resolved by aborting a transaction involved with such potential cycle, and breaking the cycle. It is often detected using a *wait-for graph* (a graph of conflicts blocked by locks from being materialized; it can be also defined as the graph of non-materialized conflicts; conflicts not materialized are not reflected in the precedence graph and do not affect serializability), which indicates which transaction is "waiting for" lock release by which transaction, and a cycle means a deadlock. Aborting one transaction per cycle is sufficient to break the cycle. Transactions aborted due to deadlock resolution are *restarted* and executed again immediately.

Other enforcing techniques

Other known mechanisms include:

- Precedence graph (or Serializability graph, Conflict graph) cycle elimination
- Two-phase locking (2PL)

- Timestamp ordering (TO)
- Serializable snapshot isolation[5] (SerializableSI)

The above (conflict) serializability techniques in their general form do not provide recoverability. Special enhancements are needed for adding recoverability.

Optimistic versus pessimistic techniques Concurrency control techniques are of three major types:

1. *Pessimistic*: In Pessimistic concurrency control a transaction blocks data access operations of other transactions upon conflicts, and conflicts are *non-materialized* until blocking is removed. This is done to ensure that operations that may violate serializability (and in practice also recoverability) do not occur.
2. *Optimistic*: In Optimistic concurrency control data access operations of other transactions are not blocked upon conflicts, and conflicts are immediately *materialized*. When the transaction reaches the *ready* state, i.e., its *running* state has been completed, possible serializability (and in practice also recoverability) violation by the transaction's operations (relatively to other running transactions) is checked: If violation has occurred, the transaction is typically *aborted* (sometimes aborting *another* transaction to handle serializability violation is preferred). Otherwise it is *committed*.
3. *Semi-optimistic*: Mechanisms that mix blocking in certain situations with not blocking in other situations and employ both materialized and non-materialized conflicts

The main differences between the technique types is the conflict types that are generated by them. A pessimistic method blocks a transaction operation upon conflict and generates a non-materialized conflict, while an optimistic method does not block and generates a materialized conflict. A semi-optimistic method generates both conflict types. Both conflict types are generated by the chronological orders in which transaction operations are invoked, independently of the type of conflict. A cycle of committed transactions (with materialized conflicts) in the *precedence graph* (conflict graph) represents a serializability violation, and should be avoided for maintaining serializability. A cycle of (non-materialized) conflicts in the *wait-for graph* represents a deadlock situation, which should be resolved by breaking the cycle. Both cycle types result from conflicts, and should be broken. At any technique type conflicts should be detected and considered, with similar overhead for both materialized and non-materialized conflicts (typically by using mechanisms like locking, while either blocking for locks, or not blocking but recording conflict for materialized conflicts). In a blocking method typically a context switching occurs upon conflict, with (additional) incurred overhead. Otherwise blocked transactions' related computing resources remain idle, unutilized, which may be a worse alternative. When conflicts do not occur frequently, optimistic methods typically have an advantage. With different transactions loads (mixes of transaction types) one technique type (i.e., either optimistic or pessimistic) may provide better performance than the other.

Unless schedule classes are *inherently blocking* (i.e., they cannot be implemented without data-access operations blocking; e.g., 2PL, SS2PL and SCO above; see chart), they can be implemented also using optimistic techniques (e.g., Serializability, Recoverability).

Serializable multi-version concurrency control

> See also Multiversion concurrency control (partial coverage)
>
> and Serializable_Snapshot_Isolation in Snapshot isolation

Multi-version concurrency control (MVCC) is a common way today to increase concurrency and performance by generating a new version of a database object each time the object is written, and allowing transactions' read operations of several last relevant versions (of each object), depending on scheduling method. MVCC can be combined with all the serializability techniques listed above (except SerializableSI which is originally MVCC based). It is utilized in most general-purpose DBMS products.

MVCC is especially popular nowadays through the *relaxed serializability* (see above) method *Snapshot isolation* (SI) which provides better performance than most known serializability mechanisms (at the cost of possible serializability violation in certain cases). SerializableSI, which is an efficient enhancement of SI to make it serializable, is intended to provide an efficient serializable solution. SerializableSI has been analyzed[5][6] via a general theory of MVCC

4.1.5 Distributed serializability

Overview

Distributed serializability is the serializability of a schedule of a transactional distributed system (e.g., a distributed database system). Such system is characterized by *distributed transactions* (also called *global transactions*), i.e., transactions that span computer processes (a process abstraction in a general sense, depending on computing environment; e.g., operating system's thread) and possibly network nodes. A distributed transaction comprises more than one *local sub-transactions* that each has states as described above for a database transaction.

A local sub-transaction comprises a single process, or more processes that typically fail together (e.g., in a single processor core). Distributed transactions imply a need in Atomic commit protocol to reach consensus among its local sub-transactions on whether to commit or abort. Such protocols can vary from a simple (one-phase) hand-shake among processes that fail together, to more sophisticated protocols, like Two-phase commit, to handle more complicated cases of failure (e.g., process, node, communication, etc. failure). Distributed serializability is a major goal of distributed concurrency control for correctness. With the proliferation of the Internet, Cloud computing, Grid computing, and small, portable, powerful computing devices (e.g., smartphones) the need for effective distributed serializability techniques to ensure correctness in and among distributed applications seems to increase.

Distributed serializability is achieved by implementing distributed versions of the known centralized techniques.[1][2] Typically all such distributed versions require utilizing conflict information (either of materialized or non-materialized conflicts, or equivalently, transaction precedence or blocking information; conflict serializability is usually utilized) that is not generated locally, but rather in different processes, and remote locations. Thus information distribution is needed (e.g., precedence relations, lock information, timestamps, or tickets). When the distributed system is of a relatively small scale, and message delays across the system are small, the centralized concurrency control methods can be used unchanged, while certain processes or nodes in the system manage the related algorithms. However, in a large-scale system (e.g., *Grid* and *Cloud*), due to the distribution of such information, substantial performance penalty is typically incurred, even when distributed versions of the methods (Vs. centralized) are used, primarily due to computer and communication latency. Also, when such information is distributed, related techniques typically do not scale well. A well-known example with scalability problems is a distributed lock manager, which distributes lock (non-materialized conflict) information across the distributed system to implement locking techniques.

4.1.6 See also

- Strong strict two-phase locking (SS2PL or Rigorousness).
- Making snapshot isolation serializable[5] in Snapshot isolation.
- Global serializability, where the *Global serializability problem* and its proposed solutions are described.
- Linearizability, a more general concept in concurrent computing

4.1.7 Notes

[1] Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman (1987): *Concurrency Control and Recovery in Database Systems* (free PDF download), Addison Wesley Publishing Company, ISBN 0-201-10715-5

[2] Gerhard Weikum, Gottfried Vossen (2001): *Transactional Information Systems*, Elsevier, ISBN 1-55860-508-8

[3] Maurice Herlihy and J. Eliot B. Moss. *Transactional memory: architectural support for lock-free data structures.* Proceedings of the 20th annual international symposium on Computer architecture (ISCA '93). Volume 21, Issue 2, May 1993.

[4] Gray, J.; Helland, P.; O'Neil, P.; Shasha, D. (1996). *The dangers of replication and a solution* (PDF). Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data. pp. 173–182. doi:10.1145/233269.233330.

[5] Michael J. Cahill, Uwe Röhm, Alan D. Fekete (2008): "Serializable isolation for snapshot databases", *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 729-738, Vancouver, Canada, June 2008, ISBN 978-1-60558-102-6 (SIGMOD 2008 best paper award)

[6] Alan Fekete (2009), "Snapshot Isolation and Serializable Execution", Presentation, Page 4, 2009, The university of Sydney (Australia). Retrieved 16 September 2009

4.1.8 References

- Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman (1987): *Concurrency Control and Recovery in Database Systems*, Addison Wesley Publishing Company, ISBN 0-201-10715-5
- Gerhard Weikum, Gottfried Vossen (2001): *Transactional Information Systems*, Elsevier, ISBN 1-55860-508-8

4.2 Isolation (database systems)

In database systems, **isolation** determines how transaction integrity is visible to other users and systems. For example, when a user is creating a Purchase Order and has created the header, but not the Purchase Order lines, is the header available for other systems/users, carrying out concurrent operations (such as a report on Purchase Orders), to see?

A lower isolation level increases the ability of many users to access data at the same time, but increases the number of concurrency effects (such as dirty reads or lost updates) users might encounter. Conversely, a higher isolation level reduces the types of concurrency effects that users may encounter, but requires more system resources

and increases the chances that one transaction will block another.[1]

Isolation is typically defined at database level as a property that defines how/when the changes made by one operation become visible to other. On older systems, it may be implemented systemically, for example through the use of temporary tables. In two-tier systems, a Transaction Processing (TP) manager is required to maintain isolation. In n-tier systems (such as multiple websites attempting to book the last seat on a flight), a combination of stored procedures and transaction management is required to commit the booking and send confirmation to the customer.[2]

Isolation is one of the ACID (Atomicity, Consistency, Isolation, Durability) properties.

4.2.1 Concurrency control

Concurrency control comprises the underlying mechanisms in a DBMS which handles isolation and guarantees related correctness. It is heavily utilized by the database and storage engines (see above) both to guarantee the correct execution of concurrent transactions, and (different mechanisms) the correctness of other DBMS processes. The transaction-related mechanisms typically constrain the database data access operations' timing (transaction schedules) to certain orders characterized as the serializability and recoverability schedule properties. Constraining database access operation execution typically means reduced performance (rates of execution), and thus concurrency control mechanisms are typically designed to provide the best performance possible under the constraints. Often, when possible without harming correctness, the serializability property is compromised for better performance. However, recoverability cannot be compromised, since such typically results in a quick database integrity violation.

Two-phase locking is the most common transaction concurrency control method in DBMSs, used to provide both serializability and recoverability for correctness. In order to access a database object a transaction first needs to acquire a lock for this object. Depending on the access operation type (e.g., reading or writing an object) and on the lock type, acquiring the lock may be blocked and postponed, if another transaction is holding a lock for that object.

4.2.2 Isolation levels

Of the four ACID properties in a DBMS (Database Management System), the isolation property is the one most often relaxed. When attempting to maintain the highest level of isolation, a DBMS usually acquires locks on data or implements multiversion concurrency control, which may result in a loss of concurrency. This requires adding logic for the application to function correctly.

Most DBMSs offer a number of *transaction isolation levels*, which control the degree of locking that occurs when selecting data. For many database applications, the majority of database transactions can be constructed to avoid requiring high isolation levels (e.g. SERIALIZABLE level), thus reducing the locking overhead for the system. The programmer must carefully analyze database access code to ensure that any relaxation of isolation does not cause software bugs that are difficult to find. Conversely, if higher isolation levels are used, the possibility of deadlock is increased, which also requires careful analysis and programming techniques to avoid.

The isolation levels defined by the ANSI/ISO SQL standard are listed as follows.

Serializable

This is the *highest* isolation level.

With a lock-based concurrency control DBMS implementation, serializability requires read and write locks (acquired on selected data) to be released at the end of the transaction. Also *range-locks* must be acquired when a SELECT query uses a ranged *WHERE* clause, especially to avoid the *phantom reads* phenomenon (see below).

When using non-lock based concurrency control, no locks are acquired; however, if the system detects a *write collision* among several concurrent transactions, only one of them is allowed to commit. See *snapshot isolation* for more details on this topic.

Repeatable reads

In this isolation level, a lock-based concurrency control DBMS implementation keeps read and write locks (acquired on selected data) until the end of the transaction. However, *range-locks* are not managed, so ***phantom reads*** can occur.

Read committed

In this isolation level, a lock-based concurrency control DBMS implementation keeps write locks (acquired on selected data) until the end of the transaction, but read locks are released as soon as the SELECT operation is performed (so the *non-repeatable reads* phenomenon can occur in this isolation level, as discussed below). As in the previous level, *range-locks* are not managed.

Putting it in simpler words, read committed is an isolation level that guarantees that any data read is committed at the moment it is read. It simply restricts the reader from seeing any intermediate, uncommitted, 'dirty' read. It makes no promise whatsoever that if the transaction reissues the read, it will find the same data; data is free to

change after it is read.

Read uncommitted

This is the *lowest* isolation level. In this level, ***dirty reads*** are allowed, so one transaction may see *not-yet-committed* changes made by other transactions.

Since each isolation level is stronger than those below, in that no higher isolation level allows an action forbidden by a lower one, the standard permits a DBMS to run a transaction at an isolation level stronger than that requested (e.g., a "Read committed" transaction may actually be performed at a "Repeatable read" isolation level).

4.2.3 Default isolation level

The *default isolation level* of different DBMS's varies quite widely. Most databases that feature transactions allow the user to set any isolation level. Some DBMS's also require additional syntax when performing a SELECT statement to acquire locks (e.g. *SELECT ... FOR UPDATE* to acquire exclusive write locks on accessed rows).

However, the definitions above have been criticized [3] as being ambiguous, and as not accurately reflecting the isolation provided by many databases:

> This paper shows a number of weaknesses in the anomaly approach to defining isolation levels. The three ANSI phenomena are ambiguous. Even their broadest interpretations do not exclude anomalous behavior. This leads to some counter-intuitive results. In particular, lock-based isolation levels have different characteristics than their ANSI equivalents. This is disconcerting because commercial database systems typically use locking. Additionally, the ANSI phenomena do not distinguish among several isolation levels popular in commercial systems.

There are also other criticisms concerning ANSI SQL's isolation definition, in that it encourages implementors to do "bad things":

> ... it relies in subtle ways on an assumption that a locking schema is used for concurrency control, as opposed to an optimistic or multiversion concurrency scheme. This implies that the proposed semantics are *ill-defined*.[4]

4.2.4 Read phenomena

The ANSI/ISO standard SQL 92 refers to three different *read phenomena* when Transaction 1 reads data that Transaction 2 might have changed.

In the following examples, two transactions take place. In the first, Query 1 is performed. Then, in the second transaction, Query 2 is performed and committed. Finally, in the first transaction, Query 1 is performed again.

The queries use the following data table:

Dirty reads

A *dirty read* (aka *uncommitted dependency*) occurs when a transaction is allowed to read data from a row that has been modified by another running transaction and not yet committed.

Dirty reads work similarly to non-repeatable reads; however, the second transaction would not need to be committed for the first query to return a different result. The only thing that may be prevented in the READ UNCOMMITTED isolation level is updates appearing out of order in the results; that is, earlier updates will always appear in a result set before later updates.

In our example, Transaction 2 changes a row, but does not commit the changes. Transaction 1 then reads the uncommitted data. Now if Transaction 2 rolls back its changes (already read by Transaction 1) or updates different changes to the database, then the view of the data may be wrong in the records of Transaction 1.

But in this case no row exists that has an id of 1 and an age of 21.

Non-repeatable reads

A *non-repeatable read* occurs, when during the course of a transaction, a row is retrieved twice and the values within the row differ between reads.

Non-repeatable reads phenomenon may occur in a lock-based concurrency control method when read locks are not acquired when performing a SELECT, or when the acquired locks on affected rows are released as soon as the SELECT operation is performed. Under the multiversion concurrency control method, *non-repeatable reads* may occur when the requirement that a transaction affected by a commit conflict must roll back is relaxed.

In this example, Transaction 2 commits successfully, which means that its changes to the row with id 1 should become visible. However, Transaction 1 has already seen a different value for *age* in that row. At the SERIALIZABLE and REPEATABLE READ isolation levels, the DBMS must return the old value for the second SELECT. At READ COMMITTED and READ UNCOMMITTED, the DBMS may return the updated value; this is a non-repeatable read.

There are two basic strategies used to prevent non-repeatable reads. The first is to delay the execution of Transaction 2 until Transaction 1 has committed or rolled back. This method is used when locking is used, and pro-

duces the serial schedule **T1, T2**. A serial schedule exhibits *repeatable reads* behaviour.

In the other strategy, as used in *multiversion concurrency control*, Transaction 2 is permitted to commit first, which provides for better concurrency. However, Transaction 1, which commenced prior to Transaction 2, must continue to operate on a past version of the database — a snapshot of the moment it was started. When Transaction 1 eventually tries to commit, the DBMS checks if the result of committing Transaction 1 would be equivalent to the schedule **T1, T2**. If it is, then Transaction 1 can proceed. If it cannot be seen to be equivalent, however, Transaction 1 must roll back with a serialization failure.

Using a lock-based concurrency control method, at the REPEATABLE READ isolation mode, the row with ID = 1 would be locked, thus blocking Query 2 until the first transaction was committed or rolled back. In READ COMMITTED mode, the second time Query 1 was executed, the age would have changed.

Under multiversion concurrency control, at the SERIALIZABLE isolation level, both SELECT queries see a snapshot of the database taken at the start of Transaction 1. Therefore, they return the same data. However, if Transaction 1 then attempted to UPDATE that row as well, a serialization failure would occur and Transaction 1 would be forced to roll back.

At the READ COMMITTED isolation level, each query sees a snapshot of the database taken at the start of each query. Therefore, they each see different data for the updated row. No serialization failure is possible in this mode (because no promise of serializability is made), and Transaction 1 will not have to be retried.

Phantom reads

A *phantom read* occurs when, in the course of a transaction, two identical queries are executed, and the collection of rows returned by the second query is different from the first.

This can occur when *range locks* are not acquired on performing a *SELECT ... WHERE* operation. The *phantom reads* anomaly is a special case of *Non-repeatable reads* when Transaction 1 repeats a ranged *SELECT ... WHERE* query and, between both operations, Transaction 2 creates (i.e. INSERT) new rows (in the target table) which fulfill that *WHERE* clause.

Note that Transaction 1 executed the same query twice. If the highest level of isolation were maintained, the same set of rows should be returned both times, and indeed that is what is mandated to occur in a database operating at the SQL SERIALIZABLE isolation level. However, at the lesser isolation levels, a different set of rows may be returned the second time.

In the SERIALIZABLE isolation mode, Query 1 would result in all records with age in the range 10 to 30 being locked, thus Query 2 would block until the first transaction was committed. In REPEATABLE READ mode, the range would not be locked, allowing the record to be inserted and the second execution of Query 1 to include the new row in its results.

4.2.5 Isolation Levels, Read Phenomena and Locks

Isolation Levels vs Read Phenomena

Anomaly Serializable is not the same as Serializable. That is, it is necessary, but not sufficient that a Serializable schedule should be free of all three phenomena types. See [1] below.

“may occur” means that the isolation level suffers that phenomenon, while "-" means that it does not suffer it.

Isolation Levels vs Lock Duration

In lock-based concurrency control, isolation level determines the duration that locks are held.
“C” - Denotes that locks are held until the transaction commits.
“S” - Denotes that the locks are held only during the currently executing statement. Note that if locks are released after a statement, the underlying data could be changed by another transaction before the current transaction commits, thus creating a violation.

4.2.6 See also

- Atomicity
- Consistency
- Durability
- Lock (database)
- Optimistic concurrency control
- Relational Database Management System
- Snapshot isolation

4.2.7 References

[1] “Isolation Levels in the Database Engine”, Technet, Microsoft, http://technet.microsoft.com/en-us/library/ms189122(v=SQL.105).aspx

[2] “The Architecture of Transaction Processing Systems”, Chapter 23, Evolution of Processing Systems, Department of Computer Science, Stony Brook University, retrieved 20 March 2014, http://www.cs.sunysb.edu/~{}liu/cse315/23.pdf

[3] "A Critique of ANSI SQL Isolation Levels" (PDF). Retrieved 29 July 2012.

[4] salesforce (2010-12-06). "Customer testimonials (SimpleGeo, CLOUDSTOCK 2010)". www.DataStax.com: DataStax. Retrieved 2010-03-09. (see above at about 13:30 minutes of the webcast!)

4.2.8 External links

- Oracle® Database Concepts, chapter 13 Data Concurrency and Consistency, Preventable Phenomena and Transaction Isolation Levels
- Oracle® Database SQL Reference, chapter 19 SQL Statements: SAVEPOINT to UPDATE, SET TRANSACTION
- in JDBC: Connection constant fields, Connection.getTransactionIsolation(), Connection.setTransactionIsolation(int)
- in Spring Framework: @Transactional, Isolation
- P.Bailis. When is "ACID" ACID? Rarely

4.3 Database transaction

A **transaction** symbolizes a unit of work performed within a database management system (or similar system) against a database, and treated in a coherent and reliable way independent of other transactions. A transaction generally represents any change in database. Transactions in a database environment have two main purposes:

1. To provide reliable units of work that allow correct recovery from failures and keep a database consistent even in cases of system failure, when execution stops (completely or partially) and many operations upon a database remain uncompleted, with unclear status.
2. To provide isolation between programs accessing a database concurrently. If this isolation is not provided, the programs' outcomes are possibly erroneous.

A database transaction, by definition, must be atomic, consistent, isolated and durable.[1] Database practitioners often refer to these properties of database transactions using the acronym ACID.

Transactions provide an "all-or-nothing" proposition, stating that each work-unit performed in a database must either complete in its entirety or have no effect whatsoever. Further, the system must isolate each transaction from other transactions, results must conform to existing constraints in the database, and transactions that complete successfully must get written to durable storage.

4.3.1 Purpose

Databases and other data stores which treat the integrity of data as paramount often include the ability to handle transactions to maintain the integrity of data. A single transaction consists of one or more independent units of work, each reading and/or writing information to a database or other data store. When this happens it is often important to ensure that all such processing leaves the database or data store in a consistent state.

Examples from double-entry accounting systems often illustrate the concept of transactions. In double-entry accounting every debit requires the recording of an associated credit. If one writes a check for $100 to buy groceries, a transactional double-entry accounting system must record the following two entries to cover the single transaction:

1. Debit $100 to Groceries Expense Account
2. Credit $100 to Checking Account

A transactional system would make both entries pass or both entries would fail. By treating the recording of multiple entries as an atomic transactional unit of work the system maintains the integrity of the data recorded. In other words, nobody ends up with a situation in which a debit is recorded but no associated credit is recorded, or vice versa.

4.3.2 Transactional databases

A **transactional database** is a DBMS where write transactions on the database are able to be rolled back if they are not completed properly (e.g. due to power or connectivity loss).

Most modern relational database management systems fall into the category of databases that support transactions.

In a database system a transaction might consist of one or more data-manipulation statements and queries, each reading and/or writing information in the database. Users of database systems consider consistency and integrity of data as highly important. A simple transaction is usually issued to the database system in a language like SQL wrapped in a transaction, using a pattern similar to the following:

1. Begin the transaction
2. Execute a set of data manipulations and/or queries
3. If no errors occur then commit the transaction and end it
4. If errors occur then rollback the transaction and end it

If no errors occurred during the execution of the transaction then the system commits the transaction. A transaction commit operation applies all data manipulations within the scope of the transaction and persists the results to the database. If an error occurs during the transaction, or if the user specifies a rollback operation, the data manipulations within the transaction are not persisted to the database. In no case can a partial transaction be committed to the database since that would leave the database in an inconsistent state.

Internally, multi-user databases store and process transactions, often by using a transaction ID or XID.

There are multiple varying ways for transactions to be implemented other than the simple way documented above. Nested transactions, for example, are transactions which contain statements within them that start new transactions (i.e. sub-transactions). *Multi-level transactions* are a variant of nested transactions where the sub-transactions take place at different levels of a layered system architecture (e.g., with one operation at the database-engine level, one operation at the operating-system level) [2] Another type of transaction is the compensating transaction.

In SQL

Transactions are available in most SQL database implementations, though with varying levels of robustness. (MySQL, for example, does not support transactions in the MyISAM storage engine, which was its default storage engine before version 5.5.)

A transaction is typically started using the command BEGIN (although the SQL standard specifies START TRANSACTION). When the system processes a COMMIT statement, the transaction ends with successful completion. A ROLLBACK statement can also end the transaction, undoing any work performed since BEGIN TRANSACTION. If autocommit was disabled using START TRANSACTION, autocommit will also be re-enabled at the transaction's end.

One can set the isolation level for individual transactional operations as well as globally. At the READ COMMITTED level, the result of any work done after a transaction has commenced, but before it has ended, will remain invisible to other database-users until it has ended. At the lowest level (READ UNCOMMITTED), which may occasionally be used to ensure high concurrency, such changes will be visible.

4.3.3 Object databases

Relational databases traditionally comprise tables with fixed size fields and thus records. Object databases comprise variable sized blobs (possibly incorporating a mimetype or serialized). The fundamental similarity though is the start and the commit or rollback.

After starting a transaction, database records or objects are locked, either read-only or read-write. Actual reads and writes can then occur. Once the user (and application) is happy, any changes are committed or rolled-back atomically, such that at the end of the transaction there is no inconsistency.

4.3.4 Distributed transactions

Database systems implement distributed transactions as transactions against multiple applications or hosts. A distributed transaction enforces the ACID properties over multiple systems or data stores, and might include systems such as databases, file systems, messaging systems, and other applications. In a distributed transaction a coordinating service ensures that all parts of the transaction are applied to all relevant systems. As with database and other transactions, if any part of the transaction fails, the entire transaction is rolled back across all affected systems.

4.3.5 Transactional filesystems

The Namesys Reiser4 filesystem for Linux[3] supports transactions, and as of Microsoft Windows Vista, the Microsoft NTFS filesystem[4] supports distributed transactions across networks.

4.3.6 See also

- Concurrency control

4.3.7 References

[1] A transaction is a group of operations that are atomic, consistent, isolated, and durable (ACID).

[2] Beeri, C., Bernstein, P.A., and Goodman, N. A model for concurrency in nested transactions systems. Journal of the ACM, 36(1):230-269, 1989

[3] namesys.com

[4] "MSDN Library". Retrieved 16 October 2014.

4.3.8 Further reading

- Philip A. Bernstein, Eric Newcomer (2009): *Principles of Transaction Processing*, 2nd Edition, Morgan Kaufmann (Elsevier), ISBN 978-1-55860-623-4

- Gerhard Weikum, Gottfried Vossen (2001), *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*, Morgan Kaufmann, ISBN 1-55860-508-8

4.3.9 External links

- c2:TransactionProcessing

4.4 Transaction processing

For other uses, see Transaction (disambiguation).
This article is about the principles of transaction processing. For specific implementations, see Transaction processing system.

In computer science, **transaction processing** is information processing that is divided into individual, indivisible operations called *transactions*. Each transaction must succeed or fail as a complete unit; it can never be only partially complete.

For example, when you purchase a book from an online bookstore, you exchange money (in the form of credit) for a book. If your credit is good, a series of related operations ensures that you get the book and the bookstore gets your money. However, if a single operation in the series fails during the exchange, the entire exchange fails. You do not get the book and the bookstore does not get your money. The technology responsible for making the exchange balanced and predictable is called transaction processing. Transactions ensure that data-oriented resources are not permanently updated unless all operations within the transactional unit complete successfully. By combining a set of related operations into a unit that either completely succeeds or completely fails, one can simplify error recovery and make one's application more reliable.

Transaction processing systems consist of computer hardware and software hosting a transaction-oriented application that performs the routine transactions necessary to conduct business. Examples include systems that manage sales order entry, airline reservations, payroll, employee records, manufacturing, and shipping.

Since most, though not necessarily all, transaction processing today is interactive the term is often treated as synonymous with *online transaction processing*.

4.4.1 Description

Transaction processing is designed to maintain a system's Integrity (typically a database or some modern filesystems) in a known, consistent state, by ensuring that interdependent operations on the system are either all completed successfully or all canceled successfully.

For example, consider a typical banking transaction that involves moving $700 from a customer's savings account to a customer's checking account. This transaction involves at least two separate operations in computer terms: debiting the savings account by $700, and crediting the checking account by $700. If one operation succeeds but the other does not, the books of the bank will not balance at the end of the day. There must therefore be a way to ensure that either both operations succeed or both fail, so that there is never any inconsistency in the bank's database as a whole.

Transaction processing links multiple individual operations in a single, indivisible transaction, and ensures that either all operations in a transaction are completed without error, or none of them are. If some of the operations are completed but errors occur when the others are attempted, the transaction-processing system "rolls back" *all* of the operations of the transaction (including the successful ones), thereby erasing all traces of the transaction and restoring the system to the consistent, known state that it was in before processing of the transaction began. If all operations of a transaction are completed successfully, the transaction is committed by the system, and all changes to the database are made permanent; the transaction cannot be rolled back once this is done.

Transaction processing guards against hardware and software errors that might leave a transaction partially completed. If the computer system crashes in the middle of a transaction, the transaction processing system guarantees that all operations in any uncommitted transactions are cancelled.

Generally, transactions are issued concurrently. If they overlap (i.e. need to touch the same portion of the database), this can create conflicts. For example, if the customer mentioned in the example above has $150 in his savings account and attempts to transfer $100 to a different person while at the same time moving $100 to the checking account, only one of them can succeed. However, forcing transactions to be processed sequentially is inefficient. Therefore, concurrent implementations of transaction processing is programmed to guarantee that the end result reflects a conflict-free outcome, the same as could be reached if executing the transactions sequentially in any order (a property called serializability). In our example, this means that no matter which transaction was issued first, either the transfer to a different person or the move to the checking account succeeds, while the other one fails.

4.4.2 Methodology

The basic principles of all transaction-processing systems are the same. However, the terminology may vary from one transaction-processing system to another, and the terms used below are not necessarily universal.

Rollback

Main article: Rollback (data management)

Transaction-processing systems ensure database integrity by recording intermediate states of the database as it is modified, then using these records to restore the database to a known state if a transaction cannot be committed. For example, copies of information on the database *prior* to its modification by a transaction are set aside by the system before the transaction can make any modifications (this is sometimes called a *before image*). If any part of the transaction fails before it is committed, these copies are used to restore the database to the state it was in before the transaction began.

Rollforward

It is also possible to keep a separate journal of all modifications to a database management system. (sometimes called *after images*). This is not required for rollback of failed transactions but it is useful for updating the database management system in the event of a database failure, so some transaction-processing systems provide it. If the database management system fails entirely, it must be restored from the most recent back-up. The back-up will not reflect transactions committed since the back-up was made. However, once the database management system is restored, the journal of after images can be applied to the database (*rollforward*) to bring the database management system up to date. Any transactions in progress at the time of the failure can then be rolled back. The result is a database in a consistent, known state that includes the results of all transactions committed up to the moment of failure.

Deadlocks

Main article: Deadlock

In some cases, two transactions may, in the course of their processing, attempt to access the same portion of a database at the same time, in a way that prevents them from proceeding. For example, transaction A may access portion X of the database, and transaction B may access portion Y of the database. If, at that point, transaction A then tries to access portion Y of the database while transaction B tries to access portion X, a *deadlock* occurs, and neither transaction can move forward. Transaction-processing systems are designed to detect these deadlocks when they occur. Typically both transactions will be cancelled and rolled back, and then they will be started again in a different order, automatically, so that the deadlock doesn't occur again. Or sometimes, just one of the deadlocked transactions will be cancelled, rolled back, and automatically restarted after a short delay.

Deadlocks can also occur among three or more transactions. The more transactions involved, the more difficult they are to detect, to the point that transaction processing systems find there is a practical limit to the deadlocks they can detect.

Compensating transaction

In systems where commit and rollback mechanisms are not available or undesirable, a compensating transaction is often used to undo failed transactions and restore the system to a previous state.

4.4.3 ACID criteria

Main article: ACID

Jim Gray defined properties of a reliable transaction system in the late 1970s under the acronym *ACID* — atomicity, consistency, isolation, and durability.[1]

Atomicity

Main article: Atomicity (database systems)

A transaction's changes to the state are atomic: either all happen or none happen. These changes include database changes, messages, and actions on transducers.

Consistency

Consistency: A transaction is a correct transformation of the state. The actions taken as a group do not violate any of the integrity constraints associated with the state.

Isolation

Even though transactions execute concurrently, it appears to each transaction T, that others executed either before T or after T, but not both.

Durability

Once a transaction completes successfully (commits), its changes to the state survive failures.

4.4.4 Benefits

Transaction processing has these benefits:

- It allows sharing of computer resources among many users
- It shifts the time of job processing to when the computing resources are less busy

- It avoids idling the computing resources without minute-by-minute human interaction and supervision
- It is used on expensive classes of computers to help amortize the cost by keeping high rates of utilization of those expensive resources

4.4.5 Implementations

Main article: Transaction processing system

Standard transaction-processing software, notably IBM's Information Management System, was first developed in the 1960s, and was often closely coupled to particular database management systems. Client–server computing implemented similar principles in the 1980s with mixed success. However, in more recent years, the distributed client–server model has become considerably more difficult to maintain. As the number of transactions grew in response to various online services (especially the Web), a single distributed database was not a practical solution. In addition, most online systems consist of a whole suite of programs operating together, as opposed to a strict client–server model where the single server could handle the transaction processing. Today a number of transaction processing systems are available that work at the inter-program level and which scale to large systems, including mainframes.

One well-known (and open) industry standard is the X/Open Distributed Transaction Processing (DTP) (see also JTA the Java Transaction API). However, proprietary transaction-processing environments such as IBM's CICS are still very popular, although CICS has evolved to include open industry standards as well.

The term 'Extreme Transaction Processing' (XTP) has been used to describe transaction processing systems with uncommonly challenging requirements, particularly throughput requirements (transactions per second). Such systems may be implemented via distributed or cluster style architectures.

4.4.6 References

[1] Gray, Jim; Reuter, Andreas. "Transaction Processing - Concepts and Techniques (Powerpoint)". Retrieved Nov 12, 2012.

4.4.7 External links

- Nuts and Bolts of Transaction Processing (1999)
- Managing Transaction Processing for SQL Database Integrity
- Transaction Processing

4.4.8 Further reading

- Gerhard Weikum, Gottfried Vossen, *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*, Morgan Kaufmann, 2002, ISBN 1-55860-508-8
- Jim Gray, Andreas Reuter, Transaction Processing — Concepts and Techniques, 1993, Morgan Kaufmann, ISBN 1-55860-190-2
- Philip A. Bernstein, Eric Newcomer, Principles of Transaction Processing, 1997, Morgan Kaufmann, ISBN 1-55860-415-4
- Ahmed K. Elmagarmid (Editor), Transaction Models for Advanced Database Applications, Morgan-Kaufmann, 1992, ISBN 1-55860-214-3

Chapter 5

Atomicity

5.1 Journaling file system

For the IBM Journaled File System, see JFS (file system).

A **journaling file system** is a file system that keeps track of changes not yet committed to the file system's main part by recording the intentions of such changes in a data structure known as a "journal", which is usually a circular log. In the event of a system crash or power failure, such file systems can be brought back online quicker with lower likelihood of becoming corrupted.[1][2]

Depending on the actual implementation, a journaling file system may only keep track of stored metadata, resulting in improved performance at the expense of increased possibility for data corruption. Alternatively, a journaling file system may track both stored data and related metadata, while some implementations allow selectable behavior in this regard.[3]

5.1.1 Rationale

Updating file systems to reflect changes to files and directories usually requires many separate write operations. This makes it possible for an interruption (like a power failure or system crash) between writes to leave data structures in an invalid intermediate state.[1]

For example, deleting a file on a Unix file system involves three steps:[4]

1. Removing its directory entry.
2. Release the inode to the pool of free inodes.
3. Return all used disk blocks to the pool of free disk blocks.

If a crash occurs after step 1 and before step 2, there will be an orphaned inode and hence a storage leak. On the other hand, if only step 2 is performed first before the crash, the not-yet-deleted file will be marked free and possibly be overwritten by something else.

Detecting and recovering from such inconsistencies normally requires a complete walk of its data structures, for example by a tool such as fsck (the file system checker).[2] This must typically be done before the file system is next mounted for read-write access. If the file system is large and if there is relatively little I/O bandwidth, this can take a long time and result in longer downtimes if it blocks the rest of the system from coming back online.

To prevent this, a journaled file system allocates a special area—the journal—in which it records the changes it will make ahead of time. After a crash, recovery simply involves reading the journal from the file system and replaying changes from this journal until the file system is consistent again. The changes are thus said to be atomic (not divisible) in that they either succeed (succeeded originally or are replayed completely during recovery), or are not replayed at all (are skipped because they had not yet been completely written to the journal before the crash occurred).

5.1.2 Techniques

Some file systems allow the journal to grow, shrink and be re-allocated just as a regular file, while others put the journal in a contiguous area or a hidden file that is guaranteed not to move or change size while the file system is mounted. Some file systems may also allow *external journals* on a separate device, such as a solid-state drive or battery-backed non-volatile RAM. Changes to the journal may themselves be journaled for additional redundancy, or the journal may be distributed across multiple physical volumes to protect against device failure.

The internal format of the journal must guard against crashes while the journal itself is being written to. Many journal implementations (such as the JBD2 layer in ext4) bracket every change logged with a checksum, on the understanding that a crash would leave a partially written change with a missing (or mismatched) checksum that can simply be ignored when replaying the journal at next remount.

Physical journals

A *physical journal* logs an advance copy of every block that will later be written to the main file system. If there is

a crash when the main file system is being written to, the write can simply be replayed to completion when the file system is next mounted. If there is a crash when the write is being logged to the journal, the partial write will have a missing or mismatched checksum and can be ignored at next mount.

Physical journals impose a significant performance penalty because every changed block must be committed *twice* to storage, but may be acceptable when absolute fault protection is required.[5]

Logical journals

A *logical journal* stores only changes to file metadata in the journal, and trades fault tolerance for substantially better write performance.[6] A file system with a logical journal still recovers quickly after a crash, but may allow unjournaled file data and journaled metadata to fall out of sync with each other, causing data corruption.

For example, appending to a file may involve three separate writes to:

1. The file's inode, to note in the file's metadata that its size has increased.
2. The free space map, to mark out an allocation of space for the to-be-appended data.
3. The newly allocated space, to actually write the appended data.

In a metadata-only journal, step 3 would not be logged. If step 3 was not done, but steps 1 and 2 are replayed during recovery, the file will be appended with garbage.

Write hazards

The write cache in most operating systems sorts its writes (using the elevator algorithm or some similar scheme) to maximize throughput. To avoid an out-of-order write hazard with a metadata-only journal, writes for file data must be sorted so that they are committed to storage before their associated metadata. This can be tricky to implement because it requires coordination within the operating system kernel between the file system driver and write cache. An out-of-order write hazard can also exist if the underlying storage cannot write blocks atomically, or does not honor requests to flush its write cache.

To complicate matters, many mass storage devices have their own write caches, in which they may aggressively reorder writes for better performance. (This is particularly common on magnetic hard drives, which have large seek latencies that can be minimized with elevator sorting.) Some journaling file systems conservatively assume such write-reordering always takes place, and sacrifice performance for correctness by forcing the device to flush its cache at certain points in the journal (called barriers in ext3 and ext4).[7]

5.1.3 Alternatives

Soft updates

Some UFS implementations avoid journaling and instead implement soft updates: they order their writes in such a way that the on-disk file system is never inconsistent, or that the only inconsistency that can be created in the event of a crash is a storage leak. To recover from these leaks, the free space map is reconciled against a full walk of the file system at next mount. This garbage collection is usually done in the background.[8]

Log-structured file systems

In log-structured file systems, the write-twice penalty does not apply because the journal itself *is* the file system: it occupies the entire storage device and is structured so that it can be traversed as would a normal file system.

Copy-on-write file systems

Full copy-on-write file systems (such as ZFS and Btrfs) avoid in-place changes to file data by writing out the data in newly allocated blocks, followed by updated metadata that would point to the new data and disown the old, followed by metadata pointing to that, and so on up to the superblock, or the root of the file system hierarchy. This has the same correctness-preserving properties as a journal, without the write-twice overhead.

5.1.4 See also

- ACID
- Comparison of file systems
- Database
- Intent log
- Journaled File System (JFS) – a file system made by IBM
- Transaction processing

5.1.5 References

[1] Jones, M Tim (2008-06-04), *Anatomy of Linux journaling file systems*, IBM DeveloperWorks, retrieved 2009-04-13

[2] Arpaci-Dusseau, Remzi H.; Arpaci-Dusseau, Andrea C. (2014-01-21), *Crash Consistency: FSCK and Journaling* (PDF), Arpaci-Dusseau Books

[3] "tune2fs(8) – Linux man page". *linux.die.net*. Retrieved February 20, 2015.

[4] File Systems from Tanenbaum, A.S. (2008). Modern operating systems (3rd ed., pp. 287). Upper Saddle River, NJ: Prentice Hall.

[5] Tweedie, Stephen (2000), "Ext3, journaling filesystem", *Proceedings of the Ottawa Linux Symposium*: 24–29

[6] Prabhakaran, Vijayan; Arpaci-Dusseau, Andrea C; Arpaci-Dusseau, Remzi H, "Analysis and Evolution of Journaling File Systems" (PDF), *2005 USENIX Annual Technical Conference* (USENIX Association).

[7] Corbet, Jonathan (2008-05-21), *Barriers and journaling filesystems*, retrieved 2010-03-06

[8] Seltzer, Margo I; Ganger, Gregory R; McKusick, M Kirk, "Journaling Versus Soft Updates: Asynchronous Metadata Protection in File Systems", *2000 USENIX Annual Technical Conference* (USENIX Association).

5.2 Atomicity (database systems)

For other uses, see Atomicity (disambiguation).

In database systems, **atomicity** (or **atomicness**; from Greek *a-tomos*, *undividable*) is one of the ACID transaction properties. In an **atomic transaction**, a series of database operations either *all* occur, or *nothing* occurs. The series of operations cannot be divided apart and executed partially from each other, which makes the series of operations "indivisible", hence the name. A guarantee of atomicity prevents updates to the database occurring only partially, which can cause greater problems than rejecting the whole series outright. In other words, atomicity means *indivisibility* and *irreducibility*.[1] As a consequence, the transaction cannot be observed to be in progress by another database client. At one moment in time, it has not yet happened, and at the next it has already occurred in whole (or nothing happened if the transaction was cancelled in progress).

The etymology of the phrase originates in the Classical Greek concept of a fundamental and indivisible component; see atom.

5.2.1 Examples

An example of atomicity is ordering an airline ticket where two actions are required: payment, and a seat reservation. The potential passenger must either:

1. both pay for and reserve a seat; OR
2. neither pay for nor reserve a seat.

The booking system does not consider it acceptable for a customer to pay for a ticket without securing the seat, nor to reserve the seat without payment succeeding.

Another example is that if one wants to transfer some amount of money from one account to another, then the user would start a procedure to do it. However, if a failure occurs, then due to atomicity, the amount will either be transferred completely or will not be even initiated. Thus atomicity protects the user from losing money due to a failed transaction.

5.2.2 Orthogonality

Atomicity does not behave completely orthogonally with regard to the other ACID properties of the transactions. For example, isolation relies on atomicity to roll back changes in the event of isolation failures such as deadlock; consistency also relies on rollback in the event of a consistency-violation by an illegal transaction. Finally, atomicity itself relies on durability to ensure the atomicity of transactions even in the face of external failures.

As a result of this, failure to detect errors and roll back the enclosing transaction may cause failures of isolation and consistency.

5.2.3 Implementation

Typically, systems implement Atomicity by providing some mechanism to indicate which transactions have started and which finished; or by keeping a copy of the data before any changes occurred (read-copy-update). Several filesystems have developed methods for avoiding the need to keep multiple copies of data, using journaling (see journaling file system). Databases usually implement this using some form of logging/journaling to track changes. The system synchronizes the logs (often the metadata) as necessary once the actual changes have successfully taken place. Afterwards, crash recovery simply ignores incomplete entries. Although implementations vary depending on factors such as concurrency issues, the principle of atomicity — i.e. complete success or complete failure — remain.

Ultimately, any application-level implementation relies on operating-system functionality. At the file-system level, POSIX-compliant systems provide system calls such as open(2) and flock(2) that allow applications to atomically open or lock a file. At the process level, POSIX Threads provide adequate synchronization primitives.

The hardware level requires atomic operations such as Test-and-set, Fetch-and-add, Compare-and-swap, or Load-Link/Store-Conditional, together with memory barriers. Portable operating systems cannot simply block interrupts to implement synchronization, since hardware

that lacks actual concurrent execution such as hyper-threading or multi-processing is now extremely rare.

