MySQL 优化技术

开发的路上，总会碰到一些**老系统，越用越慢**。“慢”的原因也许有很多，但是，博主个人觉得，数据库的设计和 sql 语句写的好坏，对系统效率的影响是最直接，最显而易见的！所以，学习一下 MySQL 的优化，还是很有必要的。当然，博主能力有限，没那么多经验，更多的是“道听途说”和“纸上谈兵”。如有不正之处，望大神开后给予指正，不胜感激！

（一）MySQL 优化技术概述

①**表的设计合理化**(符合 3NF，即符合“**三范式**”。当然也要照顾“**反范式**”，要灵活)；

②**添加适当索引(index)**(主要包括：**普通索引**、**主键索引**、**唯一索引 unique**、**全文索引**)；

③**SQL 语句本身的优化**(主要包括：**避免全表扫描**、**避免嵌套子查询**等)；

④**分表技术**(**水平分割**、**垂直分割**)；

⑤**读写分离**（其中写包括：update/delete/add）；

⑥**存储过程** （**模块化编程**，可以提高速度，但是迁移性差，对服务器压力也会逐渐增大)；

⑦**对 mysql 配置优化** （主要是修改 my.ini 配置文件的参数信息）

⑧**mysql 服务器硬件升级**

⑨**定时的去清除不需要的数据,定时进行碎片整理**(特别是使用了 MyISAM)

