第 3 章

CHAPTER 3

MySQL 元数据相关查询

3.1 show 关键字

在本书的第 1 章中介绍了 show databases 语句，show databases 可以读取 MySQL 服务器当前包含的所有数据库。例如可以使用 use 语句选择一个数据库，然后使用 show tables 语句查看当前数据库包含的所有表。

3.1.1 show 关键字查看某实例库中含有的表

show 关键字主要用于查询 MySQL 服务器相关的信息，SQL 语句如下：

```
//3.1.1 show 关键字查看某实例库中含有的表.sql

show databases;      #展示所有的实例库
use learnSQL;        #进入 learnSQL 实例库
show tables;         #展示 learnSQL 实例库中所有的表
```

运行后，结果如图 3-1 所示。

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| esif               |
| information_schema |
| learnSQL           |
| learnSQL2          |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
7 rows in set (0.01 sec)
```

图 3-1　查看实例库中表的结果集

show 关键字可以查询许多关于数据库、表、线程、状态等的实用信息，若要对 MySQL 有一个整体的了解，则掌握 show 命令是必不可少的。

展开来讲 show 命令可以查询许多 MySQL 服务器的信息，示例如下：

```
//3.1.1 show 关键字查看某实例库中含有的表.sql

show databases;                          -- 显示所有数据库
show tables;                             -- 显示数据表
show columns from table_name;            -- 显示数据表的属性
show index from table_name;              -- 显示数据表的索引
show triggers;                           -- 显示触发器
show status;                             -- 显示服务器状态
show variables;                          -- 显示服务器配置变量
```

```
show processlist;                    -- 显示服务器当前运行的线程
show grants;                         -- 显示授权
show errors;                         -- 显示最近的错误消息
```

3.1.2 show 关键字查看表结构

使用 show 关键字可以直接展示某张表的相关列信息，SQL 语句如下：

```
show columns from student;
#上下等价
desc student;
```

运行后，结果如图 3-2 所示。

Field	Type	Null	Key	Default	Extra
id	int unsigned	NO	PRI	NULL	auto_increment
age	int	YES		NULL	
sex	int	YES		NULL	
name	varchar(20)	YES		NULL	

4 rows in set (0.01 sec)

图 3-2 表结构结果集

3.1.3 show 关键字查看 binlog 日志

binlog 日志是 MySQL 中保留增、删、改内容的日志，也是 MySQL 主从同步最重要的日志，show 关键字可直接查看 binlog 日志目录，SQL 语句如下：

```
show binary logs
```

运行后，结果如图 3-3 所示。

Log_name	File_size	Encrypted
binlog.000013	11748	No
binlog.000014	35589	No
binlog.000015	47243898	No
binlog.000016	22178	No
binlog.000017	2673	No

5 rows in set (0.01 sec)

图 3-3 binlog 日志状态结果集

3.1.4 show 关键字查看相关创建语句信息

在使用 MySQL 创建相关实例库、表、视图、存储过程、函数等内容之后，皆可使用 show 语句查询其初始创建语句，以方便用户进行导入、导出等相关操作。

1. 查看实例库的创建语句

```
show create database learnSQL2;
```

运行后，结果如图 3-4 所示。

```
+-----------+------------------------------------------------------------------------------------------------------------------------------------+
| Database  | Create Database                                                                                                                    |
+-----------+------------------------------------------------------------------------------------------------------------------------------------+
| learnSQL2 | CREATE DATABASE `learnSQL2` /*!40100 DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci */ /*!80016 DEFAULT ENCRYPTION='N' */ |
+-----------+------------------------------------------------------------------------------------------------------------------------------------+
1 row in set (0.00 sec)
```

图 3-4　查看实例库创建语句的结果集

2. 查看表的创建语句

```
show create table student;
```

运行后，结果如图 3-5 所示。

```
+---------+--------------------------------------------------------------------
---------------------------------------------------------------------------------
| Table   | Create Table

+---------+--------------------------------------------------------------------
---------------------------------------------------------------------------------
| student | CREATE TABLE `student` (
  `id` int unsigned NOT NULL AUTO_INCREMENT COMMENT '学生表主键id',
  `age` int DEFAULT NULL COMMENT '年龄',
  `sex` int DEFAULT NULL COMMENT '性别',
  `name` varchar(20) DEFAULT NULL COMMENT '学生名称',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
+---------+--------------------------------------------------------------------
---------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

图 3-5　查看表创建语句的结果集

如图 3-5 所示的内容在查看时会有所困难，所以可以在 SQL 语句中增加\G 进行分行，SQL 语句如下：

```
show create table student\G;
```

运行后，结果如图 3-6 所示。

```
mysql> show create table student\G;
*************************** 1. row ***************************
       Table: student
Create Table: CREATE TABLE `student` (
  `id` int unsigned NOT NULL AUTO_INCREMENT COMMENT '学生表主键id',
  `age` int DEFAULT NULL COMMENT '年龄',
  `sex` int DEFAULT NULL COMMENT '性别',
  `name` varchar(20) DEFAULT NULL COMMENT '学生名称',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
1 row in set (0.00 sec)

ERROR:
No query specified
```

图 3-6　查看表的创建语句分行的结果集

3.1.5　show 关键字查看 MySQL 支持哪些引擎

show 关键字可以展示当前 MySQL 版本支持哪些引擎，SQL 语句如下：

```
show engines;
#上下等价
select * from information_schema.engines;
```

运行后，结果如图 3-7 所示。

```
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| Engine             | Support | Comment                                                        | Transactions | XA   | Savepoints |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
| ARCHIVE            | YES     | Archive storage engine                                         | NO           | NO   | NO         |
| BLACKHOLE          | YES     | /dev/null storage engine (anything you write to it disappears) | NO           | NO   | NO         |
| MRG_MYISAM         | YES     | Collection of identical MyISAM tables                          | NO           | NO   | NO         |
| FEDERATED          | NO      | Federated MySQL storage engine                                 | NULL         | NULL | NULL       |
| MyISAM             | YES     | MyISAM storage engine                                          | NO           | NO   | NO         |
| PERFORMANCE_SCHEMA | YES     | Performance Schema                                             | NO           | NO   | NO         |
| InnoDB             | DEFAULT | Supports transactions, row-level locking, and foreign keys     | YES          | YES  | YES        |
| MEMORY             | YES     | Hash based, stored in memory, useful for temporary tables      | NO           | NO   | NO         |
| CSV                | YES     | CSV storage engine                                             | NO           | NO   | NO         |
+--------------------+---------+----------------------------------------------------------------+--------------+------+------------+
9 rows in set (0.01 sec)
```

图 3-7　查看表的创建语句分行的结果集

3.2　数据库的系统变量元数据与 set 关键字

用户可以使用 set 关键字将用户自定义的变量存储至 MySQL 中。

3.2.1　set 关键字用于用户自定义变量

set 关键字的语法如下：

```
SET @var_name = expr [, @var_name = expr] …
```

@var_name 处可以输入用户自定义的变量名称，其中可包括字母、数字及字符组成。

“=”赋予符号可以更改为“:=”，两种是等价的。“:=”赋予符号是为了有别于 where 子句中的等号，所以在 set 关键字用于用户自定义变量时更推荐使用此赋予符号。