In NoSQL data stores with eventual consistency, the atomicity is also weaker specified than in relational database systems, and exists only in *row*s (i.e. column families).[2]

5.2.4 See also

- Atomic operation
- Transaction processing
- Long-running transaction
- Read-copy-update

5.2.5 References

[1] "atomic operation". http://www.webopedia.com/: Webopedia. Retrieved 2011-03-23. An operation during which a processor can simultaneously read a location and write it in the same bus operation. This prevents any other processor or I/O device from writing or reading memory until the operation is complete.

[2] Olivier Mallassi (2010-06-09). "Let's play with Cassandra… (Part 1/3)". http://blog.octo.com/en/: OCTO Talks!. Retrieved 2011-03-23. Atomicity is also weaker than what we are used to in the relational world. Cassandra guarantees atomicity within a ColumnFamily so for all the columns of a row.

Chapter 6

Locking

6.1 Lock (database)

A **lock**, as a **read lock** or **write lock**, is used when multiple users need to access a database concurrently. This prevents data from being corrupted or invalidated when multiple users try to read while others write to the database. Any single user can only modify those database records (that is, items in the database) to which they have applied a lock that gives them exclusive access to the record until the lock is released. Locking not only provides exclusivity to writes but also prevents (or controls) reading of unfinished modifications (AKA uncommitted data).

A read lock can be used to prevent other users from reading a record (or page) which is being updated, so that others will not act upon soon-to-be-outdated information.

6.1.1 Mechanisms for locking

There are two mechanisms for locking data in a database: *pessimistic locking*, and *optimistic locking*. In pessimistic locking a record or page is locked immediately when the lock is requested, while in an optimistic lock the record or page is only locked when the changes made to that record are updated. The latter situation is only appropriate when there is less chance of someone needing to access the record while it is locked; otherwise it cannot be certain that the update will succeed because the attempt to update the record will fail if another user updates the record first. With pessimistic locking it is guaranteed that the record will be updated.

The degree of locking can be controlled by isolation level. Change of a lock is called lock conversion and the lock may be upgraded (lock upgrade) or downgraded (lock downgrade).

6.1.2 See also

- Race condition

6.2 Record locking

Record locking is the technique of preventing simultaneous access to data in a database, to prevent inconsistent results.

The classic example is demonstrated by two bank clerks attempting to update the same bank account for two different transactions. Clerks 1 and 2 both retrieve (i.e., copy) the account's record. Clerk 1 applies and saves a transaction. Clerk 2 applies a different transaction to his saved copy, and saves the result, based on the original record and his changes, overwriting the transaction entered by clerk 1. The record no longer reflects the first transaction, as if it had never taken place.

A simple way to prevent this is to lock the file whenever a record is being modified by any user, so that no other user can save data. This prevents records from being overwritten incorrectly, but allows only one record to be processed at a time, locking out other users who need to edit records at the same time.

To allow several users to edit a database table at the same time and also prevent inconsistencies created by unrestricted access, a single record can be *locked* when retrieved for editing or updating. Anyone attempting to retrieve the same record for editing is denied write access because of the lock (although, depending on the implementation, they may be able to view the record without editing it). Once the record is saved or edits are canceled, the lock is released. Records can never be saved so as to overwrite other changes, preserving data integrity.

In database management theory, locking is used to implement *isolation* among multiple database users. This is the "I" in the acronym ACID.

A thorough and authoritative description of locking was written by Jim Gray.[1]

6.2.1 Granularity of locks

If the bank clerks (to follow the illustration above) are serving two customers, but their accounts are contained in one ledger, then the entire ledger, or one or more database tables, would need to be made available for editing to the

clerks in order for each to complete a transaction, one at a time (file locking). While safe, this method can cause unnecessary waiting.

If the clerks can remove one page from the ledger, containing the account of the current customer (plus several other accounts), then multiple customers can be serviced concurrently, provided that each customer's account is found on a different page than the others. If two customers have accounts on the same page, then only one may be serviced at a time. This is analogous to a *page level lock* in a database.

A higher degree of granularity is achieved if each individual account may be taken by a clerk. This would allow any customer to be serviced without waiting for another customer who is accessing a different account. This is analogous to a *record level lock* and is normally the highest degree of locking granularity in a database management system.

In a SQL database, a record is typically called a "row."

The introduction of granular (subset) locks creates the possibility for a situation called deadlock. Deadlock is possible when *incremental locking* (locking one entity, then locking one or more additional entities) is used. To illustrate, if two bank customers asked two clerks to obtain their account information so they could transfer some money into other accounts, the two accounts would essentially be locked. Then, if the customers told their clerks that the money was to be transferred into each other's accounts, the clerks would search for the other accounts but find them to be "in use" and wait for them to be returned. Unknowingly, the two clerks are waiting for each other, and neither of them can complete their transaction until the other gives up and returns the account. Various techniques are used to avoid such problems.

6.2.2 Use of locks

Record locks need to be managed between the entities requesting the records such that no entity is given too much service via successive **grants**, and no other entity is effectively locked out. The entities that request a lock can be either individual applications (programs) or an entire processor.

The application or system should be designed such that any lock is held for the shortest time possible. Data reading, without editing facilities, does not require a lock, and reading locked records is usually permissible.

Two main types of locks can be requested:

Exclusive locks

Exclusive locks are, as the name implies, exclusively held by a single entity, usually for the purpose of writing to the record. If the locking schema was represented by a list, the **holder list** would contain only one entry. Since this type of lock effectively blocks any other entity that requires the lock from processing, care must be used to:

- ensure the lock is held for the shortest time possible;
- not hold the lock across system or function calls where the entity is no longer running on the processor - this can lead to deadlock;
- ensure that if the entity is unexpectedly exited for any reason, the lock is freed.

Non-holders of the lock (aka **waiters**) can be held in a list that is serviced in a round robin fashion, or in a FIFO queue. This would ensure that any possible waiter would get equal chance to obtain the lock and not be locked out. To further speed up the process, if an entity has gone to sleep waiting for a lock, performance is improved if the entity is notified of the grant, instead of discovering it on some sort of system timeout driven wakeup.

Shared locks

Shared locks differ from exclusive locks in that the **holder list** can contain multiple entries. Shared locks allow all holders to read the contents of the record knowing that the record cannot be changed until after the lock has been released by all holders. Exclusive locks cannot be obtained when a record is already locked (exclusively or shared) by another entity.

If lock requests for the same entity are queued, then once a shared lock is granted, any queued shared locks may also be granted. If an exclusive lock is found next on the queue, it must wait until all shared locks have been released. As with exclusive locks, these shared locks should be held for the least time possible.

6.2.3 References

[1] Gray, Jim, and Reuter, Andreas (1993), *Distributed Transaction Processing: Concepts and Techniques*, Morgan Kaufmann, pp. 375–437, ISBN 1-55860-190-2

6.3 Two-phase locking

This article is about concurrency control. For commit consensus within a distributed transaction, see Two-phase commit protocol.

In databases and transaction processing, **two-phase locking** (**2PL**) is a concurrency control method that guarantees serializability.[1][2] It is also the name of the resulting set of database transaction schedules (histories). The protocol utilizes locks, applied by a transaction to data, which

may block (interpreted as signals to stop) other transactions from accessing the same data during the transaction's life.

By the 2PL protocol locks are applied and removed in two phases:

1. Expanding phase: locks are acquired and no locks are released.
2. Shrinking phase: locks are released and no locks are acquired.

Two types of locks are utilized by the basic protocol: *Shared* and *Exclusive* locks. Refinements of the basic protocol may utilize more lock types. Using locks that block processes, 2PL may be subject to deadlocks that result from the mutual blocking of two or more transactions.

6.3.1 Data-access locks

A lock is a system object associated with a shared resource such as a data item of an elementary type, a row in a database, or a page of memory. In a database, a lock on a database object (a data-access lock) may need to be acquired by a transaction before accessing the object. Correct use of locks prevents undesired, incorrect or inconsistent operations on shared resources by other concurrent transactions. When a database object with an existing lock acquired by one transaction needs to be accessed by another transaction, the existing lock for the object and the type of the intended access are checked by the system. If the existing lock type does not allow this specific attempted concurrent access type, the transaction attempting access is blocked (according to a predefined agreement/scheme). In practice a lock on an object does not directly block a transaction's operation upon the object, but rather blocks that transaction from acquiring another lock on the same object, needed to be held/owned by the transaction before performing this operation. Thus, with a locking mechanism, needed operation blocking is controlled by a proper lock blocking scheme, which indicates which lock type blocks which lock type.

Two major types of locks are utilized:

- **Write-lock** (**exclusive lock**) is associated with a database object by a transaction (Terminology: "the transaction locks the object," or "acquires lock for it") before *writing* (inserting/modifying/deleting) this object.
- **Read-lock** (**shared lock**) is associated with a database object by a transaction before *reading* (retrieving the state of) this object.

The common interactions between these lock types are defined by blocking behavior as follows:

- An existing *write-lock* on a database object blocks an intended *write* upon the same object (already requested/issued) by another transaction by blocking a respective *write-lock* from being acquired by the other transaction. The second write-lock will be acquired and the requested write of the object will take place (materialize) after the existing write-lock is released.
- A *write-lock* blocks an intended (already requested/issued) *read* by another transaction by blocking the respective *read-lock* .
- A *read-lock* blocks an intended *write* by another transaction by blocking the respective *write-lock* .
- A *read-lock* does not block an intended *read* by another transaction. The respective *read-lock* for the intended read is acquired (shared with the previous read) immediately after the intended read is requested, and then the intended read itself takes place.

Several variations and refinements of these major lock types exist, with respective variations of blocking behavior. If a first lock blocks another lock, the two locks are called *incompatible*; otherwise the locks are *compatible*. Often lock types blocking interactions are presented in the technical literature by a *Lock compatibility table*. The following is an example with the common, major lock types:

> **X** indicates incompatibility, i.e, a case when a lock of the first type (in left column) on an object blocks a lock of the second type (in top row) from being acquired on the same object (by another transaction). An object typically has a queue of waiting requested (by transactions) operations with respective locks. The first blocked lock for operation in the queue is acquired as soon as the existing blocking lock is removed from the object, and then its respective operation is executed. If a lock for operation in the queue is not blocked by any existing lock (existence of multiple compatible locks on a same object is possible concurrently) it is acquired immediately.
>
> **Comment:** In some publications the table entries are simply marked "compatible" or "incompatible", or respectively "yes" or "no".

6.3.2 Two-phase locking and its special cases

Two-phase locking

According to the **two-phase locking** protocol a transaction handles its locks in two distinct, consecutive phases during the transaction's execution:

1. **Expanding phase** (aka Growing phase): locks are acquired and no locks are released (the number of locks can only increase).

2. **Shrinking phase**: locks are released and no locks are acquired.

The two phase locking rule can be summarized as: never acquire a lock after a lock has been released. The serializability property is guaranteed for a schedule with transactions that obey this rule.

Typically, without explicit knowledge in a transaction on end of phase-1, it is safely determined only when a transaction has completed processing and requested commit. In this case all the locks can be released at once (phase-2).

Strict two-phase locking

To comply with the S2PL protocol a transaction needs to comply with 2PL, and release its *write (exclusive)* locks only after it has ended, i.e., being either *committed* or *aborted*. On the other hand, *read (shared)* locks are released regularly during phase 2. This protocol is not appropriate in B-trees because it causes Bottleneck (while B-trees always starts searching from the parent root).

Strong strict two-phase locking

or **Rigorousness**, or **Rigorous scheduling**, or **Rigorous two-phase locking**

To comply with **strong strict two-phase locking** (SS2PL) the locking protocol releases both *write (exclusive)* and *read (shared)* locks applied by a transaction only after the transaction has ended, i.e., only after both completing executing (being *ready*) and becoming either *committed* or *aborted*. This protocol also complies with the S2PL rules. A transaction obeying SS2PL can be viewed as having phase-1 that lasts the transaction's entire execution duration, and no phase-2 (or a degenerate phase-2). Thus, only one phase is actually left, and "two-phase" in the name seems to be still utilized due to the historical development of the concept from 2PL, and 2PL being a super-class. The SS2PL property of a schedule is also called **Rigorousness**. It is also the name of the class of schedules having this property, and an SS2PL schedule is also called a "rigorous schedule". The term "Rigorousness" is free of the unnecessary legacy of "two-phase," as well as being independent of any (locking) mechanism (in principle other blocking mechanisms can be utilized). The property's respective locking mechanism is sometimes referred to as **Rigorous 2PL**.

SS2PL is a special case of S2PL, i.e., the SS2PL class of schedules is a proper subclass of S2PL (every SS2PL schedule is also an S2PL schedule, but S2PL schedules exist that are not SS2PL).

SS2PL has been the concurrency control protocol of choice for most database systems and utilized since their early days in the 1970s. It is proven to be an effective mechanism in many situations, and provides besides Serializability also Strictness (a special case of cascadeless Recoverability), which is instrumental for efficient database recovery, and also Commitment ordering (CO) for participating in distributed environments where a CO based distributed serializability and global serializability solutions are employed. Being a subset of CO, an efficient implementation of *distributed SS2PL* exists without a distributed lock manager (DLM), while distributed deadlocks (see below) are resolved automatically. The fact that SS2PL employed in multi database systems ensures global serializability has been known for years before the discovery of CO, but only with CO came the understanding of the role of an atomic commitment protocol in maintaining global serializability, as well as the observation of automatic distributed deadlock resolution (see a detailed example of Distributed SS2PL). As a matter of fact, SS2PL inheriting properties of Recoverability and CO is more significant than being a subset of 2PL, which by itself in its general form, besides comprising a simple serializability mechanism (however serializability is also implied by CO), in not known to provide SS2PL with any other significant qualities. 2PL in its general form, as well as when combined with Strictness, i.e., Strict 2PL (S2PL), are not known to be utilized in practice. The popular SS2PL does not require marking "end of phase-1" as 2PL and S2PL do, and thus is simpler to implement. Also, unlike the general 2PL, SS2PL provides, as mentioned above, the useful Strictness and Commitment ordering properties.

Many variants of SS2PL exist that utilize various lock types with various semantics in different situations, including cases of lock-type change during a transaction. Notable are variants that use Multiple granularity locking.

Comments:

1. SS2PL Vs. S2PL: Both provide Serializability and Strictness. Since S2PL is a super class of SS2PL it may, in principle, provide more concurrency. However, no concurrency advantage is typically practically noticed (exactly same locking exists for both, with practically not much earlier lock release for S2PL), and the overhead of dealing with an end-of-phase-1 mechanism in S2PL, separate from transaction-end, is not justified. Also, while SS2PL provides Commitment ordering, S2PL does not. This explains the preference of SS2PL over S2PL.

2. Especially before 1990, but also after, in many articles and books, e.g., (Bernstein et al. 1987, p. 59),[1] the term "Strict 2PL" (S2PL) has been frequently defined by the locking protocol "Release all locks only after transaction end," which is the pro-

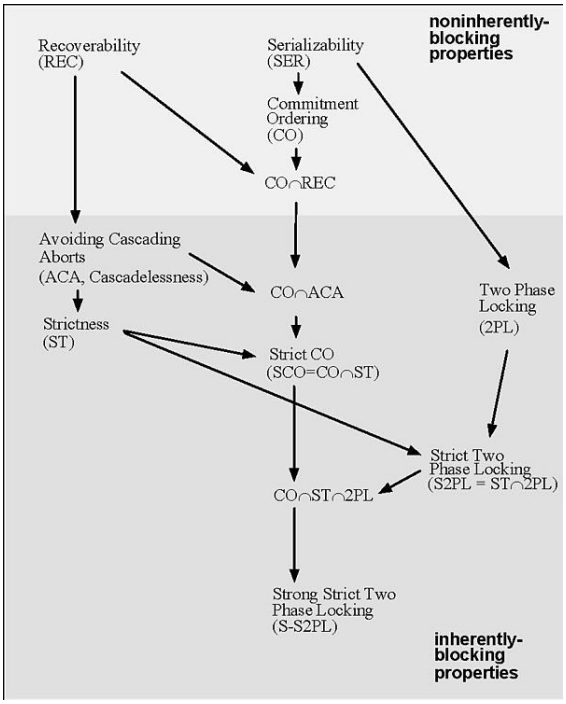
tocol of SS2PL. Thus, "Strict 2PL" could not be there the name of the intersection of Strictness and 2PL, which is larger than the class generated by the SS2PL protocol. This has caused confusion. With an explicit definition of S2PL as the intersection of Strictness and 2PL, a new name for SS2PL, and an explicit distinction between the classes S2PL and SS2PL, the articles (Breitbart et al. 1991)[3] and (Raz 1992)[4] have intended to clear the confusion: The first using the name "Rigorousness," and the second "SS2PL."

3. A more general property than SS2PL exists (a schedule super-class), **Strict commitment ordering** (Strict CO, or SCO), which as well provides both serializability, strictness, and CO, and has similar locking overhead. Unlike SS2PL, SCO does not block upon a read-write conflict (a read-lock does not block acquiring a write-lock; both SCO and SS2PL have the same behavior for write-read and write-write conflicts) at the cost of a possible delayed commit, and upon such conflict type SCO has shorter average transaction completion time and better performance than SS2PL.[5] While SS2PL obeys the *lock compatibility table* above, SCO has the following table:

> Note that though SCO releases all locks at transaction end and complies with the 2PL locking rules, SCO is not a subset of 2PL because of its different lock compatibility table. SCO allows materialized read-write conflicts between two transactions in their phases 1, which 2PL does not allow in phase-1 (see about materialized conflicts in Serializability). On the other hand 2PL allows other materialized conflict types in phase-2 that SCO does not allow at all. Together this implies that the schedule classes 2PL and SCO are incomparable (i.e., no class contains the other class).

Summary - Relationships among classes

Between any two schedule classes (define by their schedules' respective properties) that have common schedules, either one *contains* the other (*strictly contains* if they are not equal), or they are *incomparable*. The containment relationships among the 2PL classes and other major schedule classes are summarized in the following diagram. 2PL and its subclasses are *inherently blocking*, which means that no optimistic implementations for them exist (and whenever "Optimistic 2PL" is mentioned it refers to a different mechanism with a class that includes also schedules not in the 2PL class).



Schedule classes containment: *An arrow from class A to class B indicates that class A strictly contains B; a lack of a directed path between classes means that the classes are incomparable. A property is* ***inherently blocking****, if it can be enforced only by blocking transaction's data access operations until certain events occur in other transactions. (Raz 1992)*

6.3.3 Deadlocks in 2PL

Locks block data-access operations. Mutual blocking between transactions results in a *deadlock*, where execution of these transactions is stalled, and no completion can be reached. Thus deadlocks need to be resolved to complete these transactions' executions and release related computing resources. A deadlock is a reflection of a potential cycle in the *precedence graph*, that would occur without the blocking. A deadlock is resolved by aborting a transaction involved with such potential cycle, and breaking the cycle. It is often detected using a *wait-for graph* (a graph of conflicts blocked by locks from being materialized; conflicts not materialized in the database due to blocked operations are not reflected in the precedence graph and do not affect *serializability*), which indicates which transaction is "waiting for" lock release by which transaction, and a cycle means a deadlock. Aborting one transaction per cycle is sufficient to break the cycle. Transactions aborted due to deadlock resolution are executed again immediately.

In a distributed environment an atomic commitment protocol, typically the Two-phase commit (2PC) protocol, is utilized for atomicity. When recoverable data (data under

transaction control) are partitioned among 2PC participants (i.e., each data object is controlled by a single 2PC participant), then distributed (global) deadlocks, deadlocks involving two or more participants in 2PC, are resolved automatically as follows:

When SS2PL is effectively utilized in a distributed environment, then global deadlocks due to locking generate voting-deadlocks in 2PC, and are resolved automatically by 2PC (see Commitment ordering (CO), in Exact characterization of voting-deadlocks by global cycles; No reference except the CO articles is known to notice this). For the general case of 2PL, global deadlocks are similarly resolved automatically by the synchronization point protocol of phase-1 end in a distributed transaction (synchronization point is achieved by "voting" (notifying local phase-1 end), and being propagated to the participants in a distributed transaction the same way as a decision point in atomic commitment; in analogy to decision point in CO, a conflicting operation in 2PL cannot happen before phase-1 end synchronization point, with the same resulting voting-deadlock in the case of a global data-access deadlock; the voting-deadlock (which is also a locking based global deadlock) is automatically resolved by the protocol aborting some transaction involved, with a missing vote, typically using a timeout).

Comment:

> When data are partitioned among the *atomic commitment protocol* (e.g., 2PC) participants, automatic *global deadlock* resolution has been overlooked in the database research literature, though deadlocks in such systems has been a quite intensive research area:
>
> - For CO and its special case SS2PL, the automatic resolution by the *atomic commitment protocol* has been noticed only in the CO articles. However, it has been noticed in practice that in many cases global deadlocks are very infrequently detected by the dedicated resolution mechanisms, less than could be expected ("Why do we see so few global deadlocks?"). The reason is probably the deadlocks that are automatically resolved and thus not handled and uncounted by the mechanisms;
> - For 2PL in general, the automatic resolution by the (mandatory) *end-of-phase-one synchronization point protocol* (which has same voting mechanism as atomic commitment protocol, and same missing vote handling upon voting deadlock, resulting in global deadlock resolution) has not been mentioned until today (2009). Practically only the special case SS2PL is utilized, where no end-of-phase-one synchronization is needed in addition to atomic commit protocol.
>
> In a distributed environment where recoverable data are not partitioned among atomic commitment protocol participants, no such automatic resolution exists, and distributed deadlocks need to be resolved by dedicated techniques.

6.3.4 See also

- Serializability
- Lock (computer science)

6.3.5 References

[1] Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman (1987): *Concurrency Control and Recovery in Database Systems*, Addison Wesley Publishing Company, ISBN 0-201-10715-5

[2] Gerhard Weikum, Gottfried Vossen (2001): *Transactional Information Systems*, Elsevier, ISBN 1-55860-508-8

[3] Yuri Breitbart, Dimitrios Georgakopoulos, Marek Rusinkiewicz, Abraham Silberschatz (1991): "On Rigorous Transaction Scheduling", *IEEE Transactions on Software Engineering* (TSE), September 1991, Volume 17, Issue 9, pp. 954-960, ISSN: 0098-5589

[4] Yoav Raz (1992): "The Principle of Commitment Ordering, or Guaranteeing Serializability in a Heterogeneous Environment of Multiple Autonomous Resource Managers Using Atomic Commitment" (PDF), *Proceedings of the Eighteenth International Conference on Very Large Data Bases* (VLDB), pp. 292-312, Vancouver, Canada, August 1992, ISBN 1-55860-151-1 (also DEC-TR 841, Digital Equipment Corporation, November 1990)

[5] Yoav Raz (1991): "Locking Based Strict Commitment Ordering, or How to improve Concurrency in Locking Based Resource Managers", DEC-TR 844, December 1991.

Chapter 7

MVCC

7.1 Multiversion concurrency control

Multiversion concurrency control (**MCC** or **MVCC**), is a concurrency control method commonly used by database management systems to provide concurrent access to the database and in programming languages to implement transactional memory.[1]

If someone is reading from a database at the same time as someone else is writing to it, it is possible that the reader will see a half-written or inconsistent piece of data. There are several ways of solving this problem, known as concurrency control methods. The simplest way is to make all readers wait until the writer is done, which is known as a lock. This can be very slow, so MVCC takes a different approach: each user connected to the database sees a *snapshot* of the database at a particular instant in time. Any changes made by a writer will not be seen by other users of the database until the changes have been completed (or, in database terms: until the transaction has been committed.)

When an MVCC database needs to update an item of data, it will not overwrite the old data with new data, but instead mark the old data as obsolete and add the newer version elsewhere. Thus there are multiple versions stored, but only one is the latest. This allows readers to access the data that was there when they began reading, even if it was modified or deleted part way through by someone else. It also allows the database to avoid the overhead of filling in holes in memory or disk structures but requires (generally) the system to periodically sweep through and delete the old, obsolete data objects. For a document-oriented database it also allows the system to optimize documents by writing entire documents onto contiguous sections of disk—when updated, the entire document can be re-written rather than bits and pieces cut out or maintained in a linked, non-contiguous database structure.

MVCC provides *point in time* consistent views. Read transactions under MVCC typically use a timestamp or transaction ID to determine what state of the DB to read, and read these versions of the data. Read and write transactions are thus isolated from each other without any need for locking. Writes create a newer version, while concurrent reads access the older version.

7.1.1 Implementation

MVCC uses timestamps (**TS**), and *incrementing transaction IDs* (**T**), to achieve *transactional consistency*. MVCC ensures a transaction (**T**) never has to wait to *Read* a database object (**P**) by maintaining several versions of such object (**P**). Each version of the object (**P**) would have both a *Read Timestamp* (**RTS**) and a *Write Timestamp* (**WTS**) which lets a transaction ($\mathbf{T_i}$) read the most recent version of an object (**P**) which precedes the transaction's ($\mathbf{T_i}$) *Read Timestamp* (**RTS**($\mathbf{T_i}$)).

If a transaction ($\mathbf{T_i}$) wants to *Write* to an object (**P**), and if there is also another transaction ($\mathbf{T_k}$) happening to the same object (**P**), the Read Timestamp (**RTS**) of ($\mathbf{T_i}$) must precede the Read Timestamp (**RTS**) of ($\mathbf{T_k}$), (i.e., **RTS**($\mathbf{T_i}$) < **RTS**($\mathbf{T_k}$)) for the object (**P**) *Write Operation* (**WTS**) to succeed. Basically, a *Write* cannot complete if there are other outstanding transactions with an earlier Read Timestamp (**RTS**) to the same object (**P**). Think of it like standing in line at the store, you cannot complete your checkout transaction until those in front of you have completed theirs.

To restate; every object (**P**) has a *Timestamp* (**TS**), however if transaction $\mathbf{T_i}$ wants to *Write* to object (**P**), and there is a *Timestamp* (**TS**) of that transaction that is earlier than the object's current Read Timestamp, (**TS**(**P**) < **RTS**($\mathbf{T_i}$)), the transaction $\mathbf{T_i}$ is aborted and restarted. (If you try to cut in line, to check out early, go to the back of that line) Otherwise, $\mathbf{T_i}$ creates a new version of (**P**) and sets the read/write timestamp (**TS**) of the new version of (**P**) to the timestamp of the transaction **TS=TS**($\mathbf{T_i}$).[2]

The obvious drawback to this system is the cost of storing multiple versions of objects in the database. On the other hand reads are never blocked, which can be important for workloads mostly involving reading values from the database. MVCC is particularly adept at implementing true snapshot isolation, something which other methods of concurrency control frequently do either incompletely or with high performance costs.

7.1.2 Examples

Concurrent read-write

At Time = 1, the state of a database could be:

T0 wrote Object 1="Foo" and Object 2="Bar". After that T1 wrote Object 1="Hello" leaving Object 2 at its original value. The new value of Object 1 will supersede the value at 0 for all transaction that starts after T1 commits at which point version 0 of Object 1 can be garbage collected.

If a long running transaction T2 starts a read operation of Object 2 and Object 1 after T1 committed and there is a concurrent update transaction T3 which deletes Object 2 and adds Object 3="Foo-Bar", the database state will look like at time 2:

There is a new version as of time 2 of Object 2 which is marked as deleted and a new Object 3. Since T2 and T3 run concurrently T2 sees another the version of the database before 2 i.e. before T3 committed writes, as such T2 reads Object 2="Bar" and Object 1="Hello". This is how MVCC allows snapshot isolation reads in almost every case without any locks.

7.1.3 History

Multiversion concurrency control is described in some detail in the 1981 paper "Concurrency Control in Distributed Database Systems"[3] by Phil Bernstein and Nathan Goodman, then employed by the Computer Corporation of America. Bernstein and Goodman's paper cites a 1978 dissertation[4] by David P. Reed which quite clearly describes MVCC and claims it as an original work.

The first shipping, commercial database software product featuring MVCC was Digital's VAX Rdb/ELN. The second was InterBase, which is still an active, commercial product.

7.1.4 Version control systems

Any version control system that has the internal notion of a version (e.g. Subversion, Git, probably almost any current VCS with the notable exception of CVS) will provide explicit MVCC (you only ever access data by its version identifier).

Among the VCSs that don't provide MVCC at the repository level, most still work with the notion of a *working copy*, which is a file tree checked out from the repository, edited without using the VCS itself and checked in after the edit. This working copy provides MVCC while it is checked out.

7.1.5 See also

- List of databases using MVCC
- Timestamp-based concurrency control
- Clojure
- Read-copy-update
- Vector clock

7.1.6 References

[1] refs. Clojure. Retrieved on 2013-09-18.

[2] Ramakrishnan, R., & Gehrke, J. (2000). Database management systems. Osborne/McGraw-Hill.

[3] Bernstein, Philip A.; Goodman, Nathan (1981). "Concurrency Control in Distributed Database Systems". *ACM Computing Surveys*.

[4] Reed, David P. (September 21, 1978). "Naming and Synchronization in a Decentralized Computer System". *MIT dissertation*.

7.1.7 Further reading

- Gerhard Weikum, Gottfried Vossen, *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*, Morgan Kaufmann, 2002, ISBN 1-55860-508-8

7.2 Snapshot isolation

In databases, and transaction processing (transaction management), **snapshot isolation** is a guarantee that all reads made in a transaction will see a consistent snapshot of the database (in practice it reads the last committed values that existed at the time it started), and the transaction itself will successfully commit only if no updates it has made conflict with any concurrent updates made since that snapshot.

Snapshot isolation has been adopted by several major database management systems, such as SQL Anywhere, InterBase, Firebird, Oracle, PostgreSQL, MongoDB[1] and Microsoft SQL Server (2005 and later). The main reason for its adoption is that it allows better performance than serializability, yet still avoids most of the concurrency anomalies that serializability avoids (but not always all). In practice snapshot isolation is implemented within multiversion concurrency control (MVCC), where generational values of each data item (versions) are maintained: MVCC is a common way to increase concurrency and performance by generating a new version of a database object each time the object is written, and allowing transactions' read operations of several last relevant versions

(of each object). Snapshot isolation has also been used[2] to critique the ANSI SQL−92 standard's definition of isolation levels, as it exhibits none of the "anomalies" that the SQL standard prohibited, yet is not serializable (the anomaly-free isolation level defined by ANSI).

Snapshot isolation is called "serializable" mode in Oracle[3][4]<ref name"">Ask Tom : "Serializable Transaction"</ref> and PostgreSQL versions prior to 9.1,[5][6][7] which may cause confusion with the "real serializability" mode. There are arguments both for and against this decision; what is clear is that users must be aware of the distinction to avoid possible undesired anomalous behavior in their database system logic.

7.2.1 Definition

A transaction executing under snapshot isolation appears to operate on a personal *snapshot* of the database, taken at the start of the transaction. When the transaction concludes, it will successfully commit only if the values updated by the transaction have not been changed externally since the snapshot was taken. Such a write-write conflict will cause the transaction to abort.

In a *write skew* anomaly, two transactions (T1 and T2) concurrently read an overlapping data set (e.g. values V1 and V2), concurrently make disjoint updates (e.g. T1 updates V1, T2 updates V2), and finally concurrently commit, neither having seen the update performed by the other. Were the system serializable, such an anomaly would be impossible, as either T1 or T2 would have to occur "first", and be visible to the other. In contrast, snapshot isolation permits write skew anomalies.

As a concrete example, imagine V1 and V2 are two balances held by a single person, Phil. The bank will allow either V1 or V2 to run a deficit, provided the total held in both is never negative (i.e. V1 + V2 ≥ 0). Both balances are currently $100. Phil initiates two transactions concurrently, T1 withdrawing $200 from V1, and T2 withdrawing $200 from V2.

If the database guaranteed serializable transactions, the simplest way of coding T1 is to deduct $200 from V1, and then verify that V1 + V2 ≥ 0 still holds, aborting if not. T2 similarly deducts $200 from V2 and then verifies V1 + V2 ≥ 0. Since the transactions must serialize, either T1 happens first, leaving V1 = -$100, V2 = $100, and preventing T2 from succeeding (since V1 + (V2 - $200) is now -$200), or T2 happens first and similarly prevents T1 from committing.

Under snapshot isolation, however, T1 and T2 operate on private snapshots of the database: each deducts $200 from an account, and then verifies that the new total is zero, using the other account value that held when the snapshot was taken. Since neither *update* conflicts, both commit successfully, leaving V1 = V2 = -$100, and V1 + V2 = -$200.

If built on multiversion concurrency control, snapshot isolation allows transactions to proceed without worrying about concurrent operations, and more importantly without needing to re-verify all read operations when the transaction finally commits. The only information that must be stored during the transaction is a list of updates made, which can be scanned for conflicts fairly easily before being committed.

7.2.2 Workarounds

Potential inconsistency problems arising from write skew anomalies can be fixed by adding (otherwise unnecessary) updates to the transactions in order to enforce the serializability property.[8]

Materialize the conflict Add a special conflict table, which both transactions update in order to create a direct write-write conflict.

Promotion Have one transaction "update" a read-only location (replacing a value with the same value) in order to create a direct write-write conflict (or use an equivalent promotion, e.g. Oracle's SELECT FOR UPDATE).

In the example above, we can materialize the conflict by adding a new table which makes the hidden constraint explicit, mapping each person to their *total balance*. Phil would start off with a total balance of $200, and each transaction would attempt to subtract $200 from this, creating a write-write conflict that would prevent the two from succeeding concurrently. This approach violates the normal form.

Alternatively, we can promote one of the transaction's reads to a write. For instance, T2 could set V1 = V1, creating an artificial write-write conflict with T1 and, again, preventing the two from succeeding concurrently. This solution may not always be possible.

In general, therefore, snapshot isolation puts some of the problem of maintaining non-trivial constraints onto the user, who may not appreciate either the potential pitfalls or the possible solutions. The upside to this transfer is better performance.

7.2.3 History

Snapshot isolation arose from work on multiversion concurrency control databases, where multiple versions of the database are maintained concurrently to allow readers to execute without colliding with writers. Such a system allows a natural definition and implementation of such an isolation level.[2] InterBase, later owned by Borland, was acknowledged to provide SI rather than full serializability in version 4,[2] and likely permitted write-skew anomalies since its first release in 1985.[9]

Unfortunately, the ANSI SQL-92 standard was written with a lock-based database in mind, and hence is rather vague when applied to MVCC systems. Berenson *et al.* wrote a paper in 1995[2] critiquing the SQL standard, and cited snapshot isolation as an example of an isolation level that did not exhibit the standard anomalies described in the ANSI SQL-92 standard, yet still had anomalous behaviour when compared with serializable transactions.

In 2008, Cahill *et al.* showed that write-skew anomalies could be prevented by detecting and aborting "dangerous" triplets of concurrent transactions.[10] This implementation of serializability is well-suited to multiversion concurrency control databases, and has been adopted in PostgreSQL 9.1,[6][7][11] where it is referred to as "Serializable Snapshot Isolation", abbreviated to SSI. When used consistently, this eliminates the need for the above workarounds. The downside over snapshot isolation is an increase in aborted transactions. This can perform better or worse than snapshot isolation with the above workarounds, depending on workload.

7.2.4 References

[1] Multiversion concurrency control in MongoDB, MongoDB CTO: How our new WiredTiger storage engine will earn its stripes

[2] Berenson, Hal; Bernstein, Phil; Gray, Jim; Melton, Jim; O'Neil, Elizabeth; O'Neil, Patrick (1995), "A Critique of ANSI SQL Isolation Levels", *Proceedings of the 1995 ACM SIGMOD international Conference on Management of Data*, pp. 1–10, doi:10.1145/223784.223785

[3] Oracle Database Concepts 10g Release 1 (10.1) Chapter 13 : Data Concurrency and Consistency — Oracle Isolation Levels

[4] Ask Tom : On Transaction Isolation Levels

[5] PostgreSQL 9.0 Documentation: 13.2.2.1. Serializable Isolation versus True Serializability

[6] PostgreSQL 9.1 press release

[7] PostgreSQL 9.1.14 Documentation: 13.2.3. Serializable Isolation Level

[8] Fekete, Alan; Liarokapis, Dimitrios; O'Neil, Elizabeth; O'Neil, Patrick; Shasha, Dennis (2005), "Making Snapshot Isolation Serializable", *ACM Transactions on Database Systems* **30** (2): 492–528, doi:10.1145/1071610.1071615, ISSN 0362-5915

[9] Stuntz, Craig. "Multiversion Concurrency Control Before InterBase". Retrieved October 30, 2014.

[10] Michael J. Cahill, Uwe Röhm, Alan D. Fekete (2008) "Serializable isolation for snapshot databases", *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pp. 729–738, ISBN 978-1-60558-102-6 (SIGMOD 2008 best paper award)

[11] Ports, Dan R. K.; Grittner, Kevin (2012). "Serializable Snapshot Isolation in PostgreSQL" (PDF). *Proceedings of the VLDB Endowment* **5** (12): 1850–1861.

7.2.5 Further reading

- Gerhard Weikum, Gottfried Vossen, *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*, Morgan Kaufmann, 2002, ISBN 1-55860-508-8

- Khuzaima Daudjee, Kenneth Salem, *Lazy Database Replication with Snapshot Isolation*, VLDB 2006: pages 715-726

7.3 Two-phase commit protocol

"2PC" redirects here. For the play in American and Canadian football, see Two-point conversion. For the American rapper, see 2Pac. For the cryptographic protocol, see Commitment scheme.

In transaction processing, databases, and computer networking, the **two-phase commit protocol** (**2PC**) is a type of atomic commitment protocol (ACP). It is a distributed algorithm that coordinates all the processes that participate in a distributed atomic transaction on whether to *commit* or *abort* (*roll back*) the transaction (it is a specialized type of consensus protocol). The protocol achieves its goal even in many cases of temporary system failure (involving either process, network node, communication, etc. failures), and is thus widely utilized.[1][2][3] However, it is not resilient to all possible failure configurations, and in rare cases, user (e.g., a system's administrator) intervention is needed to remedy an outcome. To accommodate recovery from failure (automatic in most cases) the protocol's participants use logging of the protocol's states. Log records, which are typically slow to generate but survive failures, are used by the protocol's recovery procedures. Many protocol variants exist that primarily differ in logging strategies and recovery mechanisms. Though usually intended to be used infrequently, recovery procedures compose a substantial portion of the protocol, due to many possible failure scenarios to be considered and supported by the protocol.

In a "normal execution" of any single distributed transaction, i.e., when no failure occurs, which is typically the most frequent situation, the protocol consists of two phases:

1. The *commit-request phase* (or *voting phase*), in which a *coordinator* process attempts to prepare all the transaction's participating processes (named *participants*, *cohorts*, or *workers*) to take the necessary steps for either committing or aborting the transaction and to *vote*, either "Yes": commit (if the transaction participant's local portion execution has ended properly), or "No": abort (if a problem has been detected with the local portion), and

2. The *commit phase*, in which, based on *voting* of the cohorts, the coordinator decides whether to commit (only if *all* have voted "Yes") or abort the transaction (otherwise), and notifies the result to all the cohorts. The cohorts then follow with the needed actions (commit or abort) with their local transactional resources (also called *recoverable resources*; e.g., database data) and their respective portions in the transaction's other output (if applicable).

Note that the two-phase commit (2PC) protocol should not be confused with the two-phase locking (2PL) protocol, a concurrency control protocol.

7.3.1 Assumptions

The protocol works in the following manner: one node is designated the **coordinator**, which is the master site, and the rest of the nodes in the network are designated the **cohorts**. The protocol assumes that there is stable storage at each node with a write-ahead log, that no node crashes forever, that the data in the write-ahead log is never lost or corrupted in a crash, and that any two nodes can communicate with each other. The last assumption is not too restrictive, as network communication can typically be rerouted. The first two assumptions are much stronger; if a node is totally destroyed then data can be lost.

The protocol is initiated by the coordinator after the last step of the transaction has been reached. The cohorts then respond with an **agreement** message or an **abort** message depending on whether the transaction has been processed successfully at the cohort.

7.3.2 Basic algorithm

Commit request phase

or **voting phase**

1. The coordinator sends a **query to commit** message to all cohorts and waits until it has received a reply from all cohorts.
2. The cohorts execute the transaction up to the point where they will be asked to commit. They each write an entry to their *undo log* and an entry to their *redo log*.
3. Each cohort replies with an **agreement** message (cohort votes **Yes** to commit), if the cohort's actions succeeded, or an **abort** message (cohort votes **No**, not to commit), if the cohort experiences a failure that will make it impossible to commit.

Commit phase

or **Completion phase**

Success If the coordinator received an **agreement** message from *all* cohorts during the commit-request phase:

1. The coordinator sends a **commit** message to all the cohorts.
2. Each cohort completes the operation, and releases all the locks and resources held during the transaction.
3. Each cohort sends an **acknowledgment** to the coordinator.
4. The coordinator completes the transaction when all acknowledgments have been received.

Failure If *any* cohort votes **No** during the commit-request phase (or the coordinator's timeout **expires**):

1. The coordinator sends a **rollback** message to all the cohorts.
2. Each cohort undoes the transaction using the undo log, and releases the resources and locks held during the transaction.
3. Each cohort sends an **acknowledgement** to the coordinator.
4. The coordinator undoes the transaction when all acknowledgements have been received.

Message flow Coordinator Cohort QUERY TO COMMIT ----------> VOTE YES/NO prepare*/abort* <---------- commit*/abort* COMMIT/ROLLBACK ----------> ACKNOWLEDGMENT commit*/abort* <---------- end

An * next to the record type means that the record is forced to stable storage.[4]

7.3.3 Disadvantages

The greatest disadvantage of the two-phase commit protocol is that it is a blocking protocol. If the coordinator fails permanently, some cohorts will never resolve their transactions: After a cohort has sent an **agreement** message to the coordinator, it will block until a **commit** or **rollback** is received.

7.3.4 Implementing the two-phase commit protocol

Common architecture

In many cases the 2PC protocol is distributed in a computer network. It is easily distributed by implementing multiple dedicated 2PC components similar to each other, typically named *Transaction managers* (TMs; also referred to as *2PC agents* or Transaction Processing Monitors), that carry out the protocol's execution for each transaction (e.g., The Open Group's X/Open XA). The databases involved with a distributed transaction, the *participants*, both the coordinator and cohorts, *register* to close TMs (typically residing on respective same network nodes as the participants) for terminating that transaction using 2PC. Each distributed transaction has an ad hoc set of TMs, the TMs to which the transaction participants register. A leader, the coordinator TM, exists for each transaction to coordinate 2PC for it, typically the TM of the coordinator database. However, the coordinator role can be transferred to another TM for performance or reliability reasons. Rather than exchanging 2PC messages among themselves, the participants exchange the messages with their respective TMs. The relevant TMs communicate among themselves to execute the 2PC protocol schema above, "representing" the respective participants, for terminating that transaction. With this architecture the protocol is fully distributed (does not need any central processing component or data structure), and scales up with number of network nodes (network size) effectively.

This common architecture is also effective for the distribution of other atomic commitment protocols besides 2PC, since all such protocols use the same voting mechanism and outcome propagation to protocol participants.[1][2]

Protocol optimizations

Database research has been done on ways to get most of the benefits of the two-phase commit protocol while reducing costs by *protocol optimizations*[1][2][3] and protocol operations saving under certain system's behavior assumptions.

Presume abort and Presume commit *Presumed abort* or *Presumed commit* are common such optimizations.[2][3][5] An assumption about the outcome of transactions, either commit, or abort, can save both messages and logging operations by the participants during the 2PC protocol's execution. For example, when presumed abort, if during system recovery from failure no logged evidence for commit of some transaction is found by the recovery procedure, then it assumes that the transaction has been aborted, and acts accordingly. This means that it does not matter if aborts are logged at all, and such logging can be saved under this assumption. Typically a penalty of additional operations is paid during recovery from failure, depending on optimization type. Thus the best variant of optimization, if any, is chosen according to failure and transaction outcome statistics.

