MySQL Administration

MySQL Architecture and Components

MySQL Physical and Logical components explanation with infographics

Author: Lalit Choudhary

Contents

MySQL 5.7
References: MySQL 5.7 Reference Manual

Why this Guide?

If you are already or planning to work on MySQL, then the must thing you should know is "MySQL Architecture". Purpose of this guide is, to give quick overview about MySQL Architecture and its components.

Unlike the other RDBMS databases, MySQL is a very flexible and offers various features along with different kinds of storage engines as a plugin for different kinds of needs. Because of this, MySQL architecture and behavior will also change as per the use of storage engines, for example transactional [InnoDB] and non-transactional [MyISAM] engines data storage and SQL execution methods will be different and within the server it will use engine specific components like query execution plan, indexes, memory and buffers depending on type storage engine will get used for the SQL operation.
Will discuss more about InnoDB, since its default and main storage engine for MySQL.

MySQL Architecture

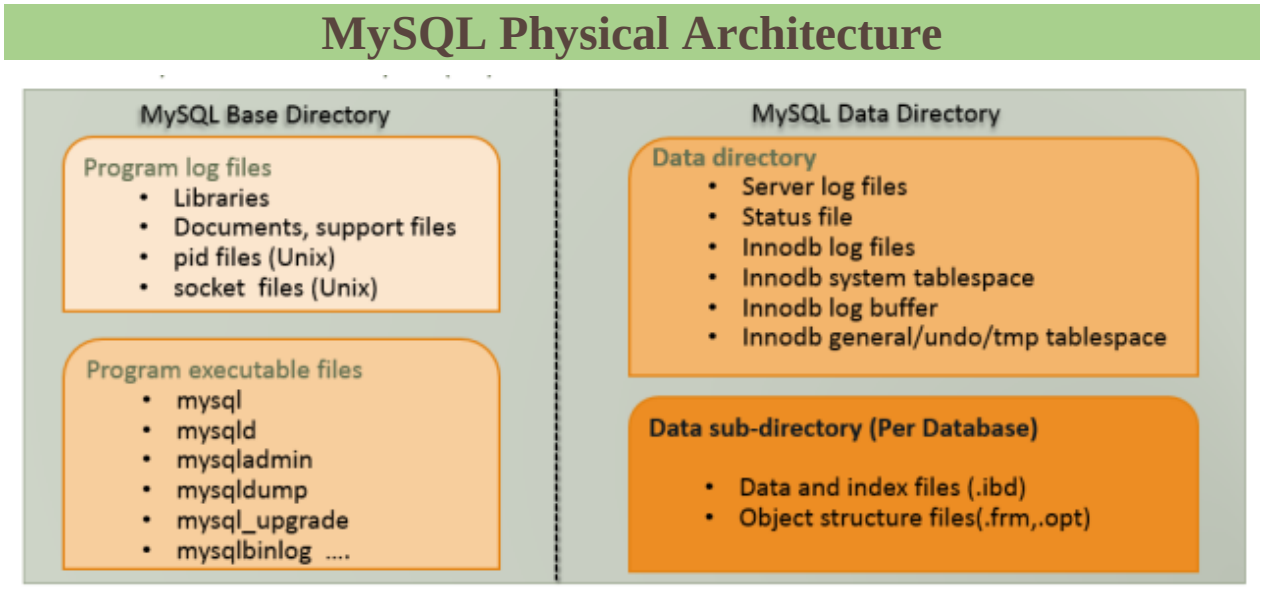


Figure 1: MySQL physical architecture

Configuration files:

auto.cnf: Contains server_uuid
my.cnf: MySQL Configuration file.

MySQL Log files:

There few important log files provided by MySQL to trace and store various types of database related operations and activities.

Error log:

Also called as 'MySQL Server Logs'. By Default enable and default location is data directory.
MySQL server/Error Log file for logging server activities like startup- shutdown events, crash-recovery events, connection errors etc.
MySQL server log file to record mysqld server activities as NOTE, WARNING, ERROR on the pre-defined severity basis.

Variable: log_error
log-error = mysqld.log

General Log:

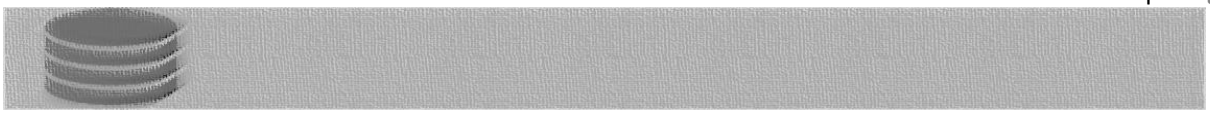
By Default this log is disable, we can enable and disable it dynamically without downtime.
To log client queries. Enabled it when you want to troubleshoot queries executed from client to see what client sending to mysqld server.

