See Also API, client, Connector/C++, Connector/J, Connector/NET, Connector/ODBC.

Connector/C++

Connector/C++ 8.0 can be used to access MySQL servers that implement a document store, or in a traditional way using SQL queries. It enables development of C++ applications using X DevAPI, or plain C applications using X DevAPI for C. It also enables development of C++ applications that use the legacy JDBC-based API from Connector/C++ 1.1. For more information, see MySQL Connector/C++ 8.0 Developer Guide.

See Also client, connector, JDBC.

Connector/J

A **JDBC** driver that provides connectivity for **client** applications developed in the **Java** programming language. MySQL Connector/J is a JDBC Type 4 driver: a pure-Java implementation of the MySQL protocol that does not rely on the MySQL **client libraries**. For full details, see MySQL Connector/J 8.0 Developer Guide.

See Also client, client libraries, connector, Java, JDBC.

Connector/NET

A MySQL **connector** for developers writing applications using languages, technologies, and frameworks such as **C#**, **.NET**, **Mono**, **Visual Studio**, **ASP.net**, and **ADO.net**.

See Also ADO.NET, ASP.net, connector, C#, Mono, Visual Studio.

Connector/ODBC

The family of MySQL ODBC drivers that provide access to a MySQL database using the industry standard Open Database Connectivity (**ODBC**) API. Formerly called MyODBC drivers. For full details, see MySQL Connector/ODBC Developer Guide.

See Also connector, ODBC.

Connector/PHP

A version of the `mysql` and `mysqli` **APIs** for **PHP** optimized for the Windows operating system.

See Also connector, PHP, PHP API.

consistent read

A read operation that uses **snapshot** information to present query results based on a point in time, regardless of changes performed by other transactions running at the same time. If queried data has been changed by another transaction, the original data is reconstructed based on the contents of the **undo log**. This technique avoids some of the **locking** issues that can reduce **concurrency** by forcing transactions to wait for other transactions to finish.

With **REPEATABLE READ isolation level**, the snapshot is based on the time when the first read operation is performed. With **READ COMMITTED** isolation level, the snapshot is reset to the time of each consistent read operation.

Consistent read is the default mode in which `InnoDB` processes `SELECT` statements in **READ COMMITTED** and **REPEATABLE READ** isolation levels. Because a consistent read does not set any locks on the tables it accesses, other sessions are free to modify those tables while a consistent read is being performed on the table.

For technical details about the applicable isolation levels, see Section 15.7.2.3, “Consistent Nonlocking Reads”.

See Also concurrency, isolation level, locking, READ COMMITTED, REPEATABLE READ, snapshot, transaction, undo log.

constraint

An automatic test that can block database changes to prevent data from becoming inconsistent. (In computer science terms, a kind of assertion related to an invariant condition.) Constraints are a crucial component of the **ACID** philosophy, to maintain data consistency. Constraints supported by MySQL include **FOREIGN KEY constraints** and **unique constraints**.

See Also ACID, foreign key, unique constraint.

counter
A value that is incremented by a particular kind of `InnoDB` operation. Useful for measuring how busy a server is, troubleshooting the sources of performance issues, and testing whether changes (for example, to configuration settings or indexes used by queries) have the desired low-level effects. Different kinds of counters are available through **Performance Schema** tables and **INFORMATION_SCHEMA** tables, particularly `INFORMATION_SCHEMA.INNODB_METRICS`.
See Also INFORMATION_SCHEMA, metrics counter, Performance Schema.

covering index
An **index** that includes all the columns retrieved by a query. Instead of using the index values as pointers to find the full table rows, the query returns values from the index structure, saving disk I/O. `InnoDB` can apply this optimization technique to more indexes than MyISAM can, because `InnoDB` **secondary indexes** also include the **primary key** columns. `InnoDB` cannot apply this technique for queries against tables modified by a transaction, until that transaction ends.

Any **column index** or **composite index** could act as a covering index, given the right query. Design your indexes and queries to take advantage of this optimization technique wherever possible.
See Also column index, composite index, index, primary key, secondary index.

CPU-bound
A type of **workload** where the primary **bottleneck** is CPU operations in memory. Typically involves read-intensive operations where the results can all be cached in the **buffer pool**.
See Also bottleneck, buffer pool, workload.

crash
MySQL uses the term “crash” to refer generally to any unexpected **shutdown** operation where the server cannot do its normal cleanup. For example, a crash could happen due to a hardware fault on the database server machine or storage device; a power failure; a potential data mismatch that causes the MySQL server to halt; a **fast shutdown** initiated by the DBA; or many other reasons. The robust, automatic **crash recovery** for **InnoDB** tables ensures that data is made consistent when the server is restarted, without any extra work for the DBA.
See Also crash recovery, fast shutdown, InnoDB, shutdown.

crash recovery
The cleanup activities that occur when MySQL is started again after a **crash**. For **InnoDB** tables, changes from incomplete transactions are replayed using data from the **redo log**. Changes that were **committed** before the crash, but not yet written into the **data files**, are reconstructed from the **doublewrite buffer**. When the database is shut down normally, this type of activity is performed during shutdown by the **purge** operation.

During normal operation, committed data can be stored in the **change buffer** for a period of time before being written to the data files. There is always a tradeoff between keeping the data files up-to-date, which introduces performance overhead during normal operation, and buffering the data, which can make shutdown and crash recovery take longer.
See Also change buffer, commit, crash, data files, doublewrite buffer, InnoDB, purge, redo log.

CRUD
Acronym for “create, read, update, delete”, a common sequence of operations in database applications. Often denotes a class of applications with relatively simple database usage (basic **DDL**, **DML** and **query** statements in **SQL**) that can be implemented quickly in any language.
See Also DDL, DML, query, SQL.

cursor
An internal MySQL data structure that represents the result set of an SQL statement. Often used with **prepared statements** and **dynamic SQL**. It works like an iterator in other high-level languages, producing each value from the result set as requested.

Although SQL usually handles the processing of cursors for you, you might delve into the inner workings when dealing with performance-critical code.
See Also dynamic SQL, prepared statement, query.

D

data definition language
See DDL.

data dictionary
Metadata that keeps track of database objects such as **tables**, **indexes**, and table **columns**. For the MySQL data dictionary, introduced in MySQL 8.0, metadata is physically located in `InnoDB` **file-per-table** tablespace files in the `mysql` database directory. For the `InnoDB` data dictionary, metadata is physically located in the `InnoDB` **system tablespace**.

Because the **MySQL Enterprise Backup** product always backs up the `InnoDB` system tablespace, all backups include the contents of the `InnoDB` data dictionary.
See Also column, file-per-table, .frm file, index, MySQL Enterprise Backup, system tablespace, table.

data directory
The directory under which each MySQL **instance** keeps the **data files** for `InnoDB` and the directories representing individual databases. Controlled by the `datadir` configuration option.
See Also data files, instance.

data files
The files that physically contain **table** and **index** data.

The `InnoDB` **system tablespace**, which holds the `InnoDB` **data dictionary** and is capable of holding data for multiple `InnoDB` tables, is represented by one or more `.ibdata` data files.

File-per-table tablespaces, which hold data for a single `InnoDB` table, are represented by a `.ibd` data file.

General tablespaces (introduced in MySQL 5.7.6), which can hold data for multiple `InnoDB` tables, are also represented by a `.ibd` data file.
See Also data dictionary, file-per-table, general tablespace, .ibd file, ibdata file, index, system tablespace, table, tablespace.

data manipulation language
See DML.

data warehouse
A database system or application that primarily runs large **queries**. The read-only or read-mostly data might be organized in **denormalized** form for query efficiency. Can benefit from the optimizations for **read-only transactions** in MySQL 5.6 and higher. Contrast with **OLTP**.
See Also denormalized, OLTP, query, read-only transaction.

database
Within the MySQL **data directory**, each database is represented by a separate directory. The InnoDB **system tablespace**, which can hold table data from multiple databases within a MySQL **instance**, is kept in **data files** that reside outside of individual database directories. When **file-per-table** mode is enabled, the **.ibd files** representing individual InnoDB tables are stored inside the database directories unless created elsewhere using the `DATA DIRECTORY` clause. General tablespaces, introduced in MySQL 5.7.6, also hold table data in **.ibd files**. Unlike file-per-table **.ibd files**, general tablespace **.ibd files** can hold table data from multiple databases within a MySQL **instance**, and can be assigned to directories relative to or independent of the MySQL data directory.

For long-time MySQL users, a database is a familiar notion. Users coming from an Oracle Database background may find that the MySQL meaning of a database is closer to what Oracle Database calls a **schema**.
See Also data files, file-per-table, .ibd file, instance, schema, system tablespace.

DCL
Data control language, a set of **SQL** statements for managing privileges. In MySQL, consists of the `GRANT` and `REVOKE` statements. Contrast with **DDL** and **DML**.
See Also DDL, DML, SQL.

DDEX provider
A feature that lets you use the data design tools within **Visual Studio** to manipulate the schema and objects within a MySQL database. For MySQL applications using **Connector/NET**, the MySQL Visual Studio Plugin acts as a DDEX provider with MySQL 5.0 and later.
See Also Visual Studio.

DDL
Data definition language, a set of **SQL** statements for manipulating the database itself rather than individual table rows. Includes all forms of the `CREATE`, `ALTER`, and `DROP` statements. Also includes the `TRUNCATE` statement, because it works differently than a `DELETE FROM table_name` statement, even though the ultimate effect is similar.

DDL statements automatically **commit** the current **transaction**; they cannot be **rolled back**.

The `InnoDB` online DDL feature enhances performance for `CREATE INDEX`, `DROP INDEX`, and many types of `ALTER TABLE` operations. See Section 15.12, "InnoDB and Online DDL" for more information. Also, the `InnoDB` file-per-table setting can affect the behavior of `DROP TABLE` and `TRUNCATE TABLE` operations.

Contrast with **DML** and **DCL**.
See Also commit, DCL, DML, file-per-table, rollback, SQL, transaction.

deadlock
A situation where different **transactions** are unable to proceed, because each holds a **lock** that the other needs. Because both transactions are waiting for a resource to become available, neither one ever releases the locks it holds.

A deadlock can occur when the transactions lock rows in multiple tables (through statements such as `UPDATE` or `SELECT ... FOR UPDATE`), but in the opposite order. A deadlock can also occur when such statements lock ranges of index records and **gaps**, with each transaction acquiring some locks but not others due to a timing issue.

For background information on how deadlocks are automatically detected and handled, see Section 15.7.5.2, "Deadlock Detection". For tips on avoiding and recovering from deadlock conditions, see Section 15.7.5.3, "How to Minimize and Handle Deadlocks".
See Also gap, lock, transaction.

deadlock detection
A mechanism that automatically detects when a **deadlock** occurs, and automatically **rolls back** one of the **transactions** involved (the **victim**). Deadlock detection can be disabled using the `innodb_deadlock_detect` configuration option.
See Also deadlock, rollback, transaction, victim.

delete
When `InnoDB` processes a `DELETE` statement, the rows are immediately marked for deletion and no longer are returned by queries. The storage is reclaimed sometime later, during the periodic garbage collection known as the **purge** operation. For removing large quantities of data, related operations with their own performance characteristics are **TRUNCATE** and **DROP**.
See Also drop, purge, truncate.

delete buffering
The technique of storing changes to secondary index pages, resulting from `DELETE` operations, in the **change buffer** rather than writing the changes immediately, so that the physical writes can be performed to minimize random I/O. (Because delete operations are a two-step process, this operation buffers the write that normally marks an index record for deletion.) It is one of the types of **change buffering**; the others are **insert buffering** and **purge buffering**.
See Also change buffer, change buffering, insert buffer, insert buffering, purge buffering.

denormalized
A data storage strategy that duplicates data across different tables, rather than linking the tables with **foreign keys** and **join** queries. Typically used in **data warehouse** applications, where the data is not updated

after loading. In such applications, query performance is more important than making it simple to maintain consistent data during updates. Contrast with **normalized**.
See Also data warehouse, foreign key, join, normalized.

descending index
A type of **index** where index storage is optimized to process `ORDER BY column DESC` clauses.
See Also index.

dictionary object cache
The dictionary object cache stores previously accessed **data dictionary** objects in memory to enable object reuse and minimize disk I/O. An **LRU**-based eviction strategy is used to evict least recently used objects from memory. The cache is comprised of several partitions that store different object types.

For more information, see Section 14.4, “Dictionary Object Cache”.
See Also data dictionary, LRU.

dirty page
A **page** in the `InnoDB` **buffer pool** that has been updated in memory, where the changes are not yet written (**flushed**) to the **data files**. The opposite of a **clean page**.
See Also buffer pool, clean page, data files, flush, page.

dirty read
An operation that retrieves unreliable data, data that was updated by another transaction but not yet **committed**. It is only possible with the **isolation level** known as **read uncommitted**.

This kind of operation does not adhere to the **ACID** principle of database design. It is considered very risky, because the data could be **rolled back**, or updated further before being committed; then, the transaction doing the dirty read would be using data that was never confirmed as accurate.

Its opposite is **consistent read**, where `InnoDB` ensures that a transaction does not read information updated by another transaction, even if the other transaction commits in the meantime.
See Also ACID, commit, consistent read, isolation level, READ UNCOMMITTED, rollback.

disk-based
A kind of database that primarily organizes data on disk storage (hard drives or equivalent). Data is brought back and forth between disk and memory to be operated upon. It is the opposite of an **in-memory database**. Although `InnoDB` is disk-based, it also contains features such as he **buffer pool**, multiple buffer pool instances, and the **adaptive hash index** that allow certain kinds of workloads to work primarily from memory.
See Also adaptive hash index, buffer pool, in-memory database.

disk-bound
A type of **workload** where the primary **bottleneck** is disk I/O. (Also known as **I/O-bound**.) Typically involves frequent writes to disk, or random reads of more data than can fit into the **buffer pool**.
See Also bottleneck, buffer pool, workload.

DML
Data manipulation language, a set of **SQL** statements for performing `INSERT`, `UPDATE`, and `DELETE` operations. The `SELECT` statement is sometimes considered as a DML statement, because the `SELECT ... FOR UPDATE` form is subject to the same considerations for **locking** as `INSERT`, `UPDATE`, and `DELETE`.

DML statements for an `InnoDB` table operate in the context of a **transaction**, so their effects can be **committed** or **rolled back** as a single unit.

Contrast with **DDL** and **DCL**.
See Also commit, DCL, DDL, locking, rollback, SQL, transaction.

document id
In the `InnoDB` **full-text search** feature, a special column in the table containing the **FULLTEXT index**, to uniquely identify the document associated with each **ilist** value. Its name is `FTS_DOC_ID` (uppercase required). The column itself must be of `BIGINT UNSIGNED NOT NULL` type, with a unique index named

FTS_DOC_ID_INDEX. Preferably, you define this column when creating the table. If InnoDB must add the column to the table while creating a FULLTEXT index, the indexing operation is considerably more expensive.
See Also full-text search, FULLTEXT index, ilist.

doublewrite buffer
InnoDB uses a file flush technique called doublewrite. Before writing **pages** to the **data files**, InnoDB first writes them to a storage area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer have completed, does InnoDB write the pages to their proper positions in the data file. If there is an operating system, storage subsystem or mysqld process crash in the middle of a page write, InnoDB can find a good copy of the page from the doublewrite buffer during **crash recovery**.

Although data is always written twice, the doublewrite buffer does not require twice as much I/O overhead or twice as many I/O operations. Data is written to the buffer itself as a large sequential chunk, with a single fsync() call to the operating system.
See Also crash recovery, data files, page, purge.

drop
A kind of **DDL** operation that removes a schema object, through a statement such as DROP TABLE or DROP INDEX. It maps internally to an ALTER TABLE statement. From an InnoDB perspective, the performance considerations of such operations involve the time that the **data dictionary** is locked to ensure that interrelated objects are all updated, and the time to update memory structures such as the **buffer pool**. For a **table**, the drop operation has somewhat different characteristics than a **truncate** operation (TRUNCATE TABLE statement).
See Also buffer pool, data dictionary, DDL, table, truncate.

DSN
Acronym for “Database Source Name”. It is the encoding for **connection** information within **Connector/ODBC**. See Configuring a Connector/ODBC DSN on Windows for full details. It is the equivalent of the **connection string** used by **Connector/NET**.
See Also connection, connection string, Connector/NET, Connector/ODBC.

dynamic cursor
A type of **cursor** supported by **ODBC** that can pick up new and changed results when the rows are read again. Whether and how quickly the changes are visible to the cursor depends on the type of table involved (transactional or non-transactional) and the isolation level for transactional tables. Support for dynamic cursors must be explicitly enabled.
See Also cursor, ODBC.

dynamic row format
An InnoDB row format. Because long variable-length column values are stored outside of the page that holds the row data, it is very efficient for rows that include large objects. Since the large fields are typically not accessed to evaluate query conditions, they are not brought into the **buffer pool** as often, resulting in fewer I/O operations and better utilization of cache memory.

As of MySQL 5.7.9, the default row format is defined by innodb_default_row_format, which has a default value of DYNAMIC.

For additional information about InnoDB DYNAMIC row format, see DYNAMIC Row Format.
See Also buffer pool, file format, row format.

dynamic SQL
A feature that lets you create and execute **prepared statements** using more robust, secure, and efficient methods to substitute parameter values than the naive technique of concatenating the parts of the statement into a string variable.
See Also prepared statement.

dynamic statement
A **prepared statement** created and executed through **dynamic SQL**.
See Also dynamic SQL, prepared statement.

E

early adopter
: A stage similar to **beta**, when a software product is typically evaluated for performance, functionality, and compatibility in a non-mission-critical setting.
See Also beta.