Tree two-phase commit protocol The **Tree 2PC protocol**[2] (also called *Nested 2PC*, or *Recursive 2PC*) is a common variant of 2PC in a computer network, which better utilizes the underlying communication infrastructure. The participants in a distributed transaction are typically invoked in an order which defines a tree structure, the *invocation tree*, where the participants are the nodes and the edges are the invocations (communication links). The same tree is commonly utilized to complete the transaction by a 2PC protocol, but also another communication tree can be utilized for this, in principle. In a tree 2PC the coordinator is considered the root ("top") of a communication tree (inverted tree), while the cohorts are the other nodes. The coordinator can be the node that originated the transaction (invoked recursively (transitively) the other participants), but also another node in the same tree can take the coordinator role instead. 2PC messages from the coordinator are propagated "down" the tree, while messages to the coordinator are "collected" by a cohort from all the cohorts below it, before it sends the appropriate message "up" the tree (except an **abort** message, which is propagated "up" immediately upon receiving it or if the current cohort initiates the abort).

The **Dynamic two-phase commit** (Dynamic two-phase commitment, D2PC) **protocol**[2][6] is a variant of Tree 2PC with no predetermined coordinator. It subsumes several optimizations that have been proposed earlier. **Agreement** messages (**Yes** votes) start to propagate from all the leaves, each leaf when completing its tasks on behalf of the transaction (becoming *ready*). An intermediate (non leaf) node sends when *ready* an **agreement** message to the last (single) neighboring node from which **agreement** message has not yet been received. The coordinator is determined dynamically by racing **agreement** messages over the transaction tree, at the place where they collide. They collide either at a transaction tree node, to be the coordinator, or on a tree edge. In the latter case one of the two edge's nodes is elected as a coordinator (any node). D2PC is time optimal (among all the instances of a specific transaction tree, and any specific Tree 2PC protocol implementation; all instances have the same tree; each instance has a different node as coordinator): By choosing an optimal coordinator D2PC commits both the coordinator and each cohort in minimum possible time, allowing the earliest possible release of locked resources in each transaction participant (tree node).

7.3.5 See also

- Atomic commit
- Commit (data management)

- Three-phase commit protocol
- XA
- Paxos algorithm
- Two Generals' Problem

7.3.6 References

[1] Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman (1987): *Concurrency Control and Recovery in Database Systems*, Chapter 7, Addison Wesley Publishing Company, ISBN 0-201-10715-5

[2] Gerhard Weikum, Gottfried Vossen (2001): *Transactional Information Systems*, Chapter 19, Elsevier, ISBN 1-55860-508-8

[3] Philip A. Bernstein, Eric Newcomer (2009): *Principles of Transaction Processing*, 2nd Edition, Chapter 8, Morgan Kaufmann (Elsevier), ISBN 978-1-55860-623-4

[4] C. Mohan, Bruce Lindsay and R. Obermarck (1986): "Transaction management in the R* distributed database management system",*ACM Transactions on Database Systems (TODS)*, Volume 11 Issue 4, Dec. 1986, Pages 378 - 396

[5] C. Mohan, Bruce Lindsay (1985): "Efficient commit protocols for the tree of processes model of distributed transactions",*ACM SIGOPS Operating Systems Review*, 19(2),pp. 40-52 (April 1985)

[6] Yoav Raz (1995): "The Dynamic Two Phase Commitment (D2PC) protocol ",*Database Theory — ICDT '95*, *Lecture Notes in Computer Science*, Volume 893/1995, pp. 162-176, Springer, ISBN 978-3-540-58907-5

7.3.7 External links

- Two Phase Commit protocol explained in Pictures by exploreDatabase

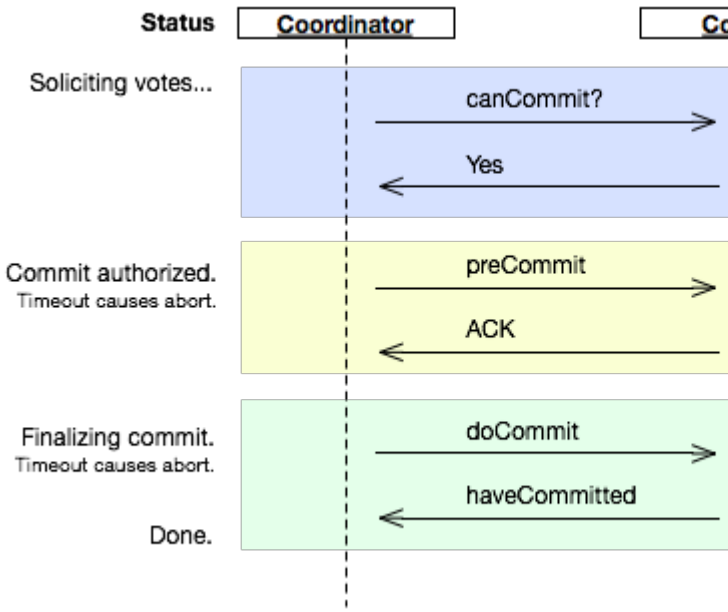
7.4 Three-phase commit protocol

In computer networking and databases, the **three-phase commit protocol** (**3PC**)[1] is a distributed algorithm which lets all nodes in a distributed system agree to commit a transaction. Unlike the two-phase commit protocol (2PC) however, 3PC is non-blocking. Specifically, 3PC places an upper bound on the amount of time required before a transaction either commits or aborts. This property ensures that if a given transaction is attempting to commit via 3PC and holds some resource locks, it will release the locks after the timeout.

3PC was originally described by Dale Skeen and Michael Stonebraker in their paper, "A Formal Model of Crash Recovery in a Distributed System".[1] In that work, they modeled 2PC as a system of non-deterministic finite state automata and proved that it is not resilient to a random single site failure. The basic observation is that in 2PC, while one site is in the "prepared to commit" state, the other may be in either the "commit" or the "abort" state. From this analysis, they developed 3PC to avoid such states and it is thus resilient to such failures.

7.4.1 Protocol Description

In describing the protocol, we use terminology similar to that used in the two-phase commit protocol. Thus we have a single coordinator site leading the transaction and a set of one or more cohorts being directed by the coordinator.



Coordinator

1. The coordinator receives a transaction request. If there is a failure at this point, the coordinator aborts the transaction (i.e. upon recovery, it will consider the transaction aborted). Otherwise, the coordinator sends a **canCommit?** message to the cohorts and moves to the waiting state.
2. If there is a failure, timeout, or if the coordinator receives a **No** message in the waiting state, the coordinator aborts the transaction and sends an **abort** message to all cohorts. Otherwise the coordinator will receive **Yes** messages from all cohorts within the time window, so it sends **preCommit** messages to all cohorts and moves to the prepared state.
3. If the coordinator succeeds in the prepared state, it will move to the commit state. However if the coordinator times out while waiting for an acknowledgement from a cohort, it will abort the transaction. In

the case where all acknowledgements are received, the coordinator moves to the commit state as well.

Cohort

1. The cohort receives a **canCommit?** message from the coordinator. If the cohort agrees it sends a **Yes** message to the coordinator and moves to the prepared state. Otherwise it sends a **No** message and aborts. If there is a failure, it moves to the abort state.
2. In the prepared state, if the cohort receives an **abort** message from the coordinator, fails, or times out waiting for a commit, it aborts. If the cohort receives a **preCommit** message, it sends an **ACK** message back and awaits a final **commit** or **abort**.
3. If, after a cohort member receives a **preCommit** message, the coordinator fails or times out, the cohort member goes forward with the commit.

7.4.2 Motivation

A Two-phase commit protocol cannot dependably recover from a failure of both the **coordinator** and a cohort member during the **Commit phase**. If only the **coordinator** had failed, and no cohort members had received a **commit** message, it could safely be inferred that no **commit** had happened. If, however, both the **coordinator** and a cohort member failed, it is possible that the failed cohort member was the first to be notified, and had actually done the **commit**. Even if a new **coordinator** is selected, it cannot confidently proceed with the operation until it has received an agreement from **all** cohort members ... and hence must block until all cohort members respond.

The Three-phase commit protocol eliminates this problem by introducing the **Prepared to commit** state. If the **coordinator** fails before sending **preCommit** messages, the **cohort** will unanimously agree that the operation was **aborted**. The **coordinator** will not send out a **doCommit** message until **all** cohort members have **ACK**ed that they are **Prepared to commit**. This eliminates the possibility that **any** cohort member actually completed the transaction before **all** cohort members were aware of the decision to do so (an ambiguity that necessitated indefinite blocking in the Two-phase commit protocol).

7.4.3 Disadvantages

The main disadvantage to this algorithm is that it cannot recover in the event the network is segmented in any manner. The original 3PC algorithm assumes a fail-stop model, where processes fail by crashing and crashes can be accurately detected, and does not work with network partitions or asynchronous communication.

Keidar and Dolev's E3PC[2] algorithm eliminates this disadvantage.

The protocol requires at least 3 round trips to complete, needing a minimum of 3 round trip times (RTTs). This is potentially a long latency to complete each transaction.

7.4.4 References

[1] Skeen, Dale; Stonebraker, M. (May 1983). "A Formal Model of Crash Recovery in a Distributed System". *IEEE Transactions on Software Engineering* **9** (3): 219–228. doi:10.1109/TSE.1983.236608.

[2] Keidar, Idit; Danny Dolev (December 1998). "Increasing the Resilience of Distributed and Replicated Database Systems". *Journal of Computer and System Sciences (JCSS)* **57** (3): 309–324. doi:10.1006/jcss.1998.1566.

7.4.5 See also

- Two-phase commit protocol

Chapter 8

Scaling

8.1 Scalability

Scalability is the capability of a system, network, or process to handle a growing amount of work, or its potential to be enlarged in order to accommodate that growth.[1] For example, it can refer to the capability of a system to increase its total output under an increased load when resources (typically hardware) are added. An analogous meaning is implied when the word is used in an economic context, where scalability of a company implies that the underlying business model offers the potential for economic growth within the company.

Scalability, as a property of systems, is generally difficult to define[2] and in any particular case it is necessary to define the specific requirements for scalability on those dimensions that are deemed important. It is a highly significant issue in electronics systems, databases, routers, and networking. A system whose performance improves after adding hardware, proportionally to the capacity added, is said to be a ***scalable system***.

An algorithm, design, networking protocol, program, or other system is said to *scale* if it is suitably efficient and practical when applied to large situations (e.g. a large input data set, a large number of outputs or users, or a large number of participating nodes in the case of a distributed system). If the design or system fails when a quantity increases, it *does not scale*. In practice, if there are a large number of things (n) that affect scaling, then resource requirements (for example, algorithmic time-complexity) must grow less than n^2 as n increases. An example is a search engine, that must scale not only for the number of users, but for the number of objects it indexes. Scalability refers to the ability of a site to increase in size as demand warrants.[3]

The concept of scalability is desirable in technology as well as business settings. The base concept is consistent – the ability for a business or technology to accept increased volume without impacting the contribution margin (= revenue − variable costs). For example, a given piece of equipment may have a capacity for 1–1000 users, while beyond 1000 users additional equipment is needed or performance will decline (variable costs will increase and reduce contribution margin).

8.1.1 Measures

Scalability can be measured in various dimensions, such as:

- *Administrative scalability*: The ability for an increasing number of organizations or users to easily share a single distributed system.
- *Functional scalability*: The ability to enhance the system by adding new functionality at minimal effort.
- *Geographic scalability*: The ability to maintain performance, usefulness, or usability regardless of expansion from concentration in a local area to a more distributed geographic pattern.
- *Load scalability*: The ability for a distributed system to easily expand and contract its resource pool to accommodate heavier or lighter loads or number of inputs. Alternatively, the ease with which a system or component can be modified, added, or removed, to accommodate changing load.
- *Generation scalability* refers to the ability of a system to scale up by using new generations of components. Thereby, *heterogeneous scalability* is the ability to use the components from different vendors.[4]

8.1.2 Examples

- A routing protocol is considered scalable with respect to network size, if the size of the necessary routing table on each node grows as O(log *N*), where *N* is the number of nodes in the network.
- A scalable online transaction processing system or database management system is one that can be upgraded to process more transactions by adding new processors, devices and storage, and which can be upgraded easily and transparently without shutting it down.
- Some early peer-to-peer (P2P) implementations of Gnutella had scaling issues. Each node query

flooded its requests to all peers. The demand on each peer would increase in proportion to the total number of peers, quickly overrunning the peers' limited capacity. Other P2P systems like BitTorrent scale well because the demand on each peer is independent of the total number of peers. There is no centralized bottleneck, so the system may expand indefinitely without the addition of supporting resources (other than the peers themselves).

- The distributed nature of the Domain Name System allows it to work efficiently even when all hosts on the worldwide Internet are served, so it is said to "scale well".

8.1.3 Horizontal and vertical scaling

Methods of adding more resources for a particular application fall into two broad categories: horizontal and vertical scaling.[5]

- To **scale horizontally** (or **scale out**) means to add more nodes to a system, such as adding a new computer to a distributed software application. An example might involve scaling out from one Web server system to three. As computer prices have dropped and performance continues to increase, high-performance computing applications such as seismic analysis and biotechnology workloads have adopted low-cost "commodity" systems for tasks that once would have required supercomputers. System architects may configure hundreds of small computers in a cluster to obtain aggregate computing power that often exceeds that of computers based on a single traditional processor. The development of high-performance interconnects such as Gigabit Ethernet, InfiniBand and Myrinet further fueled this model. Such growth has led to demand for software that allows efficient management and maintenance of multiple nodes, as well as hardware such as shared data storage with much higher I/O performance. *Size scalability* is the maximum number of processors that a system can accommodate.[4]

- To **scale vertically** (or **scale up**) means to add resources to a single node in a system, typically involving the addition of CPUs or memory to a single computer. Such vertical scaling of existing systems also enables them to use virtualization technology more effectively, as it provides more resources for the hosted set of operating system and application modules to share. Taking advantage of such resources can also be called "scaling up", such as expanding the number of Apache daemon processes currently running. *Application scalability* refers to the improved performance of running applications on a scaled-up version of the system.[4]

There are tradeoffs between the two models. Larger numbers of computers means increased management complexity, as well as a more complex programming model and issues such as throughput and latency between nodes; also, some applications do not lend themselves to a distributed computing model. In the past, the price difference between the two models has favored "scale up" computing for those applications that fit its paradigm, but recent advances in virtualization technology have blurred that advantage, since deploying a new virtual system over a hypervisor (where possible) is almost always less expensive than actually buying and installing a real one. Configuring an existing idle system has always been less expensive than buying, installing, and configuring a new one, regardless of the model.

8.1.4 Database scalability

A number of different approaches enable databases to grow to very large size while supporting an ever-increasing rate of transactions per second. Not to be discounted, of course, is the rapid pace of hardware advances in both the speed and capacity of mass storage devices, as well as similar advances in CPU and networking speed.

One technique supported by most of the major database management system (DBMS) products is the partitioning of large tables, based on ranges of values in a key field. In this manner, the database can be *scaled out* across a cluster of separate database servers. Also, with the advent of 64-bit microprocessors, multi-core CPUs, and large SMP multiprocessors, DBMS vendors have been at the forefront of supporting multi-threaded implementations that substantially *scale up* transaction processing capacity.

Network-attached storage (NAS) and Storage area networks (SANs) coupled with fast local area networks and Fibre Channel technology enable still larger, more loosely coupled configurations of databases and distributed computing power. The widely supported X/Open XA standard employs a global transaction monitor to coordinate distributed transactions among semi-autonomous XA-compliant database resources. Oracle RAC uses a different model to achieve scalability, based on a "shared-everything" architecture that relies upon high-speed connections between servers.

While DBMS vendors debate the relative merits of their favored designs, some companies and researchers question the inherent limitations of relational database management systems. GigaSpaces, for example, contends that an entirely different model of distributed data access and transaction processing, Space based architecture, is required to achieve the highest performance and scalability. On the other hand, Base One makes the case for extreme scalability without departing from mainstream relational database technology.[6] For specialized applications, NoSQL architectures such as Google's BigTable

can further enhance scalability. Google's massively distributed Spanner technology, positioned as a successor to BigTable, supports general-purpose database transactions and provides a more conventional SQL-based query language.[7]

8.1.5 Strong versus eventual consistency (storage)

In the context of scale-out data storage, scalability is defined as the maximum storage cluster size which guarantees full data consistency, meaning there is only ever one valid version of stored data in the whole cluster, independently from the number of redundant physical data copies. Clusters which provide "lazy" redundancy by updating copies in an asynchronous fashion are called 'eventually consistent'. This type of scale-out design is suitable when availability and responsiveness are rated higher than consistency, which is true for many web file hosting services or web caches (*if you want the latest version, wait some seconds for it to propagate*). For all classical transaction-oriented applications, this design should be avoided.[8]

Many open source and even commercial scale-out storage clusters, especially those built on top of standard PC hardware and networks, provide eventual consistency only. Idem some NoSQL databases like CouchDB and others mentioned above. Write operations invalidate other copies, but often don't wait for their acknowledgements. Read operations typically don't check every redundant copy prior to answering, potentially missing the preceding write operation. The large amount of metadata signal traffic would require specialized hardware and short distances to be handled with acceptable performance (i.e. act like a non-clustered storage device or database).

Whenever strong data consistency is expected, look for these indicators:

- the use of InfiniBand, Fibrechannel or similar low-latency networks to avoid performance degradation with increasing cluster size and number of redundant copies.
- short cable lengths and limited physical extent, avoiding signal runtime performance degradation.
- majority / quorum mechanisms to guarantee data consistency whenever parts of the cluster become inaccessible.

Indicators for eventually consistent designs (not suitable for transactional applications!) are:

- write performance increases linearly with the number of connected devices in the cluster.
- while the storage cluster is partitioned, all parts remain responsive. There is a risk of conflicting updates.

8.1.6 Performance tuning versus hardware scalability

It is often advised to focus system design on hardware scalability rather than on capacity. It is typically cheaper to add a new node to a system in order to achieve improved performance than to partake in performance tuning to improve the capacity that each node can handle. But this approach can have diminishing returns (as discussed in performance engineering). For example: suppose 70% of a program can be sped up if parallelized and run on multiple CPUs instead of one. If α is the fraction of a calculation that is sequential, and $1-\alpha$ is the fraction that can be parallelized, the maximum speedup that can be achieved by using P processors is given according to Amdahl's Law:

$\frac{1}{\alpha+\frac{1-\alpha}{P}}$.

Substituting the value for this example, using 4 processors we get

$\frac{1}{0.3+\frac{1-0.3}{4}} = 2.105$.

If we double the compute power to 8 processors we get

$\frac{1}{0.3+\frac{1-0.3}{8}} = 2.581$.

Doubling the processing power has only improved the speedup by roughly one-fifth. If the whole problem was parallelizable, we would, of course, expect the speed up to double also. Therefore, throwing in more hardware is not necessarily the optimal approach.

8.1.7 Weak versus strong scaling

In the context of high performance computing there are two common notions of scalability:

- The first is *strong scaling*, which is defined as how the solution time varies with the number of processors for a fixed *total* problem size.
- The second is *weak scaling*, which is defined as how the solution time varies with the number of processors for a fixed problem size *per processor*.[9]

8.1.8 See also

- Asymptotic complexity
- Computational complexity theory
- Data Defined Storage
- Extensibility

- Gustafson's law
- List of system quality attributes
- Load balancing (computing)
- Lock (computer science)
- NoSQL
- Parallel computing
- Scalable Video Coding (SVC)
- Similitude (model)

8.1.9 References

[1] Bondi, André B. (2000). *Characteristics of scalability and their impact on performance*. Proceedings of the second international workshop on Software and performance - WOSP '00. p. 195. doi:10.1145/350391.350432. ISBN 158113195X.

[2] See for instance, Hill, Mark D. (1990). "What is scalability?". *ACM SIGARCH Computer Architecture News* **18** (4): 18. doi:10.1145/121973.121975. and Duboc, Leticia; Rosenblum, David S.; Wicks, Tony (2006). *A framework for modelling and analysis of software systems scalability*. Proceeding of the 28th international conference on Software engineering - ICSE '06. p. 949. doi:10.1145/1134285.1134460. ISBN 1595933751.

[3] Laudon, Kenneth Craig; Traver, Carol Guercio (2008). *E-commerce: Business, Technology, Society*. Pearson Prentice Hall/Pearson Education. ISBN 9780136006459.

[4] Hesham El-Rewini and Mostafa Abd-El-Barr (Apr 2005). *Advanced Computer Architecture and Parallel Processing*. John Wiley & Son. p. 66. ISBN 978-0-471-47839-3. Retrieved Oct 2013.

[5] Michael, Maged; Moreira, Jose E.; Shiloach, Doron; Wisniewski, Robert W. (March 26, 2007). *Scale-up x Scale-out: A Case Study using Nutch/Lucene*. 2007 IEEE International Parallel and Distributed Processing Symposium. p. 1. doi:10.1109/IPDPS.2007.370631. ISBN 1-4244-0909-8.

[6] Base One (2007). "Database Scalability - Dispelling myths about the limits of database-centric architecture". Retrieved May 23, 2007.

[7] "Spanner: Google's Globally-Distributed Database" (PDF). OSDI'12 Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation. 2012. pp. 251–264. ISBN 978-1-931971-96-6. Retrieved September 30, 2012.

[8] "Eventual consistency by Werner Vogels".

[9] "CSE - CSE - The Weak Scaling of DL_POLY 3". Stfc.ac.uk. Retrieved 2014-03-08.

8.1.10 External links

- Architecture of a Highly Scalable NIO-Based Server - an article about writing scalable server in Java (java.net).
- Links to diverse learning resources - page curated by the memcached project.
- Scalable Definition - by The Linux Information Project (LINFO)
- Scale in Distributed Systems B. Clifford Neumann, In: *Readings in Distributed Computing Systems*, IEEE Computer Society Press, 1994

8.2 Shard (database architecture)

A **database shard** is a horizontal partition of data in a database or search engine. Each individual partition is referred to as a **shard** or **database shard**. Each shard is held on a separate database server instance, to spread load.

Some data within a database remains present in all shards,[notes 1] but some only appears in a single shard. Each shard (or server) acts as the *single* source for this subset of data.[1]

8.2.1 Database architecture

Horizontal partitioning is a database design principle whereby *rows* of a database table are held separately, rather than being split into columns (which is what normalization and vertical partitioning do, to differing extents). Each partition forms part of a **shard**, which may in turn be located on a separate database server or physical location.

There are numerous advantages to the horizontal partitioning approach. Since the tables are divided and distributed into multiple servers, the total number of rows in each table in each database is reduced. This reduces index size, which generally improves search performance. A database shard can be placed on separate hardware, and multiple shards can be placed on multiple machines. This enables a distribution of the database over a large number of machines, which means that the load can be spread out over multiple machines, greatly improving performance. In addition, if the database shard is based on some real-world segmentation of the data (e.g., European customers v. American customers) then it may be possible to infer the appropriate shard membership easily and automatically, and query only the relevant shard.[2] Disadvantages include :

- A heavier reliance on the interconnect between servers

- Increased latency when querying, especially where more than one shard must be searched.
    - Data or indexes are often only sharded one way, so that some searches are optimal, and others are slow or impossible.
- Issues of consistency and durability due to the more complex failure modes of a set of servers, which often result in systems making no guarantees about cross-shard consistency or durability.

In practice, sharding is complex. Although it has been done for a long time by hand-coding (especially where rows have an obvious grouping, as per the example above), this is often inflexible. There is a desire to support sharding automatically, both in terms of adding code support for it, and for identifying candidates to be sharded separately. Consistent hashing is one form of automatic sharding to spread large loads across multiple smaller services and servers.[3]

Where distributed computing is used to separate load between multiple servers (either for performance or reliability reasons), a shard approach may also be useful.

8.2.2 Shards compared to horizontal partitioning

Horizontal partitioning splits one or more tables by row, usually within a *single* instance of a schema and a database server. It may offer an advantage by reducing index size (and thus search effort) provided that there is some obvious, robust, implicit way to identify in which table a particular row will be found, without first needing to search the index, e.g., the classic example of the 'CustomersEast' and 'CustomersWest' tables, where their zip code already indicates where they will be found.

Sharding goes beyond this: it partitions the problematic table(s) in the same way, but it does this across potentially *multiple* instances of the schema. The obvious advantage would be that search load for the large partitioned table can now be split across multiple servers (logical or physical), not just multiple indexes on the same logical server.

Splitting shards across multiple isolated instances requires more than simple horizontal partitioning. The hoped-for gains in efficiency would be lost, if querying the database required *both* instances to be queried, just to retrieve a simple dimension table. Beyond partitioning, sharding thus splits large partitionable tables across the servers, while smaller tables are replicated as complete units.

This is also why sharding is related to a shared nothing architecture—once sharded, each shard can live in a totally separate logical schema instance / physical database server / data center / continent. There is no ongoing need to retain shared access (from between shards) to the other unpartitioned tables in other shards.

This makes replication across multiple servers easy (simple horizontal partitioning does not). It is also useful for worldwide distribution of applications, where communications links between data centers would otherwise be a bottleneck.

There is also a requirement for some notification and replication mechanism between schema instances, so that the unpartitioned tables remain as closely synchronized as the application demands. This is a complex choice in the architecture of sharded systems: approaches range from making these effectively read-only (updates are rare and batched), to dynamically replicated tables (at the cost of reducing some of the distribution benefits of sharding) and many options in between.

8.2.3 Support for shards

Apache HBase HBase supports automatic sharding.[4]

Azure SQL Database Elastic Database tools Elastic Database tools enables the data-tier of an application to scale out and in via industry-standard sharding practices[5]

CUBRID CUBRID supports sharding from version 9.0

dbShards CodeFutures dbShards is a product dedicated to database shards.[6]

Elasticsearch Elasticsearch enterprise search server provides sharding capabilities.[7]

eXtreme Scale eXtreme Scale is a cross-process in-memory key/value datastore (a variety of NoSQL datastore). It uses sharding to achieve scalability across processes for both data and MapReduce-style parallel processing.[8]

Hibernate ORM Hibernate Shards provides support for shards, although there has been little activity since 2007.[9][10]

IBM Informix IBM has supported sharding in Informix since version 12.1 xC1 as part of the MACH11 technology. Informix 12.10 xC2 added full compatibility with MongoDB drivers, allowing the mix of regular relational tables with NoSQL collections, while still supporting sharding, failover and ACID properties.[11][12]

MonetDB the open-source column-store MonetDB supports read-only sharding as its July 2015 release.[13]

MongoDB MongoDB supports sharding from version 1.6

MySQL Cluster Auto-Sharding: Database is automatically and transparently partitioned across low-cost commodity nodes, allowing scale-out of read and write queries, without requiring changes to the application.[14]

MySQL Fabric (part of MySQL utilities) includes support for sharding.[15]

Oracle NoSQL Database

Oracle NoSQL Database supports automatic sharding and elastic, online expansion of the cluster (adding more shards).

OrientDB OrientDB supports sharding from version 1.7

pg_shard a sharding extension for PostgreSQL. It shards and replicates PostgreSQL tables for horizontal scale and for high availability. The extension also seamlessly distributes SQL statements without requiring any changes to applications.[16]

Plugin for Grails Grails supports sharding using the Grails Sharding Plugin.[17]

Ruby ActiveRecord Octopus works as a database sharding and replication extension for the ActiveRecord ORM.

ScaleBase's Data Traffic Manager ScaleBase's Data Traffic Manager is a software product dedicated to automating MySQL database sharding without requiring changes to applications.[18]

Shard Query Open Source parallel query engine for MySQL.[19]

Solr Search Server Solr enterprise search server provides sharding capabilities.[20]

Spanner Spanner, Google's global-scale distributed database, shards data across multiple Paxos state machines to scale to "millions of machines across hundreds of datacenters and trillions of database rows".[21]

SQLAlchemy ORM SQLAlchemy is a data-mapper for the Python programming language that provides sharding capabilities.[22]

Teradata

The DWH of Teradata was the first massive parallel database.

8.2.4 Disadvantages of sharding

Sharding a database table before it has been optimized locally causes premature complexity. Sharding should be used only when all other options for optimization are inadequate. The introduced complexity of database sharding causes the following potential problems:

- **Increased complexity of SQL** - Increased bugs because the developers have to write more complicated SQL to handle sharding logic.
- **Sharding introduces complexity** - The sharding software that partitions, balances, coordinates, and ensures integrity can fail.
- **Single point of failure** - Corruption of one shard due to network/hardware/systems problems causes failure of the entire table.
- **Failover servers more complex** - Failover servers must themselves have copies of the fleets of database shards.
- **Backups more complex** - Database backups of the individual shards must be coordinated with the backups of the other shards.
- **Operational complexity added** - Adding/removing indexes, adding/deleting columns, modifying the schema becomes much more difficult.

These historical complications of do-it-yourself sharding are now being addressed by independent software vendors who provide autosharding solutions.

8.2.5 Etymology

The word "shard" in a database context may have been introduced by the CCA's "System for Highly Available Replicated Data".[23] There has been speculation[24] that the term might be derived from the 1997 MMORPG Ultima Online, but the SHARD database system predates this by at least nine years.

However, the SHARD system appears[25] to have used its redundant hardware only for *replication* and not for horizontal partitioning. It is not known whether present-day use of the term "shard" is derived from the CCA system, but in any case it refers to a different use of redundant hardware in database systems.

8.2.6 See also

- Shared nothing architecture

8.2.7 References

[1] Typically 'supporting' data such as dimension tables

[1] Pramod J. Sadalage; Martin Fowler (2012), "4: Distribution Models", *NoSQL Distilled*, ISBN 0321826620

[2] Rahul Roy (July 28, 2008). "Shard - A Database Design".

[3] Ries, Eric. "Sharding for Startups".

[4] "Apache HBase Sharding".

[5] "Introducing Elastic Scale preview for Azure SQL Database".

[6] "dbShards product overview".

[7] "Index Shard Allocation".

[8] http://publib.boulder.ibm.com/infocenter/wxsinfo/v7r1/index.jsp?topic=%2Fcom.ibm.websphere.extremescale.over.doc%2

[9] "Hibernate Shards". 2007-02-08.

[10] "Hibernate Shards".

[11] "New Grid queries for Informix".

[12] "NoSQL support in Informix".

[13] "MonetDB July2015 Released". 31 August 2015.

[14] "MySQL Cluster Features & Benefits". 2012-11-23.

[15] "MySQL Fabric sharding quick start guide".

[16] "pg_shard PostgreSQL extension".

[17] "Grails Sharding Plugin".

[18] "ScaleBase's Data Traffic Manager product architecture overview".

[19] "Shard Query".

[20] "Distributed Search".

[21] Corbett, James C; Dean, Jeffrey; Epstein, Michael; Fikes, Andrew; Frost, Christopher; Furman, JJ; Ghemawat, Sanjay; Heiser, Christopher; Hochschild, Peter; Hsieh, Wilson; Kanthak, Sebastian; Kogan, Eugene; Li, Hongyi; Lloyd, Alexander; Melnik, Sergey; Mwaura, David; Nagle, David; Quinlan, Sean; Rao, Rajesh; Rolig, Lindsay; Saito, Yasushi; Szymaniak, Michal; Taylor, Christopher; Wang, Ruth; Woodford, Dale. "Spanner: Google's Globally-Distributed Database" (PDF). *Proceedings of OSDI 2012*. Google. Retrieved 24 February 2014. |first8= missing |last8= in Authors list (help)

[22] "Basic example of using the SQLAlchemy Sharding API.".

[23] Sarin, DeWitt & Rosenburg, *Overview of SHARD: A System for Highly Available Replicated Data*, Technical Report CCA-88-01, Computer Corporation of America, May 1988

[24] Koster, Raph (2009-01-08). "Database "sharding" came from UO?". *Raph Koster's Website*. Retrieved 2015-01-17.

[25] Sarin & Lynch, *Discarding Obsolete Information in a Replicated Database System*, IEEE Transactions on Software Engineering vol SE-13 no 1, January 1987

8.2.8 External links

- Informix JSON data sharding

8.3 Optimistic concurrency control

Optimistic concurrency control (**OCC**) is a concurrency control method applied to transactional systems such as relational database management systems and software transactional memory. OCC assumes that multiple transactions can frequently complete without interfering with each other. While running, transactions use data resources without acquiring locks on those resources. Before committing, each transaction verifies that no other transaction has modified the data it has read. If the check reveals conflicting modifications, the committing transaction rolls back and can be restarted.[1] Optimistic concurrency control was first proposed by H.T. Kung.[2]

OCC is generally used in environments with low data contention. When conflicts are rare, transactions can complete without the expense of managing locks and without having transactions wait for other transactions' locks to clear, leading to higher throughput than other concurrency control methods. However, if contention for data resources is frequent, the cost of repeatedly restarting transactions hurts performance significantly; it is commonly thought that other concurrency control methods have better performance under these conditions. However, locking-based ("pessimistic") methods also can deliver poor performance because locking can drastically limit effective concurrency even when deadlocks are avoided.

8.3.1 OCC phases

More specifically, OCC transactions involve these phases:

- **Begin**: Record a timestamp marking the transaction's beginning.
- **Modify**: Read database values, and tentatively write changes.
- **Validate**: Check whether other transactions have modified data that this transaction has used (read

or written). This includes transactions that completed after this transaction's start time, and optionally, transactions that are still active at validation time.

- **Commit/Rollback**: If there is no conflict, make all changes take effect. If there is a conflict, resolve it, typically by aborting the transaction, although other resolution schemes are possible. Care must be taken to avoid a TOCTTOU bug, particularly if this phase and the previous one are not performed as a single atomic operation.

8.3.2 Web usage

The stateless nature of HTTP makes locking infeasible for web user interfaces. It's common for a user to start editing a record, then leave without following a "cancel" or "logout" link. If locking is used, other users who attempt to edit the same record must wait until the first user's lock times out.

HTTP does provide a form of built-in OCC: The GET method returns an ETag for a resource and subsequent PUTs use the ETag value in the If-Match headers; while the first PUT will succeed, the second will not, as the value in If-Match is based on the first version of the resource.[3]

Some database management systems offer OCC natively - without requiring special application code. For others, the application can implement an OCC layer outside of the database, and avoid waiting or silently overwriting records. In such cases, the form includes a hidden field with the record's original content, a timestamp, a sequence number, or an opaque token. On submit, this is compared against the database. If it differs, the conflict resolution algorithm is invoked.

Examples

- MediaWiki's edit pages use OCC.[4]
- Bugzilla uses OCC; edit conflicts are called "mid-air collisions".[5]
- The Ruby on Rails framework has an API for OCC.[6]
- The Grails framework uses OCC in its default conventions.[7]
- The GT.M database engine uses OCC for managing transactions[8] (even single updates are treated as mini-transactions).
- Microsoft's Entity Framework (including Code-First) has built-in support for OCC based on a binary timestamp value.[9]
- Mimer SQL is a DBMS that only implements optimistic concurrency control.[10]
- Google App Engine data store uses OCC.[11]
- The Elasticsearch search engine supports OCC via the version attribute.[12]
- The MonetDB column-oriented database management system's transaction management scheme is based on OCC.[13]
- Most implementations of software transactional memory use optimistic locking.
- Redis provides OCC through WATCH command.[14]

8.3.3 See also

- Server Message Block#Opportunistic locking

8.3.4 References

[1] Johnson, Rohit (2003). "Common Data Access Issues". *Expert One-on-One J2EE Design and Development*. Wrox Press. ISBN 0-7645-4385-7.

[2] Kung, H.T. (1981). "On Optimistic Methods for Concurrency Control". ACM Transactions on Database Systems.

[3] "Editing the Web - Detecting the Lost Update Problem Using Unreserved Checkout". *W3C Note*. 10 May 1999.

[4] Help:Edit conflict

[5] "Bugzilla: FAQ: Administrative Questions". *Mozilla Wiki*. 11 April 2012.

[6] "Module ActiveRecord::Locking". *Rails Framework Documentation*.

[7] "Object Relational Mapping (GORM)". *Grails Framework Documentation*.

[8] "Transaction Processing". *GT.M Programmers Guide UNIX Edition*.

[9] "Tip 19 – How to use Optimistic Concurrency with the Entity Framework". *MSDN Blogs*. 19 May 2009.

- Most revision control systems support the "merge" model for concurrency, which is OCC.

[10] "Transaction Concurrency - Optimistic Concurrency Control". *Mimer Developers - Features*. 26 February 2010.

[11] "The Datastore". *What Is Google App Engine?*. 27 August 2010.

[12] "Elasticsearch - Guide - Index API". *Elasticsearch Guide*. 22 March 2012.

[13] "Transactions - MonetDB". 16 January 2013.

[14] "Transactions in Redis".

8.3.5 External links

- Kung, H. T.; John T. Robinson (June 1981). "On optimistic methods for concurrency control". *ACM Transactions on Database Systems* **6** (2): 213–226. doi:10.1145/319566.319567.
- Enterprise JavaBeans, 3.0, By Bill Burke, Richard Monson-Haefel, Chapter 16. Transactions, Section 16.3.5. Optimistic Locking, Publisher: O'Reilly, Pub Date: May 16, 2006,Print ISBN 0-596-00978-X,
- Hollmann, Andreas (May 2009). "Multi-Isolation: Virtues and Limitations" (PDF). *Multi-Isolation (what is between pessimistic and optimistic locking)*. 01069 Gutzkovstr. 30/F301.2, Dresden: HappyGuys Software GbR. p. 8. Retrieved 2013-05-16.

8.4 Partition (database)

A **partition** is a division of a logical database or its constituent elements into distinct independent parts. Database partitioning is normally done for manageability, performance or availability reasons.

8.4.1 Benefits of multiple partitions

A popular and favourable application of partitioning is in a distributed database management system. Each partition may be spread over multiple nodes, and users at the node can perform local transactions on the partition. This increases performance for sites that have regular transactions involving certain views of data, whilst maintaining availability and security.

8.4.2 Partitioning criteria

Current high end relational database management systems provide for different criteria to split the database. They take a *partitioning key* and assign a partition based on certain criteria. Common criteria are:

Range partitioning Selects a partition by determining if the partitioning key is inside a certain range. An example could be a partition for all rows where the column zipcode has a value between 70000 and 79999.

List partitioning A partition is assigned a list of values. If the partitioning key has one of these values, the partition is chosen. For example all rows where the column Country is either Iceland, Norway, Sweden, Finland or Denmark could build a partition for the Nordic countries.

Hash partitioning The value of a hash function determines membership in a partition. Assuming there are four partitions, the hash function could return a value from 0 to 3.

Composite partitioning allows for certain combinations of the above partitioning schemes, by for example first applying a range partitioning and then a hash partitioning. Consistent hashing could be considered a composite of hash and list partitioning where the hash reduces the key space to a size that can be listed.

8.4.3 Partitioning methods

The partitioning can be done by either building separate smaller databases (each with its own tables, indices, and transaction logs), or by splitting selected elements, for example just one table.

Horizontal partitioning (also see ***shard***) involves putting different rows into different tables. Perhaps customers with ZIP codes less than 50000 are stored in CustomersEast, while customers with ZIP codes greater than or equal to 50000 are stored in CustomersWest. The two partition tables are then CustomersEast and CustomersWest, while a view with a union might be created over both of them to provide a complete view of all customers.

Vertical partitioning involves creating tables with fewer columns and using additional tables to store the remaining columns.[1] Normalization also involves this splitting of columns across tables, but vertical partitioning goes beyond that and partitions columns even when already normalized. Different physical storage might be used to realize vertical partitioning as well; storing infrequently used or very wide columns on a different device, for example, is a method of vertical partitioning. Done explicitly or implicitly, this type of partitioning is called "row splitting" (the row is split by its columns). A common form of vertical partitioning is to split dynamic data (slow to find) from static data (fast to find) in a table where the dynamic data is not used as often as the static. Creating a view across the two newly created tables restores the original table with a performance penalty, however performance will increase when accessing the static data e.g. for statistical analysis.

8.4.4 See also

- Shard (database architecture)

8.4.5 References

[1] Vertical Partitioning Algorithms for Database Design, by Shamkant Navathe, Stefano Ceri, Gio Wiederhold, and Jinglie Dou, Stanford University 1984

8.4.6 External links

- IBM DB2 partitioning
- MySQL partitioning
- Oracle partitioning
- SQL Server partitions
- PostgreSQL partitioning
- Sybase ASE 15.0 partitioning
- MongoDB partitioning
- ScimoreDB partitioning
- VoltDB partitioning

8.5 Distributed transaction

A **distributed transaction** is a database transaction in which two or more network hosts are involved. Usually, hosts provide **transactional resources**, while the **transaction manager** is responsible for creating and managing a global transaction that encompasses all operations against such resources. Distributed transactions, as any other transactions, must have all four ACID (atomicity, consistency, isolation, durability) properties, where atomicity guarantees all-or-nothing outcomes for the unit of work (operations bundle).

Open Group, a vendor consortium, proposed the X/Open Distributed Transaction Processing (DTP) Model (X/Open XA), which became a de facto standard for behavior of transaction model components.

Databases are common transactional resources and, often, transactions span a couple of such databases. In this case, a distributed transaction can be seen as a database transaction that must be synchronized (or provide ACID properties) among multiple participating databases which are distributed among different physical locations. The isolation property (the I of ACID) poses a special challenge for multi database transactions, since the (global) serializability property could be violated, even if each database provides it (see also global serializability). In practice most commercial database systems use strong strict two phase locking (SS2PL) for concurrency control, which ensures global serializability, if all the participating databases employ it. (see also commitment ordering for multidatabases.)

A common algorithm for ensuring correct completion of a distributed transaction is the two-phase commit (2PC). This algorithm is usually applied for updates able to commit in a short period of time, ranging from couple of milliseconds to couple of minutes.

There are also long-lived distributed transactions, for example a transaction to book a trip, which consists of booking a flight, a rental car and a hotel. Since booking the flight might take up to a day to get a confirmation, two-phase commit is not applicable here, it will lock the resources for this long. In this case more sophisticated techniques that involve multiple undo levels are used. The way you can undo the hotel booking by calling a desk and cancelling the reservation, a system can be designed to undo certain operations (unless they are irreversibly finished).

In practice, long-lived distributed transactions are implemented in systems based on Web Services. Usually these transactions utilize principles of Compensating transactions, Optimism and Isolation Without Locking. X/Open standard does not cover long-lived DTP.

Several modern technologies, including Enterprise Java Beans (EJBs) and Microsoft Transaction Server (MTS) fully support distributed transaction standards.

8.5.1 See also

Java Transaction API (JTA)

8.5.2 References

- "Web-Services Transactions". *Web-Services Transactions*. Retrieved May 2, 2005.
- "Nuts And Bolts Of Transaction Processing". *Article about Transaction Management*. Retrieved May 3, 2005.
- "A Detailed Comparison of Enterprise JavaBeans (EJB) & The Microsoft Transaction Server (MTS) Models".

8.5.3 Further reading

- Gerhard Weikum, Gottfried Vossen, *Transactional information systems: theory, algorithms, and the practice of concurrency control and recovery*, Morgan Kaufmann, 2002, ISBN 1-55860-508-8

Chapter 9

Examples

9.1 Redis

This article is about Redis software. For Redis people, see Romani people.

Redis is a data structure server. It is open-source, networked, in-memory, and stores keys with optional durability. The development of Redis has been sponsored by Redis Labs since June 2015.[3] Before that, it was sponsored by Pivotal Software[4] and by VMware.[5][6] According to the monthly ranking by DB-Engines.com, Redis is the most popular key-value database.[7] Redis has also been ranked the #1 NoSQL (and #4 database) in User Satisfaction and Market Presence based on user reviews,[8] the most popular NoSQL database in containers,[9] and the #2 NoSQL among Top 50 Developer Tools & Services.[10] The name Redis means REmote DIctionary Server.[11]

9.1.1 Supported languages

Many languages have Redis bindings, including:[12] ActionScript, C, C++, C#, Chicken Scheme, Clojure, Common Lisp, D, Dart, Erlang, Go, Haskell, Haxe, Io, Java, JavaScript (Node.js), Julia, Lua, Objective-C, Perl, PHP, Pure Data, Python, R,[13] Racket, Ruby, Rust, Scala, Smalltalk and Tcl.

