链滴

6-MySQL 架构和性能优化（重点）

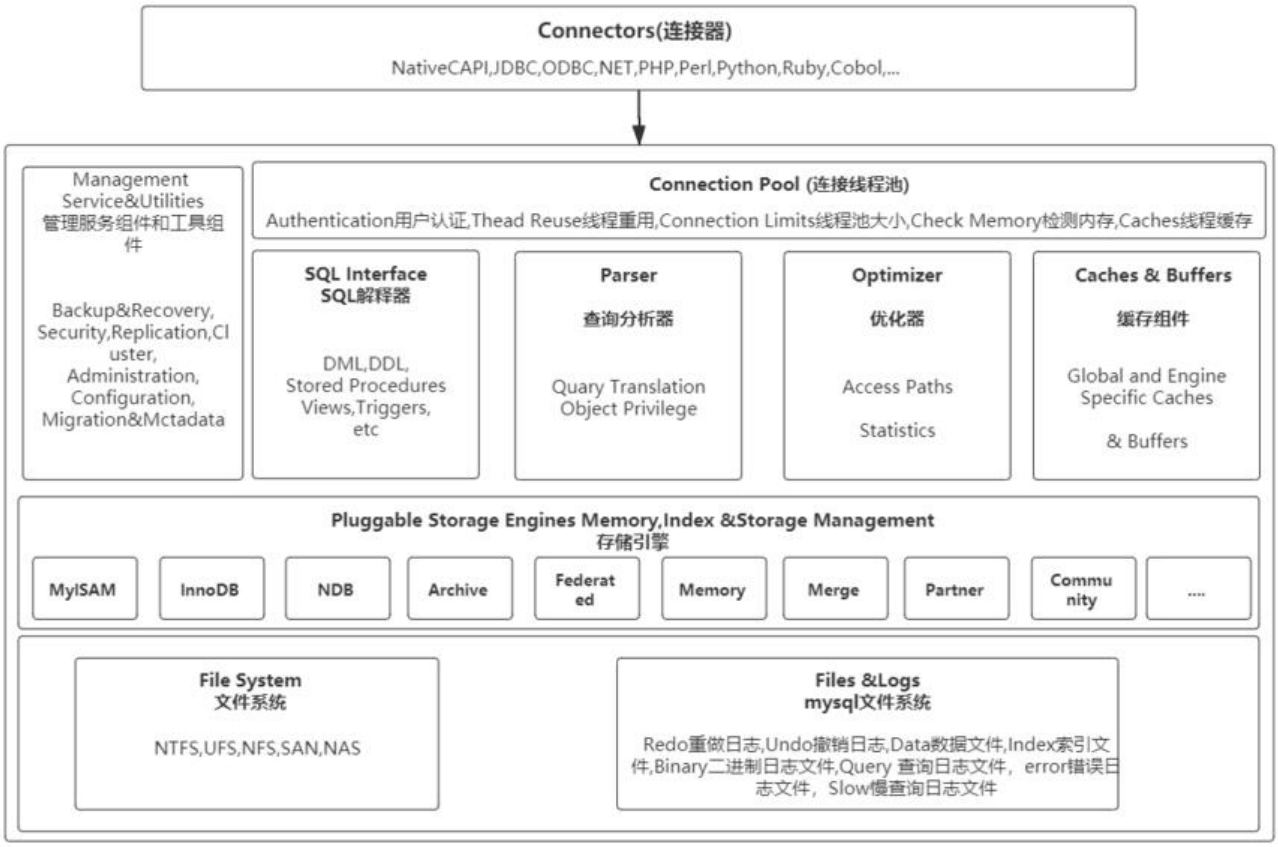
作者：Carey

原文链接：https://ld246.com/article/1614774809524

来源网站：链滴

4 MySQL 架构和性能优化

4.1 MySQL架构



原文链接：6-MySQL 架构和性能优化（重点）

MySQL是C/S 架构的

4.1.1 connectors（连接器）

可供Native C API、JDBC、ODBC、NET、PHP、Perl、Python、Ruby、Cobol 等连接mysql；ODC叫开放数据库（系统）互联，open database connection；JDBC是主要用于java语言利用较为底的驱动连接数据库；以上这些，站在编程角度可以理解为连入数据库管理系统的驱动，站在mysql角称作专用语言对应的链接器.任何链接器连入mysql以后，mysql是单进程多线程模型的，因此，每个户连接，都会创建一个单独的连接线程；其实mysql连接也有长短连接两种方式，使用mysql客户端入数据库后，直到使用quit命令才退出，可认为是长连接；使用mysql中的-e选项，在mysql客户端服务器端申请运行一个命令后则立即退出，也就意味着连接会立即断开；所以，mysql也支持长短连类似于两种类型；所以，用户连入mysql后，创建一个连接线程，完成之后，能够通过这个链接线程成接收客户端发来的请求，为其处理请求，构建响应报文并发给客户端；由于是单进程模型，就意味必须要维持一个线程池，跟之前介绍过的varnish很接近，需要一个线程池来管理这众多线程是如何众多客户端的并发请求，完成并发响应的，组件connection pool就是实现这样功能；

4.1.2 connection pool（连接线程池）

connection pool对于mysql而言，它所实现的功能，包括authentication认证，用户发来的账号密是否正确要完成认证功能；thread reuse线程重用功能，一般当一个用户连接进来以后要用一个线程响应它，而后当用户退出这个线程有可能并非被销毁，而是把它清理完以后，重新收归到线程池当中空闲线程中去，以完成所谓的线程重用；connection limit 线程池的大小决定了连接并发数量的上限例如，最多容纳100线程，一旦到达此上限后续到达的连接请求则只能排队或拒绝连接；check memoy用来检测内存，caches实现线程缓存；整个都属于线程池的功能.当用户请求之后，通过线程池建立个用户连接，这个线程一直存在，然后用户就通过这个会话，发送对应的SQL语句到服务器端.

4.1.3 sql interface（SQL解释器，SQL接口）

服务器收到SQL语句后，要对语句完成执行，首先要能理解sql语句需要有sql解释器或叫sql接口sql inerface就可理解为是整个mysql的外壳，就像shell是linux操作系统的外壳一样；用户无论通过哪种链器发来的基本的SQL请求，当然，事实上通过native C API也有发过来的不是SQL 请求，而仅仅是对AI中的传递参数后的调用；不是SQL语句不过都统统理解为sql语句罢了；对SQL而言分为DDL 和DML种类型，但是无论哪种类型，提交以后必须交给内核，让内核来运行，在这之前必须要告诉内核哪个命令，哪个是选项，哪些是参数，是否存在语法错误等等；因此，这个整个SQL 接口就是一个完完整的sql命令的解释器，并且这个sql接口还有提供完整的sql接口应该具备的功能，比如支持所谓过程式程，支持代码块的实现像存储过程、存储函数，触发器、必要时还要实现部署一个关系型数据库应该备的基本组件例如视图等等，其实都在sql interface这个接口实现的；SQL接口做完词法分析、句法析后，要分析语句如何执行让parser解析器或分析器实现

4.1.4 parser（查询分析器）

parser是专门的分析器，这个分析器并不是分析语法问题的，语法问题在sql接口时就能发现是否有错了，一个语句没有问题，就要做执行分析，所谓叫查询翻译，把一个查询语句给它转换成对应的能够本地执行的特定操作；比如说看上去是语句而背后可能是执行的一段二进制指令，这个时候就完成对的指令，还要根据用户请求的对象，比如某一字段查询内容是否有对应数据的访问权限，或叫对象访权限；在数据库中库、表、字段、字段中的数据有时都称为object，叫一个数据库的对象，用户认证通过，并不意味着就能一定能访问数据库上的所有数据，所以说，mysql的认证大概分为两过程都要成，第一是连入时需要认证账号密码是否正确这是authentication，然后，验证成功后用户发来sql语还要验证用户是否有权限获取它期望请求获取的数据；这个称为object privilege，这一切都是由parsr分析器进行的

4.1.5 Optimizer（优化器）

分析器分析完成之后，可能会生成多个执行树，这意味着为了能够达到访问期望访问到的目的，可能多条路径都可实现，就像文件系统一样可以使用相对路径也可使用绝对路径；它有多种方式，在多种径当中一定有一个是最优的，类似路由选择，因此，优化器就要去衡量多个访问路径中哪一个代价或销是最小的，这个开销的计算要依赖于索引等各种内部组件来进行评估；而且这个评估的只是近似值同时还要考虑到当前mysql内部在实现资源访问时统计数据，比如，根据判断认为是1号路径的开销小的，但是众多统计数据表明发往1号路径的访问的资源开销并不小，并且比3号路径大的多，因此，能会依据3号路径访问；这就是所谓的优化器它负责检查多条路径，每条路径的开销，然后评估开销这个评估根据内部的静态数据，索引，根域根据动态生成的统计数据来判定每条路径的开销大小，因这里还有statics；一旦优化完成之后，还要生成统计数据，这就是优化器的作用；如果没有优化器mysql执行语句是最慢的，其实优化还包括一种功能，一旦选择完一条路径后，例如用户给的这个命令执起来，大概需要100个开销，如果通过改写语句能够达到同样目的可能只需要30个开销；于是，优化还要试图改写sql语句；所以优化本身还包括查询语句的改写；一旦优化完成，接下来就交给存储引完成.

4.1.6 cache和buffer（缓存组件）

事实上，整个存取过程，尤其是访问比较热点的数据，也不可能每一次当用户访问时或当某SQL语句到时再临时从磁盘加载到内存中，因此，为了能够加上整个性能，mysql的有些存储引擎可以实现，频繁访问到的热点数据，统统装入内存，用户访问、修改时直接在内存中操作，只不过周期性的写入盘上而已，比如像InnoDB，所以caches和buffers组件就是实现此功能的；MySQL为了执行加速，为它会不断访问数据，而随计算机来说io是最慢的一环，尤其是磁盘io，所以为了加速都载入内存中理；这就需要MySQL 维护cache和buffer缓存或缓冲；这是由MySQL 服务器自己维护的；有很多存引擎自己也有cache和buffer

4.1.7 Pluggable Storage Engines（插件式存储引擎）

mysql是插件式存储引擎，它就能够替换使用选择多种不同的引擎，MyISAM是MySQL 经典的存储擎之一，InnoDB是由Innobase Oy公司所开发，2006年五月由甲骨文公司并购提供给MySQL的，NB主要用于MySQL Cluster 分布式集群环境，archive做归档的等等，还有许多第三方开发的存储引；存储引擎负责把具体分析的结果完成对磁盘上文件路径访问的转换，数据库中的行数据都是存储在盘块上的，因此存储引擎要把数据库数据映射为磁盘块，并把磁盘块加载至内存中；进程实现数据处时，是不可能直接访问磁盘上的数据的，因为它没有权限，只有让内核来把它所访问的数据加载至内中以后，进程在内存中完成修改，由内核再负责把数据存回磁盘；对于文件系统而言，数据的存储都以磁盘块方式存储的，但是，mysql在实现数据组织时，不完全依赖于磁盘，而是把磁盘块再次组织更大一级的逻辑单位，类似于lvm中的PE或LE的形式；其实，MySQL的存储引擎在实现数据管理时，是在文件系统之上布设文件格式，对于文件而言在逻辑层上还会再次组织成一个逻辑单位，这个逻辑位称为mysql的数据块datablock 一般为16k ，对于关系型数据库，数据是按行存储的；一般一行数都是存储在一起的，因此，MySQL 在内部有一个datablock，在datablock可能存储一行数据，也可存放了n行数据；将来在查询加载一行数据时，内核会把整个一个数据数据块加载至内存中，而mysq存储引擎，就从中挑出来某一行返回给查询者，是这样实现的；所以整个存储是以datablock在底层其最终级别的.