在赋予符号之后的 expr 表达式内可以使用数字、字符串或表达式。

在 MySQL 的 SQL 语句中，通常以“;”作为一句话的结尾。用户自定义变量之后，set 关键字需要结尾，后续 select 语句需要再次结尾，SQL 语句如下：

```
//3.2.1 set 关键字用于用户自定义变量.sql

set @v1 = '41';
set @v2 = '41' + 6;
set @v3 = @v2 - @v1;
select @v1, @v2, @v3;
#上下等价
set @v4 := '41';
set @v5 := '41' + 6;
set @v6 := @v2 - @v1;
select @v4, @v5, @v6;
```

分别执行上述 4 句话，运行后的效果如图 3-8 所示。

此处展示的效果类似于其他语言中的数据变量的定义与调用，在 MySQL 的查询中也经常可以用到，set 关键字的表达式的写法如下：

```
set @v1 = (select sal from emp limit 1);
select @v1;
```

在 set 关键字的表达式中只能返回 1 行 1 列，并用括号进行包围表达式才能经过校验。运行后，效果如图 3-9 所示。

```
mysql> set @v1 = '41';
Query OK, 0 rows affected (0.00 sec)

mysql> set @v2 = '41'+6;
Query OK, 0 rows affected (0.00 sec)

mysql> set @v3 = @v2-@v1;
Query OK, 0 rows affected (0.00 sec)

mysql> select @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| 41   |   47 |    6 |
+------+------+------+
1 row in set (0.00 sec)
```

图 3-8　set 自定义变量效果

```
mysql> set @v1 = (select sal from emp limit 1);
Query OK, 0 rows affected (0.01 sec)

mysql> select @v1;
+------+
| @v1  |
+------+
|  800 |
+------+
1 row in set (0.01 sec)
```

图 3-9　set 自定义变量效果

set 关键字的表达式写法若含有多列，则报错如下：

```
Operand should contain 1 column(s)
```

set 关键字的表达式写法若含有多行，则报错如下：

```
Subquery returns more than 1 row
```

3.2.2　set 关键字用于环境变量

MySQL 中含有许多系统变量（也可称为环境变量），系统变量是 MySQL 运行所需的重要元数据。

部分 MySQL 的系统变量由 MySQL 的配置文件（my.ini 文件或者 my.cnf 文件）进行配置和管理，部分 MySQL 的系统变量存储在 MySQL 缓存中。

当需要修改由配置文件配置的系统变量时，需要重启 MySQL 服务才能执行成功。若在配置文件中不含有任何系统变量的设置，则 MySQL 在启动时将以默认的方式对其进行设置。

show 关键字可以展示当前 MySQL 全部的系统变量，SQL 语句如下：

```
show variables;
```

服务器维护着两种系统变量，即全局变量（Global Variables）和会话变量（Session Variables）。全局变量影响 MySQL 服务的整体运行方式，而会话变量仅影响当前具体客户端连接的操作。

每个客户端成功连接服务器后都会产生与之对应的会话（Session）。会话期间 MySQL 服务实例会在服务器内存中生成与该会话对应的会话变量，会话变量的初始值是全局变量值的复制。

show 关键字可以展示当前 MySQL 全局的系统变量，SQL 语句如下：

```
show global variables;
```

show 关键字可以展示 MySQL 当前会话的系统变量。当前会话的系统变量指本次打开的 MySQL 会话中的系统变量，若关闭本次会话后重新打开，则一些之前被设置为当前会话级的系统变量会变回全局的系统变量，SQL 语句如下：

```
show session variables;
```

在 MySQL 中通常使用两种方式修改 MySQL 相关的系统变量：

(1) 修改 MySQL 的配置文件(my.ini 文件或者 my.cnf 文件)。

(2) 通过 set 关键字进行设置。

通过 set 关键字修改全局变量的 SQL 语句如下：

```
set global innodb_file_per_table = on;
#上下等价
set @@global.innodb_file_per_table = ON;
```

通过 set 关键字修改会话变量的 SQL 语句如下：

```
set @@session.pseudo_thread_id = 5;
#上下等价
set session pseudo_thread_id = 5;
```

在 MySQL 中通常以“@”符号作为用户自行设置的变量，以“@@”符号作为系统变量。在用户的 set 关键字的语句没有指定是全局变量还是会话变量的情况下，默认将设置为会话变量，SQL 语句如下：

```
set @@sort_buffer_size = 50000;
```

无论是 MySQL 的会话变量还是 MySQL 的全局变量，重启 MySQL 之后都会恢复为默认配置。

在 MySQL 的配置文件(my.ini 文件或者 my.cnf 文件)的[mysqld]配置内编写 MySQL 的系统变量可以保证 MySQL 重启之后，该系统变量仍然有效。MySQL 配置文件的示例如下：

```
//3.2.2 set 关键字用于环境变量.sql

[mysqld]
port = 3306
basedir = /home/zhangfangxing/mysql/base
datadir = /home/zhangfangxing/mysql/data
max_connections = 100
query_cache_size = 0
table_cache = 256
tmp_table_size = 35M
thread_cache_size = 8
key_buffer_size = 55M
```

```
read_rnd_buffer_size = 256K
sort_buffer_size = 256K
```

MySQL 配置文件的释义如表 3-1 所示。

表 3-1　配置文件变量释义

变 量 名 称	释　　义
port	MySQL 监听的端口号
basedir	MySQL 安装路径
datadir	MySQL 数据的存储位置
max_connections	允许同时访问 MySQL 服务器的最大连接数，其中一个连接是保留的，留给管理员专用
query_cache_size	查询时的缓存大小，缓存中可以存储以前通过 SELECT 语句查询过的信息，再次查询时就可以直接从缓存中取出信息，可以改善查询效率
table_open_cache	所有进程打开表的总数
tmp_table_size	内存中每个临时表允许的最大大小
thread_cache_size	缓存的最大线程数
key_buffer_size	关键词的缓存大小
read_rnd_buffer_size	将排序后的数据存入该缓存的大小
sort_buffer_size	用于排序的缓存大小

以上数据只是 MySQL 变量中的一小部分，皆可通过 MySQL 的 show 关键字进行查询，SQL 语句如下：

```
//3.2.2 set 关键字用于环境变量.sql

show variables where variable_name = 'port';
show variables where variable_name = 'basedir';
show variables where variable_name = 'datadir';
show variables where variable_name = 'max_connections';
show variables where variable_name = 'tmp_table_size';
show variables where variable_name = 'thread_cache_size';
show variables where variable_name = 'key_buffer_size';
show variables where variable_name = 'sort_buffer_size';
```