Variables:
general_log
general_log_file

Slow Query log:

By Default this log is disable, we can enable and disable it dynamically without downtime. The slow query log contain SQL query with additional info that took more than specified long_query_time (in seconds) to execute.
Threshold time in seconds for a query execution, to mark that as slow query.

Variables:
slow_query_log
slow_query_log_file



Binary log:

Also called as 'binlog', it contains events for database related changes like table creation and data changes.
By Default this log is disable, we cannot enable and disable it dynamically and requires downtime.
This log is very useful for *Point in Time* recovery as well as for *Replication*, it is mandatory to binlog enabled on MySQL server if you want to use MySQL Replication.
Binary file has binary format, to see the contents in this file we can use **mysqlbinlog** utility which will convert this file in text format and same can be used for Point in time restore to apply incremental changes.

Variables:
`log_bin`

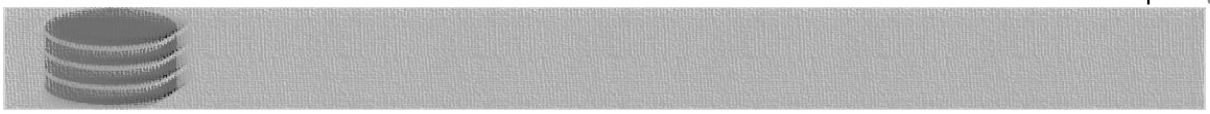
Miscellaneous files:

basedir = dir_name
Directory location to the MySQL binaries installation.

datadir = dir_name
Directory location to the MySQL database data, status and log files.

pid-file = file_name
Filename in which the mysqld server should write its process ID information.

socket = file_name, -S file_name
On Unix, the name of the Unix socket file to use, for connections made using a named pipe to a local server.



