# Glossary

Please note that the definitions in this glossary are short and simple, intended to convey the core idea but not the full subtleties of a term. For more detail, please follow the references into the main text.

**asynchronous**
: Not waiting for something to complete (e.g., sending data over the network to another node), and not making any assumptions about how long it is going to take. See "Synchronous Versus Asynchronous Replication" on page 153, "Synchronous Versus Asynchronous Networks" on page 284, and "System Model and Reality" on page 306.

**atomic**

1. In the context of concurrent operations: describing an operation that appears to take effect at a single point in time, so another concurrent process can never encounter the operation in a "half-finished" state. See also *isolation*.
2. In the context of transactions: grouping together a set of writes that must either all be committed or all be rolled back, even if faults occur. See "Atomicity" on page 223 and "Atomic Commit and Two-Phase Commit (2PC)" on page 354.

**backpressure**
: Forcing the sender of some data to slow down because the recipient cannot keep up with it. Also known as *flow control*. See "Messaging Systems" on page 441.

**batch process**
: A computation that takes some fixed (and usually large) set of data as input and produces some other data as output, without modifying the input. See Chapter 10.

**bounded**
: Having some known upper limit or size. Used for example in the context of network delay (see "Timeouts and Unbounded Delays" on page 281) and datasets (see the introduction to Chapter 11).

**Byzantine fault**
: A node that behaves incorrectly in some arbitrary way, for example by sending contradictory or malicious messages to other nodes. See "Byzantine Faults" on page 304.

**cache**
: A component that remembers recently used data in order to speed up future reads of the same data. It is generally not complete: thus, if some data is missing from the cache, it has to be fetched from some underlying, slower data storage

system that has a complete copy of the data.

**CAP theorem**
A widely misunderstood theoretical result that is not useful in practice. See "The CAP theorem" on page 336.

**causality**
The dependency between events that arises when one thing "happens before" another thing in a system. For example, a later event that is in response to an earlier event, or builds upon an earlier event, or should be understood in the light of an earlier event. See "The "happens-before" relationship and concurrency" on page 186 and "Ordering and Causality" on page 339.

**consensus**
A fundamental problem in distributed computing, concerning getting several nodes to agree on something (for example, which node should be the leader for a database cluster). The problem is much harder than it seems at first glance. See "Fault-Tolerant Consensus" on page 364.

**data warehouse**
A database in which data from several different OLTP systems has been combined and prepared to be used for analytics purposes. See "Data Warehousing" on page 91.

**declarative**
Describing the properties that something should have, but not the exact steps for how to achieve it. In the context of queries, a query optimizer takes a declarative query and decides how it should best be executed. See "Query Languages for Data" on page 42.

**denormalize**
To introduce some amount of redundancy or duplication in a *normalized* dataset, typically in the form of a *cache* or *index*, in order to speed up reads. A denormalized value is a kind of precomputed query result, similar to a materialized view. See "Single-Object and Multi-Object Operations" on page 228 and "Deriving several views from the same event log" on page 461.

**derived data**
A dataset that is created from some other data through a repeatable process, which you could run again if necessary. Usually, derived data is needed to speed up a particular kind of read access to the data. Indexes, caches, and materialized views are examples of derived data. See the introduction to Part III.

**deterministic**
Describing a function that always produces the same output if you give it the same input. This means it cannot depend on random numbers, the time of day, network communication, or other unpredictable things.

**distributed**
Running on several nodes connected by a network. Characterized by *partial failures*: some part of the system may be broken while other parts are still working, and it is often impossible for the software to know what exactly is broken. See "Faults and Partial Failures" on page 274.

**durable**
Storing data in a way such that you believe it will not be lost, even if various faults occur. See "Durability" on page 226.

**ETL**
Extract–Transform–Load. The process of extracting data from a source database, transforming it into a form that is more suitable for analytic queries, and loading it into a data warehouse or batch processing system. See "Data Warehousing" on page 91.

**failover**
In systems that have a single leader, failover is the process of moving the leadership role from one node to another. See "Handling Node Outages" on page 156.

