Part 12 数据库其它调优策略

1. 数据库调优的措施

1.1 调优的目标

- 尽可能`节省系统资源`，以便系统可以提供更大负荷的服务。(吞吐量更大)。
- 合理的结构设计和参数调整，以提高用户操作`响应的速度`。(响应速度更快)
- 减少系统的瓶颈，提高MySQL数据库整体的性能。

1.2 如何定位调优问题

调优目标不仅仅是更快，因为用户在不同时间段访问服务器遇到的瓶颈不同。

- 双十一促销的时候会带来大规模的`并发访问`。
- 用户在进行不同业务操作的时候，数据库的`事务处理`和`SQL查询`都会有所不同。

确定调优问题的方式：

- **用户的反馈(主要)**
- **日志分析（主要)**
- 服务器资源使用监控
  - 通过监控服务器的CPU、内存、I/o等使用情况，可以实时了解服务器的性能使用，与历史情况进行对比。
- 数据库内部状况监控
  - 在数据库的监控中，`活动会话（Active Session)监控`是一个重要的指标。通过它，你可以清楚地了解数据库当前是否处于非常繁忙的状态，是否存在SQL堆积等。
- 其它
  - 除了活动会话监控以外，我们也可以对`事务、锁等待`等进行监控，这些都可以帮助我们对数据库的运行状态有更全面的认识。

1.3 调优的维度和步骤

第1步：选择适合的DBMS

- **事务性处理**以及**安全性要求高**，可以选择商业的数据库产品，比如采用SQL Server、Oracle。
  - 优势：在事务处理和查询性能上都比较强；`单表存储上亿条数据`是没有问题的；如果数据表设计得好，即使不采用`分库分表`的方式，查询效率也不差。
- **开源的MySQL**，有很多存储引擎可以选择
  - 事务处理：InnoDB
  - 非事务处理：MyISAM
- NoSQL有`键值型数据库`、`文档型数据库`、`搜索引擎`、`列式存储`和`图形数据库`。
  - 这些数据库的优缺点和使用场景各有不同，比如列式存储数据库可以大幅度降低系统的I/O，适合于分布式文件系统，但如果数据需要频繁地增删改，那么列式存储就不太适用了。

第2步：优化表设计

数据表的设计方式也直接影响了后续的SQL查询语句。

- RDBMS中，每个对象都可以定义为一张表，表与表之间的关系代表了对象之间的关系。
- 如果用的是MySQL，我们还可以根据不同表的使用需求，选择不同的存储引擎。
- 表结构要尽量`遵循三范式的原则`。这样可以让数据结构更加清晰规范减少冗余字段，同时也减少了在更新，插入和删除数据时等异常情况的发生。
- 如果`查询`应用比较多，尤其是需要进行`多表联查`的时候，可以采用`反范式`进行优化。反范式采用`空间换时间`的方式，通过增加冗余字段提高查询的效率。
- `表字段的数据类型选择`，关系到了查询效率的高低以及存储空间的大小。
    - 如果字段可以用数值类型就不要采用字符类型;字符长度要尽可能设计得短一些。
    - 针对字符类型来说，长度固定采用CHAR，不固定采用VARCHAR。

第3步：优化逻辑查询

SQL查询优化，可以分为`逻辑查询优化`和`物理查询优化`。

逻辑查询优化就是通过改变sQL语句的内容让sQL执行效率更高效，采用的方式是对sQL语句进行等价变换，对查询进行重写。SQL的查询重写包括了子查询优化、等价谓词重写、视图重写、条件简化、连接消除和嵌套连接消除等。

第4步：优化物理查询

物理查询优化是在确定了逻辑查询优化之后，采用物理优化技术(比如索引等)，通过计算代价模型对各种可能的访问路径进行估算，从而找到执行方式中代价最小的作为执行计划。**在这个部分中，我们需要掌握的重点是对索引的创建和使用。**但需要根据情况创建索引：