MySQL Logical Architecture

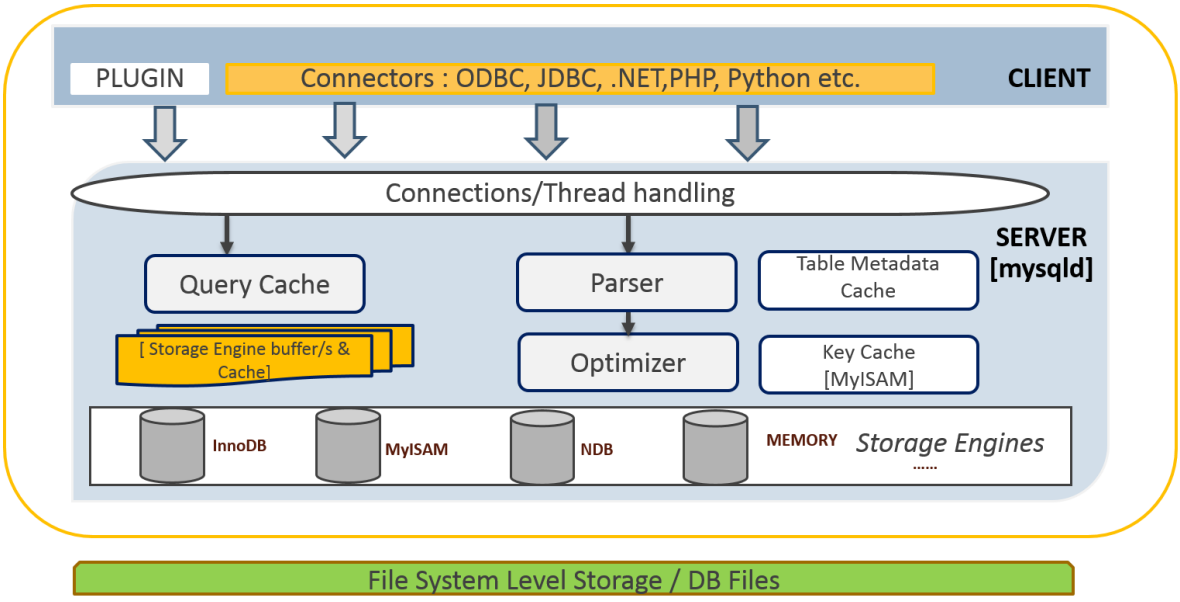


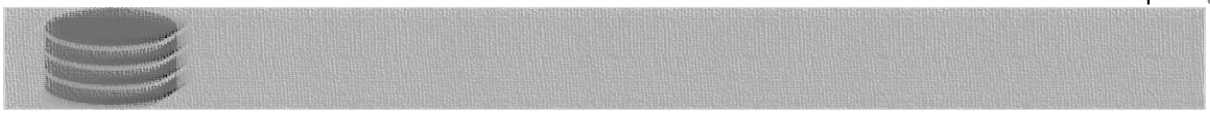
Figure 2: MySQL logical architecture

Client:
Utility/tool to connect and communicate to MySQL server.
Eg. mysql

Server:
MySQL instance where actual data getting stored and data processing is happening.

mysqld:
MySQL Server **daemon** program which runs in the background and manages database related incoming and outgoing requests from clients. **mysqld** is a multi-threaded process which allows multiple connection and manages them in parallel for MySQL server.

MySQL memory allocation:
Main MySQL memory is dynamic, examples innodb_buffer_pool_size (from 5.7.5), key_buffer_size etc. and working on shared nothing principal which means, every session has unique execution plan and we can share data sets only for the same session.



GLOBAL Level:

- Allocated once
- Shared by the server process and its threads

SESSION Level:

- Allocated for each mysql client session
- Dynamically allocated and deallocated
- Used for handling query result
- Buffer size per session

Connection/Thread Handling:
Manages client connections/sessions [mysql threads]. Task like Authentication (user credentials) and Authorization (User access privileges).

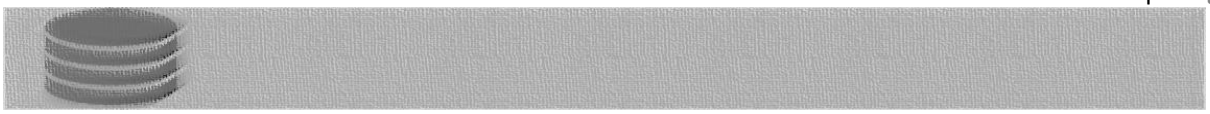
Parser:
Check for SQL syntax by checking every character in SQL query and generate *SQL_ID* for each SQL query.

Optimizer:
Creates efficient query execution plan as per the storage engine. It will rewrite a query for optimal execution.
Example: InnoDB has shared buffer so optimizer will get pre-cached data from it. Using table statistics optimizer will generate an execution plan for a SQL query.

Metadata cache:
Cache for object metadata information and db objects stats..

Query cache:
Shared identical queries from memory. If an identical query from client found in a query cache then, MySQL server retrieves the result from the query cache for that query rather than parsing and executing it again. It's a shared cache for sessions, so a result set generated by one client can be send in response to the same query issued by another client. Query cache based on *SQL_ID*.SELECT data into view is the best example of pre-cache data using query cache.

Key cache:
Cache table indexes. In MySQL keys are nothing but indexes (In oracle keys are constraints) if index size is small then it will cache index structure and data leaf. If an index is large, then it will only cache index structure.



Storage Engines

Storage engine is pluggable component that defines mysql server objects behavior on physical [storage] and logical [execution] levels. Storage engine responsible for SQL statement execution and fetching data from data files. Use as a plugin and can load/unload from running MySQL server. Few of them as following,

1. InnoDB :
   - Fully transactional ACID.
   - Offers REDO and UNDO for transactions.
   - Data storage in tablespace:

   – Multiple files [.frm is for object structure, .ibd is for data + index]
   – Logical object structure using InnoDB data and log buffer.

   - Row-level locking.
   - Tablespace storage mechanism.

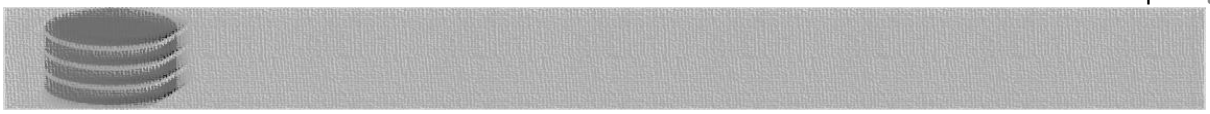
2. NDB (For MySQL Cluster):
   - Fully Transactional and ACID Storage engine.
   - Distribution execution of data and using multiple mysqld.
   - NDB use logical data with own buffer for each NDB engine.
   - Offers REDO and UNDO for transactions.
   - Row-level locking.
   - Horizontal scaling of data.

