第12章_数据库其它调优策略

讲师：尚硅谷-宋红康（江湖人称：康师傅）

官网： http://www.atguigu.com

---

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

如何确定呢？一般情况下，有如下几种方式：

- **用户的反馈（主要）**
- **日志分析（主要）**
- **服务器资源使用监控**
- **数据库内部状况监控**
- **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。

1.4 调优的维度和步骤

我们需要调优的对象是整个数据库管理系统，它不仅包括 SQL 查询，还包括数据库的部署配置、架构等。从这个角度来说，我们思考的维度就不仅仅局限在 SQL 优化上了。通过如下的步骤我们进行梳理：

第1步：选择适合的 DBMS

第2步：优化表设计

第3步：优化逻辑查询

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。

第5步：使用 Redis 或 Memcached 作为缓存

除了可以对 SQL 本身进行优化以外，我们还可以请外援提升查询的效率。

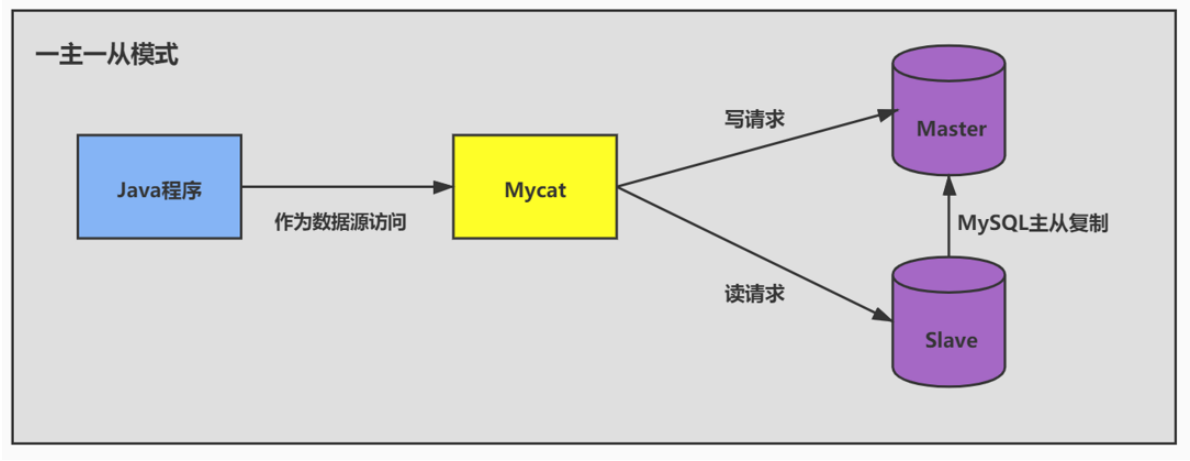
因为数据都是存放到数据库中，我们需要从数据库层中取出数据放到内存中进行业务逻辑的操作，当用户量增大的时候，如果频繁地进行数据查询，会消耗数据库的很多资源。如果我们将常用的数据直接放到内存中，就会大幅提升查询的效率。

