
Mysql 数据库优化

1 优化概述

存储层：存储引擎、字段类型选择、范式设计

设计层：索引、缓存、分区(分表)

架构层：多个 mysql 服务器设置，读写分离(主从模式)

sql 语句层：多个 sql 语句都可以达到目的的情况下，要选择性能高、速度快的 sql 语句

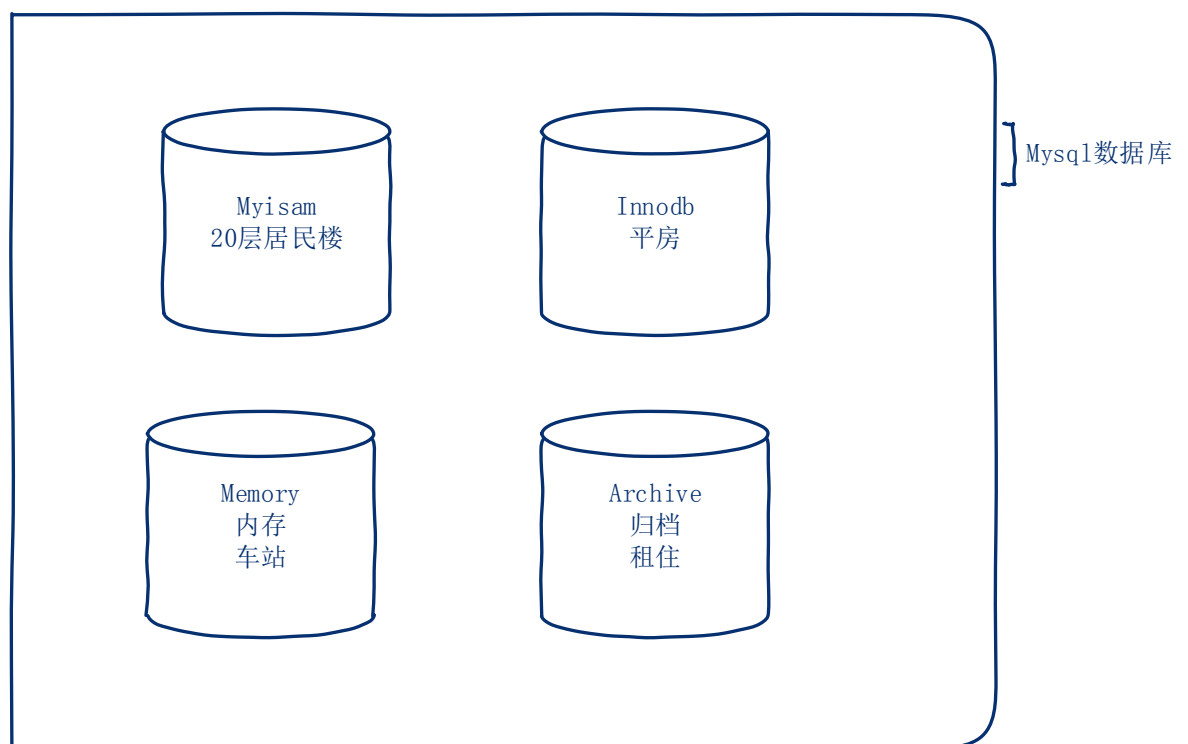
2 存储引擎

什么是存储引擎：

我们使用的数据是通过一定的技术存储在数据库当中的，数据库的数据是以文件形式组织的硬盘当中的。技术不只一种，并且每种技术有自己独特的性能和功能体现。

存储数据的技术和其功能的合并就称为“存储引擎”。

在 mysql 中经常使用的存储引擎：Myisam 或 Innodb 等等。



数据库的数据存储在不同的存储引擎里边，所有的特性就与当前的存储引擎有一定关联。

需要按照项目的需求、特点选择不同的存储引擎。

查看 mysql 中支持的全部存储引擎:

```
mysql> show engines;
```

Engine	Support	Comment
	Transactions	XA Savepoints
FEDERATED	NO	Federated MySQL storage engine
MRG_MYISAM	YES	Collection of identical MyISAM tables
MyISAM	YES	MyISAM storage engine
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)
CSV	YES	CSV storage engine
MEMORY	YES	Hash based, stored in memory, useful for temporary tables
ARCHIVE	YES	Archive storage engine
InnoDB	DEFAULT	Supports transactions, row-level locking, and foreign keys
PERFORMANCE_SCHEMA	YES	Performance Schema