3. MyISAM:
   - Non-transactional storage engine.
   - No UNDO/REDO log support.
   - Speed for read.
   - Data storage in files and use key, metadata and query cache.

   – FRM for table structure
   – MYI for table index
   – MYD for table data

   - Table-level locking.

4. MEMORY:
   - Non-transactional storage engine.
   - All data stored in memory other than table metadata and structure.



- Table-level locking.

5. ARCHIVE:
   - Non-transactional storage engine,
   - Store large amounts of compressed and unindexed data.
   - Allow INSERT, REPLACE, and SELECT sql operations.
   - DELETE or UPDATE sql operations not permitted.
   - Table-level locking.

6. CSV:
   - Stores data in flat files using comma-separated values format.
   - Table structure need be created within MySQL server (.frm)

MySQL Connection:

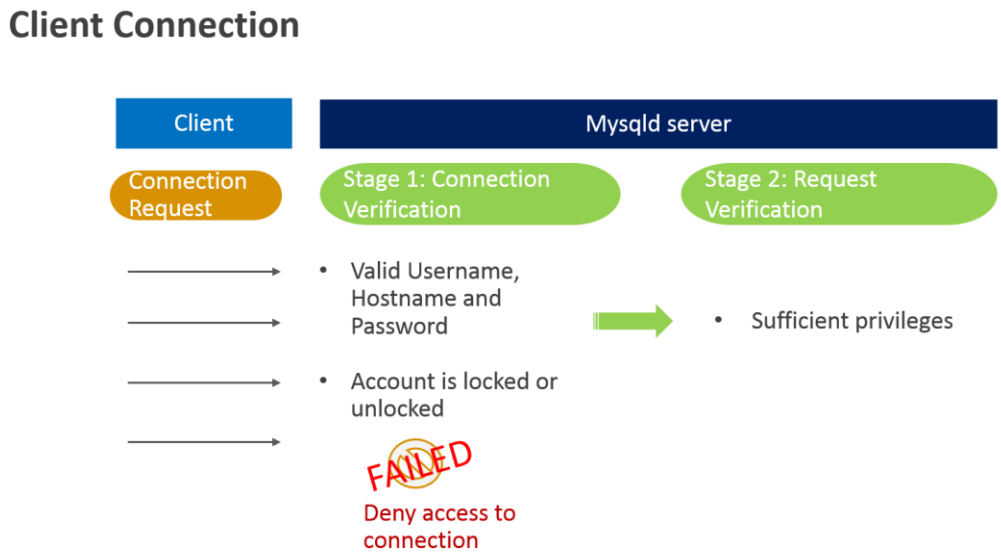


Figure 3: MySQL Client Connection

SQL execution

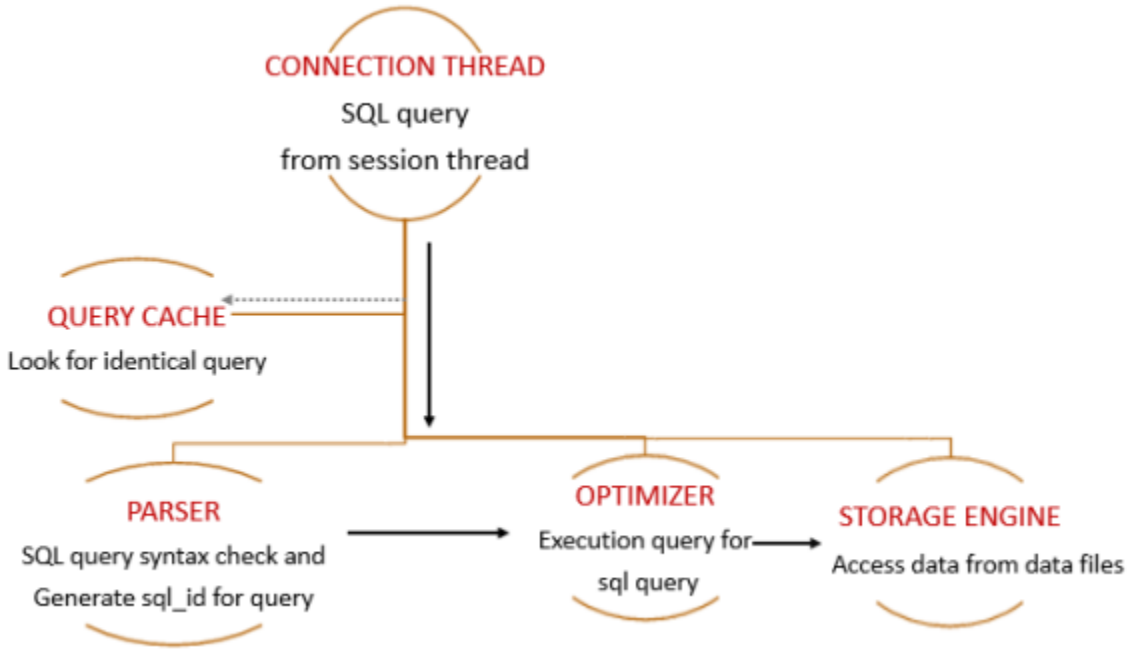


Figure 4: SQL Query execution