9.1.2 Data types

Redis maps keys to types of values. A key difference between Redis and other structured storage systems is that Redis supports not only strings, but also abstract data types:

- Lists of strings
- Sets of strings (collections of non-repeating unsorted elements)
- Sorted sets of strings (collections of non-repeating elements ordered by a floating-point number called score)
- Hash tables where keys and values are strings
- HyperLogLogs used for approximated set cardinality size estimation.

The type of a value determines what operations (called commands) are available for the value itself. Redis supports high-level, atomic, server-side operations like intersection, union, and difference between sets and sorting of lists, sets and sorted sets.

9.1.3 Persistence

Redis typically holds the whole dataset in memory. Versions up to 2.4 could be configured to use what they refer to as virtual memory[14] in which some of the dataset is stored on disk, but this feature is deprecated. Persistence is now reached in two different ways: one is called snapshotting, and is a semi-persistent durability mode where the dataset is asynchronously transferred from memory to disk from time to time, written in RDB dump format. Since version 1.1 the safer alternative is AOF, an append-only file (a journal) that is written as operations modifying the dataset in memory are processed. Redis is able to rewrite the append-only file in the background in order to avoid an indefinite growth of the journal.

By default, Redis syncs data to the disk at least every 2 seconds, with more or less robust options available if needed. In the case of a complete system failure on default settings, only a few seconds of data would be lost.[15]

9.1.4 Replication

Redis supports master-slave replication. Data from any Redis server can replicate to any number of slaves. A slave may be a master to another slave. This allows Redis to implement a single-rooted replication tree. Redis slaves can be configured to accept writes, permitting intentional and unintentional inconsistency between instances. The Publish/Subscribe feature is fully implemented, so a client of a slave may SUBSCRIBE to a channel and receive a full feed of messages PUBLISHed to

the master, anywhere up the replication tree. Replication is useful for read (but not write) scalability or data redundancy.[16]

9.1.5 Performance

When the durability of data is not needed, the in-memory nature of Redis allows it to perform extremely well compared to database systems that write every change to disk before considering a transaction committed.[11] There is no notable speed difference between write and read operations. Redis operates as a single process and single-threaded. Therefore a single Redis instance cannot utilize parallel execution of tasks e.g. stored procedures (Lua scripts).

9.1.6 Clustering

The Redis project has a cluster specification,[17] but the cluster feature is currently in Beta stage.[18] According to a news post by Redis creator Sanfilippo, the first production version of Redis cluster (planned for beta release at end of 2013),[19] will support automatic partitioning of the key space and hot resharding, but will support only single key operations.[20] In the future Redis Cluster is planned to support up to 1000 nodes, fault tolerance with heartbeat and failure detection, incremental versioning ("epochs") to prevent conflicts, slave election and promotion to master, and publish/subscribe between all cluster nodes.[17][18][21]

9.1.7 See also

- NoSQL

9.1.8 References

[1] An interview with Salvatore Sanfilippo, creator of Redis, working out of Sicily, January 4, 2011, by Stefano Bernardi, EU-Startups

[2] Pivotal People—Salvatore Sanfilippo, Inventor of Redis, July 17, 2013, By Stacey Schneider, Pivotal P.O.V.

[3]

[4] Redis Sponsors – Redis

[5] VMware: the new Redis home

[6] VMWare: The Console: VMware hires key developer for Redis

[7] DB-Engines Ranking of Key-value Stores

[8] Best NoSQL Databases: Fall 2015 Report from G2 Crowd

[9] The Current State of Container Usage

[10] Top 50 Developer Tools and Services of 2014

[11] "FAQ, Redis".

[12] Redis language bindings

[13] CRAN – Package rredis

[14] Redis documentation "Virtual Memory", *redis.io*, accessed January 18, 2011.

[15] Redis persistence demystified, 26 March 2012, antirez weblog

[16] ReplicationHowto – redis – A persistent key-value database with built-in net interface written in ANSI-C for Posix systems – Google Project Hosting

[17] Redis Cluster Specification, Redis.io, Retrieved 2013-12-25.

[18] Redis Cluster Tutorial, Redis.io, Retrieved 2014-06-14.

[19] Redis Download Page, Redis.io, Retrieved 2013-12-25.

[20] News about Redis: 2.8 is shaping, I'm back on Cluster, Antirez Weblog - Salvatore Sanfilippo, Retrieved 2013-12-25.

[21] Redis Cluster - a Pragmatic Approach to Distribution, Redis.io, Retrieved 2013-12-25.

Notes

- Jeremy Zawodny, *Redis: Lightweight key/value Store That Goes the Extra Mile*, Linux Magazine, August 31, 2009
- Isabel Drost and Jan Lehnard (29 October 2009), Happenings: NoSQL Conference, Berlin, The H. Slides for the Redis presentation. Summary.
- Billy Newport (IBM): "Evolving the Key/Value Programming Model to a Higher Level" Qcon Conference 2009 San Francisco.
- A Mishra: "Install and configure Redis on Centos/Fedora server".

9.1.9 External links

- Official website
- redis on GitHub
- Redis Mailing List Archives
- Redis Articles Collection

9.2 MongoDB

MongoDB (from *humongous*) is a cross-platform document-oriented database. Classified as a NoSQL database, MongoDB eschews the traditional table-based relational database structure in favor of JSON-like documents with dynamic schemas (MongoDB calls the format BSON), making the integration of data in certain types of applications easier and faster. Released under a combination of the GNU Affero General Public License and the Apache License, MongoDB is free and open-source software.

First developed by the software company MongoDB Inc. in October 2007 as a component of a planned platform as a service product, the company shifted to an open source development model in 2009, with MongoDB offering commercial support and other services.[2] Since then, MongoDB has been adopted as backend software by a number of major websites and services, including Craigslist, eBay, and Foursquare among others. As of July 2015, MongoDB is the fourth most popular type of database management system, and the most popular for document stores.[3]

9.2.1 History

9.2.2 Main features

Some of the features include:[14]

Document-oriented

Instead of taking a business subject and breaking it up into multiple relational structures, MongoDB can store the business subject in the minimal number of documents. For example, instead of storing title and author information in two distinct relational structures, title, author, and other title-related information can all be stored in a single document called Book.[15]

Ad hoc queries

MongoDB supports search by field, range queries, regular expression searches. Queries can return specific fields of documents and also include user-defined JavaScript functions.

Indexing

Any field in a MongoDB document can be indexed (indices in MongoDB are conceptually similar to those in RDBMSes). Secondary indices are also available.

Replication

MongoDB provides high availability with replica sets.[16] A replica set consists of two or more copies of the data. Each replica set member may act in the role of primary or secondary replica at any time. The primary replica performs all writes and reads by default. Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically conducts an election process to determine which secondary should become the primary. Secondaries can also perform read operations, but the data is eventually consistent by default.

Load balancing

MongoDB scales horizontally using sharding.[17] The user chooses a shard key, which determines how the data in a collection will be distributed. The data is split into ranges (based on the shard key) and distributed across multiple shards. (A shard is a master with one or more slaves.) MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure. Automatic configuration is easy to deploy, and new machines can be added to a running database.

File storage

MongoDB can be used as a file system, taking advantage of load balancing and data replication features over multiple machines for storing files.

This function, called Grid File System,[18] is included with MongoDB drivers and available for development languages (see "Language Support" for a list of supported languages). MongoDB exposes functions for file manipulation and content to developers. GridFS is used, for example, in plugins for NGINX[19] and lighttpd.[20] Instead of storing a file in a single document, GridFS divides a file into parts, or chunks, and stores each of those chunks as a separate document.[21]

In a multi-machine MongoDB system, files can be distributed and copied multiple times between machines transparently, thus effectively creating a load-balanced and fault-tolerant system.

Aggregation

MapReduce can be used for batch processing of data and aggregation operations. The aggregation framework enables users to obtain the kind of results for which the SQL GROUP BY clause is used.

Server-side JavaScript execution

JavaScript can be used in queries, aggregation functions (such as MapReduce), and sent directly to the database to be executed.

Capped collections

MongoDB supports fixed-size collections called capped collections. This type of collection maintains insertion order and, once the specified size has been reached, behaves like a circular queue.

9.2.3 Criticisms

In some failure scenarios where an application can access two distinct MongoDB processes, but these processes cannot access each other, it is possible for MongoDB to return stale reads. In this scenario it is also possible for MongoDB to acknowledge writes that will be rolled back.[22]

Before version 2.2, concurrency control was implemented on a per-mongod basis. With version 2.2, concurrency control was implemented at the database level.[23] Since version 3.0,[24] pluggable storage engines were introduced, and each storage engine may implement concurrency control differently.[25] With MongoDB 3.0 concurrency control is implemented at the collection level for the MMAPv1 storage engine,[26] and at the document level with the WiredTiger storage engine.[27] With versions prior to 3.0, one approach to increase concurrency is to use sharding.[28] In some situations, reads and writes will yield their locks. If MongoDB predicts a page is unlikely to be in memory, operations will yield their lock while the pages load. The use of lock yielding expanded greatly in 2.2.[29]

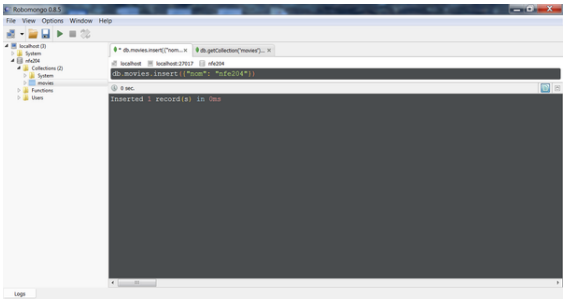
Another criticism is related to the limitations of MongoDB when used on 32-bit systems.[30] In some cases, this was due to inherent memory limitations.[31] MongoDB recommends 64-bit systems and that users provide sufficient RAM for their working set. Compose, a provider of managed MongoDB infrastructure, recommends a scaling checklist for large systems.[32]

Additionally, MongoDB does not support collation-based sorting and is limited to byte-wise comparison via memcmp,[33] which will not provide correct ordering for many non-English languages[34] when used with a Unicode encoding.

9.2.4 Architecture

Language support

MongoDB has official drivers for a variety of popular programming languages and development environments.[35] There are also a large number of unofficial or community-supported drivers for other programming languages and frameworks.[36]



Record insertion in MongoDB with Robomongo 0.8.5.

Management and graphical front-ends

Most administration is done from command line tools such as the mongo shell because MongoDB does not include a GUI-style administrative interface. There are third-party projects that offer user interfaces for administration and data viewing.[37]

Licensing and support

MongoDB is available for free under the GNU Affero General Public License.[38] The language drivers are available under an Apache License. In addition, MongoDB Inc. offers proprietary licenses for MongoDB.[2]

9.2.5 Performance

United Software Associates published a benchmark using Yahoo's Cloud Serving Benchmark as a basis of all the tests. MongoDB provides greater performance than Couchbase Server or Cassandra in all the tests they ran, in some cases by as much as 25x.[39]

Another benchmark for top NoSQL databases utilizing Amazon's Elastic Compute Cloud that was done by End Point arrived at opposite results, placing MongoDB last among the tested databases.[40]

9.2.6 Production deployments

Large-scale deployments of MongoDB are tracked by MongoDB Inc. Notable users of MongoDB include:

- Adobe: Adobe Experience Manager is intended to accelerate development of digital experiences that increase customer loyalty, engagement and demand. Adobe uses MongoDB to store petabytes of data in the large-scale content repositories underpinning the Experience Manager.[41]
- Amadeus IT Group uses MongoDB for its back-end software.[42]

- The Compact Muon Solenoid at CERN uses MongoDB as the primary back-end for the *Data Aggregation System* for the Large Hadron Collider.[43]
- Craigslist: With 80 million classified ads posted every month, Craigslist needs to archive billions of records in multiple formats, and must be able to query and report on these archives at runtime. Craigslist migrated from MySQL to MongoDB to support its active archive, with continuous availability mandated for regulatory compliance across 700 sites in 70 different countries.[44]
- eBay uses MongoDB in the search suggestion and the internal Cloud Manager *State Hub*.[45]
- FIFA (video game series): EA Sports FIFA is the world's best-selling sports video game franchise. To serve millions of players, EA's Spearhead development studio selected MongoDB[46] to store user data and game state. Auto-sharding makes it simple to scale MongoDB across EA's 250+ servers with no limits to growth as EA FIFA wins more fans.
- Foursquare deploys MongoDB on Amazon AWS to store venues and user check-ins into venues.[47]
- LinkedIn uses MongoDB as its backend DB.[48]
- McAfee: MongoDB powers McAfee Global Threat Intelligence (GTI), a cloud-based intelligence service that correlates data from millions of sensors around the globe. Billions of documents are stored and analyzed in MongoDB to deliver real-time threat intelligence to other McAfee end-client products.[49]
- MetLife uses MongoDB for "The Wall", a customer service application providing a "360-degree view" of MetLife customers.[50]
- SAP uses MongoDB in the SAP PaaS.[51]
- Shutterfly uses MongoDB for its photo platform. As of 2013, the photo platform stores 18 billion photos uploaded by Shutterfly's 7 million users.[52]
- Tuenti uses MongoDB as its backend DB.[53]
- Yandex: The largest search engine in Russia uses MongoDB to manage all user and metadata for its file sharing service. MongoDB has scaled[54] to support tens of billions of objects and TBs of data, growing at 10 million new file uploads per day.

9.2.7 See also

- NoSQL
- Server-side scripting
- MEAN, a solutions stack using MongoDB as the database
- HyperDex, a NoSQL database providing the MongoDB API with stronger consistency guarantees

9.2.8 References

[1] "Release Notes for MongoDB 3.0". MongoDB.

[2] "10gen embraces what it created, becomes MongoDB Inc.". *Gigaom*. Retrieved 27 August 2013.

[3] "Popularity ranking of database management systems". *db-engines.com*. Solid IT. Retrieved 2015-07-04.

[4] "Release Notes for MongoDB 1.2.x". Retrieved 2015-11-29.

[5] "Release Notes for MongoDB 1.4". Retrieved 2015-11-29.

[6] "Release Notes for MongoDB 1.6". Retrieved 2015-11-29.

[7] "Release Notes for MongoDB 1.8". Retrieved 2015-11-29.

[8] "Release Notes for MongoDB 2.0". Retrieved 2015-11-29.

[9] "Release Notes for MongoDB 2.2". Retrieved 2015-11-29.

[10] "Release Notes for MongoDB 2.4". Retrieved 2015-11-29.

[11] "Release Notes for MongoDB 2.6". Retrieved 2015-11-29.

[12] "Release Notes for MongoDB 3.0". Retrieved 2015-11-29.

[13] "Development Release Notes for 3.2 Release Candidate". Retrieved 2015-11-29.

[14] MongoDB. "MongoDB Developer Manual". MongoDB.

[15] Data Modeling for MongoDB

[16] MongoDB. "Introduction to Replication". MongoDB.

[17] MongoDB. "Introduction to Sharding". MongoDB.

[18] MongoDB. "GridFS article on MongoDB Developer's Manual". MongoDB.

[19] "NGINX plugin for MongoDB source code". *GitHub*.

[20] "lighttpd plugin for MongoDB source code". *Bitbucket*.

[21] Malick Md. "MongoDB overview". *Expertstown*.

[22] Kyle Kingsbury (2015-04-20). "Call me maybe: MongoDB stale reads". Retrieved 2015-07-04.

[23] "MongoDB Jira Ticket 4328". *jira.mongodb.org*.

[24] Eliot Horowitz (2015-01-22). "Renaming Our Upcoming Release to MongoDB 3.0". MongoDB. Retrieved 2015-02-23.

[25] "MongoDB 2.8 release". MongoDB.

[26] MongoDB. "MMAPv1 Concurrency Improvement". MongoDB.

[27] MongoDB. "WiredTiger Concurrency and Compression". MongoDB.

[28] MongoDB. "FAQ Concurrency - How Does Sharding Affect Concurrency". MongoDB.

[29] MongoDB. "FAQ Concurrency - Do Operations Ever Yield the Lock". MongoDB.

[30] MongoDB (8 July 2009). "32-bit Limitations". MongoDB.

[31] David Mytton (25 September 2012). "Does Everybody Hate MongoDB". *Server Density*.

[32] https://blog.compose.io/mongodb-scaling-to-100gb-and-beyond/. Missing or empty |title= (help)

[33] "memcmp". *cppreference.com*. 31 May 2013. Retrieved 26 April 2014.

[34] "MongoDB Jira ticket 1920". *jira.mongodb.org*.

[35] MongoDB. "MongoDB Drivers and Client Libraries". MongoDB. Retrieved 2013-07-08.

[36] MongoDB. "Community Supported Drivers". MongoDB. Retrieved 2014-07-09.

[37] MongoDB. "Admin UIs". Retrieved 15 September 2015.

[38] MongoDB. "The AGPL". *The MongoDB NoSQL Database Blog*. MongoDB.

[39] United Software Associates. "High Performance Benchmarking: MongoDB and NoSQL Systems" (PDF).

[40] End Point (13 April 2015). "Benchmarking Top NoSQL Databases; Apache Cassandra, Couchbase, HBase, and MongoDB" (PDF).

[41] MongoDB. "Adobe Experience Manager". MongoDB.

[42] "Presentation by Amadeus 11/2014". MongoDB.

[43] "Holy Large Hadron Collider, Batman!". MongoDB.

[44] MongoDB. "Craigslist". MongoDB.

[45] "MongoDB at eBay". *Slideshare*.

[46] "MongoDB based FIFA Online". MongoDB.

[47] "Experiences Deploying MongoDB on AWS". MongoDB.

[48] "Presentation by LinkedIn". MongoDB.

[49] MongoDB. "McAfee is Improving Global Cybersecurity with MongoDB". MongoDB.

[50] Doug Henschen (13 May 2013). "Metlife uses nosql for customer service". Information Week. Retrieved 8 November 2014.

[51] Richard Hirsch (30 September 2011). "The Quest to Understand the Use of MongoDB in the SAP PaaS".

[52] Guy Harrison (28 January 2011). "Real World NoSQL: MongoDB at Shutterfly". *Gigaom*.

[53] "We host the MongoDB user group meetup at our office".

[54] "Yandex: MongoDB". Yandex.

9.2.9 Bibliography

- Hoberman, Steve (June 1, 2014), *Data Modeling for MongoDB* (1st ed.), Technics Publications, p. 226, ISBN 978-1-935504-70-2
- Banker, Kyle (March 28, 2011), *MongoDB in Action* (1st ed.), Manning, p. 375, ISBN 978-1-935182-87-0
- Chodorow, Kristina; Dirolf, Michael (September 23, 2010), *MongoDB: The Definitive Guide* (1st ed.), O'Reilly Media, p. 216, ISBN 978-1-4493-8156-1
- Pirtle, Mitch (March 3, 2011), *MongoDB for Web Development* (1st ed.), Addison-Wesley Professional, p. 360, ISBN 978-0-321-70533-4
- Hawkins, Tim; Plugge, Eelco; Membrey, Peter (September 26, 2010), *The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing* (1st ed.), Apress, p. 350, ISBN 978-1-4302-3051-9

9.2.10 External links

- Official website

9.3 PostgreSQL

PostgreSQL, often simply **Postgres**, is an object-relational database management system (ORDBMS) with an emphasis on extensibility and standards-compliance. As a database server, its primary function is to store data securely, supporting best practices, and to allow for retrieval at the request of other software applications. It can handle workloads ranging from small single-machine applications to large Internet-facing applications with many concurrent users.

PostgreSQL implements the majority of the SQL:2011 standard,[9][10] is ACID-compliant and transactional (including most DDL statements) avoiding locking issues using multiversion concurrency control (MVCC), provides immunity to dirty reads and full serializability; handles complex SQL queries using many indexing methods that are not available in other databases; has updateable views and materialized views, triggers, foreign keys; supports functions and stored procedures, and other

expandability,[11] and has a large number of extensions written by third parties. In addition to the possibility of working with the major proprietary and open source databases, PostgreSQL supports migration from them, by its extensive standard SQL support and available migration tools. Proprietary extensions in databases such as Oracle can be emulated by built-in and third-party open source compatibility extensions. Recent versions also provide replication of the database itself for availability and scalability.

PostgreSQL is cross-platform and runs on many operating systems including Linux, FreeBSD, OS X, Solaris, and Microsoft Windows. On OS X, PostgreSQL has been the default database starting with Mac OS X 10.7 Lion Server, [12][13][14] and PostgreSQL client tools are bundled with in the desktop edition. The vast majority of Linux distributions have it available in supplied packages.

PostgreSQL is developed by the PostgreSQL Global Development Group, a diverse group of many companies and individual contributors.[15] It is free and open-source software, released under the terms of the PostgreSQL License, a permissive free-software license.

9.3.1 Name

PostgreSQL's developers pronounce it /ˈpoʊstgrɛs ˌkjuːˈɛl/.[16] It is abbreviated as *Postgres*, its original name. Because of ubiquitous support for the SQL Standard among most relational databases, the community considered changing the name back to Postgres. However, the PostgreSQL Core Team announced in 2007 that the product would continue to use the name PostgreSQL.[17] The name refers to the project's origins as a "post-Ingres" database, being a development from University Ingres DBMS (*Ingres* being an acronym for *INteractive Graphics Retrieval System*).[18][19]

9.3.2 History

PostgreSQL evolved from the Ingres project at the University of California, Berkeley. In 1982 the leader of the Ingres team, Michael Stonebraker, left Berkeley to make a proprietary version of Ingres.[18] He returned to Berkeley in 1985, and started a post-Ingres project to address the problems with contemporary database systems that had become increasingly clear during the early 1980s. The new project, POSTGRES, aimed to add the fewest features needed to completely support types.[20] These features included the ability to define types and to fully describe relationships – something used widely before but maintained entirely by the user. In POSTGRES, the database "understood" relationships, and could retrieve information in related tables in a natural way using *rules*. POSTGRES used many of the ideas of Ingres, but not its code.[21]

Starting in 1986, the POSTGRES team published a number of papers describing the basis of the system, and by 1987 had a prototype version shown at the 1988 ACM SIGMOD Conference. The team released version 1 to a small number of users in June 1989, then version 2 with a re-written rules system in June 1990. Version 3, released in 1991, again re-wrote the rules system, and added support for multiple storage managers and an improved query engine. By 1993, the great number of users began to overwhelm the project with requests for support and features. After releasing version 4.2[22] on June 30, 1994—primarily a cleanup—the project ended. Berkeley had released POSTGRES under an MIT-style license, which enabled other developers to use the code for any use. At the time, POSTGRES used an Ingres-influenced POSTQUEL query language interpreter, which could be interactively used with a console application named monitor.

In 1994, Berkeley graduate students Andrew Yu and Jolly Chen replaced the POSTQUEL query language interpreter with one for the SQL query language, creating Postgres95. The front-end program monitor was also replaced by psql. Yu and Chen released the code on the web.

On July 8, 1996, Marc Fournier at Hub.org Networking Services provided the first non-university development server for the open-source development effort.[1] With the participation of Bruce Momjian and Vadim B. Mikheev, work began to stabilize the code inherited from Berkeley. The first open-source version was released on August 1, 1996.

In 1996, the project was renamed to PostgreSQL to reflect its support for SQL. The online presence at the website PostgreSQL.org began on October 22, 1996.[23] The first PostgreSQL release formed version 6.0 on January 29, 1997. Since then a group of developers and volunteers around the world have maintained the software as The PostgreSQL Global Development Group.

The PostgreSQL project continues to make major releases (approximately annually) and minor "bugfix" releases, all available under its free and open-source software PostgreSQL License. Code comes from contributions from proprietary vendors, support companies, and open-source programmers at large. See also Release history below.

9.3.3 Multiversion concurrency control (MVCC)

PostgreSQL manages concurrency through a system known as multiversion concurrency control (MVCC), which gives each transaction a "snapshot" of the database, allowing changes to be made without being visible to other transactions until the changes are committed. This largely eliminates the need for read locks, and ensures the database maintains the ACID (atomicity, consistency,

isolation, durability) principles in an efficient manner. PostgreSQL offers three levels of transaction isolation: Read Committed, Repeatable Read and Serializable. Because PostgreSQL is immune to dirty reads, requesting a Read Uncommitted transaction isolation level provides read committed instead. Prior to PostgreSQL 9.1, requesting Serializable provided the same isolation level as Repeatable Read. PostgreSQL 9.1 and later support full serializability via the serializable snapshot isolation (SSI) technique.[24]

9.3.4 Storage and replication

Replication

PostgreSQL, beginning with version 9.0, includes built-in binary replication, based on shipping the changes (write-ahead logs) to replica nodes asynchronously.

Version 9.0 also introduced the ability to run read-only queries against these replicated nodes, where earlier versions would only allow that after promoting them to be a new master. This allows splitting read traffic among multiple nodes efficiently. Earlier replication software that allowed similar read scaling normally relied on adding replication triggers to the master, introducing additional load onto it.

Beginning from version 9.1, PostgreSQL also includes built-in synchronous replication[25] that ensures that, for each write transaction, the master waits until at least one replica node has written the data to its transaction log. Unlike other database systems, the durability of a transaction (whether it is asynchronous or synchronous) can be specified per-database, per-user, per-session or even per-transaction. This can be useful for work loads that do not require such guarantees, and may not be wanted for all data as it will have some negative effect on performance due to the requirement of the confirmation of the transaction reaching the synchronous standby.

There can be a mixture of synchronous and asynchronous standby servers. A list of synchronous standby servers can be specified in the configuration which determines which servers are candidates for synchronous replication. The first in the list which is currently connected and actively streaming is the one that will be used as the current synchronous server. When this fails, it falls to the next in line.

Synchronous multi-master replication is currently not included in the PostgreSQL core. Postgres-XC which is based on PostgreSQL provides scalable synchronous multi-master replication,[26] available in version 1.1 is licensed under the same license as PostgreSQL. A similar project is called Postgres-XL and is available under the Mozilla Public License.[27]

The community has also written some tools to make managing replication clusters easier, such as repmgr.

There are also several asynchronous trigger-based replication packages for PostgreSQL. These remain useful even after introduction of the expanded core capabilities, for situations where binary replication of an entire database cluster is not the appropriate approach:

- Slony-I
- Londiste, part of SkyTools (developed by Skype)
- Bucardo multi-master replication (developed by Backcountry.com)[28]
- SymmetricDS multi-master, multi-tier replication

Indexes

PostgreSQL includes built-in support for regular B-tree and hash indexes, and two types of inverted indexes: generalized search trees (GiST) and generalized inverted indexes (GIN). Hash indexes are implemented, but discouraged because they cannot be recovered after a crash or power loss. In addition, user-defined index methods can be created, although this is quite an involved process. Indexes in PostgreSQL also support the following features:

- Expression indexes can be created with an index of the result of an expression or function, instead of simply the value of a column.
- Partial indexes, which only index part of a table, can be created by adding a WHERE clause to the end of the CREATE INDEX statement. This allows a smaller index to be created.
- The planner is capable of using multiple indexes together to satisfy complex queries, using temporary in-memory bitmap index operations.
- As of PostgreSQL 9.1, *k*-nearest neighbors (*k*-NN) indexing (also referred to KNN-GiST) provides efficient searching of "closest values" to that specified, useful to finding similar words, or close objects or locations with geospatial data. This is achieved without exhaustive matching of values.
- In PostgreSQL 9.2 and above, index-only scans often allow the system to fetch data from indexes without ever having to access the main table.

Schemas

In PostgreSQL, all objects (with the exception of roles and tablespaces) are held within a schema. Schemas effectively act like namespaces, allowing objects of the same name to co-exist in the same database.

By default, databases are created with the "public" schema, but any additional schemas can be added, and the

public schema isn't mandatory. A “search_path” determines the order in which schemas are checked on unqualified objects (those without a prefixed schema), which can be configured on a database or role level. The search path, by default, contains the special schema name of "$user”, which first looks for a schema named after the connected database user (e.g. if the user “dave” were connected, it would first look for a schema also named “dave” when referring to any objects). If such a schema is not found, it then proceeds to the next schema. New objects are created in whichever valid schema (one that presently exists) is listed first in the search path.

Data types

A wide variety of native data types are supported, including:

- Boolean
- Arbitrary precision numerics
- Character (text, varchar, char)
- Binary
- Date/time (timestamp/time with/without timezone, date, interval)
- Money
- Enum
- Bit strings
- Text search type
- Composite
- HStore (an extension enabled key-value store within PostgreSQL)
- Arrays (variable length and can be of any data type, including text and composite types) up to 1 GB in total storage size
- Geometric primitives
- IPv4 and IPv6 addresses
- CIDR blocks and MAC addresses
- XML supporting XPath queries
- UUID
- JSON (since version 9.2), and a faster binary JSONB (since version 9.4; not the same as BSON[29])

In addition, users can create their own data types which can usually be made fully indexable via PostgreSQL's GiST infrastructure. Examples of these include the geographic information system (GIS) data types from the PostGIS project for PostgreSQL.

There is also a data type called a “domain”, which is the same as any other data type but with optional constraints defined by the creator of that domain. This means any data entered into a column using the domain will have to conform to whichever constraints were defined as part of the domain.

Starting with PostgreSQL 9.2, a data type that represents a range of data can be used which are called range types. These can be discrete ranges (e.g. all integer values 1 to 10) or continuous ranges (e.g. any point in time between 10:00 am and 11:00 am). The built-in range types available include ranges of integers, big integers, decimal numbers, time stamps (with and without time zone) and dates.

Custom range types can be created to make new types of ranges available, such as IP address ranges using the inet type as a base, or float ranges using the float data type as a base. Range types support inclusive and exclusive range boundaries using the [] and () characters respectively. (e.g. '[4,9)' represents all integers starting from and including 4 up to but not including 9.) Range types are also compatible with existing operators used to check for overlap, containment, right of etc.

User-defined objects

New types of almost all objects inside the database can be created, including:

- Casts
- Conversions
- Data types
- Domains
- Functions, including aggregate functions and window functions
- Indexes including custom indexes for custom types
- Operators (existing ones can be overloaded)
- Procedural languages

Inheritance

Tables can be set to inherit their characteristics from a “parent” table. Data in child tables will appear to exist in the parent tables, unless data is selected from the parent table using the ONLY keyword, i.e. SELECT * FROM

ONLY parent_table;. Adding a column in the parent table will cause that column to appear in the child table.

Inheritance can be used to implement table partitioning, using either triggers or rules to direct inserts to the parent table into the proper child tables.

As of 2010, this feature is not fully supported yet—in particular, table constraints are not currently inheritable. All check constraints and not-null constraints on a parent table are automatically inherited by its children. Other types of constraints (unique, primary key, and foreign key constraints) are not inherited.

Inheritance provides a way to map the features of generalization hierarchies depicted in Entity Relationship Diagrams (ERD) directly into the PostgreSQL database.

Other storage features

- Referential integrity constraints including foreign key constraints, column constraints, and row checks
- Binary and textual large-object storage
- Tablespaces
- Per-column collation (from 9.1)
- Online backup
- Point-in-time recovery, implemented using write-ahead logging
- In-place upgrades with pg_upgrade for less downtime (supports upgrades from 8.3.x and later)

9.3.5 Control and connectivity

Foreign data wrappers

As of version 9.1, PostgreSQL can link to other systems to retrieve data via foreign data wrappers (FDWs). These can take the form of any data source, such as a file system, another RDBMS, or a web service. This means regular database queries can use these data sources like regular tables, and even join multiple data sources together.

Interfaces

PostgreSQL has several interfaces available and is also widely supported among programming language libraries. Built-in interfaces include libpq (PostgreSQL's official C application interface) and ECPG (an embedded C system). External interfaces include:

- libpqxx: C++ interface
- PostgresDAC: PostgresDAC (for Embarcadero RadStudio/Delphi/CBuilder XE-XE3)
- DBD::Pg: Perl DBI driver
- JDBC: JDBC interface
- Lua: Lua interface
- Npgsql: .NET data provider
- ST-Links SpatialKit: Link Tool to ArcGIS
- PostgreSQL.jl: Julia interface
- node-postgres: Node.js interface
- pgoledb: OLEDB interface
- psqlODBC: ODBC interface
- psycopg2:[30] Python interface (also used by HTSQL)
- pgtclng: Tcl interface
- pyODBC: Python library
- php5-pgsql: PHP driver based on libpq
- postmodern: A Common Lisp interface
- pq: A pure Go PostgreSQL driver for the Go database/sql package. The driver passes the compatibility test suite.[31]

Procedural languages

Procedural languages allow developers to extend the database with custom subroutines (functions), often called *stored procedures*. These functions can be used to build triggers (functions invoked upon modification of certain data) and custom aggregate functions. Procedural languages can also be invoked without defining a function, using the "DO" command at SQL level.

Languages are divided into two groups: "Safe" languages are sandboxed and can be safely used by any user. Procedures written in "unsafe" languages can only be created by superusers, because they allow bypassing the database's security restrictions, but can also access sources external to the database. Some languages like Perl provide both safe and unsafe versions.

PostgreSQL has built-in support for three procedural languages:

- Plain SQL (safe). Simpler SQL functions can get expanded inline into the calling (SQL) query, which saves function call overhead and allows the query optimizer to "see inside" the function.
- PL/pgSQL (safe), which resembles Oracle's PL/SQL procedural language and SQL/PSM.

- C (unsafe), which allows loading custom shared libraries into the database. Functions written in C offer the best performance, but bugs in code can crash and potentially corrupt the database. Most built-in functions are written in C.

In addition, PostgreSQL allows procedural languages to be loaded into the database through extensions. Three language extensions are included with PostgreSQL to support Perl, Python and Tcl. There are external projects to add support for many other languages, including Java, JavaScript (PL/V8), R.

Triggers

Triggers are events triggered by the action of SQL DML statements. For example, an INSERT statement might activate a trigger that checks if the values of the statement are valid. Most triggers are only activated by either INSERT or UPDATE statements.

Triggers are fully supported and can be attached to tables. In PostgreSQL 9.0 and above, triggers can be per-column and conditional, in that UPDATE triggers can target specific columns of a table, and triggers can be told to execute under a set of conditions as specified in the trigger's WHERE clause. As of PostgreSQL 9.1, triggers can be attached to views by utilising the INSTEAD OF condition. Views in versions prior to 9.1 can have rules, though. Multiple triggers are fired in alphabetical order. In addition to calling functions written in the native PL/pgSQL, triggers can also invoke functions written in other languages like PL/Python or PL/Perl.

Asynchronous notifications

PostgreSQL provides an asynchronous messaging system that is accessed through the NOTIFY, LISTEN and UNLISTEN commands. A session can issue a NOTIFY command, along with the user-specified channel and an optional payload, to mark a particular event occurring. Other sessions are able to detect these events by issuing a LISTEN command, which can listen to a particular channel. This functionality can be used for a wide variety of purposes, such as letting other sessions know when a table has updated or for separate applications to detect when a particular action has been performed. Such a system prevents the need for continuous polling by applications to see if anything has yet changed, and reducing unnecessary overhead. Notifications are fully transactional, in that messages are not sent until the transaction they were sent from is committed. This eliminates the problem of messages being sent for an action being performed which is then rolled back.

Many of the connectors for PostgreSQL provide support for this notification system (including libpq, JDBC, Npgsql, psycopg and node.js) so it can be used by external applications.

Rules

Rules allow the "query tree" of an incoming query to be rewritten. Rules, or more properly, "Query Re-Write Rules", are attached to a table/class and "Re-Write" the incoming DML (select, insert, update, and/or delete) into one or more queries that either replace the original DML statement or execute in addition to it. Query Re-Write occurs after DML statement parsing, but before query planning.

Other querying features

- Transactions
- Full text search
- Views
  - Materialized views[32]
  - Updateable views[33]
  - Recursive views[34]
- Inner, outer (full, left and right), and cross joins
- Sub-selects
  - Correlated sub-queries[35]
- Regular expressions[36]
- Common table expressions and writable common table expressions
- Encrypted connections via TLS (current versions do not use vulnerable SSL, even with that configuration option)[37]
- Domains
- Savepoints
- Two-phase commit
- TOAST (*The Oversized-Attribute Storage Technique*) is used to transparently store large table attributes (such as big MIME attachments or XML messages) in a separate area, with automatic compression.
- Embedded SQL is implemented using preprocessor. SQL code is first written embedded into C code. Then code is run through ECPG preprocessor, which replaces SQL with calls to code library. Then code can be compiled using a C compiler. Embedding works also with C++ but it does not recognize all C++ constructs.

9.3.6 Security

PostgreSQL manages its internal security on a per-role basis. A role is generally regarded to be a user (a role that can log in), or a group (a role of which other roles are members). Permissions can be granted or revoked on any object down to the column level, and can also allow/prevent the creation of new objects at the database, schema or table levels.

The sepgsql extension (provided with PostgreSQL as of version 9.1) provides an additional layer of security by integrating with SELinux. This utilises PostgreSQL's SECURITY LABEL feature.

PostgreSQL natively supports a broad number of external authentication mechanisms, including:

- password (either MD5 or plain-text)
- GSSAPI
- SSPI
- Kerberos
- ident (maps O/S user-name as provided by an ident server to database user-name)
- peer (maps local user name to database user name)
- LDAP
  - Active Directory
- RADIUS
- certificate
- PAM

The GSSAPI, SSPI, Kerberos, peer, ident and certificate methods can also use a specified "map" file that lists which users matched by that authentication system are allowed to connect as a specific database user.

These methods are specified in the cluster's host-based authentication configuration file (pg_hba.conf), which determines what connections are allowed. This allows control over which user can connect to which database, where they can connect from (IP address/IP address range/domain socket), which authentication system will be enforced, and whether the connection must use TLS.

9.3.7 Upcoming features

Upcoming features in 9.5, in order of commit, include:

- IMPORT FOREIGN SCHEMA to import foreign tables from a foreign schema, meaning tables no longer have to be manually configured[38]
- ALTER TABLE ... SET LOGGED / UNLOGGED for switching tables between logged and unlogged states[39]
- Row-Level Security Policies for controlling which rows are visible or can be added to a table[40]
- SKIP LOCKED for row-level locks[41]
- BRIN (Block Range Indexes) to speed up queries on very large tables[42]
- Parallel VACUUMing with vacuumdb tool[43]
- Foreign tables can inherit and be inherited from[44]
- pg_rewind tool to efficiently resynchronise failed primary to new primary[45]
- Index-only scans on GiST indexes[46]
- CREATE TRANSFORM for mapping data type structures to procedural language data types[47]
- INSERT ... ON CONFLICT DO NOTHING/UPDATE which provides "UPSERT"-style functionality[48]
- JSONB-modifying operators and functions[49]
- TABLESAMPLE clause to specify random sampling[50]
- GROUPING SETS, CUBE and ROLLUP support[51]

Upcoming features in 9.6, in order of commit, include:

- Parallel sequential scan[52]

9.3.8 Add-ons

- MADlib: an open source analytics library for PostgreSQL providing mathematical, statistical and machine-learning methods for structured and unstructured data
- MySQL migration wizard: included with EnterpriseDB's PostgreSQL installer (source code also available)[53]
- Performance Wizard: included with EnterpriseDB's PostgreSQL installer (source code also available)[53]
- pgRouting: extended PostGIS to provide geospatial routing functionality[54] (GNU GPL)
- PostGIS: a popular add-on which provides support for geographic objects (GNU GPL)
- Postgres Enterprise Manager: a non-free tool consisting of a service, multiple agents, and a GUI which provides remote monitoring, management, reporting, capacity planning and tuning[55]
- ST-Links SpatialKit: Extension for directly connecting to spatial databases[56]

9.3.9 Benchmarks and performance

Many informal performance studies of PostgreSQL have been done.[57] Performance improvements aimed at improving scalability started heavily with version 8.1. Simple benchmarks between version 8.0 and version 8.4 showed that the latter was more than 10 times faster on read-only workloads and at least 7.5 times faster on both read and write workloads.[58]

The first industry-standard and peer-validated benchmark was completed in June 2007 using the Sun Java System Application Server (proprietary version of GlassFish) 9.0 Platform Edition, UltraSPARC T1-based Sun Fire server and PostgreSQL 8.2.[59] This result of 778.14 SPECjAppServer2004 JOPS@Standard compares favourably with the 874 JOPS@Standard with Oracle 10 on an Itanium-based HP-UX system.[57]

In August 2007, Sun submitted an improved benchmark score of 813.73 SPECjAppServer2004 JOPS@Standard. With the system under test at a reduced price, the price/performance improved from $US 84.98/JOPS to $US 70.57/JOPS.[60]

The default configuration of PostgreSQL uses only a small amount of dedicated memory for performance-critical purposes such as caching database blocks and sorting. This limitation is primarily because older operating systems required kernel changes to allow allocating large blocks of shared memory.[61] PostgreSQL.org provides advice on basic recommended performance practice in a wiki.[62]

In April 2012, Robert Haas of EnterpriseDB demonstrated PostgreSQL 9.2's linear CPU scalability using a server with 64 cores.[63]

Matloob Khushi performed benchmarking between Postgresql 9.0 and MySQL 5.6.15 for their ability to process genomic data. In his performance analysis he found that PostgreSQL extracts overlapping genomic regions 8x times faster than MySQL using two datasets of 80,000 each forming random human DNA regions. Insertion and data uploads in PostgreSQL were also better, although general searching capability of both databases was almost equivalent . [64]

9.3.10 Platforms

PostgreSQL is available for the following operating systems: Linux (all recent distributions), Windows (Windows 2000 SP4 and later) (compilable by e.g. Visual Studio, now with up to most recent 2015 version), DragonFly BSD, FreeBSD, OpenBSD, NetBSD, Mac OS X,[14] AIX, BSD/OS, HP-UX, IRIX, OpenIndiana,[65] OpenSolaris, SCO OpenServer, SCO UnixWare, Solaris and Tru64 Unix. In 2012, support for the following obsolete systems was removed (still supported in 9.1):[66] DG/UX, NeXTSTEP, SunOS 4, SVR4, Ultrix 4, and Univel. Most other Unix-like systems should also work.