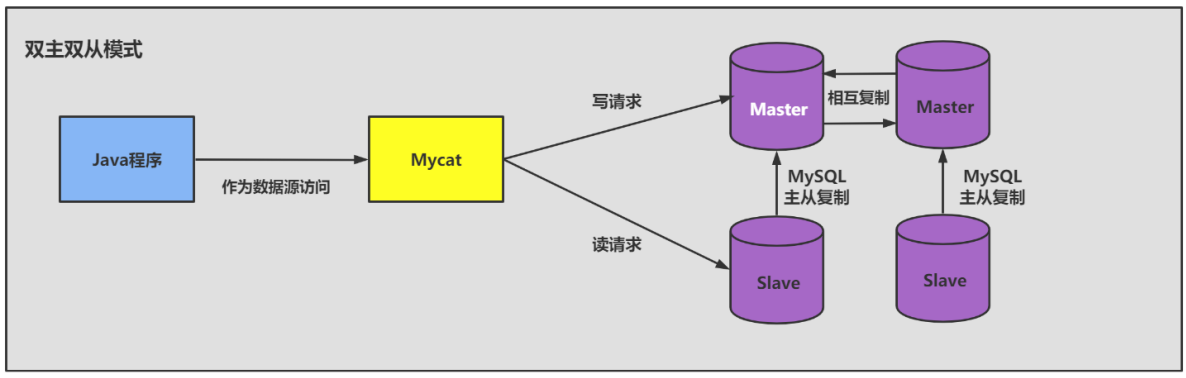
键值存储数据库可以帮我们解决这个问题。

常用的键值存储数据库有 Redis 和 Memcached，它们都可以将数据存放到内存中。

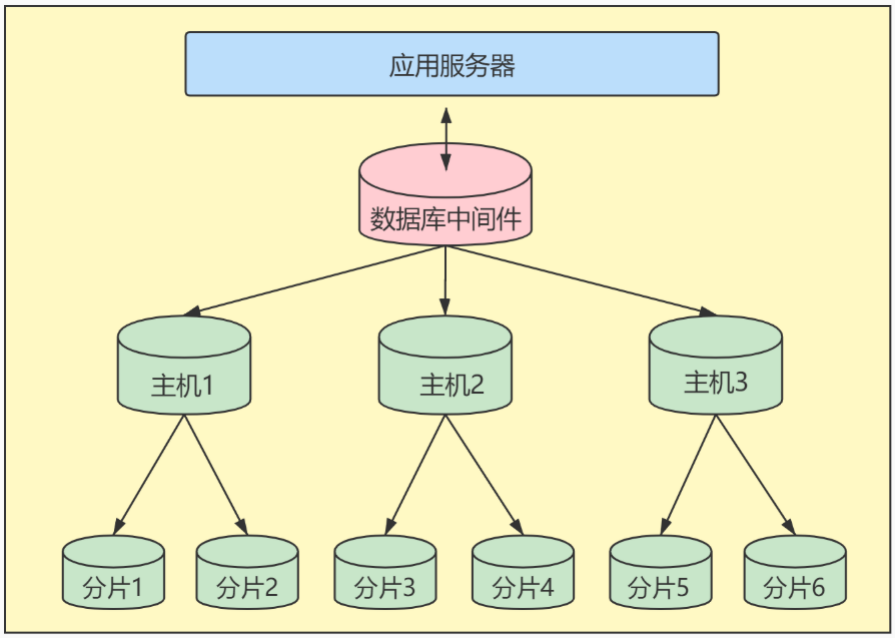
第6步：库级优化

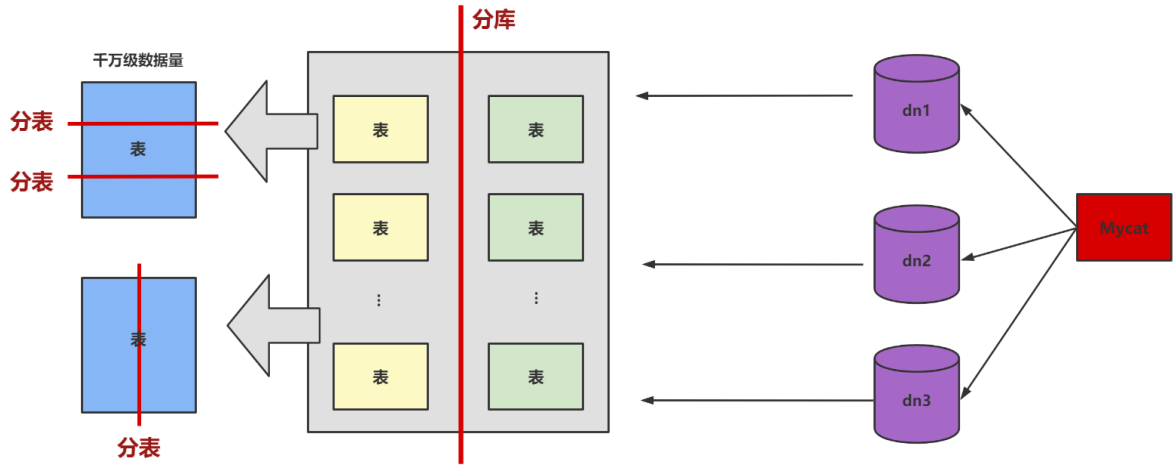
1、读写分离





2、数据分片





> 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

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。 硬件的性能瓶颈直接决定MySQL数据库的运行速度和效率。针对性能瓶颈提高硬件配置，可以提高MySQL数据库查询、更新的速度。 （1） 配置较大的内存 （2） 配置高速磁盘系统 （3） 合理分布磁盘I/O （4） 配置多处理器