2.1 InnoDB

数据库每个数据表的数据设计三方面信息: 表结构、数据、索引

```
! InnoDB ! NO ! NO ! NO !  
! InnoDB ! DEFAULT ! Supports transactions, row-level locking, and f  
foreign keys ! YES ! YES ! YES !  
! PERFORMANCE_SCHEMA ! YES ! Performance Schema  
! NO ! NO ! NO !
```

技术特点: 支持事务、行级锁定、外键

2.1.1 表结构、数据、索引的物理存储

创建一个 innodb 数据表:

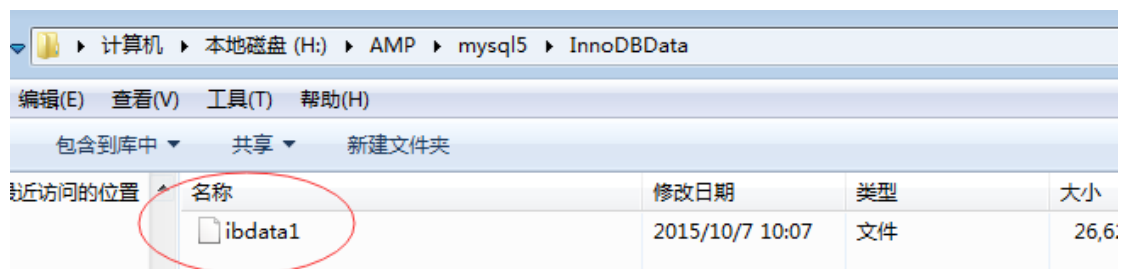
```
1  
2 create table order1(  
3     id int not null auto_increment,  
4     order_num varchar(32),  
5     primary key (id)  
6 ) engine=innodb charset=utf8;  
7
```

表结构文件:

计算机 > 本地磁盘 (H:) > AMP > mysql5 > data > data > shop0407					
编辑(E) 查看(V) 工具(T) 帮助(H)					
包含到库中 ▾ 共享 ▾ 新建文件夹					
近访问的位置	名称	修改日期	类型	大小	长度
	db.opt	2015/5/29 9:23	OPT 文件	1 KB	
	order1.frm	2015/10/7 10:07	FRM 文件	9 KB	
	english.frm	2015/5/29 9:25	FRM 文件	9 KB	
	message.frm	2015/5/29 9:25	FRM 文件	9 KB	
	sw_auth.frm	2015/5/29 9:25	FRM 文件	9 KB	

该类型 数据、索引 的物理文件位置：

所有 innodb 表的数据和索引信息都存储在以下 ibdata1 文件中



名称	修改日期	类型	大小
ibdata1	2015/10/7 10:07	文件	26,6

给 innodb 类型表 的数据和索引 创建自己对应的存储空间：

默认情况下每个 innodb 表的 数据和索引 不会创建单独的文件存储

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

设置变量，使得每个 innodb 表有独特的数据和索引 存储文件：

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

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

mysql>
```

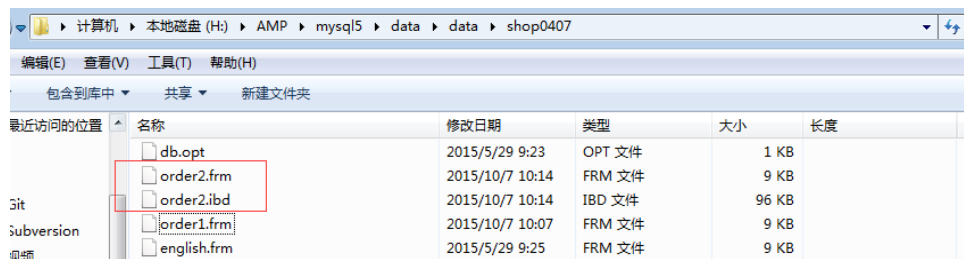
重新创建 order2 数据表：

```
8
9 create table order2(
10     id int not null auto_increment,
11     order_num varchar(32),
12     primary key (id)
13 )engine=innodb charset=utf8;
14
```

此时 order2 数据表有单独的 数据和索引 存储文件：

（后期无论 innodb_file_per_table 的设置状态如何变化，order2 的

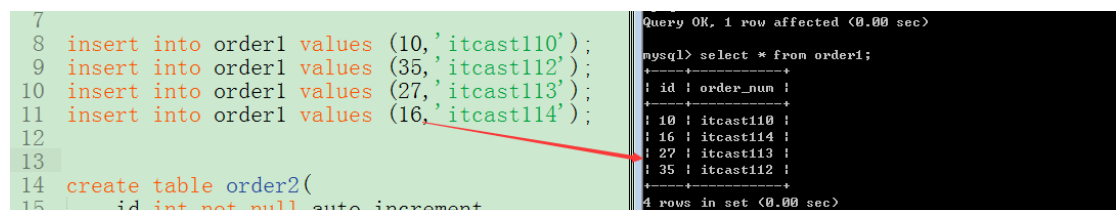
数据和索引都有独立的存储位置)



名称	修改日期	类型	大小	长度
db.opt	2015/5/29 9:23	OPT 文件	1 KB	
order2.frm	2015/10/7 10:14	FRM 文件	9 KB	
order2.ibd	2015/10/7 10:14	IBD 文件	96 KB	
order1.frm	2015/10/7 10:07	FRM 文件	9 KB	
english.frm	2015/5/29 9:25	FRM 文件	9 KB	

2.1.2 数据存储顺序

innodb 表数据的存储是按照主键的顺序排列每个写入的数据。