- `单表扫描`：对于单表扫描来说，我们可以全表扫描所有的数据，也可以局部扫描。
- `两张表的连接`：常用的连接方式包括了嵌套循环连接、HASH连接和合并连接。
- `多张表的连接`：多张数据表进行连接的时候，`顺序`很重要，因为不同的连接路径查询的效率不同，搜索空间也会不同。我们在进行多表连接的时候，搜索空间可能会达到`很高的数据量级`，巨大的搜索空间显然会占用更多的资源，因此我们需要通过调整连接顺序，将搜索空间调整在一个可接受的范围内。

第5步：使用Redis或Memcached作为缓存

用户量增大的时候会频繁查询，消耗资源，如果能将常用数据放在内存中，会大幅提升查询效率，可以用键值存储数据库可以帮我们解决这个问题。常用的键值存储数据库有Redis和Memcached。

可靠性来说，**Redis支持持久化**，可以让我们的数据保存在硬盘上，不过这样一来性能消耗也会比较大。而Memcached仅仅是内存存储，不支持持久化。

支持的数据类型来说，Redis 比Memcached要多，它不仅支持 key-value类型的数据，还支持List，Set，Hash等数据结构。

- 当我们有持久化需求或者是更高级的数据处理需求的时候，就可以使用Redis。

- 如果是简单的key-value存储，则可以使用Memcached。

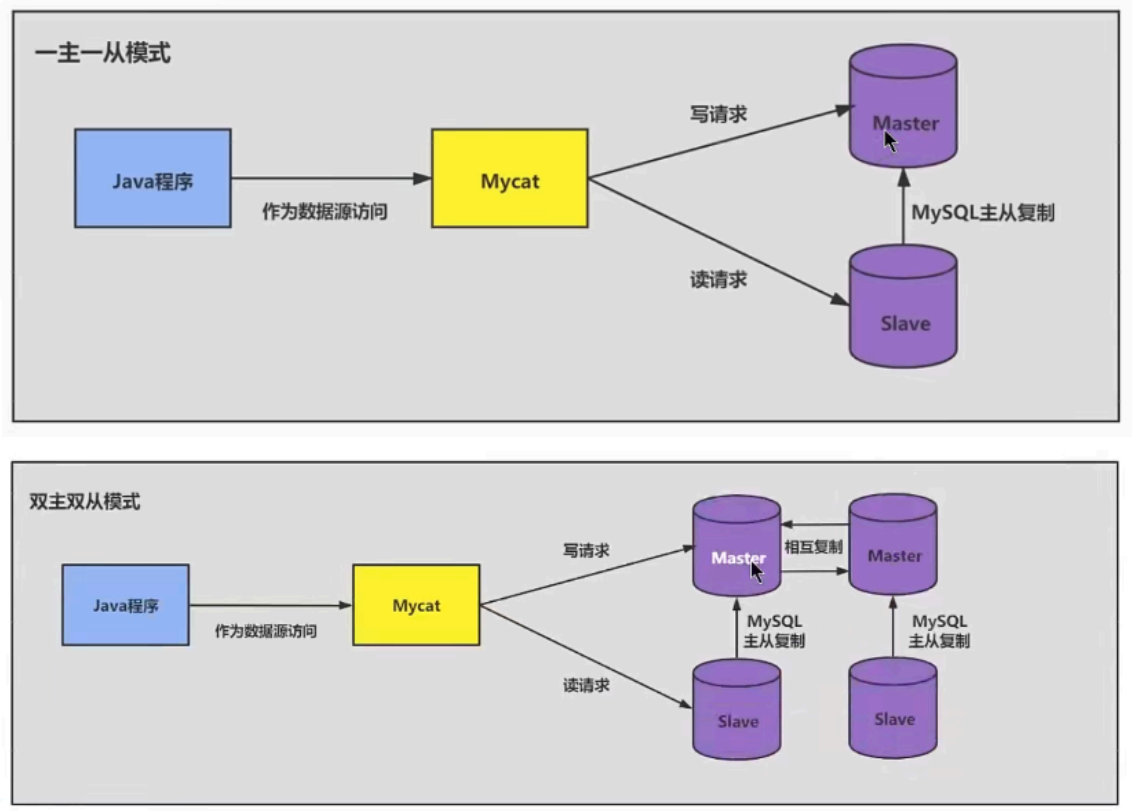
通常我们对于查询响应要求高的场景（响应时间短，吞吐量大），可以考虑内存数据库，毕竟术业有专攻。