3.2.3　sql_mode 变量

MySQL 中的 sql_mode 为 SQL 语句校验变量，也是 MySQL 8.0 中最需要重视的变量之一，该变量可为空值，默认值如下：

```
//3.2.3 sql_mode 变量.sql

show variables where variable_name = 'sql_mode';
-- 返回
-- only_full_group_by,strict_trans_tables,no_zero_in_date,no_zero_date,error_for_division_
by_zero,no_engine_substitution
```

在 sql_mode 的设置下，MySQL 允许执行部分非法语句。

为了保证 SQL 语句的正确性，生产数据库与测试数据库的 sql_mode 值必须保证相同，否则会出现一条 SQL 语句在测试环境下可正常执行，但在生产情况下却没法正常执行的现象。

only_full_group_by：在该校验模式下将影响 SQL 语句中 group by 子句的编写方式。若在 select 中的列没有在 group by 的子句中出现，则该 SQL 是不合法的。此校验强行要求列在 group by 子句中标识出来。此校验是 MySQL 8.0.0 默认自带的校验，由于此校验是 MySQL 后期追加的校验，因此很多低版本 MySQL 的 SQL 语句无法跟高版本兼容，建议数据库部署时删掉此校验。

no_auto_value_on_zero：在该校验模式下将影响自增长列的插入。默认设置下插入 0 或 null 代表生成下一个自增长值。设置该值后可插入值为 0，但是会导致数据混乱的问题，建议数据库部署时删掉此校验。

strict_trans_tables：在该校验模式下，若一个值不能插入一个事务表中，则中断当前的操作，对非事务表不做限制。

no_zero_in_date：在该校验模式下，不允许日期和月份为 0。

no_zero_date：在该校验模式下，插入零日期会抛出错误而不是警告。

error_for_division_by_zero：在该校验模式下，在 insert 或 update 过程中，若数据被删除，则产生错误而非警告。

no_auto_create_user：在该校验模式下，禁止使用 MySQL 的 grant 命令创建密码为空的用户。

no_engine_substitution：在该校验模式下，若需要的存储引擎被禁用或未编译，则会抛出错误。当不设置此值时用默认的存储引擎替代并抛出一个异常。

pipes_as_concat：在该校验模式下，将“||”符号视为字符串的连接操作符而非或运算符，和字符串的拼接函数 concat()相类似。此校验模式主要为了兼容 Oracle 的 SQL 语句。

ansi_quotes：在该校验模式下，不能用双引号来引用字符串，双引号将被解释为识别符。

3.2.4 根据用户自定义变量增加列的行号

在日常编写 SQL 的过程中，若在 MySQL 5 版本上期望在结果集中增加列的行号，则需要使用用户自定义变量。后文会提到在 MySQL 8.0 版本之后可直接使用窗口函数 row_number()解决此问题。

查询公司表，SQL 语句如下：

```
select e.ename,e.job from emp e;
```

运行后，结果集如图 3-10 所示。

使用 set 关键字设置用户自定义变量增加结果集行号，SQL 语句如下：

```
#将用户自定义变量 rownum 设置为 0
set @rownum := 0;
#编写 SQL 语句增加行号查询
select @rownum:=@rownum+1,e.ename,e.job from emp e;
```

运行后，结果集如图 3-11 所示。

ename	job
张三	店员
李四	售货员
王五	售货员
赵六	经理
薛七	售货员
陈八	经理
吴九	经理
寅十一	文员
王十二	总经理
黄十三	售货员
毛十四	店员
陈十五	店员
张十六	文员
刘十七	店员

14 rows in set (0.26 sec)

图 3-10　EMP 表的结果集

@rownum:=@rownum+1	ename	job
1	张三	店员
2	李四	售货员
3	王五	售货员
4	赵六	经理
5	薛七	售货员
6	陈八	经理
7	吴九	经理
8	寅十一	文员
9	王十二	总经理
10	黄十三	售货员
11	毛十四	店员
12	陈十五	店员
13	张十六	文员
14	刘十七	店员

14 rows in set, 1 warning (0.00 sec)

图 3-11　EMP 表增加序号的结果集

关于此语句需要注意以下内容：

（1）因为以上 SQL 含有两个分号（“;”符号），所以其为两句 SQL 语句，第一句设置了 rownum 变量，第二句根据 rownum 变量进行查询。

（2）因为 SQL 的 select 本身含有指针的特性，即新的一行被扫描之后，SQL 相当于重新查看一次该行的所有列，所以可以在新一行的列中让 rownum 重新赋予新的值，新的值为 rownum＋1。本质上 rownum 只是利用了 SQL 语言的指针特性，与循环类似。

（3）用户变量不会在 SQL 语句结束后自行重置，若再次运行此 SQL 语句且不重置用户变量，则 SQL 执行结果如图 3-12 所示。

@rownum:=@rownum+1	ename	job
15	张三	店员
16	李四	售货员
17	王五	售货员
18	赵六	经理
19	薛七	售货员
20	陈八	经理
21	吴九	经理
22	寅十一	文员
23	王十二	总经理
24	黄十三	售货员
25	毛十四	店员
26	陈十五	店员
27	张十六	文员
28	刘十七	店员

14 rows in set, 1 warning (0.00 sec)

图 3-12　不重置变量的效果

若希望在将两条语句整合成一条语句时仅使用一句 SQL 在 emp 表增加序号，则可以不使用 set 关键字而使用用户变量，SQL 语句如下：

```
//3.2.4 根据用户自定义变量增加列的行号.sql

select
   (@myrow := @myrow + 1) AS line,
   e.ename,
   e.job
from
   ( select ename, job from emp ) e,
```

```
    ( select @myrow := 0 ) r;

-- 以上 SQL 在 FROM 处隐性地将用户变量设置为 0,所以可以在列处直接调用
-- FROM 关键字只执行一次,但是 SELECT 关键字将执行 N 次
```

运行后,结果集如图 3-13 所示。

```
+------+--------+--------+
| line | ename  | job    |
+------+--------+--------+
|    1 | 张三   | 店员   |
|    2 | 李四   | 售货员 |
|    3 | 王五   | 售货员 |
|    4 | 赵六   | 经理   |
|    5 | 薛七   | 售货员 |
|    6 | 陈八   | 经理   |
|    7 | 吴九   | 经理   |
|    8 | 寅十一 | 文员   |
|    9 | 王十二 | 总经理 |
|   10 | 黄十三 | 售货员 |
|   11 | 毛十四 | 店员   |
|   12 | 陈十五 | 店员   |
|   13 | 张十六 | 文员   |
|   14 | 刘十七 | 店员   |
+------+--------+--------+
14 rows in set, 2 warnings (0.12 sec)
```

图 3-13　仅使用一句 SQL 在 emp 表增加序号的结果集

3.3　表的元数据

计算机编程语言中常常含有通用的概念,例如在 Java 中有 Class 类用于获取对类的定义,以及有 Method 类用于获取函数的定义。

Java 程序员可以通过 Class 类的 getClass()函数获取类的反射,类的反射中会包含类名、引用包名等相关定义类信息,这些信息可被看作 Java 的元数据。

