# 第5章

# information_schema数据库

MySQL 自带三个系统数据库用于监控 MySQL，这三个数据库分别从不同的角度观察 MySQL 数据库。

（1）information_schema 数据库主要保存 MySQL 的静态元数据。

（2）performance_schema 数据库主要保存与 MySQL 性能相关的动态数据。

（3）sys 数据库是建立在前两个数据库基础上的数据库，它包括一些视图、存储过程和函数，用于方便 DBA（database administrator，数据库管理员）、开发人员和操作人员的日常工作。

本章介绍 information_schema 数据库的相关知识，第 6、7 章将分别介绍 performance_schema 数据库和 sys 数据库。

## 5.1 数据组成

系统数据库 information_schema 中保存着 MySQL 的静态元数据。它和系统数据库 performance_schema 不同，performance_schema 主要保存动态元数据，而 information_schema 主要保存静态元数据，也有少量的动态元数据。information_schema 实际上是一个虚拟数据库，它并不包含任何实际的数据，它是唯一一个在操作系统的文件系统上没有对应目录的数据库，因此该数据库中的表只能被查询，不能被修改。如下 SQL 语句可查询出 information_schema 中表类型和数量：

```
mysql > select table_type, count( * ) from information_schema.tables where table_schema =
'information_schema' group by table_type;
+--------------+-----------+
| TABLE_TYPE   | count( * ) |
+--------------+-----------+
| SYSTEM VIEW  |        79 |
+--------------+-----------+
```

```
1 row in set (0.01 sec)
```

可以看到，在 MySQL 8.0 中 information_schema 数据库共有 79 个表，这些表都是同一个类型 SYSTEM VIEW，严格地说它们都是视图，并不是表，information_schema 中并没有 BASE TABLE 类型的表。

### 5.1.1 静态元数据

information_schema 中的静态元数据来源于数据字典，但为了保护 MySQL 的内部数据和对 MySQL 后续版本的兼容，用户并不能直接查询数据字典，例如，表 mysql.schemata 保存着数据库的信息，运行如下代码直接查询该表会被拒绝：

```
mysql>  select * from mysql.schemata;
ERROR 3554 (HY000): Access to data dictionary table 'mysql.schemata' is rejected.
```

但可以通过查询 information_schema 中的 schemata 视图从而得到同样的信息，代码如下：

```
mysql> select schema_name from information_schema.schemata;
+--------------------+
| SCHEMA_NAME        |
+--------------------+
| mysql              |
| information_schema |
| performance_schema |
| sys                |
| sbtest             |
| mydb               |
| tpcc1000           |
| sakila             |
| percona_schema     |
+--------------------+
9 rows in set (0.00 sec)
```

如上查询语句和 show databases 的效果一样，代码如下：

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mydb               |
| mysql              |
| percona_schema     |
| performance_schema |
| sakila             |
| sbtest             |
| sys                |
| tpcc1000           |
+--------------------+
9 rows in set (0.00 sec)
```

查看 information_schema.schemata 的定义如下：

```
mysql > show create table information_schema.schemata \G
*************************** 1. row ***************************
                View: SCHEMATA
         Create View: CREATE ALGORITHM = UNDEFINED
DEFINER = 'mysql.infoschema'@'localhost' SQL SECURITY DEFINER VIEW
'information_schema'.'SCHEMATA' AS
select 'cat'.'name' AS
'CATALOG_NAME','sch'.'name' AS 'SCHEMA_NAME','cs'.'name' AS 'DEFAULT_CHARACTER_SET_NAME','col'.
'name' AS 'DEFAULT_COLLATION_NAME',NULL
AS 'SQL_PATH','sch'.'default_encryption' AS 'DEFAULT_ENCRYPTION' from
((('mysql'.'schemata' 'sch' join 'mysql'.'catalogs' 'cat' on(('cat'.'id' =
'sch'.'catalog_id'))) join 'mysql'.'collations' 'col'
on(('sch'.'default_collation_id' = 'col'.'id'))) join
'mysql'.'character_sets' 'cs' on(('col'.'character_set_id' = 'cs'.'id')))
where (0 <> can_access_database('sch'.'name'))
character_set_client: utf8
collation_connection: utf8_general_ci
1 row in set (0.00 sec)
```

上面的 SQL 语句格式比较乱，把视图定义部分的 SQL 语句放到 MySQL 的图形工具 workbench 中进行格式整理，整理后的 SQL 语句如下：

```
SELECT
    'cat'.'name' AS 'CATALOG_NAME',
    'sch'.'name' AS 'SCHEMA_NAME',
    'cs'.'name' AS 'DEFAULT_CHARACTER_SET_NAME',
    'col'.'name' AS 'DEFAULT_COLLATION_NAME',
    NULL AS 'SQL_PATH',
    'sch'.'default_encryption' AS 'DEFAULT_ENCRYPTION'
FROM
    ((('mysql'.'schemata' 'sch'
    JOIN 'mysql'.'catalogs' 'cat' ON (('cat'.'id' = 'sch'.'catalog_id')))
    JOIN 'mysql'.'collations' 'col' ON (('sch'.'default_collation_id' = 'col'.'id')))
    JOIN 'mysql'.'character_sets' 'cs' ON (('col'.'character_set_id' = 'cs'.'id')))
WHERE
    (0 <> CAN_ACCESS_DATABASE('sch'.'name'))
```

可以看到表 information scchema.schemata 实际上是建立在 mysql.schemata 等表上的视图。

### 5.1.2 动态元数据

information_schema 数据库中除了静态元数据外，还有少量的动态元数据，这些动态元数据来源于数据库的存储引擎，如 innodb_metrics 表和统计信息表。与统计信息相关的字段如下：