第6步：库级优化

库级优化是站在数据库的维度上进行的优化策略，比如控制一个库中的数据表数量。另外，单一的数据库总会遇到各种限制，不如取长补短，利用"外援"的方式。通过`主从架构`优化我们的读写策略，通过对数据库进行垂直或者水平切分，突破单一数据库或数据表的访问限制，提升查询的性能。

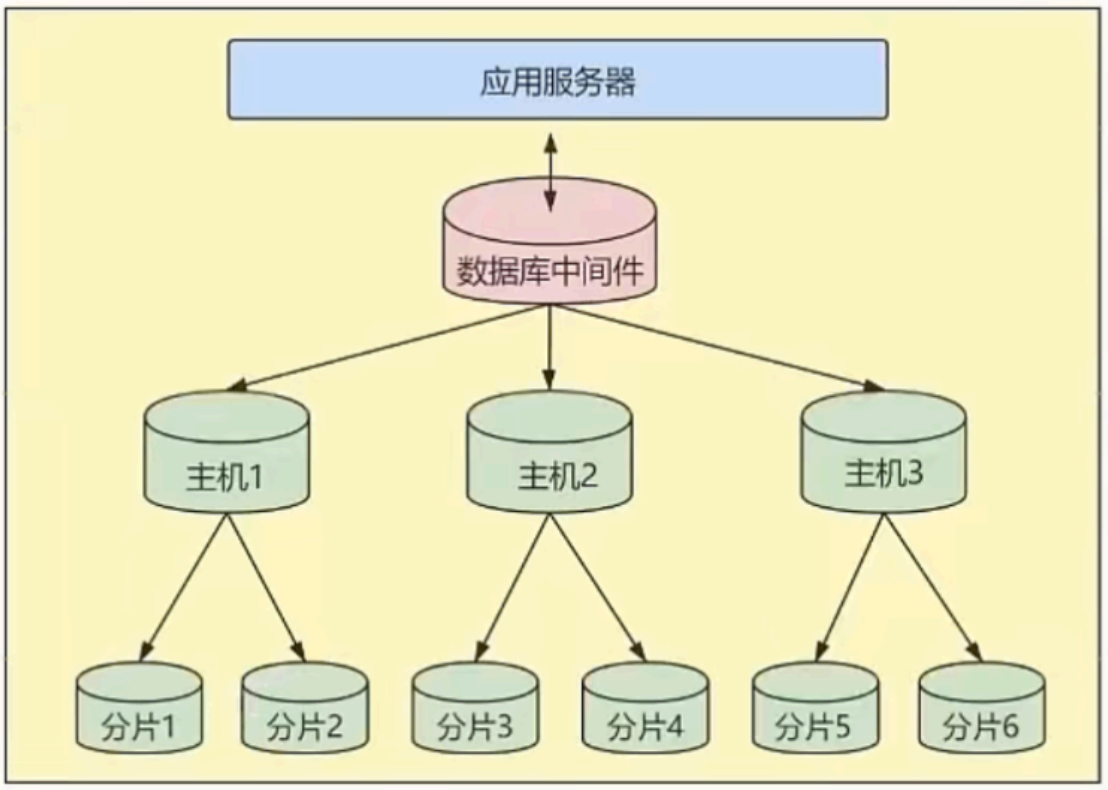
- 读写分离

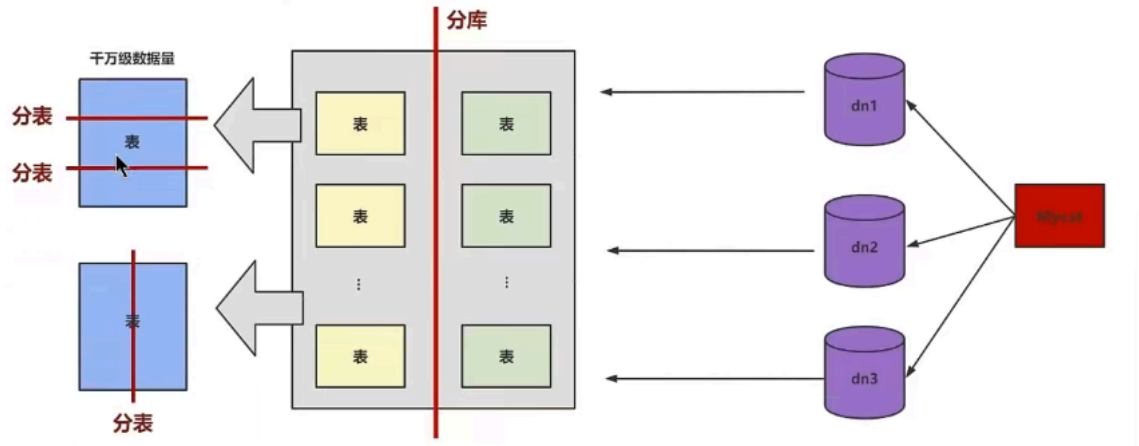
  我们可以采用`读写分离`的方式降低主数据库的负载，比如用主数据库(master)完成写操作，用从数据库(slave)完成读操作。



- 数据分片

  对数据库分库分表。当数据量级达到千万级以上时，有时候我们需要把一个数据库切成多份，放到不同的数据库服务器上，减少对单一数据库服务器的访问压力。如果你使用的是MySQL，就可以使用MySQL自带的分区表功能，当然你也可以考虑自己做`垂直拆分（分库）`、`水平拆分（分表）`、`垂直+水平拆分（分库分表）`。





> 但需要注意的是，分拆在提升数据库性能的同时，也会增加维护和使用成本。

2. 优化MySQL服务器

两个角度：硬件方面，MySQL服务的参数方面。一般只有专业的数据库管理员才能进行这类优化。对于可以定制参数的操作系统，也可以针对MySQL进行操作系统优化。

2.1 优化服务器硬件

服务器的硬件性能直接决定着MySQL数据库的性能。

- 配置较大的内存
- 配置高速磁盘系统
- 合理分布磁盘I/O，把磁盘I/O分散在多个设备上，以减少资源竞争提高并行操作能力。
- 配置多处理器，MySQL是多线程的数据库，多处理器可同时执行多个线程。