4.1.8 File System（物理文件）

一个数据库提供了3种视图，物理视图就是看到的对应的文件系统存储为一个个的文件，MySQL的数文件类型，常见的有redo log重做日志，undo log撤销日志，data是真正的数据文件，index是索引件，binary log是二进制日志文件，error log错误日志，query log查询日志，slow query log慢查日志，在复制架构中还存在中继日志文件，跟二进制属于同种格式；这是mysql数据文件类型，也就物理视图；逻辑视图这是在mysql接口上通过存储引擎把mysql文件尤其是data文件，给它映射为一个关系型数据库应该具备组成部分，比如表，一张表在底层是一个数据文件而已，里面组织的就是dat

block，最终映射为磁盘上文件系统的block，然后再次映射为本地扇区的存储，但是整个mysql需要他们映射成一个二维关系表的形式，需要依赖sql接口以及存储引擎共同实现；所以，把底层数据文映射成关系型数据库的组件就是逻辑视图；DBA 就是关注内部组件是如何运作的，并且定义、配置运作模式，而链接器都是终端用户通过链接器的模式进入数据库来访问数据；数据集可能非大，每一用户可能只有一部分数据的访问权限，这个时候，最终的终端用户所能访问到的数据集合称作用户视；

4.1.9 Management Service & Utilities（管理服务组件和工具组）

为了保证MySQL运作还提供了管理和服务工具，例如:备份恢复工具，安全工具，复制工具，集群服，管理、配置、迁移、元数据等工具

4.2 存储引擎

Feature	MyISAM	Memory	InnoDB	Archive	NDB
Storage limits	256TB	RAM	64TB	None	384EB
Transactions	No	No	Yes	No	Yes
Locking granularity	Table	Table	Row	Row	Row
MVCC	No	No	Yes	No	No
Geospatial data type support	Yes	No	Yes	Yes	Yes
Geospatial indexing support	Yes	No	Yes [a]	No	No
B-tree indexes	Yes	Yes	Yes	No	No
T-tree indexes	No	No	No	No	Yes
Hash indexes	No	Yes	No [b]	No	Yes
Full-text search indexes	Yes	No	Yes [c]	No	No
Clustered indexes	No	No	Yes	No	No
Data caches	No	N/A	Yes	No	Yes
Index caches	Yes	N/A	Yes	No	Yes
Compressed data	Yes [d]	No	Yes [e]	Yes	No
Encrypted data [f]	Yes	Yes	Yes	Yes	Yes
Cluster database support	No	No	No	No	Yes
Replication support [g]	Yes	Limited [h]	Yes	Yes	Yes
Foreign key support	No	No	Yes	No	Yes [i]
Backup / point-in-time recovery [j]	Yes	Yes	Yes	Yes	Yes
Query cache support	Yes	Yes	Yes	Yes	Yes
Update statistics for data dictionary	Yes	Yes	Yes	Yes	Yes

MySQL中的数据用各种不同的技术存储在文件（或者内存）中。这些技术中的每一种技术都使用不的存储机制、索引技巧、锁定水平并且最终提供广泛的不同的功能和能力,此种技术称为存擎,MySQL持多种存储引擎其中目前应用最广泛的是InnoDB和MyISAM两种

官方参考资料：

https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/storage-engines.html
https://docs.oracle.com/cd/E17952_01/mysql-5.7-en/storage-engines.html

4.2.1 MyISAM 存储引擎

MyISAM 引擎特点

- 不支持事务
- 表级锁定

- 读写相互阻塞，写入不能读，读时不能写
- 只缓存索引
- 不支持外键约束
- 不支持聚簇索引
- 读取数据较快，占用资源较少
- 不支持MVCC（多版本并发控制机制）高并发
- 崩溃恢复性较差
- MySQL5.5.5 前默认的数据库引擎

MyISAM 存储引擎适用场景

- 只读（或者写较少）
- 表较小（可以接受长时间进行修复操作）

MyISAM 引擎文件

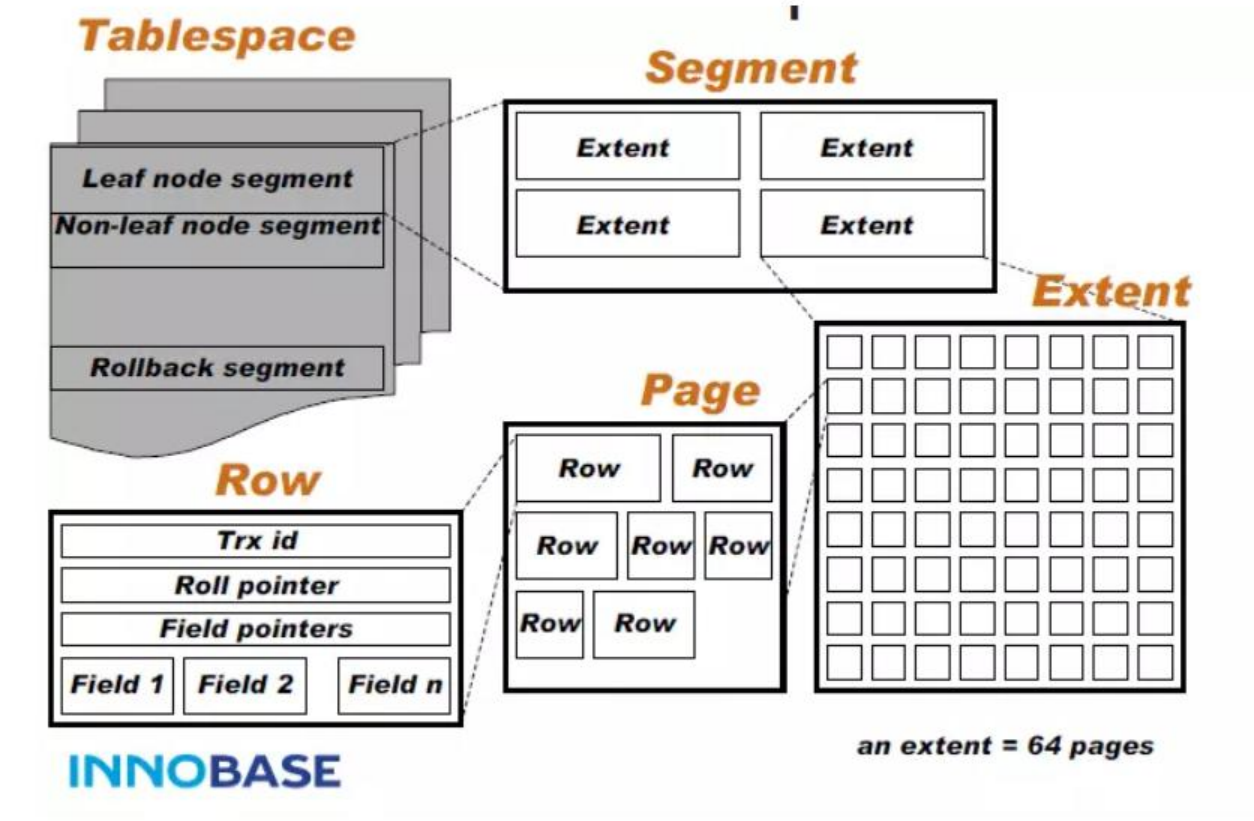
- tbl_name.frm 表格式定义
- tbl_name.MYD 数据文件
- tbl_name.MYI 索引文件

4.2.2 InnoDB 引擎

InnoDB引擎特点

- 行级锁
- 支持事务，适合处理大量短期事务
- 读写阻塞与事务隔离级别相关
- 可缓存数据和索引
- 支持聚簇索引
- 崩溃恢复性更好
- 支持MVCC高并发
- 从MySQL5.5后支持全文索引
- 从MySQL5.5.5开始为默认的数据库引擎

InnoDB数据库文件



InnoDB表空间管理

- 所有InnoDB表的数据和索引放置于同一个表空间中

数据文件：ibdata1, ibdata2,存放在datadir定义的目录下
表格式定义：tb_name.frm,存放在datadir定义的每个数据库对应的目录下

- 每个表单独使用一个表空间存储表的数据和索引

两类文件放在对应每个数据库独立目录中
数据文件(存储数据和索引)：tb_name.ibd
表格式定义：tb_name.frm

启用：innodb_file_per_table=ON (MariaDB 5.5以后版是默认值)

4.2.3 其它存储引擎

- Performance_Schema：Performance_Schema数据库使用
- Memory ： 将所有数据存储在RAM中，以便在需要快速查找参考和其他类似数据的环境中进行快访问。适用存放临时数据。引擎以前被称为HEAP引擎
- MRG_MyISAM：使MySQL DBA或开发人员能够对一系列相同的MyISAM表进行逻辑分组，并将们作为一个对象引用。适用于VLDB(Very Large Data Base)环境，如数据仓库
- Archive ： 为存储和检索大量很少参考的存档或安全审核信息，只支持SELECT和INSERT操作；支行级锁和专用缓存区
- Federated联合：用于访问其它远程MySQL服务器一个代理，它通过创建一个到远程MySQL服务的客户端连接，并将查询传输到远程服务器执行，而后完成数据存取，提供链接单独MySQL服务器能力，以便从多个物理服务器创建一个逻辑数据库。非常适合分布式或数据集市环境

● BDB：可替代InnoDB的事务引擎，支持COMMIT、ROLLBACK和其他事务特性Cluster/NDB：MyQL的簇式数据库引擎，尤其适合于具有高性能查找要求的应用程序，这类查找需求还要求具有最高的常工作时间和可用性

● CSV：CSV存储引擎使用逗号分隔值格式将数据存储在文本文件中。可以使用CSV引擎以CSV格式入和导出其他软件和应用程序之间的数据交换

● BLACKHOLE ： 黑洞存储引擎接受但不存储数据，检索总是返回一个空集。该功能可用于分布式数库设计，数据自动复制，但不是本地存储

● example："stub"引擎，它什么都不做。可以使用此引擎创建表，但不能将数据存储在其中或从中索。目的是作为例子来说明如何开始编写新的存储引擎

4.2.4 管理存储引擎

查看mysql支持的存储引擎

```
show engines;
```

查看当前默认的存储引擎

```
show variables like '%storage_engine%';
```

设置默认的存储引擎

```
vim /etc/my.cnf
[mysqld]
default_storage_engine= InnoDB
```

查看库中所有表使用的存储引擎

```
show table status from testdb;
show table status from testdb\G;
```

查看库中指定表的存储引擎

```
show table status like 'events_list';
show table status like 'events_list'\G
show create table events_list;
```

设置表的存储引擎：

```
CREATE TABLE tb_name(... ) ENGINE=InnoDB;
#修改表存储引擎，切记如果表中有数据请勿修改
ALTER TABLE tb_name ENGINE=InnoDB;
```

4.3 MySQL 中的系统数据库

● **mysql 数据库**

是mysql的核心数据库，类似于Sql Server中的master库，主要负责存储数据库的用户、权限设置、键字等mysql自己需要使用的控制和管理信息

● **information_schema 数据库**

MySQL 5.0之后产生的，一个虚拟数据库，物理上并不存在information_schema数据库类似与"数据典"，提供了访问数据库元数据的方式，即数据的数据。比如数据库名或表名，列类型，访问权限（加细化的访问方式）

- **performance_schema 数据库**

MySQL 5.5开始新增的数据库，主要用于收集数据库服务器性能参数,库里表的存储引擎均为PERFORANCE_SCHEMA，用户不能创建存储引擎为PERFORMANCE_SCHEMA的表

- **sys 数据库**

MySQL5.7之后新增加的数据库，库中所有数据源来自performance_schema。目标是把performanc_schema的把复杂度降低，让DBA能更好的阅读这个库里的内容。让DBA更快的了解DataBase的运情况

4.4 服务器配置和状态

可以通过mysqld选项，服务器系统变量和服务器状态变量进行MySQL的配置和查看状态

官方帮助：可以查询各种系统变量参数的详细信息

https://dev.mysql.com/doc/refman/8.0/en/server-option-variable-reference.html
https://dev.mysql.com/doc/refman/5.7/en/server-option-variable-reference.html
https://mariadb.com/kb/en/library/full-list-of-mariadb-options-system-and-status-variables/

注意：

- 其中有些参数支持运行时修改，会立即生效
- 有些参数不支持动态修改，且只能通过修改配置文件，并重启服务器程序生效
- 有些参数作用域是全局的，为所有会话设置
- 有些可以为每个用户提供单独（会话）的设置

4.4.1 服务器选项

注意: 服务器选项用横线,不用下划线

获取mysqld的可用选项列表：

```
#查看mysqld可用选项列表和及当前值
mysqld --verbose --help

#获取mysqld当前启动选项
mysqld --print-defaults
```

设置服务器选项方法：

1. 在命令行中设置

```
shell> /usr/bin/mysqld_safe --skip-name-resolve=1
shell> /usr/libexec/mysqld --basedir=/usr
```

2. 在配置文件my.cnf中设置

```
vim /etc/my.cnf
[mysqld]
skip_name_resolve=1
skip-grant-tables
```

范例: skip-grant-tables是服务器选项,但不是系统变量

```
mysql> show variables like 'skip_grant_tables';
Empty set (0.00 sec)
```

