## MySQL 面试题汇总

#### 1.说一下 MySQL 执行一条查询语句的内部执行过程？

答：MySQL 执行一条查询的流程如下：

* 客户端先通过连接器连接到 MySQL 服务器；
* 连接器权限验证通过之后，先查询是否有查询缓存，如果有缓存（之前执行过此语句）则直接返回缓存数据，如果没有缓存则进入分析器；
* 分析器会对查询语句进行语法分析和词法分析，判断 SQL 语法是否正确，如果查询语法错误会直接返回给客户端错误信息，如果语法正确则进入优化器；
* 优化器是对查询语句进行优化处理，例如一个表里面有多个索引，优化器会判别哪个索引性能更好；
* 优化器执行完就进入执行器，执行器则开始执行语句进行查询比对了，直到查询到满足条件的所有数据，然后进行返回。

#### 2.MySQL 查询缓存有什么优缺点？

答：MySQL 查询缓存功能是在连接器之后发生的，它的优点是效率高，如果已经有缓存则会直接返回结果。

查询缓存的缺点是失效太频繁导致缓存命中率比较低，任何更新表操作都会清空查询缓存，因此导致查询缓存非常容易失效。

#### 3.MySQL 的常用引擎都有哪些？

答：MySQL 的常用引擎有 InnoDB、MyISAM、Memory 等，从 MySQL 5.5.5 版本开始 InnoDB 就成为了默认的存储引擎。

#### 4.常用的存储引擎 InnoDB 和 MyISAM 有什么区别？

答：InnoDB 和 MyISAM 最大的区别是 InnoDB 支持事务，而 MyISAM 不支持事务，它们其他主要区别如下：

* InnoDB 支持崩溃后安全恢复，MyISAM 不支持崩溃后安全恢复；
* InnoDB 支持行级锁，MyISAM 不支持行级锁，只支持到表锁；
* InnoDB 支持外键，MyISAM 不支持外键；
* MyISAM 性能比 InnoDB 高；
* MyISAM 支持 FULLTEXT 类型的全文索引，InnoDB 不支持 FULLTEXT 类型的全文索引，但是 InnoDB 可以使用 sphinx 插件支持全文索引，并且效果更好；
* InnoDB 主键查询性能高于 MyISAM。

#### 5.什么叫回表查询？

答：普通索引查询到主键索引后，回到主键索引树搜索的过程，我们称为回表查询。

#### 6.如果把一个 InnoDB 表的主键删掉，是不是就没有主键，就没办法进行回表查询了？

答：不是，如果把主键删掉了，那么 InnoDB 会自己生成一个长度为 6 字节的 rowid 作为主键。

#### 7.一张自增表中有三条数据，删除两条数据之后重启数据库，再新增一条数据，此时这条数据的 ID 是几？

答：如果这张表的引擎是 MyISAM，那么 ID=4，如果是 InnoDB 那么 ID=2（MySQL 8 之前的版本）。

#### 8.什么是独立表空间和共享表空间？它们的区别是什么？

答：共享表空间指的是数据库的所有表数据，索引文件全部放在一个文件中，默认这个共享表空间的文件路径在 data 目录下。

独立表空间：每一个表都将会生成以独立的文件方式来进行存储。

共享表空间和独立表空间最大的区别是如果把表放再共享表空间，即使表删除了空间也不会删除，因此表依然很大，而独立表空间如果删除表就会清除空间。

#### 9.清空表的所有数据性能最好的语句是？

A：delete from t  
B：delete t  
C：drop table t  
D：truncate table t

答：D

题目解析：truncate 清除表数据不会写日志，delete 要写日志，因此 truncate 的效率要高于 delete。

#### 10.唯一索引和普通索引哪个性能更好？

答：唯一索引和普通索引的性能对比分为以下两种情况：

* 对于查询来说两者都是从索引树进行查询，性能几乎没有任何区别；
* 对于更新操作来说，因为唯一索引需要先将数据读取到内存，然后需要判断是否有冲突，因此比唯一索引要多了判断操作，从而性能就比普通索引性能要低。