2.2 优化MySQL的参数

通过优化MySQL的参数可以提高资源利用率，从而达到提高MySQL服务器性能的目的。

MySQL服务的配置参数都在`my.cnf`或者`my. ini`文件的[mysqld]组中。配置完参数以后，需要重新启动MySQL服务才会生效。

下面对几个对性能影响比较大的参数进行详细介绍。

- innodb_buffer_pool_size
  - Mysql数据库最重要的参数之一，表示InnoDB类型的表和索引的最大缓存。它不仅仅缓存索引数据，还会缓存表的数据。这个值越大，查询的速度就会越快。但是这个值太大会影响操作系统的性能。
- key_buffer_size
  - 索引缓冲区的大小。索引缓冲区是所有的线程共享。增加索引缓冲区可以得到更好处理的索引(对所有读和多重写)。当然，这个值不是越大越好，它的大小取决于内存的大小。如果这个值太大，就会导致操作系统频繁换页，也会降低系统性能。对于内存在`4GB`左右的服务器该参数可设置为`256M`或`384M`。
- table_cache
  - 表示同时打开的表的个数。这个值越大，能够同时打开的表的个数越多。物理内存越大，设置就越大。默认为2402，调到512-1024最佳。这个值不是越大越好，因为同时打开的表太多会影响操作系统的性能。
- query_cache_size
  - 表示查询缓冲区的大小。可以通过在MySQL控制台观察
    - 如果Qcache_lowmem_prunes的值非常大，则表明经常出现缓冲不够的情况，就要增加Query_cache_size的值。
    - 如果Qcache_hits的值非常大，则表明查询缓冲使用非常频繁，如果该值较小反而会影响效率，那么可以考虑不用查询缓存。
    - 如果Qcache_free_blocks的值非常大，则表明缓冲区中碎片很多。
  - **MySQL8.0之后失效**。该参数需要和query_cache_type配合使用。
- query_cache_type
  - 值是0时，所有的查询都不使用查询缓存区。但是query_cache_type=0并不会导致MySQL释放query_cache_size所配置的缓存区内存。
  - 值为1时，所有的查询都将使用查询缓存区，除非在查询语句中指定`SQL_NO_CACHE`，SELECT SQL_NO_CACHE* FROM tbl_name。
  - 值为2时，只有在查询语句中使用`SQL_CACHE`关键字，查询才会使用查询缓存区。使用查询缓存区可以提高查询的速度，这种方式**只适用于修改操作少且经常执行相同的查询操作的情况。**
- sort_buffer_size
  - 表示每个需要进行排序的线程分配的缓冲区的大小。增加这个参数的值可以提高`ORDER BY`或`GROUP BY`操作的速度。默认数值是2097144字节(约2MB)。对于内存在4GB左右的服务器推荐设置为6-8M，如果有100个连接，那么实际分配的总共排序缓冲区大小为100 ×6 = 600MB。
- join_buffer_size = 8M
  - 表示联合查询操作所能使用的缓冲区大小，和sort_buffer_size一样，该参数对应的分配内存也是每个连接独享。

- read_buffer_size
  - 表示每个线程**连续扫描**时为扫描的每个表分配的缓冲区的大小(字节）。当线程从表中连续读取记录时需要用到这个缓冲区。SET SESSION read_buffer_size=n可以临时设置该参数的值。默认为64K,可以设置为4M。