4.4.2 服务器系统变量

服务器系统变量：可以分全局和会话两种

注意: 系统变量用下划线,不用横线

获取系统变量

```
SHOW GLOBAL VARIABLES; #只查看global变量
SHOW [SESSION] VARIABLES; #查看所有变量(包括global和session)

#查看指定的系统变量
SHOW VARIABLES LIKE 'VAR_NAME';
SELECT @@VAR_NAME;

#查看选项和部分变量
[root@centos8 ~]#mysqladmin variables
```

修改服务器变量的值：

```
#修改全局变量：仅对修改后新创建的会话有效；对已经建立的会话无效
SET GLOBAL system_var_name=value;
SET @@global.system_var_name=value;
#修改会话变量：
SET [SESSION] system_var_name=value;
SET @@[session.]system_var_name=value;
```

范例：修改mysql的最大并发连接数

```
#默认值151
mysql> show variables like 'max_connections';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_connections | 151   |
+-----------------+-------+
1 row in set (0.00 sec)
mysql> set global max_connections=2000;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'max_connections';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
```

```
| max_connections | 2000  |
+-----------------+-------+
1 row in set (0.00 sec)

[16:29:49 root@centos8 ~]#vim /etc/my.cnf.d/mysql-server.cnf
[mysqld]
max_connections = 8000

[16:46:02 root@centos8 ~]#systemctl restart mysqld.service

mysql> show variables like 'max_connections';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_connections | 8000  |
+-----------------+-------+
1 row in set (0.00 sec)
```

4.4.3 服务器状态变量

服务器状态变量：分全局和会话两种

状态变量（只读）：用于保存mysqld运行中的统计数据的变量，不可更改

```
SHOW GLOBAL STATUS;
SHOW [SESSION] STATUS;
```

范例：

```
mysql> show status like "innodb_page_size";
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| Innodb_page_size | 16384 |
+------------------+-------+
1 row in set (0.00 sec)

mysql> mysql> show status like "com_select";
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Com_select    | 1     |
+---------------+-------+
1 row in set (0.00 sec)
```

4.4.4 服务器变量 SQL_MODE

SQL_MODE：对其设置可以完成一些约束检查的工作,可分别进行全局的设置或当前会话的设置

参考：

https://mariadb.com/kb/en/library/sql-mode/

https://dev.mysql.com/doc/refman/5.7/en/server-options.html#option_mysqld_sql-mode

常见MODE:

- NO_AUTO_CREATE_USER： 禁止GRANT创建密码为空的用户
- NO_ZERO_DATE：在严格模式，不允许使用'0000-00-00'的时间
- ONLY_FULL_GROUP_BY： 对于GROUP BY聚合操作，如果在SELECT中的列，没有在GROUP BY出现，那么将认为这个SQL是不合法的
- NO_BACKSLASH_ESCAPES： 反斜杠""作为普通字符
非转义字符
- PIPES_AS_CONCAT： 将"||"视为连接操作符而非"或"运算符

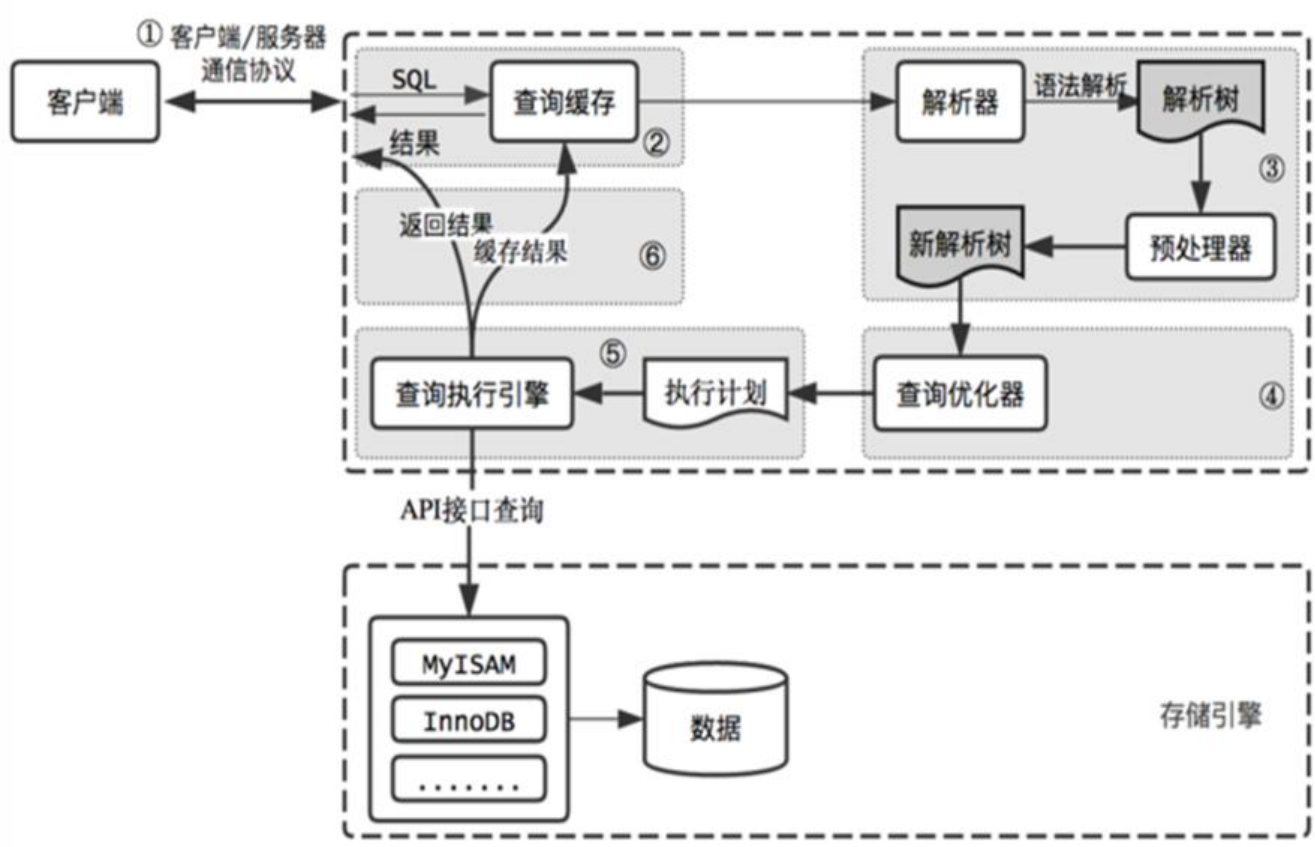
范例:

```
set sql_mode='ONLY_FULL_GROUP_BY';
```

4.5 Query Cache 查询缓存

4.5.1 查询缓存原理

查询执行路径



查询缓存原理

缓存SELECT操作或预处理查询的结果集和SQL语句，当有新的SELECT语句或预处理查询语句请求，去查询缓存，判断是否存在可用的记录集，判断标准：与缓存的SQL语句，是否完全一样，区分大小写

优缺点

不需要对SQL语句做任何解析和执行，当然语法解析必须通过在先，直接从Query Cache中获得查询果，提高查询性能查询缓存的判断规则，不够智能，也即提高了查询缓存的使用门槛，降低效率查询存的使用，会增加检查和清理Query Cache中记录集的开销

哪些查询可能不会被缓存

- 查询语句中加了SQL_NO_CACHE参数
- 查询语句中含有获得值的函数，包含:自定义函数，如：NOW() ，CURDATE()、GET_LOCK()、RAD()、CONVERT_TZ()等
- 对系统数据库的查询：mysql、information_schema 查询语句中使用SESSION级别变量或存储过中的局部变量
- 查询语句中使用了LOCK IN SHARE MODE、FOR UPDATE的语句，查询语句中类似SELECT ...INT 导出数据的语句
- 对临时表的查询操作
- 存在警告信息的查询语句
- 不涉及任何表或视图的查询语句
- 某用户只有列级别权限的查询语句
- 事务隔离级别为Serializable时，所有查询语句都不能缓存

4.5.2 查询缓存相关的服务器变量

- query_cache_min_res_unit：查询缓存中内存块的最小分配单位，默认4k，较小值会减少浪费，会导致更频繁的内存分配操作，较大值会带来浪费，会导致碎片过多，内存不足
- query_cache_limit：单个查询结果能缓存的最大值，单位字节，默认为1M，对于查询结果过大而法缓存的语句，建议使用SQL_NO_CACHE
- query_cache_size：查询缓存总共可用的内存空间；单位字节，必须是1024的整数倍，最小值40K，低于此值有警报
- query_cache_wlock_invalidate：如果某表被其它的会话锁定，是否仍然可以从查询缓存中返回结，默认值为OFF，表示可以在表被其它会话锁定的场景中继续从缓存返回数据；ON则表示不允许
- query_cache_type：是否开启缓存功能，取值为ON, OFF, DEMAND

4.5.3 SELECT 语句的缓存控制

SQL_CACHE：显式指定存储查询结果于缓存之中

SQL_NO_CACHE：显式查询结果不予缓存

query_cache_type 参数变量

- query_cache_type的值为OFF或0时，查询缓存功能关闭
- query_cache_type的值为ON或1时，查询缓存功能打开，SELECT的结果符合缓存条件即会缓存，则，不予缓存，显式指定SQL_NO_CACHE，不予缓存，此为默认值
- query_cache_type的值为DEMAND或2时，查询缓存功能按需进行，显式指定SQL_CACHE的SELET语句才会缓存；其它均不予缓存

官方帮助：

https://mariadb.com/kb/en/library/server-system-variables/#query_cache_type

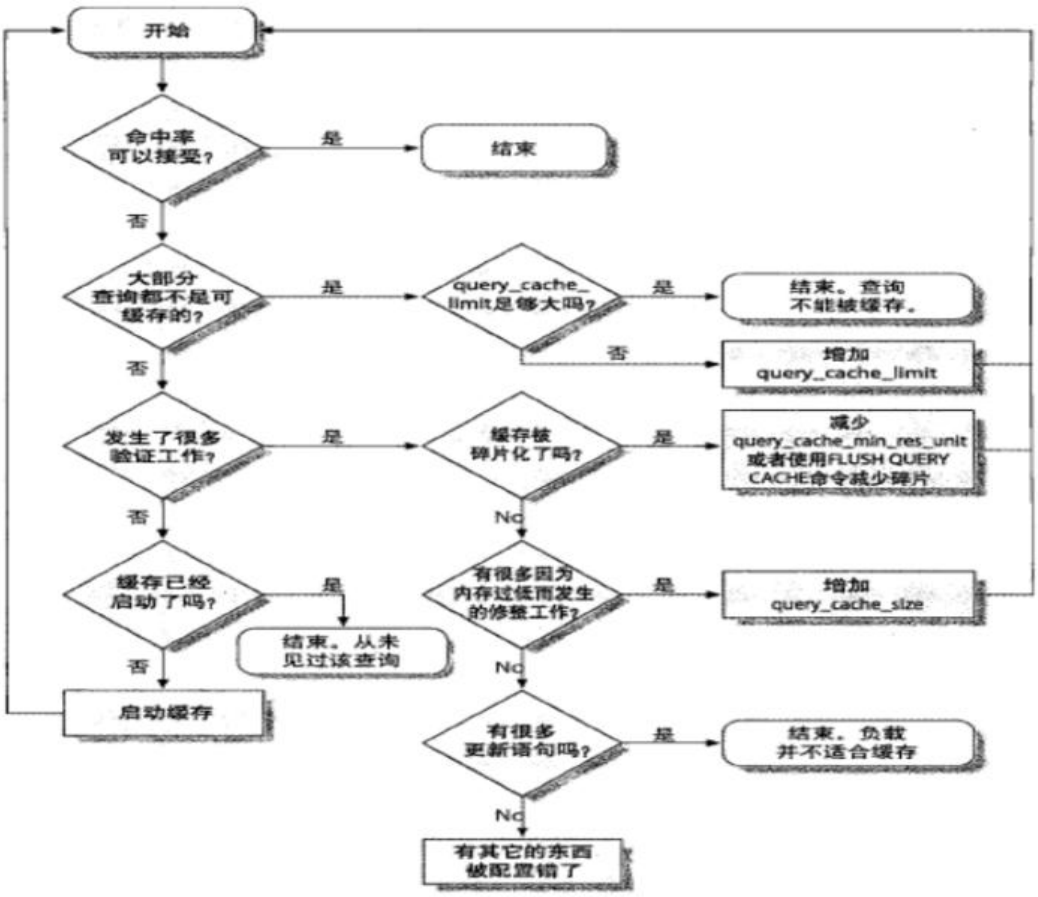
https://dev.mysql.com/doc/refman/5.7/en/query-cache-configuration.html