MySQL 元数据是关于数据库本身的数据,用于描述数据库的结构和属性。

3.3.1　表的元数据查询

前文提到过 MySQL 依靠 information 实例库存储 MySQL 有关库和表的信息,例如 information_schema.tables 表存储针对表的定义,包括表的名称、表的总行数、表的创建时间、表的最后修改时间等相关内容。

依靠 information_schema.table_constraints 表存储针对表的约束进行查询。

后续在学习 MySQL 调优的过程中要学会详细查看 information_schema 实例库存储的各个表,information_schema 实例库中部分涉及 MySQL 调优的表如表 3-2 所示。

表 3-2　体验特性结果集

information_schema 中部分涉及调优的表	主要存储内容
files	存储表空间数据的文件
innodb_buffer_page	InnoDB 缓冲池中的页面
innodb_buffer_page_lru	InnoDB 缓冲池中页面的 LRU 排序

续表

information_schema 中部分涉及调优的表	主要存储内容
innodb_buffer_pool_stats	InnoDB 缓冲池统计
innodb_cmp	压缩 InnoDB 表相关的操作的状态
innodb_cmp_per_index	压缩 InnoDB 表和索引相关的操作状态
innodb_cmp_per_index_reset	压缩 InnoDB 表和索引相关的操作状态
innodb_cmp_reset	压缩 InnoDB 表相关的操作的状态
innodb_cmpmem	InnoDB 缓冲池中压缩页面的状态
innodb_cmpmem_reset	InnoDB 缓冲池中压缩页面的状态
innodb_ft_config	InnoDB 表全文本索引和相关处理的元数据
innodb_metrics	InnoDB 性能信息
innodb_trx	活跃的 InnoDB 事务信息
innodb_temp_table_info	关于活动用户创建的 InnoDB 临时表的信息
innodb_sys_indexes	InnoDB 索引元数据
optimizer_trace	优化器跟踪活动生成的信息
partitions	表格分区信息
processlist	有关当前执行线程的信息
statistics	表索引统计

在实际工作中若只是想知道简单的信息，则只需使用 show 关键字，毕竟 show 关键字运行后所查询的仍然是 information_schema 实例库中的众多表，其中包括了 information_schema. tables 和 information_schema. engines 等。

关键字的出现其实并不意外，例如 show 类关键字帮助简化查询元数据、新增数据库时使用的 create 关键字、删除数据库时使用的 drop 关键字，本质上都是在操纵 information_schema 实例库下的各个表。

因为元数据表互相之间的关系过于复杂，无法达到让人在其中快速地做出对表的 DLL 操作，所以出现了众多简化操作的关键字，用于辅助日常工作。

不过一旦涉及了数据库性能的优化，仍然需要数据库管理员对 MySQL 基础的各个表有足够的了解，尤其是 MySQL 的 information_schema 实例库中的 InnoDB 引擎系列表、information_schema 实例库中的线程池系列表、information_schema 实例库中的连接控制表等重要性能指标表，能够快速地定位数据查询缓慢、数据库内存占用过高、大数据量无法导出或迁移等问题。

MySQL 查看表的元数据的 SQL 语句如下：

```
//3.3.1 表的元数据查询.sql

select
   table_name as 表名称,
   table_rows as 表总行数,
   create_time as 表创建时间,
   update_time as 表最后修改时间,
```

```
    table_collation as 表排序规则,
    engine as 表引擎
from
    information_schema.tables
where
    table_schema = 'learnSQL2';
```

运行后,结果集如图 3-14 所示。

表名称	表总行数	表创建时间	表最后修改时间	表排序规则	表引擎
course	4	2022-10-15 14:21:41	NULL	utf8mb4_0900_ai_ci	InnoDB
score	11	2022-10-15 14:24:07	NULL	utf8mb4_0900_ai_ci	InnoDB
student	5	2022-10-15 14:21:28	2022-10-22 22:53:13	utf8mb4_0900_ai_ci	InnoDB
t1	3	2022-10-20 22:29:06	2022-10-23 16:58:09	utf8mb4_0900_ai_ci	InnoDB
t2	4	2022-10-27 16:45:12	NULL	utf8mb4_0900_ai_ci	InnoDB
teacher	5	2022-10-15 14:21:18	NULL	utf8mb4_0900_ai_ci	InnoDB
tt	1	2022-10-22 17:01:23	2022-10-22 17:03:51	utf8mb4_0900_ai_ci	InnoDB

图 3-14　查看表元数据的结果集

在 information_schema 实例库中的 tables 表中,除了图 3-14 中展示的数据之外,存储着有关于用户自行创建的表的其他元数据内容,tables 表的字段释义如表 3-3 所示。

表 3-3　information_schema.tables 表的字段释义

列　　值	释　　义
table_catalog	数据表登记目录
table_schema	数据表所属的数据库名
table_name	表名称
table_type	表类型[system view|base table]
engine	表引擎[MyISAM|CSV|InnoDB]
version	版本
row_format	行格式[DEFAULT | FIXED | DYNAMIC | COMPRESSED | REDUNDANT | COMPACT]
table_rows	表总行数
avg_row_length	平均长度
data_length	数据长度
max_data_length	最大数据长度
index_length	索引长度
data_free	空间碎片
auto_increment	做自增主键的自动增量当前值
create_time	表创建时间
update_time	表最后修改时间
check_time	表的检查时间
table_collation	表排序规则
checksum	校验和
create_options	创建选项
table_comment	表的注释、备注

base table 为基础表，是用户在 create table 时所创建的实体表，称为 base 表，也可称为基表。

system view 为用户在 create view 时所创建的视图。

注意：view 视图不仅在此处有存储，后文会继续提到在 information_schmea.view 表中也有相应存储。

3.3.2　表信息中的 row_format 字段

MySQL 中 row_format(行格式)用于设置表的行存储格式。存储格式决定了其物理存储方式。物理存储方式会影响 DML(数据库操纵)与 DQL(数据库查询)的性能。

1. 行格式简述

InnoDB 引擎支持 4 种行格式，分别为 redundant(冗余型行格式)、compact(紧凑型行格式)、dynamic(动态型格式)和 compressed(压缩型行格式)，其 4 种行格式简明释义如表 3-4 所示。

表 3-4　InnoDB 中 4 种行格式的简明释义

行　格　式	是否紧凑存储	是否可变长度存储	大索引前缀支持	是否支持压缩	支持的表空间
redundant	否	否	否	否	系统、通用、独立表空间
compact	是	否	否	否	系统、通用、独立表空间
dynamic	是	是	是	否	系统、通用、独立表空间
compressed	是	是	是	是	通用、独立表空间

redundant 行格式兼容 MySQL 旧版的特性，与 redundant 行格式相比，compact 行格式减少了 20%的存储空间，但代价是增加了某些操作时的 CPU 占用率。若服务器的压力来自缓存命中率或者磁盘 IO 限制，则 compact 的行格式读取速度会更快。若查询压力来自 CPU，则使用 compact 的行格式时速度会更慢。

dynamic 行格式提供了与 compact 行格式相同的存储特性，但为大型的可变长列(例如 varchar、varbinary、blob 和 text 等类型)增强了存储性能，并支持大型的索引键前缀。dynamic 行格式最多支持 3072 字节的索引前缀。