- innodb_flush_log_at_trx_commit
  - 表示何时将缓冲区的数据写入日志文件，并且将日志文件写入磁盘中。该参数对于innoDB引擎非常重要。该参数的默认值为1。
  - 值为0时，表示`每秒1次`的频率将数据写入日志文件并将日志文件写入磁盘。每个事务的commit并不会触发前面的任何操作。该模式速度最快，但不太安全，mysqld进程的崩溃会导致上一秒钟所有事务数据的丢失。
  - 值为1时，表示`每次提交事务时`将数据写入日志文件并将日志文件写入磁盘进行同步。该模式是最安全的，但也是最慢的一种方式。因为每次事务提交或事务外的指令都需要把日志写入(flush）硬盘。
  - 值为2时，表示`每次提交事务时`将数据写入日志文件，每隔1秒将日志文件写入磁盘。该模式速度较快，也比0安全，只有在操作系统崩溃或者系统断电的情况下，上一秒钟所有事务数据才可能丢失。
- innodb_log_buffer_size
  - InnoDB存储引擎的`事务日志所使用的缓冲区`。为了提高性能，也是先将信息写入Innodb Log Buffer中，当满足innodb_flush_log_trx_commit参数所设置的相应条件(或者日志缓冲区写满)之后，才会将日志写到文件（或者同步到磁盘)中。
- max_connections
  - 表示`允许连接到MySQL数据库的最大数量`，默认值是`151`。如果状态变量 connection_errors_max_connections 不为零，并且一直增长，则说明不断有连接请求因数据库连接数已达到允许最大值而失败，这是可以考虑增大max_connections的值。在Linux平台下，性能好的服务器，支持500-1000个连接不是难事，需要根据服务器性能进行评估设定。这个连接数`不是越大越好`，因为这些连接会浪费内存的资源。过多的连接可能会导致MySQL服务器僵死。
- back_log
  - 用于`控制MySQL监听TCP端口时设置的积压请求栈大小`。如果MySql的连接数达到max_connections时，新来的请求将会被存在堆栈中，以等待某一连接释放资源，该堆栈的数量即back_log，如果等待连接的数量超过back_log，将不被授予连接资源，将会报错。5.6.6版本之前默认值为50，之后的版本默认为50+(max_connections / 5)，对于Linux系统推荐设置为小于512的整数，但最大不超过900。如果需要数据库在较短的时间内处理大量连接请求，可以考虑适当增大back_log的值。
- thread_cache_size
  - 线程池缓存线程数量的大小，当客户端断开连接后将当前线程缓存起来，当在接到新的连接请求时快速响应无需创建新的线程。这尤其对那些使用短连接的应用程序来说可以极大的提高创建连接的效率。那么为了提高性能可以增大该参数的值。默认为60，可以设置为120。
  - 可以通过几个MySQL状态值来适当调整线程池的大小：Threads_cached，Threads_connected，Threads_created，Threads_running。
  - 当Threads_cached越来越少，但Threads_connected 始终不降，且Threads_created持续升高，可适当增加thread_cache_size的大小。
- wait_timeout
  - 指定`一个请求的最大连接时间`，对于4GB左右内存的服务器可以设置为5-10。