#### 11.left join 和 right join 的区别是什么？

答：left join 和 right join 的区别如下：

* left join（左联结），返回左表全部记录和右表联结字段相等的记录；
* right join（右联结），返回右表全部记录和左表联结字段相等的记录。

#### 12.什么是最左匹配原则？它的生效原则有哪些？

答：最左匹配原则也叫最左前缀原则，是 MySQL 中的一个重要原则，指的是索引以最左边为起点任何连续的索引都能匹配上，当遇到范围查询（>、<、between、like）就会停止匹配。 生效原则来看以下示例，比如表中有一个联合索引字段 index(a,b,c)：

* where a=1 只使用了索引 a；
* where a=1 and b=2 只使用了索引 a,b；
* where a=1 and b=2 and c=3 使用a,b,c；
* where b=1 or where c=1 不使用索引；
* where a=1 and c=3 只使用了索引 a；
* where a=3 and b like ‘xx%’ and c=3 只使用了索引 a,b。

#### 13.以下 or 查询有什么问题吗？该如何优化？

select \* from t where num=10 or num=20;

答：如果使用 or 查询会使 MySQL 放弃索引而全表扫描，可以改为：

select \* from t where num=10

union

select \* from t where num=20;

#### 14.事务是什么？它有什么特性？

答：事务是一系列的数据库操作，是数据库应用的基本单位。

在 MySQL 中只有 InnoDB 引擎支持事务，它的四个特性如下：

* 原子性（Atomic），要么全部执行，要么全部不执行；
* 一致性（Consistency），事务的执行使得数据库从一种正确状态转化为另一种正确状态；
* 隔离性（Isolation），在事务正确提交之前，不允许把该事务对数据的任何改变提供给其他事务；
* 持久性（Durability），事务提交后，其结果永久保存在数据库中。

#### 15.MySQL 中有几种事务隔离级别？分别是什么？

答：MySQL 中有四种事务隔离级别，分别是：

* read uncommited，未提交读，读到未提交数据；
* read committed，读已提交，也叫不可重复读，两次读取到的数据不一致；
* repetable read，可重复读；
* serializable，串行化，读写数据都会锁住整张表，数据操作不会出错，但并发性能极低，开发中很少用到。

MySQL 默认使用 repetable read 的事务隔离级别。

#### 16.如何设置 MySQL 的事务隔离级别？

答：MySQL 事务隔离级别 mysql.cnf 文件里设置的（默认目录 /etc/my.cnf），在文件的文末添加配置：

transaction-isolation = REPEATABLE-READ

可用的配置值：READ-UNCOMMITTED、READ-COMMITTED、REPEATABLE-READ、SERIALIZABLE。

#### 17.MySQL 出现了中文乱码该如何解决？

答：解决 MySQL 中文乱码的问题，可以设置全局编码或设置某个数据库或表的编码为 utf8。 设置全局编码：

set character\_set\_client='utf8';  
set character\_set\_connection='utf8';  
set character\_set\_results='utf8';

设置数据库的编码：

alter database db character set utf8;

设置表的编码：

alter table t character set utf8;

#### 18.InnoDB 为什么要使用 B+ 树，而不是 B 树、Hash、红黑树或二叉树？

答：因为 B 树、Hash、红黑树或二叉树存在以下问题。

* B 树：不管叶子节点还是非叶子节点，都会保存数据，这样导致在非叶子节点中能保存的指针数量变少（有些资料也称为扇出），指针少的情况下要保存大量数据，只能增加树的高度，导致 IO 操作变多，查询性能变低。
* Hash：虽然可以快速定位，但是没有顺序，IO 复杂度高。
* 二叉树：树的高度不均匀，不能自平衡，查找效率跟数据有关（树的高度），并且 IO 代价高。
* 红黑树：树的高度随着数据量增加而增加，IO 代价高。