（二）表的设计合理化

1.“三范式”（3NF）的概念

第一范式：1NF 是对属性的**原子性约束**，要求属性(列)具有原子性，不可再分解；(**只要是关系型数据库都满足 1NF**)；

第二范式：2NF 是在 1NF 满足的基础上，对记录的**惟一性约束**，要求记录有惟一标识，即实体的惟一性；

第三范式：3NF 是在 2NF 满足的基础上，对**字段冗余性的约束**，要求字段没有冗余，也可以称为**消除依赖传递**。

2.“反范式”（3NF）的概念

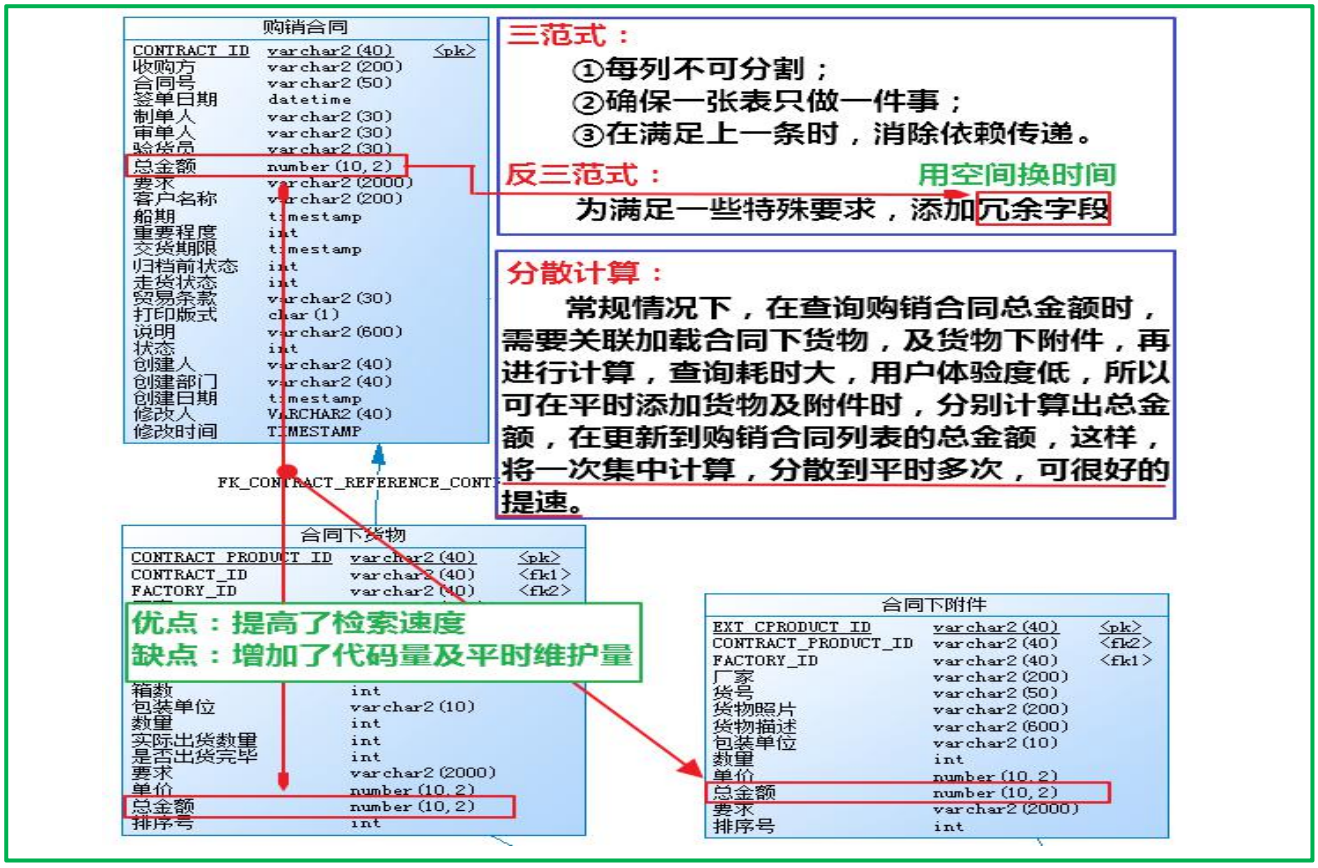
反范式：是通过**增加冗余数据**来提高数据库读性能的过程。

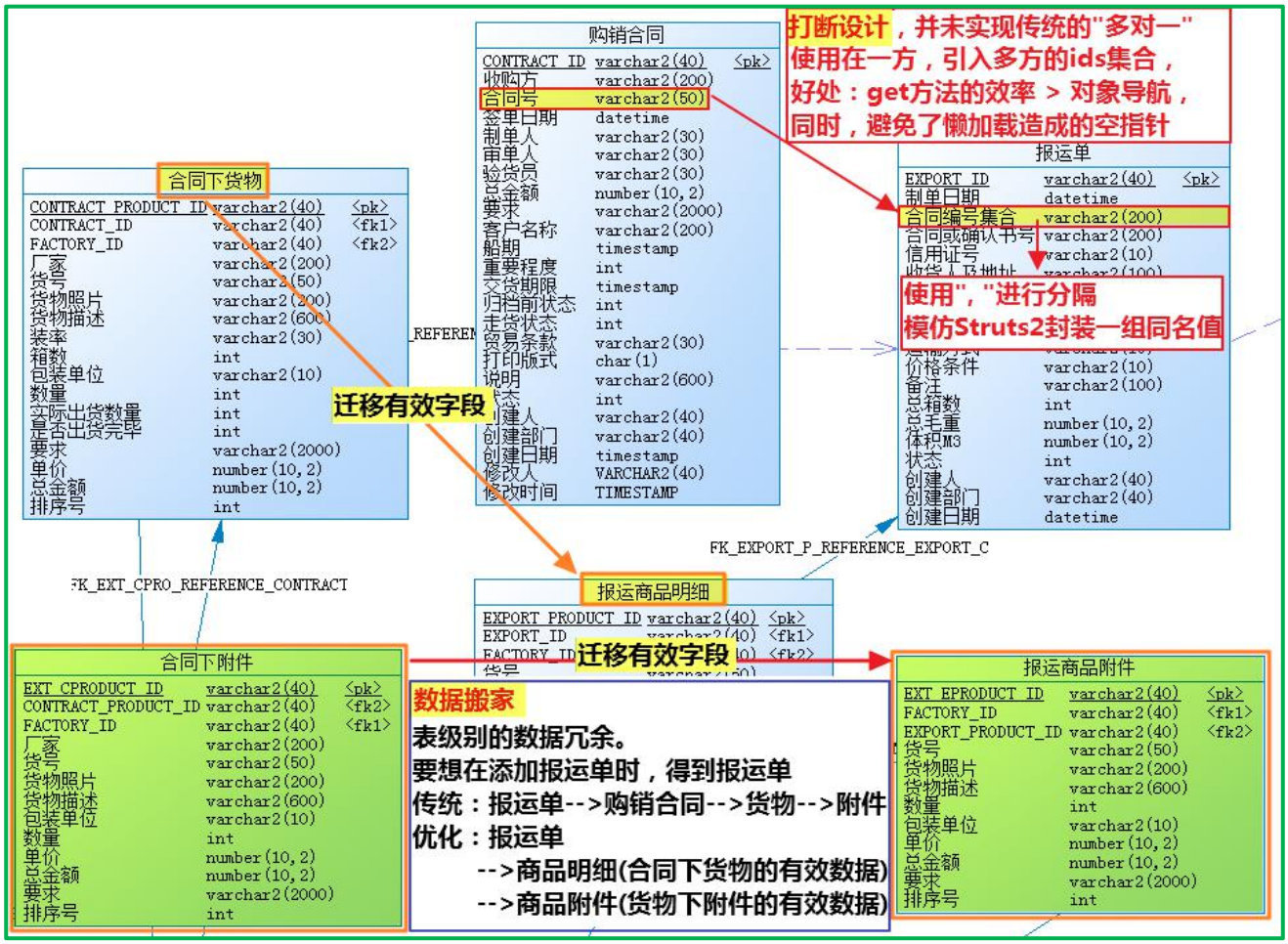
反范式出现的原因：当我们的业务所涉及的表非常多，经常会有多表联查，这样它的效率就会大打折扣，这时我们就可以考虑使用“反范式”。增加必要的，有效的冗余字段，**用空间来换取时间**,在查询时减少或者是避免过多表之间的联查。