4.5.4 查询缓存相关的状态变量

SHOW GLOBAL STATUS LIKE 'Qcache%';

- Qcache_free_blocks：处于空闲状态 Query Cache中内存 Block 数
- Qcache_total_blocks：Query Cache 中总Block，当Qcache_free_blocks相对此值较大时，可能内存碎片，执行FLUSH QUERY CACHE清理碎片
- Qcache_free_memory：处于空闲状态的 Query Cache 内存总量
- Qcache_hits：Query Cache 命中次数
- Qcache_inserts：向 Query Cache 中插入新的 Query Cache 的次数，即没有命中的次数
- Qcache_lowmem_prunes：记录因为内存不足而被移除出查询缓存的查询数
- Qcache_not_cached：没有被 Cache 的 SQL 数，包括无法被 Cache 的 SQL 以及由于query_cach_type 设置的不会被 Cache 的 SQL语句
- Qcache_queries_in_cache：在 Query Cache 中的 SQL 数量

4.5.5 查询的优化



4.5.6 命中率和内存使用率估算

- 查询缓存中内存块的最小分配单位query_cache_min_res_unit：

(query_cache_size - Qcache_free_memory) / Qcache_queries_in_cache

- 查询缓存命中率：

Qcache_hits / (Qcache_hits + Qcache_inserts) * 100%

- 查询缓存内存使用率：

(query_cache_size - qcache_free_memory) / query_cache_size * 100%

范例：

```
[root@centos7 ~]#vim /etc/my.cnf
[mysqld]
query_cache_type=ON
query_cache_size=10M
[root@centos8 ~]#systemctl restart mariadb
[root@centos8 ~]#mysql -uroot -p
MariaDB [(none)]> show variables like 'query_cache%';
+------------------------------+---------+
| Variable_name                | Value   |
+------------------------------+---------+
| query_cache_limit            | 1048576 |
| query_cache_min_res_unit     | 4096    |
| query_cache_size             | 1048576 |
| query_cache_strip_comments   | OFF     |
| query_cache_type             | ON      |
| query_cache_wlock_invalidate | OFF     |
+------------------------------+---------+
6 rows in set (0.001 sec)
MariaDB [(none)]> SHOW GLOBAL STATUS LIKE 'Qcache%';
+-------------------------+---------+
| Variable_name           | Value   |
+-------------------------+---------+
| Qcache_free_blocks      | 1       |
| Qcache_free_memory      | 1027736 |
| Qcache_hits             | 3       |
| Qcache_inserts          | 3       |
| Qcache_lowmem_prunes    | 0       |
| Qcache_not_cached       | 0       |
| Qcache_queries_in_cache | 3       |
| Qcache_total_blocks     | 8       |
+-------------------------+---------+
8 rows in set (0.003 sec)
```

4.5.7 MySQL 8.0 变化

MySQL8.0 取消查询缓存的功能

尽管MySQL Query Cache旨在提高性能，但它存在严重的可伸缩性问题，并且很容易成为严重的瓶
。

自MySQL 5.6（2013）以来，默认情况下已禁用查询缓存，其不能与多核计算机上在高吞吐量工作载情况下进行扩展。

另外有时因为查询缓存往往弊大于利。比如:查询缓存的失效非常频繁，只要有对一个表的更新，这个上的所有的查询缓存都会被清空。因此很可能你费劲地把结果存起来，还没使用呢，就被一个更新全空了。对于更新压力大的数据库来说，查询缓存的命中率会非常低。除非你的业务有一张静态表，很时间更新一次，比如系统配置表，那么这张表的查询才适合做查询缓存。

目前大多数应用都把缓存做到了应用逻辑层，比如:使用redis或者memcached

4.6 INDEX 索引

4.6.1 索引介绍

索引：是排序的快速查找的特殊数据结构，定义作为查找条件的字段上，又称为键key，索引通过存引擎实现

优点:

- 索引可以降低服务需要扫描的数据量，减少了IO次数
- 索引可以帮助服务器避免排序和使用临时表
- 索引可以帮助将随机I/O转为顺序I/O

缺点:

- 占用额外空间，影响插入速度

索引类型:

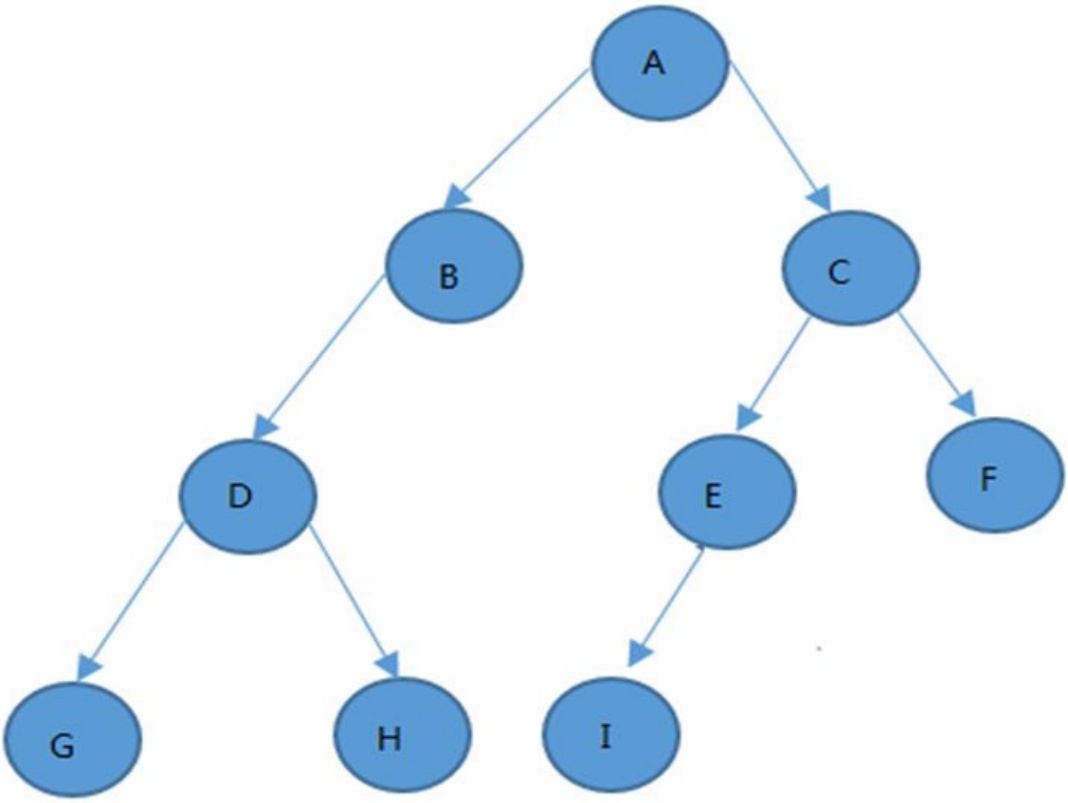
- B+ TREE、HASH、R TREE、FULL TEXT
- 聚簇（集）索引、非聚簇索引：数据和索引是否存储在一起
- 主键索引、二级（辅助）索引
- 稠密索引、稀疏索引：是否索引了每一个数据项
- 简单索引、组合索引: 是否是多个字段的索引
- 左前缀索引：取前面的字符做索引
- 覆盖索引：从索引中即可取出要查询的数据，性能高