#### 19.MySQL 是如何处理死锁？

答：MySQL 对待死锁常见的两种策略：

* 通过 innodb\_lock\_wait\_timeout 来设置超时时间，一直等待直到超时；
* 发起死锁检测，发现死锁之后，主动回滚死锁中的某一个事务，让其他事务继续执行。

#### 20.什么是全局锁？它的应用场景有哪些？

答：全局锁就是对整个数据库实例加锁，它的典型使用场景就是做全量逻辑备份，这个时候整个库会处于完全的只读状态。

#### 21.使用全局锁会导致什么问题？

答：使用全局锁会使整个系统不能执行更新操作，所有的更新业务会出于等待状态；如果你是在从库进行备份，则会导致主从同步严重延迟。

#### 22.InnoDB 存储引擎有几种锁算法？

答：InnoDB 的锁算法包括以下三种：

* Record Lock — 单个行记录上的锁；
* Gap Lock — 间隙锁，锁定一个范围，不包括记录本身；
* Next-Key Lock — 锁定一个范围，包括记录本身。

#### 23.InnoDB 如何实现行锁？

答：只有通过索引条件检索数据，InnoDB 才使用行级锁，否则 InnoDB 将使用表锁。使用 for update 来实现行锁，具体脚本如下：

select \* from t where id=1 for update

其中 id 字段必须有索引。

#### 24.MySQL 性能指标都有哪些？如何得到这些指标？

答：MySQL 最重要的性能指标有以下两个：

* QPS（Queries Per Second），每秒查询数，一台数据库每秒能够处理的查询次数；
* TPS（Transactions Per Second），每秒处理事务数。

这些性能指标可以通过 show status 来查询当前数据库状态的结果信息中估算出来，show status 会有 300 多条状态信息记录，其中以下这些信息 QPS 和 TPS 有关系：

* Uptime，服务器已经运行的时间，单位秒；
* Questions，已经发送给数据库查询数；
* Com\_select，查询次数，实际查询次数；
* Com\_insert，插入次数；
* Com\_delete，删除次数；
* Com\_update，更新次数；
* Com\_commit，事务次数；
* Com\_rollback，回滚次数。

#### 25.MySQL 中的重要日志分为哪几个？

**① 错误日志** ：用来记录 MySQL 服务器运行过程中的错误信息，比如，无法加载 MySQL 数据库的数据文件，或权限不正确等都会被记录在此，还有复制环境下，从服务器进程的信息也会被记录进错误日志。默认情况下，错误日志是开启的，且无法被禁止。默认情况下，错误日志是存储在数据库的数据文件目录中，名称为 hostname.err，其中 hostname 为服务器主机名。在 MySQL 5.5.7 之前，数据库管理员可以删除很长时间之前的错误日志，以节省服务器上的硬盘空间， MySQL 5.5.7 之后，服务器将关闭此项功能，只能使用重命名原来的错误日志文件，手动冲洗日志创建一个新的，命令为：

mv hostname.err hostname.err.old

mysqladmin flush-logs

**② 查询日志** ：查询日志在 MySQL 中被称为 general log（通用日志），查询日志里的内容不要被“查询日志”误导，认为里面只存储 select 语句，其实不然，查询日志里面记录了数据库执行的所有命令，不管语句是否正确，都会被记录，具体原因如下:

* insert 查询为了避免数据冲突，如果此前插入过数据，则当前插入的数据如果跟主键或唯一键的数据重复那肯定会报错；
* update 时也会查询因为更新的时候很可能会更新某一块数据；
* delete 查询，只删除符合条件的数据；

因此都会产生日志，在并发操作非常多的场景下，查询信息会非常多，那么如果都记录下来会导致 IO 非常大，影响 MySQL 性能。因此如果不是在调试环境下，是不建议开启查询日志功能的。