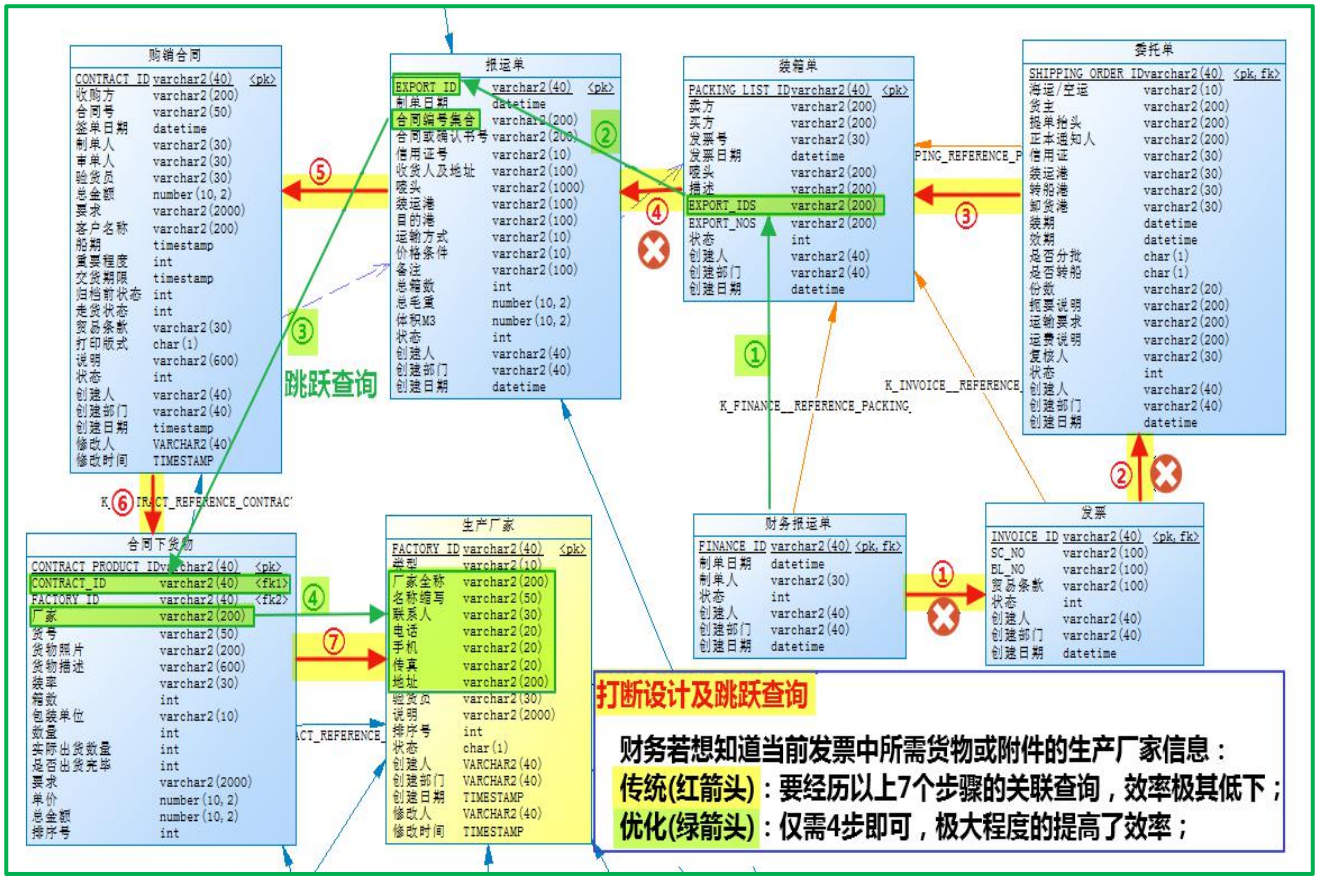
3.举例

以**传智博客视频教程（业内良心，某宝 2 元钱买能一堆）**中一个老项目为例，当时听得时候，觉得老师讲得神乎其神，列举其中的几个关键词，大家听听：**三范式与反三范式**、**分**

散计算思想、打断设计、添加冗余、跳跃查询、数据搬家等等。咋一听很能“蒙人”，后来博主画图总结了一下，其实，一说就破，很简单。







（三）Sql 语句自身的优化

1. SHOW [SESSION|GLOBAL] STATUS 指令的应用

```
#1. 累计启动多少秒
SHOW STATUS LIKE 'uptime'

#2. 累计查询/新增/修改/删除多少次
SHOW STATUS LIKE 'com_select'
SHOW STATUS LIKE 'com_insert'
SHOW STATUS LIKE 'com_update'
SHOW STATUS LIKE 'com_delete'
#注意：默认是 SESSION 当前会话级别的统计，全局的需要加上 GLOBAL 关键字
#默认的，当前会话级别的统计
SHOW SESSION STATUS LIKE 'com_select'
#全局的，统计从始至终
SHOW GLOBAL STATUS LIKE 'com_select'

#3. 显示慢查询次数
SHOW GLOBAL STATUS LIKE 'slow_queries'
#显示当前慢查询时间，默认 10 秒
SHOW VARIABLES LIKE 'long_query_time'
#修改慢查询时间
SET long_query_time = 5
```

2.如何记录及定位慢查询

在默认情况下，我们的 mysql 不会记录慢查询，需要在启动 mysql 时候，指定记录慢查询才可以：

```
bin\mysqld.exe - -safe-mode  - -slow-query-log
[mysql5.5 以上可以在 my.ini 指定]
```

先关闭 mysql,再启动，如果启用了慢查询日志，默认把这个文件放在 my.ini 文件中记录的位置：

```
#Path to the database root
datadir="C:/ProgramData/MySQL/MySQL Server 5.5/Data/"
```

查看慢查询日志：默认为数据目录 data 中的 host-name-slow.log。

（四）建立适当的索引

说起提高数据库性能，索引是最物美价廉的东西了。不用加内存，不用改程序，不用调 sql，只要执行个正确的‘create index’，查询速度就可能提高百倍千倍，这可真有诱惑力。可是天下没有免费的午餐，查询速度的提高是以插入、更新、删除的速度为代价的，这些写操作，增加了大量的 I/O。

索引主要分为：普通索引、主键索引、唯一索引、全文索引四大类，下面进行逐一说明：

1.索引添加

（1）主键索引添加

当一张表，把某个列设为主键的时候，则该列就是主键索引：

```
create table aaa
(id int unsigned primary key auto_increment ,name varchar(32) not null defaul
'');
```

这时 id 列就是主键索引。

如果你创建表时，没有指定主键索引，也可以在创建表后，再添加（不常用），指令：

```
alter table 表名 add primary key (列名)
```