```
7
8 insert into order1 values (10, 'itcast110');
9 insert into order1 values (35, 'itcast112');
10 insert into order1 values (27, 'itcast113');
11 insert into order1 values (16, 'itcast114');
12
13
14 create table order2(
15     id int not null auto increment.
```

```
Query OK, 1 row affected (0.00 sec)

mysql> select * from order1;
+----+-----+
| id | order_num |
+----+-----+
| 10 | itcast110 |
| 16 | itcast114 |
| 27 | itcast113 |
| 35 | itcast112 |
+----+-----+
4 rows in set (0.00 sec)
```

该特点决定了该类型表的写入操作较慢。

2.1.3 事务、外键

该类型数据表支持事务、外键

事务：把许多写入(增、改、删)的 sql 语句捆绑在一起，要么执行、要么不执行

事务经常用于与“钱”有关的方面。

四个特性：原子、一致、持久、隔离

具体操作：

start transaction;

许多写入 sql 语句

sql 语句有问题

rollback; 回滚

commit; 提交

rollback 和 commit 只能执行一个

外键：两个数据表 A 和 B，B 表的主键是 A 表的普通字段，在 A 表看这个普通的字段就是该表的“外键”，外键的使用有“约束”。

约束：以上两个表，必须先写 B 表的数据，再写 A 表的数据

并且 A 表的外键取值必须来之 B 表的主键 id 值，不能超过其范围。

真实项目里边很少使用“外键”，因为有约束。

2.1.4 并发性

该类型表的并发性非常高

多人同时操作该数据表

为了操作数据表的时候，数据内容不会随便发生变化，要对信息进行“锁定”

该类型锁定级别为：行锁。只锁定被操作的当前记录。

2.2 Myisam

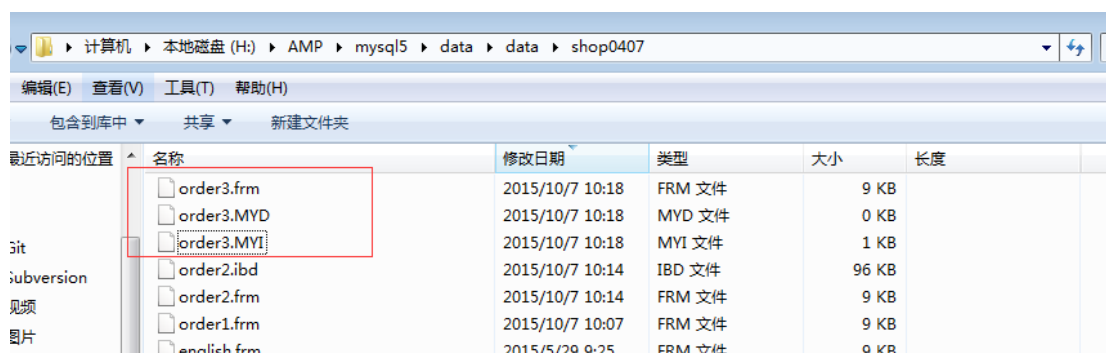
2.2.1 结构、数据、索引独立存储

该类型的数据表 表结构、数据、索引 都有独立的存储文件：

创建 Myisam 数据表