```
STATISTICS.CARDINALITY
```

```
TABLES.AUTO_INCREMENT
TABLES.AVG_ROW_LENGTH
TABLES.CHECKSUM
TABLES.CHECK_TIME
TABLES.CREATE_TIME
TABLES.DATA_FREE
TABLES.DATA_LENGTH
TABLES.INDEX_LENGTH
TABLES.MAX_DATA_LENGTH
TABLES.TABLE_ROWS
TABLES.UPDATE_TIME
```

这些字段保存着与表和索引相关的动态统计信息，这些信息从存储引擎刷新到数据字典表 mysql.index_stats 和 mysql.table_stats 中（这两个表用户不能直接访问），为了提高查询的效率，这两个表的信息都存放在缓存中，缓存刷新的时间由系统参数 information_schema_stats_expiry 决定，该参数的默认值是 86400 秒，也就是 24 小时。因此有时会出现查询到的统计信息较陈旧的情况，如果要查询实时的统计信息，可以把 information_schema_stats_expiry 设置为 0，这样每次查询的统计信息都来自存储引擎，这也是 MySQL 5.7 的做法。下面看一个例子。

首先查询表 sakila.actor 的统计信息 auto_increment、update_time 和 table_rows，相关代码及其运行结果如下：

```
mysql > select auto_increment, update_time, table_rows from information_schema.tables where
table_schema = 'sakila' and table_name = 'actor';
+----------------+---------------------+------------+
| AUTO_INCREMENT | UPDATE_TIME         | TABLE_ROWS |
+----------------+---------------------+------------+
|            201 | 2021-07-20 17:46:56 |        200 |
+----------------+---------------------+------------+
1 row in set (0.04 sec)
```

使用 show table status 命令查询的结果和从 information_schema.tables 表里查询的结果一样，代码如下：

```
mysql > show table status like 'actor'\G
*************************** 1. row ***************************
           Name: actor
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: 200
 Avg_row_length: 81
    Data_length: 16384
Max_data_length: 0
   Index_length: 16384
      Data_free: 0
 Auto_increment: 201
    Create_time: 2021-07-20 17:46:43
    Update_time: 2021-07-20 17:46:56
```

```
     Check_time: NULL
      Collation: utf8mb4_0900_ai_ci
       Checksum: NULL
 Create_options:
        Comment:
1 row in set (0.01 sec)
```

然后向表 sakila.actor 中插入一条记录，命令如下：

```
mysql > insert into sakila.actor(first_name,last_name) values('si','Li');
Query OK, 1 row affected (0.00 sec)
```

第二次查询表 sakila.actor 的统计信息，相关代码及其运行结果如下：

```
mysql > select auto_increment,update_time,table_rows from information_schema.tables where
table_schema = 'sakila' and table_name = 'actor';
+----------------+---------------------+------------+
| AUTO_INCREMENT | UPDATE_TIME         | TABLE_ROWS |
+----------------+---------------------+------------+
|            201 | 2021-07-20 17:46:56 |        200 |
+----------------+---------------------+------------+
1 row in set (0.00 sec)
```

从以上运行结果可以看出没有变化，再使用 show table status 查询，代码如下：

```
mysql > show table status like 'actor'\G
*************************** 1. row ***************************
           Name: actor
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: 200
 Avg_row_length: 81
    Data_length: 16384
Max_data_length: 0
   Index_length: 16384
      Data_free: 0
Auto_increment: 201
    Create_time: 2021-07-20 17:46:43
    Update_time: 2021-07-20 17:46:56
     Check_time: NULL
      Collation: utf8mb4_0900_ai_ci
       Checksum: NULL
 Create_options:
        Comment:
1 row in set (0.01 sec)
```

结果还是没有变化。现在把 information_schema_stats_expiry 设置为 0 后，第三次查询表 sakila.actor 的统计信息，代码如下：

```
mysql > set session information_schema_stats_expiry = 0;
Query OK, 0 rows affected (0.00 sec)
```

```
mysql > select auto_increment, update_time, table_rows from information_schema.tables where
table_schema = 'sakila' and table_name = 'actor';
+----------------+---------------------+------------+
| AUTO_INCREMENT | UPDATE_TIME         | TABLE_ROWS |
+----------------+---------------------+------------+
|            202 | 2021-07-20 17:47:34 |        201 |
+----------------+---------------------+------------+
1 row in set (0.00 sec)