PostgreSQL works on any of the following instruction set architectures: x86 and x86-64 on Windows and other operating systems; these are supported on other than Windows: IA-64 Itanium, PowerPC, PowerPC 64, S/390, S/390x, SPARC, SPARC 64, Alpha, ARMv8-A (64-bit)[67] and older ARM (32-bit, including older such as ARMv6 in Raspberry Pi[68]), MIPS, MIPSel, M68k, and PA-RISC. It is also known to work on M32R, NS32k, and VAX. In addition to these, it is possible to build PostgreSQL for an unsupported CPU by disabling spinlocks.[69]

9.3.11 Database administration

See also: Comparison of database tools

Open source front-ends and tools for administering PostgreSQL include:

psql The primary front-end for PostgreSQL is the psql command-line program, which can be used to enter SQL queries directly, or execute them from a file. In addition, psql provides a number of meta-commands and various shell-like features to facilitate writing scripts and automating a wide variety of tasks; for example tab completion of object names and SQL syntax.

pgAdmin The pgAdmin package is a free and open source graphical user interface administration tool for PostgreSQL, which is supported on many computer platforms.[70] The program is available in more than a dozen languages. The first prototype, named pgManager, was written for PostgreSQL 6.3.2 from 1998, and rewritten and released as pgAdmin under the GNU General Public License (GPL) in later months. The second incarnation (named pgAdmin II) was a complete rewrite, first released on January 16, 2002. The third version, pgAdmin III, was originally released under the Artistic License and then released under the same license as PostgreSQL. Unlike prior versions that were written in Visual Basic, pgAdmin III is written in C++, using the wxWidgets framework allowing it to run on most common operating systems. The query tool includes a scripting language called pgScript for supporting admin and development tasks. In December 2014, Dave Page, the pgAdmin project founder and primary developer,[71] announced that with the shift towards web-based models work has started on pgAdmin 4 with the aim of facilitating Cloud deployments.[72] Although still at the concept stage,[73] the plan is to build a single Python-based pgAdmin that users can either deploy on a web server or run from their desktop.

phpPgAdmin phpPgAdmin is a web-based administration tool for PostgreSQL written in PHP and based on the popular phpMyAdmin interface originally written for MySQL administration.[74]

PostgreSQL Studio PostgreSQL Studio allows users to perform essential PostgreSQL database development tasks from a web-based console. PostgreSQL Studio allows users to work with cloud databases without the need to open firewalls.[75]

TeamPostgreSQL AJAX/JavaScript-driven web interface for PostgreSQL. Allows browsing, maintaining and creating data and database objects via a web browser. The interface offers tabbed SQL editor with auto-completion, row-editing widgets, click-through foreign key navigation between rows and tables, 'favorites' management for commonly used scripts, among other features. Supports SSH for both the web interface and the database connections. Installers are available for Windows, Mac and Linux, as well as a simple cross-platform archive that runs from a script.[76]

LibreOffice/OpenOffice.org Base LibreOffice/OpenOffice.org Base can be used as a front-end for PostgreSQL.[77][78]

pgFouine The pgFouine PostgreSQL log analyzer generates detailed reports from a PostgreSQL log file and provides VACUUM analysis.[79]

A number of companies offer proprietary tools for PostgreSQL. They often consist of a universal core that is adapted for various specific database products. These tools mostly share the administration features with the open source tools but offer improvements in data modeling, importing, exporting or reporting.

9.3.12 Prominent users

Prominent organizations and products that use PostgreSQL as the primary database include:

- Yahoo! for web user behavioral analysis, storing two petabytes and claimed to be the largest data warehouse using a heavily modified version of PostgreSQL with an entirely different column-based storage engine and different query processing layer. While for performance, storage, and query purposes the database bears little resemblance to PostgreSQL, the front-end maintains compatibility so that Yahoo can use many off-the-shelf tools already written to interact with PostgreSQL.[80][81]
- In 2009, social networking website MySpace used Aster Data Systems's nCluster database for data warehousing, which was built on unmodified PostgreSQL.[82][83]
- Geni.com uses PostgreSQL for their main genealogy database.[84]
- OpenStreetMap, a collaborative project to create a free editable map of the world.[85]
- Afilias, domain registries for .org, .info and others.[86]
- Sony Online multiplayer online games.[87]
- BASF, shopping platform for their agribusiness portal.[88]
- Reddit social news website.[89]
- Skype VoIP application, central business databases.[90]
- Sun xVM, Sun's virtualization and datacenter automation suite.[91]
- MusicBrainz, open online music encyclopedia.[92]
- The International Space Station for collecting telemetry data in orbit and replicating it to the ground.[93]
- MyYearbook social networking site.[94]
- Instagram, a popular mobile photo sharing service[95]
- Disqus, an online discussion and commenting service[96]
- TripAdvisor, travel information website of mostly user-generated content[97]

PostgreSQL is offered by some major vendors as software as a service:

- Heroku, a platform as a service provider, has supported PostgreSQL since the start in 2007.[98] They offer value-add features like full database "rollback" (ability to restore a database from any point in time),[99] which is based on WAL-E, open source software developed by Heroku.[100]
- In January 2012, EnterpriseDB released a cloud version of both PostgreSQL and their own proprietary Postgres Plus Advanced Server with automated provisioning for failover, replication, load-balancing, and scaling. It runs on Amazon Web Services.[101]
- VMware offers vFabric Postgres for private clouds on vSphere since May 2012.[102]
- In November 2013, Amazon.com announced that they are adding PostgreSQL to their Relational Database Service offering.[103][104]

9.3.13 Proprietary derivatives and support

Although the license allows proprietary products based on Postgres, the code did not develop in the proprietary space at first. The first main offshoot originated when Paula Hawthorn (an original Ingres team member who moved from Ingres) and Michael Stonebraker formed Illustra Information Technologies to make a proprietary product based on POSTGRES.

In 2000, former Red Hat investors created the company Great Bridge to make a proprietary product based on PostgreSQL and compete against proprietary database vendors. Great Bridge sponsored several PostgreSQL developers and donated many resources back to the community,[105] but by late 2001 closed due to tough competition from companies like Red Hat and to poor market conditions.[106][107]

In 2001, Command Prompt, Inc. released Mammoth PostgreSQL, a proprietary product based on PostgreSQL. In 2008, Command Prompt, Inc. released the source under the original license. Command Prompt, Inc. continues to support the PostgreSQL community actively through developer sponsorships and projects including PL/Perl, PL/php, and hosting of community projects such as the PostgreSQL build farm.

In January 2005, PostgreSQL received backing by database vendor Pervasive Software, known for its Btrieve product which was ubiquitous on the Novell NetWare platform. Pervasive announced commercial support and community participation and achieved some success. In July 2006, Pervasive left the PostgreSQL support market.[108]

In mid-2005, two other companies announced plans to make proprietary products based on PostgreSQL with focus on separate niche markets. EnterpriseDB added functionality to allow applications written to work with Oracle to be more readily run with PostgreSQL. Greenplum contributed enhancements directed at data warehouse and business intelligence applications, including the BizGres project.

In October 2005, John Loiacono, executive vice president of software at Sun Microsystems, commented: "We're not going to OEM Microsoft but we are looking at PostgreSQL right now,"[109] although no specifics were released at that time. By November 2005, Sun had announced support for PostgreSQL.[110] By June 2006, Sun Solaris 10 (June 2006 release) shipped with PostgreSQL.

In August 2007, EnterpriseDB introduced Postgres Plus (originally called EnterpriseDB Postgres), a preconfigured distribution of PostgreSQL including many contrib modules and add-on components.[111]

In 2011, 2ndQuadrant became a Platinum Sponsor of PostgreSQL, in recognition of their long-standing contributions and developer sponsorship. 2ndQuadrant employ one of the largest teams of PostgreSQL contributors and provide professional support for open source PostgreSQL.

Many other companies have used PostgreSQL as the base for their proprietary database projects, e.g. Truviso, Netezza, ParAccel (used in Amazon Redshift[112]). In many cases the products have been enhanced so much that the software has been forked, though with some features cherry-picked from later releases.

9.3.14 Release history

9.3.15 See also

- Comparison of relational database management systems

9.3.16 References

[1] "Happy Birthday, PostgreSQL!". PostgreSQL Global Development Group. July 8, 2008.

[2] "2015-10-08 Security Update Release". *PostgreSQL*. The PostgreSQL Global Development Group. 2015-10-08. Retrieved 2015-10-08.

[3] "PostgreSQL licence approved by OSI". Crynwr. 2010-02-18. Retrieved 2010-02-18.

[4] "OSI PostgreSQL Licence". Open Source Initiative. 2010-02-20. Retrieved 2010-02-20.

[5] "License". PostgreSQL Global Development Group. Retrieved 2010-09-20.

[6] "Debian -- Details of package postgresql in sid". *debian.org*.

[7] "Licensing:Main". *FedoraProject*.

[8] "PostgreSQL". *fsf.org*.

[9] "SQL Conformance". postgresql.org. 2013-04-04. Retrieved 2013-08-28.

[10] "Appendix D. SQL Conformance". *PostgreSQL 9 Documentation*. PostgreSQL Global Development Group. 2009 [1996]. Retrieved 2013-04-01.

[11] "What is PostgreSQL?". *PostgreSQL 9.3.0 Documentation*. PostgreSQL Global Development Group. Retrieved 2013-09-20.

[12] "Lion Server: MySQL not included". 2011-08-04. Retrieved 2011-11-12.

[13] "OS X Lion Server — Technical Specifications". 2011-08-04. Retrieved 2011-11-12.

[14] http://www.postgresql.org/download/macosx/

[15] "Contributor Profiles". PostgreSQL. Retrieved December 17, 2011.

[16] Audio sample, 5.6k MP3

[17] "Project name — statement from the core team". archives.postgresql.org. 2007-11-16. Retrieved 2007-11-16.

[18] Stonebraker, M; Rowe, LA (May 1986). *The design of POSTGRES* (PDF). Proc. 1986 ACM SIGMOD Conference on Management of Data. Washington, DC. Retrieved 2011-12-17.

[19]

[20] Stonebraker, M; Rowe, LA. *The POSTGRES data model* (PDF). Proceedings of the 13th International Conference on Very Large Data Bases. Brighton, England: Morgan Kaufmann Publishers. pp. 83–96. ISBN 0-934613-46-X.

[21] Pavel Stehule (9 June 2012). "Historie projektu PostgreSQL" (in Czech).

[22] "University POSTGRES, Version 4.2". 1999-07-26.

[23] Page, Dave (2015-04-07). "Re: 20th anniversary of PostgreSQL ?". *pgsql-advocacy* (Mailing list). Retrieved 9 April 2015.

[24] Dan R. K. Ports, Kevin Grittner (2012). "Serializable Snapshot Isolation in PostgreSQL" (PDF). *Proceedings of the VLDB Endowment* **5** (12): 1850–1861.

[25] *PostgreSQL 9.1 with synchronous replication* (news), H Online

[26] *Postgres-XC project page* (website), Postgres-XC

[27] *Postgres-XL product page* (website), TransLattice

[28] Marit Fischer (2007-11-10). "Backcountry.com finally gives something back to the open source community" (Press release). Backcountry.com.

[29] Geoghegan, Peter (March 23, 2014). "What I think of jsonb".

[30] "PostgreSQL + Python — Psycopg". *initd.org*.

[31] "SQL database drivers". *Go wiki*. golang.org. Retrieved 22 June 2015.

[32] "Add a materialized view relations.". 2013-03-04. Retrieved 2013-03-04.

[33] "Support automatically-updatable views.". 2012-12-08. Retrieved 2012-12-08.

[34] "Add CREATE RECURSIVE VIEW syntax". 2013-02-01. Retrieved 2013-02-28.

[35] Momjian, Bruce (2001). "Subqueries". *PostgreSQL: Introduction and Concepts*. Addison-Wesley. ISBN 0-201-70331-9. Retrieved 2010-09-25.

[36] Bernier, Robert (2 February 2006). "Using Regular Expressions in PostgreSQL". O'Reilly Media. Retrieved 2010-09-25.

[37] "A few short notes about PostgreSQL and POODLE". *hagander.net*.

[38] "Implement IMPORT FOREIGN SCHEMA.". 2014-07-10. Retrieved 2014-09-11.

[39] "Implement ALTER TABLE .. SET LOGGED / UNLOGGED". 2014-08-22. Retrieved 2014-08-27.

[40] "Row-Level Security Policies (RLS)". 2014-09-19. Retrieved 2014-09-19.

[41] "Implement SKIP LOCKED for row-level locks". 2014-10-07. Retrieved 2014-10-07.

[42] "BRIN: Block Range Indexes". 2014-11-07. Retrieved 2014-11-09.

[43] "vacuumdb: enable parallel mode". 2015-01-23. Retrieved 2015-01-28.

[44] "Allow foreign tables to participate in inheritance.". 2015-03-22. Retrieved 2015-03-26.

[45] "Add pg_rewind, for re-synchronizing a master server after failback.". 2015-03-23. Retrieved 2015-03-26.

[46] "Add support for index-only scans in GiST.". 2015-03-26. Retrieved 2015-03-26.

[47] "Add transforms feature". 2015-04-26. Retrieved 2015-04-27.

[48] "Add support for INSERT ... ON CONFLICT DO NOTHING/UPDATE". 2015-05-08. Retrieved 2015-05-11.

[49] "Additional functions and operators for jsonb". 2015-05-12. Retrieved 2015-05-16.

[50] "TABLESAMPLE, SQL Standard and extensible". 2015-05-15. Retrieved 2015-05-16.

[51] "Support GROUPING SETS, CUBE and ROLLUP.". 2015-05-16. Retrieved 2015-05-16.

[52] "Generate parallel sequential scan plans in simple cases.". 2015-11-11. Retrieved 2015-11-18.

[53] "Postgres Plus Downloads". *Company website*. EnterpriseDB. Retrieved November 12, 2011.

[54] *pgRouting*, PostLBS

[55] "Postgres Enterprise Manager". *Company website*. EnterpriseDB. Retrieved November 12, 2011.

[56] *ST Links*

[57] Josh Berkus (2007-07-06). "PostgreSQL publishes first real benchmark". Retrieved 2007-07-10.

[58] György Vilmos (2009-09-29). "PostgreSQL history". Retrieved 2010-08-28.

[59] "SPECjAppServer2004 Result". SPEC. 2007-07-06. Retrieved 2007-07-10.

[60] "SPECjAppServer2004 Result". SPEC. 2007-07-04. Retrieved 2007-09-01.

[61] "Managing Kernel Resources". *PostgreSQL Manual*. PostgreSQL.org. Retrieved November 12, 2011.

[62] Greg Smith, Robert Treat, and Christopher Browne. "Tuning your PostgreSQL server". *Wiki*. PostgreSQL.org. Retrieved November 12, 2011.

[63] Robert Haas (2012-04-03). "Did I Say 32 Cores? How about 64?". Retrieved 2012-04-08.

[64] Matloob Khushi "Benchmarking database performance for genomic data.", *J Cell Biochem.*, 2015 Jun;116(6):877-83. doi: 10.1002/jcb.25049.

[65] "oi_151a Release Notes". OpenIndiana. Retrieved 2012-04-07.

[66] "Git — postgresql.git/commitdiff". Git.postgresql.org. Retrieved 2012-07-08.

[67] "AArch64 planning BoF at DebConf". *debian.org*.

[68] http://raspberrypg.org/2015/06/step-5-update-installing-postgresql-on-my-raspberry-pi-1-and-2/

[69] "Supported Platforms". PostgreSQL Global Development Group. Retrieved 2012-04-06.

[70] "pgAdmin: PostgreSQL administration and management tools". *website*. Retrieved November 12, 2011.

[71] "pgAdmin Development Team". *pgadmin.org*. Retrieved 22 June 2015.

[72] Dave, Page. "The story of pgAdmin". *Dave's Postgres Blog*. pgsnake.blogspot.co.uk. Retrieved 7 December 2014.

[73] Dave, Page. "pgAdmin 4". *Git Repository - The next generation of pgAdmin*. git.postgresql.org.

[74] phpPgAdmin Project (2008-04-25). "About phpPgAdmin". Retrieved 2008-04-25.

[75] PostgreSQL Studio (2013-10-09). "About PostgreSQL Studio". Retrieved 2013-10-09.

[76] "TeamPostgreSQL website". 2013-10-03. Retrieved 2013-10-03.

[77] oooforum.org (2010-01-10). "Back Ends for OpenOffice". Retrieved 2011-01-05.

[78] libreoffice.org (2012-10-14). "Base features". Retrieved 2012-10-14.

[79] Greg Smith (15 October 2010). *PostgreSQL 9.0 High Performance*. Packt Publishing. ISBN 978-1-84951-030-1.

[80] Eric Lai (2008-05-22). "Size matters: Yahoo claims 2-petabyte database is world's biggest, busiest". Computerworld.

[81] Thomas Claburn (2008-05-21). "Yahoo Claims Record With Petabyte Database". InformationWeek.

[82] Emmanuel Cecchet (May 21, 2009). *Building PetaByte Warehouses with Unmodified PostgreSQL* (PDF). PGCon 2009. Retrieved November 12, 2011.

[83] "MySpace.com scales analytics for all their friends" (PDF). case study. Aster Data. June 15, 2010. Archived (PDF) from the original on November 14, 2010. Retrieved November 12, 2011.

[84] "Last Weekend's Outage". *Blog*. Geni. 2011-08-01.

[85] "Database". *Wiki*. OpenStreetMap.

[86] *PostgreSQL affiliates .ORG domain*, AU: Computer World

[87] *Sony Online opts for open-source database over Oracle*, Computer World

[88] *A Web Commerce Group Case Study on PostgreSQL* (PDF) (1.2 ed.), PostgreSQL

[89] "Architecture Overview". *Reddit software wiki*. Reddit. 27 March 2014. Retrieved 2014-11-25.

[90] "PostgreSQL at Skype". Skype Developer Zone. 2006. Retrieved 2007-10-23.

[91] "How Much Are You Paying For Your Database?". Sun Microsystems blog. 2007. Retrieved 2007-12-14.

[92] "Database — MusicBrainz". MusicBrainz Wiki. Retrieved 5 February 2011.

[93] Duncavage, Daniel P (2010-07-13). "NASA needs Postgres-Nagios help".

[94] Roy, Gavin M (2010). "PostgreSQL at myYearbook.com" (talk). USA East: PostgreSQL Conference.

[95] "Keeping Instagram up with over a million new users in twelve hours". Instagram-engineering.tumblr.com. 2011-05-17. Retrieved 2012-07-07.

[96] "Postgres at Disqus". Retrieved May 24, 2013.

[97] Matthew Kelly (27 March 2015). *At The Heart Of A Giant: Postgres At TripAdvisor*. PGConf US 2015. (Presentation video)

[98] Alex Williams (1 April 2013). "Heroku Forces Customer Upgrade To Fix Critical PostgreSQL Security Hole". TechCrunch.

[99] Barb Darrow (11 November 2013). "Heroku gussies up Postgres with database roll-back and proactive alerts". GigaOM.

[100] Craig Kerstiens (26 September 2013). "WAL-E and Continuous Protection with Heroku Postgres". Heroku blog.

[101] "EnterpriseDB Offers Up Postgres Plus Cloud Database". Techweekeurope.co.uk. 2012-01-27. Retrieved 2012-07-07.

[102] Al Sargent (15 May 2012). "Introducing VMware vFabric Suite 5.1: Automated Deployment, New Components, and Open Source Support". VMware blogs.

[103] Jeff (14 November 2013). "Amazon RDS for PostgreSQL — Now Available". Amazon Web Services Blog.

[104] Alex Williams (14 November 2013). "PostgreSQL Now Available On Amazon's Relational Database Service". TechCrunch.

[105] Maya Tamiya (2001-01-10). "Interview: Bruce Momjian". LWN.net. Retrieved 2007-09-07.

[106] "Great Bridge ceases operations" (Press release). Great Bridge. 2001-09-06. Retrieved 2007-09-07.

[107] Nikolai Bezroukov (1 July 2004). "The Sunset of Linux Hype". Portraits of Open Source Pioneers. NORFOLK, Va., September 6, 2001 -- Great Bridge LLC, the company that pioneered commercial distribution and support of the PostgreSQL open source database, announced today that it has ceased business operations

[108] John Farr (2006-07-25). "Open letter to the PostgreSQL Community". Pervasive Software. Archived from the original on 2007-02-25. Retrieved 2007-02-13.

[109] Rodney Gedda (2005-10-05). "Sun's software chief eyes databases, groupware". Computerworld. Retrieved 2007-02-13.

[110] "Sun Announces Support for Postgres Database on Solaris 10" (Press release). Sun Microsystems. 2005-11-17. Retrieved 2007-02-13.

[111] "EnterpriseDB Announces First-Ever Professional-Grade PostgreSQL Distribution for Linux" (Press release). EnterpriseDB. 2007-08-07. Retrieved 2007-08-07.

[112] http://docs.aws.amazon.com/redshift/latest/dg/c_redshift-and-postgres-sql.html

[113] "Versioning policy". PostgreSQL Global Development Group. Retrieved 2012-01-30.

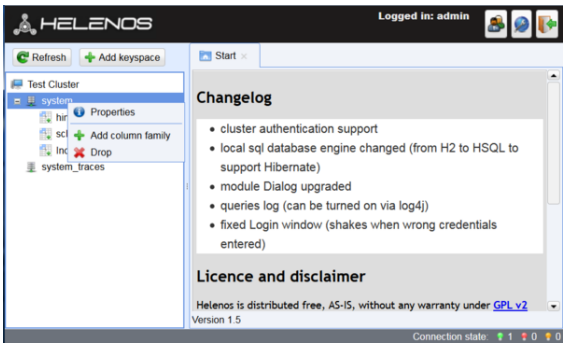
9.3.17 Further reading

- Obe, Regina; Hsu, Leo (July 8, 2012). *PostgreSQL: Up and Running*. O'Reilly. ISBN 1-4493-2633-1.
- Krosing, Hannu; Roybal, Kirk (June 15, 2013). *PostgreSQL Server Programming*. Packt Publishing. ISBN 9781849516983.
- Riggs, Simon; Krosing, Hannu (October 27, 2010). *PostgreSQL 9 Administration Cookbook*. Packt Publishing. ISBN 1-84951-028-8.
- Smith, Greg (October 15, 2010). *PostgreSQL 9 High Performance*. Packt Publishing. ISBN 1-84951-030-X.
- Gilmore, W. Jason; Treat, Robert (February 27, 2006). *Beginning PHP and PostgreSQL 8: From Novice to Professional*. Apress. p. 896. ISBN 1-59059-547-5.
- Douglas, Korry (August 5, 2005). *PostgreSQL* (Second ed.). Sams. p. 1032. ISBN 0-672-32756-2.
- Matthew, Neil; Stones, Richard (April 6, 2005). *Beginning Databases with PostgreSQL* (Second ed.). Apress. p. 664. ISBN 1-59059-478-9.
- Worsley, John C; Drake, Joshua D (January 2002). *Practical PostgreSQL*. O'Reilly Media. p. 636. ISBN 1-56592-846-6.

9.3.18 External links

- Official website
- PostgreSQL wiki
- https://www.depesz.com/
  - https://explain.depesz.com/ – PostgreSQL's explain analyze made readable
- PostgreSQL at DMOZ

9.4 Apache Cassandra



Helenos is a graphical user interface for Cassandra

Apache Cassandra is an open source distributed database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers robust support for clusters spanning multiple datacenters,[1] with asynchronous masterless replication allowing low latency operations for all clients.

Cassandra also places a high value on performance. In 2012, University of Toronto researchers studying NoSQL systems concluded that "In terms of scalability, there is a clear winner throughout our experiments. Cassandra achieves the highest throughput for the maximum number of nodes in all experiments" although "this comes at the price of high write and read latencies."[2]

9.4.1 History

Apache Cassandra was initially developed at Facebook to power their Inbox Search feature by Avinash Lakshman (one of the authors of Amazon's Dynamo) and Prashant Malik. It was released as an open source project on Google code in July 2008.[3] In March 2009, it became an Apache Incubator project.[4] On February 17, 2010 it graduated to a top-level project.[5]

It was named after the Greek mythological prophet Cassandra.[6]

Releases after graduation include

- 0.6, released Apr 12 2010, added support for integrated caching, and Apache Hadoop MapReduce[7]
- 0.7, released Jan 08 2011, added secondary indexes and online schema changes[8]
- 0.8, released Jun 2 2011, added the Cassandra Query Language (CQL), self-tuning memtables, and support for zero-downtime upgrades[9]
- 1.0, released Oct 17 2011, added integrated compression, leveled compaction, and improved read performance[10]
- 1.1, released Apr 23 2012, added self-tuning caches, row-level isolation, and support for mixed ssd/spinning disk deployments[11]
- 1.2, released Jan 2 2013, added clustering across virtual nodes, inter-node communication, atomic batches, and request tracing[12]
- 2.0, released Sep 4 2013, added lightweight transactions (based on the Paxos consensus protocol), triggers, improved compactions
- 2.0.4, released Dec 30 2013, added allowing specifying datacenters to participate in a repair, client encryption support to sstableloader, allow removing snapshots of no-longer-existing CFs[13]
- 2.1.0 released Sep 10 2014 [14]
- 2.1.6 released June 8, 2015
- 2.1.7 released June 22, 2015
- 2.2.0 released July 20, 2015
- 2.2.2 released October 5, 2015

9.4.2 Licensing and support

Apache Cassandra is an Apache Software Foundation project, so it has an Apache License (version 2.0).

9.4.3 Main features

Decentralized Every node in the cluster has the same role. There is no single point of failure. Data is distributed across the cluster (so each node contains different data), but there is no master as every node can service any request.

Supports replication and multi data center replication Replication strategies are configurable.[15] Cassandra is designed as a distributed system, for deployment of large numbers of nodes across multiple data centers. Key features of Cassandra's distributed architecture are specifically tailored for multiple-data center deployment, for redundancy, for failover and disaster recovery.

Scalability Read and write throughput both increase linearly as new machines are added, with no downtime or interruption to applications.

Fault-tolerant Data is automatically replicated to multiple nodes for fault-tolerance. Replication across multiple data centers is supported. Failed nodes can be replaced with no downtime.

Tunable consistency Writes and reads offer a tunable level of consistency, all the way from "writes never fail" to "block for all replicas to be readable", with the quorum level in the middle.[16]

MapReduce support Cassandra has Hadoop integration, with MapReduce support. There is support also for Apache Pig and Apache Hive.[17]

Query language Cassandra introduces CQL (Cassandra Query Language), a SQL-like alternative to the traditional RPC interface. Language drivers are available for Java (JDBC), Python (DBAPI2), Node.JS (Helenus), Go (gocql) and C++.[18]

Below an example of keyspace creation, including a column family in CQL 3.0:[19]

CREATE KEYSPACE MyKeySpace WITH REPLICATION = { 'class' : 'SimpleStrategy', 'replication_factor' : 3 }; USE MyKeySpace; CREATE COLUMNFAMILY MyColumns (id text, Last text, First text, PRIMARY KEY(id)); INSERT INTO MyColumns (id, Last, First) VALUES ('1', 'Doe', 'John'); SELECT * FROM MyColumns;

Which gives:

id | first | last ----+-------+------ 1 | John | Doe (1 rows)

9.4.4 Data model

Cassandra is essentially a hybrid between a key-value and a column-oriented (or tabular) database. Its data model is a partitioned row store with tunable consistency.[16] Rows are organized into tables; the first component of a table's primary key is the partition key; within a partition, rows are clustered by the remaining columns of the key.[20] Other columns may be indexed separately from the primary key.[21]

Tables may be created, dropped, and altered at run-time without blocking updates and queries.[22]

Cassandra does not support joins or subqueries. Rather, Cassandra emphasizes denormalization through features like collections.[23]

A column family (called "table" since CQL 3) resembles a table in an RDBMS. Column families contain rows and columns. Each row is uniquely identified by a row key. Each row has multiple columns, each of which has a name, value, and a timestamp. Unlike a table in an RDBMS, different rows in the same column family do not have to share the same set of columns, and a column may be added to one or multiple rows at any time.[24]

Each key in Cassandra corresponds to a value which is an object. Each key has values as columns, and columns are grouped together into sets called column families. Thus, each key identifies a row of a variable number of elements. These column families could be considered then as tables. A table in Cassandra is a distributed multi dimensional map indexed by a key. Furthermore, applications can specify the sort order of columns within a Super Column or Simple Column family.

9.4.5 Clustering

When the cluster for Apache Cassandra is designed, an important point is to select the right partitioner. Two partitioners exist:[25]

1. RandomPartitioner (RP): This partitioner randomly distributes the key-value pairs over the network, resulting in a good load balancing. Compared to OPP, more nodes have to be accessed to get a number of keys.
2. OrderPreservingPartitioner (OPP): This partitioner distributes the key-value pairs in a natural way so that similar keys are not far away. The advantage is that fewer nodes have to be accessed. The drawback is the uneven distribution of the key-value pairs.

9.4.6 Prominent users

- @WalmartLabs[26] (previously Kosmix) uses Cassandra with SSD
- Amadeus IT Group uses Cassandra for some of their back-end systems.
- Apple uses 100,000 Cassandra nodes, as revealed at Cassandra Summit San Francisco 2015,[27] although it has not elaborated for which products, services or features.
- AppScale uses Cassandra as a back-end for Google App Engine applications[28]
- CERN used Cassandra-based prototype for its ATLAS experiment to archive the online DAQ system's monitoring information[29]
- Cisco's WebEx uses Cassandra to store user feed and activity in near real time.[30]
- Cloudkick uses Cassandra to store the server metrics of their users.[31]
- Constant Contact uses Cassandra in their email and social media marketing applications.[32] Over 200 nodes are deployed.
- Digg, a large social news website, announced on Sep 9th, 2009 that it is rolling out its use of Cassandra[33] and confirmed this on March 8, 2010.[34] TechCrunch has since linked Cassandra to Digg v4 reliability criticisms and recent company struggles.[35] Lead engineers at Digg later rebuked these criticisms as red herring and blamed a lack of load testing.[36]
- Facebook used Cassandra to power Inbox Search, with over 200 nodes deployed.[37] This was abandoned in late 2010 when they built Facebook Messaging platform on HBase as they "found Cassandra's eventual consistency model to be a difficult pattern".[38]
- Formspring uses Cassandra to count responses, as well as store Social Graph data (followers, following, blockers, blocking) for 26 Million accounts with 10 million responses a day[39]
- IBM has done research in building a scalable email system based on Cassandra.[40]
- Mahalo.com uses Cassandra to record user activity logs and topics for their Q&A website[41][42]
- Netflix uses Cassandra as their back-end database for their streaming services[43][44]
- Nutanix appliances use Cassandra to store metadata and stats.[45]
- Ooyala built a scalable, flexible, real-time analytics engine using Cassandra[46]
- Openwave uses Cassandra as a distributed database and as a distributed storage mechanism for their next generation messaging platform[47]
- OpenX is running over 130 nodes on Cassandra for their OpenX Enterprise product to store and replicate advertisements and targeting data for ad delivery[48]
- Plaxo has "reviewed 3 billion contacts in [their] database, compared them with publicly available data sources, and identified approximately 600 million unique people with contact info."[49]
- PostRank used Cassandra as their backend database[50]
- Rackspace is known to use Cassandra internally.[51]

- Reddit switched to Cassandra from memcacheDB on March 12, 2010[52] and experienced some problems in May due to insufficient nodes in their cluster.[53]
- RockYou uses Cassandra to record every single click for 50 million Monthly Active Users in real-time for their online games[54]
- SoundCloud uses Cassandra to store the dashboard of their users[55]
- Talentica Software uses Cassandra as a back-end for Analytics Application with Cassandra cluster of 30 nodes and inserting around 200GB data on daily basis.[56]
- Tibbo Systems uses Cassandra as configuration and event storage for AggreGate Platform.
- Twitter announced it was planning to move entirely from MySQL to Cassandra,[57][58] though soon after retracted this, keeping Tweets in MySQL while using Cassandra for analytics.[59]
- Urban Airship uses Cassandra with the mobile service hosting for over 160 million application installs across 80 million unique devices[60]
- Wikimedia uses Cassandra as backend storage for its public-facing REST Content API.[61]
- Zoho uses Cassandra for generating the inbox preview in their Zoho#Zoho Mail service

Facebook moved off its pre-Apache Cassandra deployment in late 2010 when they replaced Inbox Search with the Facebook Messaging platform.[38] In 2012, Facebook began using Apache Cassandra in its Instagram unit.[62]

Cassandra is the most popular wide column store,[63] and in September 2014 surpassed Sybase to become the 9th most popular database, close behind Microsoft Access and SQLite.[64]

9.4.7 See also

Academic background

- BigTable - Original distributed database by Google
- Distributed database
- Distributed hash table (DHT)
- Dynamo (storage system) - Cassandra borrows many elements from Dynamo
- NoSQL

Commercial companies

- DataStax
- Impetus Technologies
- Instaclustr

Alternatives

- Apache Accumulo—Secure Apache Hadoop based distributed database.
- Aerospike
- Berkeley DB
- Druid (open-source data store)
- Apache HBase—Apache Hadoop based distributed database. Very similar to BigTable
- HyperDex
- Hypertable—Apache Hadoop based distributed database. Very similar to BigTable
- MongoDB
- Riak
- ScyllaDB

9.4.8 References

[1] Casares, Joaquin (2012-11-05). "Multi-datacenter Replication in Cassandra". DataStax. Retrieved 2013-07-25. Cassandra's innate datacenter concepts are important as they allow multiple workloads to be run across multiple datacenters…

[2] Rabl, Tilmann; Sadoghi, Mohammad; Jacobsen, Hans-Arno; Villamor, Sergio Gomez-; Mulero -, Victor Muntes; Mankovskii, Serge (2012-08-27). "Solving Big Data Challenges for Enterprise Application Performance Management" (PDF). VLDB. Retrieved 2013-07-25. In terms of scalability, there is a clear winner throughout our experiments. Cassandra achieves the highest throughput for the maximum number of nodes in all experiments... this comes at the price of high write and read latencies

[3] Hamilton, James (July 12, 2008). "Facebook Releases Cassandra as Open Source". Retrieved 2009-06-04.

[4] "Is this the new hotness now?". Mail-archive.com. 2009-03-02. Archived from the original on 25 April 2010. Retrieved 2010-03-29.

[5] "Cassandra is an Apache top level project". Mail-archive.com. 2010-02-18. Archived from the original on 28 March 2010. Retrieved 2010-03-29.

[6] http://kellabyte.com/2013/01/04/the-meaning-behind-the-name-of-apache-cassandra/

[7] The Apache Software Foundation Announces Apache Cassandra Release 0.6 : The Apache Software Foundation Blog

[8] The Apache Software Foundation Announces Apache Cassandra 0.7 : The Apache Software Foundation Blog

[9] [Cassandra-user] [RELEASE] 0.8.0 - Grokbase

[10] Cassandra 1.0.0. Is Ready for the Enterprise

[11] The Apache Software Foundation Announces Apache Cassandra™ v1.1 : The Apache Software Foundation Blog

[12] "The Apache Software Foundation Announces Apache Cassandra™ v1.2 : The Apache Software Foundation Blog". *apache.org*. Retrieved 11 December 2014.

[13] Eric Evans. "[Cassandra-User] [RELEASE] Apache Cassandra 2.0.4". *qnalist.com*. Retrieved 11 December 2014.

[14] Sylvain Lebresne (10 September 2014). "[VOTE SUCCESS] Release Apache Cassandra 2.1.0". *mail-archive.com*. Retrieved 11 December 2014.

[15] "Deploying Cassandra across Multiple Data Centers". *DataStax*. Retrieved 11 December 2014.

[16] DataStax (2013-01-15). "About data consistency". Retrieved 2013-07-25.

[17] "Hadoop Support" article on Cassandra's wiki

[18] "DataStax C/C++ Driver for Apache Cassandra". *DataStax*. Retrieved 15 December 2014.

[19] https://cassandra.apache.org/doc/cql3/CQL.html

[20] Ellis, Jonathan (2012-02-15). "Schema in Cassandra 1.1". DataStax. Retrieved 2013-07-25.

[21] Ellis, Jonathan (2010-12-03). "What's new in Cassandra 0.7: Secondary indexes". DataStax. Retrieved 2013-07-25.

[22] Ellis, Jonathan (2012-03-02). "The Schema Management Renaissance in Cassandra 1.1". DataStax. Retrieved 2013-07-25.

[23] Lebresne, Sylvain (2012-08-05). "Coming in 1.2: Collections support in CQL3". DataStax. Retrieved 2013-07-25.

[24] DataStax. "Apache Cassandra 0.7 Documentation - Column Families". *Apache Cassandra 0.7 Documentation*. Retrieved 29 October 2012.

[25] Williams, Dominic. "Cassandra: RandomPartitioner vs OrderPreservingPartitioner". http://wordpress.com/: WordPress.com. Retrieved 2011-03-23. When building a Cassandra cluster, the "key" question (sorry, that's weak) is whether to use the RandomPartitioner (RP), or the OrdengPartitioner (OPP). These control how your data is distributed over your nodes. Once you have chosen your partitioner, you cannot change without wiping your data, so think carefully! The problem with OPP: If the distribution of keys used by individual column families is different, their sets of keys will not fall evenly across the ranges assigned to nodes. Thus nodes will end up storing preponderances of keys (and the associated data) corresponding to one column family or another. If as is likely column families store differing quantities of data with their keys, or store data accessed according to differing usage patterns, then some nodes will end up with disproportionately more data than others, or serving more "hot" data than others.

[26] "@WalmartLabs". *walmartlabs.com*. Retrieved 11 December 2014.

[27] Luca Martinetti: Apple runs more than 100k [production] Cassandra nodes. on Twitter

[28] "Datastores on Appscale".

[29] "A Persistent Back-End for the ATLAS Online Information Service (P-BEAST)".

[30] "Re: Cassandra users survey". Mail-archive.com. 2009-11-21. Archived from the original on 17 April 2010. Retrieved 2010-03-29.

[31] 4 Months with Cassandra, a love story |Cloudkick, manage servers better

[32] Finley, Klint (2011-02-18). "This Week in Consolidation: HP Buys Vertica, Constant Contact Buys Bantam Live and More". Read Write Enterprise.

[33] Eure, Ian. "Looking to the future with Cassandra".

[34] Quinn, John. "Saying Yes to NoSQL; Going Steady with Cassandra".

[35] Schonfeld, Erick. "As Digg Struggles, VP Of Engineering Is Shown The Door".

[36] "Is Cassandra to Blame for Digg v4's Failures?".

[37] "Niet compatibele browser". Facebook. Retrieved 2010-03-29.

[38] Muthukkaruppan, Kannan. "The Underlying Technology of Messages".

[39] Cozzi, Martin (2011-08-31). "Cassandra at Formspring".

[40] "BlueRunner: Building an Email Service in the Cloud" (PDF). ieee.org. 2009-07-20. Retrieved 2010-03-29.

[41] "Mahalo.com powered by Apache Cassandra™" (PDF). *DataStax.com*. Santa Clara, CA, USA: DataStax. 2012-04-10. Retrieved 2014-06-13.

[42] Watch Cassandra at Mahalo.com |DataStax Episodes |Blip

[43] Cockcroft, Adrian (2011-07-11). "Migrating Netflix from Datacenter Oracle to Global Cassandra". *slideshare.net*. Retrieved 2014-06-13.

[44] Izrailevsky, Yury (2011-01-28). "NoSQL at Netflix".

[45] "Nutanix Bible".

[46] Ooyala (2010-05-18). "Designing a Scalable Database for Online Video Analytics" (PDF). *DataStax.com*. Mountain View CA, USA. Retrieved 2014-06-14.

[47] Mainstay LLC (2013-11-11). "DataStax Case Study of Openwave Messaging" (PDF). *DataStax.com*. Santa Clara, CA, USA: DataStax. Retrieved 2014-06-15.

[48] Ad Serving Technology - Advanced Optimization, Forecasting, & Targeting |OpenX

[49] Smalley, Preston (2011-03-20). "An important milestone - and it's only the beginning!".

[50] Grigorik, Ilya (2011-03-29). "Webpulp TV: Scaling PostRank with Ilya Grigorik".

[51] "Hadoop and Cassandra (at Rackspace)". Stu Hood. 2010-04-23. Retrieved 2011-09-01.

[52] david [ketralnis] (2010-03-12). "what's new on reddit: She who entangles men". blog.reddit. Archived from the original on 25 March 2010. Retrieved 2010-03-29.

[53] Posted by the reddit admins at (2010-05-11). "blog.reddit -- what's new on reddit: reddit's May 2010 "State of the Servers" report". blog.reddit. Archived from the original on 14 May 2010. Retrieved 2010-05-16.

[54] Pattishall, Dathan Vance (2011-03-23). "Cassandra is my NoSQL Solution but".

[55] "Cassandra at SoundCloud".

[56] cite web|url=http://www.talentica.com[]

[57] Popescu, Alex. "Cassandra @ Twitter: An Interview with Ryan King". myNoSQL. Archived from the original on 1 March 2010. Retrieved 2010-03-29.

[58] Babcock, Charles. "Twitter Drops MySQL For Cassandra - Cloud databases". InformationWeek. Archived from the original on 2 April 2010. Retrieved 2010-03-29.

[59] King, Ryan (2010-07-10). "Cassandra at Twitter Today". *blog.twitter.com*. San Francisco, CA, USA: Twitter. Retrieved 2014-06-20.

[60] Onnen, Erik. "From 100s to 100s of Millions".

[61] Wicke, Gabriel. "Wikimedia REST content API is now available in beta".

[62] Rick Branson (2013-06-26). "Cassandra at Instagram". DataStax. Retrieved 2013-07-25.

[63] DB-Engines. "DB-Engines Ranking of Wide Column Stores".

[64] DB-Engines. "DB-Engines Ranking".

9.4.9 Bibliography

- Hewitt, Eben (December 15, 2010). *Cassandra: The Definitive Guide* (1st ed.). O'Reilly Media. p. 300. ISBN 978-1-4493-9041-9.
- Capriolo, Edward (July 15, 2011). *Cassandra High Performance Cookbook* (1st ed.). Packt Publishing. p. 324. ISBN 1-84951-512-3.

9.4.10 External links

- Lakshman, Avinash (2008-08-25). "Cassandra - A structured storage system on a P2P Network". Engineering @ Facebook's Notes. Retrieved 2014-06-17.
- "The Apache Cassandra Project". Forest Hill, MD, USA: The Apache Software Foundation. Retrieved 2014-06-17.
- "Project Wiki". Forest Hill, MD, USA: The Apache Software Foundation. Retrieved 2014-06-17.
- Hewitt, Eben (2010-12-01). "Adopting Apache Cassandra". *infoq.com*. InfoQ, C4Media Inc. Retrieved 2014-06-17.
- Lakshman, Avinash; Malik, Prashant (2009-08-15). "Cassandra - A Decentralized Structured Storage System" (PDF). *cs.cornell.edu*. The authors are from Facebook. Retrieved 2014-06-17.
- Ellis, Jonathan (2009-07-29). "What Every Developer Should Know About Database Scalability". *slideshare.net*. Retrieved 2014-06-17. From the OSCON 2009 talk on RDBMS vs. Dynamo, BigTable, and Cassandra.
- "Cassandra-RPM - Red Hat Package Manager (RPM) build for the Apache Cassandra project". *code.google.com*. Menlo Park, CA, USA: Google Project Hosting. Retrieved 2014-06-17.
- Roth, Gregor (2012-10-14). "Cassandra by example - the path of read and write requests". *slideshare.net*. Retrieved 2014-06-17.
- Mansoor, Umer (2012-11-04). "A collection of Cassandra tutorials". Retrieved 2015-02-08.
- Bushik, Sergey (2012-10-22). "A vendor-independent comparison of NoSQL databases: Cassandra, HBase, MongoDB, Riak". *NetworkWorld*. Framingham, MA, USA and Staines, Middlesex, UK: IDG. Retrieved 2014-06-17.

9.5 Berkeley DB

Berkeley DB (**BDB**) is a software library that provides a high-performance embedded database for key/value data. Berkeley DB is written in C with API bindings for C++, C#, PHP, Java, Perl, Python, Ruby, Tcl, Smalltalk, and many other programming languages. BDB stores arbitrary key/data pairs as byte arrays, and supports multiple data items for a single key. Berkeley DB is not a relational database.[1]

BDB can support thousands of simultaneous threads of control or concurrent processes manipulating databases as large as 256 terabytes,[2] on a wide variety of operating

systems including most Unix-like and Windows systems, and real-time operating systems. "Berkeley DB" is also used as the common brand name for three distinct products: Oracle Berkeley DB, Berkeley DB Java Edition, and Berkeley DB XML. These three products all share a common ancestry and are currently under active development at Oracle Corporation.

9.5.1 Origin

Berkeley DB originated at the University of California, Berkeley as part of BSD, Berkeley's version of the Unix operating system. After 4.3BSD (1986), the BSD developers attempted to remove or replace all code originating in the original AT&T Unix from which BSD was derived. In doing so, they needed to rewrite the Unix database package.[3] A non-AT&T-copyrighted replacement, due to Seltzer and Yigit,[4] attempted to create a disk hash table that performed better than any of the existing Dbm libraries. Berkeley DB itself was first released in 1991 and later included with 4.4BSD.[3] In 1996 Netscape requested that the authors of Berkeley DB improve and extend the library, then at version 1.86, to suit Netscape's requirements for an LDAP server[5] and for use in the Netscape browser. That request led to the creation of Sleepycat Software. This company was acquired by Oracle Corporation in February 2006, which continues to develop and sell Berkeley DB.