Eiffel
: A programming language including many object-oriented features. Some of its concepts are familiar to **Java** and **C#** developers. For the open-source Eiffel **API** for MySQL, see Section 29.13, “MySQL Eiffel Wrapper”.
See Also API, C#, Java.

embedded
: The embedded MySQL server library (**libmysqld**) makes it possible to run a full-featured MySQL server inside a **client** application. The main benefits are increased speed and more simple management for embedded applications.
See Also client, libmysqld.

error log
: A type of **log** showing information about MySQL startup and critical runtime errors and **crash** information. For details, see Section 5.4.2, “The Error Log”.
See Also crash, log.

eviction
: The process of removing an item from a cache or other temporary storage area, such as the `InnoDB` **buffer pool**. Often, but not always, uses the **LRU** algorithm to determine which item to remove. When a **dirty page** is evicted, its contents are **flushed** to disk, and any dirty **neighbor pages** might be flushed also.
See Also buffer pool, dirty page, flush, LRU, neighbor page.

exception interceptor
: A type of **interceptor** for tracing, debugging, or augmenting SQL errors encountered by a database application. For example, the interceptor code could issue a `SHOW WARNINGS` statement to retrieve additional information, and add descriptive text or even change the type of the exception returned to the application. Because the interceptor code is only called when SQL statements return errors, it does not impose any performance penalty on the application during normal (error-free) operation.

  In **Java** applications using **Connector/J**, setting up this type of interceptor involves implementing the `com.mysql.jdbc.ExceptionInterceptor` interface, and adding a `exceptionInterceptors` property to the **connection string**.

  In **Visual Studio** applications using **Connector/NET**, setting up this type of interceptor involves defining a class that inherits from the `BaseExceptionInterceptor` class and specifying that class name as part of the connection string.
  See Also Connector/J, Connector/NET, interceptor, Java, Visual Studio.

exclusive lock
: A kind of **lock** that prevents any other **transaction** from locking the same row. Depending on the transaction **isolation level**, this kind of lock might block other transactions from writing to the same row, or might also block other transactions from reading the same row. The default `InnoDB` isolation level, **REPEATABLE READ**, enables higher **concurrency** by allowing transactions to read rows that have exclusive locks, a technique known as **consistent read**.
See Also concurrency, consistent read, isolation level, lock, REPEATABLE READ, shared lock, transaction.

extent
: A group of **pages** within a **tablespace**. For the default **page size** of 16KB, an extent contains 64 pages. In MySQL 5.6, the page size for an `InnoDB` instance can be 4KB, 8KB, or 16KB, controlled by the `innodb_page_size` configuration option. For 4KB, 8KB, and 16KB pages sizes, the extent size is always 1MB (or 1048576 bytes).

Support for 32KB and 64KB `InnoDB` page sizes was added in MySQL 5.7.6. For a 32KB page size, the extent size is 2MB. For a 64KB page size, the extent size is 4MB.

`InnoDB` features such as **segments**, **read-ahead** requests and the **doublewrite buffer** use I/O operations that read, write, allocate, or free data one extent at a time.
See Also doublewrite buffer, page, page size, read-ahead, segment, tablespace.

F

.frm file
A file containing the metadata, such as the table definition, of a MySQL table. `.frm` files were removed in MySQL 8.0 but are still used in earlier MySQL releases. In MySQL 8.0, data previously stored in `.frm` files is stored in **data dictionary** tables.
See Also data dictionary, MySQL Enterprise Backup, system tablespace.

failover
The ability to automatically switch to a standby server in the event of a failure. In the MySQL context, failover involves a standby database server. Often supported within **J2EE** environments by the application server or framework.
See Also Connector/J, J2EE.

Fast Index Creation
A capability first introduced in the InnoDB Plugin, now part of MySQL in 5.5 and higher, that speeds up creation of `InnoDB` **secondary indexes** by avoiding the need to completely rewrite the associated table. The speedup applies to dropping secondary indexes also.

Because index maintenance can add performance overhead to many data transfer operations, consider doing operations such as `ALTER TABLE ... ENGINE=INNODB` or `INSERT INTO ... SELECT * FROM ...` without any secondary indexes in place, and creating the indexes afterward.

In MySQL 5.6, this feature becomes more general. You can read and write to tables while an index is being created, and many more kinds of `ALTER TABLE` operations can be performed without copying the table, without blocking **DML** operations, or both. Thus in MySQL 5.6 and higher, this set of features is referred to as **online DDL** rather than Fast Index Creation.

For related information, see Section 15.12, “InnoDB and Online DDL”.
See Also DML, index, online DDL, secondary index.

fast shutdown
The default **shutdown** procedure for `InnoDB`, based on the configuration setting `innodb_fast_shutdown=1`. To save time, certain **flush** operations are skipped. This type of shutdown is safe during normal usage, because the flush operations are performed during the next startup, using the same mechanism as in **crash recovery**. In cases where the database is being shut down for an upgrade or downgrade, do a **slow shutdown** instead to ensure that all relevant changes are applied to the **data files** during the shutdown.
See Also crash recovery, data files, flush, shutdown, slow shutdown.

file format
The file format for `InnoDB` tables.
See Also file-per-table, .ibd file, ibdata file, row format.

file-per-table
A general name for the setting controlled by the `innodb_file_per_table` option, which is an important configuration option that affects aspects of `InnoDB` file storage, availability of features, and I/O characteristics. As of MySQL 5.6.7, `innodb_file_per_table` is enabled by default.

With the `innodb_file_per_table` option enabled, you can create a table in its own **.ibd file** rather than in the shared **ibdata files** of the **system tablespace**. When table data is stored in an individual **.ibd file**,

you have more flexibility to choose **row formats** required for features such as data **compression**. The `TRUNCATE TABLE` operation is also faster, and reclaimed space can be used by the operating system rather than remaining reserved for `InnoDB`.

The **MySQL Enterprise Backup** product is more flexible for tables that are in their own files. For example, tables can be excluded from a backup, but only if they are in separate files. Thus, this setting is suitable for tables that are backed up less frequently or on a different schedule.
See Also compressed row format, compression, file format, .ibd file, ibdata file, innodb_file_per_table, MySQL Enterprise Backup, row format, system tablespace.

fill factor
In an `InnoDB` **index**, the proportion of a **page** that is taken up by index data before the page is split. The unused space when index data is first divided between pages allows for rows to be updated with longer string values without requiring expensive index maintenance operations. If the fill factor is too low, the index consumes more space than needed, causing extra I/O overhead when reading the index. If the fill factor is too high, any update that increases the length of column values can cause extra I/O overhead for index maintenance. See Section 15.6.2.2, "The Physical Structure of an InnoDB Index" for more information.
See Also index, page.

fixed row format
This row format is used by the `MyISAM` storage engine, not by `InnoDB`. If you create an `InnoDB` table with the option `ROW_FORMAT=FIXED` in MySQL 5.7.6 or earlier, `InnoDB` uses the **compact row format** instead, although the `FIXED` value might still show up in output such as `SHOW TABLE STATUS` reports. As of MySQL 5.7.7, `InnoDB` returns an error if `ROW_FORMAT=FIXED` is specified.
See Also compact row format, row format.

flush
To write changes to the database files, that had been buffered in a memory area or a temporary disk storage area. The `InnoDB` storage structures that are periodically flushed include the **redo log**, the **undo log**, and the **buffer pool**.

Flushing can happen because a memory area becomes full and the system needs to free some space, because a **commit** operation means the changes from a transaction can be finalized, or because a **slow shutdown** operation means that all outstanding work should be finalized. When it is not critical to flush all the buffered data at once, `InnoDB` can use a technique called **fuzzy checkpointing** to flush small batches of pages to spread out the I/O overhead.
See Also buffer pool, commit, fuzzy checkpointing, redo log, slow shutdown, undo log.

flush list
An internal `InnoDB` data structure that tracks **dirty pages** in the **buffer pool**: that is, **pages** that have been changed and need to be written back out to disk. This data structure is updated frequently by `InnoDB` internal **mini-transactions**, and so is protected by its own **mutex** to allow concurrent access to the buffer pool.
See Also buffer pool, dirty page, LRU, mini-transaction, mutex, page, page cleaner.

foreign key
A type of pointer relationship, between rows in separate `InnoDB` tables. The foreign key relationship is defined on one column in both the **parent table** and the **child table**.

In addition to enabling fast lookup of related information, foreign keys help to enforce **referential integrity**, by preventing any of these pointers from becoming invalid as data is inserted, updated, and deleted. This enforcement mechanism is a type of **constraint**. A row that points to another table cannot be inserted if the associated foreign key value does not exist in the other table. If a row is deleted or its foreign key value changed, and rows in another table point to that foreign key value, the foreign key can be set up to prevent the deletion, cause the corresponding column values in the other table to become **null**, or automatically delete the corresponding rows in the other table.

One of the stages in designing a **normalized** database is to identify data that is duplicated, separate that data into a new table, and set up a foreign key relationship so that the multiple tables can be queried like a single table, using a **join** operation.

See Also child table, FOREIGN KEY constraint, join, normalized, NULL, parent table, referential integrity, relational.

FOREIGN KEY constraint
The type of **constraint** that maintains database consistency through a **foreign key** relationship. Like other kinds of constraints, it can prevent data from being inserted or updated if data would become inconsistent; in this case, the inconsistency being prevented is between data in multiple tables. Alternatively, when a **DML** operation is performed, `FOREIGN KEY` constraints can cause data in **child rows** to be deleted, changed to different values, or set to **null**, based on the `ON CASCADE` option specified when creating the foreign key.
See Also child table, constraint, DML, foreign key, NULL.

FTS
In most contexts, an acronym for **full-text search**. Sometimes in performance discussions, an acronym for **full table scan**.
See Also full table scan, full-text search.

full backup
A **backup** that includes all the **tables** in each MySQL **database**, and all the databases in a MySQL **instance**. Contrast with **partial backup**.
See Also backup, database, instance, partial backup, table.

full table scan
An operation that requires reading the entire contents of a table, rather than just selected portions using an **index**. Typically performed either with small lookup tables, or in data warehousing situations with large tables where all available data is aggregated and analyzed. How frequently these operations occur, and the sizes of the tables relative to available memory, have implications for the algorithms used in query optimization and managing the **buffer pool**.

The purpose of indexes is to allow lookups for specific values or ranges of values within a large table, thus avoiding full table scans when practical.
See Also buffer pool, index.

full-text search
The MySQL feature for finding words, phrases, Boolean combinations of words, and so on within table data, in a faster, more convenient, and more flexible way than using the SQL `LIKE` operator or writing your own application-level search algorithm. It uses the SQL function `MATCH()` and **FULLTEXT indexes**.
See Also FULLTEXT index.

FULLTEXT index
The special kind of **index** that holds the **search index** in the MySQL **full-text search** mechanism. Represents the words from values of a column, omitting any that are specified as **stopwords**. Originally, only available for `MyISAM` tables. Starting in MySQL 5.6.4, it is also available for **InnoDB** tables.
See Also full-text search, index, InnoDB, search index, stopword.

fuzzy checkpointing
A technique that **flushes** small batches of **dirty pages** from the **buffer pool**, rather than flushing all dirty pages at once which would disrupt database processing.
See Also buffer pool, dirty page, flush.

G

GA
"Generally available", the stage when a software product leaves **beta** and is available for sale, official support, and production use.
See Also beta.

GAC
Acronym for "Global Assembly Cache". A central area for storing libraries (**assemblies**) on a **.NET** system. Physically consists of nested folders, treated as a single virtual folder by the **.NET** CLR.
See Also .NET, assembly.

gap
A place in an InnoDB **index** data structure where new values could be inserted. When you lock a set of rows with a statement such as SELECT ... FOR UPDATE, InnoDB can create locks that apply to the gaps as well as the actual values in the index. For example, if you select all values greater than 10 for update, a gap lock prevents another transaction from inserting a new value that is greater than 10. The **supremum record** and **infimum record** represent the gaps containing all values greater than or less than all the current index values.
See Also concurrency, gap lock, index, infimum record, isolation level, supremum record.

gap lock
A **lock** on a **gap** between index records, or a lock on the gap before the first or after the last index record. For example, SELECT c1 FROM t WHERE c1 BETWEEN 10 and 20 FOR UPDATE; prevents other transactions from inserting a value of 15 into the column t.c1, whether or not there was already any such value in the column, because the gaps between all existing values in the range are locked. Contrast with **record lock** and **next-key lock**.

Gap locks are part of the tradeoff between performance and **concurrency**, and are used in some transaction **isolation levels** and not others.
See Also gap, infimum record, lock, next-key lock, record lock, supremum record.

general log
See general query log.

general query log
A type of **log** used for diagnosis and troubleshooting of SQL statements processed by the MySQL server. Can be stored in a file or in a database table. You must enable this feature through the general_log configuration option to use it. You can disable it for a specific connection through the sql_log_off configuration option.

Records a broader range of queries than the **slow query log**. Unlike the **binary log**, which is used for replication, the general query log contains SELECT statements and does not maintain strict ordering. For more information, see Section 5.4.3, “The General Query Log”.
See Also binary log, log, slow query log.

general tablespace
A shared InnoDB **tablespace** created using CREATE TABLESPACE syntax. General tablespaces can be created outside of the MySQL data directory, are capable of holding multiple **tables**, and support tables of all row formats. General tablespaces were introduced in MySQL 5.7.6.

Tables are added to a general tablespace using CREATE TABLE *tbl_name* ... TABLESPACE [=] *tablespace_name* or ALTER TABLE *tbl_name* TABLESPACE [=] *tablespace_name* syntax.

Contrast with **system tablespace** and **file-per-table** tablespace.

For more information, see Section 15.6.3.3, “General Tablespaces”.
See Also file-per-table, system tablespace, table, tablespace.

generated column
A column whose values are computed from an expression included in the column definition. A generated column can be **virtual** or **stored**.
See Also base column, stored generated column, virtual generated column.

generated stored column
See stored generated column.

generated virtual column
See virtual generated column.

Glassfish

See Also J2EE.

global temporary tablespace
: A *temporary tablespace* that stores *rollback segments* for changes made to user-created temporary tables.
See Also temporary tablespace.

global transaction
: A type of **transaction** involved in **XA** operations. It consists of several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends **ACID** properties "up a level" so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties.
See Also ACID, transaction, XA.

group commit
: An `InnoDB` optimization that performs some low-level I/O operations (log write) once for a set of **commit** operations, rather than flushing and syncing separately for each commit.
See Also binary log, commit.

GUID
: Acronym for "globally unique identifier", an ID value that can be used to associate data across different databases, languages, operating systems, and so on. (As an alternative to using sequential integers, where the same values could appear in different tables, databases, and so on referring to different data.) Older MySQL versions represented it as `BINARY(16)`. Currently, it is represented as `CHAR(36)`. MySQL has a `UUID()` function that returns GUID values in character format, and a `UUID_SHORT()` function that returns GUID values in integer format. Because successive GUID values are not necessarily in ascending sort order, it is not an efficient value to use as a primary key for large InnoDB tables.

H

hash index
: A type of **index** intended for queries that use equality operators, rather than range operators such as greater-than or `BETWEEN`. It is available for `MEMORY` tables. Although hash indexes are the default for `MEMORY` tables for historic reasons, that storage engine also supports **B-tree** indexes, which are often a better choice for general-purpose queries.

  MySQL includes a variant of this index type, the **adaptive hash index**, that is constructed automatically for `InnoDB` tables if needed based on runtime conditions.
  See Also adaptive hash index, B-tree, index, InnoDB.

HDD
: Acronym for "hard disk drive". Refers to storage media using spinning platters, usually when comparing and contrasting with **SSD**. Its performance characteristics can influence the throughput of a **disk-based** workload.
See Also disk-based, SSD.

heartbeat
: A periodic message that is sent to indicate that a system is functioning properly. In a **replication** context, if the **source** stops sending such messages, one of the **replicas** can take its place. Similar techniques can be used between the servers in a cluster environment, to confirm that all of them are operating properly.
See Also replication, source.

high-water mark
: A value representing an upper limit, either a hard limit that should not be exceeded at runtime, or a record of the maximum value that was actually reached. Contrast with **low-water mark**.
See Also low-water mark.

history list
: A list of **transactions** with delete-marked records scheduled to be processed by the `InnoDB` **purge** operation. Recorded in the **undo log**. The length of the history list is reported by the command `SHOW ENGINE INNODB STATUS`. If the history list grows longer than the value of the `innodb_max_purge_lag` configuration option, each **DML** operation is delayed slightly to allow the purge operation to finish **flushing** the deleted records.

Also known as **purge lag**.
See Also DML, flush, purge, purge lag, rollback segment, transaction, undo log.

hole punching
Releasing empty blocks from a page. The `InnoDB` **transparent page compression** feature relies on hole punching support. For more information, see Section 15.9.2, “InnoDB Page Compression”.
See Also sparse file, transparent page compression.

host
The network name of a database server, used to establish a **connection**. Often specified in conjunction with a **port**. In some contexts, the IP address `127.0.0.1` works better than the special name `localhost` for accessing a database on the same server as the application.
See Also connection, localhost, port.

hot
A condition where a row, table, or internal data structure is accessed so frequently, requiring some form of locking or mutual exclusion, that it results in a performance or scalability issue.

Although “hot” typically indicates an undesirable condition, a **hot backup** is the preferred type of backup.
See Also hot backup.

hot backup
A backup taken while the database is running and applications are reading and writing to it. The backup involves more than simply copying data files: it must include any data that was inserted or updated while the backup was in process; it must exclude any data that was deleted while the backup was in process; and it must ignore any changes that were not committed.

The Oracle product that performs hot backups, of `InnoDB` tables especially but also tables from `MyISAM` and other storage engines, is known as **MySQL Enterprise Backup**.

The hot backup process consists of two stages. The initial copying of the data files produces a **raw backup**. The **apply** step incorporates any changes to the database that happened while the backup was running. Applying the changes produces a **prepared** backup; these files are ready to be restored whenever necessary.
See Also apply, MySQL Enterprise Backup, prepared backup, raw backup.

I

.ibd file
The data file for **file-per-table** tablespaces and general tablespaces. File-per-table tablespace `.ibd` files contain a single table and associated index data. **General tablespace** `.ibd` files may contain table and index data for multiple tables.

The `.ibd` file extension does not apply to the **system tablespace**, which consists of one or more **ibdata files**.

If a file-per-table tablespace or general tablespace is created with the `DATA DIRECTORY =` clause, the `.ibd` file is located at the specified path, outside the normal data directory.

When a `.ibd` file is included in a compressed backup by the **MySQL Enterprise Backup** product, the compressed equivalent is a `.ibz` file.
See Also database, file-per-table, general tablespace, ibdata file, .ibz file, innodb_file_per_table, MySQL Enterprise Backup, system tablespace.

.ibz file
When the **MySQL Enterprise Backup** product performs a **compressed backup**, it transforms each **tablespace** file that is created using the **file-per-table** setting from a `.ibd` extension to a `.ibz` extension.