所谓大型索引键前缀指的是给 text 文本等大型字符串设置的索引，该索引所需存储空间同样较大。

compressed 行格式提供了与 dynamic 行格式相同的存储特性和功能，但增加了对表和对索引数据的压缩支持。

目前 MySQL 8.0 推荐使用 dynamic 动态行格式和 compressed 压缩行格式。

在表设计上，若一张表里面不存在 varchar、varbinary、blob 和 text 等相关变形字段，则该表被称为静态表，每条记录所占用的字节一样。静态表的优点是读取快，缺点是浪费额外

一部分空间。

若一张表里面存在 varchar、varbinary、blob 和 text 等相关变形字段，则该表被称为动态表，该表的 row_format 是 dynamic，其每条记录所占用的字节均是动态的。

因为动态表的优点是节省空间，所以搜索查询量大的表一般以空间来换取时间，即被设计成静态表。缺点是增加读取的时间开销。

2. 查询默认行格式

MySQL 8.0 默认行格式为 dynamic，可通过 show 关键字进行查看，SQL 语句如下：

```
show variables like 'innodb_default_row_format';
#上下等价
select @@innodb_default_row_format;
```

运行后，结果集如图 3-15 所示。

```
+---------------------------+---------+
| Variable_name             | Value   |
+---------------------------+---------+
| innodb_default_row_format | dynamic |
+---------------------------+---------+
1 row in set (0.01 sec)
```

图 3-15 查看 MySQL 8.0 的默认行格式结果集

3. 修改默认行格式

修改默认行格式，SQL 语句如下：

```
set global innodb_default_row_format = DYNAMIC;
```

运行后，结果如下：

```
Query OK, 0 rows affected (0.01 sec)
```

4. 修改默认行格式的注意事项

因为 compressed 行格式所支持的表空间不同，所以不能将 compressed 设置为默认行格式，只能在 create table 和 alter table 语句中出现。

若强行将 compressed 行格式设置为 MySQL 默认变量，则报错如下：

```
ERROR 1231 (42000): Variable 'innodb_default_row_format' can't be set to the value of 'COMPRESSED'
```

行格式在互相转换时可能会出现更改类型的问题，例如当将 fixed 格式转换为 dynamic 格式时可能会将字段的 char 类型转换成 varchar 类型。

当将 dynamic 行格式转换成 fixed 行格式时可能将字段的 varchar 类型转换成 char 类型。

建议在初始化表设置行格式之后，尽量不要对其进行修改，强行修改可能对已存储的数据造成不可逆的损害。

5. 创建表时设置行格式

创建表时，同样可以单独给某张表设置行存储格式，SQL 语句如下：

```
create table test_row_format(
   T1 varchar(255),
   T2 INT
)row_format = COMPRESSED;
```

运行后，结果如下：

```
Query OK, 0 rows affected (0.01 sec)
```

创建表后，可查看当前表的存储格式，可参考 3.2.1 节，也可使用如下 SQL 语句：

```
show table status where name = 'test_row_format';
```

6. 修改表的行格式

使用 alter table 语句创建新列的 SQL，SQL 语句如下：

```
alter table test_row_format row_format = dynamic;
```

运行后，结果如下：

```
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

7. 修改表追加字段时设置行格式

使用 alter table 语句创建新列，SQL 语句如下：

```
alter table test_row_format add column(T3 int),row_format = DYNAMIC;
```

运行后，结果如下：

```
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

8. 其他相关注意事项

redundant 行格式和 compact 行格式支持最大 767 字节的最大索引前缀长度，而 dynamic 行格式和 compressed 行格式支持最大 3072 字节的最大索引前缀长度。简而言之，dynamic 行格式和 compressed 行格式对大字段的索引支持力度要大于 redundant 行格式和 compact 行格式。

compressed 行格式的表空间支持力度不如其他 3 种，若一定要修改行格式，则需先确定好所有已存在数据的表空间、索引最大长度等方面内容皆兼容两种行格式，在此前提下才能进行转换。

在 MySQL 集群复制的环境下，若 InnoDB 的 row_format 变量在主服务器上被设置为 dynamic 行格式，并且在从服务器上被设置为 compact 行格式，则某些 DDL(数据库定义)语句在主服务器上成功，但在从服务器上可能失败。

因为 MySQL 集群复制的环境必须保证，所以主服务器节点 master 和从服务器节点 slave 的 row_format 要保持一致。

除了 DDL 语句之外，导入不同的数据、不同的数据结构都有异常的风险。届时各节点数据结构不统一、数据不统一将极难对数据库进行处理和维护，所以初始化时一定要检查是否一致。

3.3.3 表信息中的 data_free 字段

data_free 字段释义：每当 MySQL 数据库使用 delete 语句删除一行内容时，在 MySQL 底层中该段空间就会被留空，而在一段时间内的大量删除操作会使留空的空间变得比存储列表内容所使用的空间更大。

若进行新的插入操作，则 MySQL 将尝试利用留空的区域，但仍然无法将其彻底占用，所以此刻可使用 optimize 关键字或 alter table 关键字对其进行优化。

3.3.4 MySQL 各表占用磁盘空间计算方式

MySQL 占用磁盘空间为 information_schema.tables 表中 Data_length 数据长度＋Index_length 索引长度，具体公式如下：

```
(Data_length + Index_length ) /1024 /1024 = 磁盘占用空间(MB)
```

在实际工作中查询 MySQL 某库实际占用空间的 SQL 语句如下：

```
//代码位置：全书代码/3.3.4 MySQL 各表占用磁盘空间计算方式.sql

select
   table_name,
   table_rows,
   data_length + index_length as length,
   concat(round((data_length + index_length)/1024/1024,2),'mb') as data
from
   information_schema.tables
where
   table_schema = 'esif'
order by
   length desc
```

上述 SQL 的具体函数后文均会有逐步解答，在此只展示一下语句。为了体现出效果，笔者在自身公司测试环境下查询公司的 esif 实例库，运行后，结果集如图 3-16 所示。

TABLE_NAME	TABLE_ROWS	length	DATA
mm_material_task_log	906746	26353860608	25133.00MB
ec_data_records	10866836	3371302912	3215.13MB
qm_check_result	3770065	1447231488	1380.19MB
qce_flow	71084	1419722752	1353.95MB
mm_technology_paramvalue	2696087	1024737280	977.27MB
im_inspectplan_dt	2069014	900857856	859.13MB
mm_alarm_exception_log	11427	767049728	731.52MB
qce_message	621297	736673792	702.55MB
mm_temhum_record	8977688	712867840	679.84MB
qce_flow_copy1	28345	657260544	626.81MB