Since its initial release, Berkeley DB has gone through various versions. Each major release cycle has introduced a single new major feature generally layering on top of the earlier features to add functionality to the product. The 1.x releases focused on managing key/value data storage and are referred to as "Data Store" (DS). The 2.x releases added a locking system enabling concurrent access to data. This is what is known as "Concurrent Data Store" (CDS). The 3.x releases added a logging system for transactions and recovery, called "Transactional Data Store" (TDS). The 4.x releases added the ability to replicate log records and create a distributed highly available single-master multi-replica database. This is called the "High Availability" (HA) feature set. Berkeley DB's evolution has sometimes led to minor API changes or log format changes, but very rarely have database formats changed. Berkeley DB HA supports online upgrades from one version to the next by maintaining the ability to read and apply the prior release's log records.

The FreeBSD and OpenBSD operating systems continue to use Berkeley DB 1.8x for compatibility reasons;[6] Linux-based operating systems commonly include several versions to accommodate for applications still using older interfaces/files.

Starting with the 6.0/12c releases, all Berkeley DB products are licensed under the GNU AGPL.[7] Up until then Berkeley DB was redistributed under the 4-clause BSD license (before version 2.0), and the Sleepycat Public License, which is an OSI-approved open-source license as well as an FSF-approved free software license.[8][9] The product ships with complete source code, build script, test suite, and documentation. The code quality and general utility along with the licensing terms have led to its use in a multitude of free and open-source software. Those who do not wish to abide by the terms of the GNU AGPL, or use an older version with the Sleepycat Public License, have the option of purchasing another proprietary license for redistribution from Oracle Corporation. This technique is called dual licensing.

Berkeley DB includes compatibility interfaces for some historic Unix database libraries: dbm, ndbm and hsearch (a System V library for creating in-memory hash tables).

9.5.2 Architecture

Berkeley DB has an architecture notably simpler than that of other database systems like relational database management systems. For example, like SQLite, it does not provide support for network access — programs access the database using in-process API calls. Oracle added support for SQL in 11g R2 release based on the popular SQLite API by including a version of SQLite in Berkeley DB.[10] There is third party support for PL/SQL in Berkeley DB via a commercial product named Metatranz StepSqlite.[11]

A program accessing the database is free to decide how the data is to be stored in a record. Berkeley DB puts no constraints on the record's data. The record and its key can both be up to four gigabytes long.

Despite having a simple architecture, Berkeley DB supports many advanced database features such as ACID transactions, fine-grained locking, hot backups and replication.

9.5.3 Editions

The name "Berkeley DB" is given to three different products:

1. Berkeley DB
2. Berkeley DB Java Edition
3. Berkeley DB XML

Each edition has separate database libraries, despite the common branding. The first is the traditional Berkeley DB, written in C. It contains several database implementations, including a B-Tree and one built around extendible hashing. It supports multiple language bindings, including C/C++, Java (via JNI), C# .NET, Perl and Python.

Berkeley DB Java Edition (JE) is a pure Java database management library. Its design resembles that of Berkeley DB without replicating it exactly, and has a feature set that includes many of those found in the traditional Berkeley DB and others that are specific to the Java Edition. It has a log structured storage architecture, which gives it different performance and concurrency characteristics. Three APIs are available—a Direct Persistence Layer which is "Plain Old Java Objects" (POJO); one which is based on the Java Collections Framework (an object persistence approach); and one based on the traditional Berkeley DB API. The Berkeley DB Java Edition High Availability option (Replication) is available. Note that traditional Berkeley DB also supports a Java API, but it does so via JNI and thus requires an installed native library.

The Berkeley DB XML database specializes in the storage of XML documents, supporting XQuery via XQilla. It is implemented as an additional layer on top of (a legacy version of) Berkeley DB and the Xerces library. DB XML is written in C++ and supports multiple language bindings, including C++, Java (via JNI), Perl and Python.

9.5.4 Programs that use Berkeley DB

Berkeley DB provides the underlying storage and retrieval system of several LDAP servers, database systems, and many other proprietary and free/open source applications. Notable software that use Berkeley DB for data storage include:

- 389 Directory Server - An open-source LDAP server from the Fedora Project.
- Bitcoin Core - The first implementation of the Bitcoin cryptocurrency.
- Bogofilter – A free/open source spam filter that saves its wordlists using Berkeley DB.
- Carbonado – An open source relational database access layer.
- Citadel – A free/open source groupware platform that keeps all of its data stores, including the message base, in Berkeley DB.
- Cyrus IMAP Server – A free/open source IMAP and POP3 server, developed by Carnegie Mellon University
- Evolution - A free/open source mail client; contacts are stored in addressbook.db using Berkeley DB
- GlusterFS - Prior to v3.4, GlusterFS included a BDB storage translator.
- Jabberd2 – An Extensible Messaging and Presence Protocol server
- KDevelop – An IDE for Linux and other Unix-like operating systems
- Movable Type (until version 4.0) – A proprietary weblog publishing system developed by California-based Six Apart
- memcachedb - A persistence-enabled variant of memcached
- MySQL database system – Prior to v5.1, MySQL included a BDB data storage backend.
- OpenLDAP – A free/open source implementation of the Lightweight Directory Access Protocol (LDAP)
- Oracle NoSQL - A NoSQL distributed key-value database
- Oracle Retail Predictive Application Server (RPAS) - RPAS (since 12.x?) uses Berkeley DB as the underlying persistence layer for the MOLAP engine used in several Oracle Retail Planning and Supply Chain products. (Berkeley DB replaced the previous Accumate/Acumen persistence layer used since the original development of the RPAS product by Neil Thall Associates, which was no longer supported by its final owner, Lucent and no longer sufficiently scalable).
- Postfix – A fast, secure, easy-to-administer MTA for Linux/Unix systems
- Parallel Virtual File System (PVFS) – A parallel file system for HPC clusters.[12]
- Red Dwarf - A server framework originally developed by Sun, now open sourced, commonly used for game development.
- RPM – The RPM Package Manager uses Berkeley DB to retain its internal database of packages installed on a system
- Sendmail - A popular MTA for Linux/Unix systems
- Spamassassin – An anti-spam application
- Subversion – A version control system designed specifically to replace CVS
- Sun Grid Engine – A free and open source distributed resource management system.

9.5.5 Licensing

Oracle Corporation makes versions 2.0 and higher of Berkeley DB available under a dual license.[13] The Sleepycat license is a 2-clause BSD license with an additional copyleft clause similar to the GNU GPL version 2's Section 3, requiring source code of an application using Berkeley DB to be made available for a nominal fee.

As of Berkeley DB release 6.0, the Oracle Corporation has relicensed Berkeley DB under the GNU AGPL v3.[14]

As of July 2011, Oracle's list price for non-copyleft Berkeley DB licenses varies between 900 and 13,800 USD per processor.[15] Embedded usage within the Oracle Retail Predictive Application Server (RPAS) does not require an additional license payment.

Sleepycat License

Sleepycat License (sometimes referred to as **Berkeley Database License** or the **Sleepycat Public License**) is an OSI-approved open source license used by Oracle Corporation for the open-source editions of Berkeley DB, Berkeley DB Java Edition and Berkeley DB XML embedded database products older than version 6.0. (Starting with version 6.0 the open-source editions are instead licensed under the GNU AGPL v3.) The name of this license is derived from the name of the company which commercially sold the Berkeley DB products, Sleepycat Software, which was acquired by Oracle in 2006. Oracle continued to use the name "Sleepycat License" for Berkeley DB, despite not using the term "Sleepycat" in any other documentation until it changed to GNU AGPL with version 6.

According to the Free Software Foundation,[16] it qualifies as a free software license, and is compatible with the GPL.

The license is a strong form of copyleft because it mandates that redistributions in any form not only include the source code of Berkeley DB, but also "any accompanying software that uses the DB software". It is possible to circumvent this strict licensing policy through the purchase of a commercial software license from Oracle Corporation consisting of terms and conditions which are negotiated at the time of sale. This is an example of dual licensing.

The effect of the dual license creates financial exposure for commercial organizations, since there is considerable risk of becoming liable for payment of license fees to Oracle. Some people consider it to be a "sneaky" license. Mike Olson, co-founder and CEO of Sleepycat Software and Cloudera, said that "This is good business if you can get it, but your relationship with your customer begins based on a threat and that's not a really healthy place to start out."[17]

9.5.6 References

[1] Berkeley DB Reference Guide: What is Berkeley DB not?. Doc.gnu-darwin.org (2001-05-31). Retrieved on 2013-09-18.

[2] http://doc.gnu-darwin.org/am_misc/dbsizes.html Berkeley DB Reference Guide: Database limits Retrieved on 2013-09-19

[3] Olson, Michael A.; Bostic, Keith; Seltzer, Margo (1999). "Berkeley DB" (PDF). *Proc. FREENIX Track, USENIX Annual Tech. Conf*. Retrieved October 20, 2009.

[4] Seltzer, Margo; Yigit, Ozan (1991). "A New Hashing Package for UNIX". *Proc. USENIX Winter Tech. Conf*. Retrieved October 20, 2009.

[5] Brunelli, Mark (March 28, 2005). "A Berkeley DB primer". *Enterprise Linux News*. Retrieved December 28, 2008.

[6] "db(3)". Retrieved April 12, 2009.

[7] [Berkeley DB Announce] Major Release: Berkeley DB 12gR1 (12.1.6.0). Retrieved July 5, 2013.

[8] "The Sleepycat License". Open Source Initiative. October 31, 2006. Retrieved December 28, 2008.

[9] "Licenses". Free Software Foundation. December 10, 2008. Retrieved December 28, 2008.

[10] "Twitter / Gregory Burd: @humanications We didn't r ...".

[11] "Official Berkeley DB FAQ". Oracle Corporation. Retrieved March 30, 2010. Does Berkeley DB support PL/SQL?

[12] RCE 35: PVFS Parallel Virtual FileSystem

[13] "Open Source License for Berkeley DB". Oracle Corporation. For a license to use the Berkeley DB software under conditions other than those described here, or to purchase support for this software, please contact berkeleydb-info_us@oracle.com.

[14] "Major Release: Berkeley DB 12gR1 (12.1.6.0)". June 10, 2013. Retrieved July 15, 2013.

[15] http://www.oracle.com/us/corporate/pricing/technology-price-list-070617.pdf

[16] Various Licenses and Comments about Them - Free Software Foundation

[17] Mike Olson (co-founder and CEO of Sleepycat Software and Cloudera), lecture to Stanford University entrepreneurship students, 2013.11.13

9.5.7 External links

- Oracle Berkeley DB Site
- Berkeley DB Programmer's Reference Guide
- Licensing pitfalls for Oracle Technology Products
- *The Berkeley DB Book* by Himanshu Yadava
- Launchpad.net - Berkeley DB at Launchpad
- Oracle Licensing Knowledge Net
- Oracle Berkeley DB Licensing Information
- Text of the Sleepycat License (old)

9.6 Memcached

Memcached (pronunciation: mem-cash-dee) is a general-purpose distributed memory caching system. It is often used to speed up dynamic database-driven websites by caching data and objects in RAM to reduce the number of times an external data source (such as a database or API) must be read.

Memcached is free and open-source software, licensed under the Revised BSD license.[2] Memcached runs on Unix-like operating systems (at least Linux and OS X) and on Microsoft Windows. It depends on the libevent library.

Memcached's APIs provide a very large hash table distributed across multiple machines. When the table is full, subsequent inserts cause older data to be purged in least recently used (LRU) order.[3][4] Applications using Memcached typically layer requests and additions into RAM before falling back on a slower backing store, such as a database.

The size of this hash table is often very large. It is limited to available memory across all the servers in the cluster of servers in a data centre. Where high volume, wide audience web publishing requires it, this may stretch to many gigabytes. Memcached can be equally valuable for situations where either the number of requests for content is high, or the cost of generating a particular piece of content is high.

Memcached was originally developed by Danga Interactive for LiveJournal, but is now used by many other systems, including MocoSpace,[5] YouTube,[6] Reddit,[7] Survata,[8] Zynga,[9] Facebook,[10][11][12] Orange,[13] Twitter,[14] Tumblr[15] and Wikipedia.[16] Engine Yard and Jelastic are using Memcached as part of their platform as a service technology stack[17][18] and Heroku offers several Memcached services[19] as part of their platform as a service. Google App Engine, AppScale, Microsoft Azure and Amazon Web Services also offer a Memcached service through an API.[20][21][22][23]

9.6.1 History

Memcached was first developed by Brad Fitzpatrick for his website LiveJournal, on May 22, 2003.[24][25][26] It was originally written in Perl, then later rewritten in C by Anatoly Vorobey, then employed by LiveJournal.[27]

9.6.2 Software architecture

The system uses a client–server architecture. The servers maintain a key–value associative array; the clients populate this array and query it by key. Keys are up to 250 bytes long and values can be at most 1 megabyte in size.

Clients use client-side libraries to contact the servers which, by default, expose their service at port 11211. Each client knows all servers; the servers do not communicate with each other. If a client wishes to set or read the value corresponding to a certain key, the client's library first computes a hash of the key to determine which server to use. Then it contacts that server. This gives a simple form of sharding and scalable shared-nothing architecture across the servers. The server computes a second hash of the key to determine where to store or read the corresponding value.

The servers keep the values in RAM; if a server runs out of RAM, it discards the oldest values. Therefore, clients must treat Memcached as a transitory cache; they cannot assume that data stored in Memcached is still there when they need it. Other databases, such as MemcacheDB, Couchbase Server, provide persistent storage while maintaining Memcached protocol compatibility.

If all client libraries use the same hashing algorithm to determine servers, then clients can read each other's cached data.

A typical deployment has several servers and many clients. However, it is possible to use Memcached on a single computer, acting simultaneously as client and server.

Security

Most deployments of Memcached are within trusted networks where clients may freely connect to any server. However, sometimes Memcached is deployed in untrusted networks or where administrators want to exercise control over the clients that are connecting. For this purpose Memcached can be compiled with optional SASL authentication support. The SASL support requires the binary protocol.

A presentation at BlackHat USA 2010 revealed that a number of large public websites had left Memcached open to inspection, analysis, retrieval, and modification of data.[28]

Even within a trusted organisation, the flat trust model of memcached may have security implications. For efficient simplicity, all Memcached operations are treated equally. Clients with a valid need for access to low-security entries within the cache gain access to *all* entries within the cache, even when these are higher-security and that client has no justifiable need for them. If the cache key can be either predicted, guessed or found by exhaustive searching, its cache entry may be retrieved.

Some attempt to isolate setting and reading data may be made in situations such as high volume web publishing. A farm of outward-facing content servers have *read* access to memcached containing published pages or page components, but no write access. Where new content is published (and is not yet in memcached), a request is instead sent to content generation servers that are not pub-

lically accessible to create the content unit and add it to memcached. The content server then retries to retrieve it and serve it outwards.

9.6.3 Example code

Note that all functions described on this page are pseudocode only. Memcached calls and programming languages may vary based on the API used.

Converting database or object creation queries to use Memcached is simple. Typically, when using straight database queries, example code would be as follows:

function get_foo(int userid) { data = db_select("SELECT * FROM users WHERE userid = ?", userid); return data; }

After conversion to Memcached, the same call might look like the following

function get_foo(int userid) { /* first try the cache */ data = memcached_fetch("userrow:" + userid); if (!data) { /* not found : request database */ data = db_select("SELECT * FROM users WHERE userid = ?", userid); /* then store in cache until next get */ memcached_add("userrow:" + userid, data); } return data; }

The client would first check whether a Memcached value with the unique key "userrow:userid" exists, where userid is some number. If the result does not exist, it would select from the database as usual, and set the unique key using the Memcached API add function call.

However, if only this API call were modified, the server would end up fetching incorrect data following any database update actions: the Memcached "view" of the data would become out of date. Therefore, in addition to creating an "add" call, an update call would also be needed using the Memcached set function.

function update_foo(int userid, string dbUpdateString) { /* first update database */ result = db_execute(dbUpdateString); if (result) { /* database update successful : fetch data to be stored in cache */ data = db_select("SELECT * FROM users WHERE userid = ?", userid); /* the previous line could also look like data = createDataFromDBString(dbUpdateString); */ /* then store in cache until next get */ memcached_set("userrow:" + userid, data); } }

This call would update the currently cached data to match the new data in the database, assuming the database query succeeds. An alternative approach would be to invalidate the cache with the Memcached delete function, so that subsequent fetches result in a cache miss. Similar action would need to be taken when database records were deleted, to maintain either a correct or incomplete cache.

9.6.4 See also

- Aerospike
- phpFastCache - Supported MemCached, MemCache, WinCache, APC and Files.
- Couchbase Server
- Redis
- Mnesia
- MemcacheDB
- MySQL - directly supports the Memcached API as of version 5.6.[29]
- Oracle Coherence - directly supports the Memcached API as of version 12.1.3.[30]
- GigaSpaces XAP - support Memcached with high availability, transaction support[31]
- Hazelcast
- Cassandra

9.6.5 References

[1] "Release notes for Release 1.4.22". Retrieved 2015-04-06.

[2] "Memcached license". GitHub. Retrieved 2014-06-27.

[3] "Memcached NewOverview".

[4] "Memcached NewUserInternals".

[5] MocoSpace Architecture - 3 Billion Mobile Page Views a Month. High Scalability (2010-05-03). Retrieved on 2013-09-18.

[6] Cuong Do Cuong (Engineering manager at YouTube/Google) (June 23, 2007). *Seattle Conference on Scalability: YouTube Scalability* (Online Video - 26th minute). Seattle: Google Tech Talks.

[7] Steve Huffman on Lessons Learned at Reddit

[8]

[9] How Zynga Survived FarmVille

[10] Facebook Developers Resources

[11] Scaling Memcached at Facebook

[12] NSDI '13: Scaling Memcache at Facebook

[13] Orange Developers

[14] It's Not Rocket Science, But It's Our Work

[15] Engineer â€“ Core Applications Group job at Tumblr in New York, NY, powered by JobScore. Jobscore.com. Retrieved on 2013-09-18.

[16] MediaWiki Memcached

[17] Engine Yard Technology Stack

[18] Jelastic Memcached System

[19] Heroku Memcached add-ons

[20] Using Memcache - Google App Engine - Google Code

[21] http://appscale.cs.ucsb.edu Memcached in AppScale

[22] About In-Role Cache for Windows Azure Cache. Msdn.microsoft.com. Retrieved on 2013-09-18.

[23] Amazon ElastiCache. Aws.amazon.com. Retrieved on 2013-09-18.

[24] changelog: livejournal. Community.livejournal.com (2003-05-22). Retrieved on 2013-09-18.

[25] brad's life - weather, running, distributed cache daemon. Brad.livejournal.com (2003-05-22). Retrieved on 2013-09-18.

[26] lj_dev: memcached. Community.livejournal.com (2003-05-27). Retrieved on 2013-09-18.

[27] lj_dev: memcached. Lj-dev.livejournal.com (2003-05-27). Retrieved on 2013-09-18.

[28] BlackHat Write-up: go-derper and mining memcaches

[29] "Speedy MySQL 5.6 takes aim at NoSQL, MariaDB."

[30]

[31]

9.6.6 External links

- Official website
- Memcached wiki and faq
- PHP Memcached Manager with Tag Support
- membase
- Memcached and Ruby
- go-memcached - Memcached implementation in Go
- QuickCached - Memcached server implementation in Java
- nsmemcache - memcache client for AOL Server
- Memcached implementation on Windows 8/8.1

Commercially supported distributions

- Couchbase Server (formerly Membase) offers a Memcached "bucket type" (free for use, subscription support available)
- GigaSpaces Java based Memcached (free community edition, fault tolerance)
- Hazelcast Memcached clustered, elastic, fault-tolerant, Java based Memcached (free for use, subscription support available)

9.7 BigTable

Bigtable is a compressed, high performance, and proprietary data storage system built on Google File System, Chubby Lock Service, SSTable (log-structured storage like LevelDB) and a few other Google technologies. On May 6, 2015, a public version of Bigtable was launched as **Google Cloud Bigtable**.[1] Bigtable also underlies Google Datastore,[2] which is available as a part of the Google Cloud Platform.

9.7.1 History

Bigtable development began in 2004[3] and is now used by a number of Google applications, such as web indexing,[4] MapReduce, which is often used for generating and modifying data stored in Bigtable,[5] Google Maps,[6] Google Book Search, "My Search History", Google Earth, Blogger.com, Google Code hosting, YouTube,[7] and Gmail.[8] Google's reasons for developing its own database include scalability and better control of performance characteristics.[9]

Google's Spanner RDBMS is layered on an implementation of Bigtable with a Paxos group for two-phase commits to each table. Google F1 was built using Spanner to replace an implementation based on MySQL.[10]

9.7.2 Design

Bigtable maps two arbitrary string values (row key and column key) and timestamp (hence three-dimensional mapping) into an associated arbitrary byte array. It is not a relational database and can be better defined as a sparse, distributed multi-dimensional sorted map.[11]:1 Bigtable is designed to scale into the petabyte range across "hundreds or thousands of machines, and to make it easy to add more machines [to] the system and automatically start taking advantage of those resources without any reconfiguration".[12]

Each table has multiple dimensions (one of which is a field for time, allowing for versioning and garbage collection). Tables are optimized for Google File System (GFS) by being split into multiple *tablets* – segments of the table are split along a row chosen such that the tablet will be ~200 megabytes in size. When sizes threaten to grow beyond a specified limit, the tablets are compressed using the algorithm BMDiff[13][14] and the Zippy compression algorithm[15] publicly known and open-sourced as Snappy,[16] which is a less space-optimal variation of LZ77 but more efficient in terms of computing time. The locations in the GFS of tablets are recorded as database entries in multiple special tablets, which are called "META1" tablets. META1 tablets are found by querying the single "META0" tablet, which typically resides on a server of its own since it is often queried by clients as to the location of the "META1" tablet which

itself has the answer to the question of where the actual data is located. Like GFS's master server, the META0 server is not generally a bottleneck since the processor time and bandwidth necessary to discover and transmit META1 locations is minimal and clients aggressively cache locations to minimize queries.

9.7.3 Other similar software

- Apache Accumulo — built on top of Hadoop, ZooKeeper, and Thrift. Has cell-level access labels and a server-side programming mechanism. Written in Java.
- Apache Cassandra — brings together Dynamo's fully distributed design and Bigtable's data model. Written in Java.
- Apache HBase — Provides Bigtable-like support on the Hadoop Core.[17] Has cell-level access labels and a server-side programming mechanism too. Written in Java.
- Hypertable — Hypertable is designed to manage the storage and processing of information on a large cluster of commodity servers.[18] Written in C++.
- "KDI", *Bluefish*, GitHub — Kosmix attempt to make a Bigtable clone. Written in C++.
- LevelDB — Google's embedded key/value store that uses similar design concepts as the Bigtable tablet.[19]

9.7.4 See also

- Amazon SimpleDB
- Big data
- Distributed data stores, an overview
- Dynamo (storage system)
- Column-oriented DBMS
- Hadoop

9.7.5 References

[1] http://googlecloudplatform.blogspot.com/2015/05/introducing-Google-Cloud-Bigtable.html

[2] http://googledevelopers.blogspot.com/2013/05/get-started-with-google-cloud-datastore.html

[3] Kumar, Aswini, Whitchcock, Andrew, ed., *Google's BigTable*, First an overview. BigTable has been in development since early 2004 and has been in active use for about eight months (about February 2005)..

[4] Chang, Fay; Dean, Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson C; Wallach, Deborah A; Burrows, Michael 'Mike'; Chandra, Tushar; Fikes, Andrew; Gruber, Robert E (2006), "Bigtable: A Distributed Storage System for Structured Data", *Research* (PDF), Google .

[5] Chang et al. 2006, p. 3: 'Bigtable can be used with MapReduce, a framework for running large-scale parallel computations developed at Google. We have written a set of wrappers that allow a Bigtable to be used both as an input source and as an output target for MapReduce jobs'

[6] Whitchcock, Andrew, *Google's BigTable*, There are currently around 100 cells for services such as Print, Search History, Maps, and Orkut.

[7] Cordes, Kyle (2007-07-12), *YouTube Scalability* (talk), Their new solution for thumbnails is to use Google's BigTable, which provides high performance for a large number of rows, fault tolerance, caching, etc. This is a nice (and rare?) example of actual synergy in an acquisition..

[8] "How Entities and Indexes are Stored", *Google App Engine*, Google Code.

[9] Chang et al. 2006, Conclusion: 'We have described Bigtable, a distributed system for storing structured data at Google... Our users like the performance and high availability provided by the Bigtable implementation, and that they can scale the capacity of their clusters by simply adding more machines to the system as their resource demands change over time... Finally, we have found that there are significant advantages to building our own storage solution at Google. We have gotten a substantial amount of flexibility from designing our own data model for Bigtable.'

[10] Shute, Jeffrey 'Jeff'; Oancea, Mircea; Ellner, Stephan; Handy, Benjamin 'Ben'; Rollins, Eric; Samwel, Bart; Vingralek, Radek; Whipkey, Chad; Chen, Xin; Jegerlehner, Beat; Littlefield, Kyle; Tong, Phoenix (2012), "Summary; F1 — the Fault-Tolerant Distributed RDBMS Supporting Google's Ad Business", *Research* (presentation), Sigmod: Google, p. 19, We've moved a large and critical application suite from MySQL to F1.

[11] Chang et al. 2006.

[12] "Google File System and BigTable", *Radar* (World Wide Web log), Database War Stories (7), O'Reilly, May 2006.

[13] "Google Bigtable, Compression, Zippy and BMDiff". 2008-10-12. Archived from the original on 1 May 2013. Retrieved 14 April 2015..

[14] McIlroy, Bentley. *Data compression using long common strings*. DCC '99. IEEE..

[15] "Google's Bigtable", *Outer court* (Weblog), 2005-10-23.

[16] "Snappy", *Code* (project), Google.

[17] "Background; HBase", *Hadoop Core* (wiki), Apache.

[18] "About", *Hyper table*.

[19] "Leveldb file layout and compactions", *Code*, Google.

9.7.6 Bibliography

- Chang, Fay; Dean, Jeffrey; Ghemawat, Sanjay; Hsieh, Wilson C; Wallach, Deborah A; Burrows, Michael 'Mike'; Chandra, Tushar; Fikes, Andrew; Gruber, Robert E (2006), "Bigtable: A Distributed Storage System for Structured Data", *Research* (PDF), Google .

9.7.7 External links

- *BigTable: A Distributed Structured Storage System*, Washington. *Video*, Google.
  - *UWTV* (video).
  - Witchcock, Andrew, *Google's BigTable* (notes on the official presentation).
- Carr, David F (2006-07-06), "How Google Works", *Baseline*.
- "Is the Relational Database Doomed?", *Read-write web*.

Chapter 10

Text and image sources, contributors, and licenses

10.1 Text

- **Database** *Source:* https://en.wikipedia.org/wiki/Database?oldid=693733035 *Contributors:* Paul Drye, NathanBeach, Dreamyshade, LC~enwiki, Robert Merkel, Zundark, The Anome, Stephen Gilbert, Sjc, Andre Engels, Chuckhoffmann, Fubar Obfusco, Ben-Zin~enwiki, Maury Markowitz, Waveguy, Imran, Leandrod, Stevertigo, Edward, Ubiquity, Michael Hardy, JeffreyYasskin, Fuzzie, Pnm, Ixfd64, TakuyaMurata, SebastianHelm, Pcb21, CesarB, MartinSpamer, ArnoLagrange, Ahoerstemeier, Haakon, Nanshu, Angela, Bogdangiusca, Cyan, Poor Yorick, Mxn, Mulad, Feedmecereal, Jay, Greenrd, Wik, DJ Clayworth, Tpbradbury, E23~enwiki, Furrykef, Morwen, Sandman~enwiki, Finlay McWalter, Jni, Chuunen Baka, Robbot, Noldoaran, Sander123, Craig Stuntz, Chrism, Chris 73, Vespristiano, Chocolateboy, Netizen, Nurg, Romanm, Lowellian, Pingveno, Tualha, Rursus, Rothwellisretarded, Jondel, TittoAssini, Hadal, Vikreykja, Mushroom, HaeB, Pengo, SpellBott, Tobias Bergemann, Stirling Newberry, Psb777, Giftlite, Graeme Bartlett, SamB, Sarchand~enwiki, Arved, Inter, Kenny sh, Levin, Peruvianllama, Everyking, Ciciban, Ssd, Niteowlneils, Namlemez, Mboverload, SWAdair, Bobblewik, Wmahan, Gadfium, SarekOfVulcan, Quadell, Kevins, Antandrus, Beland, OverlordQ, Rdsmith4, APH, Troels Arvin, Gscshoyru, Ohka-, Sonett72, Trevor MacInnis, Canterbury Tail, Bluemask, Zro, Grstain, Mike Rosoft, DanielCD, Shipmaster, EugeneZelenko, AnjaliSinha, KeyStroke, Discospinster, Rich Farmbrough, Rhobite, Lovelac7, Pak21, C12H22O11, Andrewferrier, Mumonkan, Kzzl, Paul August, Edgarde, Djordjes, S.K., Elwikipedista~enwiki, CanisRufus, *drew, MBisanz, Karmafist, Kiand, Cpereyra, Tom, Causa sui, Chrax, PatrikR, Hurricane111, Mike Schwartz, Smalljim, Wipe, John Vandenberg, Polluks, Ejrrjs, JeffTan, Nk, Franl, Alphax, Railgun, Sleske, Sam Korn, Nsaa, Mdd, HasharBot~enwiki, Jumbuck, Storm Rider, Alansohn, Tablizer, Etxrge, Guy Harris, Arthena, Keenan Pepper, Ricky81682, Riana, Aza-Toth, Zippanova, Kocio, PaePae, Velella, Skybrian, Helixblue, Filx, Frankman, Danhash, Max Naylor, Harej, Mathewforyou, W mccall, Ringbang, Chirpy, Djsasso, Dan100, Brookie, Isfisk, Marasmusine, Simetrical, Reinoutr, Woohookitty, Mindmatrix, Camw, Arcann, 25or6to4, Decrease789, Mazca, Pol098, Commander Keane, Windsok, Ruud Koot, Tabletop, Bbatsell, KingsleyIdehen, DeirdreGerhardt, GregorB, AnmaFinotera, Plrk, Crucis, Prashanthns, TrentonLipscomb, Turnstep, PeregrineAY, Dysepsion, Mandarax, Wulfila, MassGalactusUniversum, Graham87, Qwertyus, DePiep, Jclemens, Sjakkalle, Rjwilmsi, Koavf, DeadlyAssassin, Vary, Carbonite, GlenPeterson, Feydey, Eric Burnett, Jb-adder, ElKevbo, The wub, Sango123, FlaBot, Doc glasgow, Latka, GnuDoyng, Jstaniek, RexNL, AndriuZ, Intgr, Antrax, Ahunt, Imnotminkus, JonathanFreed, King of Hearts, Chobot, Visor, Phearlez, DVdm, Bkhouser, NSR, Cornellrockey, Rimonu, YurikBot, Wavelength, Sceptre, JarrahTree, Phantomsteve, Michael Slone, Woseph, Fabartus, Toquinha, GLaDOS, SpuriousQ, RadioFan2 (usurped), Akamad, Stephenb, Rsrikanth05, Cryptic, Cpuwhiz11, BobStepno, Wimt, SamJohnston, RadioKirk, NawlinWiki, Wiki alf, Jonathan Webley, Jaxl, Milo99, Welsh, Joel7687, SAE1962, Journalist, Nick, Aaron Brenneman, RayMetz100, Matticus78, Larsinio, Mikeblas, Ezeu, Zwobot, Supten, Dbfirs, JMRyan, Bluerocket, LindaEllen, Samir, DeadEyeArrow, Werdna, User27091, Mugunth Kumar, SimonMorgan, Lod, Twelvethirteen, Deville, Theodolite, Zzuuzz, Mike Dillon, Closedmouth, Arthur Rubin, Fang Aili, Th1rt3en, GraemeL, JoanneB, Alasdair, Echartre, JLaTondre, ArielGold, Stuhacking, Kungfuadam, Mhkay, Bernd in Japan, GrinBot~enwiki, DVD R W, Jonearles, CIreland, Victor falk, Pillefj, SmackBot, Hydrogen Iodide, McGeddon, WikiuserNI, Unyoyega, Pgk, AnonUser, Davewild, AutumnSnow, Brick Thrower, Stifle, Jab843, PJM, Kslays, Edgar181, Lexo, David Fuchs, Siebren, Yamaguchi??, Gilliam, Donama, Ohnoitsjamie, Chaojoker, Chris the speller, TimBentley, MikeSy, Thumperward, Nafclark, Oli Filth, MalafayaBot, Silly rabbit, Robocoder, Xx236, Deli nk, Jerome Charles Potts, Baa, Robth, DHN-bot~enwiki, Methnor, Colonies Chris, Darth Panda, Can't sleep, clown will eat me, Frap, Chlewbot, Paul E Ester, Edivorce, Allan McInnes, Pax85, Mugaliens, Khoikhoi, COMPFUNK2, Soosed, Cybercobra, Jwy, Jdlambert, Dreadstar, Insineratehymn, Hgilbert, BryanG, Ultraexactzz, RayGates, Daniel.Cardenas, Kukini, Kkailas, SashatoBot, Krashlandon, Jasimab, Srikeit, Kuru, Jonwynne, Microchip08, Tazmaniacs, Gobonobo, PeeAeMKay, Sir Nicholas de Mimsy-Porpington, Lguzenda, Tim Q. Wells, Minna Sora no Shita, Joffeloff, HeliXx, IronGargoyle, 16@r, MarkSutton, Slakr, Tasc, Beetstra, Noah Salzman, Wikidrone, Babbling.Brook, Childzy, Optakeover, Waggers, Ryulong, ThumbFinger, DougBarry, Asyndeton, Dead3y3, Iridescent, Mrozlog, TwistOfCain, Paul Foxworthy, Igoldste, Benni39, Dwolt, DEddy, Courcelles, Linkspamremover, Navabromberger, Dkastner, Tawkerbot2, Flubeca, LessHeard vanU, Megatronium, FatalError, JForget, Comps, VoxLuna, Spdegabrielle, Thatperson, Ahy1, CmdrObot, Ale jrb, Ericlaw02, Iced Kola, KyraVixen, Kushal one, GHe, Constructive, Dgw, Argon233, FlyingToaster, Moreschi, Sewebster, Simeon, Joshnpowell, Ubiq, Cantras, Mato, Gogo Dodo, Parzi, Chasingsol, Pascal.Tesson, Dancter, SymlynX, Tawkerbot4, Shirulashem, DumbBOT, Chrislk02, Alaibot, IComputerSaysNo, SpK, Omicronpersei8, UberScienceNerd, Cavanagh, Click23, Mattisse, Thijs!bot, Epbr123, Qwyrxian, HappyInGeneral, Andyjsmith, CynicalMe, Mojo Hand, Philippe, Eric3K, Peashy, Maxferrario, Mentifisto, AntiVandalBot, Majorly, Luna Santin, Widefox, Seaphoto, Turlo Lomon, MrNoblet, EarthPerson, Kbthompson, Credema, Spartaz, Lfstevens, Deadbeef, JAnDbot, Eric Bekins, MER-C, BlindEagle, The Transhumanist, Blood Red Sandman, RIH-V, Andonic, PhilKnight, Saiken79, LittleOldMe, Jdrumgoole, Magioladitis, Karlhahn, Bongwarrior, VoABot II, Hasek is the best, JamesBWatson, Think outside the box, Lucyin, Twsx, WODUP, Cic, Jvhertum, Bubba hotep, Culverin, Danieljamesscott, Adrian J. Hunter, 28421u2232nfenfcenc, Stdazi, Wwmbes, Cpl

Syx, Kunaldeo, Kozmando, Chris G, DerHexer, JaGa, Ahodgkinson, Oroso, Leaderofearth, MartinBot, Ironman5247, Arjun01, NAHID, Poeloq, CableCat, Rettetast, R'n'B, NetManage, Tgeairn, J.delanoy, Pharaoh of the Wizards, Trusilver, Rohitj.iitk, Bogey97, Ayecee, Uncle Dick, Maurice Carbonaro, Jesant13, Ginsengbomb, Darth Mike, Gzkn, Bcartolo, BrokenSphere, Katalaveno, Afluegel, Chriswiki, Damn-Randall, Girl2k, NewEnglandYankee, SJP, Gregfitzy, Kraftlos, Madth3, Madhava 1947, Jackacon, Juliancolton, Cometstyles, Ryager, Raspalchima, Seanust 1, Lamp90, Bonadea, Aditya gopal3, Pdcook, Ja 62, TheNewPhobia, DigitalEnthusiast, Squids and Chips, Cardinal-Dan, Ryanslater, Ryanslater2, Siteobserver, Lights, VolkovBot, Amaraiel, Thedjatclubrock, Alain Amiouni, Indubitably, JustinHagstrom, WOSlinker, Barneca, N25696, Erikrj, Philip Trueman, Lingwitt, TXiKiBoT, Wiki tiki tr, Moogwrench, Vipinhari, Technopat, Caster23, GDonato, ScMeGr, Olinga, Ann Stouter, Anonymous Dissident, Cyberjoac, Qxz, Gozzy345, Lradrama, Sintaku, Clarince63, Twebby, BwDraco, Jackfork, LeaveSleaves, Wya 7890, Mannafredo, Amd628, Zhenqinli, Hankhuck, Gwizard, Synthebot, Kingius, Bblank, Why Not A Duck, Atkinsdc, Pjoef, Aepanico, Logan, HybridBoy, Thehulkmonster, D. Recorder, Calutuigor, SieBot, Fooker69, Calliopejen1, Praba tuty, Kimera Kat, Jauerback, LeeHam2007, Caltas, Eagleal, Triwbe, Yintan, TalkyLemon, Keilana, Bentogoa, Flyer22 Reborn, Radon210, Oda Mari, JCLately, Jojalozzo, Hxhbot, Le Pied-bot~enwiki, Sucker666, Theory of deadman, KoshVorlon, 10285658sd-saa, Mkeranat, Fratrep, Macy, ChorizoLasagna, Autumn Wind, Maxime.Debosschere, Spazure, Paulinho28, Vanished User 8902317830, G12kid, Pinkadelica, Treekids, Denisarona, Escape Orbit, Levin Carsten, Kanonkas, VanishedUser sdu9aya9fs787sads, Explicit, Beeblebrox, ClueBot, Frsparrow, Phoenix-wiki, Hugsandy, Strongsauce, Avenged Eightfold, The Thing That Should Not Be, Buzzimu, Jan1nad, Poterxu, Supertouch, Unbuttered Parsnip, Garyzx, Zipircik, SuperHamster, Boing! said Zebedee, Doddsy1993, Niceguyedc, Sam Barsoom, Blanchardb, Dylan620, Mikey180791, Puchiko, Mspraveen, Vivacewwxu~enwiki, Veryprettyfish, Robert Skyhawk, Drumroll99, Excirial, Pumpmeup, M4gnum0n, Dbates1999, LaosLos, Northernhenge, Eeekster, Tyler, Odavy, Cenarium, Lunchscale, Peter.C, Jotterbot, MECiAf., Huntthetroll, Tictacsir, Thehelpfulone, Inspector 34, Thingg, Aitias, DerBorg, Subash.chandran007, Versus22, Burner0718, Johnuniq, SoxBot III, Apparition11, DumZiBoT, Jmanigold, XLinkBot, EastTN, Gsallis, PrePress, Avoided, Pee Tern, Galzigler, Noctibus, Qwertykris, Dsimic, Osarius, HexaChord, Rakeki, Addbot, Proofreader77, Pyfan, Willking1979, Some jerk on the Internet, Betterusername, Non-dropframe, Captain-tucker, Ngpd, Fgnievinski, Fieldday-sunday, JephapE, Xhelllox, Vishnava, CanadianLinuxUser, Fluffernutter, Cevalsi, Cambalachero, CarsracBot, DFS454, Glane23, FiriBot, SDSWIKI, Roux, Favonian, Doniago, Exor674, AtheWeatherman, Jasper Deng, Hotstaff, Evildeathmath, Tide rolls, Nicoosuna, Kivar2, Matěj Grabovský, Dart88, Gail, David0811, Duyanfang, Jarble, Arbitrarily0, LuK3, Informatwr, Ben Ben, Luckas-bot, Yobot, Sudarevic, 2D, OrgasGirl, Bunnyhop11, Fraggle81, Gishac, MarcoAurelio, Pvjohnson, Nallimbot, SwisterTwister, Srdju001, Peter Flass, Bbb23, N1RK4UDSK714, AnomieBOT, AmritasyaPutra, Rubinbot, Sonia, Jim1138, JackieBot, Piano non troppo, Kingpin13, Ulric1313, Imfargo, Flewis, Bluerasberry, Materialscientist, Kimsey0, Citation bot, OllieFury, BlurTento, Clark89, Darthvader023, Xqbot, Anders Torlind, Kimberly ayoma, Sythy2, Llyntegid, Addihockey10, Capricorn42, Bcontins, 4twenty42o, Craftyminion, Grim23, Yossman007, Preet91119, Tonydent, GrouchoBot, Call me Bubba, Kekekecakes, Bjcubsfan, Earlypsychosis, Prunesqualer, Crashdoom, Amaury, Doulos Christos, Sophus Bie, The Wiki Octopus, IElonex!, Shadowjams, М И Ф, Oho1, Dougofborg, Al Wiseman, Chtuw, Captain-n00dle, Gonfus, Prari, FrescoBot, Sock, Riverraisin, Fortdj33, Blackguard SF, Dogposter, Mark Renier, StaticVision, HJ Mitchell, Sae1962, Wifione, Weetoddid, ZenerV, Kwiki, Javert, ZooPro, Winterst, Shadowseas, Pinethicket, I dream of horses, HRoestBot, Grsmca, LittleWink, 10metreh, Supreme Deliciousness, Hamtechperson, Sissi's bd, 28nebraska, Jschnur, Xfact, RedBot, Btilm, MastiBot, Rotanagol, Bharath357, Σ, 05winsjp, Psaajid, Meaghan, Abhikumar1995, Jandalhandler, Refactored, FoxBot, TobeBot, Mercy11, كاشف عق‌ی‌ل, KotetsuKat, ItsZippy, Lotje, Callanecc, Writeread82, Vrenator, Reidh21234, Reaper Eternal, Luizfsc, TheGrimReaper NS, Xin0427, Suffusion of Yellow, SnoFox, BluCreator, Colindolly, TheMesquito, Minimac, Thinktdub, Heysim0n, RazorXX8, DARTH SIDIOUS 2, Lingliu07, KILLERKEA23, Onel5969, Leonnicholls07, Mean as custard, Helloher, ArwinJ, Kvasilev, Regancy42, FetchcommsAWB, Timbits82, Aj.robin, Salvio giuliano, Skamecrazy123, Rollins83, EmausBot, John of Reading, FFGeyer, Armen1304, Heymid, ScottyBerg, Lflores92201, Beta M, Dewritech, GoingBatty, RA0808, RenamedUser01302013, Itafran2010, Knbanker, Winner 42, Carbo1200, Wikipelli, K6ka, Sheeana, Ceyjan, Serketan, AsceticRose, Anirudh Emani, Tudorol, Komal.Ar, Pete1248, Savh, Ravinjit, Joshwa1234567890, Fæ, NicatronTg, M.badnjki, Alpha Quadrant (alt), Tuhl, Makecat, Ocaasi, OnePt618, Tolly4bolly, W163, TyA, L Kensington, Mayur, Donner60, Mentibot, MainFrame, RockMagnetist, Nz101, Matthewrbowker, Peter Karlsen, GregWPhoto, GrayFullbuster, Rishu arora11, DASHBotAV, Kellyk99, 28bot, Rocketrod1960, Diamondland, ClueBot NG, SpikeTorontoRCP, Mechanical digger, Jack Greenmaven, MelbourneStar, Satellizer, Dancayta, Chester Markel, Name Omitted, Bwhynot14, Millermk, Theimmaculatechemist, Lsschwar, Bowlderizer, Zhoravdb, Widr, Danim, Ugebgroup8, CasualVisitor, Vibhijain, Franky21, Jk2q3jrklse, Cammo33, Oddbodz, Lbausalop, Cambapp, Strike Eagle, Calabe1992, Doorknob747, Lowercase sigmabot, BG19bot, Freebiekr, MilerWhite, Machdohvah, Tomatronster, Northamerica1000, Wiki13, MusikAnimal, Frze, Dan653, AwamerT, Allecher, Mark Arsten, Somchai1029, Vincent Liu, Compfreak7, 110808028 amol, Altaïr, Foxfax555, Rj Haseeb, Alzpp, Bfugett, Jbrune, Thomasryno, Afree10, Glacialfox, Rowan Adams, Admfirepanther, Era7bd, Soumark, Maxmarengo, Manikandan 2030, Branzman, Melucky2getu, Fylbecatulous, Plavozont, Carliitaeliza, IkamusumeFan, Several Pending, Pratyya Ghosh, Zhaofeng Li, Mrt3366, VNeumann, ChrisGualtieri, Christophe.billiottet, Lovefamosos, Maty18, Mediran, Khazar2, Deathlasersonline, Saturdayswiki, ???, Codename Lisa, Mukherjeeassociates, Cerabot~enwiki, Malvikiran, Cheolsoo, R3miixasim, Pebau.grandauer, TwoTwoHello, Lugia2453, Frosty, SFK2, Graphium, Rafaelschp, 069952497a, Reatlas, Phamnhatkhanh, Epicgenius, P2Peter, Acetotyce, Rockonomics, Eyesnore, Moazzam chand, JamesMoose, Jabby11, EvergreenFir, Menublogger, Backendgaming, PappaAvMin, Mike99999, Gburd, Babitaarora, MJunkCat, Boli1107, JJdaboss, Ray Lightyear, BentlijDB, Hshoemark, Melody Lavender, Ginsuloft, D Eaketts, Eddiecarter1, Eddiejcarter, Mwaci11, Gajurahman, Manul, IrfanSha, AddWittyNameHere, Dkwebsub, JunWan, WPGA2345, Verajohne, Phinicle, Title302, JaconaFrere, ElijahLloyd97, Suelru, 7Sidz, Monkbot, JewishMonser69, Rajat Kant Singh, Davidcopperfield123, Sunrocket89, Nomonomnom, Samster0708, Krushna124, Cabral88, MisteArndon, KizzyCode, Uoy ylgu dratsab, Hillysilly, FSahar, Thedinesh4u, Boybudz321, Jesseminisis, ChamithN, Crystallizedcarbon, Eurodyne, JensLechtenboerger, Papapasan, Is8ac, Torvolt, Rgeurts, DiscantX, Maurolepis-Dreki, Top The Ball, Godfarther48, Jack0898, Rob12467, Gfsfg, Uthinkurspecial, Asdafsd, KasparBot, SaltySloth, Jrgreene2, Timoutiwin, Smedley Rhyse-Frockmorton, Communal t, BadSprad, AldrinRemoto, Vinurarulz, Yeshii 909, Cani talk and Anonymous: 2088

