

Chapter 14 MySQL Data Dictionary

**Table** **of** **Contents**

[14.1 Data Dictionary Schema](#_bookmark1) [3023](#_bookmark1)

[14.2 Removal File- Metadata Storage](#_bookmark2)ofbased [3024](#_bookmark2)

[14.3 Transactional Storage Dictionary Data](#_bookmark3)of [3025](#_bookmark3)

[14.4 Dictionary Object Cache](#_bookmark4) [3025](#_bookmark4)

[14.5 INFORMATION\_SCHEMA Data Dictionary Integration](#_bookmark5)and [3026](#_bookmark5)

14.6 Serialized Dictionary Information (SDI) 3028

14.7 Data Dictionary Usage Differences 3028

14.8 Data Dictionary Limitations 3030

MySQL Server incorporates a transactional data dictionary that stores information about database objects. In previous MySQL releases, dictionary data was stored in metadata files, nontransactional tables, and storage engine-specific data dictionaries.

This chapter describes the main features, benefits, usage differences, and limitations of the data dictionary. For other implications of the data dictionary feature, refer to the “Data Dictionary Notes”

section in the [MySQL 8.0 Release Notes](https://dev.mysql.com/doc/relnotes/mysql/8.0/en/).

Benefits of the MySQL data dictionary include:

• Simplicity of a centralized data dictionary schema that uniformly stores dictionary data. See [Section 14.1, “Data Dictionary Schema”](#_bookmark1) .

• Removal of file-based metadata storage. See [Section 14.2, “Removal of File-based Metadata](#_bookmark2) [Storage”](#_bookmark2) .

• Transactional, crash-safe storage of dictionary data. See [Section 14.3, “Transactional Storage of](#_bookmark3) [Dictionary Data”](#_bookmark3) .

• Uniform and centralized caching for dictionary objects. See [Section 14.4, “Dictionary Object Cache”](#_bookmark4) .

• A simpler and improved implementation for some INFORMATION\_SCHEMA tables. See [Section 14.5,](#_bookmark5) [“INFORMATION\_SCHEMA and Data Dictionary Integration”](#_bookmark5) .

• Atomic DDL. See Section 13.1.1, “Atomic Data Definition Statement Support” .

**Important**

A data dictionary-enabled server entails some general operational differences compared to a server that does not have a data dictionary; see Section 14.7, “Data Dictionary Usage Differences” . Also, for upgrades to MySQL 8.0, the upgrade procedure differs somewhat from previous MySQL releases and requires that you verify the upgrade readiness of your installation by checking specific prerequisites. For more information, see Section 2.10, “Upgrading MySQL” , particularly Section 2.10.5, “Preparing Your Installation for Upgrade” .

**14.1** **Data** **Dictionary** **Schema**

Data dictionary tables are protected and may only be accessed in debug builds of MySQL. However, MySQL supports access to data stored in data dictionary tables through INFORMATION\_SCHEMA tables and SHOW statements. For an overview of the tables that comprise the data dictionary, see Data Dictionary Tables.

MySQL system tables still exist in MySQL 8.0 and can be viewed by issuing a SHOW TABLES statement on the mysql system database. Generally, the difference between MySQL data dictionary



tables and system tables is that data dictionary tables contain metadata required to execute SQL queries, whereas system tables contain auxiliary data such as time zone and help information. MySQL system tables and data dictionary tables also differ in how they are upgraded. The MySQL server manages data dictionary upgrades. SQL server. See [How the Data Dictionary is Upgraded](#_bookmark6). Upgrading MySQL system tables requires running the full MySQL upgrade procedure. See Section 2.10.3, “What the MySQL Upgrade Process Upgrades” .

**How** **the** **Data** **Dictionary** **is** **Upgraded**

New versions of MySQL may include changes to data dictionary table definitions. Such changes are present in newly installed versions of MySQL, but when performing an in-place upgrade of MySQL binaries, changes are applied when the MySQL server is restarted using the new binaries. At startup, the data dictionary version of the server is compared to the version information stored in the data dictionary to determine if data dictionary tables should be upgraded. If an upgrade is necessary and supported, the server creates data dictionary tables with updated definitions, copies persisted metadata to the new tables, atomically replaces the old tables with the new ones, and reinitializes the data dictionary. If an upgrade is not necessary, startup continues without updating the data dictionary tables.