**fault-tolerant**
Able to recover automatically if something goes wrong (e.g., if a machine crashes or a network link fails). See "Reliability" on page 6.

**flow control**
See *backpressure*.

**follower**
A replica that does not directly accept any writes from clients, but only processes data changes that it receives from a leader. Also known as a *secondary*, *slave*, *read replica*, or *hot standby*. See "Leaders and Followers" on page 152.

**full-text search**
Searching text by arbitrary keywords, often with additional features such as matching similarly spelled words or synonyms. A full-text index is a kind of *secondary index* that supports such queries. See "Full-text search and fuzzy indexes" on page 88.

**graph**
A data structure consisting of *vertices* (things that you can refer to, also known as *nodes* or *entities*) and *edges* (connections from one vertex to another, also known as *relationships* or *arcs*). See "Graph-Like Data Models" on page 49.

**hash**
A function that turns an input into a random-looking number. The same input always returns the same number as output. Two different inputs are very likely to have two different numbers as output, although it is possible that two different inputs produce the same output (this is called a *collision*). See "Partitioning by Hash of Key" on page 203.

**idempotent**
Describing an operation that can be safely retried; if it is executed more than once, it has the same effect as if it was only executed once. See "Idempotence" on page 478.

**index**
A data structure that lets you efficiently search for all records that have a particular value in a particular field. See "Data Structures That Power Your Database" on page 70.

**isolation**
In the context of transactions, describing the degree to which concurrently executing transactions can interfere with each other. *Serializable* isolation provides the strongest guarantees, but weaker isolation levels are also used. See "Isolation" on page 225.

**join**
To bring together records that have something in common. Most commonly used in the case where one record has a reference to another (a foreign key, a document reference, an edge in a graph) and a query needs to get the record that the reference points to. See "Many-to-One and Many-to-Many Relationships" on page 33 and "Reduce-Side Joins and Grouping" on page 403.

**leader**
When data or a service is replicated across several nodes, the leader is the designated replica that is allowed to make changes. A leader may be elected through some protocol, or manually chosen by an administrator. Also known as the *primary* or *master*. See "Leaders and Followers" on page 152.

**linearizable**
Behaving as if there was only a single copy of data in the system, which is updated by atomic operations. See "Linearizability" on page 324.

**locality**
A performance optimization: putting several pieces of data in the same place if they are frequently needed at the same time. See "Data locality for queries" on page 41.

**lock**
A mechanism to ensure that only one thread, node, or transaction can access something, and anyone else who wants to access the same thing must wait until the lock is released. See "Two-Phase Locking (2PL)" on page 257 and "The leader and the lock" on page 301.

**log**
An append-only file for storing data. A *write-ahead log* is used to make a storage engine resilient against crashes (see "Making B-trees reliable" on page 82), a *log-structured* storage engine uses logs as its primary storage format (see "SSTables and LSM-Trees" on page 76), a *replication log* is used to copy writes from a leader to followers (see "Leaders and Followers" on page 152), and an *event log* can represent a data stream (see "Partitioned Logs" on page 446).

**materialize**
To perform a computation eagerly and write out its result, as opposed to calculating it on demand when requested. See "Aggregation: Data Cubes and Materialized Views" on page 101 and "Materialization of Intermediate State" on page 419.

**node**
An instance of some software running on a computer, which communicates with other nodes via a network in order to accomplish some task.

**normalized**
Structured in such a way that there is no redundancy or duplication. In a normalized database, when some piece of data changes, you only need to change it in one place, not many copies in many different places. See "Many-to-One and Many-to-Many Relationships" on page 33.

**OLAP**
Online analytic processing. Access pattern characterized by aggregating (e.g., count, sum, average) over a large number of records. See "Transaction Processing or Analytics?" on page 90.

**OLTP**
Online transaction processing. Access pattern characterized by fast queries that read or write a small number of records, usually indexed by key. See "Transaction Processing or Analytics?" on page 90.

**partitioning**
Splitting up a large dataset or computation that is too big for a single machine into smaller parts and spreading them across several machines. Also known as *sharding*. See Chapter 6.