查询日志的开启有助于帮助我们分析哪些语句执行密集，执行密集的 select 语句对应的数据是否能够被缓存，同时也可以帮助我们分析问题，因此，可以根据自己的实际情况来决定是否开启查询日志。

查询日志模式是关闭的，可以通过以下命令开启查询日志：

set global general\_log=1

set global log\_output=‘table’;

general\_log=1 为开启查询日志，0 为关闭查询日志，这个设置命令即时生效，不用重启 MySQL 服务器。

**③ 慢日志** ：慢查询会导致 CPU、IOPS、内存消耗过高，当数据库遇到性能瓶颈时，大部分时间都是由于慢查询导致的。开启慢查询日志，可以让 MySQL 记录下查询超过指定时间的语句，之后运维人员通过定位分析，能够很好的优化数据库性能。默认情况下，慢查询日志是不开启的，只有手动开启了，慢查询才会被记录到慢查询日志中。使用如下命令记录当前数据库的慢查询语句：

set global slow\_query\_log=‘ON’;

使用 set global slow\_query\_log=‘ON’ 开启慢查询日志，只是对当前数据库有效，如果 MySQL 数据库重启后就会失效。因此如果要永久生效，就要修改配置文件 my.cnf，设置 slow\_query\_log=1 并重启 MySQL 服务器。

**④ redo log（重做日志）** ：为了最大程度的避免数据写入时，因为 IO 瓶颈造成的性能问题，MySQL 采用了这样一种缓存机制，先将数据写入内存中，再批量把内存中的数据统一刷回磁盘。为了避免将数据刷回磁盘过程中，因为掉电或系统故障带来的数据丢失问题，InnoDB 采用 redo log 来解决此问题。

**⑤ undo log（回滚日志）** ：用于存储日志被修改前的值，从而保证如果修改出现异常，可以使用 undo log 日志来实现回滚操作。

undo log 和 redo log 记录物理日志不一样，它是逻辑日志，可以认为当 delete 一条记录时，undo log 中会记录一条对应的 insert 记录，反之亦然，当 update 一条记录时，它记录一条对应相反的 update 记录，当执行 rollback 时，就可以从 undo log 中的逻辑记录读取到相应的内容并进行回滚。undo log 默认存放在共享表空间中，在 ySQL 5.6 中，undo log 的存放位置还可以通过变量 innodb\_undo\_directory 来自定义存放目录，默认值为“.”表示 datadir 目录。

**⑥ bin log（二进制日志）** ：是一个二进制文件，主要记录所有数据库表结构变更，比如，CREATE、ALTER TABLE 等，以及表数据修改，比如，INSERT、UPDATE、DELETE 的所有操作，bin log 中记录了对 MySQL 数据库执行更改的所有操作，并且记录了语句发生时间、执行时长、操作数据等其他额外信息，但是它不记录 SELECT、SHOW 等那些不修改数据的 SQL 语句。 binlog 的作用如下：

* 恢复（recovery）：某些数据的恢复需要二进制日志。比如，在一个数据库全备文件恢复后，用户可以通过二进制日志进行 point-in-time 的恢复；
* 复制（replication）：其原理与恢复类似，通过复制和执行二进制日志使一台远程的 MySQL 数据库（一般称为 slave 或者 standby）与一台 MySQL 数据库（一般称为 master 或者 primary）进行实时同步；
* 审计（audit）：用户可以通过二进制日志中的信息来进行审计，判断是否有对数据库进行注入攻击。