The compression applied during backup is distinct from the **compressed row format** that keeps table data compressed during normal operation. A compressed backup operation skips the compression step for a

tablespace that is already in compressed row format, as compressing a second time would slow down the backup but produce little or no space savings.
See Also compressed backup, compressed row format, file-per-table, .ibd file, MySQL Enterprise Backup, tablespace.

I/O-bound
See disk-bound.

ib-file set
The set of files managed by InnoDB within a MySQL database: the **system tablespace**, **file-per-table** tablespace files, and **redo log** files. Depending on MySQL version and InnoDB configuration, may also include **general tablespace**, **temporary tablespace**, and **undo tablespace** files. This term is sometimes used in detailed discussions of InnoDB file structures and formats to refer to the set of files managed by InnoDB within a MySQL database.
See Also database, file-per-table, general tablespace, redo log, system tablespace, temporary tablespace, undo tablespace.

ibbackup_logfile
A supplemental backup file created by the **MySQL Enterprise Backup** product during a **hot backup** operation. It contains information about any data changes that occurred while the backup was running. The initial backup files, including ibbackup_logfile, are known as a **raw backup**, because the changes that occurred during the backup operation are not yet incorporated. After you perform the **apply** step to the raw backup files, the resulting files do include those final data changes, and are known as a **prepared backup**. At this stage, the ibbackup_logfile file is no longer necessary.
See Also apply, hot backup, MySQL Enterprise Backup, prepared backup, raw backup.

ibdata file
A set of files with names such as ibdata1, ibdata2, and so on, that make up the InnoDB **system tablespace**. For information about the structures and data that reside in the system tablespace ibdata files, see Section 15.6.3.1, “The System Tablespace”.

Growth of the ibdata files is influenced by the innodb_autoextend_increment configuration option.
See Also change buffer, data dictionary, doublewrite buffer, file-per-table, .ibd file, innodb_file_per_table, system tablespace, undo log.

ibtmp file
The InnoDB **temporary tablespace data file** for non-compressed InnoDB **temporary tables** and related objects. The configuration file option, innodb_temp_data_file_path, allows users to define a relative path for the temporary tablespace data file. If innodb_temp_data_file_path is not specified, the default behavior is to create a single auto-extending 12MB data file named ibtmp1 in the data directory, alongside ibdata1.
See Also data files, temporary table, temporary tablespace.

ib_logfile
A set of files, typically named ib_logfile0 and ib_logfile1, that form the **redo log**. Also sometimes referred to as the **log group**. These files record statements that attempt to change data in InnoDB tables. These statements are replayed automatically to correct data written by incomplete transactions, on startup following a crash.

This data cannot be used for manual recovery; for that type of operation, use the **binary log**.
See Also binary log, log group, redo log.

ilist
Within an InnoDB **FULLTEXT index**, the data structure consisting of a document ID and positional information for a token (that is, a particular word).
See Also FULLTEXT index.

implicit row lock
A row lock that InnoDB acquires to ensure consistency, without you specifically requesting it.

See Also row lock.

in-memory database
A type of database system that maintains data in memory, to avoid overhead due to disk I/O and translation between disk blocks and memory areas. Some in-memory databases sacrifice durability (the “D” in the **ACID** design philosophy) and are vulnerable to hardware, power, and other types of failures, making them more suitable for read-only operations. Other in-memory databases do use durability mechanisms such as logging changes to disk or using non-volatile memory.

MySQL features that address the same kinds of memory-intensive processing include the `InnoDB` **buffer pool**, **adaptive hash index**, and **read-only transaction** optimization, the `MEMORY` storage engine, the `MyISAM` key cache, and the MySQL query cache.
See Also ACID, adaptive hash index, buffer pool, disk-based, read-only transaction.

incremental backup
A type of **hot backup**, performed by the **MySQL Enterprise Backup** product, that only saves data changed since some point in time. Having a full backup and a succession of incremental backups lets you reconstruct backup data over a long period, without the storage overhead of keeping several full backups on hand. You can restore the full backup and then apply each of the incremental backups in succession, or you can keep the full backup up-to-date by applying each incremental backup to it, then perform a single restore operation.

The granularity of changed data is at the **page** level. A page might actually cover more than one row. Each changed page is included in the backup.
See Also hot backup, MySQL Enterprise Backup, page.

index
A data structure that provides a fast lookup capability for **rows** of a **table**, typically by forming a tree structure (**B-tree)** representing all the values of a particular **column** or set of columns.

`InnoDB` tables always have a **clustered index** representing the **primary key**. They can also have one or more **secondary indexes** defined on one or more columns. Depending on their structure, secondary indexes can be classified as **partial**, **column**, or **composite** indexes.

Indexes are a crucial aspect of **query** performance. Database architects design tables, queries, and indexes to allow fast lookups for data needed by applications. The ideal database design uses a **covering index** where practical; the query results are computed entirely from the index, without reading the actual table data. Each **foreign key** constraint also requires an index, to efficiently check whether values exist in both the **parent** and **child** tables.

Although a B-tree index is the most common, a different kind of data structure is used for **hash indexes**, as in the `MEMORY` storage engine and the `InnoDB` **adaptive hash index**. **R-tree** indexes are used for spatial indexing of multi-dimensional information.
See Also adaptive hash index, B-tree, child table, clustered index, column index, composite index, covering index, foreign key, hash index, parent table, partial index, primary key, query, R-tree, row, secondary index, table.

index cache
A memory area that holds the token data for `InnoDB` **full-text search**. It buffers the data to minimize disk I/O when data is inserted or updated in columns that are part of a **FULLTEXT index**. The token data is written to disk when the index cache becomes full. Each `InnoDB FULLTEXT` index has its own separate index cache, whose size is controlled by the configuration option `innodb_ft_cache_size`.
See Also full-text search, FULLTEXT index.

index condition pushdown
Index condition pushdown (ICP) is an optimization that pushes part of a `WHERE` condition down to the storage engine if parts of the condition can be evaluated using fields from the **index**. ICP can reduce the number of times the **storage engine** must access the base table and the number of times the MySQL server must access the storage engine. For more information, see Section 8.2.1.6, “Index Condition Pushdown Optimization”.
See Also index, storage engine.

index hint
Extended SQL syntax for overriding the **indexes** recommended by the optimizer. For example, the `FORCE INDEX`, `USE INDEX`, and `IGNORE INDEX` clauses. Typically used when indexed columns have unevenly distributed values, resulting in inaccurate **cardinality** estimates.
See Also cardinality, index.

index prefix
In an **index** that applies to multiple columns (known as a **composite index**), the initial or leading columns of the index. A query that references the first 1, 2, 3, and so on columns of a composite index can use the index, even if the query does not reference all the columns in the index.
See Also composite index, index.

index statistics
See statistics.

infimum record
A **pseudo-record** in an **index**, representing the **gap** below the smallest value in that index. If a transaction has a statement such as `SELECT ... FROM ... WHERE col < 10 FOR UPDATE;`, and the smallest value in the column is 5, it is a lock on the infimum record that prevents other transactions from inserting even smaller values such as 0, -10, and so on.
See Also gap, index, pseudo-record, supremum record.

INFORMATION_SCHEMA
The name of the **database** that provides a query interface to the MySQL **data dictionary**. (This name is defined by the ANSI SQL standard.) To examine information (metadata) about the database, you can query tables such as `INFORMATION_SCHEMA.TABLES` and `INFORMATION_SCHEMA.COLUMNS`, rather than using `SHOW` commands that produce unstructured output.

The `INFORMATION_SCHEMA` database also contains tables specific to **InnoDB** that provide a query interface to the `InnoDB` data dictionary. You use these tables not to see how the database is structured, but to get real-time information about the workings of `InnoDB` tables to help with performance monitoring, tuning, and troubleshooting.
See Also data dictionary, database, InnoDB.

InnoDB
A MySQL component that combines high performance with **transactional** capability for reliability, robustness, and concurrent access. It embodies the **ACID** design philosophy. Represented as a **storage engine**; it handles tables created or altered with the `ENGINE=INNODB` clause. See Chapter 15, *The InnoDB Storage Engine* for architectural details and administration procedures, and Section 8.5, “Optimizing for InnoDB Tables” for performance advice.

In MySQL 5.5 and higher, `InnoDB` is the default storage engine for new tables and the `ENGINE=INNODB` clause is not required.

`InnoDB` tables are ideally suited for **hot backups**. See Section 30.2, “MySQL Enterprise Backup Overview” for information about the **MySQL Enterprise Backup** product for backing up MySQL servers without interrupting normal processing.
See Also ACID, hot backup, MySQL Enterprise Backup, storage engine, transaction.

innodb_autoinc_lock_mode
The `innodb_autoinc_lock_mode` option controls the algorithm used for **auto-increment locking**. When you have an auto-incrementing **primary key**, you can use statement-based replication only with the setting `innodb_autoinc_lock_mode=1`. This setting is known as *consecutive* lock mode, because multi-row inserts within a transaction receive consecutive auto-increment values. If you have `innodb_autoinc_lock_mode=2`, which allows higher concurrency for insert operations, use row-based replication rather than statement-based replication. This setting is known as *interleaved* lock mode, because multiple multi-row insert statements running at the same time can receive **auto-increment** values that are interleaved. The setting `innodb_autoinc_lock_mode=0` should not be used except for compatibility purposes.

Consecutive lock mode (`innodb_autoinc_lock_mode=1`) is the default setting prior to MySQL 8.0.3. As of MySQL 8.0.3, interleaved lock mode (`innodb_autoinc_lock_mode=2`) is the default, which reflects the change from statement-based to row-based replication as the default replication type.
See Also auto-increment, auto-increment locking, mixed-mode insert, primary key.

innodb_file_per_table
An important configuration option that affects many aspects of `InnoDB` file storage, availability of features, and I/O characteristics. In MySQL 5.6.7 and higher, it is enabled by default. The `innodb_file_per_table` option turns on **file-per-table** mode. With this mode enabled, a newly created `InnoDB` table and associated indexes can be stored in a file-per-table **.ibd file**, outside the **system tablespace**.

This option affects the performance and storage considerations for a number of SQL statements, such as `DROP TABLE` and `TRUNCATE TABLE`.

Enabling the `innodb_file_per_table` option allows you to take advantage of features such as table **compression** and named-table backups in **MySQL Enterprise Backup**.

For more information, see `innodb_file_per_table`, and Section 15.6.3.2, “File-Per-Table Tablespaces”.
See Also compression, file-per-table, .ibd file, MySQL Enterprise Backup, system tablespace.

innodb_lock_wait_timeout
The `innodb_lock_wait_timeout` option sets the balance between **waiting** for shared resources to become available, or giving up and handling the error, retrying, or doing alternative processing in your application. Rolls back any `InnoDB` transaction that waits more than a specified time to acquire a **lock**. Especially useful if **deadlocks** are caused by updates to multiple tables controlled by different storage engines; such deadlocks are not **detected** automatically.
See Also deadlock, deadlock detection, lock, wait.

innodb_strict_mode
The `innodb_strict_mode` option controls whether `InnoDB` operates in **strict mode**, where conditions that are normally treated as warnings, cause errors instead (and the underlying statements fail).
See Also strict mode.

insert
One of the primary **DML** operations in **SQL**. The performance of inserts is a key factor in **data warehouse** systems that load millions of rows into tables, and **OLTP** systems where many concurrent connections might insert rows into the same table, in arbitrary order. If insert performance is important to you, you should learn about **InnoDB** features such as the **insert buffer** used in **change buffering**, and **auto-increment** columns.
See Also auto-increment, change buffering, data warehouse, DML, InnoDB, insert buffer, OLTP, SQL.

insert buffer
The former name of the **change buffer**. In MySQL 5.5, support was added for buffering changes to secondary index pages for `DELETE` and `UPDATE` operations. Previously, only changes resulting from `INSERT` operations were buffered. The preferred term is now *change buffer*.
See Also change buffer, change buffering.

insert buffering
The technique of storing changes to secondary index pages, resulting from `INSERT` operations, in the **change buffer** rather than writing the changes immediately, so that the physical writes can be performed to minimize random I/O. It is one of the types of **change buffering**; the others are **delete buffering** and **purge buffering**.

Insert buffering is not used if the secondary index is **unique**, because the uniqueness of new values cannot be verified before the new entries are written out. Other kinds of change buffering do work for unique indexes.
See Also change buffer, change buffering, delete buffering, insert buffer, purge buffering, unique index.

insert intention lock
A type of **gap lock** that is set by `INSERT` operations prior to row insertion. This type of **lock** signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each

other if they are not inserting at the same position within the gap. For more information, see Section 15.7.1, “InnoDB Locking”.
See Also gap lock, lock, next-key lock.

instance
A single **mysqld** daemon managing a **data directory** representing one or more **databases** with a set of **tables**. It is common in development, testing, and some **replication** scenarios to have multiple instances on the same **server** machine, each managing its own data directory and listening on its own port or socket. With one instance running a **disk-bound** workload, the server might still have extra CPU and memory capacity to run additional instances.
See Also data directory, database, disk-bound, mysqld, replication, server, table.

instrumentation
Modifications at the source code level to collect performance data for tuning and debugging. In MySQL, data collected by instrumentation is exposed through an SQL interface using the `INFORMATION_SCHEMA` and `PERFORMANCE_SCHEMA` databases.
See Also INFORMATION_SCHEMA, Performance Schema.

intention exclusive lock
See intention lock.

intention lock
A kind of **lock** that applies to the table, used to indicate the kind of lock the **transaction** intends to acquire on rows in the table. Different transactions can acquire different kinds of intention locks on the same table, but the first transaction to acquire an *intention exclusive* (IX) lock on a table prevents other transactions from acquiring any S or X locks on the table. Conversely, the first transaction to acquire an *intention shared* (IS) lock on a table prevents other transactions from acquiring any X locks on the table. The two-phase process allows the lock requests to be resolved in order, without blocking locks and corresponding operations that are compatible. For more information about this locking mechanism, see Section 15.7.1, “InnoDB Locking”.
See Also lock, lock mode, locking, transaction.

intention shared lock
See intention lock.

interceptor
Code for instrumenting or debugging some aspect of an application, which can be enabled without recompiling or changing the source of the application itself.
See Also command interceptor, Connector/J, Connector/NET, exception interceptor.

intrinsic temporary table
An optimized internal `InnoDB` temporary table used by the *optimizer*.
See Also optimizer.

inverted index
A data structure optimized for document retrieval systems, used in the implementation of `InnoDB` **full-text search**. The `InnoDB` **FULLTEXT index**, implemented as an inverted index, records the position of each word within a document, rather than the location of a table row. A single column value (a document stored as a text string) is represented by many entries in the inverted index.
See Also full-text search, FULLTEXT index, ilist.

IOPS
Acronym for **I/O operations per second**. A common measurement for busy systems, particularly **OLTP** applications. If this value is near the maximum that the storage devices can handle, the application can become **disk-bound**, limiting **scalability**.
See Also disk-bound, OLTP, scalability.

isolation level
One of the foundations of database processing. Isolation is the **I** in the acronym **ACID**; the isolation level is the setting that fine-tunes the balance between performance and reliability, consistency, and reproducibility of results when multiple **transactions** are making changes and performing queries at the same time.

From highest amount of consistency and protection to the least, the isolation levels supported by InnoDB are: **SERIALIZABLE**, **REPEATABLE READ**, **READ COMMITTED**, and **READ UNCOMMITTED**.

With `InnoDB` tables, many users can keep the default isolation level (*REPEATABLE READ*) for all operations. Expert users might choose the **READ COMMITTED** level as they push the boundaries of scalability with **OLTP** processing, or during data warehousing operations where minor inconsistencies do not affect the aggregate results of large amounts of data. The levels on the edges (**SERIALIZABLE** and **READ UNCOMMITTED**) change the processing behavior to such an extent that they are rarely used.
See Also ACID, OLTP, READ COMMITTED, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction.

J

J2EE
: Java Platform, Enterprise Edition: Oracle's enterprise Java platform. It consists of an API and a runtime environment for enterprise-class Java applications. For full details, see http://www.oracle.com/technetwork/java/javaee/overview/index.html. With MySQL applications, you typically use **Connector/J** for database access, and an application server such as **Tomcat** or **JBoss** to handle the middle-tier work, and optionally a framework such as **Spring**. Database-related features often offered within a J2EE stack include a **connection pool** and **failover** support.
See Also connection pool, Connector/J, failover, Java, JBoss, Spring, Tomcat.

Java
: A programming language combining high performance, rich built-in features and data types, object-oriented mechanisms, extensive standard library, and wide range of reusable third-party modules. Enterprise development is supported by many frameworks, application servers, and other technologies. Much of its syntax is familiar to **C** and **C++** developers. To write Java applications with MySQL, you use the **JDBC** driver known as **Connector/J**.
See Also C, Connector/J, C++, JDBC.

JBoss
: See Also J2EE.

JDBC
: Abbreviation for “Java Database Connectivity”, an **API** for database access from **Java** applications. Java developers writing MySQL applications use the **Connector/J** component as their JDBC driver.
See Also API, Connector/J, J2EE, Java.

JNDI
: See Also Java.

join
: A **query** that retrieves data from more than one table, by referencing columns in the tables that hold identical values. Ideally, these columns are part of an `InnoDB` **foreign key** relationship, which ensures **referential integrity** and that the join columns are **indexed**. Often used to save space and improve query performance by replacing repeated strings with numeric IDs, in a **normalized** data design.
See Also foreign key, index, normalized, query, referential integrity.

K

KDC
: See key distribution center.

key distribution center
: In Kerberos, the key distribution center comprises an authentication server (AS) and a ticket-granting server (TGS).

See Also authentication server, ticket-granting ticket.

keystore

See Also SSL.

KEY_BLOCK_SIZE
An option to specify the size of data pages within an `InnoDB` table that uses **compressed row format**. The default is 8 kilobytes. Lower values risk hitting internal limits that depend on the combination of row size and compression percentage.

For `MyISAM` tables, `KEY_BLOCK_SIZE` optionally specifies the size in bytes to use for index key blocks. The value is treated as a hint; a different size could be used if necessary. A `KEY_BLOCK_SIZE` value specified for an individual index definition overrides a table-level `KEY_BLOCK_SIZE` value.
See Also compressed row format.