4.6.2 索引结构

参考链接 : https://www.cs.usfca.edu/~galles/visualization/Algorithms.html

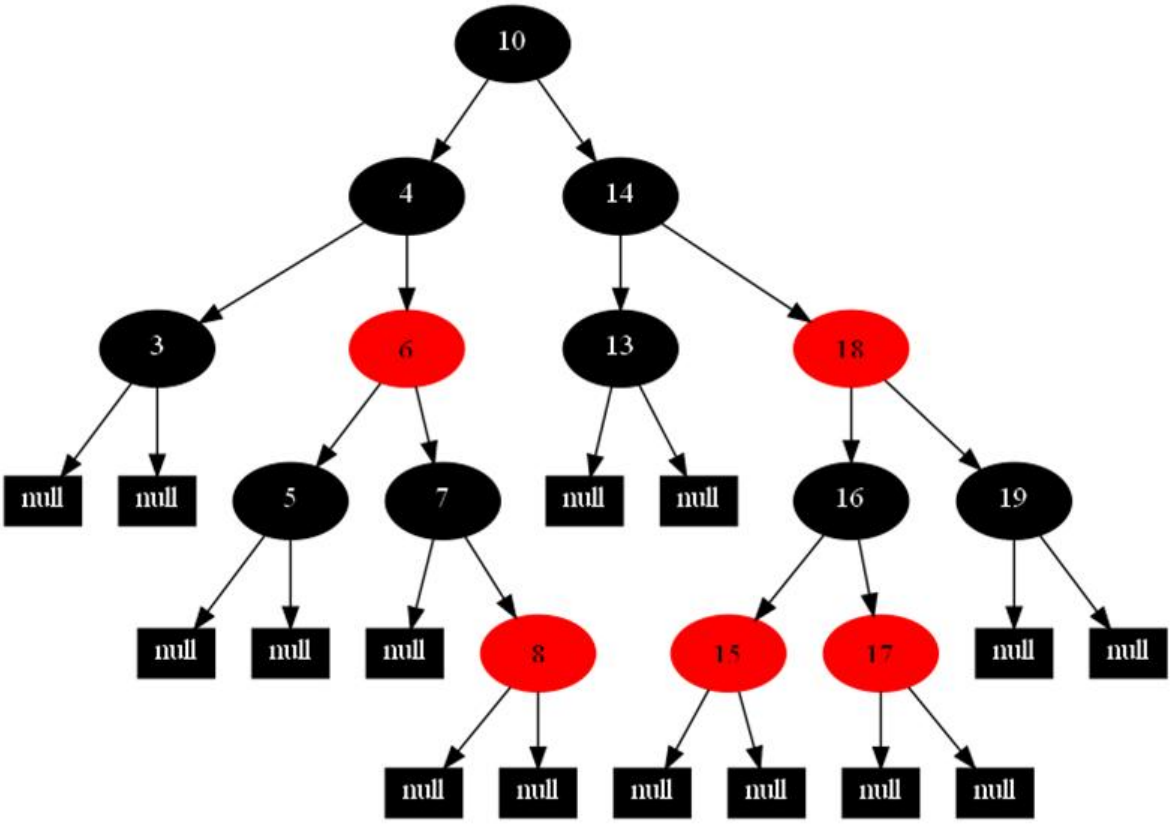
二叉树

参考链接: https://www.cs.usfca.edu/~galles/visualization/BST.html



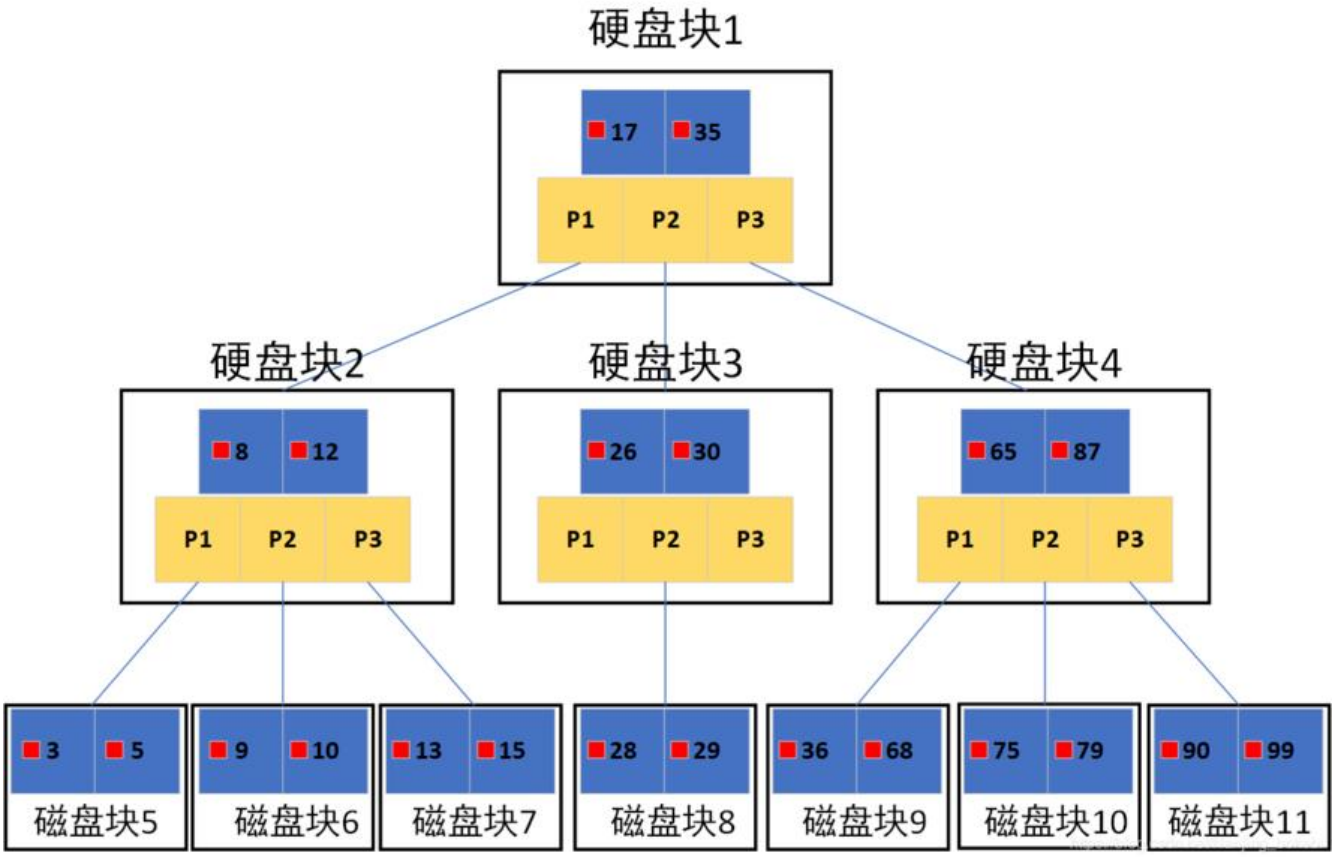
红黑树

参考链接:https://www.cs.usfca.edu/~galles/visualization/RedBlack.html



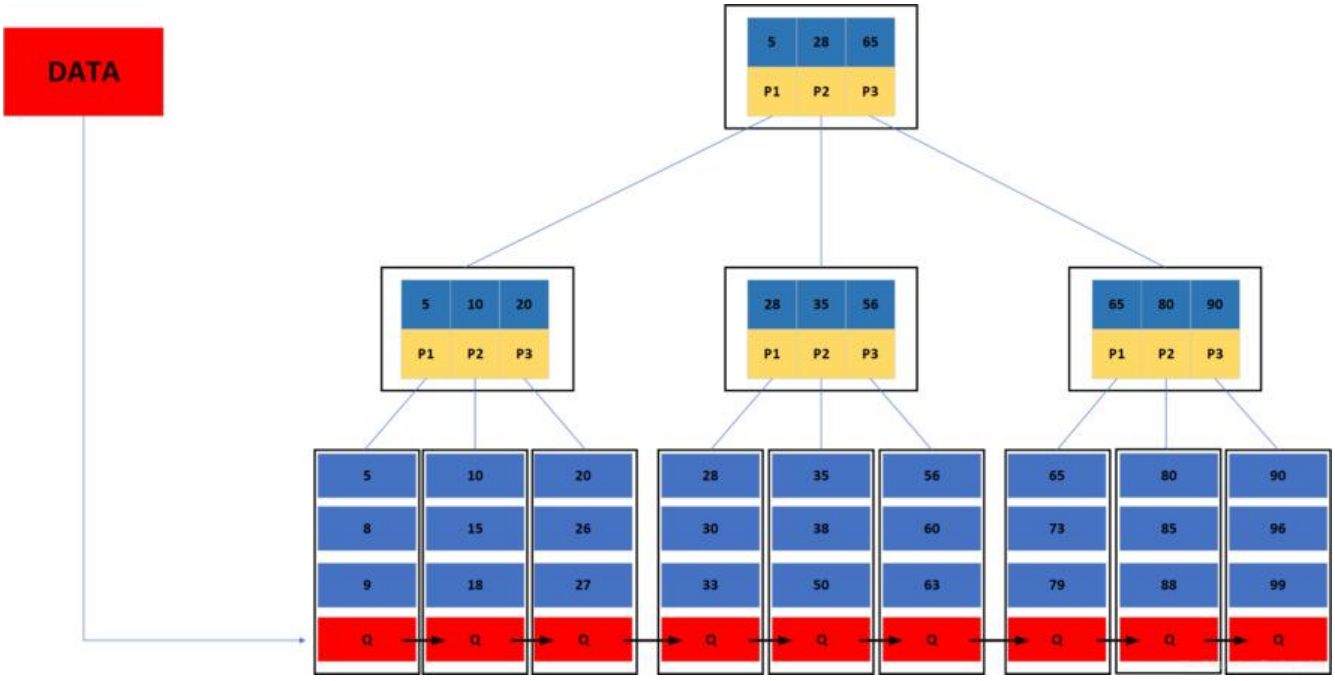
B-Tree 索引

参考链接: https://www.cs.usfca.edu/~galles/visualization/BTree.html



B+Tree索引

参考链接: https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html



B+Tree索引：按顺序存储，每一个叶子节点到根结点的距离是相同的；左前缀索引，适合查询范围的数据

面试题: InnoDB中一颗的B+树可以存放多少行数据?

假设定义一颗B+树高度为2，即一个根节点和若干叶子节点。那么这棵B+树的存放总行记录数=根节指针数*
单个叶子记录的行数。这里先计算叶子节点，B+树中的单个叶子节点的大小为16K，假设每一条目为K，那么
记录数即为16(16k/1K=16)，然后计算非叶子节点能够存放多少个指针，假设主键ID为bigint类型，么
长度为8字节，而指针大小在InnoDB中是设置为6个字节，这样加起来一共是14个字节。那么通过页小/(主
键ID大小+指针大小），即16384/14=1170个指针，所以一颗高度为2的B+树能存放16*1170=1872条这样
的记录。根据这个原理就可以算出一颗高度为3的B+树可以存放16*1170*1170=21902400条记录。以在
InnoDB中B+树高度一般为2-3层，它就能满足千万级的数据存储

可以使用B+Tree索引的查询类型：(假设前提: 姓,名,年龄三个字段建立了一个复合索引)

- 全值匹配：精确所有索引列，如：姓wang，名xiaochun，年龄30
- 匹配最左前缀：即只使用索引的第一列，如：姓wang
- 匹配列前缀：只匹配一列值开头部分，如：姓以w开头的记录
- 匹配范围值：如：姓ma和姓wang之间
- 精确匹配某一列并范围匹配另一列：如：姓wang,名以x开头的记录
- 只访问索引的查询

B+Tree索引的限制：

- 如不从最左列开始，则无法使用索引，如：查找名为xiaochun，或姓为g结尾
- 不能跳过索引中的列：如：查找姓wang，年龄30的，只能使用索引第一列

特别提示：

索引列的顺序和查询语句的写法应相匹配，才能更好的利用索引

为优化性能，可能需要针对相同的列但顺序不同创建不同的索引来满足不同类型的查询需求

Hash索引

Hash索引：基于哈希表实现，只有精确匹配索引中的所有列的查询才有效，索引自身只存储索引列应的哈希值和数据指针，索引结构紧凑，查询性能好

Memory存储引擎支持显式hash索引，InnoDB和MyISAM存储引擎不支持

适用场景：只支持等值比较查询，包括=, <=>, IN()

不适合使用hash索引的场景

- 不适用于顺序查询：索引存储顺序的不是值的顺序
- 不支持模糊匹配
- 不支持范围查询
- 不支持部分索引列匹配查找：如A，B列索引，只查询A列索引无效

地理空间数据索引R-Tree（Geospatial indexing）

MyISAM支持地理空间索引，可使用任意维度组合查询，使用特有的函数访问，常用于做地理数据存储，使用不多

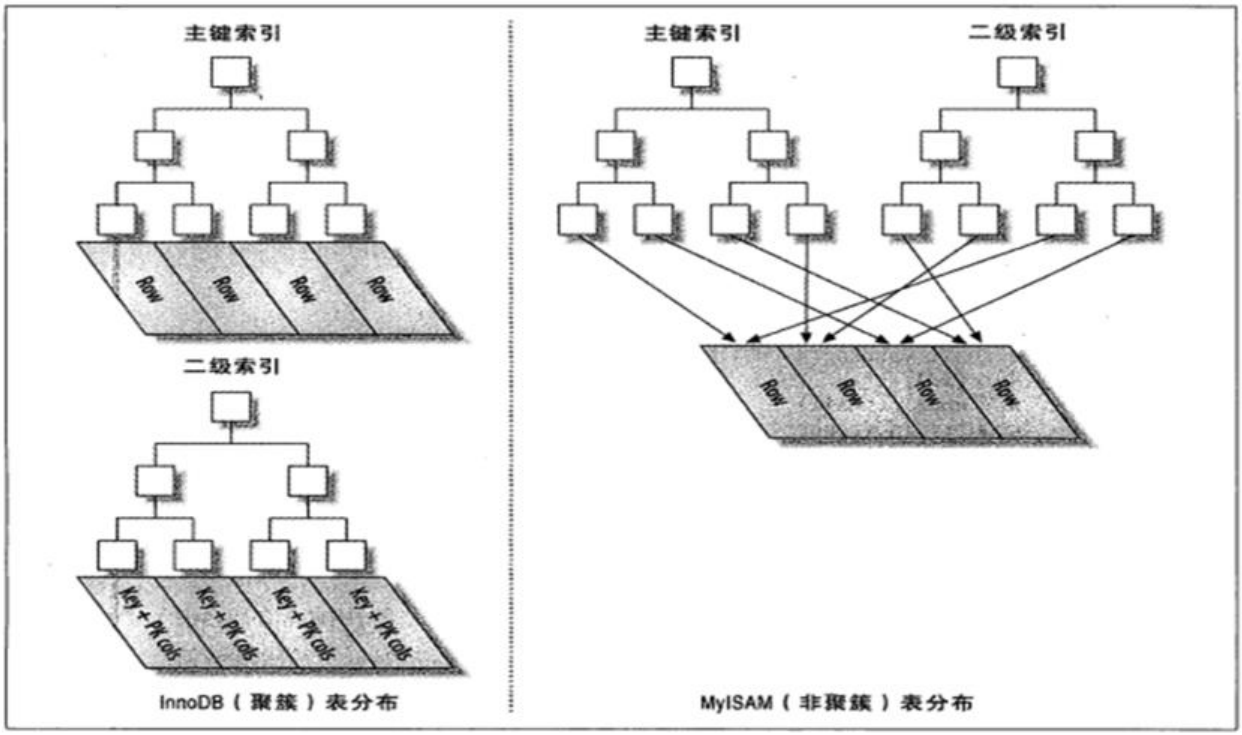
InnoDB从MySQL5.7之后也开始支持

全文索引(FULLTEXT)