- interactive_timeout
    - 表示服务器在关闭连接前等待行动的秒数。

> 这几个参数的设置一方面影响数据库效率，一方面影响系统效率，不是越大越好。

给出一份my.cnf的参考配置:

```
[mysqld]port = 3306serverid = 1
socket = /tmp/mysql.sockskip-locking

#避免MySQL的外部锁定，减少出错几率增强稳定性。
skip-name-resolve
#禁止MySQL对外部连接进行DNS解析，使用这一选项可以消除MySQL进行DNS解析的时间。但需要注意，如果开启该选项，则所有远程主机连接授权都要使用IP地址方式，否则MySQL将无法正常处理连接请求!
back_log = 384
key_buffer_size = 256M
max_allowed_packet = 4M
thread_stack = 256K
table_cache = 128K
sort_buffer_size = 6M
read_buffer_size =4M
read_rnd_buffer_size=16M
join_buffer_size = 8M
myisam_sort_buffer_size = 64M
table_cache = 512
thread_cache_size = 64
query_cache_size = 64M
tmp_table_size = 256M
max_connections = 768
max_connecterrors = 10000000
wait_timeout= 10
thread_concurrency = 8
#该参数取值为服务器逻辑CPU数量*2，在本例中，服务器有2颗物理CPU，而每颗物理CPU又支持H.T超线程，所以实际取值为4*2=8
skip-networking
#开启该选项可以彻底关闭MySQL的TCP/IP连接方式，如果WEB服务器是以远程连接的方式访问MySQL数据库服务器则不要开启该选项!否则将无法正常连接!
table_cache=1024
innodb_additional_mem_pool_size=4M
#默认为2M
innodb_flush_log_at_trx_commit=1
innodb_log_buffer_size=2M
#默认为1M
innodb_thread_concurrency=8
#你的服务器CPU有几个就设置为几。建议用默认一般为8
tmp_table_size=64M
#默认为16M，调到64-256最挂
thread_cache_size=120
query_cache_size=32M
```

2.3 举例

出现像双十一的短时间内高访问时的情况需要做什么处理?

为了解决这个问题，一共调整3个系统参数，分别是

- InnoDB_flush_log_at_trx_commit
  - 默认为1，每次提交事务的时候，都把数据写入日志，并把日志写入磁盘。
  - 但是虽然安全，效率却很低。可修改为0或2，修改为2时又能保证出现故障时损失数据较小。
- lnnoDB_buffer_pool_size
  - InnoDB存储引擎使用`缓存来存储索引和数据`。如果使用的MySQL服务器是数据库专属服务器，可以增加该参数，使得磁盘读写次数大幅降低。
- InnoDB_buffer_pool_instances
  - 这个参数可以将InnoDB的缓存区分成几个部分，这样可以提高系统的`并行处理能力`，因为可以允许多个进程同时处理不同部分的缓存区。
  - 我们把InnoDB_buffer_pool_instances 的值修改为64，意思就是把 InnoDB的缓存区分战64个分区，这样就可以同时有`多个进程`进行数据操作，CPU的效率就高多了。修改好了系统参数的值，要重启MysQL数据库服务器。