Other storage engines like InnoDB, NDB are having logical structure for data and they have their own data buffer. This buffer define on storage engine level.

About InnoDB Storage Engine

- Fully transactional ACID.
- Row-level locking.
- Offers REDO and UNDO for transactions.
- Data storage in tablespace:
  – Multiple data files
  – Logical object structure using InnoDB data and log buffer
- Use shared file to store objects [Data and Index in the same file]
- InnoDB data is 100% of a logical structure, data stored physically.
- InnoDB Read physical data and build logical structure[Blocks and Rows]
- Logical storage called as TABLESPACE.

InnoDB storage engine architecture

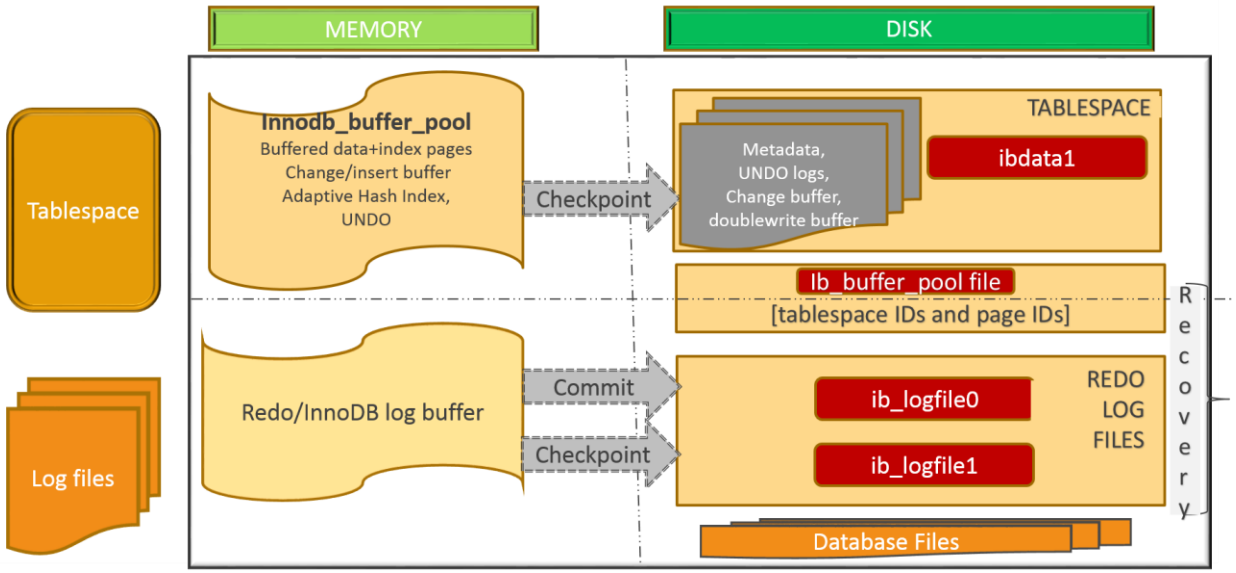


Figure 5: InnoDB Storage engine architecture

Tablespace:

Storage for InnoDB is divided into tablespaces. A tablespace is a logical structure associated with multiple data files (objects). Each tablespace contains pages (blocks), extents and segments.