（2）普通索引

一般来说，普通索引的创建，是先创建表，然后再创建普通索引，比如：

```
create table 表名(
    id int unsigned,
    name varchar(32)
)create index 索引名 on 表 (列1,列名2);
```

（3）创建全文索引

全文索引，主要是针对文件，文本的检索，比如文章，全文索引针对 MyISAM 有用。

创建 ：

```
CREATE TABLE articles (
```

```
      id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
      title VARCHAR(200),
      body TEXT,
      FULLTEXT (title,body)
    )engine=myisam charset utf8;
INSERT INTO articles (title,body) VALUES
    ('MySQL Tutorial','DBMS stands for DataBase ...'),
    ('How To Use MySQL Well','After you went through a ...'),
    ('Optimizing MySQL','In this tutorial we will show ...'),
    ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
    ('MySQL vs. YourSQL','In the following database comparison ...'),
    ('MySQL Security','When configured properly, MySQL ...');
```

如何使用全文索引：

```
错误用法：
select * from articles where body like '%mysql%'; 【不会使用到全文索引】
证明：
explain select * from articles where body like '%mysql%'
正确的用法是：
select * from articles where match(title,body) against('database');
```

*注意事项：

1. 在 mysql 中 fulltext 索引只针对 MyISAM 生效；

2. mysql 自己提供的 fulltext 针对英文生效->sphinx（coreseek）技术处理中文；

3. 使用方法是 match(字段名..) against('关键字')；

4. 全文索引一个叫停止词，因为在一个文本中，创建索引是一个无穷大的数，因此，对一些常用词和字符，就不会创建，这些词，称为停止词。

（4）唯一索引

①当表的某列被指定为 unique 约束时，这列就是一个唯一索引。

```
create table ddd(id int primary key auto_increment , name varchar(32)
unique);
```

这时，name 列就是一个唯一索引。

注意：unique 字段可以为 NULL，并可以有多 NULL，但是如果是具体内容，则不能重复。

主键字段，不能为 NULL，也不能重复。

②在创建表后，再去创建唯一索引。

```
create table eee(id int primary key auto_increment, name varchar(32));
create unique index 索引名 on 表名 (列表..);
```

2. 查询索引

```
desc 表名 【该方法的缺点是：不能够显示索引名.】
show index(es) from 表名\G
show keys from 表名\G
```

3. 删除

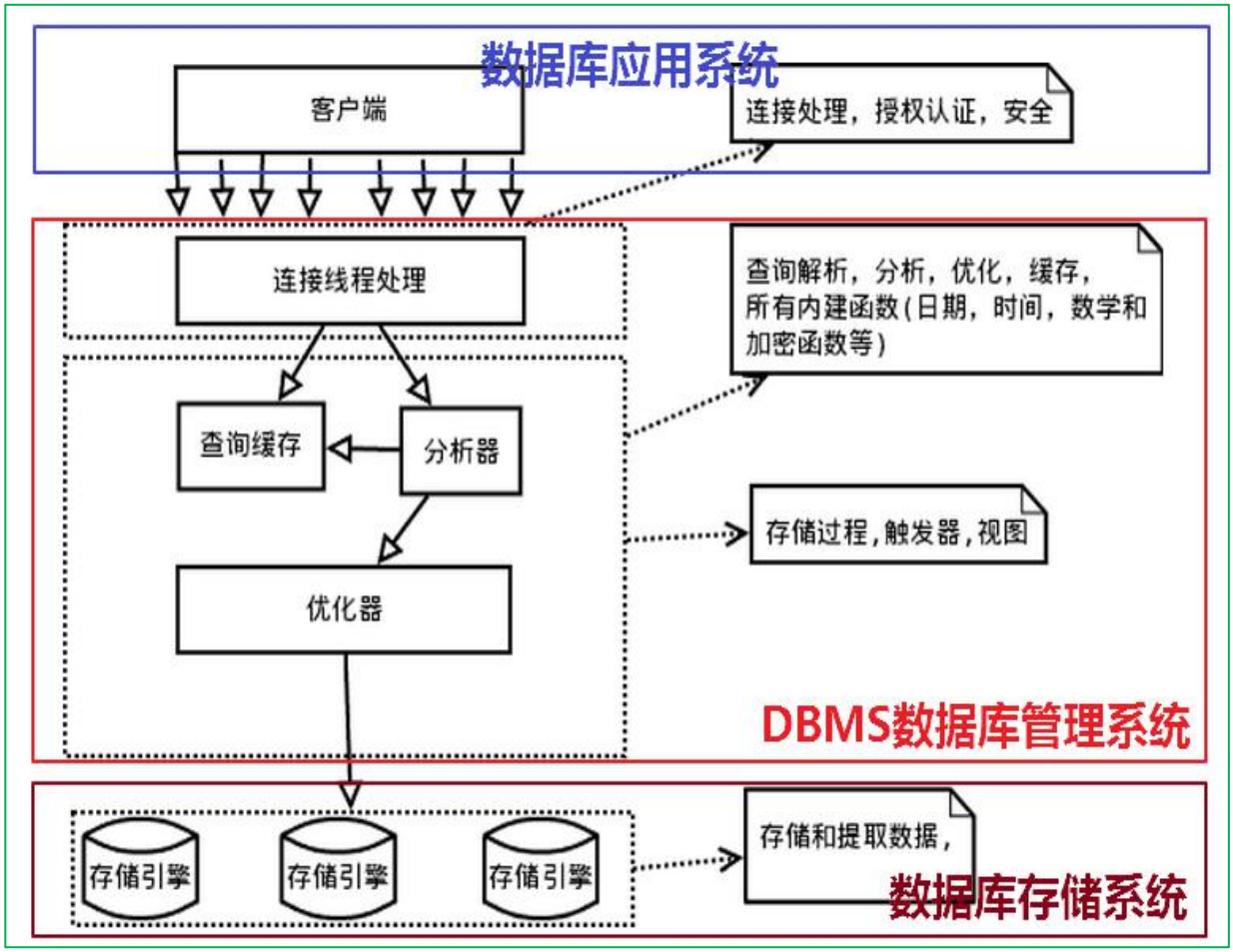
```
alter table 表名 drop index 索引名;
如果删除主键索引。
alter table 表名 drop primary key        [这里有一个小问题]
```