L

latch
A lightweight structure used by `InnoDB` to implement a **lock** for its own internal memory structures, typically held for a brief time measured in milliseconds or microseconds. A general term that includes both **mutexes** (for exclusive access) and **rw-locks** (for shared access). Certain latches are the focus of `InnoDB` performance tuning. Statistics about latch use and contention are available through the **Performance Schema** interface.
See Also lock, locking, mutex, Performance Schema, rw-lock.

libmysql
Informal name for the **libmysqlclient** library.
See Also libmysqlclient.

libmysqlclient
The library file, named `libmysqlclient.a` or `libmysqlclient.so`, that is typically linked into **client** programs written in **C**. Sometimes known informally as **libmysql** or the **mysqlclient** library.
See Also client, libmysql, mysqlclient.

libmysqld
This **embedded** MySQL server library makes it possible to run a full-featured MySQL server inside a **client** application. The main benefits are increased speed and more simple management for embedded applications. You link with the `libmysqld` library rather than **libmysqlclient**. The API is identical between all three of these libraries.
See Also client, embedded, libmysql, libmysqlclient.

lifecycle interceptor
A type of **interceptor** supported by **Connector/J**. It involves implementing the interface `com.mysql.jdbc.ConnectionLifecycleInterceptor`.
See Also Connector/J, interceptor.

list
The `InnoDB` **buffer pool** is represented as a list of memory **pages**. The list is reordered as new pages are accessed and enter the buffer pool, as pages within the buffer pool are accessed again and are considered newer, and as pages that are not accessed for a long time are **evicted** from the buffer pool. The buffer pool is divided into **sublists**, and the replacement policy is a variation of the familiar **LRU** technique.
See Also buffer pool, eviction, LRU, page, sublist.

load balancing
A technique for scaling read-only connections by sending query requests to different slave servers in a replication or Cluster configuration. With **Connector/J**, load balancing is enabled through the `com.mysql.jdbc.ReplicationDriver` class and controlled by the configuration property `loadBalanceStrategy`.
See Also Connector/J, J2EE.

localhost

See Also connection.

lock
The high-level notion of an object that controls access to a resource, such as a table, row, or internal data structure, as part of a **locking** strategy. For intensive performance tuning, you might delve into the actual structures that implement locks, such as **mutexes** and **latches**.
See Also latch, lock mode, locking, mutex.

lock escalation
An operation used in some database systems that converts many **row locks** into a single **table lock**, saving memory space but reducing concurrent access to the table. `InnoDB` uses a space-efficient representation for row locks, so that **lock** escalation is not needed.
See Also locking, row lock, table lock.

lock mode
A shared (S) **lock** allows a **transaction** to read a row. Multiple transactions can acquire an S lock on that same row at the same time.

An exclusive (X) lock allows a transaction to update or delete a row. No other transaction can acquire any kind of lock on that same row at the same time.

Intention locks apply to the table, and are used to indicate what kind of lock the transaction intends to acquire on rows in the table. Different transactions can acquire different kinds of intention locks on the same table, but the first transaction to acquire an intention exclusive (IX) lock on a table prevents other transactions from acquiring any S or X locks on the table. Conversely, the first transaction to acquire an intention shared (IS) lock on a table prevents other transactions from acquiring any X locks on the table. The two-phase process allows the lock requests to be resolved in order, without blocking locks and corresponding operations that are compatible.
See Also intention lock, lock, locking, transaction.

locking
The system of protecting a **transaction** from seeing or changing data that is being queried or changed by other transactions. The **locking** strategy must balance reliability and consistency of database operations (the principles of the **ACID** philosophy) against the performance needed for good **concurrency**. Fine-tuning the locking strategy often involves choosing an **isolation level** and ensuring all your database operations are safe and reliable for that isolation level.
See Also ACID, concurrency, isolation level, locking, transaction.

locking read
A `SELECT` statement that also performs a **locking** operation on an `InnoDB` table. Either `SELECT ... FOR UPDATE` or `SELECT ... LOCK IN SHARE MODE`. It has the potential to produce a **deadlock**, depending on the **isolation level** of the transaction. The opposite of a **non-locking read**. Not allowed for global tables in a **read-only transaction**.

`SELECT ... FOR SHARE` replaces `SELECT ... LOCK IN SHARE MODE` in MySQL 8.0.1, but `LOCK IN SHARE MODE` remains available for backward compatibility.

See Section 15.7.2.4, “Locking Reads”.
See Also deadlock, isolation level, locking, non-locking read, read-only transaction.

log
In the `InnoDB` context, “log” or “log files” typically refers to the **redo log** represented by the **ib_logfile*N*** files. Another type of `InnoDB` log is the **undo log**, which is a storage area that holds copies of data modified by active transactions.

Other kinds of logs that are important in MySQL are the **error log** (for diagnosing startup and runtime problems), **binary log** (for working with replication and performing point-in-time restores), the **general query log** (for diagnosing application problems), and the **slow query log** (for diagnosing performance problems).

See Also binary log, error log, general query log, ib_logfile, redo log, slow query log, undo log.

log buffer
The memory area that holds data to be written to the **log files** that make up the **redo log**. It is controlled by the `innodb_log_buffer_size` configuration option.
See Also log file, redo log.

log file
One of the **ib_logfile*N*** files that make up the **redo log**. Data is written to these files from the **log buffer** memory area.
See Also ib_logfile, log buffer, redo log.

log group
The set of files that make up the **redo log**, typically named `ib_logfile0` and `ib_logfile1`. (For that reason, sometimes referred to collectively as **ib_logfile**.)
See Also ib_logfile, redo log.

logical
A type of operation that involves high-level, abstract aspects such as tables, queries, indexes, and other SQL concepts. Typically, logical aspects are important to make database administration and application development convenient and usable. Contrast with **physical**.
See Also logical backup, physical.

logical backup
A **backup** that reproduces table structure and data, without copying the actual data files. For example, the **`mysqldump`** command produces a logical backup, because its output contains statements such as `CREATE TABLE` and `INSERT` that can re-create the data. Contrast with **physical backup**. A logical backup offers flexibility (for example, you could edit table definitions or insert statements before restoring), but can take substantially longer to **restore** than a physical backup.
See Also backup, mysqldump, physical backup, restore.

loose_
A prefix added to `InnoDB` configuration options after server **startup**, so any new configuration options not recognized by the current level of MySQL do not cause a startup failure. MySQL processes configuration options that start with this prefix, but gives a warning rather than a failure if the part after the prefix is not a recognized option.
See Also startup.

low-water mark
A value representing a lower limit, typically a threshold value at which some corrective action begins or becomes more aggressive. Contrast with **high-water mark**.
See Also high-water mark.

LRU
An acronym for “least recently used”, a common method for managing storage areas. The items that have not been used recently are **evicted** when space is needed to cache newer items. `InnoDB` uses the LRU mechanism by default to manage the **pages** within the **buffer pool**, but makes exceptions in cases where a page might be read only a single time, such as during a **full table scan**. This variation of the LRU algorithm is called the **midpoint insertion strategy**. For more information, see Section 15.5.1, “Buffer Pool”.
See Also buffer pool, eviction, full table scan, midpoint insertion strategy, page.

LSN
Acronym for “log sequence number”. This arbitrary, ever-increasing value represents a point in time corresponding to operations recorded in the **redo log**. (This point in time is regardless of **transaction** boundaries; it can fall in the middle of one or more transactions.) It is used internally by `InnoDB` during **crash recovery** and for managing the **buffer pool**.

Prior to MySQL 5.6.3, the LSN was a 4-byte unsigned integer. The LSN became an 8-byte unsigned integer in MySQL 5.6.3 when the redo log file size limit increased from 4GB to 512GB, as additional bytes were required

to store extra size information. Applications built on MySQL 5.6.3 or later that use LSN values should use 64-bit rather than 32-bit variables to store and compare LSN values.

In the **MySQL Enterprise Backup** product, you can specify an LSN to represent the point in time from which to take an **incremental backup**. The relevant LSN is displayed by the output of the `mysqlbackup` command. Once you have the LSN corresponding to the time of a full backup, you can specify that value to take a subsequent incremental backup, whose output contains another LSN for the next incremental backup.
See Also buffer pool, crash recovery, incremental backup, MySQL Enterprise Backup, redo log, transaction.

M

.MRG file
A file containing references to other tables, used by the `MERGE` storage engine. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup, mysqlbackup command.

.MYD file
A file that MySQL uses to store data for a `MyISAM` table.
See Also .MYI file, MySQL Enterprise Backup, mysqlbackup command.

.MYI file
A file that MySQL uses to store indexes for a `MyISAM` table.
See Also .MYD file, MySQL Enterprise Backup, mysqlbackup command.

master
See source.

master thread
An `InnoDB` **thread** that performs various tasks in the background. Most of these tasks are I/O related, such as writing changes from the **change buffer** to the appropriate secondary indexes.

To improve **concurrency**, sometimes actions are moved from the master thread to separate background threads. For example, in MySQL 5.6 and higher, **dirty pages** are **flushed** from the **buffer pool** by the **page cleaner** thread rather than the master thread.
See Also buffer pool, change buffer, concurrency, dirty page, flush, page cleaner, thread.

MDL
Acronym for “metadata lock”.
See Also metadata lock.

medium trust
Synonym for **partial trust**. Because the range of trust settings is so broad, “partial trust” is preferred, to avoid the implication that there are only three levels (low, medium, and full).
See Also Connector/NET, partial trust.

memcached
A popular component of many MySQL and **NoSQL** software stacks, allowing fast reads and writes for single values and caching the results entirely in memory. Traditionally, applications required extra logic to write the same data to a MySQL database for permanent storage, or to read data from a MySQL database when it was not cached yet in memory. Now, applications can use the simple `memcached` protocol, supported by client libraries for many languages, to communicate directly with MySQL servers using `InnoDB` or `NDB` tables. These NoSQL interfaces to MySQL tables allow applications to achieve higher read and write performance than by issuing SQL statements directly, and can simplify application logic and deployment configurations for systems that already incorporate `memcached` for in-memory caching.

The `memcached` interface to `InnoDB` tables is available in MySQL 5.6 and higher; see Section 15.20, “InnoDB memcached Plugin” for details. The `memcached` interface to `NDB` tables is available in NDB Cluster 7.2 and later; see http://dev.mysql.com/doc/ndbapi/en/ndbmemcache.html for details.

See Also InnoDB, NoSQL.

merge
To apply changes to data cached in memory, such as when a page is brought into the **buffer pool**, and any applicable changes recorded in the **change buffer** are incorporated into the page in the buffer pool. The updated data is eventually written to the **tablespace** by the **flush** mechanism.
See Also buffer pool, change buffer, flush, tablespace.

metadata lock
A type of **lock** that prevents **DDL** operations on a table that is being used at the same time by another **transaction**. For details, see Section 8.11.4, “Metadata Locking”.

Enhancements to **online** operations, particularly in MySQL 5.6 and higher, are focused on reducing the amount of metadata locking. The objective is for DDL operations that do not change the table structure (such as `CREATE INDEX` and `DROP INDEX` for `InnoDB` tables) to proceed while the table is being queried, updated, and so on by other transactions.
See Also DDL, lock, online, transaction.

metrics counter
A feature implemented by the `INNODB_METRICS` table in the **INFORMATION_SCHEMA**, in MySQL 5.6 and higher. You can query **counts** and totals for low-level `InnoDB` operations, and use the results for performance tuning in combination with data from the **Performance Schema**.
See Also counter, INFORMATION_SCHEMA, Performance Schema.

midpoint insertion strategy
The technique of initially bringing **pages** into the `InnoDB` **buffer pool** not at the “newest” end of the list, but instead somewhere in the middle. The exact location of this point can vary, based on the setting of the `innodb_old_blocks_pct` option. The intent is that pages that are only read once, such as during a **full table scan**, can be aged out of the buffer pool sooner than with a strict **LRU** algorithm. For more information, see Section 15.5.1, “Buffer Pool”.
See Also buffer pool, full table scan, LRU, page.

mini-transaction
An internal phase of `InnoDB` processing, when making changes at the **physical** level to internal data structures during **DML** operations. A mini-transaction (mtr) has no notion of **rollback**; multiple mini-transactions can occur within a single **transaction**. Mini-transactions write information to the **redo log** that is used during **crash recovery**. A mini-transaction can also happen outside the context of a regular transaction, for example during **purge** processing by background threads.
See Also commit, crash recovery, DML, physical, purge, redo log, rollback, transaction.

mixed-mode insert
An `INSERT` statement where **auto-increment** values are specified for some but not all of the new rows. For example, a multi-value `INSERT` could specify a value for the auto-increment column in some cases and `NULL` in other cases. `InnoDB` generates auto-increment values for the rows where the column value was specified as `NULL`. Another example is an `INSERT ... ON DUPLICATE KEY UPDATE` statement, where auto-increment values might be generated but not used, for any duplicate rows that are processed as `UPDATE` rather than `INSERT` statements.

Can cause consistency issues between **source** and **replica** servers in a **replication** configuration. Can require adjusting the value of the **innodb_autoinc_lock_mode** configuration option.
See Also auto-increment, innodb_autoinc_lock_mode, replica, replication, source.

MM.MySQL
An older JDBC driver for MySQL that evolved into **Connector/J** when it was integrated with the MySQL product.
See Also Connector/J.

Mono
An Open Source framework developed by Novell, that works with **Connector/NET** and **C#** applications on Linux platforms.

See Also Connector/NET, C#.

mtr
See mini-transaction.

multi-core
A type of processor that can take advantage of multithreaded programs, such as the MySQL server.

multiversion concurrency control
See MVCC.

mutex
Informal abbreviation for “mutex variable”. (Mutex itself is short for “mutual exclusion”.) The low-level object that `InnoDB` uses to represent and enforce exclusive-access **locks** to internal in-memory data structures. Once the lock is acquired, any other process, thread, and so on is prevented from acquiring the same lock. Contrast with **rw-locks**, which `InnoDB` uses to represent and enforce shared-access **locks** to internal in-memory data structures. Mutexes and rw-locks are known collectively as **latches**.
See Also latch, lock, Performance Schema, Pthreads, rw-lock.

MVCC
Acronym for “multiversion concurrency control”. This technique lets `InnoDB` **transactions** with certain **isolation levels** perform **consistent read** operations; that is, to query rows that are being updated by other transactions, and see the values from before those updates occurred. This is a powerful technique to increase **concurrency**, by allowing queries to proceed without waiting due to **locks** held by the other transactions.

This technique is not universal in the database world. Some other database products, and some other MySQL storage engines, do not support it.
See Also ACID, concurrency, consistent read, isolation level, lock, transaction.

my.cnf
The name, on Unix or Linux systems, of the MySQL **option file**.
See Also my.ini, option file.

my.ini
The name, on Windows systems, of the MySQL **option file**.
See Also my.cnf, option file.

MyODBC drivers
Obsolete name for **Connector/ODBC**.
See Also Connector/ODBC.

mysql
The `mysql` program is the command-line interpreter for the MySQL database. It processes **SQL** statements, and also MySQL-specific commands such as `SHOW TABLES`, by passing requests to the `mysqld` daemon.
See Also mysqld, SQL.

MySQL Enterprise Backup
A licensed product that performs **hot backups** of MySQL databases. It offers the most efficiency and flexibility when backing up `InnoDB` tables, but can also back up `MyISAM` and other kinds of tables.
See Also hot backup, InnoDB.

mysqlbackup command
A command-line tool of the **MySQL Enterprise Backup** product. It performs a **hot backup** operation for `InnoDB` tables, and a warm backup for `MyISAM` and other kinds of tables. See Section 30.2, “MySQL Enterprise Backup Overview” for more information about this command.
See Also hot backup, MySQL Enterprise Backup, warm backup.

mysqlclient
The informal name for the library that is implemented by the file **libmysqlclient**, with extension `.a` or `.so`.

See Also libmysqlclient.

mysqld
: `mysqld`, also known as MySQL Server, is a single multithreaded program that does most of the work in a MySQL installation. It does not spawn additional processes. MySQL Server manages access to the MySQL data directory that contains databases, tables, and other information such as log files and status files.

`mysqld` runs as a Unix daemon or Windows service, constantly waiting for requests and performing maintenance work in the background.
See Also instance, mysql.

MySQLdb
: The name of the open-source **Python** module that forms the basis of the MySQL **Python API**.
See Also Python, Python API.

mysqldump
: A command that performs a **logical backup** of some combination of databases, tables, and table data. The results are SQL statements that reproduce the original schema objects, data, or both. For substantial amounts of data, a **physical backup** solution such as **MySQL Enterprise Backup** is faster, particularly for the **restore** operation.
See Also logical backup, MySQL Enterprise Backup, physical backup, restore.

N

.NET

See Also ADO.NET, ASP.net, Connector/NET, Mono, Visual Studio.

native C API
: Synonym for **libmysqlclient**.
See Also libmysql.

natural key
: An indexed column, typically a **primary key**, where the values have some real-world significance. Usually advised against because:

- If the value should ever change, there is potentially a lot of index maintenance to re-sort the **clustered index** and update the copies of the primary key value that are repeated in each **secondary index**.
- Even seemingly stable values can change in unpredictable ways that are difficult to represent correctly in the database. For example, one country can change into two or several, making the original country code obsolete. Or, rules about unique values might have exceptions. For example, even if taxpayer IDs are intended to be unique to a single person, a database might have to handle records that violate that rule, such as in cases of identity theft. Taxpayer IDs and other sensitive ID numbers also make poor primary keys, because they may need to be secured, encrypted, and otherwise treated differently than other columns.

Thus, it is typically better to use arbitrary numeric values to form a **synthetic key**, for example using an **auto-increment** column.
See Also auto-increment, clustered index, primary key, secondary index, synthetic key.

neighbor page
: Any **page** in the same **extent** as a particular page. When a page is selected to be **flushed**, any neighbor pages that are **dirty** are typically flushed as well, as an I/O optimization for traditional hard disks. In MySQL 5.6 and up, this behavior can be controlled by the configuration variable `innodb_flush_neighbors`; you might turn that setting off for SSD drives, which do not have the same overhead for writing smaller batches of data at random locations.
See Also dirty page, extent, flush, page.

next-key lock

A combination of a **record lock** on the index record and a gap lock on the gap before the index record.
See Also gap lock, locking, record lock.

non-locking read

A **query** that does not use the `SELECT ... FOR UPDATE` or `SELECT ... LOCK IN SHARE MODE` clauses. The only kind of query allowed for global tables in a **read-only transaction**. The opposite of a **locking read**. See Section 15.7.2.3, “Consistent Nonlocking Reads”.

