# 概述

关系数据库通常包含多个表。数据库实际是表的集合，数据库的数据或信息都是存储在表中的。表是对数据进行存储和操作的一种逻辑结构。对用户而言，一个表表示一个数据库对象。

数据库中的表与人们日常生活中使用的表格类似，也是由行（Row）和列（Column）组成。列由同类的信息组成，每列又称为一个字段，每列的标题称为字段名。行包括了若干列信息项，一行数据称为一条记录，表达有一定意义的信息组合。一个表由一条或多条记录组成，没有记录的表称为空表。每个表中通常都有一个主关键字，用于唯一地确定一条记录。经常见到的成绩表就是一种表，它由行和列组成，并且可以通过名字来识别数据，列包含了列的名字、数据类型以及列的其他属性；行包含了列的记录或者数据。

# 欢迎界面

命令的结束符，用“;”或者“\g”结束。

客户端的连接ID，这个数字记录了MySQL服务到目前为止的连接次数；每个新连接都会自动加1。MySQL 服务器的版本，如果是“5.1.9-beta-log”，说明是 5.1.9 的测试版；如果是标准版，则会用standard代替beta。

通过“help;”或者“\h”命令来显示帮助内容，通过“\c”命令来清除命令行buffer。

在mysql>提示符后面输入所要执行的SQL语句，每个SQL语句以分号（；）或者“\g”结束，按回车键执行。

# 表分类

在SQL中，并不是所有的表都是相同的。有些表是永久的，有些表则是临时的。有些表是模式对象，而有些表则包含在模块中，所有的模块表也是临时的。

## 永久表

永久表保存存储在数据库中的SQL数据。它是一种最常见的表，如果没有特别说明，通常所说的表就是指永久表。只要表的定义存在，永久表就始终存在。它的创建语句为CREAT TABLE。

## 临时表

参考：

<https://dev.mysql.com/doc/refman/5.7/en/create-temporary-table.html>

### 全局临时表

这种表只有在SQL会话的上下文引用该表的定义时实际的表才会存在，对话结束后表就不再存在，不能从一个SQL会话访问在另一个会话中创建的表。

全局临时表的创建语句为CREAT GLOBAL TEMPORARY TABLE。

### 局部临时表

和全局临时表一样，局部临时表只有在SQL会话的过程中才能被引用，并且不能从另一个SQL会话对其进行访问。

而与全局临时表不同之处在于：我们在SQL会话内的任何地方都可以访问全局临时表；而局部临时表只有在相关的SQL模块内才能被访问。

局部临时表的创建语句为CREAT LOCAL TEMPORARY TABLE。

当操作非常大的表时，你可能偶尔需要运行很多查询获得一个大量数据小的子集，不是对整个表运行这些查询，而是让MySQL每次找出所需的少数记录，将记录选择到一个临时表可能更快些，然后再这些表上运行查询。

创建临时表很容易，给正常的CREATE TABLE语句加上TEMPORARY关键字即可。

临时表将在连接MySQL期间存在，当你断开的时候，MySQL将自动删除表并释放所有空间，当然你可以在仍然连接的时候删除表并释放空间。

如果在你创建名为tmp\_table临时表时名为tmp\_table的表在数据库中已经存在，临时表将有必要屏蔽（隐藏）非临时表tmp\_table。

如果声明临时表是一个HEAP表，MySQL允许你指定在内存中创建它：

CREATE TEMPORARY TABLE tmp\_table(

)TYPE=HEAP;

因为HEAP表存储在内存中，对它运行的查询比磁盘上的临时表快些。然而，HEAP表与一般的表有些不同，且有自身的限制。

# 派生表/虚表

当主查询中包含派生表，或者当select语句中包含union子句，或者select语句语句中包含一个字段的order by子句（对另一个字段的group by子句）时，MySQL为了完成查询，则需要自动创建临时表存储临时结果集，这种临时表是由MySQL自行创建，自行维护。

对于自动创建的临时表，由于内存临时表的性能更加优越，MySQL总是首先使用内存临时表，而当内存临时表变得太大时，达到某个预知的时候，内存临时表就转存为外存临时表。也就是说，外存临时表是内存临时表在存储空间上的一种延伸。内存临时表转存为外存临时表的阈值由系统变量max\_heap\_table\_size和tmp-table\_size的较小值决定。

派生表是从select语句返回的虚拟表，派生表类似于临时表，但是在select语句中使用派生表比临时表简单的多，因此它不需要创建临时表的步骤，所以当select语句的from子句中使用独立子查询时，我们将其称为派生表。