mysql > show table status like 'actor'\G
*************************** 1. row ***************************
           Name: actor
         Engine: InnoDB
        Version: 10
     Row_format: Dynamic
           Rows: 201
 Avg_row_length: 81
    Data_length: 16384
Max_data_length: 0
   Index_length: 16384
      Data_free: 0
 Auto_increment: 202
    Create_time: 2021-07-20 17:46:43
    Update_time: 2021-07-20 17:47:34
     Check_time: NULL
      Collation: utf8mb4_0900_ai_ci
       Checksum: NULL
 Create_options:
        Comment:
1 row in set (0.00 sec)
```

可以发现，actor 表的统计信息已经更新到当前值了。该实验展示了系统参数 information_schema_stats_expiry 是如何影响查询动态统计信息结果的。另外，执行 analyze table 命令后，会立即更新 mysql.index_stats 和 mysql.table_stats 的缓存。

## 5.2 MySQL 8.0 中的优化

information_schema 从 MySQL 5.0 开始使用，长期以来用户一直抱怨该数据库的查询效率低。但这个问题在 MySQL 8.0 中彻底得到了解决。这得益于在 MySQL 8.0 中，数据字典中的数据被迁移到了事务性的表中。

### 5.2.1 数据字典的优化

在 MySQL 8.0 之前，数据字典的数据存放在文件系统中，包括如下这些类型文件。

- .frm 文件：表定义文件。
- .par 文件：分区定义文件。

- .trn 文件：触发器命名空间文件。
- .trg 文件：触发器参数文件。
- .isl 文件：指向不在 datadir 目录的表空间文件的链接。
- db.opt 文件：数据库配置文件。

这些文件在 MySQL 8.0 中都被取消了，文件中的数据全部放入系统数据库 mysql 中，这些表的存储引擎都是 InnoDB，表空间是 mysql.ibd，位于 datadir 目录下。

这样可以利用 InnoDB 的事务特性实现 DDL(data definition language，数据定义语言)的原子化，如下面的 DDL 语句：

```
mysql > drop table tab_exit,tab_not_exist;
ERROR 1051 (42S02): Unknown table 'sakila.tab_not_exist'
```

在 MySQL 8.0 中，第二个表删除失败，drop 语句作为一个事务失败，第一个表不会被删除；但在 MySQL 5.7 中，第二个表删除失败时，第一个表已经被删除了。

### 5.2.2 查询方式的变化

MySQL 8.0 数据字典的数据存放到事务性的表中带来了很多好处，如 DDL 语句的原子化、不受文件系统 Bug 的影响、崩溃恢复等，但最大的好处是解决了长期以来困扰用户的查询性能的问题。对于大型的数据库，可对数据字典的查询性能提高数十倍甚至数百倍。因为在 MySQL 8.0 中，数据字典的数据存放到表中，对它们进行查询时，可以充分利用 SQL 语句的特性，如索引、连接等。而在 MySQL 8.0 之前，MySQL 需要先从文件中读取数据，然后结合存储引擎里的信息组成临时表，再对临时表进行查询，这个过程效率很低。下面先看一个在 MySQL 5.7 中查询数据字典语句的执行计划：

```
mysql > explain select table_name from information_schema.tables where table_schema like
'sakila%' and table_name like 'ac%'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: tables
   partitions: NULL
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: Using where; Skip_open_table; Scanned all databases
1 row in set, 1 warning (0.00 sec)
```

从 type 字段中的 ALL 和 Extra 字段中的 Scanned all databases 可以看到，该执行计划需要对所有数据库和表定义文件进行扫描，这个成本非常高，再看看同样的查询在 MySQL 8.0 中的执行计划：

```
mysql >  explain select table_name from information_schema.tables where table_schema like
'sakila%' and table_name like 'ac%'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: cat
   partitions: NULL
         type: index
possible_keys: PRIMARY
          key: name
      key_len: 194
          ref: NULL
         rows: 1
     filtered: 100.00
        Extra: Using index
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: sch
   partitions: NULL
         type: ref
possible_keys: PRIMARY,catalog_id
          key: catalog_id
      key_len: 8
          ref: mysql.cat.id
         rows: 10
     filtered: 11.11
        Extra: Using where; Using index
*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: tbl
   partitions: NULL
         type: ref
possible_keys: schema_id
          key: schema_id
      key_len: 8
          ref: mysql.sch.id
         rows: 50
     filtered: 11.11
        Extra: Using where
*************************** 4. row ***************************
           id: 1
  select_type: SIMPLE
        table: col
   partitions: NULL
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 8
          ref: mysql.tbl.collation_id
```

```
          rows: 1
      filtered: 100.00
         Extra: Using index
*************************** 5. row ***************************
           id: 1
  select_type: SIMPLE
        table: ts
   partitions: NULL
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 8
          ref: mysql.tbl.tablespace_id
         rows: 1
     filtered: 100.00
        Extra: Using index
*************************** 6. row ***************************
           id: 1
  select_type: SIMPLE
        table: stat
   partitions: NULL
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 388
          ref: mysql.sch.name,mysql.tbl.name
         rows: 1
     filtered: 100.00
        Extra: Using index
6 rows in set, 1 warning (0.01 sec)
```

可以看到，在 MySQL 8.0 中，查询计划充分利用了索引和连接，对不需要数据的过滤率非常高。此时再使用 show warnings 命令看看被优化器重写了的 SQL 语句，如下所示：

