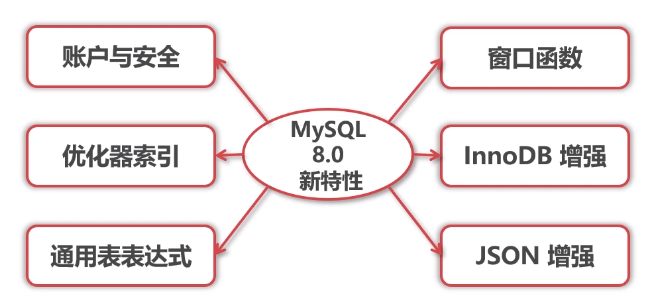
# 8.0新特性



# 账户与安全

## 用户创建和授权

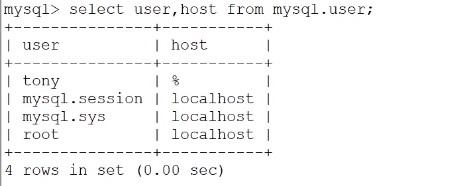
MySQL8.0用户创建和授权分开执行：

create user ‘tony’@’%’ identified by ‘Tony@2018’

grant all privileges on \*.\* to ‘tony’@’%’;

MySQL5.7用户创建和授权可以使用grant语句一次性完成：

grant all privileges on \*.\* to ‘tony’@’%’ identified by ‘passwd’;

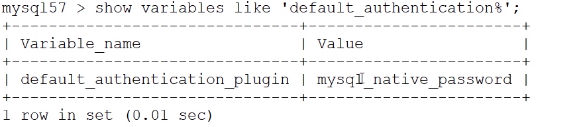


## 认证插件更新

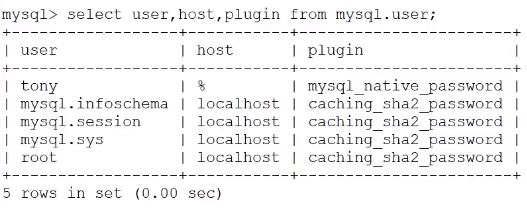
MySQL8.0中默认的身份认证插件是caching\_sha2\_password，替代了之前的mysql\_native\_password。

用户可以通过defalt\_authentication\_plugin或mysql.user表看到这种变化：

MySQL5.7：



MySQL8.0：



注：如果服务端mysql升级了，但是客户端没有升级，可能使用新的插件会导致连接错误。可以修改系统配置文件修改为原来的认证。

## 密码管理

MySQL8.0开始允许限制重复使用以前的密码：

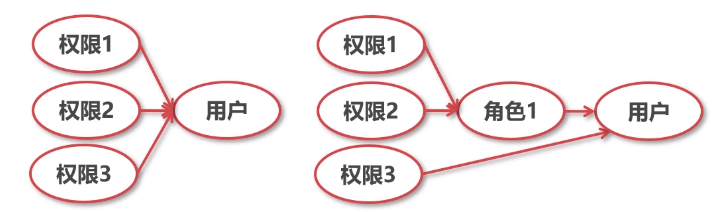
password\_history=3 //新密码不能和最近3次使用的相同

password\_reuse\_internal=90 //新密码不能和最近90天使用的密码相同

password\_require\_current=ON //修改密码时需要提供当前密码

## 角色管理

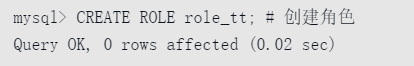
MySQL8.0提供了角色管理的新功能，角色是一组权限的集合。



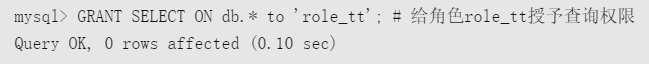
在MySQL 8.0数据库中，角色可以看成是一些权限的集合，为用户赋予统一的角色，权限的修改直接通过角色来进行，无须为每个用户单独授权。

操作步骤：

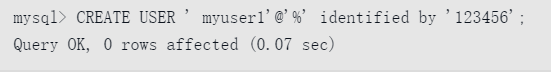
创建角色，执行语句如下：



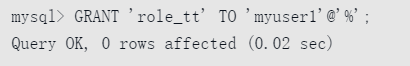
给角色授予权限，执行语句如下：



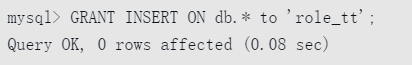
创建用户myuser1，执行语句如下：



为用户myuser1赋予角色role\_tt，执行语句如下：



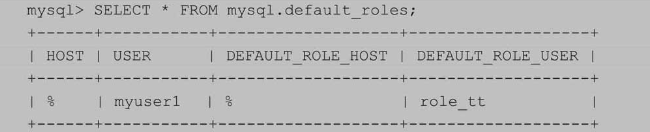
给角色role\_tt增加insert权限，执行语句如下：



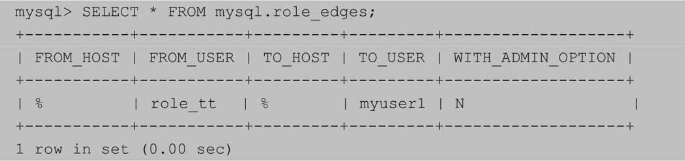
给角色role\_tt删除insert权限，执行语句如下：



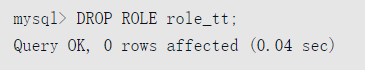
查看默认角色信息，执行语句如下：



查看角色与用户关系，执行语句如下：



删除角色，执行语句如下：



**全局变量持久化**

在MySQL数据库中，全局变量可以通过SET GLOBAL语句来设置。例如，设置服务器语句超时的限制，可以通过设置系统变量max\_execution\_time来实现：