派生表一般在from子句中使用，如：

select \* from (select \* from table) table1;

# 元数据表

在日常工作中，我们经常会遇到类似下面的应用场景：

删除数据库test1下所有前缀为tmp的表；

将数据库test1下所有存储引擎为myisam的表改为innodb。

对于这类需求，在MySQL 5.0之前只能通过 show tables、show create table或者 show tablestatus 等命令来得到指定数据库下的表名和存储引擎，但这些命令显示内容有限且不适合进行字符串的批量编辑。如果表很多，则操作起来非常低效。

MySQL 5.0之后，提供了一个新的数据库 information\_schema，用来记录MySQL中的元数据信息。元数据指的是数据的数据，比如表名、列名、列类型、索引名等表的各种属性名称。这个库比较特殊，它是一个虚拟数据库，物理上并不存在相关的目录和文件；库里 show tables显示的各种“表”也并不是实际存在的物理表，而全部是视图。对于上面的两个需求，可以简单地通过两个命令得到需要的SQL语句：

select concat('drop table test1.',table\_name,';') from tables where table\_schema='test1' and table\_name like 'tmp%';

select concat('alter table test1.',table\_name,' engine=innodb;') from tables wheretable\_schema='test1' and engine='MyISAM';

下面列出一些比较常用的视图。

SCHEMATA：该表提供了当前mysql实例中所有数据库的信息，show databases的结果取之此表。

TABLES：该表提供了关于数据库中的表的信息（包括视图），详细表述了某个表属于哪个schema、表类型、表引擎、创建时间等信息。show tables from schemaname的结果取之此表。

COLUMNS：该表提供了表中的列信息，详细表述了某张表的所有列以及每个列的信息。show columns from schemaname.tablename的结果取之此表。

STATISTICS：该表提供了关于表索引的信息。show index from schemaname.tablename的结果取之此表。

# 操作

## 数据库操作

### 创建

CREATE DATABASE test;

另外 4 个数据库，它们都是安装MySQL时系统自动创建的，其各自功能如下。

information\_schema：主要存储了系统中的一些数据库对象信息，比如用户表信息、列信息、权限信息、字符集信息、分区信息等。

cluster：存储了系统的集群信息。

mysql：存储了系统的用户权限信息。

test：系统自动创建的测试数据库，任何用户都可以使用。

### 删除

删除数据库的语法很简单，如下所示：drop database dbname;

**注意：**数据库删除后，下面的所有表数据都会全部删除，所以删除前一定要仔细检查并做好相应备份。

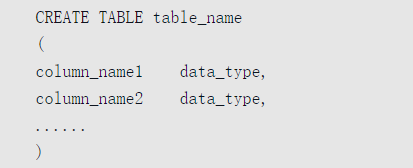
## 表操作

### 创建

常用的创建数据库表的方法有两种：一种是通过数据库管理系统（DBMS）提供的交互式创建工具创建，另一种是通过SQL直接创建。

#### CREATE TABLE

在SQL中，创建数据库表的基本关键字为Create Table，在其后要指明创建的数据库表的名称，接着要分别定义表中各列的名称、数据类型等。语法如下：



表的名字对大小写不敏感。表名要紧接在Create Table关键词的后面，且第一个字符必须是A～Z之一，其余的字符可以是字母，也可以是“\_”、“#”、“$”和“@”等符号。表中各列的定义在括号中完成，且各列之间以逗号隔开。不同的表，其列名可以相同，但是在同一个表中，不允许出现相同的列名。在定义了列名后，我们一定要指明该列的数据类型。

#### CREATE LIKE

创建的表不会丢失索引信息。

#### CREATE AS SELECT

创建表的同时插入数据，但是这种方式创建的表会丢失索引信息，不建议使用。

参考：

<https://mp.weixin.qq.com/s?__biz=MzI1OTU2MDA4NQ==&mid=2247489202&idx=1&sn=d02bd20bb31f6f563013049f2bbf900f&chksm=ea765148dd01d85e97c6535655641ae336e3da43608d3ca1a4506790f1ed2278faa001c71f47&mpshare=1&scene=24&srcid=0311SVyjA00yyiXosDoSLCFH&sharer_sharetime=1615462574981&sharer_shareid=33f795d236f19ac7c128b2e279563f84#rd>