`SELECT ... FOR SHARE` replaces `SELECT ... LOCK IN SHARE MODE` in MySQL 8.0.1, but `LOCK IN SHARE MODE` remains available for backward compatibility.
See Also locking read, query, read-only transaction.

non-repeatable read

The situation when a query retrieves data, and a later query within the same **transaction** retrieves what should be the same data, but the queries return different results (changed by another transaction committing in the meantime).

This kind of operation goes against the **ACID** principle of database design. Within a transaction, data should be consistent, with predictable and stable relationships.

Among different **isolation levels**, non-repeatable reads are prevented by the **serializable read** and **repeatable read** levels, and allowed by the **consistent read**, and **read uncommitted** levels.
See Also ACID, consistent read, isolation level, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction.

nonblocking I/O

An industry term that means the same as **asynchronous I/O**.
See Also asynchronous I/O.

normalized

A database design strategy where data is split into multiple tables, and duplicate values condensed into single rows represented by an ID, to avoid storing, querying, and updating redundant or lengthy values. It is typically used in **OLTP** applications.

For example, an address might be given a unique ID, so that a census database could represent the relationship **lives at this address** by associating that ID with each member of a family, rather than storing multiple copies of a complex value such as **123 Main Street, Anytown, USA**.

For another example, although a simple address book application might store each phone number in the same table as a person's name and address, a phone company database might give each phone number a special ID, and store the numbers and IDs in a separate table. This normalized representation could simplify large-scale updates when area codes split apart.

Normalization is not always recommended. Data that is primarily queried, and only updated by deleting entirely and reloading, is often kept in fewer, larger tables with redundant copies of duplicate values. This data representation is referred to as **denormalized**, and is frequently found in data warehousing applications.
See Also denormalized, foreign key, OLTP, relational.

NoSQL

A broad term for a set of data access technologies that do not use the **SQL** language as their primary mechanism for reading and writing data. Some NoSQL technologies act as key-value stores, only accepting single-value reads and writes; some relax the restrictions of the **ACID** methodology; still others do not require a pre-planned **schema**. MySQL users can combine NoSQL-style processing for speed and simplicity with SQL operations for flexibility and convenience, by using the **memcached** API to directly access some kinds of MySQL tables. The `memcached` interface to `InnoDB` tables is available in MySQL 5.6 and higher; see Section 15.20, “InnoDB memcached Plugin” for details. The `memcached` interface to `NDB` tables is available in NDB Cluster 7.2 and later; see ndbmemcache—Memcache API for NDB Cluster (NO LONGER SUPPORTED).
See Also ACID, InnoDB, memcached, schema, SQL.

NOT NULL constraint
A type of **constraint** that specifies that a **column** cannot contain any **NULL** values. It helps to preserve **referential integrity**, as the database server can identify data with erroneous missing values. It also helps in the arithmetic involved in query optimization, allowing the optimizer to predict the number of entries in an index on that column.
See Also column, constraint, NULL, primary key, referential integrity.

NULL
A special value in **SQL**, indicating the absence of data. Any arithmetic operation or equality test involving a `NULL` value, in turn produces a `NULL` result. (Thus it is similar to the IEEE floating-point concept of NaN, "not a number".) Any aggregate calculation such as `AVG()` ignores rows with `NULL` values, when determining how many rows to divide by. The only test that works with `NULL` values uses the SQL idioms `IS NULL` or `IS NOT NULL`.

`NULL` values play a part in **index** operations, because for performance a database must minimize the overhead of keeping track of missing data values. Typically, `NULL` values are not stored in an index, because a query that tests an indexed column using a standard comparison operator could never match a row with a `NULL` value for that column. For the same reason, unique indexes do not prevent `NULL` values; those values simply are not represented in the index. Declaring a `NOT NULL` constraint on a column provides reassurance that there are no rows left out of the index, allowing for better query optimization (accurate counting of rows and estimation of whether to use the index).

Because the **primary key** must be able to uniquely identify every row in the table, a single-column primary key cannot contain any `NULL` values, and a multi-column primary key cannot contain any rows with `NULL` values in all columns.

Although the Oracle database allows a `NULL` value to be concatenated with a string, `InnoDB` treats the result of such an operation as `NULL`.
See Also index, primary key, SQL.

O

.OPT file
A file containing database configuration information. Files with this extension are included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup, mysqlbackup command.

ODBC
Acronym for Open Database Connectivity, an industry-standard API. Typically used with Windows-based servers, or applications that require ODBC to communicate with MySQL. The MySQL ODBC driver is called **Connector/ODBC**.
See Also Connector/ODBC.

off-page column
A column containing variable-length data (such as `BLOB` and `VARCHAR`) that is too long to fit on a **B-tree** page. The data is stored in **overflow pages**. The **DYNAMIC** row format is more efficient for such storage than the older **COMPACT** row format.
See Also B-tree, compact row format, dynamic row format, overflow page.

OLTP
Acronym for "Online Transaction Processing". A database system, or a database application, that runs a workload with many **transactions**, with frequent writes as well as reads, typically affecting small amounts of data at a time. For example, an airline reservation system or an application that processes bank deposits. The data might be organized in **normalized** form for a balance between **DML** (insert/update/delete) efficiency and **query** efficiency. Contrast with **data warehouse**.

With its **row-level locking** and **transactional** capability, **InnoDB** is the ideal storage engine for MySQL tables used in OLTP applications.
See Also data warehouse, DML, InnoDB, query, row lock, transaction.

online
A type of operation that involves no downtime, blocking, or restricted operation for the database. Typically applied to **DDL**. Operations that shorten the periods of restricted operation, such as **fast index creation**, have evolved into a wider set of **online DDL** operations in MySQL 5.6.

In the context of backups, a **hot backup** is an online operation and a **warm backup** is partially an online operation.
See Also DDL, Fast Index Creation, hot backup, online DDL, warm backup.

online DDL
A feature that improves the performance, concurrency, and availability of `InnoDB` tables during **DDL** (primarily `ALTER TABLE`) operations. See Section 15.12, "InnoDB and Online DDL" for details.

The details vary according to the type of operation. In some cases, the table can be modified concurrently while the `ALTER TABLE` is in progress. The operation might be able to be performed without a table copy, or using a specially optimized type of table copy. DML log space usage for in-place operations is controlled by the `innodb_online_alter_log_max_size` configuration option.

This feature is an enhancement of the **Fast Index Creation** feature in MySQL 5.5.
See Also DDL, Fast Index Creation, online.

optimistic
A methodology that guides low-level implementation decisions for a relational database system. The requirements of performance and **concurrency** in a relational database mean that operations must be started or dispatched quickly. The requirements of consistency and **referential integrity** mean that any operation could fail: a transaction might be rolled back, a **DML** operation could violate a constraint, a request for a lock could cause a deadlock, a network error could cause a timeout. An optimistic strategy is one that assumes most requests or attempts succeed, so that relatively little work is done to prepare for the failure case. When this assumption is true, the database does little unnecessary work; when requests do fail, extra work must be done to clean up and undo changes.

`InnoDB` uses optimistic strategies for operations such as **locking** and **commits**. For example, data changed by a transaction can be written to the data files before the commit occurs, making the commit itself very fast, but requiring more work to undo the changes if the transaction is rolled back.

The opposite of an optimistic strategy is a **pessimistic** one, where a system is optimized to deal with operations that are unreliable and frequently unsuccessful. This methodology is rare in a database system, because so much care goes into choosing reliable hardware, networks, and algorithms.
See Also commit, concurrency, DML, locking, pessimistic, referential integrity.

optimizer
The MySQL component that determines the best **indexes** and **join** order to use for a **query**, based on characteristics and data distribution of the relevant **tables**.
See Also index, join, query, table.

option
A configuration parameter for MySQL, either stored in the **option file** or passed on the command line.

For the **options** that apply to **InnoDB** tables, each option name starts with the prefix `innodb_`.
See Also InnoDB, option, option file.

option file
The file that holds the configuration **options** for the MySQL instance. Traditionally, on Linux and Unix this file is named `my.cnf`, and on Windows it is named `my.ini`.
See Also configuration file, my.cnf, my.ini, option.

overflow page
Separately allocated disk **pages** that hold variable-length columns (such as `BLOB` and `VARCHAR`) that are too long to fit on a **B-tree** page. The associated columns are known as **off-page columns**.
See Also B-tree, off-page column, page.

P

.par file
: A file containing partition definitions. Files with this extension are included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.

  With the introduction of native partitioning support for `InnoDB` tables in MySQL 5.7.6, `.par` files are no longer created for partitioned `InnoDB` tables. Partitioned `MyISAM` tables continue to use `.par` files in MySQL 5.7. In MySQL 8.0, partitioning support is only provided by the `InnoDB` storage engine. As such, `.par` files are no longer used as of MySQL 8.0.
  See Also MySQL Enterprise Backup, mysqlbackup command.

page
: A unit representing how much data `InnoDB` transfers at any one time between disk (the **data files**) and memory (the **buffer pool**). A page can contain one or more **rows**, depending on how much data is in each row. If a row does not fit entirely into a single page, `InnoDB` sets up additional pointer-style data structures so that the information about the row can be stored in one page.

  One way to fit more data in each page is to use **compressed row format**. For tables that use BLOBs or large text fields, **compact row format** allows those large columns to be stored separately from the rest of the row, reducing I/O overhead and memory usage for queries that do not reference those columns.

  When `InnoDB` reads or writes sets of pages as a batch to increase I/O throughput, it reads or writes an **extent** at a time.

  All the `InnoDB` disk data structures within a MySQL instance share the same **page size**.
  See Also buffer pool, compact row format, compressed row format, data files, extent, page size, row.

page cleaner
: An `InnoDB` background **thread** that **flushes dirty pages** from the **buffer pool**. Prior to MySQL 5.6, this activity was performed by the **master thread**. The number of page cleaner threads is controlled by the `innodb_page_cleaners` configuration option, introduced in MySQL 5.7.4.
  See Also buffer pool, dirty page, flush, master thread, thread.

page size
: For releases up to and including MySQL 5.5, the size of each `InnoDB` **page** is fixed at 16 kilobytes. This value represents a balance: large enough to hold the data for most rows, yet small enough to minimize the performance overhead of transferring unneeded data to memory. Other values are not tested or supported.

  Starting in MySQL 5.6, the page size for an `InnoDB` **instance** can be either 4KB, 8KB, or 16KB, controlled by the `innodb_page_size` configuration option. As of MySQL 5.7.6, `InnoDB` also supports 32KB and 64KB page sizes. For 32KB and 64KB page sizes, `ROW_FORMAT=COMPRESSED` is not supported and the maximum record size is 16KB.

  Page size is set when creating the MySQL instance, and it remains constant afterward. The same page size applies to all `InnoDB` **tablespaces**, including the **system tablespace**, **file-per-table** tablespaces, and **general tablespaces**.

  Smaller page sizes can help performance with storage devices that use small block sizes, particularly for **SSD** devices in **disk-bound** workloads, such as for **OLTP** applications. As individual rows are updated, less data is copied into memory, written to disk, reorganized, locked, and so on.
  See Also disk-bound, file-per-table, general tablespace, instance, OLTP, page, SSD, system tablespace, tablespace.

parent table
: The table in a **foreign key** relationship that holds the initial column values pointed to from the **child table**. The consequences of deleting, or updating rows in the parent table depend on the `ON UPDATE` and `ON DELETE` clauses in the foreign key definition. Rows with corresponding values in the child table could be automatically deleted or updated in turn, or those columns could be set to `NULL`, or the operation could be prevented.

See Also child table, foreign key.

partial backup
: A **backup** that contains some of the **tables** in a MySQL database, or some of the databases in a MySQL instance. Contrast with **full backup**.
See Also backup, full backup, table.

partial index
: An **index** that represents only part of a column value, typically the first N characters (the **prefix**) of a long `VARCHAR` value.
See Also index, index prefix.

partial trust
: An execution environment typically used by hosting providers, where applications have some permissions but not others. For example, applications might be able to access a database server over a network, but be "sandboxed" with regard to reading and writing local files.
See Also Connector/NET.

Performance Schema
: The `performance_schema` schema, in MySQL 5.5 and up, presents a set of tables that you can query to get detailed information about the performance characteristics of many internal parts of the MySQL server.
See Chapter 27, *MySQL Performance Schema*.
See Also INFORMATION_SCHEMA, latch, mutex, rw-lock.

Perl
: A programming language with roots in Unix scripting and report generation. Incorporates high-performance regular expressions and file I/O. Large collection of reusable modules available through repositories such as CPAN.
See Also Perl API.

Perl API
: An open-source **API** for MySQL applications written in the **Perl** language. Implemented through the `DBI` and `DBD::mysql` modules. For details, see Section 29.9, "MySQL Perl API".
See Also API, Perl.

persistent statistics
: A feature that stores **index** statistics for `InnoDB` **tables** on disk, providing better **plan stability** for **queries**. For more information, see Section 15.8.10.1, "Configuring Persistent Optimizer Statistics Parameters".
See Also index, optimizer, plan stability, query, table.

pessimistic
: A methodology that sacrifices performance or concurrency in favor of safety. It is appropriate if a high proportion of requests or attempts might fail, or if the consequences of a failed request are severe. `InnoDB` uses what is known as a pessimistic **locking** strategy, to minimize the chance of **deadlocks**. At the application level, you might avoid deadlocks by using a pessimistic strategy of acquiring all locks needed by a transaction at the very beginning.

  Many built-in database mechanisms use the opposite **optimistic** methodology.
  See Also deadlock, locking, optimistic.

phantom
: A row that appears in the result set of a query, but not in the result set of an earlier query. For example, if a query is run twice within a **transaction**, and in the meantime, another transaction commits after inserting a new row or updating a row so that it matches the `WHERE` clause of the query.

  This occurrence is known as a phantom read. It is harder to guard against than a **non-repeatable read**, because locking all the rows from the first query result set does not prevent the changes that cause the phantom to appear.

  Among different **isolation levels**, phantom reads are prevented by the **serializable read** level, and allowed by the **repeatable read**, **consistent read**, and **read uncommitted** levels.

See Also consistent read, isolation level, non-repeatable read, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction.

PHP

A programming language originating with web applications. The code is typically embedded as blocks within the source of a web page, with the output substituted into the page as it is transmitted by the web server. This is in contrast to applications such as CGI scripts that print output in the form of an entire web page. The PHP style of coding is used for highly interactive and dynamic web pages. Modern PHP programs can also be run as command-line or GUI applications.

MySQL applications are written using one of the **PHP APIs**. Reusable modules can be written in **C** and called from PHP.

Another technology for writing server-side web pages with MySQL is **ASP.net**.
See Also ASP.net, C, PHP API.

PHP API

Several **APIs** are available for writing MySQL applications in the **PHP** language: the original MySQL API (`Mysql`) the MySQL Improved Extension (`Mysqli`) the MySQL Native Driver (`Mysqlnd`) the MySQL functions (`PDO_MYSQL`), and Connector/PHP. For details, see MySQL and PHP.
See Also API, PHP.

physical

A type of operation that involves hardware-related aspects such as disk blocks, memory pages, files, bits, disk reads, and so on. Typically, physical aspects are important during expert-level performance tuning and problem diagnosis. Contrast with **logical**.
See Also logical, physical backup.

physical backup

A **backup** that copies the actual data files. For example, the **`mysqlbackup`** command of the **MySQL Enterprise Backup** product produces a physical backup, because its output contains data files that can be used directly by the `mysqld` server, resulting in a faster **restore** operation. Contrast with **logical backup**.
See Also backup, logical backup, MySQL Enterprise Backup, restore.

PITR

Acronym for **point-in-time recovery**.
See Also point-in-time recovery.

plan stability

A property of a **query execution plan**, where the optimizer makes the same choices each time for a given **query**, so that performance is consistent and predictable.
See Also query, query execution plan.

point-in-time recovery

The process of restoring a **backup** to recreate the state of the database at a specific date and time. Commonly abbreviated “PITR”. Because it is unlikely that the specified time corresponds exactly to the time of a backup, this technique usually requires a combination of a **physical backup** and a **logical backup**. For example, with the **MySQL Enterprise Backup** product, you restore the last backup that you took before the specified point in time, then replay changes from the **binary log** between the time of the backup and the PITR time.
See Also backup, binary log, logical backup, MySQL Enterprise Backup, physical backup.

port

The number of the TCP/IP socket the database server listens on, used to establish a **connection**. Often specified in conjunction with a **host**. Depending on your use of network encryption, there might be one port for unencrypted traffic and another port for **SSL** connections.
See Also connection, host, SSL.

prefix

See index prefix.

prepared backup
A set of backup files, produced by the **MySQL Enterprise Backup** product, after all the stages of applying **binary logs** and **incremental backups** are finished. The resulting files are ready to be **restored**. Prior to the apply steps, the files are known as a **raw backup**.
See Also binary log, hot backup, incremental backup, MySQL Enterprise Backup, raw backup, restore.

prepared statement
An SQL statement that is analyzed in advance to determine an efficient execution plan. It can be executed multiple times, without the overhead for parsing and analysis each time. Different values can be substituted for literals in the `WHERE` clause each time, through the use of placeholders. This substitution technique improves security, protecting against some kinds of SQL injection attacks. You can also reduce the overhead for converting and copying return values to program variables.

Although you can use prepared statements directly through SQL syntax, the various **Connectors** have programming interfaces for manipulating prepared statements, and these APIs are more efficient than going through SQL.
See Also client-side prepared statement, connector, server-side prepared statement.

primary key
A set of columns—and by implication, the index based on this set of columns—that can uniquely identify every row in a table. As such, it must be a unique index that does not contain any `NULL` values.

`InnoDB` requires that every table has such an index (also called the **clustered index** or **cluster index**), and organizes the table storage based on the column values of the primary key.