使用SET GLOBAL语句设置的变量值只会临时生效。数据库重启后，服务器又会从MySQL配置文件中读取变量的默认值。

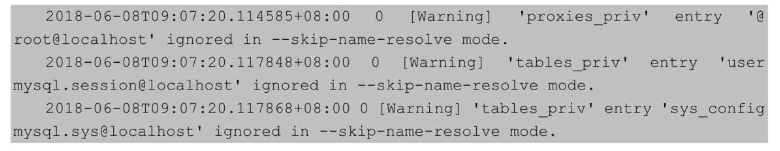
MySQL 8.0版本新增了SET PERSIST命令。例如，设置服务器的最大连接数为1000：



MySQL会将该命令的配置保存到数据目录下的mysqld-auto.cnf文件中，下次启动时会读取该文件，用其中的配置来覆盖默认的配置文件。

# 日志

在MySQL 8.0版本中，日志分类将更加详细。例如，在错误信息中添加了错误信息编号[MY-010311]和错误所属子系统[Server]。在MySQL 5.7版本中，部分错误日志如下：



在MySQL 8.0版本中，部分错误日志如下：



# 优化器索引

## 隐藏索引

MySQL8.0开始支持隐藏索引（invisible index），不可见索引。

隐藏索引不会被优化器使用，但仍然需要进行维护。

应用场景：软删除、灰度发布。

软删除：比如我们需要测试不使用某个索引对系统性能的影响，以前的方案是删除当前索引，如果有影响，需要这个索引就必须重新建立，这样维护成本太高了。在8.0版本可以将这个待测试的索引设置为隐藏索引，这样实际查询中就不会使用这个索引了，测试后如果真需要删除再执行删除索引操作。

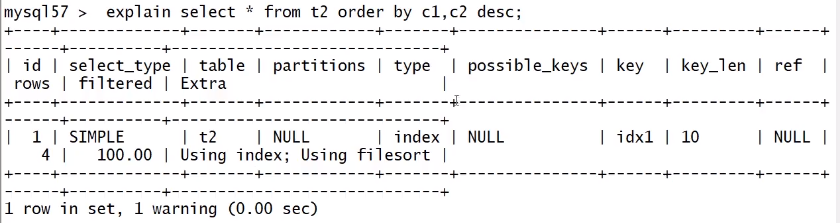
## 降序索引

在MySQL 8.0之前，MySQL在语法上已经支持降序索引，但实际上创建的仍然是升序索引。MySQL8.0开始真正支持降序索引（descending index）。