Upgrade of data dictionary tables is an atomic operation, which means that all of the data dictionary tables are upgraded as necessary or the operation fails. If the upgrade operation fails, server startup fails with an error. In this case, the old server binaries can be used with the old data directory to start the server. When the new server binaries are used again to start the server, the data dictionary upgrade is reattempted.

Generally, after data dictionary tables are successfully upgraded, it is not possible to restart the server using the old server binaries. As a result, downgrading MySQL server binaries to a previous MySQL version is not supported after data dictionary tables are upgraded.

The mysqld --no-dd-upgrade option can be used to prevent automatic upgrade of data dictionary tables at startup. When --no-dd-upgrade is specified, and the server finds that the data dictionary version of the server is different from the version stored in the data dictionary, startup fails with an error stating that the data dictionary upgrade is prohibited.

**Viewing** **Data** **Dictionary** **Tables** **Using** **a** **Debug** **Build** **of** **MySQL**

Data dictionary tables are protected by default but can be accessed by compiling MySQL with debugging support (using the -DWITH\_DEBUG=1 CMake option) and specifying the

+d,skip\_dd\_table\_access\_check debug option and modifier. For information about compiling debug builds, see Section 5.9. 1. 1, “Compiling MySQL for Debugging” .

**Warning**

Modifying or writing to data dictionary tables directly is not recommended and may render your MySQL instance inoperable.

After compiling MySQL with debugging support, use this SET statement to make data dictionary tables visible to the mysql client session:

mysql> **SET** **SESSION** **debug='+d,skip\_dd\_table\_access\_check';** Use this query to retrieve a list of data dictionary tables: mysql> **SELECT** **name,** **schema\_id,** **hidden,** **type** **FROM** **mysql.tables** **where** **schema\_id=1** **AND** **hidden='System';** Use SHOW CREATE TABLE to view data dictionary table definitions. For example: mysql> **SHOW** **CREATE** **TABLE** **mysql.catalogs\G**

**14.2** **Removal** **of** **File-based** **Metadata** **Storage**

In previous MySQL releases, dictionary data was partially stored in metadata files. Issues with file- based metadata storage included expensive file scans, susceptibility to file system-related bugs,

complex code for handling of replication and crash recovery failure states, and a lack of extensibility that made it difficult to add metadata for new features and relational objects.

The metadata files listed below are removed from MySQL. Unless otherwise noted, data previously stored in metadata files is now stored in data dictionary tables.

• .frm files: Table metadata files. With the removal of .frm files:

• The 64KB table definition size limit imposed by the .frm file structure is removed.

• The Information Schema TABLES table's VERSION column reports a hardcoded value of 10, which is the last .frm file version used in MySQL 5.7.

• .par files: Partition definition files. InnoDB stopped using partition definition files in MySQL 5.7 with the introduction of native partitioning support for InnoDB tables.

• .TRN files: Trigger namespace files.

• .TRG files: Trigger parameter files.

• .isl files: InnoDB Symbolic Link files containing the location of file-per-table tablespace files created outside of the data directory.

• db.opt files: Database configuration files. These files, one per database directory, contained database default character set attributes.

• ddl\_log.log file: The file contained records of metadata operations generated by data definition statements such as DROP TABLE and ALTER TABLE.

**14.3** **Transactional** **Storage** **of** **Dictionary** **Data**

The data dictionary schema stores dictionary data in transactional ( InnoDB) tables. Data dictionary tables are located in the mysql database together with non-data dictionary system tables.

Data dictionary tables are created in a single InnoDB tablespace named mysql.ibd, which resides in the MySQL data directory. The mysql.ibd tablespace file must reside in the MySQL data directory and its name cannot be modified or used by another tablespace.

Dictionary data is protected by the same commit, rollback, and crash-recovery capabilities that protect user data that is stored in InnoDB tables.

**14.4** **Dictionary** **Object** **Cache**

The dictionary object cache is a shared global cache that stores previously accessed data dictionary objects in memory to enable object reuse and minimize disk I/O. Similar to other cache mechanisms used by MySQL, the dictionary object cache uses an LRU-based eviction strategy to evict least recently used objects from memory.

The dictionary object cache comprises cache partitions that store different object types. Some cache partition size limits are configurable, whereas others are hardcoded.

• **tablespace** **definition** **cache** **partition**: Stores tablespace definition objects. The tablespace\_definition\_cache option sets a limit for the number of tablespace definition objects that can be stored in the dictionary object cache. The default value is 256.