When choosing primary key values, consider using arbitrary values (a **synthetic key**) rather than relying on values derived from some other source (a **natural key**).
See Also clustered index, index, natural key, synthetic key.

principal
The Kerberos term for a named entity, such as a user or server.
See Also service principal name, user principal name.

process
An instance of an executing program. The operating system switches between multiple running processes, allowing for a certain degree of **concurrency**. On most operating systems, processes can contain multiple **threads** of execution that share resources. Context-switching between threads is faster than the equivalent switching between processes.
See Also concurrency, thread.

pseudo-record
An artificial record in an index, used for **locking** key values or ranges that do not currently exist.
See Also infimum record, locking, supremum record.

Pthreads
The POSIX threads standard, which defines an API for threading and locking operations on Unix and Linux systems. On Unix and Linux systems, `InnoDB` uses this implementation for **mutexes**.
See Also mutex.

purge
A type of garbage collection performed by one or more separate background threads (controlled by `innodb_purge_threads`) that runs on a periodic schedule. Purge parses and processes **undo log** pages from the **history list** for the purpose of removing clustered and secondary index records that were marked for deletion (by previous `DELETE` statements) and are no longer required for **MVCC** or **rollback**. Purge frees undo log pages from the history list after processing them.
See Also history list, MVCC, rollback, undo log.

purge buffering
The technique of storing changes to secondary index pages, resulting from `DELETE` operations, in the **change buffer** rather than writing the changes immediately, so that the physical writes can be performed to minimize random I/O. (Because delete operations are a two-step process, this operation buffers the write that

normally purges an index record that was previously marked for deletion.) It is one of the types of **change buffering**; the others are **insert buffering** and **delete buffering**.
See Also change buffer, change buffering, delete buffering, insert buffer, insert buffering.

purge lag
Another name for the `InnoDB` **history list**. Related to the `innodb_max_purge_lag` configuration option.
See Also history list, purge.

purge thread
A **thread** within the `InnoDB` process that is dedicated to performing the periodic **purge** operation. In MySQL 5.6 and higher, multiple purge threads are enabled by the `innodb_purge_threads` configuration option.
See Also purge, thread.

Python
A programming language used in a broad range of fields, from Unix scripting to large-scale applications. Includes runtime typing, built-in high-level data types, object-oriented features, and an extensive standard library. Often used as a “glue” language between components written in other languages. The MySQL **Python API** is the open-source **MySQLdb** module.
See Also MySQLdb, Python API.

Python API

See Also API, Python.

Q

query
In **SQL**, an operation that reads information from one or more **tables**. Depending on the organization of data and the parameters of the query, the lookup might be optimized by consulting an **index**. If multiple tables are involved, the query is known as a **join**.

For historical reasons, sometimes discussions of internal processing for statements use “query” in a broader sense, including other types of MySQL statements such as **DDL** and **DML** statements.
See Also DDL, DML, index, join, SQL, table.

query execution plan
The set of decisions made by the optimizer about how to perform a **query** most efficiently, including which **index** or indexes to use, and the order in which to **join** tables. **Plan stability** involves the same choices being made consistently for a given query.
See Also index, join, plan stability, query.

query log
See general query log.

quiesce
To reduce the amount of database activity, often in preparation for an operation such as an `ALTER TABLE`, a **backup**, or a **shutdown**. Might or might not involve doing as much **flushing** as possible, so that **InnoDB** does not continue doing background I/O.

In MySQL 5.6 and higher, the syntax `FLUSH TABLES ... FOR EXPORT` writes some data to disk for `InnoDB` tables that make it simpler to back up those tables by copying the data files.
See Also backup, flush, InnoDB, shutdown.

R

R-tree
A tree data structure used for spatial indexing of multi-dimensional data such as geographical coordinates, rectangles or polygons.
See Also B-tree.

RAID

Acronym for “Redundant Array of Inexpensive Drives”. Spreading I/O operations across multiple drives enables greater **concurrency** at the hardware level, and improves the efficiency of low-level write operations that otherwise would be performed in sequence.
See Also concurrency.

random dive

A technique for quickly estimating the number of different values in a column (the column's **cardinality**). `InnoDB` samples pages at random from the index and uses that data to estimate the number of different values.
See Also cardinality.

raw backup

The initial set of backup files produced by the **MySQL Enterprise Backup** product, before the changes reflected in the **binary log** and any **incremental backups** are applied. At this stage, the files are not ready to **restore**. After these changes are applied, the files are known as a **prepared backup**.
See Also binary log, hot backup, ibbackup_logfile, incremental backup, MySQL Enterprise Backup, prepared backup, restore.

READ COMMITTED

An **isolation level** that uses a **locking** strategy that relaxes some of the protection between **transactions**, in the interest of performance. Transactions cannot see uncommitted data from other transactions, but they can see data that is committed by another transaction after the current transaction started. Thus, a transaction never sees any bad data, but the data that it does see may depend to some extent on the timing of other transactions.

When a transaction with this isolation level performs `UPDATE ... WHERE` or `DELETE ... WHERE` operations, other transactions might have to wait. The transaction can perform `SELECT ... FOR UPDATE`, and `LOCK IN SHARE MODE` operations without making other transactions wait.

`SELECT ... FOR SHARE` replaces `SELECT ... LOCK IN SHARE MODE` in MySQL 8.0.1, but `LOCK IN SHARE MODE` remains available for backward compatibility.
See Also ACID, isolation level, locking, REPEATABLE READ, SERIALIZABLE, transaction.

read phenomena

Phenomena such as **dirty reads**, **non-repeatable reads**, and **phantom** reads which can occur when a transaction reads data that another transaction has modified.
See Also dirty read, non-repeatable read, phantom.

READ UNCOMMITTED

The **isolation level** that provides the least amount of protection between transactions. Queries employ a **locking** strategy that allows them to proceed in situations where they would normally wait for another transaction. However, this extra performance comes at the cost of less reliable results, including data that has been changed by other transactions and not committed yet (known as **dirty read**). Use this isolation level with great caution, and be aware that the results might not be consistent or reproducible, depending on what other transactions are doing at the same time. Typically, transactions with this isolation level only do queries, not insert, update, or delete operations.
See Also ACID, dirty read, isolation level, locking, transaction.

read view

An internal snapshot used by the **MVCC** mechanism of `InnoDB`. Certain **transactions**, depending on their **isolation level**, see the data values as they were at the time the transaction (or in some cases, the statement) started. Isolation levels that use a read view are **REPEATABLE READ**, **READ COMMITTED**, and **READ UNCOMMITTED**.
See Also isolation level, MVCC, READ COMMITTED, READ UNCOMMITTED, REPEATABLE READ, transaction.

read-ahead

A type of I/O request that prefetches a group of **pages** (an entire **extent**) into the **buffer pool** asynchronously, in case these pages are needed soon. The linear read-ahead technique prefetches all the

pages of one extent based on access patterns for pages in the preceding extent. The random read-ahead technique prefetches all the pages for an extent once a certain number of pages from the same extent are in the buffer pool. Random read-ahead is not part of MySQL 5.5, but is re-introduced in MySQL 5.6 under the control of the `innodb_random_read_ahead` configuration option.
See Also buffer pool, extent, page.

read-only transaction
A type of **transaction** that can be optimized for `InnoDB` tables by eliminating some of the bookkeeping involved with creating a **read view** for each transaction. Can only perform **non-locking read** queries. It can be started explicitly with the syntax `START TRANSACTION READ ONLY`, or automatically under certain conditions. See Section 8.5.3, "Optimizing InnoDB Read-Only Transactions" for details.
See Also non-locking read, read view, transaction.

record lock
A lock on an index record. For example, `SELECT c1 FROM t WHERE c1 = 10 FOR UPDATE;` prevents any other transaction from inserting, updating, or deleting rows where the value of `t.c1` is 10. Contrast with **gap lock** and **next-key lock**.
See Also gap lock, lock, next-key lock.

redo
The data, in units of records, recorded in the **redo log** when DML statements make changes to `InnoDB` tables. It is used during **crash recovery** to correct data written by incomplete **transactions**. The ever-increasing **LSN** value represents the cumulative amount of redo data that has passed through the redo log.
See Also crash recovery, DML, LSN, redo log, transaction.

redo log
A disk-based data structure used during **crash recovery**, to correct data written by incomplete **transactions**. During normal operation, it encodes requests to change `InnoDB` table data, which result from SQL statements or low-level API calls. Modifications that did not finish updating the **data files** before an unexpected **shutdown** are replayed automatically.

The redo log is physically represented on disk as a set of redo log files. Redo log data is encoded in terms of records affected; this data is collectively referred to as **redo**. The passage of data through the redo log is represented by an ever-increasing **LSN** value.

For more information, see Section 15.6.5, "Redo Log"
See Also crash recovery, data files, ib_logfile, log buffer, LSN, redo, shutdown, transaction.

redo log archiving
An `InnoDB` feature that, when enabled, sequentially writes redo log records to an archive file to avoid potential loss of data than can occur when a backup utility fails to keep pace with redo log generation while a backup operation is in progress. For more information, see Redo Log Archiving.
See Also redo log.

redundant row format
The oldest `InnoDB` **row format**. Prior to MySQL 5.0.3, it was the only row format available in `InnoDB`. From MySQL 5.0.3 to MySQL 5.7.8, the default row format is **COMPACT**. As of MySQL 5.7.9, the default row format is defined by the `innodb_default_row_format` configuration option, which has a default setting of **DYNAMIC**. You can still specify the **REDUNDANT** row format for compatibility with older `InnoDB` tables.

For more information, see Section 15.10, "InnoDB Row Formats".
See Also compact row format, dynamic row format, row format.

referential integrity
The technique of maintaining data always in a consistent format, part of the **ACID** philosophy. In particular, data in different tables is kept consistent through the use of **foreign key constraints**, which can prevent changes from happening or automatically propagate those changes to all related tables. Related mechanisms include the **unique constraint**, which prevents duplicate values from being inserted by mistake, and the **NOT NULL constraint**, which prevents blank values from being inserted by mistake.
See Also ACID, FOREIGN KEY constraint, NOT NULL constraint, unique constraint.

relational

An important aspect of modern database systems. The database server encodes and enforces relationships such as one-to-one, one-to-many, many-to-one, and uniqueness. For example, a person might have zero, one, or many phone numbers in an address database; a single phone number might be associated with several family members. In a financial database, a person might be required to have exactly one taxpayer ID, and any taxpayer ID could only be associated with one person.

The database server can use these relationships to prevent bad data from being inserted, and to find efficient ways to look up information. For example, if a value is declared to be unique, the server can stop searching as soon as the first match is found, and it can reject attempts to insert a second copy of the same value.

At the database level, these relationships are expressed through SQL features such as **columns** within a table, unique and `NOT NULL` **constraints**, **foreign keys**, and different kinds of join operations. Complex relationships typically involve data split between more than one table. Often, the data is **normalized**, so that duplicate values in one-to-many relationships are stored only once.

In a mathematical context, the relations within a database are derived from set theory. For example, the `OR` and `AND` operators of a `WHERE` clause represent the notions of union and intersection.
See Also ACID, column, constraint, foreign key, normalized.

relevance

In the **full-text search** feature, a number signifying the similarity between the search string and the data in the **FULLTEXT index**. For example, when you search for a single word, that word is typically more relevant for a row where it occurs several times in the text than a row where it appears only once.
See Also full-text search, FULLTEXT index.

REPEATABLE READ

The default **isolation level** for `InnoDB`. It prevents any rows that are queried from being changed by other **transactions**, thus blocking **non-repeatable reads** but not **phantom** reads. It uses a moderately strict **locking** strategy so that all queries within a transaction see data from the same snapshot, that is, the data as it was at the time the transaction started.

When a transaction with this isolation level performs `UPDATE ... WHERE`, `DELETE ... WHERE`, `SELECT ... FOR UPDATE`, and `LOCK IN SHARE MODE` operations, other transactions might have to wait.

`SELECT ... FOR SHARE` replaces `SELECT ... LOCK IN SHARE MODE` in MySQL 8.0.1, but `LOCK IN SHARE MODE` remains available for backward compatibility.
See Also ACID, consistent read, isolation level, locking, phantom, transaction.

repertoire

Repertoire is a term applied to character sets. A character set repertoire is the collection of characters in the set. See Section 10.2.1, “Character Set Repertoire”.

replica

A database **server** machine in a **replication** topology that receives changes from another server (the **source**) and applies those same changes. Thus it maintains the same contents as the source, although it might lag somewhat behind.

In MySQL, replicas are commonly used in disaster recovery, to take the place of a source that fails. They are also commonly used for testing software upgrades and new settings, to ensure that database configuration changes do not cause problems with performance or reliability.

Replicas typically have high workloads, because they process all the **DML** (write) operations relayed from the source, as well as user queries. To ensure that replicas can apply changes from the source fast enough, they frequently have fast I/O devices and sufficient CPU and memory to run multiple database instances on the same server. For example, the source might use hard drive storage while the replicas use **SSD**s.
See Also DML, replication, server, source, SSD.

replication

The practice of sending changes from a **source**, to one or more **replicas**, so that all databases have the same data. This technique has a wide range of uses, such as load-balancing for better scalability, disaster

recovery, and testing software upgrades and configuration changes. The changes can be sent between the databases by methods called **row-based replication** and **statement-based replication**.
See Also replica, row-based replication, source, statement-based replication.

restore
The process of putting a set of backup files from the **MySQL Enterprise Backup** product in place for use by MySQL. This operation can be performed to fix a corrupted database, to return to some earlier point in time, or (in a **replication** context) to set up a new **replica**. In the **MySQL Enterprise Backup** product, this operation is performed by the `copy-back` option of the `mysqlbackup` command.
See Also hot backup, MySQL Enterprise Backup, mysqlbackup command, prepared backup, replica, replication.

rollback
A **SQL** statement that ends a **transaction**, undoing any changes made by the transaction. It is the opposite of **commit**, which makes permanent any changes made in the transaction.

By default, MySQL uses the **autocommit** setting, which automatically issues a commit following each SQL statement. You must change this setting before you can use the rollback technique.
See Also ACID, autocommit, commit, SQL, transaction.

rollback segment
The storage area containing the **undo logs**. Rollback segments have traditionally resided in the **system tablespace**. As of MySQL 5.6, rollback segments can reside in **undo tablespaces**. As of MySQL 5.7, rollback segments are also allocated to the *global temporary tablespace*.
See Also global temporary tablespace, system tablespace, undo log, undo tablespace.

row
The logical data structure defined by a set of **columns**. A set of rows makes up a **table**. Within `InnoDB` **data files**, each **page** can contain one or more rows.

Although `InnoDB` uses the term **row format** for consistency with MySQL syntax, the row format is a property of each table and applies to all rows in that table.
See Also column, data files, page, row format, table.

row format
The disk storage format for **rows** of an `InnoDB` **table**. As `InnoDB` gains new capabilities such as **compression**, new row formats are introduced to support the resulting improvements in storage efficiency and performance.

The row format of an `InnoDB` table is specified by the `ROW_FORMAT` option or by the `innodb_default_row_format` configuration option (introduced in MySQL 5.7.9). Row formats include `REDUNDANT`, `COMPACT`, `COMPRESSED`, and `DYNAMIC`. To view the row format of an `InnoDB` table, issue the `SHOW TABLE STATUS` statement or query `InnoDB` table metadata in the `INFORMATION_SCHEMA`.
See Also compact row format, compressed row format, compression, dynamic row format, redundant row format, row, table.

row lock
A **lock** that prevents a row from being accessed in an incompatible way by another **transaction**. Other rows in the same table can be freely written to by other transactions. This is the type of **locking** done by **DML** operations on **InnoDB** tables.

Contrast with **table locks** used by `MyISAM`, or during **DDL** operations on `InnoDB` tables that cannot be done with **online DDL**; those locks block concurrent access to the table.
See Also DDL, DML, InnoDB, lock, locking, online DDL, table lock, transaction.

row-based replication
A form of **replication** where events are propagated from the **source** specifying how to change individual rows on the **replica**. It is safe to use for all settings of the `innodb_autoinc_lock_mode` option.
See Also auto-increment locking, innodb_autoinc_lock_mode, replica, replication, source, statement-based replication.

row-level locking
: The **locking** mechanism used for **InnoDB** tables, relying on **row locks** rather than **table locks**. Multiple **transactions** can modify the same table concurrently. Only if two transactions try to modify the same row does one of the transactions wait for the other to complete (and release its row locks).
See Also InnoDB, locking, row lock, table lock, transaction.

Ruby
: A programming language that emphasizes dynamic typing and object-oriented programming. Some syntax is familiar to **Perl** developers.
See Also API, Perl, Ruby API.

Ruby API
: `mysql2`, based based on the **libmysqlclient** API library, is available for Ruby programmers developing MySQL applications. For more information, see Section 29.11, “MySQL Ruby APIs”.
See Also libmysql, Ruby.

rw-lock
: The low-level object that `InnoDB` uses to represent and enforce shared-access **locks** to internal in-memory data structures following certain rules. Contrast with **mutexes**, which `InnoDB` uses to represent and enforce exclusive access to internal in-memory data structures. Mutexes and rw-locks are known collectively as **latches**.

  `rw-lock` types include `s-locks` (shared locks), `x-locks` (exclusive locks), and `sx-locks` (shared-exclusive locks).

  - An `s-lock` provides read access to a common resource.
  - An `x-lock` provides write access to a common resource while not permitting inconsistent reads by other threads.
  - An `sx-lock` provides write access to a common resource while permitting inconsistent reads by other threads. `sx-locks` were introduced in MySQL 5.7 to optimize concurrency and improve scalability for read-write workloads.

  The following matrix summarizes rw-lock type compatibility.

	S	*SX*	*X*
S	Compatible	Compatible	Conflict
SX	Compatible	Conflict	Conflict
X	Conflict	Conflict	Conflict

  See Also latch, lock, mutex, Performance Schema.