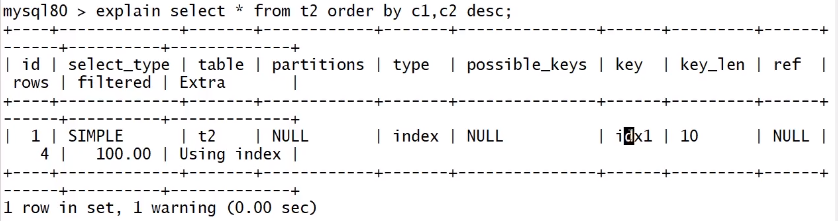
只有InnoDB存储引擎支持降序索引，只支持BTREE降序索引。

MySQL8.0不再对GROUP BY操作进行隐式排序。

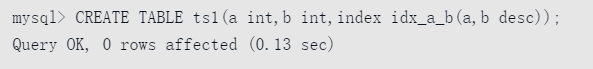
MySQL5.7：



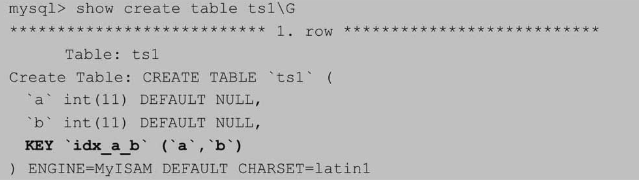
MySQL8.0：



分别在MySQL 5.7版本和MySQL 8.0版本中创建数据表ts1，结果如下：

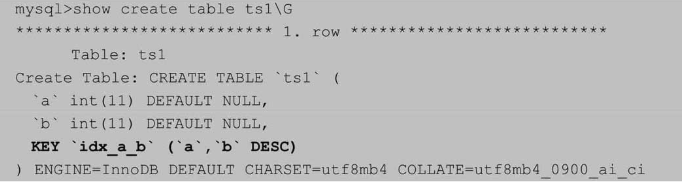


在MySQL 5.7版本中查看数据表ts1的结构，结果如下：



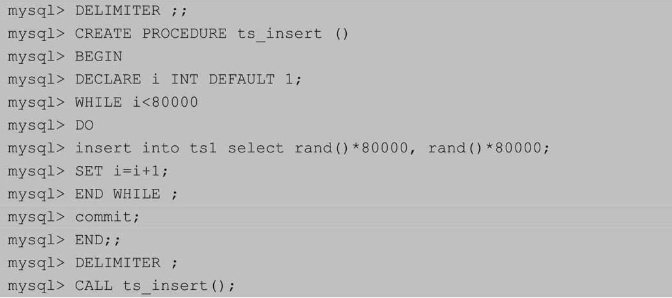
从结果可以看出，索引仍然是默认的升序。

在MySQL 8.0版本中查看数据表ts1的结构，结果如下：

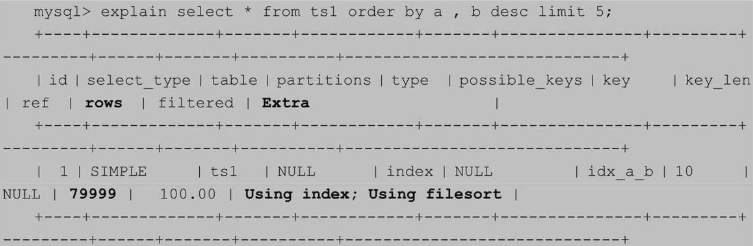


从结果可以看出，索引已经是降序了。下面继续测试降序索引在执行计划中的表现。

分别在MySQL 5.7版本和MySQL 8.0版本中的数据表ts1中插入8万条随机数据，执行语句如下：



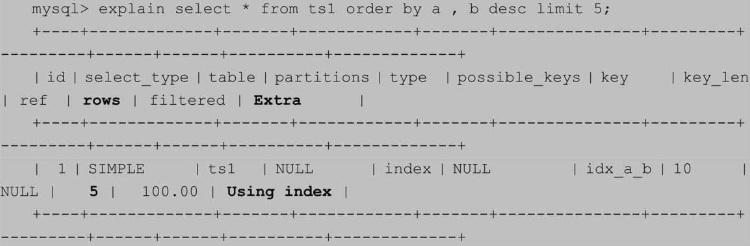
在MySQL 5.7版本中查看数据表ts1的执行计划，结果如下：



从结果可以看出，执行计划中扫描数为79999，而且使用了Using filesort。

**提示：**Using filesort是MySQL里一种速度比较慢的外部排序，如果能避免是最好的结果。多数情况下，管理员可以通过优化索引来尽量避免出现Using filesort，从而提高数据库执行速度。

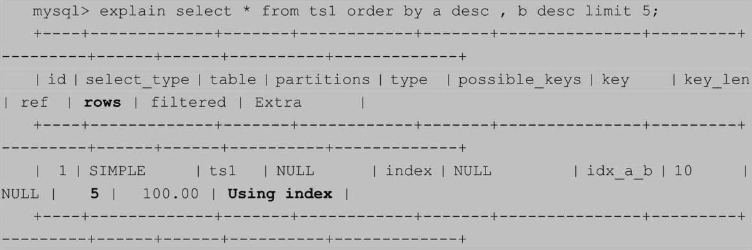
在MySQL 8.0版本中查看数据表ts1的执行计划，结果如下：



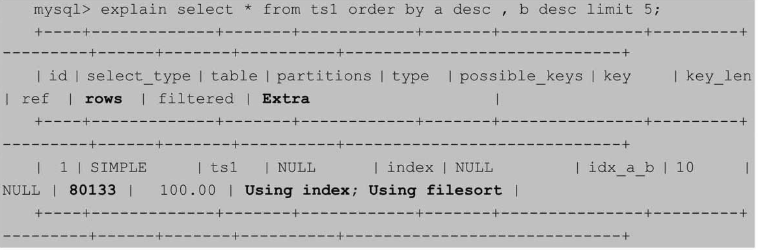
从结果可以看出，执行计划中扫描数为5，而且没有使用Using filesort。

**注意：**降序索引只是对查询中特定的排序顺序有效，如果使用不当，反而查询效率更低。例如上述查询排序条件改为“order by a desc, b desc”，MySQL 5.7的执行计划要明显好于MySQL 8.0。

将排序条件修改为“order by a desc, b desc”后，下面来对比不同版本中执行计划的效果。在MySQL 5.7版本中查看数据表ts1的执行计划，结果如下：



在MySQL 8.0版本中查看数据表ts1的执行计划，结果如下：



从结果可以看出，修改后MySQL 5.7的执行计划要明显好于MySQL 8.0。

## 函数索引

MySQL8/0/13开始支持在索引中使用函数（表达式）的值。

支持降序索引，支持JSON数据的索引。

函数索引基于虚拟列功能实现。

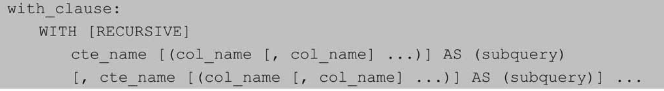
## 直方图

# 通用表达式

MySQL8.0开始支持通用表表达式（CTE），即WITH子句。

通用表表达式简称为CTE（Common Table Expressions）。CTE是命名的临时结果集，作用范围是当前语句。CTE可以理解成一个可以复用的子查询，当然跟子查询还是有点区别的，CTE可以引用其他CTE，但子查询不能引用其他子查询。