图 3-16 查看 MySQL 各表占用磁盘空间的结果集

以 MySQL 各表占用磁盘空间计算方式进行演化，自然可以继续写出“查看 MySQL 各库占用磁盘空间计算方式”“MySQL 单个库占用磁盘空间计算方式”“MySQL 所有库占用磁盘空间计算方式”等相关内容。

注意：不要直接修改 information_schema.tables 表。

3.3.5　利用 optimize 关键字优化空间碎片

optimize 关键字使用的前提是需要开启数据库安全模式及 old_alter_table 系统变量。old_alter_table 参数代表服务器不会使用处理 alter table 操作的优化方法。

optimize 关键字和 alter table 操作的本质都是 MySQL 数据库会先锁定某张表，然后对该表进行复制，再迅速将之前的表删除，对第 2 张表进行改名。

简而言之，程序员可以通过 alter table 的操作优化方法复制基表，复制出来新的基表自然不含空间碎片。

optimize 关键字的原理与 alter table 的原理类似，在官网的表述中若 optimize table table.name 命令执行失败，则可尝试执行 alter table tests engine＝'innodb';命令。

查询系统变量中的 old_alter_table 参数，SQL 语句如下：

```
show variables like '%old_alter_table%'
```

运行后，结果集如图 3-17 所示。

因为 MySQL 系统变量中的 old_alter_table 并未开启，所以可以使用 MySQL 中的 set 关键字设置数据库的临时变量，开启 old_alter_table 参数的 SQL 语句如下：

```
set @@old_alter_table = ON;
```

运行后，结果如下：

```
Query OK, 0 rows affected (0.01 sec)
```

再次查询 old_alter_table 参数，结果集如图 3-18 所示。

```
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| old_alter_table | OFF   |
+-----------------+-------+
```

图 3-17　查看 old_alter_table 参数是否开启的结果集

```
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| old_alter_table | ON    |
+-----------------+-------+
1 row in set (0.03 sec)
```

图 3-18　再次查看 old_alter_table 参数是否开启的结果集

查看 tt 表的相关信息语句，SQL 语句如下：

```
//3.3.5 利用 optimize 关键字优化空间碎片.sql

select
      table_name as 表名称,
```

```
        table_rows as 表总行数,
        create_time as 表创建时间,
        update_time as 表最后修改时间,
        table_collation as 表排序规则,
        engine as 表引擎,
        data_free as 空间碎片
from
        information_schema.tables
where
        table_schema = 'learnSQL2'
        and
        table_name = 'tt';
```

运行后,结果集如图 3-19 所示。

表名称	表总行数	表创建时间	表最后修改时间	表排序规则	表引擎	空间碎片
tt	1	2022-10-22 17:01:23	2022-10-22 17:03:51	utf8mb4_0900_ai_ci	InnoDB	0

图 3-19　insert 之后的 tt 表相关数据的结果集

可以看到单纯使用 insert 关键字,无论新增了多少数据都不会产生空间碎片,在 insert 大量数据之后可直接进行删除。删除 tt 表的 SQL 语句如下:

```
delete from tt;
```

运行后,结果如下:

```
Query OK, 1 row affected (0.37 sec)
```

为了方便测试,虽然 tt 数据只插入了 1 条数据,但是用的是 longtext 数据类型,该条数据中的内容特别多,删除 tt 表全部的新增数据之后,再次执行查看 tt 表的相关信息的 SQL 语句,结果集如图 3-20 所示。

表名称	表总行数	表创建时间	表最后修改时间	表排序规则	表引擎	空间碎片
tt	0	2022-10-22 17:01:23	2022-10-28 10:02:32	utf8mb4_0900_ai_ci	InnoDB	24117248

图 3-20　删除之后的 tt 表相关数据的结果集

此时可观察到 tt 表中含有大量的空间碎片。使用 optimize 关键字对表空间碎片进行优化,SQL 语句如下:

```
optimize table tt;
```

运行后,结果集如图 3-21 所示。

Table	Op	Msg_type	Msg_text
learnsql2.tt	optimize	note	Table does not support optimize, doing recreate + analyze instead
learnsql2.tt	optimize	status	OK

2 rows in set (0.05 sec)

图 3-21　运行 optimize 关键字的结果集

此处出现了提示 Table does not support optimize, doing recreate + analyze instead，提醒用户该表不支持优化，而应执行重新创建＋分析。该提示属于提示/警告的类型，不属于报错，再次执行查看 tt 表的相关信息的 SQL 语句，运行后结果集如图 3-22 所示。

```
+----------+----------+---------------------+---------------------+--------------------+--------+----------+
| 表名称   | 表总行数 | 表创建时间          | 表最后修改时间      | 表排序规则         | 表引擎 | 空间碎片 |
+----------+----------+---------------------+---------------------+--------------------+--------+----------+
| tt       |        0 | 2022-10-28 10:03:56 | NULL                | utf8mb4_0900_ai_ci | InnoDB |        0 |
+----------+----------+---------------------+---------------------+--------------------+--------+----------+
1 row in set (0.01 sec)
```

图 3-22　优化之后 tt 表相关数据的结果集

可查看最终 tt 表的空间碎片归零了。整体运行过程如图 3-23 所示。

```
+----------+----------+---------------------+---------------------+--------------------+--------+----------+
| 表名称   | 表总行数 | 表创建时间          | 表最后修改时间      | 表排序规则         | 表引擎 | 空间碎片 |
+----------+----------+---------------------+---------------------+--------------------+--------+----------+
| tt       |        1 | 2022-10-22 17:01:23 | 2022-10-22 17:03:51 | utf8mb4_0900_ai_ci | InnoDB |        0 |
+----------+----------+---------------------+---------------------+--------------------+--------+----------+
1 row in set (0.00 sec)

mysql> delete from tt;
Query OK, 1 row affected (0.37 sec)

mysql> select   table_name as 表名称,  table_rows as 表总行数,      create_time as 表创建时间,      update_time as 表最后修改时间,
m   information_schema.tables   where   table_schema = 'learnSQL2' and table_name = 'tt';
+----------+----------+---------------------+---------------------+--------------------+--------+----------+
| 表名称   | 表总行数 | 表创建时间          | 表最后修改时间      | 表排序规则         | 表引擎 | 空间碎片 |
+----------+----------+---------------------+---------------------+--------------------+--------+----------+
| tt       |        0 | 2022-10-22 17:01:23 | 2022-10-28 10:02:32 | utf8mb4_0900_ai_ci | InnoDB |    24117248 |
+----------+----------+---------------------+---------------------+--------------------+--------+----------+
1 row in set (0.00 sec)

mysql> optimize table tt;
+--------------+----------+----------+-------------------------------------------------------------------+
| Table        | Op       | Msg_type | Msg_text                                                          |
+--------------+----------+----------+-------------------------------------------------------------------+
| learnsql2.tt | optimize | note     | Table does not support optimize, doing recreate + analyze instead |
| learnsql2.tt | optimize | status   | OK                                                                |
+--------------+----------+----------+-------------------------------------------------------------------+
2 rows in set (0.05 sec)

mysql> select   table_name as 表名称,  table_rows as 表总行数,      create_time as 表创建时间,      update_time as 表最后修改时间,
m   information_schema.tables   where   table_schema = 'learnSQL2' and table_name = 'tt';
+----------+----------+---------------------+---------------------+--------------------+--------+----------+
| 表名称   | 表总行数 | 表创建时间          | 表最后修改时间      | 表排序规则         | 表引擎 | 空间碎片 |
+----------+----------+---------------------+---------------------+--------------------+--------+----------+
| tt       |        0 | 2022-10-28 10:03:56 | NULL                | utf8mb4_0900_ai_ci | InnoDB |        0 |
+----------+----------+---------------------+---------------------+--------------------+--------+----------+
1 row in set (0.01 sec)
```