```
mysql > show warnings\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select 'tbl'.'name' AS 'TABLE_NAME' from 'mysql'.'tables' 'tbl' join
'mysql'.'schemata' 'sch' join 'mysql'.'catalogs' 'cat' left join 'mysql'.'collations' 'col' on(('col'.'id' =
'tbl'.'collation_id')) left join 'mysql'.'tablespaces' 'ts' on(('ts'.'id' = 'tbl'.'tablespace_id'))
left join 'mysql'.'table_stats' 'stat' on((('stat'.'schema_name' = 'sch'.'name') and ('stat'.'table_
name' = 'tbl'.'name'))) where (('tbl'.'schema_id' = 'sch'.'id') and ('sch'.'catalog_id' = 'cat'.
'id') and ('sch'.'name' like 'sakila%') and ('tbl'.'name' like 'ac%') and (0 <> can_access_table
('sch'.'name','tbl'.'name')) and (0 <> is_visible_dd_object('tbl'.'hidden')))
1 row in set (0.00 sec)
```

可以看到，实际上是使用标准的 SQL 语句来查询 mysql 数据库中的表。通过这两个执行计划的对比可以看出，在 MySQL 8.0 中对数据字典的查询效率有了质的飞跃。

## 5.3 权限

用户对 information_schema 进行查询时，需要有相应的权限才能查询对应的对象。例如，对 information_schema 中的 tables 进行查询时，只能查询到用户有查询权限的表，如一个只有 usage 权限的用户查询 tables 表时，除了 information_schema 的表外，其他表都看不到。例如，如下代码查询当前用户的权限：

```
mysql > show grants;
+-----------------------------------+
| Grants for lisi@ %                |
+-----------------------------------+
| GRANT USAGE ON * . * TO 'lisi'@'% '        |
+-----------------------------------+
1 row in set (0.00 sec)
```

可以发现，当前用户只有 usage 的权限，使用该用户查询 information_schema. tables 中的表：

```
mysql > select count( * ) from information_schema. tables where table_schema <>'information_
schema';
+-----------+
| count( * )   |
+-----------+
|         0   |
+-----------+
1 row in set (0.00 sec)
```

可以发现，除了 information_schema 数据库中的表之外，一个表也查不到。现在把 sakila. actor 表的查询权限赋予只有 usage 权限的用户，命令如下：

```
mysql > grant select on sakila. actor to lisi;
```

再次查询 information_schema. tables 表，就可以看到 sakila. actor 表了，相关命令及其运行结果如下：

```
mysql > select table_schema,table_name from  information_schema. tables where table_schema <>
'information_schema';
+---------------+-------------+
| TABLE_SCHEMA      | TABLE_NAME     |
+---------------+-------------+
| sakila            | actor          |
+---------------+-------------+
1 row in set (0.00 sec)
```

管理类的权限也一样，例如，对 InnoDB 表（以 innodb_开头的表）进行查询时，需要 PROCESS 权限，如果没有，会遇到下面的情况：

```
mysql >  select * from information_schema. innodb_tables;
ERROR 1227 (42000): Access denied; you need (at least one of) the PROCESS privilege(s) for this
operation
```

对 processlist 表进行查询时，如果没有 PROCESS 权限，只能看到当前用户的线程，其他用户的连接都看不到，相关命令及其运行结果如下：

```
mysql > select id,user,host from information_schema.processlist;
+-------+------+-----------+
| id    | user | host      |
+-------+------+-----------+
| 41264 | lisi | localhost |
+-------+------+-----------+
1 row in set (0.01 sec)
```

使用 show 语句进行查询时，对权限的要求和查询 information_schema 数据库的要求一样。

## 5.4 视图说明

按照 information_schema 中视图保存的信息，可以将这些视图分为四类，分别为数据库视图、实例视图、性能视图和权限视图。下面分别说明。

### 5.4.1 数据库视图

系统数据库 information_schema 中用于保存与数据库相关的视图占大部分，简单说明如下。

- CHECK_CONSTRAINTS：该视图在 MySQL 8.0.16 版本之后才有，它保存着表中定义的与 CHECK 限制相关的信息。
- COLUMNS：与表中字段相关的信息。
- COLUMN_STATISTICS：字段的直方图统计信息。
- EVENTS：定时任务(EVENT)的信息。
- FILES：表空间和对应文件的信息。
- KEY_COLUMN_USAGE：键字段相关的信息，包括主键、外键、唯一索引键相关的信息。
- INNODB_COLUMNS：InnoDB 表中字段的元信息。
- INNODB_DATAFILES：InnoDB 表空间 ID 和对应的文件。
- INNODB_FIELDS：InnoDB 索引对应的字段。
- INNODB_FOREIGN：InnoDB 表的外键信息。
- INNODB_FOREIGN_COLS：与 InnoDB 表外键相关的字段。
- INNODB_FT_CONFIG：InnoDB 表的全文索引的配置信息，查询该表时，先设置系统参数 innodb_ft_aux_table 指向要查询的包含全文索引的表，例如：

```
mysql > set global innodb_ft_aux_table = 'sakila/film_text';
Query OK, 0 rows affected (0.00 sec)