CTE的语法格式如下：



使用WITH语句创建CTE的情况如下：

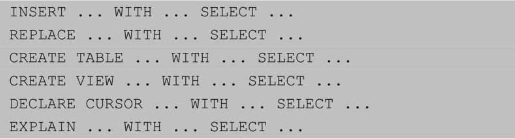
1. SELECT、UPDATE、DELETE语句的开头：



1. 在子查询的开头：



1. 紧接SELECT，在包含SELECT声明的语句之前：



## 非递归CTE

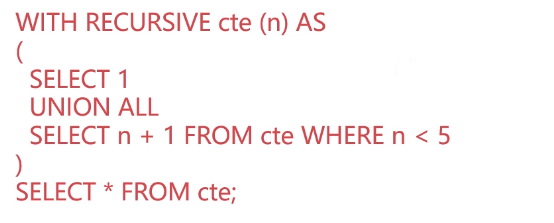


注：CTE更加清晰，后面可以任意实用前面定义的CTE表达式，更加方便，类似增加了一个编程功能的扩展。

## 递归CTE

WITH子句必须以WITH RECURSIVE开头。CTE递归子查询包括两部分：seed查询和recursive查询，中间由union [all]或union distinct分隔。seed查询会被执行一次，以创建初始数据子集。recursive查询会被重复执行以返回数据子集，直到获得完整结果集。当迭代不会生成任何新行时，递归会停止。

递归CTE在查询中引用自己的定义，实用RECURSIVE表示。



## 递归限制

递归表达式的查询中需要包含一个终止递归的条件：

cte\_max\_recursion\_depth：最大递归深度/调用次数

max\_execution\_time：SQL最长执行时间（没有定义最大递归深度的时候通过这个限制）

# 加密函数

加密函数主要用来对数据进行加密和界面处理，以保证某些重要数据不被别人获取。这些函数在保证数据库安全时非常有用。

## 加密函数MD5(str)

MD5(str)为字符串算出一个MD5 128比特校验和。该值以32位十六进制数字的二进制字符串形式返回，若参数为NULL，则会返回NULL。

## 加密函数SHA(str)

SHA(str)从原明文密码str计算并返回加密后的密码字符串，当参数为NULL时，返回NULL。SHA加密算法比MD5更加安全。

## 加密函数SHA2(str, hash\_length)

SHA2(str, hash\_length)使用hash\_length作为长度，加密str。hash\_length支持的值为224、256、384、512和0。其中，0等同于256。

# 窗口函数

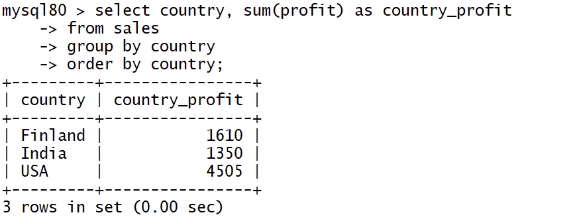
## 概述

MySQL8.0支持窗口函数（Window Function），也称分析函数。

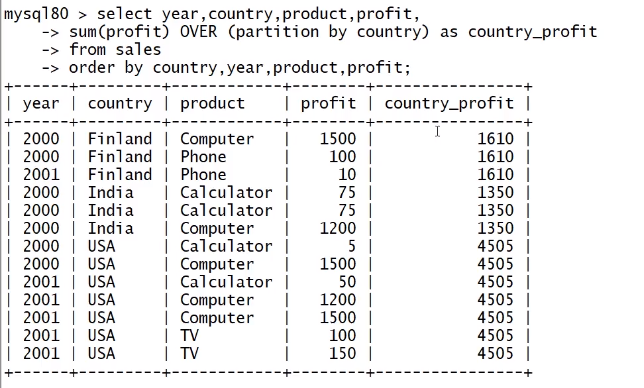
窗口函数与分组聚合函数类似，但是每一行数据都生成一个结果。

聚合窗口函数：SUM/AVG/COUNT/MAX/MIN等等。

常规操作：



窗口函数：



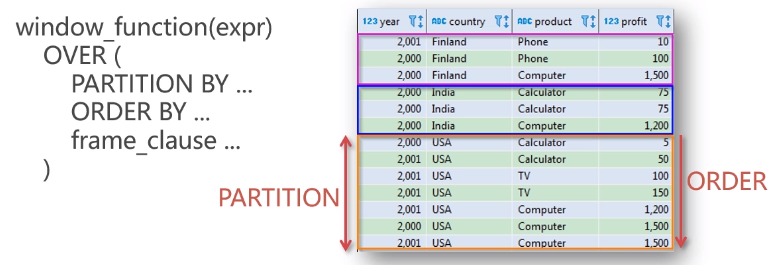
## 专用窗口函数

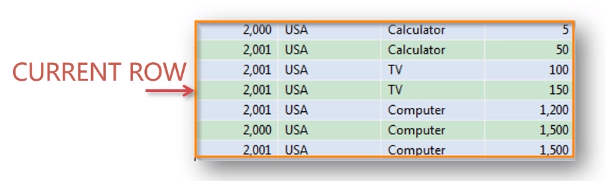
ROW\_NUMBER()/RANK()/DENSE\_RANK()/PERCENT\_RANK()

FIRST\_VALUE()/LAST\_VALUE()/LEAD()/LAG()

CUME\_DIST()/NTH\_VALUE()/NTILE()

## 窗口





# InnoDB增强

## 默认字符集改为utf8mb4

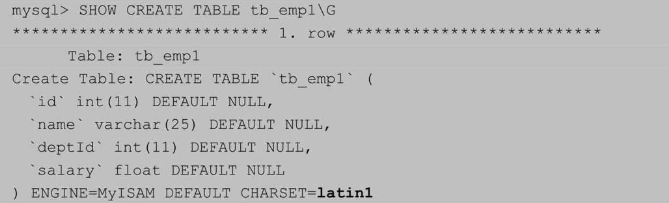
在MySQL 8.0版本之前，默认字符集为latin1，utf8字符集指向的是utf8mb3。网站开发人员在数据库设计的时候往往会将编码修改为utf8字符集。如果遗忘修改默认的编码，就会出现乱码的问题。从MySQL 8.0开始，数据库的默认编码改为utf8mb4，从而避免了上述的乱码问题。