图 3-23　整体运行过程

若该数据库为 MySQL 导入的 data 文件夹或者导入的数据库文件，则可能会导致 optimize 关键字执行时失败，该空间碎片可能永久无法删除或每次只能删除额外的一部分，例如空间碎片 50 000＋时只能删除其中 10 000＋的碎片。

在进行实际操作时，可先查看 MySQL 的 data 文件夹大小，删除后可发现该文件夹明显缩小了。

3.3.6　查看表中的约束

MySQL 查看表的相关约束的 SQL 语句如下：

```
//3.3.6 查看表中的约束.sql

select
      tc.constraint_catalog as 约束所属的目录,
```

```
        tc.table_schema as 约束所属的数据库名称,
        tc.table_name as 表的名称,
        constraint_type as 约束类型
from
        information_schema.table_constraints tc
where
        tc.constraint_schema = 'learnSQL2';
```

运行后,结果集如图 3-24 所示。

约束所属的目录	约束所属的数据库名称	表的名称	约束类型
def	learnSQL2	course	PRIMARY KEY
def	learnSQL2	dept	PRIMARY KEY
def	learnSQL2	emp	PRIMARY KEY
def	learnSQL2	student	PRIMARY KEY
def	learnSQL2	teacher	PRIMARY KEY
def	learnSQL2	test_regexp	PRIMARY KEY
def	learnSQL2	tt	PRIMARY KEY

7 rows in set (0.00 sec)

图 3-24　查看表的相关约束结果集

MySQL 在 information_schema.table_constraints 表中存储了表的相关约束信息,此信息可通过 show 关键字便捷地进行查找。

例如使用如下 show 关键字的 SQL 语句可以查询 dept 表中的约束,SQL 语句如下:

```
show index from dept;
```

运行后,结果集如图 3-25 所示。

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
dept	0	PRIMARY	1	deptno	A	4	NULL	NULL		BTREE			YES	NULL

1 row in set (0.01 sec)

图 3-25　查看 dept 表中约束的结果集

show 关键字在展示不清时,可以使用\G 语句进行输出,SQL 语句如下:

```
show index from dept \G;
```

运行后,结果集如图 3-26 所示。

```
*************************** 1. row ***************
        Table: dept
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: deptno
    Collation: A
  Cardinality: 4
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
      Visible: YES
   Expression: NULL
1 row in set (0.00 sec)
```

图 3-26　分行输出的结果集

针对 show index 字段进行释义如表 3-5 所示。

表 3-5　show index 的字段释义

列　　值	释　　义
table	表名称
non_unique	若索引不能包含重复项，则为 0，若可以包含重复项，则为 1
key_name	索引的名称。若索引是主键，则名称始终为 primary
seq_in_index	索引中的列序号，以 1 开头
column_name	列的名称
collation	如何在索引中对列进行排序。可以有值 a(升序)或 null(未排序)
cardinality	估计索引中唯一值的数量，但是该值不一定准确
sub_part	索引前缀。若列仅部分索引，则索引字符的数量； 若整个列已索引，则为 null
packed	指示按键的包装方式。若不是，则为 null
null	若列可能含有 null 值，则输出 yes，若不含 null 值，则输出''
index_type	使用的索引方法(btree、fulltext、hash、rtree)
comment	有关其自身列中未描述的索引的信息，例如若索引被禁用，则返回 disabled
index_comment	创建索引时，使用 comment 属性为索引提供的任何注释

3.4　列的元数据

MySQL 查看列的元数据，SQL 语句如下：

```
//3.4 列的元数据.sql

select
   c.table_name as 所属表名称,
   c.column_name as 列的名称,
   c.column_default as 列的默认值,
   c.is_nullable as 是否可空,
   c.data_type as 数据类型,
   c.column_type as 列的数据类型,
   c.column_key as 索引类型,
   c.column_comment as 列的注释
from
   information_schema.columns c
where
   c.table_name = 'dept';                    #此例子查询的 dept 表中的列数据
```

运行后，结果如图 3-27 所示。

在 MySQL 中含有普通索引 index、默认约束 default、唯一约束 unique、检查约束 check、非空约束 not null、主键约束 primary key、外键约束 foreign key。

所属表名称	列的名称	列的默认值	是否可空	数据类型	列的数据类型	索引类型	列的注释
dept	deptno	NULL	NO	int	int unsigned	PRI	部门编号
dept	dname	NULL	YES	varchar	varchar(255)		部门名称
dept	loc	NULL	YES	varchar	varchar(255)		部门地点
dept	deptno	NULL	NO	int	int unsigned	PRI	部门编号
dept	dname	NULL	YES	varchar	varchar(255)		部门名称
dept	loc	NULL	YES	varchar	varchar(255)		部门地点

6 rows in set (0.01 sec)

图 3-27　查看列元数据的结果集

在 column_key 索引类型字段中，若 column_key 为空，则该列要么没有索引，要么仅作为多列非唯一索引中的辅助列进行索引。

若 column_key 是 pri，则该列是 primary key 或多列 primary key 中的列之一。

若 column_key 是 uni，则该列是 unique 索引的第 1 列。unique 索引允许多个 null 值，但可以通过检查 null 列来判断该列是否允许 null。

若 column_key 是 mul，则该列是非唯一索引的第 1 列，其中允许在列中多次出现给定值。

若多个 column_key 值适用于表的给定列，则 column_key 将按 pri、uni、mul 的顺序显示具有最高优先级的值。

若 unique 索引不能包含 null 值，并且表中没有 primary key，则可以将其显示为 pri。

若几列形成复合 unique 索引，则 unique 索引可能会显示为 mul。

3.5　用户权限的元数据

MySQL 依靠 mysql.user 表查询用户权限的元数据。因为 mysql.user 表中含有的列过多，所以本节分批对其进行讲解。

除了 mysql.user 表存储着用户的相关权限信息之外，procs_priv 表存储着存储过程和存储函数的操作权限，db 表存储着实例库的操作权限，mysql.tables_priv 表存储着对表操作的权限，mysql.columns_priv 表存储着对列操作的权限。