4. 修改

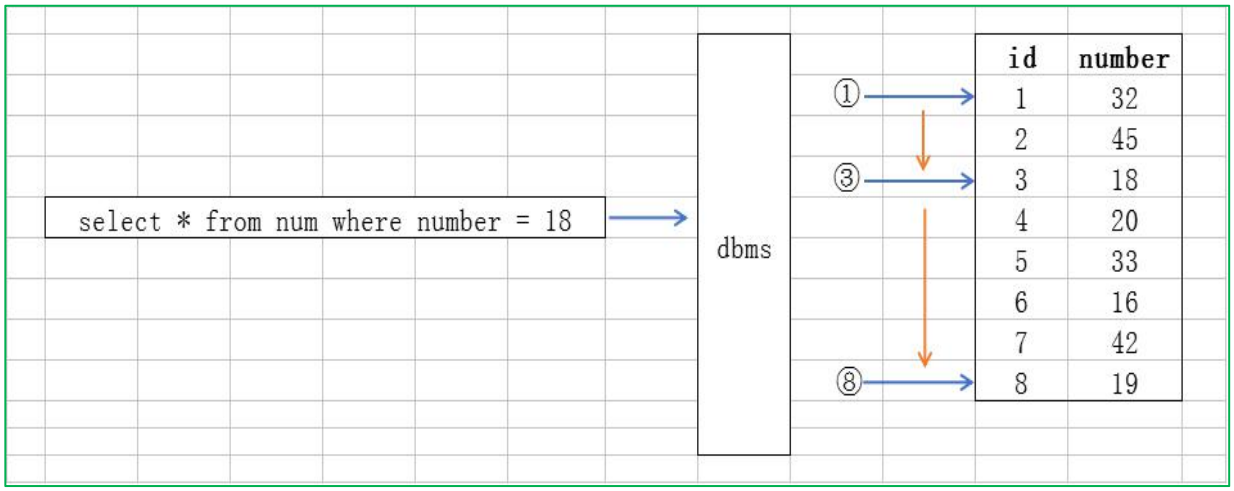
先删除，再重新创建.

5. 索引的原理

（1）数据库的三层结构简图



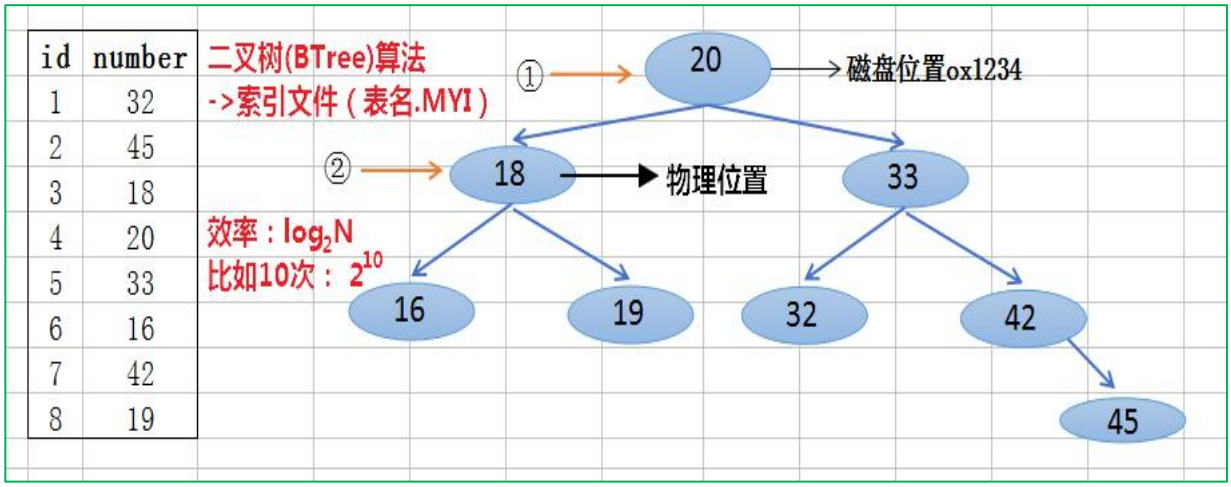
（2）原始的查询图解



原始的查询方法，查到 number = 3 后，仍然会往后查询，以为不确保后面是否有重复数

据，所以是全表检索。至少要查询 8 次。

（2）对 number 建立索引后的查询图解



建立索引后，数据库会将索引字段，进行基于二叉树（B Tree / B+ Tree）的形式改造，并存储在“表名.MYI”的文件中。其后，再次查询时，只需查询 2 次即可，而且树节点记录的是物理地址，可以直接定位到元素，这样它的效率就大大的提高了。

6. 索引的代价

①占用磁盘空间;

②对 DML 操作有影响，变慢;