SQL语句“create table <table\_name> as select ...”用于创建普通表或临时表，并物化select的结果。某些应用程序使用这种结构来创建表的副本。一条语句完成所有工作，因此您无需创建表结构或使用其他语句来复制结构。

与此同时，这种语句存在许多问题：

1、您不为新表创建索引

2、您在一个事务中混合了事务性和非事务性语句时，与任何DDL一样，它将提交当前和未完成的事务

3、使用基于GTID的复制时不支持 CREATE TABLE ... SELECT

4、在语句完成之前，元数据锁不会释放

##### 事务问题

CREATE TABLE AS SELECT语句可以把事物变得很糟糕

让我们想象一下，我们需要将钱从一个账户转移到另一个账户（经典示例）。但除了转移资金外，我们还需要计算费用。开发人员决定创建一个表来执行复杂的计算。

然后事务看起来像这样：

sql  
 begin;  
 update accounts set amount = amount - 100000 where account\_id=123;  
 -- now we calculate fees  
 create table as select ... join ...  
 update accounts set amount = amount + 100000 where account\_id=321;  
 commit;

"create table as select ... join ..."会提交一个事务，这是不安全的。如果出现错误，第二个帐户显然不会被已经提交的第二个帐户借记贷记！

我们可以使用"create temporary table …"来修复问题，而不是"create table … "，因为允许临时表创建。

##### GTID问题

如果在启用GTID时尝试使用CREATE TABLE AS SELECT（并且ENFORCE\_GTID\_CONSISTENCY = 1），则会出现此错误：

General error: 1786 CREATE TABLE ... SELECT is forbidden when @@GLOBAL.ENFORCE\_GTID\_CONSISTENCY = 1.

 应用程序代码可能会中断。

##### 元数据锁问题

CREATE TABLE AS SELECT的元数据锁定问题鲜为人知。（[有关元数据锁定的更多信息](https://dev.mysql.com/doc/refman/5.7/en/metadata-locking.html)）。请注意：MySQL元数据锁与InnoDB死锁、行级锁、表级锁是不同的。

以下速模拟演示了元数据锁定：

\*\*会话1:\*\*

mysql> create table test2 as select \* from test1;

\*\*会话2:\*\*

mysql> select \* from test2 limit 10;-- blocked statement语句被阻塞  
 This statement is waiting for the metadata lock:此语句正在等待元数据锁：

\*\*会话3:\*\*

mysql> show processlist;

+----+------+-----------+------+---------+------+---------------------------------+-------------------------------------------  
 | Id | User | Host | db | Command | Time | State | Info  
 +----+------+-----------+------+---------+------+---------------------------------+-------------------------------------------  
 | 2 | root | localhost | test | Query | 18 | Sending data | create table test2 as select \* from test1  
 | 3 | root | localhost | test | Query | 7 | Waiting for table metadata lock | select \* from test2 limit 10  
 | 4 | root | localhost | NULL | Query | 0 | NULL | show processlist  
+----+------+-----------+------+---------+------+---------------------------------+-------------------------------------------

同样地，可以采用另一种方式：慢查询可以阻塞某些DDL操作（即重命名，删除等）：

mysql> show processlistG  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 Id: 4  
 User: root  
 Host: localhost  
 db: reporting\_stage  
 Command: Query  
 Time: 0  
 State: NULL  
 Info: show processlist  
 Rows\_sent: 0  
 Rows\_examined: 0  
 Rows\_read: 0  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 Id: 5  
 User: root  
 Host: localhost  
 db: test  
 Command: Query  
 Time: 9  
 State: Copying to tmp table  
 Info: select count(\*), name from test2 group by name order by cid  
 Rows\_sent: 0  
 Rows\_examined: 0  
 Rows\_read: 0  
 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 Id: 6  
 User: root  
 Host: localhost  
 db: test  
 Command: Query  
 Time: 5  
 State: Waiting for table metadata lock  
 Info: rename table test2 to test4  
 Rows\_sent: 0  
 Rows\_examined: 0  
 Rows\_read: 0  
 3 rows in set (0.00 sec)

我们可以看到，CREATE TABLE AS SELECT可以影响其他查询。但是，这里的问题不是元数据锁本身（需要元数据锁来保持一致性）。问题是在语句完成之前不会释放元数据锁。

修复很简单：首先复制表结构，执行“ create table new\_table like old\_table”，然后执行“insert into new\_table select ...”。元数据锁仍然在创建表部分（非常短）持有，但“insert … select”部分不会持有（保持锁定的总时间要短得多）。