2.2 优化MySQL的参数

- `innodb_buffer_pool_size`：这个参数是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`：表示索引缓冲区的大小。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；Qcache_free_blocks，如果该值非常大，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`的值是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。
    - 当query_cache_type=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`，如SELECT SQL_NO_CACHE * FROM tbl_name。
    - 当query_cache_type=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2 097 144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。该参数有3个值，分别为0、1和2。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`：这是 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。

  可以通过如下几个MySQL状态值来适当调整线程池的大小：

  ```
  mysql> show global status like 'Thread%';
  +-------------------+-------+
  | Variable_name     | Value |
  +-------------------+-------+
  | Threads_cached    | 2     |
  | Threads_connected | 1     |
  | Threads_created   | 3     |
  | Threads_running   | 2     |
  +-------------------+-------+
  4 rows in set (0.01 sec)
  ```

  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。

这里给出一份my.cnf的参考配置：

> [mysqld]
>
> port = 3306 serverid = 1 socket = /tmp/mysql.sock `skip-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_connect_errors = 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`

很多情况还需要具体情况具体分析！

案例分析：见视频

（https://www.bilibili.com/video/BV1iq4y1u7vj?from=search&seid=4297501441472622157&spm_id_from=333.337.0.0）

3. 优化数据库结构

3.1 拆分表：冷热数据分离

举例1： 会员members表存储会员登录认证信息，该表中有很多字段，如id、姓名、密码、地址、电话、个人描述字段。其中地址、电话、个人描述等字段并不常用，可以将这些不常用的字段分解出另一个表。将这个表取名叫members_detail，表中有member_id、address、telephone、description等字段。这样就把会员表分成了两个表，分别为 members表 和 members_detail表 。

创建这两个表的SQL语句如下：

```
CREATE TABLE members (
    id int(11) NOT NULL AUTO_INCREMENT,
    username varchar(50) DEFAULT NULL,
    password varchar(50) DEFAULT NULL,
    last_login_time datetime DEFAULT NULL,
    last_login_ip varchar(100) DEFAULT NULL,
    PRIMARY KEY(Id)
);
CREATE TABLE members_detail (
    Member_id int(11) NOT NULL DEFAULT 0,
    address varchar(255) DEFAULT NULL,
    telephone varchar(255) DEFAULT NULL,
    description text
);
```

如果需要查询会员的基本信息或详细信息，那么可以用会员的id来查询。如果需要将会员的基本信息和详细信息同时显示，那么可以将members表和members_detail表进行联合查询，查询语句如下：

```
SELECT * FROM members LEFT JOIN members_detail on members.id =
members_detail.member_id;
```

通过这种分解可以提高表的查询效率。对于字段很多且有些字段使用不频繁的表，可以通过这种分解的方式来优化数据库的性能。

3.2 增加中间表

举例1： 学生信息表 和 班级表 的SQL语句如下：

```
CREATE TABLE `class` (
 `id` INT(11) NOT NULL AUTO_INCREMENT,
 `className` VARCHAR(30) DEFAULT NULL,
 `address` VARCHAR(40) DEFAULT NULL,
 `monitor` INT NULL ,
 PRIMARY KEY (`id`)
) ENGINE=INNODB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;

CREATE TABLE `student` (
 `id` INT(11) NOT NULL AUTO_INCREMENT,
 `stuno` INT NOT NULL ,
```

```
  `name` VARCHAR(20) DEFAULT NULL,
  `age` INT(3) DEFAULT NULL,
  `classId` INT(11) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=INNODB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

现在有一个模块需要经常查询带有学生名称（name）、学生所在班级名称（className）、学生班级班长（monitor）的学生信息。根据这种情况可以创建一个`temp_student`表。temp_student表中存储学生名称（stu_name）、学生所在班级名称（className）和学生班级班长（monitor）信息。创建表的语句如下：