7. 何时创建索引

①较频繁的作为查询条件字段应该创建索引

```
select * from emp where empno = 1
```

②唯一性太差的字段不适合单独创建索引，即使频繁作为查询条件

```
select * from emp where sex = ‘男’
```

③更新非常频繁的字段不适合创建索引

```
select * from emp where logincount = 1
```

④不会出现在 WHERE 子句中字段不该创建索

8. 使用索引时的注意事项

创建如下复合索引：

```
alter table dept add index my_ind (dname,loc);   # dname 左边的列,loc 就是右边的列
```

如果我们的表中有复合索引(索引作用在多列上)， 此时我们注意:

①对于创建的复合索引，只要查询条件使用了最左边的列，索引一般就会被使用。

```
explain select * from dept where loc= 'aaa'; #不会使用到索引
```

②对于使用 like 的查询，查询如果是‘%aaa’/‘_aaa’ 不会使用到索引，‘aaa%’/‘aaa_’ 会使用到索引。

```
explain select * from dept where dname like '%aaa'; #不会使用到索引
```

如果一定要前面有变化的值，则考虑使用全文索引->sphinx.

③如果条件中有 or，即使其中有条件带索引也不会使用。换言之，就是要求使用的所有

字段，都必须建立索引，我们建议大家尽量避免使用 or 关键字

```
select * from dept where dname=’xxx’ or loc=’xx’ or deptno=45; #不会使用到索引
```

④如果列类型是字符串，那一定要在条件中将数据使用引号引用起来。否则不使用索引。(添加时,字符串必须’’)，也就是，如果列是字符串类型，就一定要用 ‘’ 把他包括起来。

⑤如果 mysql 估计使用全表扫描要比使用索引快，则不使用索引。

8.explain 的应用

explain 可以帮助我们在不真正执行某个 sql 语句时，就执行 mysql 怎样执行，这样利用我们去分析 sql 指令。

如何查看索引使用的情况:

```
show status like ‘Handler_read%’;
```

注意：

handler_read_key：这个值越高越好，越高表示使用索引查询到的次数。

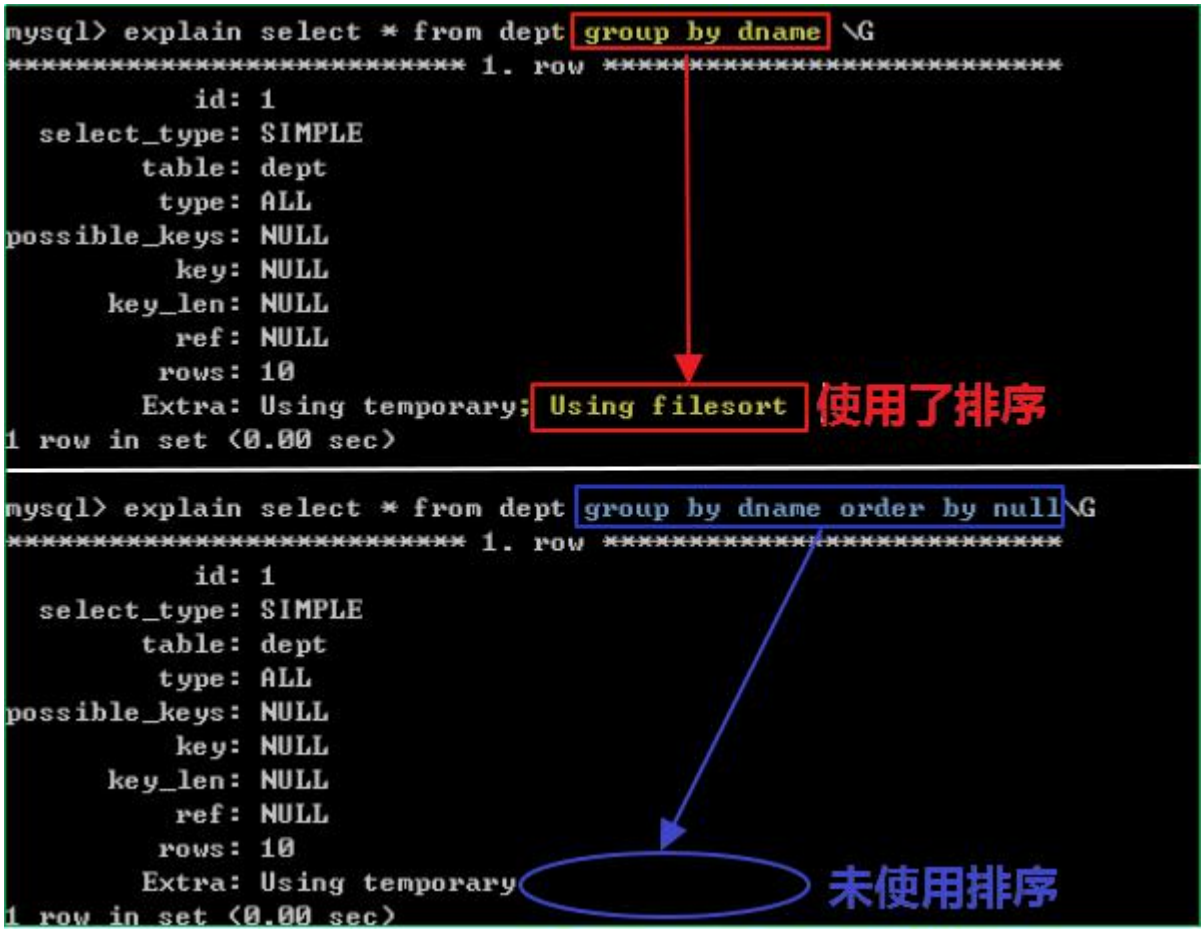
handler_read_rnd_next：这个值越高，说明查询效率越低。

（五）常用 SQL 优化

1.默认情况，在使用 group by 分组查询时，会先分组，其后还会默认对组内其他条件进行默认的排序，可能会降低速度。这与在查询中指定 order by col1, col2 类似。