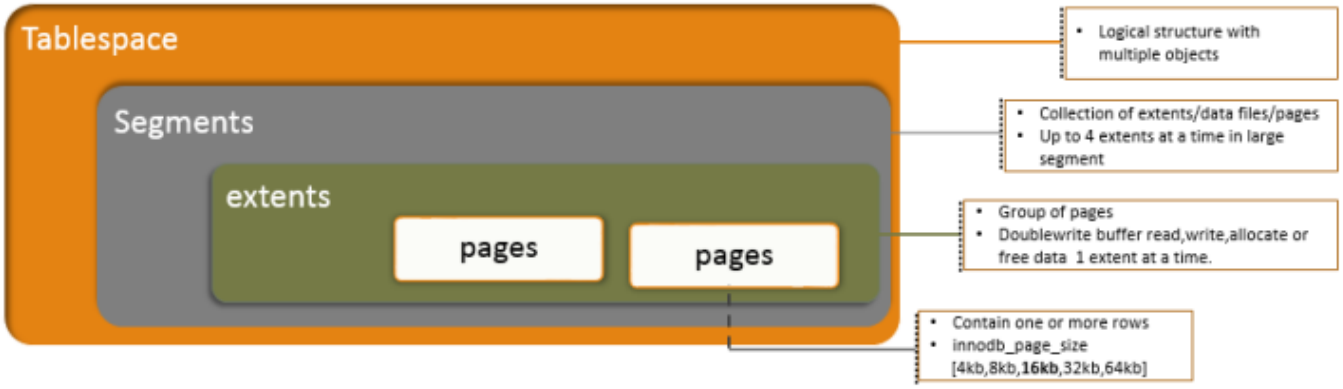


Figure 6: Tablespace overview

Pages: a smallest piece of data for InnoDB also called blocks. Default page size is 16kb and page can hold one or more rows depending on row size.
Available page sizes: 4kb, 8kb, 16kb, 32kb, 64kb
Variable name : innodb_page_size
Need to configure before initializing mysqld server.

Extents: It's a group of pages. For better I/O throughput InnoDB read or write a collection of pages i.e. one extent at a time.
For a group of pages with default page size 16kb, extent size up to 1mb.
Doublewrite buffer read/write/allocate or free data to one extent at a time.

Segments: Collection of files in InnoDB tablespace. Use up to 4 extents in a segment.

InnoDB **components**:

In Memory

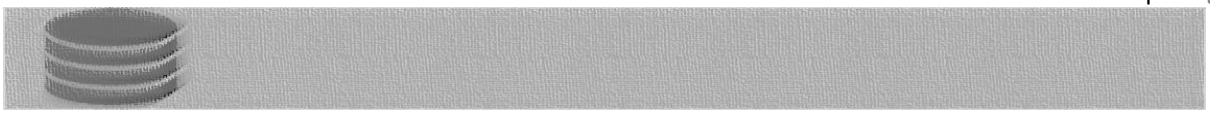
InnoDB buffer pool:
Central buffer for InnoDB storage engine. In this buffer, mysql load blocks and cache table index and data..
– The main memory where InnoDB cache table data and indexes.
– Size up to 80% of physical memory on dedicated DB server recommended.
– Shared buffer across all sessions.
– InnoDB use LRU (Least Recently Used) page replacement algorithm.
– Data that is being reused is always in the same memory.
– Data that does not use, will get phased out eventually.
Variable: `innodb_buffer_pool_size`

Change buffer:
In a memory change buffer is a part of InnoDB buffer pool and on disk, it is a part of system tablespace, so even after database restart index changes remain buffered. Change buffer is a special data structure that caches changes to secondary index pages when affected pages not in the buffer pool. Buffered Changes came from DML operation like INSERT, UPDATE, and DELETE etc.

Variable: `innodb_change_buffering`
Defined the type of changes keep in change buffer. The default is ‘all’ i.e Buffer inserts, delete-marking operations, and purges.

Adaptive Hash Index:
If a table fits almost entirely in main memory, a hash index can speed up queries by enabling direct lookup of any element, turning the index value into a sort of pointer. InnoDB has a mechanism that monitors index searches. If InnoDB notices that queries could benefit from building a hash index, it does so automatically.
Variable: `innodb_adaptive_hash_index`

Redo log buffer:
Buffer for redo logs, it hold data to be written to the redo log. Periodically data getting flushed from redo log buffer to redo logs. Flushing data from memory to disk managed by `innodb_flush_log_at_trx_commit` and `innodb_flush_log_at_timeout` configuration option.
– A large size of redo log buffer enables a large transaction to run without writing redo logs to disk before the transaction commit.
Variable: `innodb_log_buffer_size` (default 16M)



On Disk

System tablespace:
Apart from the table data storage, InnoDB's functionality requires looking for table metadata and storing and retrieving MVCC info to support ACID compliance and Transaction Isolation. It contains several types of information for InnoDB objects.