```
CREATE TABLE `temp_student` (
  `id` INT(11) NOT NULL AUTO_INCREMENT,
  `stu_name` INT NOT NULL ,
  `className` VARCHAR(20) DEFAULT NULL,
  `monitor` INT(3) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=INNODB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
```

接下来，从学生信息表和班级表中查询相关信息存储到临时表中：

```
insert into temp_student(stu_name,className,monitor)
            select s.name,c.className,c.monitor
            from student as s,class as c
            where s.classId = c.id
```

以后，可以直接从temp_student表中查询学生名称、班级名称和班级班长，而不用每次都进行联合查询。这样可以提高数据库的查询速度。

3.3 增加冗余字段

设计数据库表时应尽量遵循范式理论的规约，尽可能减少冗余字段，让数据库设计看起来精致、优雅。但是，合理地加入冗余字段可以提高查询速度。

表的规范化程度越高，表与表之间的关系就越多，需要连接查询的情况也就越多。尤其在数据量大，而且需要频繁进行连接的时候，为了提升效率，我们也可以考虑增加冗余字段来减少连接。

这部分内容在《第11章_数据库的设计规范》章节中`反范式化小节`中具体展开讲解了。这里省略。

3.4 优化数据类型

情况1：对整数类型数据进行优化。

遇到整数类型的字段可以用`INT 型`。这样做的理由是，INT 型数据有足够大的取值范围，不用担心数据超出取值范围的问题。刚开始做项目的时候，首先要保证系统的稳定性，这样设计字段类型是可以的。但在数据量很大的时候，数据类型的定义，在很大程度上会影响到系统整体的执行效率。

对于`非负型`的数据（如自增ID、整型IP）来说，要优先使用无符号整型`UNSIGNED`来存储。因为无符号相对于有符号，同样的字节数，存储的数值范围更大。如tinyint有符号为-128-127，无符号为0-255，多出一倍的存储空间。

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。

情况3：避免使用TEXT、BLOB数据类型

情况4：避免使用ENUM类型

情况5：使用TIMESTAMP存储时间

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

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

```
insert into student values(1,'zhangsan',18,1);
insert into student values(2,'lisi',17,1);
insert into student values(3,'wangwu',17,1);
insert into student values(4,'zhaoliu',19,1);
```

使用一条INSERT语句插入多条记录的情形如下：

```
insert into student values
(1,'zhangsan',18,1),
(2,'lisi',17,1),
(3,'wangwu',17,1),
(4,'zhaoliu',19,1);
```

第2种情形的插入速度要比第1种情形快。

④ 使用LOAD DATA INFILE 批量导入

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

1. 分析表

MySQL中提供了ANALYZE TABLE语句分析表，ANALYZE TABLE语句的基本语法如下：

```
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。**也就是索引列的cardinality的值与表中数据的总条数差距越大，即使查询的时候使用了该索引作为查询条件，存储引擎实际查询的时候使用的概率就越小。下面通过例子来验证下。cardinality可以通过SHOW INDEX FROM 表名查看。

2. 检查表

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`。

对于MyISAM类型的表，CHECK TABLE语句还会更新关键字统计数据。而且，CHECK TABLE也可以检查视图是否有错误，比如在视图定义中被引用的表已不存在。该语句的基本语法如下：

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

其中，tbl_name是表名；option参数有5个取值，分别是QUICK、FAST、MEDIUM、EXTENDED和CHANGED。各个选项的意义分别是：

- `QUICK`：不扫描行，不检查错误的连接。
- `FAST`：只检查没有被正确关闭的表。
- `CHANGED`：只检查上次检查后被更改的表和没有被正确关闭的表。
- `MEDIUM`：扫描行，以验证被删除的连接是有效的。也可以计算各行的关键字校验和，并使用计算出的校验和验证这一点。
- `EXTENDED`：对每行的所有关键字进行一个全面的关键字查找。这可以确保表是100%一致的，但是花的时间较长。

option只对MyISAM类型的表有效，对InnoDB类型的表无效。比如：

```
mysql> check table student;
+--------------------+-------+----------+----------+
| Table              | Op    | Msg_type | Msg_text |
+--------------------+-------+----------+----------+
| atguigudb1.student | check | status   | OK       |
+--------------------+-------+----------+----------+
1 row in set (1.84 sec)
```