- **Schema migration** *Source:* https://en.wikipedia.org/wiki/Schema_migration?oldid=687610122 *Contributors:* Wtmitchell, Malcolma, Racklever, Hu12, Paling Alchemist, AnomieBOT, Digulla, Billegge, LilHelpa, Diroussel, Snotbot, NietzscheSpeaks, Wokspoon, Andifalk, Axel.fontaine, Ebr.wolff, Dvdtknsn, JonRou, Bread2000, Gregoriomelo and Anonymous: 17
- **Star schema** *Source:* https://en.wikipedia.org/wiki/Star_schema?oldid=676280147 *Contributors:* Jketola, Random832, Jay, 1984, Remy B, Pne, Dfrankow, Beland, Asqueella, KeyStroke, Appi~enwiki, .:Ajvol:., Gothick, Diego Moya, Andrewpmk, GregorB, DePiep, Birger~enwiki, Chobot, YurikBot, Chrissi~enwiki, Od Mishehu, Vald, Mselway, Chronodm, Gilliam, Bluebot, Cralize, OrphanBot, RayGates, Michael miceli, Budyhead, JHunterJ, Bertport, Thesuperav, SqlPac, CWY2190, NishithSingh, Electrum, Kablammo, Armchairlinguist, Mwarren us, Littldo, Falcor84, Raymondwinn, Panfakes, Flyer22 Reborn, ClueBot, AndrewMWebster, Aitias, Addbot, Elsendero, Luckas-bot, Yobot, Fraggle81, JackieBot, Materialscientist, Mark Renier, D'ohBot, Crysb, Gahooa, EmausBot, Txnate, ClueBot NG, Gilderien, Ozancan, Mrityu411989, Sharad.sangle, Helpful Pixie Bot, BG19bot, Walk&check, Sitoiganap, Anubhab91, Jobin RV, Millertimebjm, Nrahimian, ChrisGualtieri, Abergquist, Surendra.konathala, Ginsuloft, Amattas, Andrew not the saint and Anonymous: 121
- **CAP** *Source:* https://en.wikipedia.org/wiki/CAP?oldid=662445807 *Contributors:* Gtrmp, RHaworth, BD2412, Wavelength, N35w101,

Capmo, Clarityfiend, Connermcd, HelenOnline, Krassotkin, Brycehughes, ClueBot NG, Mark Arsten, Oranjblud, Gnasby, Alexwho314 and Anonymous: 6

- **Eventual consistency** *Source:* https://en.wikipedia.org/wiki/Eventual_consistency?oldid=682273269 *Contributors:* The Anome, Alaric, Charles Matthews, Finlay McWalter, Ruinia, Rich Farmbrough, Dodiad, Intgr, Emersoni, Fang Aili, Mshiltonj, SmackBot, Frap, Jon-Harder, Kinotgell, Gpierre, Gregbard, Cydebot, Kovan, Momo54, Duncan.Hull, Gdupont, Siskus, Morninj, Mild Bill Hiccup, M4gnum0n, DumZiBoT, Addbot, Twimoki, Yobot, AnomieBOT, Materialscientist, Ms.wiki.us, Theclapp, DataWraith, PeaceLoveHarmony, Erik9bot, Skcpublic, Sae1962, Winterst, RedBot, GoingBatty, Mspreitz, ZéroBot, Richnice, Rob7139, ClueBot NG, BG19bot, SAuhsoj, Andrew Helwer, APerson, Dexbot, Datamaniac, Pbailis and Anonymous: 23
- **Object-relational impedance mismatch** *Source:* https://en.wikipedia.org/wiki/Object-relational_impedance_mismatch?oldid=680987835 *Contributors:* Leandrod, GCarty, Rbraunwa, Morven, Topbanana, Jeffq, Craig Stuntz, Rursus, Ambarish, Bkonrad, Esap, Jpp, SarekOfVulcan, Rich Farmbrough, Mike Schwartz, Mojo~enwiki, Pearle, Merenta, Tablizer, Diego Moya, Ruud Koot, Triddle, Msiddalingaiah, Rjwilmsi, JubalHarshaw, MarSch, Salix alba, Dmccreary, Hairy Dude, Allister MacLeod, Big Brother 1984, EngineerScotty, Grafen, ZacBowling, SAE1962, Ospalh, Scope creep, BazookaJoe, Fram, Draicone, Erik Postma, SmackBot, Brick Thrower, Chris the speller, Thumperward, Jerome Charles Potts, Colonies Chris, Frap, Cybercobra, Warren, Zsvedic, Wickethewok, Larrymcp, Hu12, CmdrObot, Jiminez, Arnonf, Pingku, Underpants, PKT, Towopedia, Ideogram, Mentifisto, Magioladitis, Dbasch, Joshua Davis, Rustyfence, STBot, J.delanoy, Cantonnier, Q Chris, Andy Dingley, Prakash Nadkarni, Creative1985, M4gnum0n, Aprock, Addbot, Sbhug1, N8allan, Agomulka, AnomieBOT, Roux-HG, Metafax1, Mark Renier, OldTownIT, GoingBatty, Rdmil, ClueBot NG, Shaddim, Widr, Kcragin, Danim, MusikAnimal, Stelpa, Thesquaregroot, BattyBot, Fraulein451, Lesser Cartographies, Alexadamson and Anonymous: 83
- **Object database** *Source:* https://en.wikipedia.org/wiki/Object_database?oldid=678990087 *Contributors:* Vtan, Hari, Ben-Zin~enwiki, Maury Markowitz, Leandrod, Stevertigo, W~enwiki, Modster, Kku, Pcb21, CesarB, Hofoen, Ronz, Rednblu, Furrykef, Robbot, Noldoaran, BenFrantzDale, Beardo, Mckaysalisbury, Gadfium, Beland, SimonArlott, Sam Hocevar, Indolering, Usrnme h8er, Klemen Kocjancic, Tordek ar, Pavel Vozenilek, Elwikipedista~enwiki, Enric Naval, Ejrrjs, Mdd, Tablizer, Zippanova, Rickyp, SteinbDJ, Voxadam, Forderud, Karnesky, Mindmatrix, Dandv, Timosa, Ruud Koot, JIP, Icey, Jivecat, Dmccreary, FlaBot, SchuminWeb, Margosbot~enwiki, Intgr, Bgwhite, YurikBot, Foxxygirltamara, Hydrargyrum, ENeville, Oberst, SAE1962, Larsinio, Voidxor, Grafikm fr, BOT-Superzerocool, Ott2, Sandstein, BSTRhino, Talyian, Wainstead, Mhkay, Mlibby, Dybdahl, SmackBot, Pintman, Kellen, Reedy, OEP, AutumnSnow, Commander Keane bot, Bluebot, Nkaku, MyNameIsVlad, RProgrammer, Turbothy, Soumyasch, Lguzenda, IronGargoyle, RichardF, DougBarry, Hu12, Britannica~enwiki, Tawkerbot2, DBooth, FatalError, LGuzenda, Ervinn, Shreyasjoshis, Singerboi22, Yaris678, Corpx, Sestoft, Torc2, Charwing, Nick Number, Mvjs, Spencer, MER-C, Cameltrader, Jbom1, Magioladitis, Hroðulf, Soulbot, 28421u2232nfenfcenc, Gwern, Wixardy, J.delanoy, Saifali1, Kozka, Tagus, VolkovBot, Rei-bot, WikipedianYknOK, Dawn Bard, JCLately, SmallRepair, Edlich, PsyberS, VanishedUser sdu9aya9fs787sads, Dinojc, ClueBot, The Thing That Should Not Be, EoGuy, Alexbot, Eeekster, Sun Creator, Lucpeuvrier~enwiki, EastTN, HarlandQPitt, Zeliboba7, Addbot, Download, Kngspook, Lightbot, Jackelfive, Yobot, Bunnyhop11, Fraggle81, Waynenilsen, AnomieBOT, Materialscientist, Xqbot, Mika au, Addbc, Garyaj, Pwwamic, FrescoBot, Sulfsby, Mark Renier, Bablind, ITOnTheMind, Shewizzy2005, Cari.tenang, Maria Johansson, Alexandre.Morgaut, Gf uip, EmausBot, John of Reading, WikitanvirBot, Booler80, Minimac's Clone, Phiarc, SvetCo, Germanviscuso, ClueBot NG, Ki2010, Danim, Secured128, Razorbliss, Compfreak7, Pradiq009, Matspca, Snow Blizzard, Osiris, Eduardofeld, Khiladi 2010, Cyberbot II, FlyingPhysicist, André Miranda Moreira, DallasClarke, Rzicari, Monkbot and Anonymous: 210
- **NoSQL** *Source:* https://en.wikipedia.org/wiki/NoSQL?oldid=692912453 *Contributors:* AxelBoldt, Maury Markowitz, Jose Icaza, Pnm, Kku, Komap, Phoe6, Ronz, Ntoll, Ehn, Timwi, Furrykef, Phil Boswell, Bearcat, Peak, Dilbert, (:Julien:), Tagishsimon, Gadfium, Coldacid, Alexf, Beland, Euphoria, Clemwang, Rfl, MMSequeira, Smyth, Leigh Honeywell, Russss, Stephen Bain, Thüringer, Walter Görlitz, Markito~enwiki, Bhaskar, PatrickFisher, YPavan, Eno~enwiki, Crosbiesmith, Marasmusine, Woohookitty, Linas, Tshanky, Barrylb, Dm~enwiki, Tabletop, MacTed, Nileshbansal, BD2412, Qwertyus, Koavf, Ceefour, Strait, Amire80, Seraphimblade, ErikHaugen, Professionalsql, Vegaswikian, Jubalkessler, ElKevbo, Dmccreary, AlisonW, RobertG, Sstrader, Intgr, Tedder, Benbovee, Wavelength, Hairy Dude, Bovineone, Morphh, SamJohnston, Mbonaci, Rjlabs, Leotohill, Poohneat, GraemeL, Volt42, HereToHelp, Jonasfagundes, JLaTondre, Shepard, Matt Heard, Benhoyt, A bit iffy, SmackBot, Fulldecent, Anastrophe, Mauls, Drttm, Gorman, Somewherepurple, KiloByte, Thumperward, Jstplace, Jerome Charles Potts, Милан Јелисавчић, Frap, DavidSol, Cybercobra, Plustgarten, Looris, ThomasMueller, Trbdavies, NickPenguin, Eedeebee, ThurnerRupert, Petr Kopač, Zaxius, Lguzenda, Heelmijnlevenlang, Omidnoorani, Mauro Bieg, Benatkin, Mjresin, Hu12, Charbelgereige, Dancrumb, Gpierre, Arto B, Raysonho, Sanspeur, Ostrolphant, ProfessorBaltasar, Netmesh, OmerMor, Neustradamus, ColdShine, Mydoghasworms, Viper007Bond, Headbomb, CharlesHoffman, Peter Gulutzan, Davidhorman, Philu, Bramante~enwiki, Nick Number, Sorenriise, Polymorph self, Widefox, QuiteUnusual, Replizwank, Lfstevens, Gstein, Syaskin, Dericofilho, Joolean, Orenfalkowitz, Kunaldeo, Kgfleischmann, Philg88, Mitpradeep, Adtadt, GimliDotNet, Lmxspice, Stimpy77, Mikek999, DatabACE, JohnPritchard, Ansh.prat, McSly, Atropos235, Lamp90, Jottinger, Anoop K Nayak, Bbulkow, Tonyrogerson, Robert1947, Rogerdpack, Billinghurst, Quiark, Kbrose, ManikSurtani, TJRC, Dawn Bard, Whimsley, DavidBourguignon, Flyer22 Reborn, Hello71, Ctxppc, Mesut.ayata, Legacypath, AndrewBass, Edlich, Drq123, CaptTofu, Stevedekorte, Rossturk, Niceguyedc, Cnorvell, Pointillist, Excirial, Zapher67, PixelBot, Dredwolff, Robhughadams, Arjayay, Razorflame, StanContributor, Irmatov, Shijucv, The-verver, Tgrall, Miami33139, XLinkBot, Phoenix720, Duncan, Fiskbil, Whooym, Techsaint, Addbot, Fmorstatter, Mortense, Mabdul, MrOllie, LaaknorBot, Chrismcnab, Alexrakia, Getmoreatp, Luckas-bot, Yobot, Amirobot, Pcap, Ebalter, Ma7dy, AnomieBOT, Angry bee, Fraktalek, White gecko, Materialscientist, Xtremejames183, Cyril Wack, Jabawack81, El33th4x0r, Gkorland, Tomdo08, Ubcule, ChristianGruen, FontOfSomeKnowledge, Rtweed1955, Omnipaedista, Sduplooy, Shadowjams, Ciges, Cekli829, Sdrkyj, FrescoBot, Nawroth, Ashtango, Sae1962, Thegreeneman5, David Paniz, Ertugka, Chenopodiaceous, Winterst, I dream of horses, Leegee23, Hoo man, Natishalom, Michael Minh, Seancribbs, Jandalhandler, Craigbeveridge, Cnwilliams, Colemala, Argv0, Justinsheehy, AdityaKishore, Javalangstring, Voodootikigod, JnRouvignac, Svesterli, Violaaa, Hoelzro, Magnuschr, Extrovrt101, Wyverald, Jeffdexter77, Uhbif19, Zond, Asafdapper, Ptab, Tobiasivarsson, Alexandre.Morgaut, Steve03Mills, Phunehehe, R39132, EmausBot, Biofinderplus, WikitanvirBot, FalseAxiom, Bdijkstra, Dewritech, GoingBatty, RA0808, Ledhed2222, MrWerewolf, EricBloch, Hloeung, ZéroBot, Weimanm, Al3xpopescu, Theandrewdavis, Mhegi, Sagarjhobalia, Mtrencseni, Phillips-Martin, Dmitri.grigoriev, H3llBot, Jnaranjo86, DamarisC, Dstainer, Bulwersator, Eco schranzer, Thomas.uhl, Lyoshenka, Inmortalnet, Really Enthusiastic, Germanviscuso, Stephen E Browne, ClueBot NG, Rabihnassar, Ki2010, Randl, Luisramos22, Fxsjy, Korrawit, Tylerskf, Castncoot, ScottConroy, Jrudisin, Mshefer, Ashtango5, Helpful Pixie Bot, Pereb, William greenly, Rpk512, GlobalsDB, DBigXray, Tuvrotya, BG19bot, Nawk, Gonim, Freshnfruity, Vychtrle, Gaborcselle, Kkbhumana, Frze, AvocatoBot, Mark Arsten, Compfreak7, Anne.naimoli, Matspca, Boshomi, Fceller, Dshelby, Brocsima, BigButterfly, Winston Chuen-Shih Yang, Griswolf, Socialuser, Ugurbost, BattyBot, Khiladi 2010, Noah Slater, Farvartish, Knudmoeller, Electricmuffin11, Mbarrenecheajr, Corrector623, Sandy.toast, F331491, Luebbert42, Holland.jg, Anujsahni, Tsvljuchsh, Makecat-bot, Fitzchak, Toopathfind, Msalvadores, Cloud-dev, Sasindar, Zhanghaohit, CJGarner, Crosstantine, Stevenguttman, Razibot, DallasClarke, Altered

Walter, Rediosoft, Tsm32, François Robere, Harpreet dandeon, LeeAMitchell, Mbroberg, Virendervermamca, Anilkumar1129, Mhgrove, FranzKraun, Jasonhpang, Nanolat, Nosql.analyst, Rzicari, Ginsuloft, Sugamsha, K0zka, Tshuva, Dodi 8238, Dabron, Mongochang, Natan.puzis, Webtrill, CafeNoName, Yasinaktimur, Monkbot, Itamar.haber, User db, Columbus240, Textractor, Maykurutu, Kamaci, Mongodbheavy, Dexterchief, RedOctober13, Nathan194, Jjrenzel, Azanebrain, Annmkelly1, Sunnyeyre, Danny996, Thomas4019, Dr. Testificate, M.D., Johnbweeks, Magic-carpet-pilot, Heymattallen, Teowey, IdlePlayground, Datadatadatadata, Kevin at aerospike, Anywhichway, Nawazdhandala, The King Breaker, Mohammad.rafigh, Mgentz, Desertrat1969 and Anonymous: 551

- **Key-value database** *Source:* https://en.wikipedia.org/wiki/Key-value_database?oldid=693289963 *Contributors:* Maury Markowitz, Bearcat, Beland, ArnoldReinhold, MZMcBride, Kgfleischmann, FrescoBot, Danos p, GGink, Altered Walter, Itamar.haber, Opencooper and Anonymous: 6
- **Document-oriented database** *Source:* https://en.wikipedia.org/wiki/Document-oriented_database?oldid=693497164 *Contributors:* Maury Markowitz, Edward, Ehn, RedWolf, Gwicke~enwiki, Cobaltbluetony, Chris Wood, Beland, Plasma east, Iznogoud~enwiki, Thorwald, Rfl, Imroy, Enric Naval, Shenme, Mdd, Arthena, Stuartyeates, Woohookitty, Mindmatrix, Dm~enwiki, Kingsleyj, MassGalactusUniversum, JIP, Dmccreary, Crazycomputers, Intgr, Sderose, KirtWalker, Wavelength, MySchizoBuddy, Cedar101, Thumperward, Jerome Charles Potts, Frap, Cybercobra, Superjordo, Eedeebee, Vivek.raman, FatalError, Spdegabrielle, Cydebot, Philu, QuiteUnusual, Dasfrosty, Plamoa, Toutoune25, Nyttend, R'n'B, Nwbeeson, Mqchen, Vishal0soni, TXiKiBoT, Kulkarninikhil, Rachkovsky, Benrcass, Antony.stubbs, Niceguyedc, Pointillist, Boleyn, Lodrian~enwiki, Addbot, Mortense, Fgnievinski, SDSWIKI, Goldzahn, Bunnyhop11, Pcap, Carleas, AnomieBOT, FreeRangeFrog, Neitherk, ChristianGruen, GrouchoBot, Rtweed1955, FrescoBot, Mark Renier, BrideOfKripkenstein, Bablind, RedBot, Refactored, Argv0, Mreftel, Hchrm, EmausBot, Akagel, EricBloch, Bxj, Staszek Lem, ClueBot NG, Rezabot, Danim, JasonNichols, Helpful Pixie Bot, Danmcg.au, Mark Arsten, Compfreak7, Luebbert42, Dodilp, CJGarner, Crosstantine, Altered Walter, Rediosoft, Hzguo, Tsm32, François Robere, Mbroberg, Cbuccella, Tshuva, ScotXW, Dodi 8238, There is a T101 in your kitchen, Webtrill, Adventurer61, Heymattallen, Datadatadatadata, ChrisChinchillaWard, Haptic-feedback, Jorl17, Oleron17, Tannerstirrat and Anonymous: 106
- **NewSQL** *Source:* https://en.wikipedia.org/wiki/NewSQL?oldid=693078739 *Contributors:* Maury Markowitz, Beland, Julian Mehnle~enwiki, Stuartyeates, MacTed, Quuxplusone, Intgr, Datamgmt, Amux, Frap, Apavlo, Kgfleischmann, Ibains, Duncan.Hull, MPH007, Phoenix720, Dsimic, MrOllie, Yobot, AnomieBOT, Noq, W Nowicki, Diego diaz espinoza, Cnwilliams, Bulwersator, UMDDatabase, BG19bot, Akim.demaille, Dexbot, Plothridge, Sanketsarang, Brianna.galloway, Mwaci99, Andygrove73, Monkbot, Sergejjurecko, Qid4475, Hvaara, Oleron17 and Anonymous: 19
- **ACID** *Source:* https://en.wikipedia.org/wiki/ACID?oldid=693458102 *Contributors:* AxelBoldt, Verloren, Matusz, PierreAbbat, Fubar Obfusco, Maury Markowitz, Zippy, Mrwojo, Edward, Michael Hardy, Kku, Markonen, Karada, Haakon, Poor Yorick, IMSoP, Clausen, GregRobson, Mcenedella, Zoicon5, Robbot, Kristof vt, RedWolf, Bernhard Bauer, Rfc1394, DHN, Jleedev, Zigger, Leonard G., Kainaw, Neilc, Beland, Saucepan, Vina, Daniel11, Urhixidur, Rfl, Discospinster, Rich Farmbrough, Thomas Willerich, Ponder, Elwikipedista~enwiki, Rlaager, Smalljim, Shenme, LuoShengli, Raja99, BlueNovember, Espoo, Anthony Appleyard, Suruena, Endersdouble, Ceyockey, Forderud, UFu, Mindmatrix, Swamp Ig, Barrylb, Kam Solusar, WadeSimMiser, Turnstep, Yurik, Rjwilmsi, Salix alba, Raztus, FayssalF, FlaBot, StephanCom, Ysangkok, Kmorozov, Fragglet, Quuxplusone, Intgr, Joonasl, Chobot, YurikBot, Personman, Pip2andahalf, Petiatil, Mskfisher, Barefootguru, Rsrikanth05, Ytcracker, Jpbowen, CPColin, Larsinio, RUL3R, Jessemerriman, MacMog, Paul Magnussen, Rms125a@hotmail.com, Saeed Jahed, Gorgan almighty, Katieh5584, Benandorsqueaks, Victor falk, KnightRider~enwiki, BonsaiViking, SmackBot, Amolshah, Renku, Bmearns, Mcherm, Gilliam, Thumperward, Prachee.j, DHN-bot~enwiki, Decibel, Corbin-Simpson, Grover cleveland, SeanAhern, Luís Felipe Braga, Acdx, Dave.excira, Accurizer, Bezenek, IronGargoyle, Lee Carre, TwistOfCain, Paul Foxworthy, MrRedwood, Jontomkittredge, FatalError, SqlPac, Ivan Pozdeev, Safalra, Christian75, DumbBOT, Surturz, Viridae, Thijs!bot, Epbr123, Marek69, Vertium, Uiteoi, Jaydlewis, Siggimund, Hmrox, AntiVandalBot, Gioto, Seaphoto, Lfstevens, Stangaa, DAllardyce, MetsBot, R'n'B, Tgeairn, Huzzlet the bot, J.delanoy, Trusilver, Inimino, It Is Me Here, Gurchzilla, NewEnglandYankee, DorganBot, Tagus, Inter16, WhiteOak2006, Reelrt, Jeff G., Af648, Drake Redcrest, Sean D Martin, Martin451, BwDraco, Noformation, Duncan.Hull, Wykypydya, Zhenqinli, Charliearcuri, Triesault, Synthebot, !dea4u, Sesshomaru, Heiser, YonaBot, Kaell, Flyer22 Reborn, JCLately, Svick, AlanUS, Siskus, Denisarona, Loren.wilton, ClueBot, Jagun, Boing! said Zebedee, Passargea, Gakusha, Excirial, SoxBot III, Trefork, Trvth, Maimai009, Addbot, Some jerk on the Internet, Ngpd, Shmuelsamuele, CanadianLinuxUser, Download, CarsracBot, Tide rolls, Yobot, Synchronism, AnomieBOT, ThinkerFeeler, Jim1138, Kd24911, Nmfon, Flewis, Materialscientist, RobertEves92, Citation bot, E2eamon, Dudegroove, Vhabacoreilc, Obersachsebot, Pontificalibus, Miym, Cole2, Tabledhote, Tct13, FrescoBot, Bluiee, Mark Renier, MGA73bot, HJ Mitchell, Sae1962, DivineAlpha, Citation bot 1, Bunyk, Redrose64, I dream of horses, Hellknowz, SpaceFlight89, Σ, Throwaway85, Vrenator, Premsurya, Noommos, Gf uip, DASHBot, RA0808, Tommy2010, TuHan-Bot, Wikipelli, John Cline, Fæ, Sckolar, Makecat, Prasannawikis, Tolly4bolly, Rob7139, Puffin, Wildrain21, ClueBot NG, Andrei S, 123Hedgehog456, Widr, Ghostdood, Novusuna, Strike Eagle, Titodutta, Calabe1992, Lowercase sigmabot, Jordonbyers, Amiramix, MPSUK, Mark Arsten, Silvrous, Yourbane, Agnt9, Klilidiplomus, Fylbecatulous, Winston Chuen-Shih Yang, Yazan kokash23, Thecodysite1, Mdann52, Polupolu890, Arr4, ZappaOMati, EuroCarGT, Kirilldoom16, Harsh 2580, Dexbot, Golfguy399, Mynameissskr, NewAspen1, Epicgenius, Woo333, 7Rius, DevonDBA, 7Sidz, Midget zombie, Kethrus, Winghouchan, Mayank0001, Bryanleungnokhin12345 and Anonymous: 489
- **Consistency (database systems)** *Source:* https://en.wikipedia.org/wiki/Consistency_(database_systems)?oldid=674938190 *Contributors:* Patrick, Greenrd, Rfl, CanisRufus, Suruena, Ewlyahoocom, Intgr, YurikBot, Ste1n, Sasuke Sarutobi, Gaius Cornelius, SmackBot, Silly rabbit, Capmo, Gregbard, StudierMalMarburg, Jeepday, JCLately, M4gnum0n, Addbot, Willking1979, Fraggle81, Obersachsebot, Mark Renier, Sae1962, SpaceFlight89, Rob7139, Encyclopedant, Monkbot, Ganeshdtor, Nerdgonewild and Anonymous: 15
- **Durability (database systems)** *Source:* https://en.wikipedia.org/wiki/Durability_(database_systems)?oldid=617369164 *Contributors:* Edward, CesarB, Clausen, LordHz, Tobias Bergemann, Saucepan, Rfl, Ewlyahoocom, SmackBot, Bluebot, Wizardman, SqlPac, Astazi, JCLately, D3fault, Addbot, Erik9bot, Mark Renier, Yourbane, JYBot and Anonymous: 8
- **Serializability** *Source:* https://en.wikipedia.org/wiki/Serializability?oldid=687103552 *Contributors:* Ahoerstemeier, Greenrd, DavidCary, ArnoldReinhold, Arthena, Ruud Koot, MassGalactusUniversum, BD2412, Rjwilmsi, Darthsco, Wavelength, That Guy, From That Show!, SmackBot, Amux, Chris the speller, Mihai Capotă, Flyguy649, Cybercobra, Paul Foxworthy, Comps, MeekMark, Paddles, Kubanczyk, Supparluca, VoABot II, Rxtreme, R'n'B, Deor, VolkovBot, Klower, JCLately, Svick, M4gnum0n, Addbot, Fyrael, Alex.mccarthy, Zorrobot, Luckas-bot, Yobot, AnomieBOT, Materialscientist, LilHelpa, Gilo1969, Miym, Omnipaedista, FrescoBot, Mark Renier, Craig Pemberton, Farhikht, Tbhotch, DRAGON BOOSTER, John of Reading, Dewritech, Fæ, Mentibot, ClueBot NG, Jack Greenmaven, Richard3120, MerlIwBot, Kgrittn, Cyberpower678, Cyberbot II and Anonymous: 50
- **Isolation (database systems)** *Source:* https://en.wikipedia.org/wiki/Isolation_(database_systems)?oldid=686923177 *Contributors:* AxelBoldt, Ramesh, IMSoP, Hadal, Tobias Bergemann, Beland, J18ter, Asqueella, Rfl, KeyStroke, EmmetCaulfield, Velella, Mindmatrix,

Swamp Ig, Mattmorgan, Mandarax, Bunchofgrapes, Ketiltrout, Ej, Michal.burda, Maxal, Chris Purcell, Ewlyahoocom, Alvin-cs, Ivansoto, RussBot, Phlip, Stefan Udrea, Laurent Van Winckel, Closedmouth, SmackBot, Khfan93, Serhio, Gracenotes, Cybercobra, Djmitche, Joe-Bot, Igoldste, Insanephantom, Nczempin, Ervinn, Slazenger, TheJC, Thijs!bot, Maverick13, JMatthews, Rsocol, Mentin, Cameltrader, Magioladitis, Hheimbuerger, MartinBot, Scku, NunoFerreira, SoCalSuperEagle, Wikidemon, Inovakov, Enigmaman, JCLately, Paul Clapham, Ian Clelland, ClueBot, Erichero, The Thing That Should Not Be, Mild Bill Hiccup, Niceguyedc, LonelyBeacon, Alexbot, Addbot, Materialscientist, Xqbot, Prunesqualer, SPKirsch, Sae1962, Searcherfinder, Irbisgreif, Sahedin, BYVoid, DARTH SIDIOUS 2, QLineOrientalist, Hrishikeshbarua, Anonymouslee, Antonio.al.al, Olof nord, Tommy2010, Olnrao, ClueBot NG, Kgrittn, BG19bot, Wiki13, Brian.low22, Snow Blizzard, Sbose7890, Tommy0605, MaryEFreeman, Kevsteppe, KasparBot and Anonymous: 176

- **Database transaction** *Source:* https://en.wikipedia.org/wiki/Database_transaction?oldid=683102718 *Contributors:* Damian Yerrick, Nixdorf, SebastianHelm, WeißNix, Ajk, Clausen, GregRobson, Owen, Craig Stuntz, RedWolf, Babbage, KellyCoinGuy, Lysy, Mintleaf~enwiki, T0m, Jason Quinn, Timo~enwiki, Neilc, Troels Arvin, Burschik, KeyStroke, Mike Schwartz, DCEdwards1966, Obradovic Goran, Haham hanuka, Jeltz, Derbeth, Forderud, Mindmatrix, TigerShark, AnmaFinotera, Turnstep, OMouse, FlaBot, Dauerad, Intgr, Karel Anthonissen, Chobot, Bgwhite, YurikBot, Matiash, Hede2000, SAE1962, Larsinio, Luc4~enwiki, Mikeblas, Adi92~enwiki, SmackBot, Georgeryp, Gilliam, Lubos, PureRED, Khukri, MegaHasher, RichMorin, 16@r, Slakr, Paul Foxworthy, Comps, SqlPac, WeggeBot, Stevag, Thijs!bot, CharlotteWebb, JAnDbot, Geniac, Cic, Leeborkman, Hbent, Idioma-bot, TXiKiBoT, Zhenqinli, Billinghurst, Gerd-HH, Daniel0524, Prakash Nadkarni, BotMultichill, Roesser, JCLately, Fratrep, OKBot, ClueBot, Binksternet, DnetSvg, M4gnum0n, Triwger, HumphreyW, Addbot, Ghettoblaster, Highguard, Sandrarossi, Lightbot, Jarble, Yobot, Pcap, AnomieBOT, Rubinbot, JackieBot, Materialscientist, Zerksis, Pepper, Mark Renier, Al3ksk, RedBot, Lingliu07, Sobia akhtar, Gf uip, K6ka, Rocketrod1960, ClueBot NG, MerlIwBot, Helpful Pixie Bot, Mostafiz93, Kirananils, AmandeepJ, ChrisGualtieri, Davew123, Lemnaminor, Appypani, Juhuyuta, Thisismyusername96 and Anonymous: 94
- **Transaction processing** *Source:* https://en.wikipedia.org/wiki/Transaction_processing?oldid=674407211 *Contributors:* Maury Markowitz, Zippy, Pratyeka, Clausen, GregRobson, Gutza, SEWilco, Craig Stuntz, Tobias Bergemann, Khalid hassani, Uzume, Beland, Abdull, Gordonjcp, Atlant, Wtmitchell, Stephan Leeds, Suruena, Mindmatrix, Ruud Koot, MONGO, Mandarax, BD2412, Chobot, Cliffb, Mikeblas, Zzuuzz, Rbpasker, Tschristoppe~enwiki, Kgf0, SmackBot, Chairman S., Agateller, BBCWatcher, Avb, Radagast83, Akulkis, Luís Felipe Braga, Joshua Scott, 16@r, JHunterJ, Beve, Baiji, Stymiee, Adolphus79, Bruvajc, Thijs!bot, Kubanczyk, JAnDbot, MER-C, Donsez, DGG, Gwern, MartinBot, Rettetast, DeKXer, Jmcw37, STBotD, Jeff G., Lear's Fool, Andy Dingley, JCLately, CutOffTies, Oxymoron83, Jan1nad, M4gnum0n, Ghaskins, Oo7nets, Addbot, MrOllie, Download, LaaknorBot, Lightbot, Wireless friend, Luckas-bot, Yobot, Pcap, Peter Flass, AnomieBOT, Jim1138, Materialscientist, Ellynwinters, Unimath, Xqbot, Mika au, Amaury, Alkamins, Mark Renier, Charleyrich, Danielle009, Awolski, Gf uip, Cbwash, Dr Jonx, ClueBot NG, CaroleHenson, Danim, Jorgenev, Helpful Pixie Bot, BG19bot, Bnicolae, JoshuaChen, Wiki-jonne, Hampton11235, ClaeszXIV and Anonymous: 90
- **Journaling file system** *Source:* https://en.wikipedia.org/wiki/Journaling_file_system?oldid=692526177 *Contributors:* Marj Tiefert, Brion VIBBER, Wesley, Uriyan, The Anome, Tarquin, Stephen Gilbert, Enchanter, Rootbeer, Ghakko, Ark~enwiki, Hephaestos, Graue, Karada, (, Emperorbma, Magnus.de, Pistnor, Furrykef, Taxman, Khym Chanur, Phil Boswell, Robbot, Scott McNay, Naddy, Tim Ivorson, Cek, David Gerard, DavidCary, AviDrissman, Mintleaf~enwiki, AlistairMcMillan, Wmahan, Karlward, Beland, Kareeser, Damieng, Mormegil, ChrisRuvolo, KeyStroke, Luxdormiens, Indil, Edward Z. Yang, Androo, MARQUIS111, Poli, Guy Harris, Apoc2400, Seans Potato Business, Bart133, MIT Trekkie, Ruud Koot, Anthony Borla, Graham87, Rjwilmsi, Raffaele Megabyte, FlaBot, Maxal, Chobot, The Rambling Man, YurikBot, Wikipedia., NickBush24, Moppet65535, Nailbiter, Daleh, Eskimbot, Chris the speller, Charles Moss, Anabus, SheeEttin, Mwtoews, Ryulong, Bitwise, RekishiEJ, Rogério Brito, Unixguy, Seven of Nine, Davidhorman, Escarbot, AntiVandalBot, Widefox, Shlomi Hillel, Gavia immer, VoABot II, Public Menace, Cpiral, WJBscribe, DorganBot, Jcea, VolkovBot, AlnoktaBOT, Dani Groner, Steve0702, Leopoldt, MadmanBot, Rdhettinger, PipepBot, BJ712, Subversive.sound, Dsimic, Addbot, Ghettoblaster, AkhtaBot, Luckas-bot, Ptbotgourou, AnomieBOT, FrescoBot, Mfwitten, Citation bot 1, Hajecate, Merlion444, Japs 88, 15turnsm, Aharris16, ClueBot NG, BG19bot, Tsjerven, Forsakensam and Anonymous: 96
- **Atomicity (database systems)** *Source:* https://en.wikipedia.org/wiki/Atomicity_(database_systems)?oldid=693440689 *Contributors:* Michael Hardy, DopefishJustin, CesarB, Rohan Jayasekera, Biggins, Gdimitr, Hadal, Jleedev, Enochlau, Ancheta Wis, ArneBab, Rfl, Smyth, Hooperbloob, EmmetCaulfield, Danhash, RJFJR, Apokrif, PeterJohnson, Marudubshinki, Nihiltres, Chris Purcell, Ewlyahoocom, Julescubtree, Snailwalker, Chobot, Korg, RussBot, Sasuke Sarutobi, Bota47, Vicarious, SmackBot, Jmendez, Betacommand, DanPope, Tim-Bentley, LinguistAtLarge, Hkmaly, Kvng, Dreftymac, JForget, Neelix, Bodragon, JPG-GR, TXiKiBoT, Synthebot, Freshbaked, JCLately, Qwertykris, Addbot, Jncraton, Fraggle81, Materialscientist, Erik9bot, Mark Renier, Sae1962, Widr, Anutural, Juan Carlos Farah, TvojaStara, Cooldudevipin and Anonymous: 37
- **Lock (database)** *Source:* https://en.wikipedia.org/wiki/Lock_(database)?oldid=664133461 *Contributors:* Greenrd, Ta bu shi da yu, Duplode, Caidence, Maxal, SmackBot, InverseHypercube, Mauro Bieg, VinnieCool, Bruvajc, Thijs!bot, Wikid77, Belenus, Cic, Jack007, Jojalozzo, Kidoshisama, Addbot, LilHelpa, Vishnu2011, Gulsig4, Potionism, ClueBot NG, Danim, Sharpshooter4008, EdwardH, ChrisGualtieri, FCutic and Anonymous: 20
- **Record locking** *Source:* https://en.wikipedia.org/wiki/Record_locking?oldid=647556070 *Contributors:* Michael Hardy, Finn-Zoltan, D6, Atlant, Pol098, SmackBot, Quaddriver, Umaguna, JCLately, Jojalozzo, Niceguyedc, Erik9bot, This lousy T-shirt, Waynelwarren, BattyBot and Anonymous: 15
- **Two-phase locking** *Source:* https://en.wikipedia.org/wiki/Two-phase_locking?oldid=684806331 *Contributors:* Michael Hardy, Poor Yorick, Clausen, Dtaylor1984, Neilc, Andreas Kaufmann, Rich Farmbrough, Nchaimov, Woohookitty, Ruud Koot, Wavelength, Aaron Schulz, SmackBot, OrangeDog, Jeskeca, Cybercobra, Comps, SeanMon, Epbr123, Touko vk, Seaphoto, Beta16, Syst3m, Lerdthenerd, Paul20070, Gerakibot, Yintan, Svick, ImageRemovalBot, Stuart.clayton.22, Addbot, Cxz111, Thomas Bjørkan, Yobot, AnomieBOT, Materialscientist, AbigailAbernathy, Craig Pemberton, John of Reading, ClueBot NG, Chrisjameskirkham, Cntras, Helpful Pixie Bot, Dexbot, Jodosma, JohnTB and Anonymous: 41
- **Multiversion concurrency control** *Source:* https://en.wikipedia.org/wiki/Multiversion_concurrency_control?oldid=689311592 *Contributors:* Poor Yorick, Palfrey, Ggaughan, Dcoetzee, Jamesday, RickBeton, Craig Stuntz, Rsfinn, DavidCary, Neilc, Chowbok, Rawlife, Troels Arvin, Rich Farmbrough, Smyth, Martpol, R. S. Shaw, Franl, Terrycojones, RJFJR, Drbreznjev, GregorB, Ssteedman, Turnstep, Rjwilmsi, Ysangkok, Chris Purcell, Intgr, YurikBot, Piet Delport, Gaius Cornelius, AmunRa, Blowdart, Naasking, JLaTondre, That Guy, From That Show!, SmackBot, Basil.bourque, Chris the speller, ThurnerRupert, Cbbrowne, Hu12, Tawkerbot2, ChrisCork, Comps, Raysonho, Gritzko, Elendal, Hga, Jordan Brown, ProfessorBaltasar, Cydebot, Cwhii, Marcuscalabresus, Nowhere man, Andrewjamesmorgan, Visik, Dougher, Tedickey, Seashorewiki, Ahodgkinson, Kiore, Yannick56, Kedawa, DanielWeinreb, Fecund, Bill.zopf, Dllahr, Arleach, Highlandsun, Dfetter, Doug4j, Danilo.Piazzalunga, MrChupon, Whimsley, Jerryobject, JCLately, Kobotbel, Breinbaas, Siskus, Tuntable, Jonathanstray,

Nthiery, CYCC, M4gnum0n, Kalotus, KyleJ1, MelonBot, XLinkBot, MystBot, Addbot, Tsunanet, Lightbot, Yobot, Pcap, Pvjohnson, AnomieBOT, Yoonforh, Drachmae, ThomasTomMueller, LilHelpa, FrescoBot, LucienBOT, DrilBot, RedBot, Seancribbs, Full-date unlinking bot, Unordained, Obankston, John of Reading, Dewritech, EricBloch, Tuhl, H3llBot, Stradafi, Dexp, CasualVisitor, BG19bot, Wasbeer, Julien2512, Snnn~enwiki, Compfreak7, Kevin 71984, Waynelwarren, Kwetal1, Giloki, JYBot, Bdempsey64, JingguoYao, Mbautin, Will Faught, Kevinroy09, Johnkarva, Plothridge, Textractor, Kanelai, Kclee968, Oleron17 and Anonymous: 108