如果查询中包括 group by 但用户想要避免排序结果的消耗，则可以使用 order by null 禁止排序。

例子：



2.尽量使用左连接（或右连接）来替代普通多表联查。因为使用 JOIN，MySQL 不需要在内存中创建临时表。

```
select * from dept,emp where dept.deptno=emp.deptno; 【普通联表查询】

select * from dept left join emp on dept.deptno=emp.deptno;  【左外连接，效率更高】
```

3.如果想要在含有 or 的查询语句中利用索引，则 or 之间的每个条件列都必须用到索引，如果没有索引，则应该考虑增加索引；

```
select * from 表名 where 条件 1='t1' or 条件 2='t2';
```

4.选择合适的存储引擎

在开发中，我们经常使用的存储引擎 MyISAM / INNODB / Memory

（1）MyISAM 存储：如果表对事务要求不高，同时是以查询和添加为主的，我们考虑使用 MyISAM 存储引擎,比如 BBS 中的发帖表，回复表。

（2）INNODB 存储（MySQL 5.5 以上默认）：对事务要求高，保存的数据都是重要数据，我们建议使用 INNODB,比如订单表，账号表。

【引申】 MyISAM 和 INNODB 的主要区别

```
①MyISAM 不支持事务；而 INNODB 支持事务；
```

②MyISAM 批量查询的速度比 INNODB 快（因为 INNODB 在插入数据时默认会排序）；

③MyISAM 支持全文索引；而 INNODB 不支持；

④MyISAM 是表锁；而 INNODB 在有索引时，默认为行锁，无索引时，为行锁；

⑤MyISAM 不支持外键；INNODB 支持外键；（在 PHP 开发中，通常不设置外键，通常是在程序中保证数据的一致）

（3）Memory 存储，比如我们数据变化频繁，不需要入库（重启 mysql 后，数据会清空），同时又频繁的查询和修改，我们考虑使用 memory，数据全部在内存中，速度极快。

特点	Myisam	InnoDB	BDB	Memory	Archive
批量插入的速度	高	低	高	高	非常高
事务安全		支持	支持		
全文索引	支持				
锁机制	表锁	行锁	页锁	表锁	行锁
存储限制	没有	64TB	没有	有	没有
B树索引	支持	支持	支持	支持	
哈希索引		支持		支持	
集群索引		支持			
数据缓存		支持		支持	
索引缓存	支持	支持		支持	
数据可压缩	支持				支持
空间使用	低	高	低	N/A	非常低
内存使用	低	高	低	中等	低
支持外键		支持			

（4）MySQL 行级锁、表级锁、页级锁

①表级：引擎 MyISAM 默认。直接锁定整张表，在你锁定期间，其它进程无法对该表进行写操作。如果你是写锁。则其它进程则读也不允许；

②行级：有索引时，引擎 INNODB 默认（无索引时，为表级锁）。仅对指定的记录进行加锁，这样其它进程还是可以对同一个表中的其它记录进行操作；

③页级：引擎 BDB 默认。表级锁速度快，但冲突多，行级冲突少，但速度慢。所以取了折衷的页级，一次锁定相邻的一组记录。

各自的特点：

①表级锁：开销小，加锁快；不会出现死锁；锁定粒度大，发生锁冲突的概率最高，并发度最低。

②行级锁：开销大，加锁慢；会出现死锁；锁定粒度最小，发生锁冲突的概率最低，并

发度也最高。

③页面锁：开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。

【补充】InnoDB 使用行锁定，BDB 使用页锁定。对于这两种存储引擎，都可能存在死锁。这是因为，在 SQL 语句处理期间，InnoDB 自动获得行锁定和 BDB 获得页锁定，而不是在事务启动时获得。

（Ⅰ）MySQL 表级锁有两种模式：

表共享读锁（Table Read Lock）和表独占写锁（Table Write Lock）。什么意思呢，就是说对 MyISAM 表进行读操作时，它不会阻塞其他用户对同一表的读请求，但会阻塞 对同一表的写操作；而对 MyISAM 表的写操作，则会阻塞其他用户对同一表的读和写操作。

MyISAM 表的读和写是串行的，即在进行读操作时不能进行写操作，反之也是一样。但在一定条件下 MyISAM 表也支持查询和插入的操作的并发进行，其机制是通过控制一个系统变量（concurrent_insert）来进行的：

①当其值设置为 0 时，不允许并发插入；

②当其值设置为 1 时，如果 MyISAM 表中没有空洞（即表中没有被删除的行），MyISAM 允许在一个进程读表的同时，另一个进程从表尾插入记录；

③当其值设置为 2 时，无论 MyISAM 表中有没有空洞，都允许在表尾并发插入记录。

MyISAM 锁调度是如何实现的呢，这也是一个很关键的问题。例如，当一个进程请求某个 MyISAM 表的读锁，同时另一个进程也请求同一表的写锁，此时 MySQL 将会如优先处理进程呢？通过研究表明，写进程将先获得锁（即使读请求先到锁等待队列）。但这也造成一个很大的缺陷，即大量的写操作会造成查询操作很难获得读锁，从而可能造成永远阻塞。所幸我们可以通过一些设置来调节 MyISAM 的调度行为。我们可通过指定参数，设置 set low_priority_updates=1,使优先级降低。