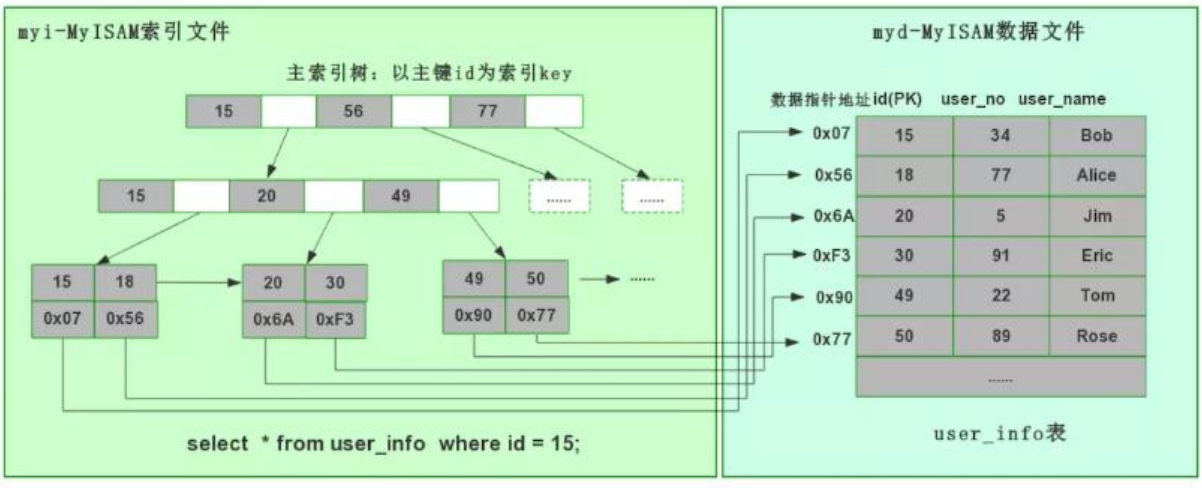
在文本中查找关键词，而不是直接比较索引中的值，类似搜索引擎

InnoDB从MySQL 5.6之后也开始支持

聚簇和非聚簇索引，主键和二级索引



MyISAM 主索引结构-非聚簇索引



MyISAM非聚簇索引

原文链接：6-MySQL 架构和性能优化（重点）

冗余和重复索引：

- 冗余索引：（A），（A，B）,注意如果同时存在,仍可能会使用(A)索引
- 重复索引：已经有索引，再次建立索引

4.6.3 索引优化

参考资料: 阿里的《Java开发手册》

https://developer.aliyun.com/topic/java2020

- 独立地使用列：尽量避免其参与运算，独立的列指索引列不能是表达式的一部分，也不能是函数的数，在where条件中，始终将索引列单独放在比较符号的一侧，尽量不要在列上进行运算（函数操作表达式操作）
- 左前缀索引：构建指定索引字段的左侧的字符数，要通过索引选择性（不重复的索引值和数据表的录总数的比值）来评估，尽量使用短索引，如果可以，应该制定一个前缀长度
- 多列索引：AND操作时更适合使用多列索引，而非为每个列创建单独的索引
- 选择合适的索引列顺序：无排序和分组时，将选择性最高放左侧
- 只要列中含有NULL值，就最好不要在此列设置索引，复合索引如果有NULL值，此列在使用时也不使用索引
- 对于经常在where子句使用的列，最好设置索引
- 对于有多个列where或者order by子句，应该建立复合索引
- 对于like语句，以 % 或者 _ 开头的不会使用索引，以 % 结尾会使用索引
- 尽量不要使用not in和<>操作,虽然可能使用索引,但性能不高
- 不要使用RLIKE正则表达式会导致索引失效
- 查询时，能不要 就不用，尽量写全字段名，比如:select id,name,age from students;
- 大部分情况连接效率远大于子查询
- 在有大量记录的表分页时使用limit
- 对于经常使用的查询，可以开启查询缓存
- 多使用explain和profile分析查询语句
- 查看慢查询日志，找出执行时间长的sql语句优化

4.6.4 管理索引

创建索引：

```
CREATE [UNIQUE] INDEX index_name ON tbl_name (index_col_name[(length)],...);
ALTER TABLE tbl_name ADD INDEX index_name(index_col_name[(length)]);
help CREATE INDEX;
```

删除索引：

```
DROP INDEX index_name ON tbl_name;
ALTER TABLE tbl_name DROP INDEX index_name(index_col_name);
```

查看索引：

```
SHOW INDEXES FROM [db_name.]tbl_name;
```

优化表空间：

```
OPTIMIZE TABLE tb_name;
```

查看索引的使用

```
SET GLOBAL userstat=1;  #MySQL无此变量
SHOW INDEX_STATISTICS;
```

范例：

```
mysql> desc students;
+----------+-----------------+------+-----+---------+----------------+
| Field    | Type            | Null | Key | Default | Extra          |
+----------+-----------------+------+-----+---------+----------------+
| StuID    | int unsigned    | NO   | PRI | NULL    | auto_increment |
| Name     | varchar(50)     | NO   |     | NULL    |                |
| Age      | tinyint unsigned | NO   |     | NULL    |                |
| Gender   | enum('F','M')   | NO   |     | NULL    |                |
| ClassID  | tinyint unsigned | YES  |     | NULL    |                |
| TeacherID | int unsigned   | YES  |     | NULL    |                |
+----------+-----------------+------+-----+---------+----------------+
6 rows in set (0.01 sec)

mysql> SHOW INDEXES FROM students\G
*************************** 1. row ***************************
Table: students
Non_unique: 0
Key_name: PRIMARY
Seq_in_index: 1
Column_name: StuID
Collation: A
Cardinality: 25
Sub_part: NULL
Packed: NULL
Null:
Index_type: BTREE
Comment:
Index_comment:
Visible: YES
Expression: NULL
1 row in set (0.00 sec)
mysql> select * from students where stuid=12;
+-------+-------------+-----+--------+---------+-----------+
| StuID | Name        | Age | Gender | ClassID | TeacherID |
+-------+-------------+-----+--------+---------+-----------+
|    12 | Wen Qingqing |  19 | F      |       1 |      NULL |
+-------+-------------+-----+--------+---------+-----------+
1 row in set (0.00 sec)
```

4.6.5 EXPLAIN 工具

可以通过EXPLAIN来分析索引的有效性,获取查询执行计划信息，用来查看查询优化器如何执行查询

参考资料： https://dev.mysql.com/doc/refman/5.7/en/explain-output.html

语法：

EXPLAIN SELECT clause

EXPLAIN输出信息说明：

列名	说明
id	执行编号，标识select所属的行。如果在语句中没子查询或关联查询，只有唯一的select，每行都将显示1。否则，内层的select语句一般会顺序编号，对应于其在原始语句中的位置
select_type	简单查询：SIMPLE| 复杂查询：PRIMARY（最外面的SELECT）、DERIVED（用于FROM中的子查询）、UNION（UNION语句的第一个之后的SELECT语句）、UNION RESUIT（匿名临时表）、SUBQUERY（简单子查询）
table	访问引用哪个表（引用某个查询，如“derived3”）
type	关联类型或访问类型，即MySQL决定的如何去查询表中的行的方式
possible_keys	查询可能会用到的索引
key	显示mysql决定采用哪个索引来优化查询
key_len	显示mysql在索引里使用的字节数
ref	当使用索引列等值查询时,与索引列进行等值匹配的对象信息
rows	为了找到所需的行而需要读取的行数，估算值，不精确。通过把所有rows列值相乘，可粗略估算整个查询会检查的行数
Extra	额外信息 Using index：MySQL将会使用覆盖索引，以避免访问表 Using where：MySQL服务器将在存储引擎检索后，再进行一次过滤 Using temporary：MySQL对结果排序时会使用临时表 Using filesort：对结果使用一个外部索引排序

说明： type显示的是访问类型，是较为重要的一个指标，结果值从好到坏依次是：NULL> system > onst > eq_ref > ref > fulltext > ref_or_null > index_merge > unique_subquery > index_subque y >range > index > ALL ，一般来说，得保证查询至少达到range级别，最好能达到ref

NULL>system>const>eq_ref>ref>fulltext>ref_or_null>index_merge>unique_subquery>index_subquery>range>index>ALL //最好到最差
备注：掌握以下10种常见的即可
NULL>system>const>eq_ref>ref>ref_or_null>index_merge>range>index>ALL

类型	说明
All	最坏的情况,全表扫描
index	和全表扫描一样。只是扫描表的时候按照索引次序进行而不是行。主要优点就是避免了排序, 但是开销仍然非常大。如在Extra列看到Using index，说明正在使用覆盖索引，只扫描索引的数据，它比按索引次序全表扫描的开销要小很多
range	范围扫描，一个有限制的索引扫描。key 列显示使用了哪个索引。当使用=、 <>、>、>=、<、<=、IS NULL、<=>、BETWEEN 或者 IN 操作符,用常量比较关键字列时,可以使用 range
ref	一种索引访问，它返回所有匹配某个单个值的行。此类索引访问只有当使用非唯一性索引或唯一性索引非唯一性前缀时才会发生。这个类型跟eq_ref不同的是，它用在关联操作只使用了索引的最左前缀，或者索引不是UNIQUE和PRIMARY KEY。ref可以用于使用=或<=>操作符的带索引的列。
eq_ref	最多只返回一条符合条件的记录。使用唯一性索引或主键查找时会发生 （高效）
const	当确定最多只会有一行匹配的时候，MySQL优化器会在查询前读取它而且只读取一次，因此非常快。当主键放入where子句时，mysql把这个查询转为一个常量（高效）
system	这是const连接类型的一种特例，表仅有一行满足条件。
Null	意味着mysql能在优化阶段分解查询语句，在执行阶段甚至用不到访问表或索引（高效）

范例：

```
explain select * from students where stuid=1;
```

范例:创建索引和使用索引

```
mysql> create index stu_name on students(name(10));
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show index from students\G
*************************** 1. row ***************************
Table: students
Non_unique: 0
Key_name: PRIMARY
Seq_in_index: 1
Column_name: StuID
Collation: A
Cardinality: 25
Sub_part: NULL
Packed: NULL
Null:
Index_type: BTREE
Comment:
Index_comment:
Visible: YES
Expression: NULL
*************************** 2. row ***************************
Table: students
Non_unique: 1
Key_name: stu_name
Seq_in_index: 1
Column_name: Name
```

```
Collation: A
Cardinality: 25
Sub_part: 10
Packed: NULL
Null:
Index_type: BTREE
Comment:
Index_comment:
Visible: YES
Expression: NULL
2 rows in set (0.00 sec)
mysql> mysql> explain select * from students where name like 'w%';
+----+-------------+----------+------------+-------+---------------+----------+---------+------+------+----------+-------------+
| id | select_type | table    | partitions | type  | possible_keys | key      | key_len | ref  | rows | filtered | Extra       |
+----+-------------+----------+------------+-------+---------------+----------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | students | NULL       | range | stu_name      | stu_name | 32      | NULL |    1 |   100.00 | Using where |
+----+-------------+----------+------------+-------+---------------+----------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
mysql> explain select * from students where name like '%x';
+----+-------------+----------+------------+------+---------------+------+---------+------+------+----------+-------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered | Extra       |
+----+-------------+----------+------------+------+---------------+------+---------+------+------+----------+-------------+
|  1 | SIMPLE      | students | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   25 |    11.11 | Using where |
+----+-------------+----------+------------+------+---------------+------+---------+------+------+----------+-------------+
1 row in set, 1 warning (0.00 sec)
```

范例: 复合索引

```
mysql> desc students;
+-----------+-----------------+------+-----+---------+----------------+
| Field     | Type            | Null | Key | Default | Extra          |
+-----------+-----------------+------+-----+---------+----------------+
| StuID     | int unsigned    | NO   | PRI | NULL    | auto_increment |
| Name      | varchar(50)     | NO   | MUL | NULL    |                |
| Age       | tinyint unsigned | NO   |     | NULL    |                |
| Gender    | enum('F','M')   | NO   |     | NULL    |                |
| ClassID   | tinyint unsigned | YES  |     | NULL    |                |
| TeacherID | int unsigned    | YES  |     | NULL    |                |
+-----------+-----------------+------+-----+---------+----------------+
#创建复合索引
mysql> create index stu_name_age on students(name,age);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> show index from students\G
```

```
*************************** 1. row ***************************
Table: students
Non_unique: 0
Key_name: PRIMARY
Seq_in_index: 1
Column_name: StuID
Collation: A
Cardinality: 25
Sub_part: NULL
Packed: NULL
Null:
Index_type: BTREE
Comment:
Index_comment:
Visible: YES
Expression: NULL
*************************** 2. row ***************************
Table: students
Non_unique: 1
Key_name: stu_name_age
Seq_in_index: 1
Column_name: Name
Collation: A
Cardinality: 25
Sub_part: NULL
Packed: NULL
Null:
Index_type: BTREE
Comment:
Index_comment:
Visible: YES
Expression: NULL
*************************** 3. row ***************************
Table: students
Non_unique: 1
Key_name: stu_name_age
Seq_in_index: 2
Column_name: Age
Collation: A
Cardinality: 25
Sub_part: NULL
Packed: NULL
Null:
Index_type: BTREE
Comment:
Index_comment:
Visible: YES
Expression: NULL
3 rows in set (0.00 sec)
#跳过查询复合索引的前面字段,后续字段的条件查询无法利用复合索引
mysql> mysql> explain select * from students where age=20;
+----+-------------+---------+------------+------+---------------+------+---------+------+-----
+----------+-------------+
| id | select_type | table    | partitions | type | possible_keys | key  | key_len | ref  | rows | filtered
```

```
 Extra       |
+----+-------------+---------+------------+------+---------------+------+---------+------+------
+----------+-------------+
|  1 | SIMPLE      | students | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   25 |    10.00
 Using where |
+----+-------------+---------+------------+------+---------------+------+---------+------+------
+----------+-------------+
#不跳过
mysql> explain select * from students where name='Duan Yu' and age=20;
+----+-------------+---------+------------+------+---------------+-------------+---------+-------
--------+------+----------+-------+
| id | select_type | table    | partitions | type | possible_keys | key         | key_len | ref         | rows
 filtered | Extra |
+----+-------------+---------+------------+------+---------------+-------------+---------+-------
--------+------+----------+-------+
|  1 | SIMPLE      | students | NULL       | ref  | stu_name_age  | stu_name_age | 153     | const,con
t |    1 |   100.00 | NULL  |
+----+-------------+---------+------------+------+---------------+-------------+---------+-------
--------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

4.7 并发控制

4.7.1 锁机制

锁类型：

- 读锁：共享锁，也称为 S 锁,只读不可写（包括当前事务），多个读互不阻塞
- 写锁：独占锁，排它锁，也称为 X 锁,写锁会阻塞其它事务（不包括当前事务）的读和写S 锁和 S 是兼容的，X 锁和其它锁都不兼容，举个例子，事务 T1 获取了一个行 r1 的 S 锁，另外事务 T2 可以即获得行 r1 的 S 锁，此时 T1 和 T2 共同获得行 r1 的 S 锁，此种情况称为锁兼容，但是另外一个事务 T2 此时如果想获得行 r1 的 X 锁，则必须等待 T1 对行 r1 锁的释放，此种情况也称为锁冲突

锁粒度：

- 表级锁：MyISAM
- 行级锁：InnoDB

实现

- 存储引擎：自行实现其锁策略和锁粒度
- 服务器级：实现了锁，表级锁，用户可显式请求

分类：

- 隐式锁：由存储引擎自动施加锁
- 显式锁：用户手动请求

锁策略：在锁粒度及数据安全性寻求的平衡机制

4.7.2 显式使用锁

帮助:https://mariadb.com/kb/en/lock-tables/

加锁