下面通过案例来对比不同的版本中默认字符集的变化。

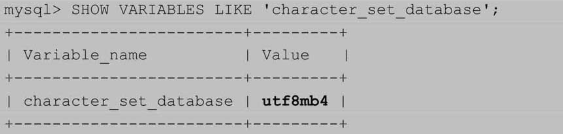
在MySQL 5.7版本中，查看数据库的默认编码，结果如下：



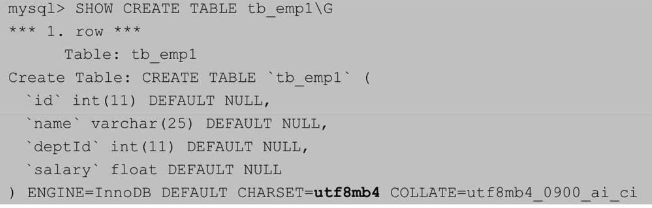
在MySQL 5.7版本中，查看数据表的默认编码，结果如下：



在MySQL 8.0版本中，测试数据库的默认编码，结果如下：



在MySQL 8.0版本中，查看数据表的默认编码，结果如下：

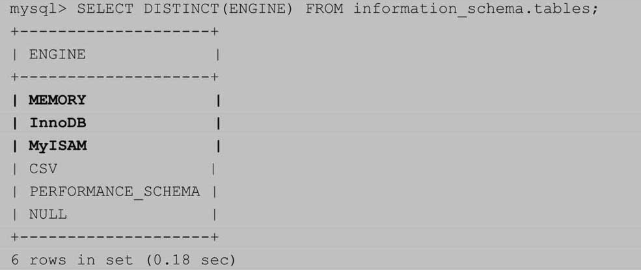


## 系统表全部为InnoDB表

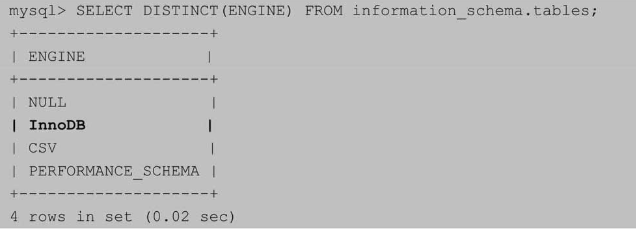
从MySQL 8.0开始，系统表全部换成事务型的InnoDB表，默认的MySQL实例将不包含任何MyISAM表，除非手动创建MyISAM表。

下面通过案例来对比不同的版本中系统表的变化。

在MySQL 5.7版本中查看系统表类型，结果如下：



在MySQL 8.0版本中查看系统表类型，结果如下：



## 集成数据字典

MySQL8.0删除了之前版本的元数据文件，例如.frm，.opt等。

将系统表（mysql）和数据字典全部改为InnoDB存储引擎。

简化了INFORMATION\_SCHEMA的实现，提高了访问性能。

提供了序列化字典信息（SDI）的支持，以及ibd2sdi工具。

数据字典使用上的差异，例如innodb\_read\_only影响所有的存储引擎；数据字典表不可见，不能直接查询和修改。

## 原子DDL操作

MySQL8.0开始支持原子DDL操作，其中与表相关的原子DDL只支持InnoDB存储引擎。DDL操作回滚日志写入到data dictionary数据字典表mysql.innodb\_ddl\_log（该表是隐藏的表，通过showtables无法看到）中，用于回滚操作。通过设置参数，可将DDL操作日志打印输出到MySQL错误日志中。

一个原子DDL操作内容包括：更新数据字典，存储引擎层的操作，在binlog中记录DDL操作。

支持与表相关的DDL：数据库、表空间、表、索引的CREATE、ALTER、DROP以及TRUNCATE TABLE。

支持的其他DDL：存储程序、触发器、视图、UDF（用户自定义函数）的CREATE、DROP以及ALTER语句。

支持账户管理相关的DDL：用户和角色的CREATE、ALTER、DROP以及适用的RENAME，以及GRANT和REVOKE语句。

## 自增列持久化

MySQL5.7以及早期版本，InnoDB自增列计数器（AUTO\_INCREMENT）的值只存储在内存中。

在MySQL 8.0之前，自增主键AUTO\_INCREMENT的值如果大于max(primary key)+1，在MySQL重启后，会重置AUTO\_INCREMENT=max(primary key)+1，这种现象在某些情况下会导致业务主键冲突或者其他难以发现的问题。