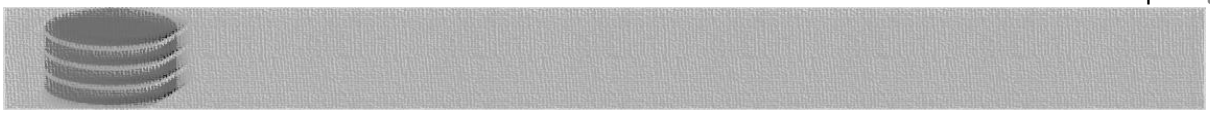
- Contains:
  Table Data Pages
  Table Index Pages
  Data Dictionary
  MVCC Control Data
  Undo Space
  Rollback Segments
  Double Write Buffer (Pages Written in the Background to avoid OS caching)
  Insert Buffer (Changes to Secondary Indexes)
- Variables:
  `innodb_data_file_path` = /ibdata/ibdata1:10M:autoextend

General tablespace:
Shared tablespace to store multiple table data. Introduce in MySQL 5.7.6. A user has to create this using CREATE TABLESPACE syntax. TABLESPACE option can be used with CREATE TABLE to create a table and ALTER TABLE to move a table in general table.
– Memory advantage over `innodb_file_per_table` storage method.
– Support both Antelope and Barracuda file formats.
– Supports all row formats and associated features.
– Possible to create outside data directory.

File-Per-Table Tablespaces:
It's a single-table tablespace that is created in its own data file in database directory rather than in the system tablespace.
By default table will created in separate data file (tablespace) when `innodb_file_per_table` option is enabled. Data file extension will be *.ibd*

Benefits:
File per-table tablespaces support DYNAMIC and COMPRESSED row formats which support features such as off-page storage for variable length data and table compression.

InnoDB data dictionary:
Storage area in system tablespace made up of internal system tables with metadata information for objects[tables, index, columns etc.]. It's a part of system table space.



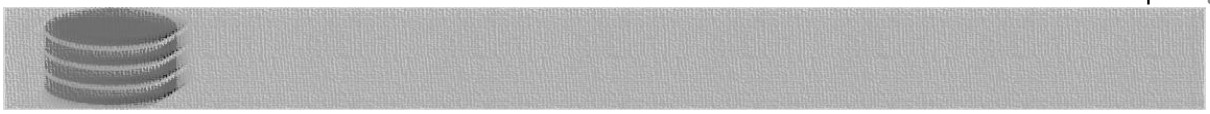
Double write buffer:
Storage area in system tablespace where InnoDB writes pages flushed from InnoDB buffer pool, before writing to their proper location in the data files.
In case mysqld process crash in the middle of a page writes, at the time of crash recovery InnoDB can find a good copy of the page from doublewrite buffer.

Even though it writing data twice there is no much I/O overhead, since Data getting written as a large sequential chunk, with a single fsync() call to the operating system.
Variable: `innodb_doublewrite` (default enable)

REDO logs:
Used during crash recovery. At the time of mysqld startup, InnoDB performs auto recovery to correct data written by incomplete transactions. Transactions that not finish updating data files before an unexpected mysqld shutdown are replayed automatically at the time of mysqld startup even before taking any connection. It uses LSN (Log Sequence Number) value.
Plenty of data changes cannot get written to disk quickly, so it will go under redo and then to the disk.
Why we need a redo for recovery?
Let's take an example, User changing data in innodb buffer and commit, somewhere it needs to go before writing into a disk. Because in the case of crash buffer data will lost, that's why we need redo logs.

– In redo, all changes will go with info like row_id, old column value, new column value, session_id and time.
– One commit complete data will get written to disk in a data file.
- InnoDB uses group commit functionality to group multiple such flush requests together to avoid one flush for each commit.
– Variables:
`innodb_log_files_in_group` = [# of redo file groups]
`innodb_log_file_size` = [Size for each redo file]

UNDO tablespace and logs:
UNDO tablespace contains one or more undo logs files.
UNDO manages consistent reads by keeping modified uncommitted data for active transaction [MVCC]. Unmodified data is retrieved from this storage area. Undo logs also called as rollback segments.
By default, UNDO logs are part of system tablespace, MySQL allows to store undo logs in separate UNDO tablespace/s [Introduce in MySQL 5.6]. Need to configure before initializing mysqld server.
– When we configure separate undo tablespace, the undo logs in the system tablespace become inactive.
– Need to configure before initializing mysqld server and cannot change after that.
– We truncate undo logs, but cannot drop.