```
LOCK TABLES tbl_name [[AS] alias] lock_type [, tbl_name [[AS] alias]
lock_type] ...
lock_type:
READ   #读锁
WRITE  #写锁
```

解锁

```
UNLOCK TABLES
```

关闭正在打开的表（清除查询缓存），通常在备份前加全局读锁

```
FLUSH TABLES [tb_name[,...]] [WITH READ LOCK]
```

查询时加写或读锁

```
SELECT clause [FOR UPDATE | LOCK IN SHARE MODE]
```

范例: 加读锁

```
mysql> lock tables students read;
Query OK, 0 rows affected (0.00 sec)

mysql> update students set classid=2 where stuid=24;
ERROR 1099 (HY000): Table 'students' was locked with a READ lock and can't be updated
mysql> unlock tables;
Query OK, 0 rows affected (0.01 sec)

mysql> update students set classid=2 where stuid=24;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

4.7.3 事务

事务 Transactions：一组原子性的 SQL语句，或一个独立工作单元

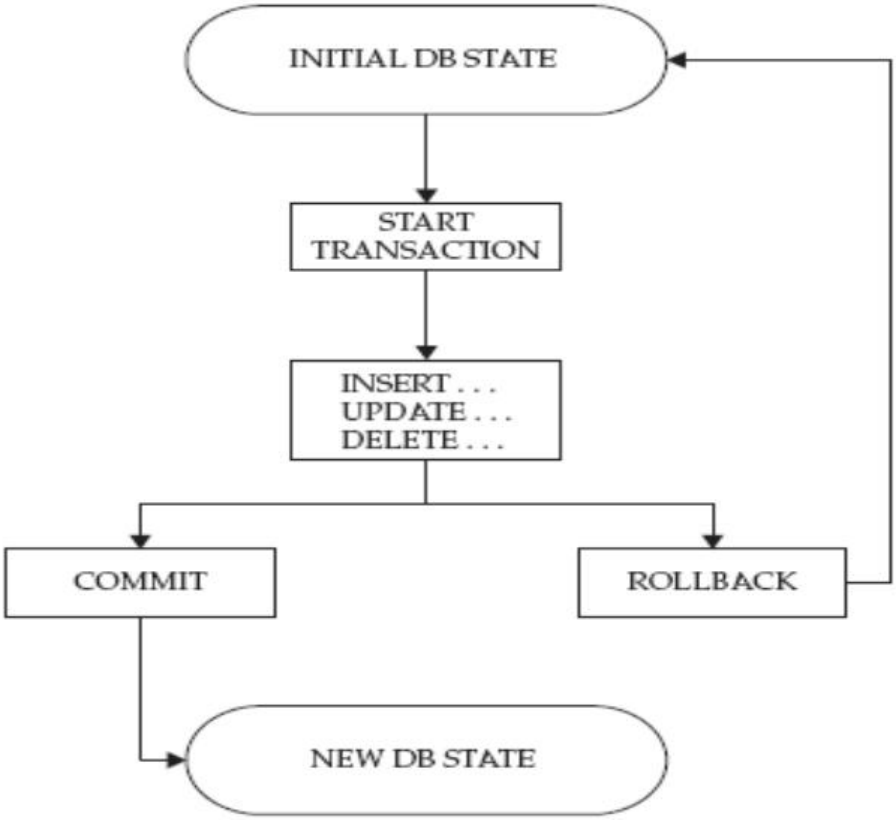
事务日志：记录事务信息，实现undo,redo等故障恢复功能

4.7.3.1 事务特性

ACID特性：

- A：atomicity 原子性；整个事务中的所有操作要么全部成功执行，要么全部失败后回滚
- C：consistency一致性；数据库总是从一个一致性状态转换为另一个一致性状态,类似于能量守恒律(N50周启皓语录)
- I：Isolation隔离性；一个事务所做出的操作在提交之前，是不能为其它事务所见；隔离有多种隔离别，实现并发
- D：durability持久性；一旦事务提交，其所做的修改会永久保存于数据库中

Transaction 生命周期



4.7.3.2 管理事务

显式启动事务：

```
BEGIN
BEGIN WORK
START TRANSACTION
```

结束事务

```
#提交,相当于vi中的wq保存退出
COMMIT
#回滚,相当于vi中的q!不保存退出
ROLLBACK
```

注意：只有事务型存储引擎中的DML语句方能支持此类操作

自动提交：

```
set autocommit={1|0}     mysql与mariadb中默认为1
```

默认为1，为0时设为非自动提交

建议：显式请求和提交事务，而不要使用"自动提交"功能

事务支持保存点：

```
SAVEPOINT identifier
ROLLBACK [WORK] TO [SAVEPOINT] identifier
RELEASE SAVEPOINT identifier
```

查看事务：

```
#查看当前正在进行的事务
SELECT * FROM INFORMATION_SCHEMA.INNODB_TRX;
#以下两张表在MySQL8.0中已取消
#查看当前锁定的事务
SELECT * FROM INFORMATION_SCHEMA.INNODB_LOCKS;
#查看当前等锁的事务
SELECT * FROM INFORMATION_SCHEMA.INNODB_LOCK_WAITS;
```

死锁：

两个或多个事务在同一资源相互占用，并请求锁定对方占用的资源的状态

范例：找到未完成的导致阻塞的事务

```
#在第一个会话中执行
mysql> begin;
Query OK, 0 rows affected (0.00 sec)

mysql> update students set classid=10;
Query OK, 25 rows affected (0.00 sec)
Rows matched: 25  Changed: 25  Warnings: 0
#在第二个会话中执行
mysql> update students set classid=20;
#在第三个会话中执行
show engine innodb status;
#查看正在进行的事务
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TRX\G
#杀掉未完成的事务
mysql> kill 10;
#查看事务锁的超时时长，默认50s
show global variables like 'innodb_lock_wait_timeout';
```

4.7.3.3 事务隔离级别

MySQL 支持四种隔离级别，事务隔离级别从上至下更加严格

隔离级别	脏读	不可重复读	幻读	加读锁
读未提交	可以出现	可以出现	可以出现	否
读提交	不允许出现	可以出现	可以出现	否
可重复读	不允许出现	不允许出现	可以出现	否
序列化	不允许出现	不允许出现	不允许出现	是

- READ UNCOMMITTED

可读取到未提交数据，产生**脏读**

● READ COMMITTED

可读取到提交数据，但未提交数据不可读，产生**不可重复读**，即可读取到多个提交数据，导致每次读数据不一致

● REPEATABLE READ

可重复读，多次读取数据都一致，产生**幻读**，即读取过程中，即使有其它提交的事务修改数据，仍只读取到未修改前的旧数据。此为MySQL默认设置

● SERIALIZABLE

可串行化，未提交的读事务阻塞修改事务（加读锁，但不阻塞读事务），或者未提交的修改事务阻塞它事务的读写（加写锁，其它事务的读，写都不可以执行）。会导致**并发性能差**

MVCC和事务的隔离级别：

MVCC（多版本并发控制机制）只在READ COMMITTED和REPEATABLE READ两个隔离级别下工作其他两个隔离级别都和MVCC不兼容,因为READ UNCOMMITTED总是读取最新的数据行，而不是符当前事务版本的数据行。而SERIALIZABLE则会对所有读取的行都加锁

指定事务隔离级别：

● 服务器变量tx_isolation(MySQL8.0改名为transaction_isolation)指定，默认为REPEATABLE-REA，可在GLOBAL和SESSION级进行设置

```
#MySQL8.0之前版本
  SET tx_isolation='READ-UNCOMMITTED|READ-COMMITTED|REPEATABLE
  READ|SERIALIZABLE'
  #MySQL8.0
  SET transaction_isolation='READ-UNCOMMITTED|READ-COMMITTED|REPEATABLE
  READ|SERIALIZABLE'
```

● 服务器选项中指定

```
vim /etc/my.cnf
[mysqld]
transaction-isolation=SERIALIZABLE
```

4.8 日志管理

MySQL 支持丰富的日志类型，如下：

● 事务日志：transaction log

● 事务日志的写入类型为"追加"，因此其操作为"顺序IO"；通常也被称为：预写式日志 write ahead logging 事务日志文件： ib_logfile0， ib_logfile1

● 错误日志 error log

● 通用日志 general log

● 慢查询日志 slow query log

● 二进制日志 binary log

● 中继日志 reley log，在主从复制架构中，从服务器用于保存从主服务器的二进制日志中读取的事件

4.8.1 事务日志

事务日志：transaction log

- redo log：实现 WAL（Write Ahead Log) ,数据更新前先记录redo log
- undo log：保存与执行的操作相反的操作,用于实现rollback

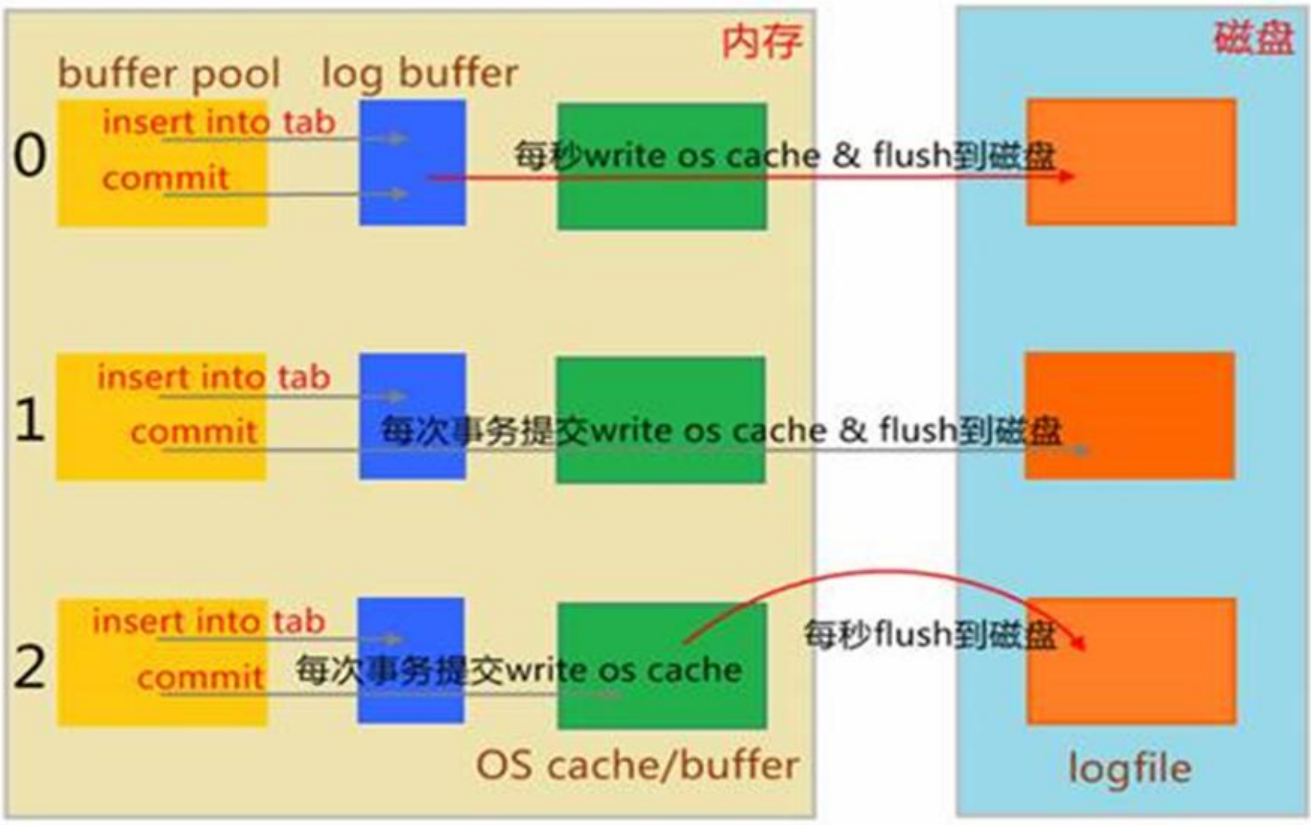
事务型存储引擎自行管理和使用，建议和数据文件分开存放

Innodb事务日志相关配置：

```
show variables like '%innodb_log%';
innodb_log_file_size   50331648 #每个日志文件大小
innodb_log_files_in_group 2     #日志组成员个数
innodb_log_group_home_dir ./ #事务文件路径
```