MySQL8.0每次变化时将自增计数器的最大值写入redo log，同时在每次检查点将其写入引擎私有的系统表。

解决了长期以来的自增字段值可能重复的bug。

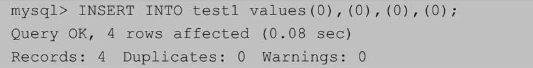
下面通过案例来对比不同的版本中自增变量是否持久化。

在MySQL 5.7版本中，测试步骤如下：

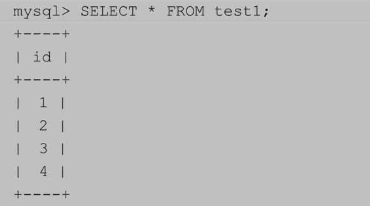
创建的数据表中包含自增主键的id字段，语句如下：



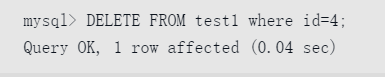
插入4个空值，执行如下：



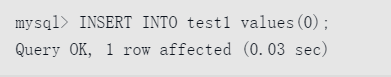
查询数据表test1中的数据，结果如下：



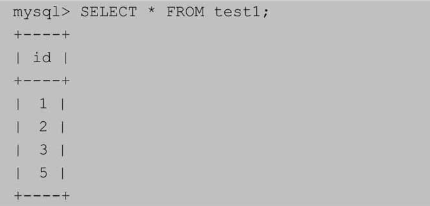
删除id为4的记录，语句如下：



再次插入一个空值，语句如下：

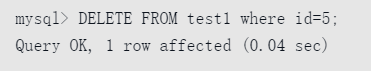


查询此时数据表test1中的数据，结果如下：

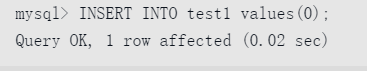


从结果可以看出，虽然删除了id为4的记录，但是再次插入空值时，并没有重用被删除的4，而是分配了5。

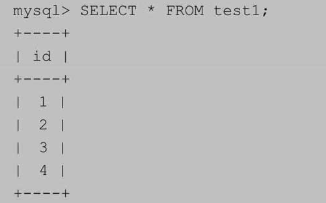
删除id为5的记录，结果如下：



重启数据库，重新插入一个空值。



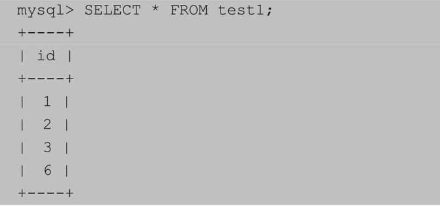
再次查询数据表test1中的数据，结果如下：



从结果可以看出，新插入的0值分配的是4，按照重启前的操作逻辑，此处应该分配6。出现上述结果的主要原因是自增主键没有持久化。在MySQL 5.7系统中，对于自增主键的分配规则，是由InnoDB数据字典内部一个计数器来决定的，而该计数器只在内存中维护，并不会持久化到磁盘中。当数据库重启时，该计数器会通过下面这种方式初始化。



在MySQL 8.0版本中，上述测试步骤最后一步的结果如下：



从结果可以看出，自增变量已经持久化了。下面讲述MySQL 8.0的解决方案。

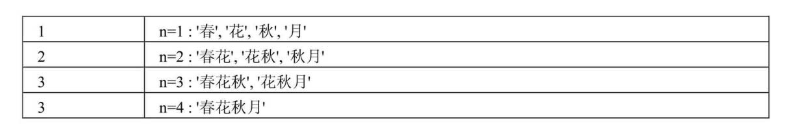
MySQL 8.0将自增主键的计数器持久化到重做日志中。每次计数器发生改变，都会将其写入重做日志中。如果数据库重启，InnoDB会根据重做日志中的信息来初始化计数器的内存值。**为了尽量减小对系统性能的影响，计数器写入到重做日志时并不会马上刷新数据库系统**。

## 全文索引加强

MySQL 8.0支持更加灵活、更加优化的全文搜索。例如，全本索引支持外部的分析器，就像MyISAM。插件可以替代内置分析器，也可以作为一个前端来使用。MySQL 8.0实现了标记优化器，这个优化器可以将查询结果传递到InnoDB，因此InnoDB可以跳过全文检索部分。

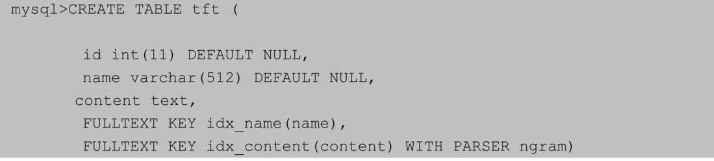
在InnoDB上实现了支持CJK（中文、日文和韩文）的全文检索。MySQL 8.0为CJK提供了一个默认的全文分析器（N-GRAM分析器）。

在全文索引中，n-gram就是一段文字里面连续的n个字的序列。例如，用n-gram来对“春花秋月”进行分词，得到的结果如表21.18所示。其中，n由参数ngram\_token\_size控制，即分词的大小，默认是2。