mysql > select *  from information_schema.innodb_ft_config;
```

```
+----------------------------+-------+
| KEY                        | VALUE |
+----------------------------+-------+
| optimize_checkpoint_limit  | 180   |
| synced_doc_id              | 0     |
| stopword_table_name        |       |
| use_stopword               | 1     |
+----------------------------+-------+
4 rows in set (0.18 sec)
```

- INNODB_FT_DELETED：保存 InnoDB 表的全文索引中被删除的行，为了提高性能，被删除的行并不是立刻从表中删除，而是保存到这个表中，当执行 optimize table 命令时才会被删除，查询该表之前也要先设置系统参数 innodb_ft_aux_table 指向要查的表。
- INNODB_FT_BEING_DELETED：当执行 optimize table 命令时，该表是 INNODB_FT_DELETED 表的一个快照。
- INNODB_FT_INDEX_CACHE：保存新插入的 InnoDB 表全文索引的行，为了提高性能，新插入的行会先保存到该表中，查询该表之前也要先设置系统参数 innodb_ft_aux_table 指向要查的表。
- INNODB_FT_INDEX_TABLE：InnoDB 表的倒序全文索引，查询该表之前也要先设置系统参数 innodb_ft_aux_table 指向要查的表。
- INNODB_INDEXES：InnoDB 表的索引信息。
- INNODB_TABLES：InnoDB 表的信息。
- INNODB_TABLESPACES：InnoDB 表空间的信息。
- INNODB_TABLESPACES_BRIEF：该视图的信息一部分来自 INNODB_TABLESPACES，文件名和路径来自 INNODB_DATAFILES。
- INNODB_TABLESTATS：InnoDB 表的统计信息。
- INNODB_TEMP_TABLE_INFO：用户创建的 InnoDB 临时表的信息，不包括优化器创建的临时表。
- INNODB_VIRTUAL：InnoDB 表的虚拟字段信息。
- PARAMETERS：存储过程和存储函数的参数，以及存储函数的返回值。
- PARTITIONS：表分区信息。
- REFERENTIAL_CONSTRAINTS：外键信息。
- ROUTINES：存储过程和存储函数的信息。
- SCHEMATA：数据库信息，SCHEMATA 是 SCHEMA 复数的一种写法，另外一种更流行的写法是 SCHEMAS。
- ST_GEOMETRY_COLUMNS：存储空间数据的字段信息。
- STATISTICS：索引定义和统计信息。
- TABLE_CONSTRAINTS：表约束的汇总信息，包括主键、唯一索引、外键、约束等。
- TABLES：表和视图的信息。如下 SQL 语句为统计实例中数据量最大的 10 个数据库的大小，这里是一个实验库，只展示两个应用的数据库：

```
mysql > select table_schema, count( * ) tables,
concat(round(sum(table_rows)/1000000,2),'m') table_rows, concat(round(sum(data_length)/
(1024 * 1024 * 1024),2),'g') data,
concat(round(sum(index_length)/(1024 * 1024 * 1024),2),'g') idx,
concat(round(sum(data_length + index_length)/(1024 * 1024 * 1024),2),'g') total_size
from information_schema.tables
where table_schema not in ("information_schema", "mysql", "performance_schema", "sys")
group by table_schema
order by sum(data_length + index_length) desc limit 10;
+---------------+--------+------------+-------+-------+------------+
| TABLE_SCHEMA  | tables | table_rows | data  | idx   | total_size |
+---------------+--------+------------+-------+-------+------------+
| tpcc1000      |      9 | 5.32m      | 0.82g | 0.18g | 1.00g      |
| sbtest        |      3 | 0.02m      | 0.00g | 0.00g | 0.01g      |
+---------------+--------+------------+-------+-------+------------+
2 rows in set (0.00 sec)
```

如下 SQL 语句统计某库下最大的 10 个表的大小：

```
mysql >  select table_schema,table_name table_name,
concat(round(data_length / (1024 * 1024), 2),'m') data_length,
concat(round(index_length / (1024 * 1024), 2),'m') index_length,
concat(round(round(data_length + index_length) / (1024 * 1024),2),'m') total_size
from information_schema.tables
where table_schema = 'sakila'
order by (data_length + index_length) desc limit 10;
+--------------+---------------+-------------+--------------+----------+
| TABLE_SCHEMA | table_name    | data_length | index_length | total_size |
+--------------+---------------+-------------+--------------+----------+
| sakila       | rental        | 1.52M       | 1.14M        | 2.66M    |
| sakila       | payment       | 1.52M       | 0.61M        | 2.13M    |
| sakila       | inventory     | 0.17M       | 0.19M        | 0.36M    |
| sakila       | film          | 0.19M       | 0.08M        | 0.27M    |
| sakila       | film_actor    | 0.19M       | 0.08M        | 0.27M    |
| sakila       | film_text     | 0.17M       | 0.02M        | 0.19M    |
| sakila       | customer      | 0.08M       | 0.05M        | 0.13M    |
| sakila       | address       | 0.09M       | 0.02M        | 0.11M    |
| sakila       | staff         | 0.06M       | 0.03M        | 0.09M    |
| sakila       | film_category | 0.06M       | 0.02M        | 0.08M    |
+--------------+---------------+-------------+--------------+----------+
10 rows in set (0.02 sec)
```

如下 SQL 语句找出数据库中所有不是 InnoDB 引擎的表：

```
mysql >  select table_schema, table_name,engine
from information_schema.tables
where engine != 'innodb' and table_schema not in ('information_schema','sys','mysql','performance
_schema');
+--------------+------------+--------+
| TABLE_SCHEMA | TABLE_NAME | ENGINE |
+--------------+------------+--------+
| test         | test1      | MyISAM |
```

```
| test              | tba             | MyISAM    |
+-------------------+-----------------+-----------+
2 rows in set (0.01 sec)
```

如下 SQL 语句找出没有主键或唯一索引的表：