为了说明不同之处，让我们看看以下两种情况：

使用“create table table\_new as select ... from table1 ”，其他应用程序连接在语句的持续时间内无法读取目标表（table\_new）（甚至“show fields from table\_new”将被阻塞）

使用“create table new\_table like old\_table”+“insert into new\_table select ...”，在“insert into new\_table select ...”这部分期间，其他应用程序连接无法读取目标表。

然而，在某些情况下，表结构事先是未知的。例如，我们可能需要物化复杂select语句的结果集，包括joins、and/or、group by。在这种情况下，我们可以使用这个技巧：

create table new\_table as select ... join ... group by ... limit 0;  
 insert into new\_table as select ... join ... group by ...

第一个语句创建一个表结构，不插入任何行（LIMIT 0）。第一个语句持有元数据锁。但是，它非常快。第二个语句实际上是在表中插入行，而不持有元数据锁。

### 查看

使用SQL语句创建好数据表之后，可以查看表结构的定义，以确认表的定义是否正确。在MySQL中，查看表结构可以使用DESCRIBE和SHOW CREATE TABLE语句。

### 修改

在实际设计和创建数据库表的时候，我们很难做到一步到位，往往需要在使用的过程中，不断地修改完善。在SQL中，我们可以采用ALTER TABLE命令来修改已经创建的表结构。使用ALTER TABLE命令可以向表中增加新列、删除已有的列、也可以修改已经创建的列。

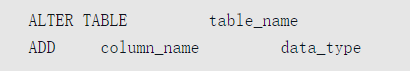
注意：对表定义的修改，不同的数据库系统有不同的限制。例如，Oracle数据库就限制对列的修改只能是加大列的宽度而不能是缩小，而且不能删除列。

对于已经创建好的表，尤其是已经有大量数据的表，如果需要做一些结构上的改变，可以先将表删除（drop），然后再按照新的表定义重建表。这样做没有问题，但是必然要做一些额外的工作，比如数据的重新加载。而且，如果有服务在访问表，也会对服务产生影响。因此，在大多数情况下，表结构的更改都使用 alter table语句。

**注意：**change和modify都可以修改表的定义，不同的是change后面需要写两次列名，不方便。**但是change的优点是可以修改列名称，modify则不能**。

**注意：**使用ALTER TABLE要极为小心，应该在进行改动前做一个完整的备份（模式和数据的备份）。数据库表的更改不能撤销，如果增加了不需要的列，可能不能删除它们。类似地，如果删除了不应该删除的列，可能会丢失该列中的所有数据

#### 增加新列



table\_name指的是要修改的表的名字，ADD关键字后面接要创建列的列名、数据类型等，当然也可以对列设置非空约束和缺省值。

当用ALTER TABLE语句向表中添加新列时，DBMS向表的列定义的尾部添加列，即在查询中将位于表的最右边。除非指定默认值，DBMS为已有行上的新列设NULL值。

由于DBMS为已有行上的新列设NULL值，当使用ALTER TABLE语句向表中添加新列时，我们不能简单地添加NOT NULL约束，还必须提供缺省值。因为如果没有提供缺省值，DBMS假设已有行上的新列为NULL值，这就和NOT NULL约束相抵触。当然，如果表中不存在数据，则不存在这个问题。

#### 删除列

同样，在使用数据库表的过程中，如果其某列信息已经无效或不再需要，为了节省数据库空间，提高查询性能，我们可以采用DROP COLUMN关键字删除表中的某列，语法如下：

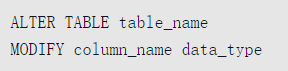


table\_name是要修改的表的名字，DROP COLUMN关键字后面接要删除列的名字。当然，一次可以删除多个列，只需要在DROP COLUMN关键字后面依次列出要删除的列的名字，中间用逗号分开即可。

#### 修改列

##### MODIFY

如果发现数据库表中某列的结构不能满足实际的需求，在不破坏数据的情况下，SQL允许利用MODIFY关键字修改表中某列的结构。常用的修改操作主要包括字符长度限制的修定和非空约束的限制或取消，语法如下：



table\_name是要修改的表的名字，MODIFY关键字后面接要修改列的列名和修改后的数据条件。

**添加NOT NULL约束语法**

使用ALTER TABLE给某列添加NOT NULL约束 的基本语法如下：

ALTER TABLE table\_name

MODIFY column\_name datatype NOT NULL;