下面通过举例来说明全文搜索功能。

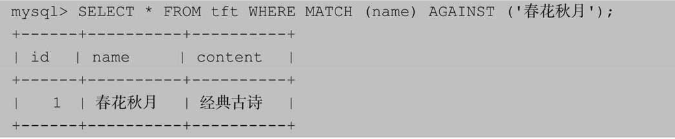
创建数据表，并设置全文检索。SQL语句如下：



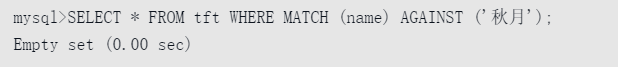
插入演示数据，SQL语句如下：



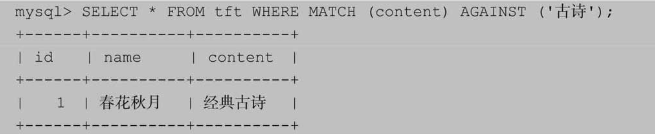
普通检索必须要是整个词才能检索到，SQL语句如下：



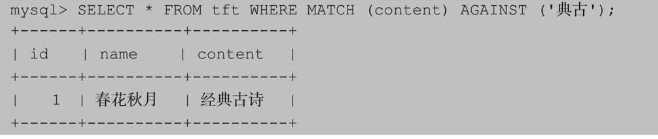
部分词是不能检索出信息的，SQL语句如下：



新的全文检索功能检索任意两个组合的记录，SQL语句如下：



再次使用全文检索功能检索任意两个组合记录，SQL语句如下：



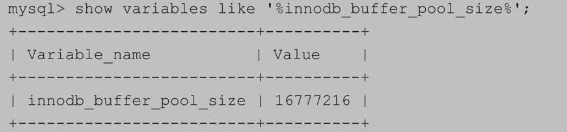
## 动态修改InnoDB缓冲池的大小

从MySQL 5.7.5版本起，MySQL支持在不重启系统的情况下动态调整innodb\_buffer\_pool\_size。调整大小的过程是以innodb\_buffer\_pool\_chunk\_size为单位迁移pages到新的内存空间，迁移进度可以通过Innodb\_buffer\_pool\_resize\_status查看。当在线修改缓冲池大小的时候，以chunk为单位进行增长或收缩。

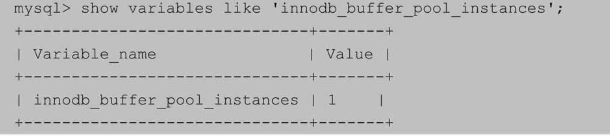
缓冲池大小是innodb\_buffer\_pool\_chunk\_size\*innodb\_buffer\_pool\_instances的倍数（128MB），如果不是，将会适当调大innodb\_buffer\_pool\_size，以满足要求。因此，可能会出现缓冲池大小的实际分配比配置文件中指定的size要大的情况。

下面举例说明如何在线调整缓冲池的大小。

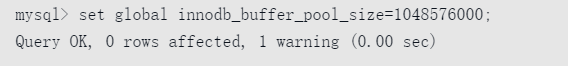
查看当前缓冲池的大小，SQL语句如下：



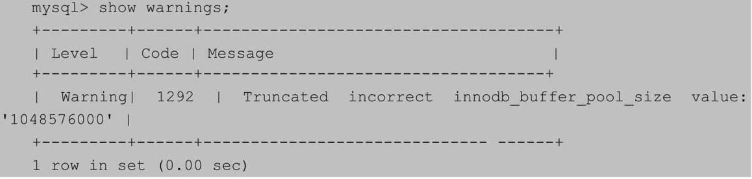
查看缓冲池中实例的个数，SQL语句如下：



动态修改缓冲池的大小为1000MB，SQL语句如下：

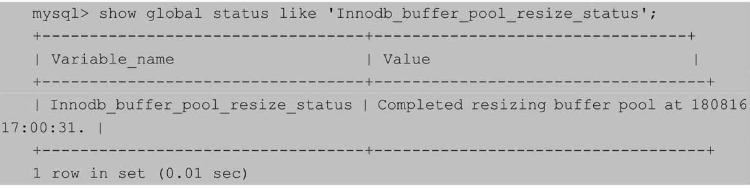


查看警告信息，SQL语句如下：

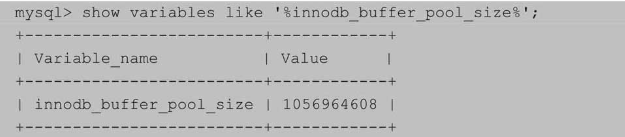


出现上述警告信息的原因是，设置1000MB不是innodb\_buffer\_pool\_chunk\_size\*innodb\_buffer\_pool\_instances的倍数，即128MB的倍数。

查看设置缓冲池的进度，SQL语句如下：



查看当前缓冲池的大小，SQL语句如下：

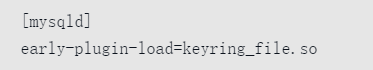


从结果可以看出，缓冲池的大小被设置成了1056964608字节（约1024MB），因为1024是128的整数倍，出现了缓冲池大小比配置文件里指定的size还大。

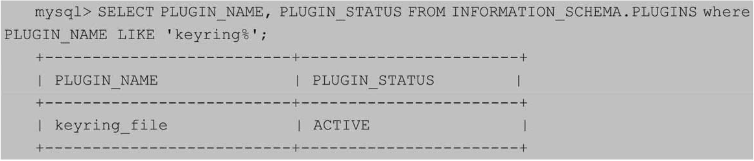
## 表空间数据加密

在MySQL 8.0中，InnoDB Tablespace Encryption支持对独享表空间的InnoDB数据文件加密，其依赖keyring plugin来进行秘钥的管理。开启加密功能需要启动参数--early-plugin-load。

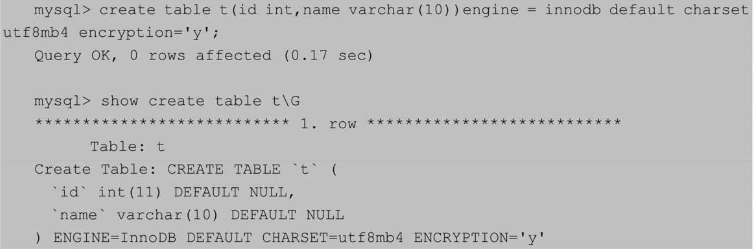
在PHP的配置文件my.ini中开启--early-plugin-load参数。