```
mysql > select t1.table_schema,
t1.table_name
from information_schema.columns t1
join information_schema.tables t2
on t1.table_schema = t2.table_schema
and t1.table_name = t2.table_name
where t1.table_schema not in
('sys', 'mysql', 'information_schema', 'performance_schema')
and t2.table_type = 'BASE TABLE'
group by t1.table_schema, t1.table_name
having group_concat(column_key) not regexp 'PRI|UNI';
```

- TABLESPACES：该视图没有用，在将来的版本中将会被删除。
- TRIGGERS：触发器定义信息。
- VIEW_ROUTINE_USAGE：该视图从 MySQL 8.0.13 版本之后才有，它保存着视图定义里的存储函数的信息。
- VIEW_TABLE_USAGE：该视图从 MySQL 8.0.16 版本之后才有，它保存着视图定义里的表和视图的信息。
- VIEWS：视图的定义。

### 5.4.2 实例视图

实例视图保存与实例相关的信息，分别说明如下。

- CHARACTER_SETS：可用的字符集。
- COLLATIONS：每个字符集的排序规则。
- COLLATION_CHARACTER_SET_APPLICABILITY：排序规则和字符集的对应关系，该视图和 COLLATIONS 视图的前两个字段一样。
- ENGINES：存储引擎的相关说明和加载状态。如下 SQL 语句查询存储引擎的分布：

```
mysql > select table_schema,engine,count( * )
from information_schema.tables
where table_schema not in  ('sys','mysql','information_schema','performance_schema')
and table_type = 'BASE TABLE'
group by table_schema,engine;
+---------------+---------+-----------+
| TABLE_SCHEMA  | ENGINE  | count( * ) |
+---------------+---------+-----------+
| test          | MyISAM  |         2 |
| sakila        | InnoDB  |        20 |
| sbtest        | InnoDB  |         2 |
| mysqlslap     | InnoDB  |         1 |
```

```
+--------------+--------+----------+
4 rows in set (0.01 sec)
```

- INNODB_FT_DEFAULT_STOPWORD：InnoDB 表默认的全文索引的停止词。
- KEYWORDS：MySQL 的关键词和是否保留。对于保留的关键词在引用时需要特别处理，如加上引号。
- PLUGINS：服务插件的信息。
- RESOURCE_GROUPS：资源组信息。
- ST_SPATIAL_REFERENCE_SYSTEMS：可用的空间参照系统清单。

### 5.4.3 性能视图

性能视图保存与实例相关的信息，分别说明如下。

- INNODB_BUFFER_PAGE：InnoDB 缓存中的页清单和使用信息，查询该视图比较消耗系统资源，最好在测试机器上实现。如下 SQL 语句为查询缓存中系统数据占用的页数、总的页数、系统页的占比：

```
mysql > select
(
select count( * ) from information_schema.innodb_buffer_page
where table_name is null or (instr(table_name, '/') = 0
and instr(table_name, '.') = 0)
) as system_pages,
( select count( * ) from information_schema.innodb_buffer_page) as total_pages,
(select round((system_pages/total_pages) * 100)) as system_page_percentage;
+--------------+-------------+-------------------------+
| system_pages | total_pages | system_page_percentage  |
+--------------+-------------+-------------------------+
|        15261 |       16384 |                      93 |
+--------------+-------------+-------------------------+
1 row in set (0.37 sec)
```

如下 SQL 语句为查询缓存中的用户数据：

```
mysql > select distinct table_name
from information_schema.innodb_buffer_page
where table_name is not null and
(instr(table_name, '/') > 0 or instr(table_name, '.') > 0)
and table_name not like ''mysql'.'innodb_ % ';
+-------------------------------------------------------------+
| table_name                                                  |
+-------------------------------------------------------------+
| 'mysql'.'columns'                                           |
| 'mysql'.'tables'                                            |
| 'mysql'.'index_column_usage'                                |
| 'mysql'.'schemata'                                          |
| 'mysql'.'tablespace_files'                                  |
...
```

如下 SQL 语句为查询索引数据在缓存中的大小：

```
mysql > select index_name,count( * ) as pages,
round(sum(if(compressed_size = 0, @@global.innodb_page_size, compressed_size))/1024/
1024) as 'total data (mb)'
from information_schema.innodb_buffer_page
group by index_name order by pages desc;
+--------------------------------+-------+------------------+
| index_name                     | pages | total data (mb)  |
+--------------------------------+-------+------------------+
| NULL                           | 15256 |              238 |
| PRIMARY                        |   962 |               15 |
| table_id                       |    20 |                0 |
| name                           |    14 |                0 |
| idx_fk_staff_id                |    11 |                0 |
...
```

- INNODB_BUFFER_PAGE_LRU：InnoDB 缓存中的页信息，特别是页在最近最少使用(least recently used，LRU)排序中的位置，这个位置决定了页被从缓存中驱逐的顺序，查询该视图也比较消耗系统资源，最好在测试机器上实现。
- INNODB_BUFFER_POOL_STATS：InnoDB 缓存池(pool)的使用，该视图中的信息和 SHOW ENGINE INNODB STATUS 命令输出的 BUFFER POOL AND MEMORY 部分的内容一样。每个池在该视图里对应一条记录，如下为查询该视图的代码及其运行结果：