除了上面介绍的几个作用外，binlog 对于事务存储引擎的崩溃恢复也有非常重要的作用，在开启 binlog 的情况下，为了保证 binlog 与 redo 的一致性，MySQL 将采用事务的两阶段提交协议。当 MySQL 系统发生崩溃时，事务在存储引擎内部的状态可能为 prepared（准备状态）和 commit（提交状态）两种，对于 prepared 状态的事务，是进行提交操作还是进行回滚操作，这时需要参考 binlog，如果事务在 binlog 中存在，那么将其提交；如果不在 binlog 中存在，那么将其回滚，这样就保证了数据在主库和从库之间的一致性。

binlog 默认是关闭状态，可以在 MySQL 配置文件（my.cnf）中通过配置参数 log-bin = [base-name] 开启记录 binlog 日志，如果不指定 base-name，则默认二进制日志文件名为主机名，并以自增的数字作为后缀，比如：mysql- bin.000001，所在目录为数据库所在目录（datadir）。

通过以下命令来查询 binlog 是否开启：

show variables like ‘log\_%’;



1

binlog 格式分为 STATEMENT、ROW 和 MIXED 三种。

* STATEMENT 格式的 binlog 记录的是数据库上执行的原生 SQL 语句。这种格式的优点是简单，简单地记录和执行这些语句，能够让主备保持同步，在主服务器上执行的 SQL 语句，在从服务器上执行同样的语句。另一个好处是二进制日志里的时间更加紧凑，因此相对而言，基于语句的复制模式不会使用太多带宽，同时也节约磁盘空间，并且通过 mysqlbinlog 工具容易读懂其中的内容。缺点就是同一条 SQL 在主库和从库上执行的时间可能稍微或很大不相同，因此在传输的二进制日志中，除了查询语句，还包括了一些元数据信息，如当前的时间戳。即便如此，还存在着一些无法被正确复制的 SQL，比如，使用 INSERT INTO TB1 VALUE(CUURENT\_DATE()) 这一条使用函数的语句插入的数据复制到当前从服务器上来就会发生变化，存储过程和触发器在使用基于语句的复制模式时也可能存在问题；另外一个问题就是基于语句的复制必须是串行化的，比如，InnoDB 的 next-key 锁等，并不是所有的存储引擎都支持基于语句的复制。
* ROW 格式是从 MySQL 5.1 开始支持基于行的复制，也就是基于数据的复制，基于行的更改。这种方式会将实际数据记录在二进制日志中，它有其自身的一些优点和缺点，最大的好处是可以正确地复制每一行数据，一些语句可以被更加有效地复制，另外就是几乎没有基于行的复制模式无法处理的场景，对于所有的 SQL 构造、触发器、存储过程等都能正确执行；它的缺点就是二进制日志可能会很大，而且不直观，因此，你不能使用 mysqlbinlog 来查看二进制日志，也无法通过看二进制日志判断当前执行到那一条 SQL 语句。现在对于 ROW 格式的二进制日志基本是标配了，主要是因为它的优势远远大于缺点，并且由于 ROW 格式记录行数据，因此可以基于这种模式做一些 DBA 工具，比如数据恢复，不同数据库之间数据同步等。
* MIXED 也是 MySQL 默认使用的二进制日志记录方式，但 MIXED 格式默认采用基于语句的复制，一旦发现基于语句的无法精确的复制时，就会采用基于行的复制。比如用到 UUID()、USER()、CURRENT\_USER()、ROW\_COUNT() 等无法确定的函数。

#### 26.redo log 和 binlog 有什么区别？

redo log（重做日志）和 binlog（归档日志）都是 MySQL 的重要的日志，它们的区别如下：

* redo log 是物理日志，记录的是“在某个数据页上做了什么修改”；
* binlog 是逻辑日志，记录的是这个语句的原始逻辑，比如“给 ID=2 这一行的 c 字段加 1”；
* redo log 是 InnoDB 引擎特有的，binlog 是 MySQL 的 Server 层实现的，所有引擎都可以使用；
* redo log 是循环写的，空间固定会用完，binlog 是可以追加写入的，“追加写”是指 binlog 文件写到一定大小后会切换到下一个，并不会覆盖以前的日志。