该语句对于检查的表可能会产生多行信息。最后一行有一个状态的 Msg_type 值，Msg_text 通常为 OK。如果得到的不是 OK，通常要对其进行修复；是 OK 说明表已经是最新的了。表已经是最新的，意味着存储引擎对这张表不必进行检查。

3. 优化表

方式1：OPTIMIZE TABLE

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`。

OPTILMIZE TABLE语句的基本语法如下：

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

LOCAL | NO_WRITE_TO_BINLOG关键字的意义和分析表相同，都是指定不写入二进制日志。

```
mysql> optimize table student;
+--------------------+----------+----------+-------------------------------------------------------------------+
| Table              | Op       | Msg_type | Msg_text                                                          |
+--------------------+----------+----------+-------------------------------------------------------------------+
| atguigudb1.student | optimize | note     | Table does not support optimize, doing recreate + analyze instead |
| atguigudb1.student | optimize | status   | OK                                                                |
+--------------------+----------+----------+-------------------------------------------------------------------+
2 rows in set (14.44 sec)
```

执行完毕，Msg_text显示

> ‘numysql.SYS_APP_USER’, ‘optimize’, ‘note’, ‘Table does not support optimize, doing recreate + analyze instead’

原因是我服务器上的MySQL是InnoDB存储引擎。

到底优化了没有呢？看官网！

https://dev.mysql.com/doc/refman/8.0/en/optimize-table.html

在MyISAM中，是先分析这张表，然后会整理相关的MySQL datafile，之后回收未使用的空间；在InnoDB中，回收空间是简单通过Alter table进行整理空间。在优化期间，MySQL会创建一个临时表，优化完成之后会删除原始表，然后会将临时表rename成为原始表。

> 说明： 在多数的设置中，根本不需要运行OPTIMIZE TABLE。即使对可变长度的行进行了大量的更新，也不需要经常运行， 每周一次 或 每月一次 即可，并且只需要对 特定的表 运行。

3.8 小结

上述这些方法都是有利有弊的。比如：

- 修改数据类型，节省存储空间的同时，你要考虑到数据不能超过取值范围；
- 增加冗余字段的时候，不要忘了确保数据一致性；
- 把大表拆分，也意味着你的查询会增加新的连接，从而增加额外的开销和运维的成本。