（Ⅱ）InnoDB 有两种模式的行锁：

1）共享锁：允许一个事务去读一行，阻止其他事务获得相同数据集的排他锁。

```
Select * from table_name where ......lock in share mode;
```

2）排他锁：允许获得排他锁的事务更新数据，阻止其他事务取得相同数据集的共享读锁和 排他写锁。

```
select * from table_name where.....for update;
```

InnoDB 行锁是通过给索引项加锁来实现的，即只有通过索引条件检索数据，InnoDB 才使用行级锁，否则将使用表锁！

（Ⅲ）插入时，更新性能优化的几个重要参数：

① bulk_insert_buffer_size 批量插入缓存大小

这个参数是针对 MyISAM 存储引擎来说的.适用于在一次性插入 100-1000+条记录时，提高效率.默认值是 8M.可以针对数据量的大小,翻倍增加;

② concurrent_insert 并发插入

当表没有空洞(删除过记录)，在某进程获取读锁的情况下,其他进程可以在表尾部进行插入;

值可以设 0 不允许并发插入，1 当表没有空洞时，执行并发插入，2 不管是否有空洞都执行并发插入（默认是 1 针对表的删除频率来设置）;

③ delay_key_write 针对 MyISAM 存储引擎,延迟更新索引

意思是说,update 记录时,先将数据 up 到磁盘,但不 up 索引,将索引存在内存里,当表关闭时,将内存索引,写到磁盘. 值为 0 不开启，1 开启. 默认开启.

④ delayed_insert_limit, delayed_insert_timeout, delayed_queue_size

延迟插入，将数据先交给内存队列，然后慢慢地插入.但是这些配置,不是所有的存储引擎都支持，目前来看，常用的 InnoDB 不支持，MyISAM 支持. 根据实际情况调大，一般默认够用了

5.能用 deciaml 的地方，尽量不要用 float;

【引申】float、double 和 decimal 的区别:

float:浮点型，占 4 个字节，数值范围为-3.4E38~3.4E38（7 个有效位）

double:双精度实型，占 8 个字节，数值范围-1.7E308~1.7E308（15 个有效位）

decimal:数字型，占 16 个字节，不存在精度损失，常用于银行帐目计算。（28 个有效位）

decimal(a,b):

a——指定指定小数点左边和右边可以存储的十进制数字的最大个数,最大精度 38。

b——指定小数点右边可以存储的十进制数字的最大个数。小数位数必须是从 0 到 a 之间的值。默认小数位数是 0。

6.对于存储引擎是 MyISAM 的数据库，如果经常做删除和修改记录的操作，要定时执行 optimize table table_name;功能对表进行碎片整理;

（六）分表技术

1.水平分割（分表）

分表是将一个大表按照一定的规则分解成多张具有独立存储空间的实体表，我们可以称

为子表，每个表都对应三个文件，MYD 数据文件，MYI 索引文件，frm 表结构文件。这些子表可以分布在同一块磁盘上，也可以在不同的机器上。app 读写的时候根据事先定义好的规则得到对应的子表名，然后去操作它。

简单理解：将一张大表，分割成多个数据类型与大表相同的子表，在访问时，根据事先定义好的规则等到对应的表名，然后去操作；

2.垂直分割（分区）

分区和分表相似，都是按照规则分解表。不同在于分表将大表分解为若干个独立的实体表，而分区是将数据分段划分在多个位置存放，可以是同一块磁盘也可以在不同的机器。分区后，表面上还是一张表，但数据散列到多个位置了。app 读写的时候操作的还是大表名字，db 自动去组织分区的数据。

简单理解：是将一张大表的某（几）列，提取成一张单独的表。一般情况两者的关系是一对一；

3.分表的几种方式

（1）mysql 集群

它并不是分表，但起到了和分表相同的作用。集群可分担数据库的操作次数，将任务分担到多台数据库上。**集群可以读写分离，减少读写压力**。从而提升数据库性能。

（2）自定义规则分表

大表可以按照业务的规则来分解为多个子表。通常为以下几种类型，也可自己定义规则。

> **①Range（范围）**：这种模式允许将数据划分不同范围。例如可以将一个表通过年份划分成若干个分区。
>
> **②Hash（哈希）**：这中模式允许通过对表的一个或多个列的 Hash Key 进行计算，最后通过这个 Hash 码不同数值对应的数据区域进行分区。例如可以建立一个对表主键进行分区的表。
>
> **③Key（键值）**：上面 Hash 模式的一种延伸，这里的 Hash Key 是 MySQL 系统产生的。
>
> **④List（预定义列表）**：这种模式允许系统通过预定义的列表的值来对数据进行分割。
>
> **⑤Composite（复合模式）**：以上模式的组合使用

（3）利用 merge 存储引擎来实现分表

这里不做详解，因为有一个更强大的技术 Mycat，可以帮我实现各种分库分表！

（七）mysql 参数调优

主要是通过修改 mysql 的 my.ini 配置文件

1.如果是 InnoDB 引擎：

```
innodb_additional_mem_pool_size = 64M
innodb_buffer_pool_size = 1G
```

2.如果是 MyISAM 引擎：

```
调整 key_buffer_size
```

当然调整参数还是要看状态，用 show status 语句可以看到当前状态，以决定改调整哪些参数。

3.还可以调整最大连接数

```
# connection limit has been reached.
max_connections=1000
```

（八）读写分离

参照《使用 Spring 配置多数据源，实现读写分离（ MySQL 实现主从复制）》 一文，

【URL】 http://blog.csdn.net/qq296398300/article/details/53994215