最开始 MySQL 里并没有 InnoDB 引擎，MySQL 自带的引擎是 MyISAM，但是 MyISAM 没有 crash-safe 的能力，binlog 日志只能用于归档。而 InnoDB 是另一个公司以插件形式引入 MySQL 的，既然只依靠 binlog 是没有 crash-safe 能力的，因此 InnoDB 使用另外一套日志系统，也就是 redo log 来实现 crash-safe 能力。

#### 27.慢查询日志的获取方式有哪些？

答：慢查询日志的常见获取方式如下。

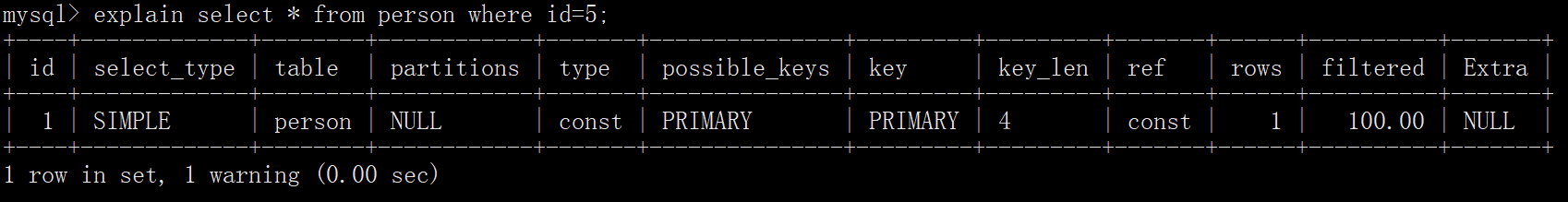
* 使用 MySQL 自带功能，开启慢查询日志，在 MySQL 的安装目录下找到 my.cnf 文件设置 slow-query-log=On 开启慢查询，慢查询默认时长为 10s，默认存储文件名为 host\_name-slow.log。
* 使用三方开源方案 zabbix，zabbix 是一个基于 Web 界面的提供分布式系统监视以及网络监视功能的企业级的开源解决方案，能监视各种网络参数，保证服务器系统的安全运营；并提供灵活的通知机制以让系统管理员快速定位/解决存在的各种问题。

#### 28.如何定位慢查询？

答：使用 MySQL 中的 explain 分析执行语句，比如：

explain select \* from t where id=5;

如下图所示：



2

其中：

* id — 选择标识符，id 越大优先级越高，越先被执行
* select\_type — 表示查询的类型。
* table — 输出结果集的表
* partitions — 匹配的分区
* type — 表示表的连接类型
* possible\_keys — 表示查询时，可能使用的索引
* key — 表示实际使用的索引
* key\_len — 索引字段的长度
* ref— 列与索引的比较
* rows — 大概估算的行数
* filtered — 按表条件过滤的行百分比
* Extra — 执行情况的描述和说明

其中最重要的就是 type 字段，type 值类型如下：

* all — 扫描全表数据
* index — 遍历索引
* range — 索引范围查找
* index\_subquery — 在子查询中使用 ref
* unique\_subquery — 在子查询中使用 eq\_ref
* ref\_or\_null — 对 null 进行索引的优化的 ref
* fulltext — 使用全文索引
* ref — 使用非唯一索引查找数据
* eq\_ref — 在 join 查询中使用主键或唯一索引关联
* const — 将一个主键放置到 where 后面作为条件查询， MySQL 优化器就能把这次查询优化转化为一个常量，如何转化以及何时转化，这个取决于优化器，这个比 eq\_ref 效率高一点

#### 29.MySQL 中常见的读写分离方案有哪些？

答：MySQL 中常见的读写分离方案通常为以下两种：

* 使用 MySQL 官方提供的数据库代理产品 MySql ProxySQL 搭建自动分配的数据库读写分离环境；
* 在程序层面配置多数据源使用代码实现读写分离。