**percentile**
A way of measuring the distribution of values by counting how many values are above or below some threshold. For example, the 95th percentile response time during some period is the time *t* such that 95% of requests in that period complete in less than *t*, and 5% take longer than *t*. See "Describing Performance" on page 13.

**primary key**
A value (typically a number or a string) that uniquely identifies a record. In many applications, primary keys are generated by the system when a record is created (e.g., sequentially or randomly); they are not usually set by users. See also *secondary index*.

**quorum**
The minimum number of nodes that need to vote on an operation before it can be considered successful. See "Quorums for reading and writing" on page 179.

**rebalance**
To move data or services from one node to another in order to spread the load fairly. See "Rebalancing Partitions" on page 209.

**replication**
Keeping a copy of the same data on several nodes (*replicas*) so that it remains

accessible if a node becomes unreachable. See Chapter 5.

**schema**
A description of the structure of some data, including its fields and datatypes. Whether some data conforms to a schema can be checked at various points in the data's lifetime (see "Schema flexibility in the document model" on page 39), and a schema can change over time (see Chapter 4).

**secondary index**
An additional data structure that is maintained alongside the primary data storage and which allows you to efficiently search for records that match a certain kind of condition. See "Other Indexing Structures" on page 85 and "Partitioning and Secondary Indexes" on page 206.

**serializable**
A guarantee that if several transactions execute concurrently, they behave the same as if they had executed one at a time, in some serial order. See "Serializability" on page 251.

**shared-nothing**
An architecture in which independent nodes—each with their own CPUs, memory, and disks—are connected via a conventional network, in contrast to shared-memory or shared-disk architectures. See the introduction to Part II.

**skew**

1. Imbalanced load across partitions, such that some partitions have lots of requests or data, and others have much less. Also known as *hot spots*. See "Skewed Workloads and Relieving Hot Spots" on page 205 and "Handling skew" on page 407.
2. A timing anomaly that causes events to appear in an unexpected, nonsequential order. See the discussions of *read skew* in "Snapshot Isolation and Repeatable Read" on page 237, *write skew* in "Write Skew and Phantoms" on page 246, and *clock skew* in "Timestamps for ordering events" on page 291.

**split brain**
A scenario in which two nodes simultaneously believe themselves to be the leader, and which may cause system guarantees to be violated. See "Handling Node Outages" on page 156 and "The Truth Is Defined by the Majority" on page 300.

**stored procedure**
A way of encoding the logic of a transaction such that it can be entirely executed on a database server, without communicating back and forth with a client during the transaction. See "Actual Serial Execution" on page 252.

**stream process**
A continually running computation that consumes a never-ending stream of events as input, and derives some output from it. See Chapter 11.

**synchronous**
The opposite of *asynchronous*.

**system of record**
A system that holds the primary, authoritative version of some data, also known as the *source of truth*. Changes are first written here, and other datasets may be derived from the system of record. See the introduction to Part III.

**timeout**
One of the simplest ways of detecting a fault, namely by observing the lack of a response within some amount of time. However, it is impossible to know whether a timeout is due to a problem with the remote node, or an issue in the network. See "Timeouts and Unbounded Delays" on page 281.

**total order**
A way of comparing things (e.g., timestamps) that allows you to always say which one of two things is greater and which one is lesser. An ordering in which

some things are incomparable (you cannot say which is greater or smaller) is called a *partial order*. See "The causal order is not a total order" on page 341.

**transaction**
Grouping together several reads and writes into a logical unit, in order to simplify error handling and concurrency issues. See Chapter 7.

**two-phase commit (2PC)**
An algorithm to ensure that several database nodes either all commit or all abort a transaction. See "Atomic Commit and Two-Phase Commit (2PC)" on page 354.

**two-phase locking (2PL)**
An algorithm for achieving serializable isolation that works by a transaction acquiring a lock on all data it reads or writes, and holding the lock until the end of the transaction. See "Two-Phase Locking (2PL)" on page 257.

**unbounded**
Not having any known upper limit or size. The opposite of *bounded*.