S

savepoint
: Savepoints help to implement nested **transactions**. They can be used to provide scope to operations on tables that are part of a larger transaction. For example, scheduling a trip in a reservation system might involve booking several different flights; if a desired flight is unavailable, you might **roll back** the changes involved in booking that one leg, without rolling back the earlier flights that were successfully booked.
See Also rollback, transaction.

scalability
: The ability to add more work and issue more simultaneous requests to a system, without a sudden drop in performance due to exceeding the limits of system capacity. Software architecture, hardware configuration, application coding, and type of workload all play a part in scalability. When the system reaches its maximum capacity, popular techniques for increasing scalability are **scale up** (increasing the capacity of existing

hardware or software) and **scale out** (adding new servers and more instances of MySQL). Often paired with **availability** as critical aspects of a large-scale deployment.
See Also availability, scale out, scale up.

scale out
A technique for increasing **scalability** by adding new servers and more instances of MySQL. For example, setting up replication, NDB Cluster, connection pooling, or other features that spread work across a group of servers. Contrast with **scale up**.
See Also scalability, scale up.

scale up
A technique for increasing **scalability** by increasing the capacity of existing hardware or software. For example, increasing the memory on a server and adjusting memory-related parameters such as `innodb_buffer_pool_size` and `innodb_buffer_pool_instances`. Contrast with **scale out**.
See Also scalability, scale out.

schema
Conceptually, a schema is a set of interrelated database objects, such as tables, table columns, data types of the columns, indexes, foreign keys, and so on. These objects are connected through SQL syntax, because the columns make up the tables, the foreign keys refer to tables and columns, and so on. Ideally, they are also connected logically, working together as part of a unified application or flexible framework. For example, the **INFORMATION_SCHEMA** and **performance_schema** databases use "schema" in their names to emphasize the close relationships between the tables and columns they contain.

In MySQL, physically, a **schema** is synonymous with a **database**. You can substitute the keyword `SCHEMA` instead of `DATABASE` in MySQL SQL syntax, for example using `CREATE SCHEMA` instead of `CREATE DATABASE`.

Some other database products draw a distinction. For example, in the Oracle Database product, a **schema** represents only a part of a database: the tables and other objects owned by a single user.
See Also database, INFORMATION_SCHEMA, Performance Schema.

SDI
Acronym for "serialized dictionary information".
See Also serialized dictionary information (SDI).

search index
In MySQL, **full-text search** queries use a special kind of index, the **FULLTEXT index**. In MySQL 5.6.4 and up, `InnoDB` and `MyISAM` tables both support `FULLTEXT` indexes; formerly, these indexes were only available for `MyISAM` tables.
See Also full-text search, FULLTEXT index.

secondary index
A type of `InnoDB` **index** that represents a subset of table columns. An `InnoDB` table can have zero, one, or many secondary indexes. (Contrast with the **clustered index**, which is required for each `InnoDB` table, and stores the data for all the table columns.)

A secondary index can be used to satisfy queries that only require values from the indexed columns. For more complex queries, it can be used to identify the relevant rows in the table, which are then retrieved through lookups using the clustered index.

Creating and dropping secondary indexes has traditionally involved significant overhead from copying all the data in the `InnoDB` table. The **fast index creation** feature makes both `CREATE INDEX` and `DROP INDEX` statements much faster for `InnoDB` secondary indexes.
See Also clustered index, Fast Index Creation, index.

segment
A division within an `InnoDB` **tablespace**. If a tablespace is analogous to a directory, the segments are analogous to files within that directory. A segment can grow. New segments can be created.

For example, within a **file-per-table** tablespace, table data is in one segment and each associated index is in its own segment. The **system tablespace** contains many different segments, because it can hold many tables and their associated indexes. Prior to MySQL 8.0, the system tablespace also includes one or more **rollback segments** used for **undo logs**.

Segments grow and shrink as data is inserted and deleted. When a segment needs more room, it is extended by one **extent** (1 megabyte) at a time. Similarly, a segment releases one extent's worth of space when all the data in that extent is no longer needed.
See Also extent, file-per-table, rollback segment, system tablespace, tablespace, undo log.

selectivity
A property of data distribution, the number of distinct values in a column (its **cardinality**) divided by the number of records in the table. High selectivity means that the column values are relatively unique, and can retrieved efficiently through an index. If you (or the query optimizer) can predict that a test in a `WHERE` clause only matches a small number (or proportion) of rows in a table, the overall **query** tends to be efficient if it evaluates that test first, using an index.
See Also cardinality, query.

semi-consistent read
A type of read operation used for `UPDATE` statements, that is a combination of **READ COMMITTED** and **consistent read**. When an `UPDATE` statement examines a row that is already locked, `InnoDB` returns the latest committed version to MySQL so that MySQL can determine whether the row matches the `WHERE` condition of the `UPDATE`. If the row matches (must be updated), MySQL reads the row again, and this time `InnoDB` either locks it or waits for a lock on it. This type of read operation can only happen when the transaction has the READ COMMITTED **isolation level**.
See Also consistent read, isolation level, READ COMMITTED.

SERIALIZABLE
The **isolation level** that uses the most conservative locking strategy, to prevent any other **transactions** from inserting or changing data that was read by this transaction, until it is finished. This way, the same query can be run over and over within a transaction, and be certain to retrieve the same set of results each time. Any attempt to change data that was committed by another transaction since the start of the current transaction, cause the current transaction to wait.

This is the default isolation level specified by the SQL standard. In practice, this degree of strictness is rarely needed, so the default isolation level for `InnoDB` is the next most strict, **REPEATABLE READ**.
See Also ACID, consistent read, isolation level, locking, REPEATABLE READ, transaction.

serialized dictionary information (SDI)
Dictionary object metadata in serialized form. SDI is stored in `JSON` format.

As of MySQL 8.0.3, SDI is present in all `InnoDB` tablespace files except for temporary tablespace and undo tablespace files. The presence of SDI in tablespace files provides metadata redundancy. For example, dictionary object metadata can be extracted from tablespace files using the `ibd2sdi` utility if the data dictionary becomes unavailable.

For a `MyISAM` table, SDI is stored in a `.sdi` metadata file in the schema directory. An SDI metadata file is required to perform an `IMPORT TABLE` operation.
See Also file-per-table, general tablespace, system tablespace, tablespace.

server
A type of program that runs continuously, waiting to receive and act upon requests from another program (the **client**). Because often an entire computer is dedicated to running one or more server programs (such as a database server, a web server, an application server, or some combination of these), the term **server** can also refer to the computer that runs the server software.
See Also client, mysqld.

server-side prepared statement
A **prepared statement** managed by the MySQL server. Historically, issues with server-side prepared statements led **Connector/J** and **Connector/PHP** developers to sometimes use **client-side prepared**

statements instead. With modern MySQL server versions, server-side prepared statements are recommended for performance, scalability, and memory efficiency.
See Also client-side prepared statement, Connector/J, Connector/PHP, prepared statement.

service principal name
The name for a Kerberos named entity that represents a service.
See Also principal.

service ticket
A Kerberos ticket that provides access to an application service, such as the service provided by a web or database server.

servlet

See Also Connector/J.

session temporary tablespace
A *temporary tablespace* that stores user-created *temporary tables* and internal temporary tables created by the *optimizer* when `InnoDB` is configured as the on-disk storage engine for internal temporary tables.
See Also optimizer, temporary table, temporary tablespace.

shared lock
A kind of **lock** that allows other **transactions** to read the locked object, and to also acquire other shared locks on it, but not to write to it. The opposite of **exclusive lock**.
See Also exclusive lock, lock, transaction.

shared tablespace
Another way of referring to the **system tablespace** or a **general tablespace**. General tablespaces were introduced in MySQL 5.7. More than one table can reside in a shared tablespace. Only a single table can reside in a *file-per-table* tablespace.
See Also general tablespace, system tablespace.

sharp checkpoint
The process of **flushing** to disk all **dirty** buffer pool pages whose redo entries are contained in certain portion of the **redo log**. Occurs before `InnoDB` reuses a portion of a log file; the log files are used in a circular fashion. Typically occurs with write-intensive **workloads**.
See Also dirty page, flush, redo log, workload.

shutdown
The process of stopping the MySQL server. By default, this process cleans up operations for **InnoDB** tables, so `InnoDB` can be **slow** to shut down, but fast to start up later. If you skip the cleanup operations, it is **fast** to shut down but the cleanup must be performed during the next restart.

The shutdown mode for `InnoDB` is controlled by the `innodb_fast_shutdown` option.
See Also fast shutdown, InnoDB, slow shutdown, startup.

slave
See replica.

slow query log
A type of **log** used for performance tuning of SQL statements processed by the MySQL server. The log information is stored in a file. You must enable this feature to use it. You control which categories of “slow” SQL statements are logged. For more information, see Section 5.4.5, “The Slow Query Log”.
See Also general query log, log.

slow shutdown
A type of **shutdown** that does additional `InnoDB` flushing operations before completing. Also known as a **clean shutdown**. Specified by the configuration parameter `innodb_fast_shutdown=0` or the command `SET GLOBAL innodb_fast_shutdown=0;`. Although the shutdown itself can take longer, that time should be saved on the subsequent startup.
See Also clean shutdown, fast shutdown, shutdown.

snapshot
A representation of data at a particular time, which remains the same even as changes are **committed** by other **transactions**. Used by certain **isolation levels** to allow **consistent reads**.
See Also commit, consistent read, isolation level, transaction.

sort buffer
The buffer used for sorting data during creation of an `InnoDB` index. Sort buffer size is configured using the `innodb_sort_buffer_size` configuration option.

source
A database server machine in a **replication** scenario that processes the initial insert, update, and delete requests for data. These changes are propagated to, and repeated on, other servers known as **replicas**.
See Also replica, replication.

space ID
An identifier used to uniquely identify an `InnoDB` **tablespace** within a MySQL instance. The space ID for the **system tablespace** is always zero; this same ID applies to all tables within the system tablespace or within a general tablespace. Each **file-per-table** tablespace and **general tablespace** has its own space ID.

Prior to MySQL 5.6, this hardcoded value presented difficulties in moving `InnoDB` tablespace files between MySQL instances. Starting in MySQL 5.6, you can copy tablespace files between instances by using the **transportable tablespace** feature involving the statements `FLUSH TABLES ... FOR EXPORT`, `ALTER TABLE ... DISCARD TABLESPACE`, and `ALTER TABLE ... IMPORT TABLESPACE`. The information needed to adjust the space ID is conveyed in the **.cfg file** which you copy along with the tablespace. See Section 15.6.1.3, “Importing InnoDB Tables” for details.
See Also .cfg file, file-per-table, general tablespace, .ibd file, system tablespace, tablespace, transportable tablespace.

sparse file
A type of file that uses file system space more efficiently by writing metadata representing empty blocks to disk instead of writing the actual empty space. The `InnoDB` **transparent page compression** feature relies on sparse file support. For more information, see Section 15.9.2, “InnoDB Page Compression”.
See Also hole punching, transparent page compression.

spin
A type of **wait** operation that continuously tests whether a resource becomes available. This technique is used for resources that are typically held only for brief periods, where it is more efficient to wait in a “busy loop” than to put the thread to sleep and perform a context switch. If the resource does not become available within a short time, the spin loop ceases and another wait technique is used.
See Also latch, lock, mutex, wait.

SPN
See service principal name.

Spring
A Java-based application framework designed for assisting in application design by providing a way to configure components.
See Also J2EE.

SQL
The Structured Query Language that is standard for performing database operations. Often divided into the categories **DDL**, **DML**, and **queries**. MySQL includes some additional statement categories such as **replication**. See Chapter 9, *Language Structure* for the building blocks of SQL syntax, Chapter 11, *Data Types* for the data types to use for MySQL table columns, Chapter 13, *SQL Statements* for details about SQL statements and their associated categories, and Chapter 12, *Functions and Operators* for standard and MySQL-specific functions to use in queries.
See Also DDL, DML, query, replication.

SQLState
An error code defined by the **JDBC** standard, for exception handling by applications using **Connector/J**.

See Also Connector/J, JDBC.

SSD
: Acronym for “solid-state drive”. A type of storage device with different performance characteristics than a traditional hard disk drive (**HDD**): smaller storage capacity, faster for random reads, no moving parts, and with a number of considerations affecting write performance. Its performance characteristics can influence the throughput of a **disk-bound** workload.
See Also disk-bound, HDD.

SSL
: Acronym for “secure sockets layer”. Provides the encryption layer for network communication between an application and a MySQL database server.
See Also keystore, truststore.

ST
: See service ticket.

startup
: The process of starting the MySQL server. Typically done by one of the programs listed in Section 4.3, “Server and Server-Startup Programs”. The opposite of **shutdown**.
See Also shutdown.

statement interceptor
: A type of **interceptor** for tracing, debugging, or augmenting SQL statements issued by a database application. Sometimes also known as a **command interceptor**.

  In **Java** applications using **Connector/J**, setting up this type of interceptor involves implementing the `com.mysql.jdbc.StatementInterceptorV2` interface, and adding a `statementInterceptors` property to the **connection string**.

  In **Visual Studio** applications using **Connector/NET**, setting up this type of interceptor involves defining a class that inherits from the `BaseCommandInterceptor` class and specifying that class name as part of the connection string.
  See Also command interceptor, connection string, Connector/J, Connector/NET, interceptor, Java, Visual Studio.

statement-based replication
: A form of **replication** where SQL statements are sent from the **source** and replayed on the **replica**. It requires some care with the setting for the `innodb_autoinc_lock_mode` option, to avoid potential timing problems with **auto-increment locking**.
See Also auto-increment locking, innodb_autoinc_lock_mode, replica, replication, row-based replication, source.

statistics
: Estimated values relating to each `InnoDB` **table** and **index**, used to construct an efficient **query execution plan**. The main values are the **cardinality** (number of distinct values) and the total number of table rows or index entries. The statistics for the table represent the data in its **primary key** index. The statistics for a **secondary index** represent the rows covered by that index.

  The values are estimated rather than counted precisely because at any moment, different **transactions** can be inserting and deleting rows from the same table. To keep the values from being recalculated frequently, you can enable **persistent statistics**, where the values are stored in `InnoDB` system tables, and refreshed only when you issue an `ANALYZE TABLE` statement.

  You can control how **NULL** values are treated when calculating statistics through the `innodb_stats_method` configuration option.

  Other types of statistics are available for database objects and database activity through the **INFORMATION_SCHEMA** and **PERFORMANCE_SCHEMA** tables.
  See Also cardinality, index, INFORMATION_SCHEMA, NULL, Performance Schema, persistent statistics, primary key, query execution plan, secondary index, table, transaction.

stemming
The ability to search for different variations of a word based on a common root word, such as singular and plural, or past, present, and future verb tense. This feature is currently supported in `MyISAM` **full-text search** feature but not in **FULLTEXT indexes** for `InnoDB` tables.
See Also full-text search, FULLTEXT index.

stopword
In a **FULLTEXT index**, a word that is considered common or trivial enough that it is omitted from the **search index** and ignored in search queries. Different configuration settings control stopword processing for `InnoDB` and `MyISAM` tables. See Section 12.10.4, "Full-Text Stopwords" for details.
See Also FULLTEXT index, search index.

storage engine
A component of the MySQL database that performs the low-level work of storing, updating, and querying data. In MySQL 5.5 and higher, **InnoDB** is the default storage engine for new tables, superceding `MyISAM`. Different storage engines are designed with different tradeoffs between factors such as memory usage versus disk usage, read speed versus write speed, and speed versus robustness. Each storage engine manages specific tables, so we refer to `InnoDB` tables, `MyISAM` tables, and so on.

The **MySQL Enterprise Backup** product is optimized for backing up `InnoDB` tables. It can also back up tables handled by `MyISAM` and other storage engines.
See Also InnoDB, MySQL Enterprise Backup, table type.

stored generated column
A column whose values are computed from an expression included in the column definition. Column values are evaluated and stored when rows are inserted or updated. A stored generated column requires storage space and can be indexed.

Contrast with **virtual generated column**.
See Also base column, generated column, virtual generated column.

stored object
A stored program or view.

stored program
A stored routine (procedure or function), trigger, or Event Scheduler event.

stored routine
A stored procedure or function.

strict mode
The general name for the setting controlled by the `innodb_strict_mode` option. Turning on this setting causes certain conditions that are normally treated as warnings, to be considered errors. For example, certain invalid combinations of options related to **file format** and **row format**, that normally produce a warning and continue with default values, now cause the `CREATE TABLE` operation to fail. `innodb_strict_mode` is enabled by default in MySQL 5.7.

MySQL also has something called strict mode. See Section 5.1.11, "Server SQL Modes".
See Also file format, innodb_strict_mode, row format.

sublist
Within the list structure that represents the **buffer pool**, pages that are relatively old and relatively new are represented by different portions of the **list**. A set of parameters control the size of these portions and the dividing point between the new and old pages.
See Also buffer pool, eviction, list, LRU.

supremum record
A **pseudo-record** in an index, representing the **gap** above the largest value in that index. If a transaction has a statement such as `SELECT ... FROM ... WHERE col > 10 FOR UPDATE;`, and the largest value in the column is 20, it is a lock on the supremum record that prevents other transactions from inserting even larger values such as 50, 100, and so on.

See Also gap, infimum record, pseudo-record.

surrogate key
Synonym name for **synthetic key**.
See Also synthetic key.

synthetic key
An indexed column, typically a **primary key**, where the values are assigned arbitrarily. Often done using an **auto-increment** column. By treating the value as completely arbitrary, you can avoid overly restrictive rules and faulty application assumptions. For example, a numeric sequence representing employee numbers might have a gap if an employee was approved for hiring but never actually joined. Or employee number 100 might have a later hiring date than employee number 500, if they left the company and later rejoined. Numeric values also produce shorter values of predictable length. For example, storing numeric codes meaning "Road", "Boulevard", "Expressway", and so on is more space-efficient than repeating those strings over and over.

Also known as a **surrogate key**. Contrast with **natural key**.
See Also auto-increment, natural key, primary key, surrogate key.

system tablespace
One or more data files (**ibdata files**) containing the metadata for `InnoDB`-related objects, and the storage areas for the **change buffer**, and the **doublewrite buffer**. It may also contain table and index data for `InnoDB` tables if tables were created in the system tablespace instead of **file-per-table** or **general tablespaces**. The data and metadata in the system tablespace apply to all **databases** in a MySQL **instance**.

Prior to MySQL 5.6.7, the default was to keep all `InnoDB` tables and indexes inside the system tablespace, often causing this file to become very large. Because the system tablespace never shrinks, storage problems could arise if large amounts of temporary data were loaded and then deleted. In MySQL 8.0, the default is **file-per-table** mode, where each table and its associated indexes are stored in a separate **.ibd file**. This default makes it easier to use `InnoDB` features that rely on `DYNAMIC` and `COMPRESSED` row formats, such as table **compression**, efficient storage of **off-page columns**, and large index key prefixes.