```
mysql > select * from information_schema.innodb_buffer_pool_stats \G
*************************** 1. row ***************************
                         POOL_ID: 0
                       POOL_SIZE: 8192
                    FREE_BUFFERS: 6216
                  DATABASE_PAGES: 1962
              OLD_DATABASE_PAGES: 704
         MODIFIED_DATABASE_PAGES: 0
              PENDING_DECOMPRESS: 0
                   PENDING_READS: 0
               PENDING_FLUSH_LRU: 0
              PENDING_FLUSH_LIST: 0
                PAGES_MADE_YOUNG: 332
            PAGES_NOT_MADE_YOUNG: 1160
           PAGES_MADE_YOUNG_RATE: 0
       PAGES_MADE_NOT_YOUNG_RATE: 0
              NUMBER_PAGES_READ: 1488
           NUMBER_PAGES_CREATED: 486
           NUMBER_PAGES_WRITTEN: 1241
                PAGES_READ_RATE: 0
              PAGES_CREATE_RATE: 0
             PAGES_WRITTEN_RATE: 0
               NUMBER_PAGES_GET: 64392
                       HIT_RATE: 0
    YOUNG_MAKE_PER_THOUSAND_GETS: 0
NOT_YOUNG_MAKE_PER_THOUSAND_GETS: 0
         NUMBER_PAGES_READ_AHEAD: 251
```

```
        NUMBER_READ_AHEAD_EVICTED: 0
                  READ_AHEAD_RATE: 0
          READ_AHEAD_EVICTED_RATE: 0
                     LRU_IO_TOTAL: 0
                   LRU_IO_CURRENT: 0
                 UNCOMPRESS_TOTAL: 0
               UNCOMPRESS_CURRENT: 0
1 row in set (0.00 sec)
```

- INNODB_CACHED_INDEXES：InnoDB 缓存中的索引页信息。如下 SQL 语句为查询缓存中的索引名、表名和每个索引占用的页：

```
mysql > select tables.name as table_name, indexes.name as
index_name,cached.n_cached_pages as n_cached_pages from
information_schema.innodb_cached_indexes as cached,
information_schema.innodb_indexes as indexes,
information_schema.innodb_tables as tables
where cached.index_id = indexes.index_id
and indexes.table_id = tables.table_id
order by n_cached_pages desc;
+----------------------+----------------------+----------------+
| table_name           | index_name           | n_cached_pages |
+----------------------+----------------------+----------------+
| sbtest/sbtest1       | PRIMARY              |            139 |
| sbtest/sbtest2       | PRIMARY              |            139 |
| sakila/rental        | PRIMARY              |             47 |
| sakila/payment       | PRIMARY              |             44 |
...
```

- INNODB_CMP：对 InnoDB 压缩表进行操作的统计信息。
- INNODB_CMP_RESET：和视图 INNODB_CMP 记录的信息一样，但只记录上次查询后变化的信息。
- INNODB_CMP_PER_INDEX：保存的信息和视图 INNODB_CMP 一样，但按照索引进行分组。
- INNODB_CMP_PER_INDEX_RESET：和视图 INNODB_CMP_PER_INDEX 记录的信息一样，但只记录上次查询后变化的信息。
- INNODB_CMPMEM：在 InnoDB 缓存池中的压缩页信息。
- INNODB_CMPMEM_RESET：和视图 INNODB_CMPMEM 记录的信息一样，但只记录上次查询后变化的信息。
- INNODB_METRICS：与 InnoDB 相关的性能信息。记录的内容和全局状态变量类似，区别是只包括 InnoDB 的信息。每个记录对应一个性能状态值，每个记录都有一个 COMMENT 字段说明记录的内容。如下 SQL 语句为查询 InnoDB 中删除行的信息：

```
mysql > select * from information_schema.innodb_metrics
where name = 'dml_deletes'\G
*************************** 1. row ***************************
           NAME: dml_deletes
```

```
       SUBSYSTEM: dml
           COUNT: 11
       MAX_COUNT: 11
       MIN_COUNT: NULL
       AVG_COUNT: 0.000008523524151793117
     COUNT_RESET: 11
MAX_COUNT_RESET: 11
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
   TIME_ENABLED: 2021-06-28 07:26:31
  TIME_DISABLED: NULL
   TIME_ELAPSED: 1290546
     TIME_RESET: NULL
         STATUS: enabled
           TYPE: status_counter
        COMMENT: Number of rows deleted
1 row in set (0.01 sec)
```

当记录的字段 STATUS 的值是 disable 时，该性能状态值没有收集。默认收集的信息很少，可以通过设置系统参数 innodb_monitor_enable、innodb_monitor_disable、innodb_monitor_reset 和 innodb_monitor_reset_all 来激活、停止和重置计数。

- INNODB_SESSION_TEMP_TABLESPACES：该视图是从 MySQL 8.0.13 版本后增加的，它保存着 InnoDB 临时表空间对应的文件信息。如下 SQL 语句为查询该视图中的内容：

```
mysql> select path,size,sstate from
information_schema.innodb_session_temp_tablespaces;
+------------------------------+-----------+-----------+
| path                         | size      | state     |
+------------------------------+-----------+-----------+
| ./#innodb_temp/temp_10.ibt   | 15728640  | ACTIVE    |
| ./#innodb_temp/temp_1.ibt    |    81920  | INACTIVE  |
| ./#innodb_temp/temp_2.ibt    |    81920  | INACTIVE  |
| ./#innodb_temp/temp_3.ibt    |    81920  | INACTIVE  |
| ./#innodb_temp/temp_4.ibt    |    81920  | INACTIVE  |
| ./#innodb_temp/temp_5.ibt    |    81920  | INACTIVE  |
| ./#innodb_temp/temp_6.ibt    |    81920  | INACTIVE  |
| ./#innodb_temp/temp_7.ibt    |    81920  | INACTIVE  |
| ./#innodb_temp/temp_8.ibt    |    81920  | INACTIVE  |
| ./#innodb_temp/temp_9.ibt    |    81920  | INACTIVE  |
+------------------------------+-----------+-----------+
10 rows in set (0.01 sec)
```

根据前面的查询内容，可以找到操作系统中的文件，代码如下：