启动参数后，查看服务器是否支持加密功能：



创建加密表空间：



## 死锁检查控制

MySQL8.0（MySQL5.7.15）增加了一个新的动态变量（innodb\_deadlock\_detect），用于控制系统是否执行InnoDB死锁检查。

对于高并发的系统，禁用死锁检查可能带来性能的提高。

## 锁定语句选项/跳过锁等待

SELECT…FOR SHARE和SELECT…FOR UPDATE中支持NOWAIT、SKIP LOCKED选项：

对于NOWAIT，如果请求的行被其他事务锁定时，语句立即返回。

对于SKIP LOCKED，从返回的结果集中移除被锁定的行。

在MySQL 5.7版本中，SELECT...FOR UPDATE语句在执行的时候，如果获取不到锁，会一直等待，直到innodb\_lock\_wait\_timeout超时。

在MySQL 8.0版本中，通过添加NOWAIT和SKIP LOCKED语法，能够立即返回。如果查询的行已经加锁，那么NOWAIT会立即报错返回，而SKIP LOCKED也会立即返回，只是返回的结果中不包含被锁定的行。

下面通过案例来理解MySQL 8.0版本中如何跳过锁等待，如表21.19所示。

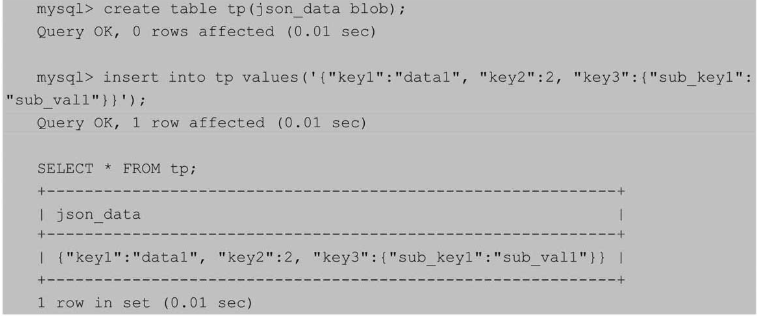


## 其他改进功能

# JSON增强

MySQL是一个关系型数据库，在MySQL 8.0之前，没有提供对非结构化数据的支持，但是如果用户有这样的需求，也可以通过MySQL的BLOB来存储非结构化的数据。

举例说明：



在本例中，使用BLOB来存储JSON数据，需要用户保证插入的数据是一个能够转换成JSON格式的字符串，因为MySQL并不保证任何正确性。在MySQL看来，这就是一个普通的字符串，并不会进行任何有效性检查。此外，提取JSON中的字段也需要在用户的代码中完成。

例如，在Python语言中提取JSON中的字段，代码如下：



这种方式虽然也能够实现JSON的存储，但是有诸多缺点，最为显著的缺点有：

●　需要用户保证JSON的正确性，如果用户插入的数据并不是一个有效的JSON 字符串，MySQL 并不会报错。

●　所有对JSON的操作，都需要在用户的代码里进行处理，不够友好。

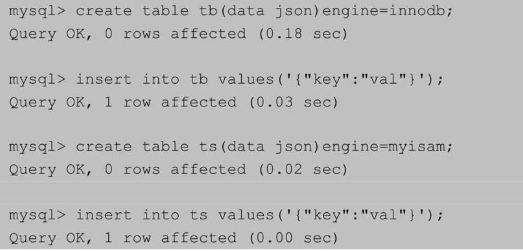
●　即使只是提取JSON中的某一个字段，也需要读出整个BLOB，效率不高。

●　无法在JSON字段上建索引。

在MySQL 8.0中，已经实现了对JSON类型的支持。MySQL本身已经是一个比较完备的数据库系统，对于底层存储并不适合有太大的改动，那么MySQL是如何支持JSON格式的呢？

MySQL 8.0对支持JSON的做法是在Server层提供一些便于操作JSON的函数，简单地将JSON编码成BLOB，然后交由存储引擎层进行处理。MySQL 8.0的JSON支持与存储引擎没有关系，MyISAM存储引擎也支持JSON格式。

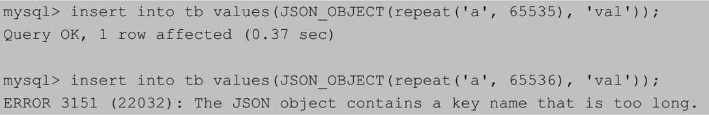
下面将举例说明：



MySQL 8.0提供了很多操作JSON的函数，都是为了提高易用性。

MySQL编码成BLOB对象，首先存放的是JSON的元素个数，然后存放的是转换成BLOB以后的字节数，接下来存放的是key pointers和value pointers。为了加快查找速度，MySQL内部会对key进行排序，以提高处理速度。

在MySQL 8.0中，key的长度只用2个字节（65535）保存，如果超过这个长度，MySQL将报错，如下所示：



在MySQL的源码中，与JSON相关的文件有：

●　json\_binary.cc

●　json\_binary.h

●　json\_dom.cc

●　json\_dom.h

●　json\_path.cc

●　json\_path.h

其中，json\_binary.cc处理JSON的编码、解码，json\_dom.cc是JSON的内存表示，json\_path.cc用于将字符串解析成JSON。对于JSON的编码，入口是json\_binary.cc文件中的serialize函数。

对于JSON的解码，即将BLOB解析成JSON对象，入口是json\_binary.cc文件中的parse\_binary函数。只要搞清楚了JSON的存储格式，这两个函数是很好理解的。

## 内联路径操作符

## JSON聚合函数

## JSON实用函数

## JSON合并函数

## JSON表函数