**注意：**

在SQL Server数据库系统中，并不支持MODIFY关键字。要修改数据库中的列，我们可以通过ALTER COLUMN关键字实现，即将MODIFY替换为ALTER COLUMN即可。

用户可以通过MODIFY关键字或者ALTER COLUMN关键字增加或减少表中某列的最多字符数，但是，当要减少表中某列的最多字符数要特别慎重。当数据库表中该列存在已有记录的字符数多于减少后的最多字符限制时，表的修改就会失败。

当然，我们也可以通过MODIFY或者ALTER COLUMN关键字增加或取消表中某列的非空约束。

##### CHANGE

语法格式：

1. 字段改名

ALTER TABLE tablename CHANGE [COLUMN] old\_col\_name column\_definition[FIRST|AFTER col\_name]

**注意：**change和modify都可以修改表的定义，不同的是change后面需要写两次列名，不方便。但是change的优点是可以修改列名称，modify则不能。

1. 修改字段排序顺序

前面介绍的字段增加和修改语法（ADD/CHANGE/MODIFY）中，都有一个可选项first|aftercolumn\_name，这个选项可以用来修改字段在表中的位置，ADD增加的新字段默认是加在表的最后位置，而CHANGE/MODIFY默认都不会改变字段的位置。

**注意：**CHANGE/FIRST|AFTER COLUMN这些关键字都属于MySQL在标准 SQL上的扩展，在其他数据库上不一定适用。

#### 增加约束

##### 添加NOT NULL约束语法

使用ALTER TABLE给某列添加NOT NULL约束的基本语法如下：

ALTER TABLE table\_name

MODIFY column\_name datatype NOT NULL;

##### 添加唯一约束语法

使用ALTER TABLE给数据表添加唯一约束的基本语法如下：

ALTER TABLE table\_name

ADD CONSTRAINT MyUniqueConstraint

UNIQUE(column1, column2...);

##### 添加CHECK约束语法

使用ALTER TABLE给数据表添加CHECK约束的基本语法如下：

ALTER TABLE table\_name

ADD CONSTRAINT MyUniqueConstraint

CHECK (CONDITION);

##### 添加主键约束语法

使用ALTER TABLE给数据表添加主键约束的基本语法如下：

ALTER TABLE table\_name

ADD CONSTRAINT MyPrimaryKey

PRIMARY KEY (column1, column2...);

#### 删除约束

使用ALTER TABLE从数据表中删除约束的基本语法如下：

ALTER TABLE table\_name

DROP CONSTRAINT MyUniqueConstraint;

### 删除

表的删除非常容易，使用DROP TABLE关键词即可实现。语法如下。



只要在DROP TABLE关键词后面接上要删除表的名字即可。这里表的删除不仅删除了表内存储的数值，而是整个表结构都被删除了，也就是该表不存在了。

### 重命名

在创建表的时候，表的名字就被确定了，但在实际应用中，有时候需要修改表的名字而不改变其他信息，这时候就可以采用SQL的重命名表命令。

不同的DBMS对表的重命名提供的命令不尽相同。在DB2、MySQL,、Oracle数据库系统中可采用RENAME关键词，而在SQL Server和Sybase数据库系统中可采用SP\_RENAME关键词重命名表。

除了可以对表进行重命名以外，我们还可以对表中的列进行重命名。在SQL Server数据库系统中，重命名表中的列同样使用SP\_RENAME关键词。

# 使用建议

## 不建议用NULL约束

参考：

<https://mp.weixin.qq.com/s?__biz=MzA3MTg4NjY4Mw==&mid=2457318165&idx=2&sn=301f608423a1155c8ea4912a3cd84ef0&chksm=88a5a121bfd228379c7a5c917b187811d52b302dceb3460a9555798122765e8d6ef20fec19f1&mpshare=1&scene=24&srcid=0330tOYjTLSM3fEJ2MNqFB8T&sharer_sharetime=1617063086069&sharer_shareid=33f795d236f19ac7c128b2e279563f84#rd>

Mysql难以优化引用可空列查询，它会使索引、索引统计和值更加复杂。可空列需要更多的存储空间，还需要mysql内部进行特殊处理。可空列被索引后，每条记录都需要一个额外的字节，还能导致MyISAM中固定大小的索引变成可变大小的索引。

不建议使用NULL约束的理由：  
 1、所有使用NULL值的情况，都可以通过一个有意义的值的表示，这样有利于代码的可读性和可维护性，并能从约束上增强业务数据的规范性。