3.5.1　查询当前 MySQL 中含有哪些用户

通过 mysql.user 表可查询当前 MySQL 中含有哪些用户，SQL 语句如下：

```
//3.5.1 查询当前 MySQL 中含有哪些用户.sql

select
   u.host as 地址,
   u.user as 用户名,
   u.plugin as 密码加密方式,
   u.authentication_string as 被加密后的密码
from
   mysql.user u;
```

运行后，结果集如图 3-28 所示。

```
+-----------+-----------------+-----------------------+------------------------------------------------------------------------+
| 地址      | 用户名          | 密码加密方式          | 被加密后的密码                                                         |
+-----------+-----------------+-----------------------+------------------------------------------------------------------------+
| localhost | mysql.infoschema | caching_sha2_password | $A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED |
| localhost | mysql.session    | caching_sha2_password | $A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED |
| localhost | mysql.sys        | caching_sha2_password | $A$005$THISISACOMBINATIONOFINVALIDSALTANDPASSWORDTHATMUSTNEVERBRBEUSED |
| localhost | root             | mysql_native_password | *92DAC818B0794C0F6D0A633B39F132F5D8D0398C                               |
+-----------+-----------------+-----------------------+------------------------------------------------------------------------+
4 rows in set (0.00 sec)
```

图 3-28　当前 MySQL 用户的结果集

host 地址指该用户被授权访问的地址。此处的 localhost 代表只有 localhost 地址的人才能登录该账户。若此处被写成 192.168.1.1，则代表只有 IP 地址为 192.168.1.1 的人才能登录该账户。若此处被写成 0.0.0.0，则代表任意 IP 地址的人都可以登录该账户。

MySQL 数据库在创建时默认含有 4 个用户，分别是 mysql.infoschema、mysql.session、mysql.sys、root。root 为最高权限的管理员，其余 3 种用户是 MySQL 防止元数据被轻易篡改而默认创建的用户。

mysql.session 用户用于内部访问服务器，由于该用户已被锁定客户端，所以无法连接，通过 mysql.session 用户可以查看 MySQL 数据库中正在运行的所有进程。

在实际工作中有将 root 账户删除的做法，因为 root 账户权限过大，所以被入侵服务器时首先会通过 root 账户进行扫描。解决方案为将 root 账户删除或重命名，并建立一个特权账户进行管理，删除 root 账户是一种常见的数据库安全运维方式。

MySQL 版本不同，密码的加密方式也不同。MySQL 8.0 调整了账号认证方式，把 caching_sha2_password 插件认证方式作为默认首选，这就导致很多需要使用密码登录的客户端登录 MySQL 8.0 时发生错误，此时，需要将 caching_sha2_password 加密插件改成 mysql_native_password 加密插件这样客户就可正常使用了，SQL 语句如下：

```
ALTER USER 'root'@'%' IDENTIFIED WITH mysql_native_password BY 'root123';
```

有关于 user 的 plugin 加密插件不用过多关注，通常只有 MySQL 第 1 次登录时，或者第 1 次创建账号时需要注意一下。

在连接错误或者登录错误时都会含有加密插件错误的提示，根据提示可自行查询或更改。

3.5.2　用户的操作权限

通过 mysql.user 表可查询当前 MySQL 中用户含有哪些权限，SQL 语句如下：

```
select * from mysql.user;
```

查看用户权限中列的释义表如表 3-6 所示。

表 3-6 查看用户权限列的释义表

列 值	字段类型	是否为空	默认值	说 明
select_priv	enum('N','Y')	NO	N	是否可通过 SELECT 命令查询数据
insert_priv	enum('N','Y')	NO	N	是否可通过 INSERT 命令插入数据
update_priv	enum('N','Y')	NO	N	是否可通过 UPDATE 命令修改现有数据
delete_priv	enum('N','Y')	NO	N	是否可通过 DELETE 命令删除现有数据
create_priv	enum('N','Y')	NO	N	是否可创建新的数据库和表
drop_priv	enum('N','Y')	NO	N	是否可删除现有数据和表
grant_priv	enum('N','Y')	NO	N	是否可将自己的权限再授予其他用户
references_priv	enum('N','Y')	NO	N	是否可创建外键约束
index_priv	enum('N','Y')	NO	N	是否可对索引进行增、删、改操作
alter_priv	enum('N','Y')	NO	N	是否可重命名和修改表结构
execute_priv	enum('N','Y')	NO	N	是否可执行存储过程
repl_client_priv	enum('N','Y')	NO	N	是否可确定复制从服务器和主服务器的位置
create_view_priv	enum('N','Y')	NO	N	是否可创建视图
show_view_priv	enum('N','Y')	NO	N	是否可查看视图
event_priv	enum('N','Y')	NO	N	是否可创建、修改和删除事件
trigger_priv	enum('N','Y')	NO	N	是否可创建和删除触发器
……	……	……	……	…… MySQL 8.0 的 user 表共 51 个参数，涉及账户的权限、安全、资源控制等内容

3.5.3 表的操作权限

在 mysql.tables_priv 表中存储着对表操作的权限，查询 mysql.tables_priv 表的语句如下：

```
select * from mysql.tables_priv;
```

运行后，结果集如图 3-29 所示。

```
+-----------+-------+---------------+------------+---------------+---------------------+------------+-------------+
| Host      | Db    | User          | Table_name | Grantor       | Timestamp           | Table_priv | Column_priv |
+-----------+-------+---------------+------------+---------------+---------------------+------------+-------------+
| localhost | mysql | mysql.session | user       | boot@         | 2022-06-23 12:52:38 | Select     |             |
| localhost | sys   | mysql.sys     | sys_config | root@localhost | 2022-06-23 12:52:38 | Select     |             |
+-----------+-------+---------------+------------+---------------+---------------------+------------+-------------+
2 rows in set (0.07 sec)
```

图 3-29 表权限的查询结果

mysql. tables_priv 表的字段说明如表 3-7 所示。

表 3-7　mysql. tables_priv 表的字段说明

列　　值	释　　义
host	主机名
db	数据库实例
user	账户
table_name	表名
grantor	哪个账户授予的权限
timestamp	授权时间
table_priv	table_name 库中的哪张表
column_priv	table_priv 表的所有列被赋予了哪些权限

3.5.4　列的操作权限

在 mysql. columns_priv 表中存储着对列操作的权限。查询 mysql. columns_priv 表的语句如下：

```
select * from mysql.columns_priv;
```

mysql. columns_priv 表的字段说明如表 3-8 所示。

表 3-8　mysql. columns_priv 表的字段说明

列　　值	释　　义
host	主机名
db	数据库实例
user	账户
table_name	表名
column_name	列名
timestamp	授权时间
column_priv	table_priv 表的 column_name 列被赋予了哪些权限