– Variables:
innodb_undo_tablespaces: # of undo tablespaces, default 0
innodb_undo_directory: Location for undo tablespace, default is data_dir with 10MB size.
innodb_undo_logs: # of undo logs, default and max value is '128'

Temporary tablespace:
Storage to keep and retrieve modified uncommitted data for temporary tables and related objects.
Introduce in MySQL 5.7.2 and use for rollback temp table changes while a server is running.
– Undo logs for temporary tables reside in the temp tablespace.
– Default tablespace file ibtmp1 getting recreated on server startup.
– Not getting used to crash recovery.
– Advantage: Performance gain by avoiding redo logging IO for temp tables and related objects.
– Variable:
innodb_temp_data_file_path = ibtmp1:12M:autoextend (default)

ACID Model

Atomicity

The ***atomicity*** aspect of the ACID model mainly involves InnoDB transactions. Related MySQL features include:

- Autocommit setting.
- COMMIT statement.
- ROLLBACK statement.
- Operational data from the INFORMATION_SCHEMA tables.

Consistency

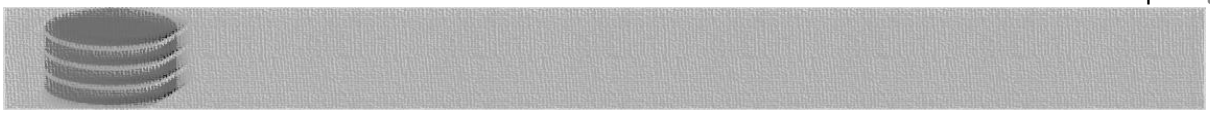
The ***consistency*** aspect of the ACID model mainly involves internal InnoDB processing to protect data from crashes. Related MySQL features include:

- InnoDB doublewrite buffer.
- InnoDB crash recovery.

Isolation

The ***isolation*** aspect of the ACID model mainly involves InnoDB transactions, in particular the isolation level that applies to each transaction. Related MySQL features include:

- Autocommit setting.
- SET ISOLATION LEVEL statement.
- The low-level details of InnoDB locking. During performance tuning, you see these details through INFORMATION_SCHEMA tables.



Durability

The ***durability*** aspect of the ACID model involves MySQL software features interacting with your particular hardware configuration. Because of the many possibilities depending on the capabilities of your CPU, network, and storage devices, this aspect is the most complicated to provide concrete guidelines for. (And those guidelines might take the form of buy "new hardware".) Related MySQL features include:

- InnoDB doublewrite buffer, turned on and off by the innodb_doublewrite configuration option.
- Configuration option innodb_flush_log_at_trx_commit.
- Configuration option sync_binlog.
- Configuration option innodb_file_per_table.

MySQL Transaction Isolation Levels

InnoDB offers all four transaction isolation levels described by the SQL:1992 standard: READ UNCOMMITTED, READ COMMITTED, REPEATABLE READ, and SERIALIZABLE. The default isolation level for InnoDB is REPEATABLE READ.

REPEATABLE READ
This is the default isolation level for InnoDB. Consistent reads within the same transaction read the snapshot established by the first read.

READ COMMITTED
Each consistent read, even within the same transaction, sets and reads its own fresh snapshot.

For locking reads (SELECT with FOR UPDATE or LOCK IN SHARE MODE), UPDATE statements, and DELETE statements, InnoDB locks only index records, not the gaps before them, and thus permits the free insertion of new records next to locked records.

If you use READ COMMITTED, you *must* use row-based binary logging.

READ UNCOMMITTED
SELECT statements are performed in a nonlocking fashion, but a possible earlier version of a row might be used. Thus, using this isolation level, such reads are not consistent. This is also called a dirty read. Otherwise, this isolation level works like READ COMMITTED.

SERIALIZABLE
This level is like REPEATABLE READ, but InnoDB implicitly converts all plain SELECT statements to SELECT ... LOCK IN SHARE MODE if autocommit is disabled. If autocommit is enabled, the SELECT is its own transaction. It therefore is known to be read only and can be serialized if performed as a consistent (nonlocking) read and need not block for other transactions.

END