2、**NULL值到非NULL的更新无法做到原地更新，更容易发生索引分裂，从而影响性能。**

注意：但**把NULL列改为NOT NULL带来的性能提示很小，除非确定它带来了问题**，否则不要把它当成优先的优化措施，最重要的是使用的列的类型的适当性。

3、NULL值在timestamp类型下容易出问题，特别是没有启用参数explicit\_defaults\_for\_timestamp。

注：mysql 中有这样的一个默认行为，如果一行数据中某些列被更新了，如果这一行中有timestamp类型的列，那么么这个timestamp列的数据也会被自动更新更新操作所发生的那个时间点；这个操作是由explicit\_defaults\_for\_timestamp这个变更控制的。

4、NOT IN、!=等负向条件查询在有NULL值的情况下返回永远为空结果，查询容易出错。

5、Null列需要更多的存储空间：需要一个额外字节作为判断是否为NULL的标志位。

## 临时表性能优化

在MySQL 8.0中，用户可以把数据库和表归组到逻辑和物理表空间中，这样做可以提高资源的利用率。

MySQL 8.0使用CREATE TABLESPACE语句来创建一个通用表空间。这个功能可以让用户自由地选择表和表空间之间的映射。例如，创建表空间和设置这个表空间应该含有什么样的表。这也让在同一个表空间的用户对所有的表分组，因此在文件系统一个单独的文件内持有他们所有的数据，同时为通用表空间实现了元数据锁。

优化普通SQL临时表性能是MySQL 8.0的目标之一。首先，通过优化临时表在磁盘中的不必要步骤，使得临时表的创建和移除成为一个轻量级的操作。将临时表移动到一个单独的表空间中，恢复临时表的过程就变得非常简单，就是在启动时重新创建临时表的单一过程。

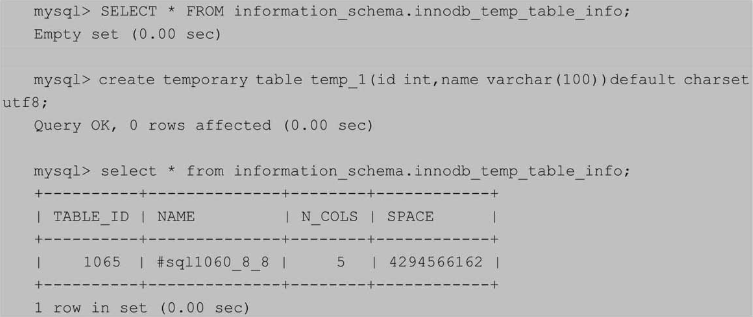
MySQL 8.0去掉了临时表中不必要的持久化。临时表仅仅在连接和会话内被创建，然后通过服务的生命周期绑定它们。通过移除不必要的UNDO和REDO日志，改变缓冲和锁，从而为临时表做了优化操作。

MySQL 8.0增加了UNDO日志一个额外的类型，这个类型的日志被保存在一个单独的临时表空间中，在恢复期间不会被调用，而是在回滚操作中才会被调用。

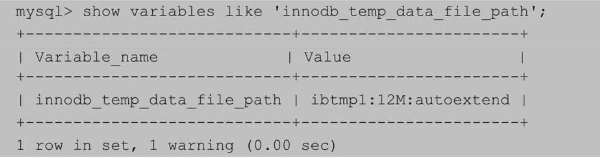
MySQL 8.0为临时表设定了一个特别类型，称之为“内在临时表”。内在临时表和普通临时表很像，只是内在临时表使用宽松的ACID和MVCC语义。

MYSQL 8.0为了提高临时表相关的性能，对临时表相关的部分进行了大幅修改，包括引入新的临时表空间（ibtmp1）；对于临时表的DDL，不持久化相关表定义；对于临时表的DML，不写redo、关闭change buffer等。

InnoDB临时表元数据不再存储于InnoDB系统表，而是存储在INNODB\_TEMP\_TABLE\_INFO中，包含所有用户和系统创建的临时表信息。该表在第一次运行select时被创建，下面举例说明。



MySQL 8.0使用了独立的临时表空间来存储临时表数据，但不能是压缩表。临时表空间在实例启动的时候进行创建、shutdown的时候进行删除，即为所有非压缩的innodb临时表提供一个独立的表空间。默认的临时表空间文件为ibtmp1，位于数据目录中。通过innodb\_temp\_data\_file\_path参数可指定临时表空间的路径和大小，默认为12MB。只有重启实例才能回收临时表空间文件ibtmp1的大小。create temporary table和using temporary table将共用这个临时表空间。