因此，你一定要结合实际的业务需求进行权衡。

4. 大表优化

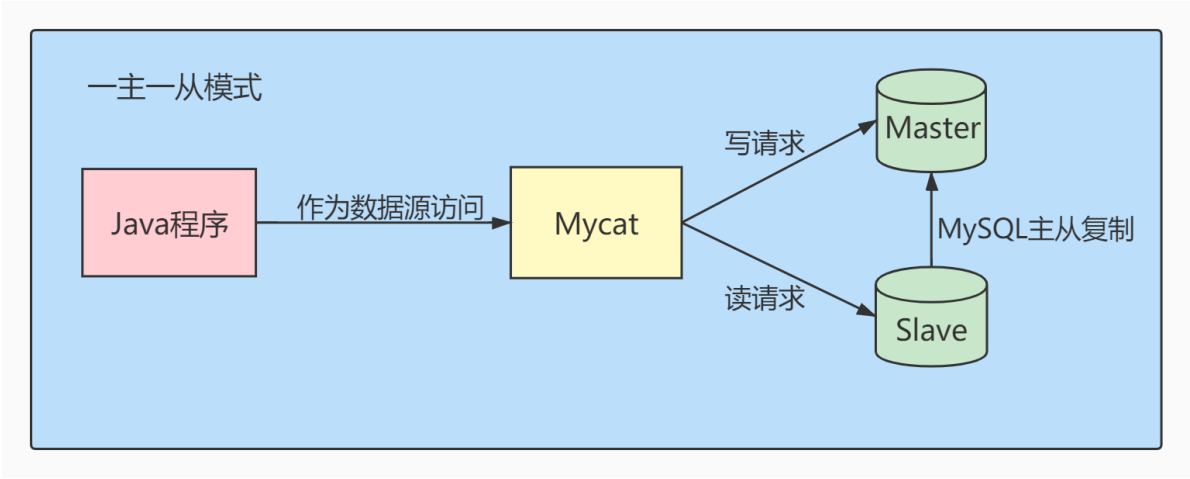
4.1 限定查询的范围

禁止不带任何限制数据范围条件的查询语句。比如：我们当用户在查询订单历史的时候，我们可以控制在一个月的范围内；

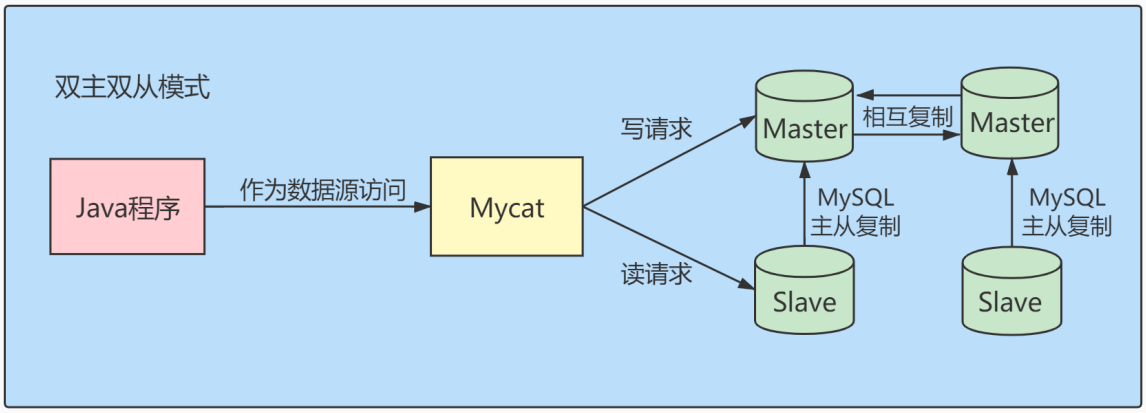
4.2 读/写分离

经典的数据库拆分方案，主库负责写，从库负责读。

- 一主一从模式：

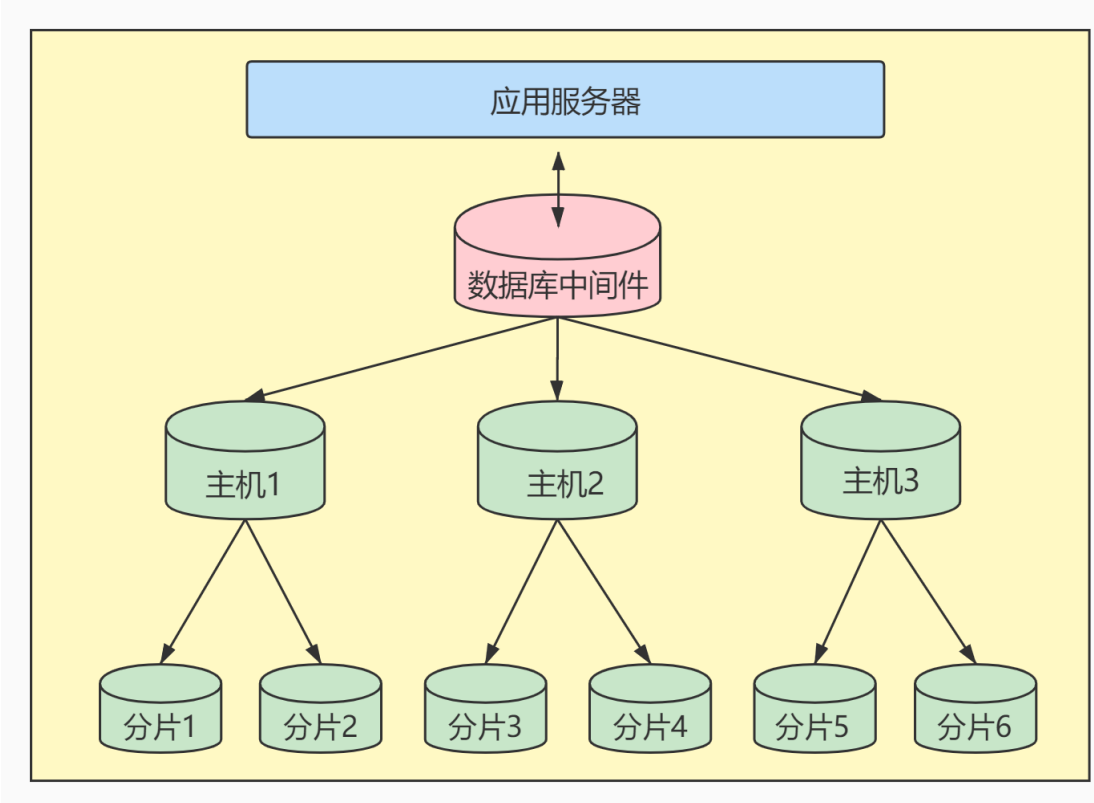


- 双主双从模式：



4.3 垂直拆分

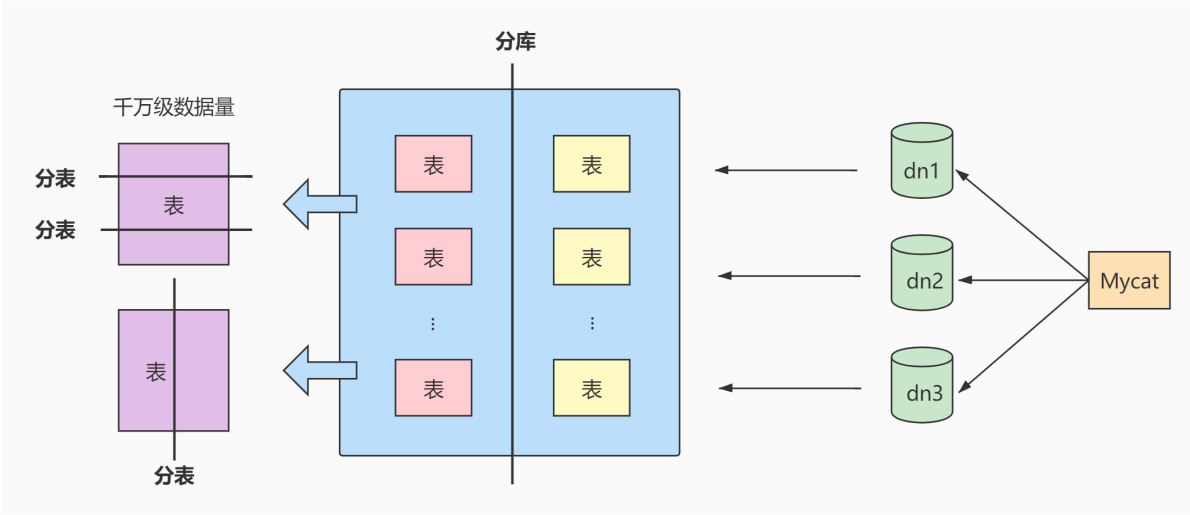
当数据量级达到 千万级 以上时，有时候我们需要把一个数据库切成多份，放到不同的数据库服务器上，减少对单一数据库服务器的访问压力。



垂直拆分的优点： 可以使得列数据变小，在查询时减少读取的Block数，减少I/O次数。此外，垂直分区可以简化表的结构，易于维护。

垂直拆分的缺点： 主键会出现冗余，需要管理冗余列，并会引起 JOIN 操作。此外，垂直拆分会让事务变得更加复杂。

4.4 水平拆分



下面补充一下数据库分片的两种常见方案：

- **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。** 当当网的 Sharding-JDBC 、阿里的TDDL是两种比较常用的实现。
- **中间件代理： 在应用和数据中间加了一个代理层。分片逻辑统一维护在中间件服务中。** 我们现在谈的 Mycat 、360的Atlas、网易的DDB等等都是这种架构的实现。

5. 其它调优策略

5.1 服务器语句超时处理

在MySQL 8.0中可以设置 `服务器语句超时的限制` ，单位可以达到 `毫秒级别` 。当中断的执行语句超过设置的毫秒数后，服务器将终止查询影响不大的事务或连接，然后将错误报给客户端。

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。默认情况下，MAX_EXECUTION_TIME的值为0，代表没有时间限制。 例如：

```
SET GLOBAL MAX_EXECUTION_TIME=2000;
```

```
SET SESSION MAX_EXECUTION_TIME=2000; #指定该会话中SELECT语句的超时时间
```

5.2 创建全局通用表空间

5.3 MySQL 8.0新特性：隐藏索引对调优的帮助