Keeping all table data in the system tablespace or in separate `.ibd` files has implications for storage management in general. The **MySQL Enterprise Backup** product might back up a small set of large files, or many smaller files. On systems with thousands of tables, the file system operations to process thousands of `.ibd` files can cause bottlenecks.

`InnoDB` introduced general tablespaces in MySQL 5.7.6, which are also represented by `.ibd` files. General tablespaces are shared tablespaces created using `CREATE TABLESPACE` syntax. They can be created outside of the data directory, are capable of holding multiple tables, and support tables of all row formats.
See Also change buffer, compression, data dictionary, database, doublewrite buffer, dynamic row format, file-per-table, general tablespace, .ibd file, ibdata file, innodb_file_per_table, instance, MySQL Enterprise Backup, off-page column, tablespace, undo log.

T

table
Each MySQL table is associated with a particular **storage engine**. **InnoDB** tables have particular **physical** and **logical** characteristics that affect performance, **scalability**, **backup**, administration, and application development.

In terms of file storage, an `InnoDB` table belongs to one of the following tablespace types:

- The shared `InnoDB` **system tablespace**, which is comprised of one or more **ibdata files**.
- A **file-per-table** tablespace, comprised of an individual **.ibd file**.
- A shared **general tablespace**, comprised of an individual `.ibd` file. General tablespaces were introduced in MySQL 5.7.6.

`.ibd` data files contain both table and **index** data.

`InnoDB` tables created in file-per-table tablespaces can use **DYNAMIC** or **COMPRESSED** row format. These row formats enable `InnoDB` features such as **compression**, efficient storage of **off-page columns**, and large index key prefixes. General tablespaces support all row formats.

The system tablespace supports tables that use **REDUNDANT**, **COMPACT**, and **DYNAMIC** row formats. System tablespace support for the **DYNAMIC** row format was added in MySQL 5.7.6.

The **rows** of an `InnoDB` table are organized into an index structure known as the **clustered index**, with entries sorted based on the **primary key** columns of the table. Data access is optimized for queries that filter and sort on the primary key columns, and each index contains a copy of the associated primary key columns for each entry. Modifying values for any of the primary key columns is an expensive operation. Thus an important aspect of `InnoDB` table design is choosing a primary key with columns that are used in the most important queries, and keeping the primary key short, with rarely changing values.
See Also backup, clustered index, compact row format, compressed row format, compression, dynamic row format, Fast Index Creation, file-per-table, .ibd file, index, off-page column, primary key, redundant row format, row, system tablespace, tablespace.

table lock
A lock that prevents any other **transaction** from accessing a table. `InnoDB` makes considerable effort to make such locks unnecessary, by using techniques such as **online DDL**, **row locks** and **consistent reads** for processing **DML** statements and **queries**. You can create such a lock through SQL using the `LOCK TABLE` statement; one of the steps in migrating from other database systems or MySQL storage engines is to remove such statements wherever practical.
See Also consistent read, DML, lock, locking, online DDL, query, row lock, table, transaction.

table scan
See full table scan.

table statistics
See statistics.

table type
Obsolete synonym for **storage engine**. We refer to `InnoDB` tables, `MyISAM` tables, and so on.
See Also InnoDB, storage engine.

tablespace
A data file that can hold data for one or more `InnoDB` **tables** and associated **indexes**.

The **system tablespace** contains the `InnoDB` **data dictionary**, and prior to MySQL 5.6 holds all other `InnoDB` tables by default.

The `innodb_file_per_table` option, enabled by default in MySQL 5.6 and higher, allows tables to be created in their own tablespaces. File-per-table tablespaces support features such as efficient storage of **off-page columns**, table compression, and transportable tablespaces. See Section 15.6.3.2, “File-Per-Table Tablespaces” for details.

`InnoDB` introduced general tablespaces in MySQL 5.7.6. General tablespaces are shared tablespaces created using `CREATE TABLESPACE` syntax. They can be created outside of the MySQL data directory, are capable of holding multiple tables, and support tables of all row formats.

MySQL NDB Cluster also groups its tables into tablespaces. See Section 23.6.11.1, “NDB Cluster Disk Data Objects” for details.
See Also compressed row format, data dictionary, data files, file-per-table, general tablespace, index, innodb_file_per_table, system tablespace, table.

Tcl
A programming language originating in the Unix scripting world. Sometimes extended by code written in **C**, **C++**, or **Java**. For the open-source Tcl **API** for MySQL, see Section 29.12, “MySQL Tcl API”.

See Also API.

temporary table
A **table** whose data does not need to be truly permanent. For example, temporary tables might be used as storage areas for intermediate results in complicated calculations or transformations; this intermediate data would not need to be recovered after a crash. Database products can take various shortcuts to improve the performance of operations on temporary tables, by being less scrupulous about writing data to disk and other measures to protect the data across restarts.

Sometimes, the data itself is removed automatically at a set time, such as when the transaction ends or when the session ends. With some database products, the table itself is removed automatically too.
See Also table.

temporary tablespace
`InnoDB` uses two types of temporary tablespace. *Session temporary tablespaces* store user-created temporary tables and internal temporary tables created by the optimizer. The *global temporary tablespace* stores *rollback segments* for changes made to user-created temporary tables.
See Also global temporary tablespace, session temporary tablespace, temporary table.

text collection
The set of columns included in a **FULLTEXT index**.
See Also FULLTEXT index.

TGS
A Kerberos ticket-granting server. TGS can also refer to the ticket-granting service provided by a ticket-granting server.
See Also ticket-granting server.

TGT
See ticket-granting ticket.

thread
A unit of processing that is typically more lightweight than a **process**, allowing for greater **concurrency**.
See Also concurrency, master thread, process, Pthreads.

ticket-granting server
In Kerberos, a server that provides tickets. The ticket-granting server (TGS) combined with an authentication server (AS) make up a key distribution center (KDC).

TGS can also refer to the ticket-granting service provided by the ticket-granting server.
See Also authentication server, key distribution center.

ticket-granting ticket
In Kerberos, a ticket-granting ticket is presented to the ticket-granting server (TGS) to obtain service tickets for service access.
See Also ticket-granting server.

Tomcat
An open source **J2EE** application server, implementing the Java Servlet and JavaServer Pages programming technologies. Consists of a web server and Java servlet container. With MySQL, typically used in conjunction with **Connector/J**.
See Also J2EE.

torn page
An error condition that can occur due to a combination of I/O device configuration and hardware failure. If data is written out in chunks smaller than the `InnoDB` **page size** (by default, 16KB), a hardware failure while writing could result in only part of a page being stored to disk. The `InnoDB` **doublewrite buffer** guards against this possibility.
See Also doublewrite buffer.

TPS

Acronym for “**transactions** per second”, a unit of measurement sometimes used in benchmarks. Its value depends on the **workload** represented by a particular benchmark test, combined with factors that you control such as the hardware capacity and database configuration.
See Also transaction, workload.

transaction

Transactions are atomic units of work that can be **committed** or **rolled back**. When a transaction makes multiple changes to the database, either all the changes succeed when the transaction is committed, or all the changes are undone when the transaction is rolled back.

Database transactions, as implemented by `InnoDB`, have properties that are collectively known by the acronym **ACID**, for atomicity, consistency, isolation, and durability.
See Also ACID, commit, isolation level, lock, rollback.

transaction ID

An internal field associated with each **row**. This field is physically changed by `INSERT`, `UPDATE`, and `DELETE` operations to record which **transaction** has locked the row.
See Also implicit row lock, row, transaction.

transparent page compression

A feature added in MySQL 5.7.8 that permits page-level compression for `InnoDB` tables that reside in **file-per-table** tablespaces. Page compression is enabled by specifying the `COMPRESSION` attribute with `CREATE TABLE` or `ALTER TABLE`. For more information, see Section 15.9.2, “InnoDB Page Compression”.
See Also file-per-table, hole punching, sparse file.

transportable tablespace

A feature that allows a **tablespace** to be moved from one instance to another. Traditionally, this has not been possible for `InnoDB` tablespaces because all table data was part of the **system tablespace**. In MySQL 5.6 and higher, the `FLUSH TABLES ... FOR EXPORT` syntax prepares an `InnoDB` table for copying to another server; running `ALTER TABLE ... DISCARD TABLESPACE` and `ALTER TABLE ... IMPORT TABLESPACE` on the other server brings the copied data file into the other instance. A separate **.cfg file**, copied along with the **.ibd file**, is used to update the table metadata (for example the **space ID**) as the tablespace is imported. See Section 15.6.1.3, “Importing InnoDB Tables” for usage information.
See Also .cfg file, .ibd file, space ID, system tablespace, tablespace.

troubleshooting

The process of determining the source of a problem. Some of the resources for troubleshooting MySQL problems include:

- Section 2.9.2.1, “Troubleshooting Problems Starting the MySQL Server”
- Section 6.2.22, “Troubleshooting Problems Connecting to MySQL”
- Section B.3.3.2, “How to Reset the Root Password”
- Section B.3.2, “Common Errors When Using MySQL Programs”
- Section 15.21, “InnoDB Troubleshooting”.

truncate

A **DDL** operation that removes the entire contents of a table, while leaving the table and related indexes intact. Contrast with **drop**. Although conceptually it has the same result as a `DELETE` statement with no `WHERE` clause, it operates differently behind the scenes: `InnoDB` creates a new empty table, drops the old table, then renames the new table to take the place of the old one. Because this is a DDL operation, it cannot be **rolled back**.

If the table being truncated contains **foreign keys** that reference another table, the truncation operation uses a slower method of operation, deleting one row at a time so that corresponding rows in the referenced table can be deleted as needed by any `ON DELETE CASCADE` clause. (MySQL 5.5 and higher do not allow this

slower form of truncate, and return an error instead if foreign keys are involved. In this case, use a `DELETE` statement instead.
See Also DDL, drop, foreign key, rollback.

truststore

See Also SSL.

tuple
A technical term designating an ordered set of elements. It is an abstract notion, used in formal discussions of database theory. In the database field, tuples are usually represented by the columns of a table row. They could also be represented by the result sets of queries, for example, queries that retrieved only some columns of a table, or columns from joined tables.
See Also cursor.

two-phase commit
An operation that is part of a distributed **transaction**, under the **XA** specification. (Sometimes abbreviated as 2PC.) When multiple databases participate in the transaction, either all databases **commit** the changes, or all databases **roll back** the changes.
See Also commit, rollback, transaction, XA.

U

undo
Data that is maintained throughout the life of a **transaction**, recording all changes so that they can be undone in case of a **rollback** operation. It is stored in **undo logs** either within the **system tablespace** (in MySQL 5.7 or earlier) or in separate **undo tablespaces**. As of MySQL 8.0, undo logs reside in undo tablespaces by default.
See Also rollback, rollback segment, system tablespace, transaction, undo log, undo tablespace.

undo buffer
See undo log.

undo log
A storage area that holds copies of data modified by active **transactions**. If another transaction needs to see the original data (as part of a **consistent read** operation), the unmodified data is retrieved from this storage area.

In MySQL 5.6 and MySQL 5.7, you can use the `innodb_undo_tablespaces` variable have undo logs reside in **undo tablespaces**, which can be placed on another storage device such as an **SSD**. In MySQL 8.0, undo logs reside in two default undo tablespaces that are created when MySQL is initialized, and additional undo tablespaces can be created using `CREATE UNDO TABLESPACE` syntax.

The undo log is split into separate portions, the **insert undo buffer** and the **update undo buffer**.
See Also consistent read, rollback segment, SSD, system tablespace, transaction, undo tablespace.

undo log segment
A collection of **undo logs**. Undo log segments exists within **rollback segments**. An undo log segment might contain undo logs from multiple transactions. An undo log segment can only be used by one transaction at a time but can be reused after it is released at transaction **commit** or **rollback**. May also be referred to as an “undo segment”.
See Also commit, rollback, rollback segment, undo log.

undo tablespace
An undo tablespace contains **undo logs**. Undo logs exist within **undo log segments**, which are contained within **rollback segments**. Rollback segments have traditionally resided in the system tablespace. As of MySQL 5.6, rollback segments can reside in undo tablespaces. In MySQL 5.6 and MySQL 5.7, the number of undo tablespaces is controlled by the `innodb_undo_tablespaces` configuration option. In MySQL 8.0, two default undo tablespaces are created when the MySQL instance is initialized, and additional undo tablespaces can be created using `CREATE UNDO TABLESPACE` syntax.

For more information, see Section 15.6.3.4, “Undo Tablespaces”.
See Also rollback segment, system tablespace, undo log, undo log segment.

Unicode
A system for supporting national characters, character sets, code pages, and other internationalization aspects in a flexible and standardized way.

Unicode support is an important aspect of the **ODBC** standard. **Connector/ODBC** 5.1 is a Unicode driver, as opposed to Connector/ODBC 3.51, which is an **ANSI** driver.
See Also ANSI, Connector/ODBC, ODBC.

unique constraint
A kind of **constraint** that asserts that a column cannot contain any duplicate values. In terms of **relational** algebra, it is used to specify 1-to-1 relationships. For efficiency in checking whether a value can be inserted (that is, the value does not already exist in the column), a unique constraint is supported by an underlying **unique index**.
See Also constraint, relational, unique index.

unique index
An index on a column or set of columns that have a **unique constraint**. Because the index is known not to contain any duplicate values, certain kinds of lookups and count operations are more efficient than in the normal kind of index. Most of the lookups against this type of index are simply to determine if a certain value exists or not. The number of values in the index is the same as the number of rows in the table, or at least the number of rows with non-null values for the associated columns.

Change buffering optimization does not apply to unique indexes. As a workaround, you can temporarily set `unique_checks=0` while doing a bulk data load into an `InnoDB` table.
See Also cardinality, change buffering, unique constraint, unique key.

unique key
The set of columns (one or more) comprising a **unique index**. When you can define a `WHERE` condition that matches exactly one row, and the query can use an associated unique index, the lookup and error handling can be performed very efficiently.
See Also cardinality, unique constraint, unique index.

UPN
See user principal name.

user principal name
The name for a Kerberos named entity that represents a user.
See Also principal.

V

variable-length type
A data type of variable length. `VARCHAR`, `VARBINARY`, and `BLOB` and `TEXT` types are variable-length types.

`InnoDB` treats fixed-length fields greater than or equal to 768 bytes in length as variable-length fields, which can be stored **off-page**. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.
See Also off-page column, overflow page.

victim
The **transaction** that is automatically chosen to be **rolled back** when a **deadlock** is detected. `InnoDB` rolls back the transaction that has updated the fewest rows.

Deadlock detection can be disabled using the `innodb_deadlock_detect` configuration option.
See Also deadlock, deadlock detection, innodb_lock_wait_timeout, transaction.

view
A stored query that when invoked produces a result set. A view acts as a virtual table.

virtual column
See virtual generated column.

virtual generated column
A column whose values are computed from an expression included in the column definition. Column values are not stored, but are evaluated when rows are read, immediately after any `BEFORE` triggers. A virtual generated column takes no storage. `InnoDB` supports secondary indexes on virtual generated columns.

Contrast with **stored generated column**.
See Also base column, generated column, stored generated column.

virtual index
A virtual index is a **secondary index** on one or more **virtual generated columns** or on a combination of virtual generated columns and regular columns or stored generated columns. For more information, see Section 13.1.20.9, “Secondary Indexes and Generated Columns”.
See Also secondary index, stored generated column, virtual generated column.

Visual Studio
For supported versions of Visual Studio, see the following references:

- Connector/NET: Connector/NET Versions
- Connector/C++ 8.0: Platform Support and Prerequisites

See Also Connector/C++, Connector/NET.

W

wait
When an operation, such as acquiring a **lock**, **mutex**, or **latch**, cannot be completed immediately, `InnoDB` pauses and tries again. The mechanism for pausing is elaborate enough that this operation has its own name, the **wait**. Individual threads are paused using a combination of internal `InnoDB` scheduling, operating system `wait()` calls, and short-duration **spin** loops.

On systems with heavy load and many transactions, you might use the output from the `SHOW INNODB STATUS` command or **Performance Schema** to determine whether threads are spending too much time waiting, and if so, how you can improve **concurrency**.
See Also concurrency, latch, lock, mutex, Performance Schema, spin.

warm backup
A **backup** taken while the database is running, but that restricts some database operations during the backup process. For example, tables might become read-only. For busy applications and websites, you might prefer a **hot backup**.
See Also backup, cold backup, hot backup.

warm up
To run a system under a typical **workload** for some time after startup, so that the **buffer pool** and other memory regions are filled as they would be under normal conditions. This process happens naturally over time when a MySQL server is restarted or subjected to a new workload.

Typically, you run a workload for some time to warm up the buffer pool before running performance tests, to ensure consistent results across multiple runs; otherwise, performance might be artificially low during the first run.

In MySQL 5.6, you can speed up the warmup process by enabling the `innodb_buffer_pool_dump_at_shutdown` and `innodb_buffer_pool_load_at_startup` configuration options, to bring the contents of the buffer pool back into memory after a restart. These options are enabled by default in MySQL 5.7. See Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.
See Also buffer pool, workload.

workload

The combination and volume of **SQL** and other database operations, performed by a database application during typical or peak usage. You can subject the database to a particular workload during performance testing to identify **bottlenecks**, or during capacity planning.
See Also bottleneck, CPU-bound, disk-bound, SQL.

write combining

An optimization technique that reduces write operations when **dirty pages** are **flushed** from the `InnoDB` **buffer pool**. If a row in a page is updated multiple times, or multiple rows on the same page are updated, all of those changes are stored to the data files in a single write operation rather than one write for each change.
See Also buffer pool, dirty page, flush.

X

XA

A standard interface for coordinating distributed **transactions**, allowing multiple databases to participate in a transaction while maintaining **ACID** compliance. For full details, see Section 13.3.8, “XA Transactions”.

XA Distributed Transaction support is enabled by default.
See Also ACID, binary log, commit, transaction, two-phase commit.

Y

young

A characteristic of a **page** in the `InnoDB` **buffer pool** meaning that it has been accessed recently, and so is moved within the buffer pool data structure, so that it is not **flushed** too soon by the **LRU** algorithm. This term is used in some **INFORMATION_SCHEMA** column names of tables related to the buffer pool.
See Also buffer pool, flush, INFORMATION_SCHEMA, LRU, page.