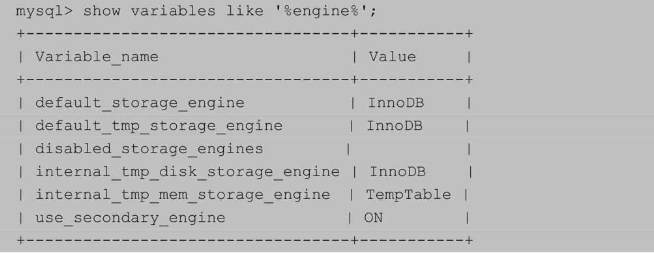


在MySQL 8.0中，临时表在连接断开或者数据库实例关闭的时候会进行删除，从而提高了性能。只有临时表的元数据使用了redo保护，保护元数据的完整性，以便异常启动后进行清理工作。

临时表的元数据在MySQL 8.0之后使用了一个独立的表（innodb\_temp\_table\_info）进行保存，不用使用redo保护，元数据也只保存在内存中。但这有一个前提，即必须使用共享的临时表空间，如果使用file-per-table，仍然需要持久化元数据，以便异常恢复清理。临时表需要undo log，用于MySQL运行时的回滚。

在MySQL 8.0中，新增一个系统选项internal\_tmp\_disk\_storage\_engine，可定义磁盘临时表的引擎类型，默认为InnoDB，可选MyISAM。在这以前，只能使用MyISAM。在MySQL 5.6.3以后新增的参数default\_tmp\_storage\_engine是控制create temporary table创建的临时表存储引擎，在以前默认是MEMORY。

查看结果如下：



# 分布式数据库临时表

## OceanBase

临时表保存仅在事务或会话期间存在的数据。

**全局临时表简介**

全局临时表用于保存一段时间内的数据，这里的一段时间可以是事务的生命周期也可以是 Session 的生命周期。

全局临时表可以对所有 Session 可见。但是，全局临时表中的数据是 Session 私有的，当前 Session 只能看到和修改自己的数据。

例如，课程安排应用程序使大学生能够创建可选的学期课程表。全局临时表中的一行代表每个课程安排。在会话期间，课程安排数据是私有的。当学生确认课程安排时，应用程序将所选时间表的行移动到普通表中。在会话结束时，数据库会自动删除全局临时表中的数据。

**全局临时表的创建方法**

用户可以通过 CREATE GLOBAL TEMPORARY TABLE 语句创建全局临时表，通过 ON COMMIT子句控制是事务临时表还是Session临时表。

**全局临时表可以创建索引，全局临时表的数据也是临时性的，数据也是Session私有的**。

示例：创建一个事务级临时表。

obclient> CREATE GLOBAL TEMPORARY TABLE tbl1(col1 INT) ON COMMIT DELETE ROWS;

Query OK, 0 rows affected

参考：

<https://www.oceanbase.com/docs/enterprise/oceanbase-database/oceanbase-database/V3.2.3/temporary-table>

## TiDB

TiDB在v5.3.0版本中引入了临时表功能。该功能针对业务中间计算结果的临时存储问题，让用户免于频繁地建表和删表等操作。用户可将业务上的中间计算数据存入临时表，用完数据后TiDB自动清理回收临时表。这避免了用户业务过于复杂，减少了表管理开销，并提升了性能。