• **schema** **definition** **cache** **partition**: Stores schema definition objects. The schema\_definition\_cache option sets a limit for the number of schema definition objects that can be stored in the dictionary object cache. The default value is 256.

• **table** **definition** **cache** **partition**: Stores table definition objects. The object limit is set to the value of max\_connections, which has a default value of 151.

The table definition cache partition exists in parallel with the table definition cache that is configured using the table\_definition\_cache configuration option. Both caches store table definitions but serve different parts of the MySQL server. Objects in one cache have no dependence on the existence of objects in the other.

• **stored** **program** **definition** **cache** **partition**: Stores stored program definition objects. The stored\_program\_definition\_cache option sets a limit for the number of stored program definition objects that can be stored in the dictionary object cache. The default value is 256.

The stored program definition cache partition exists in parallel with the stored procedure and stored function caches that are configured using the stored\_program\_cache option.

The stored\_program\_cache option sets a soft upper limit for the number of cached stored procedures or functions per connection, and the limit is checked each time a connection executes a stored procedure or function. The stored program definition cache partition, on the other hand, is a shared cache that stores stored program definition objects for other purposes. The existence of objects in the stored program definition cache partition has no dependence on the existence of objects in the stored procedure cache or stored function cache, and vice versa.

• **character** **set** **definition** **cache** **partition**: Stores character set definition objects and has a hardcoded object limit of 256.

• **collation** **definition** **cache** **partition**: Stores collation definition objects and has a hardcoded object limit of 256.

For information about valid values for dictionary object cache configuration options, refer to Section 5.1.8, “Server System Variables” .

**14.5** **INFORMATION\_SCHEMA** **and** **Data** **Dictionary** **Integration**

With the introduction of the data dictionary, the following INFORMATION\_SCHEMA tables are implemented as views on data dictionary tables:

• CHARACTER\_SETS

• CHECK\_CONSTRAINTS

• COLLATIONS

• COLLATION\_CHARACTER\_SET\_APPLICABILITY

• COLUMNS

• COLUMN\_STATISTICS

• EVENTS

• FILES

• INNODB\_COLUMNS

• INNODB\_DATAFILES

• INNODB\_FIELDS

• INNODB\_FOREIGN

• INNODB\_FOREIGN\_COLS

• INNODB\_INDEXES

• INNODB\_TABLES

• INNODB\_TABLESPACES

• INNODB\_TABLESPACES\_BRIEF

• INNODB\_TABLESTATS

• KEY\_COLUMN\_USAGE

• KEYWORDS

• PARAMETERS

• PARTITIONS

• REFERENTIAL\_CONSTRAINTS

• RESOURCE\_GROUPS

• ROUTINES

• SCHEMATA

• STATISTICS

• ST\_GEOMETRY\_COLUMNS

• ST\_SPATIAL\_REFERENCE\_SYSTEMS

• TABLES

• TABLE\_CONSTRAINTS

• TRIGGERS

• VIEWS

• VIEW\_ROUTINE\_USAGE

• VIEW\_TABLE\_USAGE

Queries on those tables are now more efficient because they obtain information from data dictionary tables rather than by other, slower means. In particular, for each INFORMATION\_SCHEMA table that is a view on data dictionary tables:

• The server no longer must create a temporary table for each query of the INFORMATION\_SCHEMA table.

• When the underlying data dictionary tables store values previously obtained by directory scans (for example, to enumerate database names or table names within databases) or file-opening operations (for example, to read information from .frm files), INFORMATION\_SCHEMA queries for those values now use table lookups instead. (Additionally, even for a non-view INFORMATION\_SCHEMA table, values such as database and table names are retrieved by lookups from the data dictionary and do not require directory or file scans.)

• Indexes on the underlying data dictionary tables permit the optimizer to construct efficient query execution plans, something not true for the previous implementation that processed the INFORMATION\_SCHEMA table using a temporary table per query.

The preceding improvements also apply to SHOW statements that display information corresponding to the INFORMATION\_SCHEMA tables that are views on data dictionary tables. For example, SHOW DATABASES displays the same information as the SCHEMATA table.

In addition to the introduction of views on data dictionary tables, table statistics contained in the STATISTICS and TABLES tables is now cached to improve INFORMATION\_SCHEMA query performance. The information\_schema\_stats\_expiry system variable defines the period of time before cached table statistics expire. The default is 86400 seconds (24 hours). If there are no cached statistics or statistics have expired, statistics are retrieved from storage engine when querying table statistics columns. To update cached values at any time for a given table, use ANALYZE TABLE