事务日志性能优化

```
innodb_flush_log_at_trx_commit=0|1|2
```



1 此为默认值，日志缓冲区将写入日志文件，并在每次事务后执行刷新到磁盘。 这是完全遵守ACID性
0 提交时没有写磁盘的操作; 而是每秒执行一次将日志缓冲区的提交的事务写入刷新到磁盘。 这样可供更好的性能，但服务器崩溃可能丢失最后一秒的事务
2 每次提交后都会写入OS的缓冲区，但每秒才会进行一次刷新到磁盘文件中。 性能比0略差一些，但作系统或停电可能导致最后一秒的交易丢失

高并发业务行业最佳实践，是使用第三种折衷配置（=2）：

1.配置为2和配置为0，性能差异并不大，因为将数据从Log Buffer拷贝到OS cache，虽然跨越用户与内核态，但毕竟只是内存的数据拷贝，速度很快
2.配置为2和配置为0，安全性差异巨大，操作系统崩溃的概率相比MySQL应用程序崩溃的概率，小多，设置为2，只要操作系统不奔溃，也绝对不会丢数据

说明：

- 设置为1，同时sync_binlog = 1表示最高级别的容错
- innodb_use_global_flush_log_at_trx_commit=0 时，将不能用SET语句重置此变量（ MariaDB10 2.6 后废弃）

4.8.2 错误日志

错误日志

- mysqld启动和关闭过程中输出的事件信息
- mysqld运行中产生的错误信息
- event scheduler运行一个event时产生的日志信息
- 在主从复制架构中的从服务器上启动从服务器线程时产生的信息

错误文件路径

```
show global variables like 'log_error';
```

记录哪些警告信息至错误日志文件

```
#CentOS7 mariadb 5.5 默认值为1
#CentOS8 mariadb 10.3 默认值为2
log_warnings=0|1|2|3...          #MySQL5.7之前
log_error_verbosity=0|1|2|3...   #MySQL8.0
```

4.8.3 通用日志

通用日志：记录对数据库的通用操作，包括:错误的SQL语句

通用日志可以保存在：file（默认值）或 table（mysql.general_log表）

通用日志相关设置

```
general_log=ON|OFF              #开启或关闭通用日志
general_log_file=HOSTNAME.log  #通用日志记录位置
log_output=TABLE|FILE|NONE     #日志输出方式，同时修改通用日志和慢查询日志
```

范例: 启用通用日志并记录至文件中

```
mysql> select @@general_log;
+---------------+
| @@general_log |
+---------------+
|             0 |
+---------------+
```

```
1 row in set (0.00 sec)

mysql> set global general_log=1;
Query OK, 0 rows affected (0.00 sec)

#默认通用日志存放在文件中
mysql> show variables like 'log_output';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_output    | FILE  |
+---------------+-------+
1 row in set (0.00 sec)
#通用文件存放路径
mysql> select @@general_log_file;
+----------------------------+
| @@general_log_file         |
+----------------------------+
| /var/lib/mysql/centos8.log |
+----------------------------+
1 row in set (0.00 sec)
```

范例：通用日志记录到表中

```
#修改通用日志，记录通用日志至mysql.general_log表中
mysql> set global log_output='table';
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'log_output';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| log_output    | TABLE |
+---------------+-------+
1 row in set (0.00 sec)
#general_log表是CSV格式的存储引擎
mysql> show table status like 'general_log'\G;
*************************** 1. row ***************************
Name: general_log
Engine: CSV
Version: 10
Row_format: Dynamic
Rows: 1
Avg_row_length: 0
Data_length: 0
Max_data_length: 0
Index_length: 0
Data_free: 0
Auto_increment: NULL
Create_time: 2021-03-01 18:50:35
Update_time: NULL
Check_time: NULL
Collation: utf8_general_ci
Checksum: NULL
```

```
Create_options:
Comment: General log
1 row in set (0.01 sec)
[19:58:19 root@centos8 ~]#file /var/lib/mysql/mysql/general_log.CSV
/var/lib/mysql/mysql/general_log.CSV: ASCII text
```

范例: 查找执行次数最多的前三条语句

```
mysql> select argument,count(argument) num from mysql.general_log group by argument o
der by num desc limit 3;
```

范例:对访问的语句进行排序

```
[20:20:35 root@centos8 ~]#mysql -e 'select argument from mysql.general_log' | awk '{sql[$0]
+}END{for(i in sql){print sql[i],i}}'|sort -nr
[20:22:41 root@centos8 ~]#mysql -e 'select argument from mysql.general_log' |sort |uniq -c |
ort -nr
```

4.8.4 慢查询日志

慢查询日志：记录执行查询时长超出指定时长的操作

慢查询相关变量

```
slow_query_log=ON|OFF #开启或关闭慢查询，支持全局和会话，只有全局设置才会生成慢查询文件
long_query_time=N     #慢查询的阀值，单位秒,默认为10s
slow_query_log_file=HOSTNAME-slow.log  #慢查询日志文件
log_slow_filter = admin,filesort,filesort_on_disk,full_join,full_scan,
query_cache,query_cache_miss,tmp_table,tmp_table_on_disk
#上述查询类型且查询时长超过long_query_time，则记录日志

log_queries_not_using_indexes=ON  #不使用索引或使用全索引扫描，不论是否达到慢查询阀值的
句是否记录日志，默认OFF，即不记录

log_slow_rate_limit = 1 #多少次查询才记录，mariadb特有
log_slow_verbosity= Query_plan,explain #记录内容
log_slow_queries = OFF    #同slow_query_log，MariaDB 10.0/MySQL 5.6.1 版后已删除
```

范例: 慢查询分析工具mysqldumpslow

```
[20:25:49 root@centos8 ~]#mysqldumpslow --help
[20:40:34 root@centos8 ~]#mysqldumpslow -s c -t 2 /var/lib/mysql/centos8-slow.log

Reading mysql slow query log from /var/lib/mysql/centos8-slow.log
Count: 9  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=4.0 (36), root[root]@localhost
select * from mysql.slow_log

Count: 8  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=1.0 (8), root[root]@localhost
select * from students where name='S'
```

4.8.5 使用 profile 工具

```
#打开后，会显示语句执行详细的过程
```

```
#mysql8.0已经弃用
set profiling = ON;
#查看语句,注意结果中的query_id值
show pshow profiles ;
#显示语句的详细执行步骤和时长
Show profile for query #
#显示cpu使用情况
Show profile cpu for query #
```

4.8.6 二进制日志(备份)

- 记录导致数据改变或潜在导致数据改变的SQL语句
- 记录已提交的日志
- 不依赖于存储引擎类型

功能：通过"重放"日志文件中的事件来生成数据副本

注意：建议二进制日志和数据文件分开存放

二进制日志记录三种格式

- 基于"语句"记录：statement，记录语句，默认模式（ MariaDB 10.2.3 版本以下 ），日志量较少
- 基于"行"记录：row，记录数据，日志量较大，更加安全，建议使用的格式,MySQL8.0默认格式
- 混合模式：mixed, 让系统自行判定该基于哪种方式进行，默认模式（ MariaDB 10.2.4及版本以上）

格式配置

```
MariaDB [hellodb]> show variables like 'binlog_format';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | MIXED |
+---------------+-------+
1 row in set (0.001 sec)
#mysql8.0默认ROW
mysql> show variables like 'binlog_format';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| binlog_format | ROW   |
+---------------+-------+
1 row in set (0.01 sec)
```

二进制日志文件的构成

```
有两类文件
1.日志文件：mysql|mariadb-bin.文件名后缀，二进制格式,如： on.000001,mariadb-bin.000002
2.索引文件：mysql|mariadb-bin.index，文本格式,记录当前已有的二进制日志文件列表
```

二进制日志相关的服务器变量：

```
sql_log_bin=ON|OFF：#是否记录二进制日志，默认ON，支持动态修改，系统变量，而非服务器选
```

log_bin=/PATH/BIN_LOG_FILE：#指定文件位置；默认OFF，表示不启用二进制日志功能，上述两都开启才可以
binlog_format=STATEMENT|ROW|MIXED：#二进制日志记录的格式，默认STATEMENT
max_binlog_size=1073741824：#单个二进制日志文件的最大体积，到达最大值会自动滚动，默认1G
#说明：文件达到上限时的大小未必为指定的精确值
binlog_cache_size=4m #此变量确定在每次事务中保存二进制日志更改记录的缓存的大小（每次连）
max_binlog_cache_size=512m #限制用于缓存多事务查询的字节大小。
sync_binlog=1|0：#设定是否启动二进制日志即时同步磁盘功能，默认0，由操作系统负责同步日志磁盘
expire_logs_days=N：#二进制日志可以自动删除的天数。 默认为0，即不自动删除

二进制日志相关配置

查看mariadb自行管理使用中的二进制日志文件列表，及大小

SHOW {BINARY | MASTER} LOGS

查看使用中的二进制日志文件

SHOW MASTER STATUS

在线查看二进制文件中的指定内容

SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]

范例:

mysql> show binlog events;
mysql> show binlog events in 'binlog.000001' from 12627\G;

mysqlbinlog：二进制日志的客户端命令工具，支持离线查看二进制日志

命令格式：

mysqlbinlog [OPTIONS] log_file...
--start-position=# 指定开始位置
--stop-position=#
--start-datetime= #时间格式：YYYY-MM-DD hh:mm:ss
--stop-datetime=
--base64-output[=name]
-v -vvv

范例：

mysqlbinlog --start-position=678 --stop-position=752 /var/lib/mysql/mariadb bin.000003 -v
mysqlbinlog --start-datetime="2018-01-30 20:30:10" --stop-datetime="2018-01-30 20:35:22" mariadb-bin.000003 -vvv

二进制日志事件的格式：

at 328

#151105 16:31:40 server id 1 end_log_pos 431 Query thread_id=1
exec_time=0 error_code=0
use `mydb`/*!*/;
SET TIMESTAMP=1446712300/*!*/;
CREATE TABLE tb1 (id int, name char(30))
/*!*/;
事件发生的日期和时间：151105 16:31:40
事件发生的服务器标识：server id 1
事件的结束位置：end_log_pos 431
事件的类型：Query
事件发生时所在服务器执行此事件的线程的ID：thread_id=1
语句的时间戳与将其写入二进制文件中的时间差：exec_time=0
错误代码：error_code=0
事件内容：
GTID：Global Transaction ID，mysql5.6以mariadb10以上版本专属属性：GTID

清除指定二进制日志

PURGE { BINARY | MASTER } LOGS { TO 'log_name' | BEFORE datetime_expr }

范例：

PURGE BINARY LOGS TO 'mariadb-bin.000003'; #删除mariadb-bin.000003之前的日志
PURGE BINARY LOGS BEFORE '2017-01-23';
PURGE BINARY LOGS BEFORE '2017-03-22 09:25:30';

删除所有二进制日志，index文件重新记数

RESET MASTER [TO #];
#删除所有二进制日志文件，并重新生成日志文件，文件名从#开始记数，默认从
1开始，一般是master主机第一次启动时执行，MariaDB 10.1.6开始支持TO #

切换日志文件：

FLUSH LOGS;