**使用场景**

TiDB临时表主要应用于以下业务场景：

缓存业务的中间临时数据，计算完成后将数据转储至普通表，临时表会自动释放。

短期内对同一数据进行多次 DML 操作。例如在电商购物车应用中，添加、修改、删除商品及完成结算，并移除购物车信息。

快速批量导入中间临时数据，提升导入临时数据的性能。

批量更新数据。将数据批量导入到数据库的临时表，修改完成后再导出到文件。

**临时表类型**

TiDB 的临时表分为本地临时表和全局临时表：

1、本地临时表的表定义和表内数据只对当前会话可见，适用于暂存会话内的中间数据。

2、全局临时表的表定义对整个TiDB集群可见，表内数据只对当前事务可见，适用于暂存事务内的中间数据。

**本地临时表**

本地临时表的语义与 MySQL 临时表一致，它有以下特性：

1、本地临时表的表定义不持久化，只在创建该表的会话内可见，其他会话无法访问该本地临时表

2、不同会话可以创建同名的本地临时表，各会话只会读写该会话内创建的本地临时表

3、本地临时表的数据对会话内的所有事务可见

4、在会话结束后，该会话创建的本地临时表会被自动删除

5、本地临时表可以与普通表同名，此时在 DDL 和 DML 语句中，普通表被隐藏，直到本地临时表被删除

用户可通过CREATE TEMPORARY TABLE语句创建本地临时表，通过DROP TABLE或DROP TEMPORARY TABLE语句删除本地临时表。

不同于MySQL，TiDB本地临时表都是外部表，SQL语句不会创建内部临时表。

**全局临时表**

全局临时表是 TiDB 的扩展功能，它有以下特性：

1、全局临时表的表定义会持久化，对所有会话可见

2、全局临时表的数据只对当前的事务内可见，事务结束后数据自动清空

3、全局临时表不能与普通表同名

用户可通过 CREATE GLOBAL TEMPORARY TABLE 语句创建全局临时表，语句末尾要加上 ON COMMIT DELETE ROWS。可通过 DROP TABLE 或 DROP GLOBAL TEMPORARY TABLE 语句删除全局临时表。

**参考：**

https://docs.pingcap.com/zh/tidb/stable/temporary-tables#%E6%9C%AC%E5%9C%B0%E4%B8%B4%E6%97%B6%E8%A1%A8

<https://docs.pingcap.com/zh/tidb/stable/temporary-tables#%E5%85%A8%E5%B1%80%E4%B8%B4%E6%97%B6%E8%A1%A8>

## PolarDB

PolarDB O引擎支持全局临时表特性，同时保留PostgreSQL社区版local temporary table的用法。

**语法**

CREATE GLOBAL TEMPORARY|TEMP TABLE table-name

{ column-definition [ , column-definition ] \* }

[ ON COMMIT {DELETE | PRESERVE} ROWS ]

1、支持ON COMMIT DELETE ROWS子句，即当前事务提交后清空临时表中的数据。

2、支持ON COMMIT PRESERVE ROWS子句，即当前事务提交后保留临时表中的数据。

3、不支持ON COMMIT DROP子句。

4、如果不指定ON COMMIT子句，则缺省行为和ON COMMIT DELETE ROWS一致。

**描述**

1、数据库内所有会话（连接）共享全局临时表的表定义，当一个会话创建了一个全局临时表时，其他会话也能使用。

2、全局临时表的数据在会话间独立，即会话A写入到全局临时表X中的数据对会话B不可见。

3、当一个会话退出时，会清空该会话中全局临时表中的数据和底层存储。

4、支持全局临时表和其他表进行join，也支持在全局临时表上创建索引（当前版本仅支持创建btree索引）和全局临时表上的索引扫描。同时，全局临时表在表上和列上的统计信息也是在会话间独立的，这是为了让全局临时表相关的查询获得更优的查询计划。

5、全局临时表支持手动进行vacuum 和analyze，用于清理垃圾数据和收集统计信息。

**示例**

create global temp table gtt1(a int primary key, b text);

# 创建全局临时表gtt1，缺省on commit delete rows，即当前事务提交时删除表中所有数据。

create global temporary table gtt2(a int primary key, b text) on commit delete rows;

#创建全局临时表gtt2，指定on commit delete rows，即当前事务提交时删除表中所有数据。

create global temp table gtt3(a int primary key, b text) on commit PRESERVE rows;

#创建全局临时表gtt3，指定on commit preserve rows，即当前事务提交时保留表中所有数据。

**运维**

PolarDB提供了一组函数，用于对全局临时表进行日常的运维工作。

1、polar\_gtt\_attached\_pid用于查看一个全局临时表正在被哪些会话使用。您可以使用该函数结合其他函数进行运维工作。

2、polar\_gtt\_att\_statistic用于查看对应一张全局临时表列上的统计信息。

3、polar\_gtt\_relstats用于查看对应一张全局临时表的表级统计信息。

这些函数是以插件形式存在的，因此在使用之前，需要创建插件。

create extension polar\_gtt;

如果您要删除一个全局临时表，则需要在当前会话正在使用这张临时表时才能删除。

具体操作流程如下：

1、使用函数polar\_gtt\_attached\_pid查询对应的全局临时表正在被哪些会话使用。

2、使用函数pg\_backend\_pid() 确定当前会话的pid。

3、使用函数pg\_terminate\_backend(pid) 剔除掉非当前会话。

4、使用函数drop table删除这个临时表。

## TDSQL

## GoldenDB