#### 30.怎样保证主备数据库无延迟？

答：通常保证主备数据库无延迟有以下三种方法。

* 每次从库执行查询请求前，先判断 seconds\_behind\_master 是否已经等于 0。如果还不等于 0 ，那就必须等到这个参数变为 0 才能执行查询请求，seconds\_behind\_master 参数是用来衡量主备延迟时间的长短。
* 对比位点确保主备无延迟。Master\_Log\_File 和 Read\_Master\_Log\_Pos，表示的是读到的主库的最新位点，Relay\_Master\_Log\_File 和 Exec\_Master\_Log\_Pos，表示的是备库执行的最新位点。
* 对比 GTID 集合确保主备无延迟。Auto\_Position=1 ，表示这对主备关系使用了 GTID 协议。Retrieved\_Gtid\_Set，是备库收到的所有日志的 GTID 集合；Executed\_Gtid\_Set，是备库所有已经执行完成的 GTID 集合。

#### 31.什么是 MySQL 多实例，如何配置 MySQL 多实例？

答：MySQL 多实例就是在同一台服务器上启用多个 MySQL 服务，它们监听不同的端口，运行多个服务进程，它们相互独立，互不影响的对外提供服务，便于节约服务器资源与后期架构扩展。 多实例的配置方法有两种：

* 一个实例一个配置文件，不同端口；
* 同一配置文件（my.cnf）下配置不同实例，基于 MySQL 的 d\_multi 工具。

#### 32.表的优化策略有哪些？

「参考答案」常见的大表优化策略如下。

* 读写分离，主库负责写，从库负责读。
* 垂直分区，根据数据属性单独拆表甚至单独拆库。
* 水平分区，保持表结构不变，根据策略存储数据分片，这样每一片数据被分散到不同的表或者库中。水平拆分只是解决了单一表数据过大的问题，表数据还在同一台机器上，对于并发能力没有什么意义，因此水平拆分最好分库。另外分片事务难以解决，跨节点 join 性能较差。

#### 33.数据库分片方案有哪些？

「参考答案」数据库分片方案有哪些？ 答：数据库创建的分片方案有两种方式：客户端代理方式和中间件代理方式。

* 客户端代理 — 分片逻辑在应用端，封装在 jar 包中，通过修改或者封装 JDBC 层来实现，比如 Sharding-JDBC、阿里 TDDL 等。
* 中间件代理 — 在应用层和数据层中间加了一个代理层。分片逻辑统一维护在中间件服务中，比如 MyCat、网易的 DDB 都是中间件代理的典型代表。

#### 34.查询语句的优化方案有哪些？

「参考答案」常见优化方案如下：

* 不做列运算，把计算都放入各个业务系统实现；
* 查询语句尽可能简单，大语句拆小语句，减少锁时间；
* 不使用 select \* 查询；
* or 查询改写成 in 查询；
* 不用函数和触发器；
* 避免 %xx 查询；
* 少用 join 查询；
* 使用同类型比较，比如 ‘123’ 和 ‘123’、123 和 123；
* 尽量避免在 where 子句中使用 != 或者 <> 操作符，查询引用会放弃索引而进行全表扫描；
* 列表数据使用分页查询，每页数据量不要太大。

#### 35.MySQL 毫无规律的异常重启，可能产生的原因是什么？该如何解决？

「参考答案」可能是积累的长连接导致内存占用太多，被系统强行杀掉导致的异常重启，因为在 MySQL 中长连接在执行过程中使用的临时内存对象，只有在连接断开的时候才会释放，这就会导致内存不断飙升，解决方案如下：

* 定期断开空闲的长连接；
* 如果是用的是 MySQL 5.7 以上的版本，可以定期执行 mysql\_reset\_connection 重新初始化连接资源，这个过程会释放之前使用的内存资源，恢复到连接刚初始化的状态。

## 更多资源下载交流请加微信：Morstrong,加入永久会员,网盘更新更快捷！

# 本资源由微信公众号：光明顶一号，提供支持