> 总结一下就是遇到CPU资源不足的问题，可以从下面2个思路去解决。
>
> - 疏通拥堵路段，消除瓶颈，让等待的时间更短;
> - 开拓新的通道，增加并行处理能力。

3. 优化数据库结构

3.1 拆分表：冷热数据分离

3.2 增加中间表

3.3 增加冗余字段

3.4 优化数据类型

3.4.1 情况1：对整数类型数据进行优化

数据量不够大的时候，直接用INT即可。如果数据量过大，对非负型数据来说，要优先使用无符号整型`UNSIGNED`来存储。

3.4.2 情况2：既可以使用文本类型也可以使用整数类型的字段，要选择使用整数类型

跟文本类型数据相比，大整数往往占用更少的存储空间，因此，在存取和比对的时候，可以占用更少的内存空间。所以，在二者皆可用的情况下，尽量使用整数类型，这样可以提高查询的效率。如：将IP地址转换成整型数据。

3.4.3 情况3：避免使用TEXT、BLOB数据类型

3.4.4 情况4：避免使用ENUM类型

3.4.5 情况5：使用TIMESTAMP存储时间

3.4.6 情况6：用DECIMAL代替FLOAT和DOUBLE存储精确浮点数

总之，遇到数据量大的项目时，一定要在充分了解业务需求的前提下，合理优化数据类型，这样才能充分发挥资源的效率，使系统达到最优。

3.5 优化插入记录的速度

3.5.1 MyISAM引擎的表

① 禁用索引

② 禁用唯一性检查

③ 使用批量插入

④ 使用LOAD DATA INFILE 批量导入

3.5.2 InnoDB引擎的表

① 禁用唯一性检查

② 禁用外键检查

③ 禁止自动提交

3.6 使用非空约束

在设计字段的时候，如果业务允许，建议尽量使用非空约束

3.7 分析表、检查表与优化表

3.7.1 分析表

```
ANALYZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name[,tbl_name]…
```

默认的，MySQL服务会将 ANALYZE TABLE语句写到binlog中，以便在主从架构中，从服务能够同步数据。可以添加参数LOCAL 或者 NO_WRITE_TO_BINLOG取消将语句写到binlog中。

使用`ANALYZE TABLE`分析表的过程中，数据库系统会自动对表加一个`只读锁`。在分析期间，只能读取表中的记录，不能更新和插入记录。ANALYZE TABLE语句能够分析InnoDB和MyISAM类型的表，但是不能作用于视图。

ANALYZE TABLE分析后的统计结果会反应到`cardinality`的值，该值统计了表中某一键所在的列不重复的值的个数。**该值越接近表中的总行数，则在表连接查询或者索引查询时，就越优先被优化器选择使用。**

3.7.2 检查表

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ... option = {QUICK | FAST | MEDIUM | EXTENDED | CHANGED}
```

MySQL中可以使用`CHECK TABLE`语句来检查表。CHECK TABLE语句能够检查InnoDB和MyISAM类型的表是否存在错误。CHECK TABLE语句在执行过程中也会给表加上`只读锁`。

3.7.3 优化表

```
OPTIMIZE [LOCAL | NO_WRITE_TO_BINLOG] TABLE tbl_name [, tbl_name] ...
```

MySQL中使用`OPTIMIZE TABLE`语句来优化表。但是，OPTILMIZE TABLE语句只能优化表中的`VARCHAR`、`BLOB`或`TEXT`类型的字段。一个表使用了这些字段的数据类型，若已经`删除`了表的一大部分数据，或者已经对含有可变长度行的表（含有VARCHAR、BLOB或TEXT列的表）进行了很多`更新`，则应使用OPTIMIZE TABLE来重新利用未使用的空间，并整理数据文件的`碎片`。

OPTIMIZE TABLE 语句对InnoDB和MyISAM类型的表都有效。该语句在执行过程中也会给表加上`只读锁`。