```
mysql> system ls /var/lib/mysql/#innodb_temp/temp_10.ibt
/var/lib/mysql/#innodb_temp/temp_10.ibt
```

如果从操作系统层面看到某个临时文件太大，可以通过该视图找到文件对应的会话。

- INNODB_TRX：InnoDB 的事务信息。如下 SQL 语句为查询超过 10 秒未提交的事务：

```
mysql > select trx_mysql_thread_id as thread_id ,
to_seconds(now()) - to_seconds(trx_started) as trx_last_time , user,host,db,trx_query
from information_schema.innodb_trx trx
join information_schema.processlist pcl
on trx.trx_mysql_thread_id = pcl.id
where trx_mysql_thread_id != connection_id()
and to_seconds(now()) - to_seconds(trx_started) >= 10;
+-----------+----------------+------+-----------+--------+-----------+
| thread_id | trx_last_time  | user | host      | db     | trx_query |
+-----------+----------------+------+-----------+--------+-----------+
|     41266 |           1031 | root | localhost | sakila | NULL      |
+-----------+----------------+------+-----------+--------+-----------+
1 row in set (0.01 sec)
```

这样的事务如果一直不能提交,可以使用如下命令将会话“杀死”:

```
mysql > kill 41266;
```

- OPTIMIZER_TRACE:当激活了系统参数 OPTIMIZER_TRACE 后,该视图中保存着优化器的跟踪信息,这些信息可以帮助进行 SQL 语句的优化。
- PROCESSLIST:MySQL 中当前线程正在执行的操作。该视图中保存的信息和 SHOW FULL PROCESSLIST 命令输出的信息一样。查询该视图的代码及其运行效果如下:

```
mysql > select * from information_schema.processlist limit 1 \G
*************************** 1. row ***************************
     ID: 41265
   USER: root
   HOST: localhost
     DB: NULL
COMMAND: Query
   TIME: 0
  STATE: executing
   INFO: select * from information_schema.processlist limit 1
1 row in set (0.01 sec)
```

- PROFILING:当会话的参数 profiling 激活后,该视图中保存着 SQL 语句的概要信息,该视图在未来的版本中将被弃用。

### 5.4.4 权限视图

与权限相关的视图有四个,表 5.1 列出了每个权限视图的记录内容和对应的系统表。

**表 5.1**

| 权限视图 | 记录内容 | 对应系统表 |
|---|---|---|
| user_privileges | 用户拥有的权限 | mysql.user 和 mysql.global_grants |
| schema_privileges | 对数据库访问的权限 | mysql.db |
| table_privileges | 对表访问的权限 | mysql.tables_priv |
| column_privileges | 对字段访问的权限 | mysql.columns_priv |

从这些视图查询到的信息和 show grants 命令查询的信息一致，如下代码说明了用这两种方式查询账号 scutech@localhost 权限的结果：

方式一：

```
mysql > show grants for scutech@localhost;
+-----------------------------------------------------------------+
| Grants for scutech@localhost                                    |
+-----------------------------------------------------------------+
| GRANT USAGE ON *.* TO 'scutech'@'localhost'                     |
| GRANT AUDIT_ADMIN ON *.* TO 'scutech'@'localhost'               |
| GRANT INSERT ON 'sakila'.'actor' TO 'scutech'@'localhost'       |
+-----------------------------------------------------------------+
3 rows in set (0.00 sec)
```

方式二：

```
mysql > select privilege_type, is_grantable, null table_schema, null table_name from information
_schema.user_privileges where grantee = '''scutech''@''localhost'''
union
select privilege_type, is_grantable, table_schema, null table_name from information_schema.
schema_privileges where grantee = '''scutech''@''localhost'''
union
select privilege_type, is_grantable, table_schema, table_name from information_schema.table_
privileges where grantee = '''scutech''@''localhost'''
union
select privilege_type, is_grantable, table_schema, table_name from information_schema.column_
privileges where grantee = '''scutech''@''localhost''';
+-----------------+--------------+--------------+------------+
| privilege_type  | is_grantable | table_schema | table_name |
+-----------------+--------------+--------------+------------+
| USAGE           | NO           | NULL         | NULL       |
| AUDIT_ADMIN     | NO           | NULL         | NULL       |
| INSERT          | NO           | sakila       | actor      |
+-----------------+--------------+--------------+------------+
3 rows in set (0.00 sec)
```

从上面的查询可以看到，用 show grants 命令查询账号 scutech@localhost 权限是从这四个权限视图查询到的该账号的权限之和。

## 5.5 实验

**MySQL 8.0 中 DDL 语句原子化**


视频演示

（1）对比同一个 DDL 语句在 MySQL 8.0 和 MySQL 5.7 中的执行结果，验证 DDL 语句在 MySQL 8.0 中实现了原子化。

（2）把 DDL 语句放在一个事务中，事务回滚后，观察 DDL 执行结果的变化。