- **Snapshot isolation** *Source:* https://en.wikipedia.org/wiki/Snapshot_isolation?oldid=690117018 *Contributors:* Craig Stuntz, Neilc, Isidore, Chowbok, Woohookitty, Ruud Koot, Ej, Ysangkok, Chris Purcell, Blowdart, Elkman, Johndburger, SmackBot, Comps, Cydebot, Andrewjamesmorgan, David Eppstein, VanishedUserABC, JCLately, Tuntable, Idleloop~enwiki, Yobot, Pcap, AnomieBOT, Citation bot, Searcherfinder, Citation bot 1, AnnHarrison, Helpful Pixie Bot, Kgrittn and Anonymous: 13
- **Two-phase commit protocol** *Source:* https://en.wikipedia.org/wiki/Two-phase_commit_protocol?oldid=690180092 *Contributors:* Pnm, Ciphergoth, Lkesteloot, Gtrmp, Neilc, Gdr, Rworsnop, Rdsmith4, Rich Farmbrough, CanisRufus, R. S. Shaw, Liao, Bestchai, Mbloore, Suruena, ReubenGarrett, Ruud Koot, Choas~enwiki, MassGalactusUniversum, YurikBot, Bayle Shanks, Daleh, Segv11, Emilong, SmackBot, Dubwai, Jdeisenh, Coredesat, Zero sharp, Comps, Pmerson, Touko vk, LenzGr, Somebody2014, MartinBot, Stephanwehner, Richard KAL, Ja 62, VolkovBot, Cyberjoac, JCLately, Svick, Yagibear, WikHead, Addbot, DOI bot, Twimoki, Yobot, AnomieBOT, Materialscientist, Uglybugger, Wktsugue, FrescoBot, Killian441, PleaseStand, EmausBot, Flegmon, ClueBot NG, Braincricket, Electriccatfish2, BG19bot, JingguoYao, SLipRaTi, Deepu2k, Hotcheese92 and Anonymous: 76
- **Three-phase commit protocol** *Source:* https://en.wikipedia.org/wiki/Three-phase_commit_protocol?oldid=676045717 *Contributors:* Chris Q, Pnm, Trevor Johns, Levin, Rworsnop, N.o.bouvin, Guy Harris, Ceyockey, Ampledata, Choas~enwiki, Rjwilmsi, GreyCat, Bayle Shanks, Aaron Schulz, Segv11, Emilong, Jsnx, Zero sharp, Megatronium, Junche, MarkKampe, TXiKiBoT, JCLately, Addbot, DOI bot, Citation bot, FrescoBot, IdishK, Joerg Bader, BG19bot, BattyBot, Remcgrath, Monkbot and Anonymous: 26
- **Scalability** *Source:* https://en.wikipedia.org/wiki/Scalability?oldid=690445208 *Contributors:* Kpjas, The Anome, Awaterl, Matusz, Michael Hardy, Kku, TakuyaMurata, Bearcat, Sander123, Jondel, Dbroadwell, SpellBott, Lysy, Javidjamae, Leonard G., Stevietheman, Gdr, Beland, Urhixidur, Hugh Mason, Ferrans, FT2, Mazi, Dtremenak, Liao, Calton, Pion, Suruena, Kusma, Mattbrundage, Tyz, Undefined~enwiki, BD2412, Rjwilmsi, Quiddity, Williamborg, Fred Bradstadt, Aapo Laitinen, FlaBot, Intgr, Dalef, Agil~enwiki, YurikBot, Whoisjohngalt, NTBot~enwiki, Michael Slone, Bovineone, Moe Epsilon, Leotohill, .marc., Xpclient, LeonardoRob0t, Stumps, SmackBot, Irnavash, KelleyCook, Ohnoitsjamie, Thumperward, Jammus, Javalenok, Ascentury, Frap, JonHarder, Cyhatch, BWDuncan, Andrei Stroe, Harryboyles, Writtenonsand, 16@r, Swartik, Hu12, UncleDouggie, Tawkerbot2, FatalError, CBM, Thijs!bot, Uiteoi, Marokwitz, Kdakin, JAnDbot, NapoliRoma, Shar1R, SunSw0rd, Raffen, Joshua Davis, FienX, RockMFR, Auroramatt, 1000Faces, NewEnglandYankee, Doria, Jottinger, Izno, VolkovBot, TXiKiBoT, CHaoTiCa, Falcon8765, Suction Man, Bpringlemeir, Paladin1979, DigitalDave42, JCLately, Luciole2013, Gp5588, Dangelow, Nvrijn, Elnon, Tearaway, Mild Bill Hiccup, Saravu2k, M4gnum0n, Friendlydata, Shiro jdn, MPH007, XLinkBot, Philippe Giabbanelli, Avoided, Klungel, MystBot, Dsimic, Addbot, Jncraton, Tonkie67, Fluffernutter, MrOllie, Latiligence, Kiril Simeonovski, Teles, Luckas-bot, Yobot, Davew haverford, Terrifictriffid, AnomieBOT, Materialscientist, Obersachsebot, Xqbot, Miym, GrouchoBot, Sae1962, MastiBot, Jandalhandler, Akolyth, Jesse V., Gf uip, EmausBot, John of Reading, Anirudh Emani, Josve05a, Cosmoskramer, AManWithNoPlan, Music Sorter, Tsipi, ChuispastonBot, ClueBot NG, Widr, Daniel Minor, Meniv, Helpful Pixie Bot, MarkusWinand, Electriccatfish2, BG19bot, ElphiBot, Wikicadger, Anbu121, Srenniw, BattyBot, CGBoas, K0zka, Mwaci11, Paul2520, Shahbazali101, Igorghisi, Vieque, Verbal.noun, Mantraman701, KasparBot, The Quixotic Potato, KealanJH and Anonymous: 121
- **Shard (database architecture)** *Source:* https://en.wikipedia.org/wiki/Shard_(database_architecture)?oldid=692288729 *Contributors:* Rfl, Kenfar, Fche, Gareth McCaughan, Winterstein, Bgwhite, Hairy Dude, Grafen, Neil Hooey, Wainstead, Deepdraft, SmackBot, Russorat, Cybercobra, Jdlambert, Steipe, Bezenek, Sanspeur, Underpants, Jadahl, Dougher, Magioladitis, Eleschinski2000, StefanPapp, Otisg, McSly, NewEnglandYankee, Cswpride, Phasma Felis, Andy Dingley, Angusmca, SmallRepair, ClueBot, Fipar, Delicious carbuncle, Dthomsen8, MystBot, Yobot, AnomieBOT, Noq, Isheden, FontOfSomeKnowledge, Rfportilla, Mopashinov, Jacosi, Tilkax, FrescoBot, X7q, Haeinous, Sae1962, Gautamsomani, Citation bot 1, Winterst, Jandalhandler, Crysb, Visvadinu, Cfupdate, EmausBot, ClueBot NG, Liran.zelkha, BG19bot, Eric-vrcl, MusikAnimal, MeganShield, Tmalone22, BattyBot, Jaybear, Drewandersonnz, Cc4fire, David9911, Dudewhereismybike, Alexandre.marini, Sodomojo, Prabhusingh25, Viam Ferream, Josephidziorek, Codingalz and Anonymous: 76
- **Optimistic concurrency control** *Source:* https://en.wikipedia.org/wiki/Optimistic_concurrency_control?oldid=655011024 *Contributors:* Karada, Poor Yorick, Nikai, Timwi, Zoicon5, DavidCary, Neilc, Beland, D6, Rich Farmbrough, Smyth, Dmeranda, Mnot, R. S. Shaw, Bhaskar, Suruena, HenryLi, Simetrical, GregorB, Intgr, Ahunt, SAE1962, SmackBot, Slamb, SmartGuy Old, Drewnoakes, NYKevin, Allan McInnes, Russorat, Cybercobra, Iridescent, Zero sharp, Nczempin, Mydoghasworms, Lfstevens, Magioladitis, JeromeJerome, Algotr, Homer Landskirty, Randomalious, SieBot, JCLately, DraX3D, OKBot, Svick, Jfromcanada, ClueBot, Sim IJskes, Methossant, M4gnum0n, Addbot, Ace of Spades, BenzolBot, Citation bot 1, RedBot, RjwilmsiBot, Helpful Pixie Bot, Noazark, Therealmatbrown, Monkbot, Bhutani.ashish14 and Anonymous: 36
- **Partition (database)** *Source:* https://en.wikipedia.org/wiki/Partition_(database)?oldid=685683849 *Contributors:* Ehn, Peak, Foonly, S.K., Stevelihn, Alai, Geoffmcgrath, Mindmatrix, Drrngrvy, Roboto de Ajvol, Yahya Abdal-Aziz, Brian1975, Mikeblas, Georgewilliamherbert, Doubleplusjeff, Ccubedd, Salobaas, Andrew.rose, JAnDbot, Mdfst13, Jamelan, Andy Dingley, Jonstephens, Angusmca, Ceva, SmallRepair, Pinkadelica, Saravu2k, Chrisarnesen, Addbot, Vishnava, Highflyerjl, MrOllie, SamatBot, Yobot, AnomieBOT, Beaddy1238, Materialscientist, Semmerich, Isheden, LucienBOT, Mark Renier, Wordstext, Troy.frericks, Habitmelon, Fholahan, Lurkfest, Jan.hasller, Vieque and Anonymous: 32
- **Distributed transaction** *Source:* https://en.wikipedia.org/wiki/Distributed_transaction?oldid=650232287 *Contributors:* Gtrmp, Saucepan, MartinBiely, Exceeder~enwiki, Ruud Koot, Jameshfisher, X42bn6, ShinyKnows, Gaius Cornelius, SmackBot, Brick Thrower, Jeskeca, Ligulembot, Comps, Pmerson, Darklilac, GL1zdA, Wikiisawesome, JCLately, Alexbot, MelonBot, MystBot, Addbot, Pcap, AnomieBOT, John of Reading, K6ka, Helpful Pixie Bot, Mediran and Anonymous: 15
- **Redis** *Source:* https://en.wikipedia.org/wiki/Redis?oldid=693848779 *Contributors:* Thebramp, Ehn, Toreau, Beland, ShakataGaNai, Jaybuffington, Stesch, Rfl, Rich Farmbrough, Plest, Philipp Weis, CONFIQ, Barrylb, Bhound89, Justin Ormont, Sdornan, Raztus, Husky, Intgr, Bgwhite, Iamfscked, Rsrikanth05, SamJohnston, Joel7687, Unforgiven24, Poohneat, Mike Dillon, Modify, Nic Doye, SmackBot, Larry Doolittle, Boul22435, Wolph~enwiki, Vid, Frap, ThurnerRupert, Heelmijnlevenlang, Vanished user ih3rjk324jdei2, Aaaidan, Sanspeur, Cydebot, Ivant, Jamitzky, Avi4now, Wdspann, HazeNZ, Jm3, Omarkonsul, Sorenriise, Gioto, Kavehmb, Adys, Scorwin, Vishal0soni, Satani, TXiKiBoT, BlackVegetable, Jonknox, Benclewett, Swillison, ImageRemovalBot, Arkanosis, M4gnum0n, Miami33139, Dthomsen8, Rreagan007, Addbot, Mortense, Jncraton, Gnukix, Tnm8, Balabiot, Jarble, Pcap, Kmerenkov, Dmarquard, BastianVenthur, Mavz0r, Xqbot, Soveran, Cole2, Jder~enwiki, FrescoBot, Ksato9700, Tóraí, Hoo man, Tim1357, Jfmantis, EmausBot, T. Canens, Spf2, Angrytoast, Djembayz, ZéroBot, Overred~enwiki, Corb555, Mark Martinec, Sitic, Adamretter, Kasirbot, Karmiq, Codingoutloud, BG19bot,

Toffanin, Anne.naimoli, Fceller, Axule, Samusman waz, Rashidusman82, User 9d3ffb53ccf204e0, Ju2ender, Dexbot, Rwky, Altered Walter, Luhnatic, Dme26, Aeon10, Nikunjsingh01, Napy65, AlexanderRedd, Wksunder, Ebr.wolff, Mindskt, Mmoreram, Itamar.haber, AbinashMishra1234, Ushnishtha, S4saurabh12, Nikkiobjectrocket, Andlpa63, Terrencepryan and Anonymous: 126

- **MongoDB** *Source:* https://en.wikipedia.org/wiki/MongoDB?oldid=693452881 *Contributors:* AxelBoldt, William Avery, Phoe6, Grendelkhan, Topbanana, Tomchiukc, Wjhonson, Sriehl, Jason Quinn, Masterhomer, Coldacid, Pgan002, Beland, Alobodig, Josephgrossberg, ShortBus, Thorwald, Perey, Rfl, Rich Farmbrough, Jakuza~enwiki, Nabla, Theinfo, Lauciusa, Stesmo, Kfogel, Koper, Mdd, PCJockey, Zachlipton, Tobych, Kocio, Nforbes, Falcorian, LOL, Adallas, Dm~enwiki, GregorB, Johnny99, VsevolodSipakov, HV, AlisonW, RobertG, Garyvdm, Kolbasz, FrankTobia, Wavelength, Hairy Dude, Youngtwig, SamJohnston, Thunderforge, Ben b, Arthur Rubin, Fram, ViperSnake151, Narkstraws, AndrewWTaylor, DrJolo, SmackBot, Mauls, KennethJ, Ohnoitsjamie, Skizzik, Jdorner, Ctrlfreak13, Frap, Meandtheshell, Ochbad, DMacks, KDIsom, Stennie, Heelmijnlevenlang, Chickencha, Hu12, MikeWazowski, Mikeyv, Alexey Feldgendler, Mineralè, Cydebot, Calorus, Widefox, Kohenkatz, Avleenvig, Magioladitis, JamesBWatson, AlbinoChocobo, David Eppstein, SBunce, Cander0000, Jackson Peebles, Lmxspice, CommonsDelinker, Qweruiop321, Grshiplett, BrianOfRugby, Serge925, TXiKiBoT, BookLubber, KickaXe, Wingedsubmariner, Crcsmnky, Tuxcantfly, Valio bg, Koryu Obihiro, Iapain wiki, Quiark, Kbrose, Adm.Wiggin, Yashwantchavan, Jojalozzo, Elibarzilay, Hello71, Svick, Solprovider, Shruti14, Gian-Pa, WDavis1911, Niceguyedc, ArlenCuss, Alexbot, Plaes, Jinlye, Supa Z, SteveMao, Megaltoid, Shijucv, The-verver, Crypticbird, Piratemurray, XLinkBot, Gerhardvalentin, MystBot, Deineka, Addbot, Moosehadley, CanadianLinuxUser, MrOllie, Jasper Deng, Peridon, Twimoki, TundraGreen, Ben Ben, Zyx, Luckas-bot, Yobot, JackPotte, Ptbotgourou, Amirobot, Wonderfl, AnomieBOT, Beaddy1238, Jim1138, Materialscientist, CoMePrAdZ, Cababunga, John Bessa, Gkorland, Flying sheep, Mackrauss, Miym, Jonas AGX, Omnipaedista, Yadavjpr, SassoBot, Ciges, Mike2782, Mu Mind, Haeinous, Jocelynp85, Marsiancba, Jandalhandler, Mdirolf, Chris Caven, LogAntiLog, OnceAlpha, Mreftel, Reaper Eternal, Difu Wu, Asafdapper, RjwilmsiBot, Mbferg, Pengwynn, WikitanvirBot, Najeeb1010, GoingBatty, Mixmax99, Al3xpopescu, Bernard.szlachta, Shuitu, Blr21000, Lateg, H3llBot, Vittyvk, Dstainer, Zephyrus Tavvier, Petrb, ClueBot NG, Mechanical digger, Thepaul0, Joefromrandb, Millermk, Ninja987, Renatovitolo, Mschneido, Masssly, Antiqueight, Nadavidson, Helpful Pixie Bot, BG19bot, Arkroll, Compfreak7, Dustinrodrigues, Cnevis, Ycallaf, Tobias.trelle, Dr. Coal, Jayadevp13, Solved009, Kizar, FeralOink, Chip123456, Ethefor, Justincheng12345-bot, Bpatrick001, Shaksharf, The Illusive Man, ChrisGualtieri, JYBot, Thenaberry1, Lan3y, Dexbot, Awesoham, Frosty, MartinMichlmayr, Samarthgahire, Jamesx12345, Mikewbaca, Alfre2v, CJGarner, DavidPKendal, CCC2012, Mahbubur-r-aaman, Amritakripa, Fc07, François Robere, Jan.hasller, The Herald, Francium1988, Ewolff42, ArmitageAmy, Tshuva, Xujizhe, Mongodb, Grouik92, Luxure, Kathir04on, ScotXW, Jameswahlin, Lvmetrics, Bhpdownloads, Mehdi2305, Dodi 8238, Dabron, Mongochang, Samohtm, Kaimast, Tazi Mehdi, Ayush3292, Mcocohen, Airuleguy, RationalBlasphemist, Swoopover, Vwsuran, Attish~enwiki, Alexeybut, Andlpa63, Andreea Panainte, Dwtkr, Iamvikas1982, Xsenechal, Anandchandak15, Mandge.rohit, Rnmandge and Anonymous: 342
- **PostgreSQL** *Source:* https://en.wikipedia.org/wiki/PostgreSQL?oldid=692506269 *Contributors:* 0, Carey Evans, Wesley, The Anome, Christopher Mahan, Aldie, Fubar Obfusco, Nate Silva, M~enwiki, Roadrunner, Maury Markowitz, TomCerul, Heron, Cwitty, Frecklefoot, Edward, Fuzzie, Gregben~enwiki, Nixdorf, Liftarn, Wwwwolf, (, Greenman, CesarB, Ahoerstemeier, KAMiKAZOW, Stevenj, Nanshu, Angela, Glenn, Sugarfish, Nikai, Ehn, Jay, Tejano, Doradus, Pedant17, Tpbradbury, Polyglot, Cleduc, Jnc, Wernher, Bevo, Joy, Fvw, Jamesday, Robbot, Chealer, Lowellian, Ianb, Stewartadcock, LX, Lasix, Weialawaga~enwiki, Oberiko, Nickdc, Levin, Fleminra, Ceejayoz, Prell, Lvr, Neilc, Chowbok, Utcursch, Pgan002, Alexf, Cbraga, ConradPino, Billposer, Gene s, Burgundavia, Karl-Henner, Cynical, Troels Arvin, GreenReaper, Deleteme42, RandalSchwartz, Imroy, Rich Farmbrough, Sesse, Oska, Ardonik, Lulu of the Lotus-Eaters, Slipstream, Gronky, Bender235, PaulMEdwards, Chibimagic, Kwamikagami, Kanzure, TommyG, Mereman, Giraffedata, Timsheridan, Minghong, Jonsafari, QuantumEleven, Stephen Bain, HasharBot~enwiki, Poli, Gary, Fchoong, Jeltz, Andrewpmk, Ringerc~enwiki, Pjacklam, Pauli133, SteinbDJ, Gmaxwell, Mindmatrix, MianZKhurrum, Dandv, Deeahbz, Jacobolus, Distalzou, Ruud Koot, Cosmicsoftceo, Bowman, GregorB, AnmaFinotera, Wisq, Turnstep, Marudubshinki, Graham87, EdDavies, Gilesmorant, KublaChaos, Silvestre Zabala, Zero0w, Mikecron, FlaBot, SchuminWeb, Gurch, Ghen, Intgr, Chobot, Reetep, Peterl, YurikBot, Manop, Gaius Cornelius, Bovineone, Varnav, Geertivp, Mipadi, Tkbwik, Randolf Richardson, Larsinio, E rulez, Snarius, BraneJ, Gsherry, Analoguedragon, Johndburger, Closedmouth, Johnsu01, MaNeMeBasat, BanzaiSi, JLaTondre, Benhoyt, Mlibby, A bit iffy, SmackBot, Nicolas Barbier, Direvus, Slamb, Faisal.akeel, Reedy, Georgeryp, Dkusnetzky, Anastrophe, Richmeister, Amux, Chris the speller, Bluebot, Wolph~enwiki, Advorak, DStoykov, Crashmatrix, Ben.c.roberts, Thumperward, Droll, Jerome Charles Potts, DHN-bot~enwiki, Decibel, Frap, Lantrix, Matchups, MattOates, Stevemidgley, Cybercobra, Emmanuel JARRI, Mwtoews, Where, Towsonu2003~enwiki, SashatoBot, Vincenzo.romano, Brian Gunderson, Avé, Misery~enwiki, Abolen, AdultSwim, MTSbot~enwiki, Peyre, DagErlingSmørgrav, Angryxpeh, Dark ixion, Hu12, FatalError, Raysonho, WeggeBot, Musashi1600, Andrew.george.hammond, Revolus, Cydebot, Krauss, Ttiotsw, Rchoate, Synergy, Ebrahim, Thijs!bot, MinorEdits, Andyjsmith, Dalahäst, Jcarle, Xzilla, AntiVandalBot, Bearheart, Marokwitz, LenzGr, Room813, Deflective, Cjkporter, Martinkunev, Dskoll, GregorySmith, Apostrophyx, NoDepositNoReturn, JamesBWatson, Bernd vdB~enwiki, Gabriel Kielland, EagleFan, Cander0000, Nevit, Gwern, Scottmacpherson, Ronbtni, Lmxspice, Zeus, R'n'B, Whale plane, Sven Klemm~enwiki, Keesiewonder, Usp, Eleven81, VolkovBot, Allencheung, Blindmatrix, Bramschoenmakers, Luke Lonergan, Kmacd, Gwinkless, HLHJ, RonaldDuncan, Dfetter, Eljope, Jibun, PieterDeBruijn, Agentq314, Anas2048, Kindofoctarine, Majeru, ChrisMiddleton, Bfcase, X-Fi6, Jojalozzo, Gorgot, Ctkeene, Ctxppc, Jonlandrum, Trevorbrooks, Martarius, ClueBot, Jhellerstein, Kl4m, Tintinobelisk, The Thing That Should Not Be, Unbuttered Parsnip, Kl4m-AWB, Nikolas Stephan, Niceguyedc, M4gnum0n, WikiNickEN, Freerangelibrarian, Pmronchi, Eeekster, TobiasPersson, Chrisarnesen, SF007, DumZiBoT, TimTay, XLinkBot, Jabberwoch, Andy318, Addbot, Mortense, Nate Wessel, Ginzel~enwiki, Fale, Luckas-bot, Yobot, Specious, AnomieBOT, Coolboy1234, Piano non troppo, Kukushk, ArthurBot, The Banner, Wikante, Gilo1969, Zenaan, FChurca, Mark Renier, W Nowicki, Alex.ryazantsev, Kwiki, Louperibot, William.temperley, Simple Bob, B3t, Tim baroon, Skyerise, Jandalhandler, Full-date unlinking bot, Bmomjian, Jons2006, Filiprem, Ravenmewtwo, RjwilmsiBot, Streapadair, EmausBot, Grldedwrdbutler, Marzalpac, Klenot, Dewritech, Peaceray, Solarra, Your Lord and Master, AvicBot, H3llBot, Demonkoryu, Gz33, கி. கார்த்திகேயன், Sbmeirow, Palosirkka, Eggyknap, Martinmarques, Tijfo098, ChuispastonBot, Rugh, Brew8028, DisneyG, ClueBot NG, Birkedit, Mangal ratna, Redneb33, Boria, Web20boom, Kasirbot, Patrias, Denys.Kravchenko, Kwetal1, Kweetal nl, Winston Chuen-Shih Yang, Dexbot, Codename Lisa, Laurenz albe, Palmbeachguy, Digoal, Praemonitus, RaphaelQS, Comp.arch, ScotXW, Lavagnino, Craigkerstiens, Erilong, Unician, Docesam, Simonriggs, X3mofile, Lucazeo, Helldalgo, E1328167, Johnlgrant, Vihorny, GCarterEDB, Transfat0g and Anonymous: 408
- **Apache Cassandra** *Source:* https://en.wikipedia.org/wiki/Apache_Cassandra?oldid=693344463 *Contributors:* Enchanter, Frecklefoot, Edward, Ronz, Stefan-S, Ehn, Hashar, Cleduc, Bearcat, Alan Liefting, Msiebuhr, Neilc, Pgan002, Beland, Euphoria, Rich Farmbrough, Bender235, Anthony Appleyard, PatrickFisher, YPavan, Runtime, Swaroopch, Mindmatrix, Timendum, Deansfa, Qwertyus, Jmhodges, Jweiss11, Vegaswikian, Intgr, Tas50, FrankTobia, SamJohnston, Formina Sage, Mipadi, Grafen, Wainstead, JLaTondre, Tommymorgan, Chris Chittleborough, AtomCrusher, SmackBot, Timoey, Aardvark92, Jdorner, Thumperward, OrangeDog, Frap, Cybercobra, Mwtoews, Acdx, Daniel.Cardenas, ArglebargleIV, Cydebot, Mblumber, Cinderblock63, Fyedernoggersnodden, Anupam, Nemnkim, Gstein, GreyTeardrop, Cander0000, Krotty, McSly, Woodjr, Mercurywoodrose, Bcantoni, Andy Dingley, Peter.vanroose, Drmies, Alexbot, Arjayay, The-verver, XLinkBot, Kolyma, Maximgr, Deineka, Addbot, Mortense, S4saurabh, Jncraton, MrOllie, Mdnahas, Jbryanscott,

Luckas-bot, Yobot, Midinastasurazz, JackPotte, Vanger13, AnomieBOT, Jim1138, Materialscientist, Pmiossec, Citation bot, ArthurBot, Xqbot, Santiagobasulto, Yadavjpr, Cgraysontx, Ciges, FrescoBot, FalconL, IO Device, Ksato9700, Sae1962, Tdmackey, Freenerd, Jadave234, Biktora, Rollins83, DASHBot, Dewritech, GoingBatty, Peaceray, Driftx, Arthurjulian, Werieth, ZéroBot, Al3xpopescu, Kylemurph, Grossenhayn, Dstainer, Billmantisco, ClueBot NG, Nmilford, Ben morphett, Elisiariocouto, Kinglarvae, Helpful Pixie Bot, YogiWanKenobi, Slebresne, BG19bot, Mark Arsten, IluvatarBot, Andrewllavore, Mmozum, Diglio.simoni, BattyBot, RichardMills65, Perlscriptgurubot, ChrisGualtieri, TheJJJunk, Viocomnetworks, Clydewylam, Hoestmelankoli, Samarthgahire, Jamesx12345, CJGarner, RobinUS2, Jonathanbellis, Wikiuser298, Jimtarber, Stuartmccaul, Stather, Mfiguiere, Dough34, Sebastibe, Khyryll, Dabron, Samohtm, Spiesche, Textractor, Wehanw, FLGMwt, Elishaoren, Andlpa63, Jericevans, Kevin at aerospike, BD2412bot, Wesko, Marissabell, Jorgebg, Barmic and Anonymous: 172

- **Berkeley DB** *Source:* https://en.wikipedia.org/wiki/Berkeley_DB?oldid=692113739 *Contributors:* Damian Yerrick, AxelBoldt, Bryan Derksen, Taw, Andre Engels, Nate Silva, DavidSJ, Danja, Phoe6, KAMiKAZOW, Docu, Mxn, GregRobson, Jay, Johnleach, Smithd, Paul W, Peak, Merovingian, Tobias Bergemann, Centrx, BenFrantzDale, AlistairMcMillan, Mendel, Rworsnop, Am088, Oneiros, Alobodig, Bumm13, Arunkv, EagleOne, RossPatterson, Rich Farmbrough, Mecanismo, Abelson, Gronky, Jarsyl, Elwikipedista~enwiki, Mike Schwartz, TommyG, JesseHogan, Justinc, Beinsane, Lystrata, TheParanoidOne, Interiot, GregorB, Tom W.M., Tokek, Turnstep, Kesla, Graham87, Art Cancro, Qwertyus, Nuptsey, Ligulem, Gene Wood, Zero0w, FlaBot, Vsion, Kmorozov, Intgr, Jac099, YurikBot, Rylz, Gaius Cornelius, SamJohnston, DragonHawk, DavidConrad, Welsh, Gregburd, Echiner~enwiki, JLaTondre, Berlotti, ViperSnake151, SmackBot, Reedy, Vald, KiloByte, Thumperward, Jerome Charles Potts, Letdorf, Eliyahu S, Where, TPO-bot, AThing, Guyjohnston, Jeberle, Rafert, Fatespeaks, Aarktica, Riffic, Eliashc, Tawkerbot2, 00112233, FatalError, Raysonho, Cydebot, Wernight, Danarmstrong, Gioto, Gilson.soares, Marokwitz, Mk*, Skarkkai, Jelloman, MER-C, Wootery, Ff1959, Cander0000, Gwern, Ariel., CommonsDelinker, Lordgilman, Rivenmyst137, Rajankila, Limn, Highlandsun, UnitedStatesian, Pcolby, Jerryobject, Oerdnj, Dead paulie, Toothrot, Explicit, Mild Bill Hiccup, Kilessan, Ecobun, Feline Hymnic, GreenGourd, Rhododendrites, SoxBot, TobiasPersson, Manorhill, Paulsheer, Brentsmith101, Drhowarddrfine, Twimoki, Legobot, Yobot, Legobot II, AnomieBOT, Arcoro01, Rubinbot, Götz, Noelkoutlis, Edrandall, Drilnoth, Scientes, UrusHyby, FrescoBot, HJ Mitchell, Nathan 314159, GoodenM, I dream of horses, Skyerise, SchreyP, Tokmeserdar, JnRouvignac, Figarobdbxml, McShark, Bolerio, DigitalKiwi, Ebrambot, Staticd, Mikhail Ryazanov, Dexp, Be..anyone, BG19bot, Compfreak7, Gm246, Chrisxue815, ChrisGualtieri, MartinMichlmayr, Hoestmelankoli, LennyWikidata, Apgiannakidis, ScotXW, Shinydiscoball and Anonymous: 123
- **Memcached** *Source:* https://en.wikipedia.org/wiki/Memcached?oldid=692531683 *Contributors:* AxelBoldt, Magnus Manske, Bryan Derksen, Tim Starling, Lemming, Angela, Den fjättrade ankan~enwiki, Ijon, Alaric, Ehn, Markhurd, Grendelkhan, Toreau, Ayucat, Rrjanbiah, TittoAssini, Mattflaschen, David Gerard, Jpo, Christopherlin, Oneishy, Neilc, Pgan002, Fpahl, Euphoria, Beginning, Gcanyon, Thorwald, Rfl, Rich Farmbrough, Gronky, Bender235, Enyo, CanisRufus, John Vandenberg, Giraffedata, Gary, Halsteadk, Diego Moya, GreggHilferding, Stephan Leeds, Danhash, Foolswisdom, Lime~enwiki, Simetrical, Barrylb, Jacobolus, Bratsche, Apokrif, MrSomeone, Rjwilmsi, Pleiotrop3, Pmc, Mahlon, Jnutting512, Anrie Nord, Pinecar, Phorque, YoavShapira, Rsrikanth05, SamJohnston, Corevette, Gslin, SmackBot, Mark Tranchant, Faisal.akeel, Oben, DStoykov, Morte, Oli Filth, ClaudiaM, Frap, Hermzz, T.J. Crowder, BrentRockwood, A5b, Evlekis, Thejerm, Sembiance, Tawkerbot2, Chnv, HDCase, Raysonho, Devis, Jasoncalacanis, Colorprobe, Grubbiv, Johnnicely, Thijs!bot, Dasani, Plausible deniability, Cherianthomas, Spotnyk, Hcobb, Mordere, Marokwitz, Pensador82, Falcor84, Mange01, Charlesyoung, AntiSpamBot, Tarinth, HansRoht, LastChanceToBe, TXiKiBoT, AgamemnonZ, Mfriedenhagen, Zkarab, BrianAker, Cnilep, Hyuu 91, Aednichols, NightRave, Monkeychubs, SHINERJ6, Kl4m-AWB, Gtstricky, Pauljchang, Herbert1000, SF007, Qgil-WMF, Addbot, ToikeOike, Raykrueger, Peridon, OlEnglish, Pjherrero, Legobot, Luckas-bot, Yobot, Timeroot, Kowser, Bub's, AnomieBOT, Materialscientist, Pmiossec, Neurocod, DataWraith, The Evil IP address, PlaysWithLife, FrescoBot, Natishalom, Enes1177, Sevi81, Jfmantis, Ingenthr, Majidkhan59, EmausBot, SBachenberg, Ajraddatz, Wikipediancag, ZéroBot, ILYA INDIGO, Javiermarinros, Zephyrus Tavvier, Smcquay, ClueBot NG, Webkul, Danim, Kronigon1, Helpful Pixie Bot, Mhmhk, Compfreak7, CitationCleanerBot, OmarMOthman, Papowell, JYBot, Dexbot, Altered Walter, RaphaelQS, Bitobor, ArmitageAmy, Mindskt, ScotXW, Dudeprgm, Kevin at aerospike, Gekart and Anonymous: 171
- **BigTable** *Source:* https://en.wikipedia.org/wiki/BigTable?oldid=692084407 *Contributors:* Jpatokal, Hashar, Greenrd, Bearcat, Bkonrad, Jason Quinn, Adpowers, Macrakis, Khalid hassani, Pgan002, Jeremykemp, Imroy, Discospinster, Brianhe, Pmsyyz, Bender235, Nileshbansal, Marudubshinki, Zoz, Rjwilmsi, Vegaswikian, Intgr, Wavelength, Cliffb, Markpeak, Corevette, Moe Epsilon, DeadEyeArrow, Elkman, Georgewilliamherbert, Matt Heard, SmackBot, Gnangarra, Drttm, Hmains, Amux, Fintler, JennyRad, Jeskeca, Audriusa, Cybercobra, Dwchin, Nakon, EIFY, A5b, Michael miceli, Larrymcp, Skim1420, John Reed Riley, Raysonho, Phoib, Franzks, Harrigan, Oo7565, Cydebot, Nearfar, Heroeswithmetaphors, Gstein, Benstown, Exerda, Gwern, Erik s paulson, Sounil, Rdquay, Vincent Lextrait, TXiKiBoT, Andy Dingley, Treekids, Dstrube, Captin411, ArlenCuss, Alexbot, Whereiswally, Replysixty, XLinkBot, Addbot, Tanhabot, Luckas-bot, Yobot, Legobot II, Wonderfl, Masharabinovich, AnomieBOT, ArthurBot, Quebec99, DataWraith, Ogendarf, Thehelpfulbot, Mateuszb, FrescoBot, Mark Renier, Sae1962, Wordstext, RedBot, MastiBot, Barauswald, Trappist the monk, Jmspaggi, DanielWaterworth, Pirroh, Ebrambot, FinalRapture, ClueBot NG, Dfarrell07, Barry K. Nathan, Danim, FreePeter3000, BG19bot, Mgr493, AvocatoBot, Goolasso, BattyBot, Paul2520, Monkbot, Seano314, Aashrayarora, Jefesaurus and Anonymous: 99

10.2 Images

- **File:ASF-logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/c/cd/ASF-logo.svg *License:* Apache License 2.0 *Contributors:* http://www.apache.org/ *Original artist:* Apache Software Foundation (ASF)
- **File:Ambox_important.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/b/b4/Ambox_important.svg *License:* Public domain *Contributors:* Own work, based off of Image:Ambox scales.svg *Original artist:* Dsmurat (talk · contribs)
- **File:BabbageKeyValueCard.tiff** *Source:* https://upload.wikimedia.org/wikipedia/commons/7/74/BabbageKeyValueCard.tiff *License:* Public domain *Contributors:* https://ia801408.us.archive.org/0/items/passagesfromlif01babbgoog/passagesfromlif01babbgoog.pdf, https://en.wikisource.org/wiki/Page:Babbage_-_Passages_from_the_Life_of_a_Philosopher.djvu/139 *Original artist:* Charles Babbage
- **File:CO-ScheduleClasses.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/db/CO-ScheduleClasses.jpg *License:* Attribution *Contributors:* Yoav Raz (1990): *The Principle of Commitment Ordering, or Guaranteeing Serializability in a Heterogeneous Environment of Multiple Autonomous Resource Managers Using Atomic Commitment.* DEC-TR 841, Digital Equipment Corporation, November 1990; Yoav Raz (1995): *The Principle of Commitment Ordering*, in yoavraz.googlepages.com *Original artist:* Yoav Raz

- **File:CodasylB.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d6/CodasylB.png *License:* CC-BY-SA-3.0 *Contributors:* "CIM: Principles of Computer Integrated Manufacturing", Jean-Baptiste Waldner, John Wiley & Sons, 1992 *Original artist:* Jean-Baptiste Waldner
- **File:Commons-logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/4/4a/Commons-logo.svg *License:* ? *Contributors:* ? *Original artist:* ?
- **File:Computer-aj_aj_ashton_01.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d7/Desktop_computer_clipart_-_Yellow_theme.svg *License:* CC0 *Contributors:* https://openclipart.org/detail/105871/computeraj-aj-ashton-01 *Original artist:* AJ from openclipart.org
- **File:Crystal_Clear_app_database.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/4/40/Crystal_Clear_app_database.png *License:* LGPL *Contributors:* All Crystal Clear icons were posted by the author as LGPL on kde-look; *Original artist:* Everaldo Coelho and YellowIcon;
- **File:Database_models.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/3/3b/Database_models.jpg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Marcel Douwe Dekker
- **File:Disambig_gray.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/5/5f/Disambig_gray.svg *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:Edit-clear.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/f/f2/Edit-clear.svg *License:* Public domain *Contributors:* The *Tango! Desktop Project*. *Original artist:*
  The people from the Tango! project. And according to the meta-data in the file, specifically: "Andreas Nilsson, and Jakub Steiner (although minimally)."
- **File:Folder_Hexagonal_Icon.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/4/48/Folder_Hexagonal_Icon.svg *License:* Cc-by-sa-3.0 *Contributors:* ? *Original artist:* ?
- **File:Free_Software_Portal_Logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/6/67/Nuvola_apps_emacs_vector.svg *License:* LGPL *Contributors:*
- Nuvola_apps_emacs.png *Original artist:* Nuvola_apps_emacs.png: David Vignoni
- **File:Gnome-searchtool.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/1/1e/Gnome-searchtool.svg *License:* LGPL *Contributors:* http://ftp.gnome.org/pub/GNOME/sources/gnome-themes-extras/0.9/gnome-themes-extras-0.9.0.tar.gz *Original artist:* David Vignoni
- **File:Helenos_for_Apache_Cassandra.PNG** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/27/Helenos_for_Apache_Cassandra.PNG *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* JackPotte
- **File:HelloWorld.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/28/HelloWorld.svg *License:* Public domain *Contributors:* Own work *Original artist:* Wooptoo
- **File:Internet_map_1024.jpg** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d2/Internet_map_1024.jpg *License:* CC BY 2.5 *Contributors:* Originally from the English Wikipedia; description page is/was here. *Original artist:* The Opte Project
- **File:LampFlowchart.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/9/91/LampFlowchart.svg *License:* CC-BY-SA-3.0 *Contributors:* vector version of Image:LampFlowchart.png *Original artist:* svg by Booyabazooka
- **File:Memcached.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/2/27/Memcached.svg *License:* Fair use *Contributors:*
  The logo may be obtained from Memcached.
  Original artist: ?
- **File:Merge-arrow.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/aa/Merge-arrow.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Merge-arrows.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/5/52/Merge-arrows.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Mergefrom.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/0/0f/Mergefrom.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:MongoDB-Logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/4/45/MongoDB-Logo.svg *License:* Fair use *Contributors:*
  The logo is from the https://www.mongodb.com/brand-resources website. *Original artist:* ?
- **File:Node.js_logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/d/d9/Node.js_logo.svg *License:* Public domain *Contributors:* http://nodejs.org/logos *Original artist:* node.js authors
- **File:Object-Oriented_Model.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/7/7c/Object-Oriented_Model.svg *License:* Public domain *Contributors:* Data Integration Glossary. *Original artist:*
  - U.S. Department of Transportation
  - vectorization: Own work
- **File:Office-book.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/a8/Office-book.svg *License:* Public domain *Contributors:* This and myself. *Original artist:* Chris Down/Tango project
- **File:People_icon.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/3/37/People_icon.svg *License:* CC0 *Contributors:* OpenClipart *Original artist:* OpenClipart
- **File:Portal-puzzle.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/f/fd/Portal-puzzle.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Postgresql_elephant.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/29/Postgresql_elephant.svg *License:* BSD *Contributors:* http://pgfoundry.org/docman/?group_id=1000089 *Original artist:* Jeff MacDonald

- **File:Question_book-new.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/9/99/Question_book-new.svg *License:* Cc-by-sa-3.0 *Contributors:*
  Created from scratch in Adobe Illustrator. Based on Image:Question book.png created by User:Equazcion *Original artist:*
  Tkgd2007
- **File:Redis_Logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/en/6/6b/Redis_Logo.svg *License:* Fair use *Contributors:* http://redis.io/images/redis-logo.svg *Original artist:* Carlos Prioglio created the logo for the copyright owner Salvatore Sanfilippo, lead developer of Redis.
- **File:Relational_key_SVG.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/4/4c/Relational_key_SVG.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* IkamusumeFan
- **File:Robomongo_0.8.5_-_insertion.png** *Source:* https://upload.wikimedia.org/wikipedia/commons/4/48/Robomongo_0.8.5_-_insertion.png *License:* CC BY-SA 4.0 *Contributors:* Own work *Original artist:* JackPotte
- **File:Star-schema-example.png** *Source:* https://upload.wikimedia.org/wikipedia/en/f/fe/Star-schema-example.png *License:* CC-BY-SA-3.0 *Contributors:*
  I created this work entirely by myself.
  Original artist:
  SqlPac (talk)
- **File:Symbol_book_class2.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/8/89/Symbol_book_class2.svg *License:* CC BY-SA 2.5 *Contributors:* Mad by Lokal_Profil by combining: *Original artist:* Lokal_Profil
- **File:Text_document_with_red_question_mark.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/a/a4/Text_document_with_red_question_mark.svg *License:* Public domain *Contributors:* Created by bdesham with Inkscape; based upon Text-x-generic.svg from the Tango project. *Original artist:* Benjamin D. Esham (bdesham)
- **File:Three-phase_commit_diagram.png** *Source:* https://upload.wikimedia.org/wikipedia/en/3/39/Three-phase_commit_diagram.png *License:* CC-BY-3.0 *Contributors:* ? *Original artist:* ?
- **File:Traditional_View_of_Data_SVG.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/4/4d/Traditional_View_of_Data_SVG.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* IkamusumeFan
- **File:Unbalanced_scales.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/f/fe/Unbalanced_scales.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Wikibooks-logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/f/fa/Wikibooks-logo.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* User:Bastique, User:Ramac et al.
- **File:Wikinews-logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/2/24/Wikinews-logo.svg *License:* CC BY-SA 3.0 *Contributors:* This is a cropped version of Image:Wikinews-logo-en.png. *Original artist:* Vectorized by Simon 01:05, 2 August 2006 (UTC) Updated by Time3000 17 April 2007 to use official Wikinews colours and appear correctly on dark backgrounds. Originally uploaded by Simon.
- **File:Wikiquote-logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/f/fa/Wikiquote-logo.svg *License:* Public domain *Contributors:* ? *Original artist:* ?
- **File:Wikisource-logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/4/4c/Wikisource-logo.svg *License:* CC BY-SA 3.0 *Contributors:* Rei-artur *Original artist:* Nicholas Moreau
- **File:Wikiversity-logo-Snorky.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/1/1b/Wikiversity-logo-en.svg *License:* CC BY-SA 3.0 *Contributors:* Own work *Original artist:* Snorky
- **File:Wikiversity-logo.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/9/91/Wikiversity-logo.svg *License:* CC BY-SA 3.0 *Contributors:* Snorky (optimized and cleaned up by verdy_p) *Original artist:* Snorky (optimized and cleaned up by verdy_p)
- **File:Wiktionary-logo-en.svg** *Source:* https://upload.wikimedia.org/wikipedia/commons/f/f8/Wiktionary-logo-en.svg *License:* Public domain *Contributors:* Vector version of Image:Wiktionary-logo-en.png. *Original artist:* Vectorized by Fvasconcellos (talk · contribs), based on original logo tossed together by Brion Vibber

10.3 Content license

- Creative Commons Attribution-Share Alike 3.0