```
15▼ create table order3(  
16     id int not null auto_increment,  
17     order_num varchar(32),  
18     primary key (id)  
19 )engine=myisam charset=utf8;  
20
```

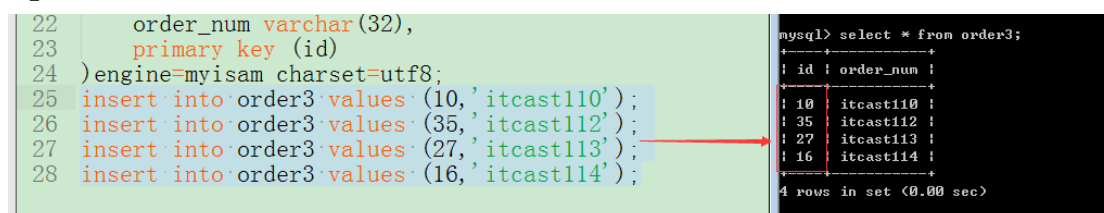
- *.frm: 表结构文件
- *.MYD: 表数据文件
- *.MYI: 表索引文件



每个 myisam 数据表的 结构、数据、索引 都有独立的存储文件
特点：独立的存储文件可以单独备份、还原。

2.2.2 数据存储顺序

myisam 表数据的存储是按照自然顺序排列每个写入的数据。



该特点决定了该类型表的写入操作较快。

2.2.3 并发性

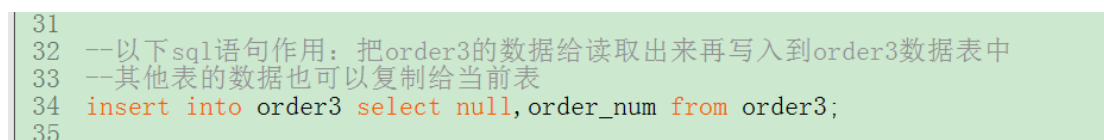
该类型并发性较低

该类型的锁定级别为：表锁

2.2.4 压缩机制

如果一个数据表的数据非常多，为了节省存储空间，需要对该表进行压缩处理。

复制当前数据表的数据：



不断复制使得 order3 数据表的数据变为 200 多万条：

```
mysql> insert into order3 select null,order_num from order3;
Query OK, 262144 rows affected (1.82 sec)
Records: 262144 Duplicates: 0 Warnings: 0

mysql> insert into order3 select null,order_num from order3;
Query OK, 524288 rows affected (3.82 sec)
Records: 524288 Duplicates: 0 Warnings: 0

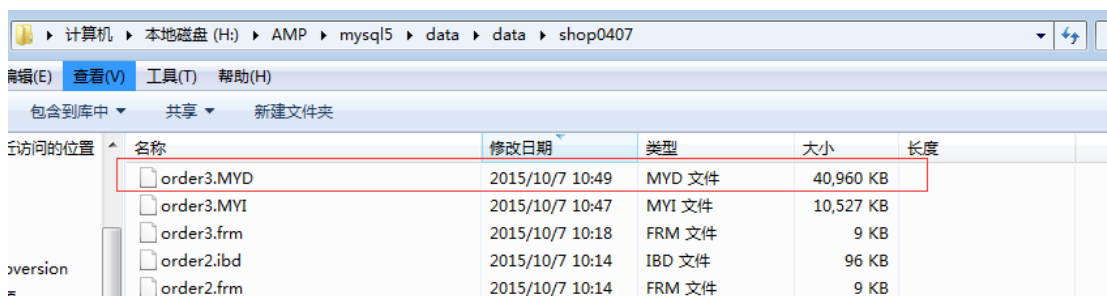
mysql> select count(*) from order3;
+-----+
| count(*) |
+-----+
| 1048576 |
+-----+
1 row in set (0.00 sec)

mysql> insert into order3 select null,order_num from order3;
Query OK, 1048576 rows affected (8.89 sec)
Records: 1048576 Duplicates: 0 Warnings: 0

mysql> select count(*) from order3;
+-----+
| count(*) |
+-----+
| 2097152 |
+-----+
1 row in set (0.00 sec)

mysql>
```

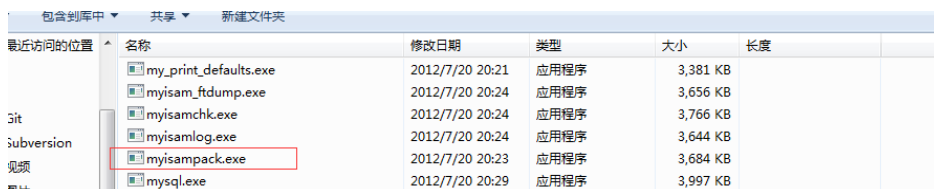
对应的存储该 200 万条信息的文件的物理大小为 40 多兆：



名称	修改日期	类型	大小	长度
order3.MYD	2015/10/7 10:49	MYD 文件	40,960 KB	
order3.MYI	2015/10/7 10:47	MYI 文件	10,527 KB	
order3.frm	2015/10/7 10:18	FRM 文件	9 KB	
order2.ibd	2015/10/7 10:14	IBD 文件	96 KB	
order2.frm	2015/10/7 10:14	FRM 文件	9 KB	

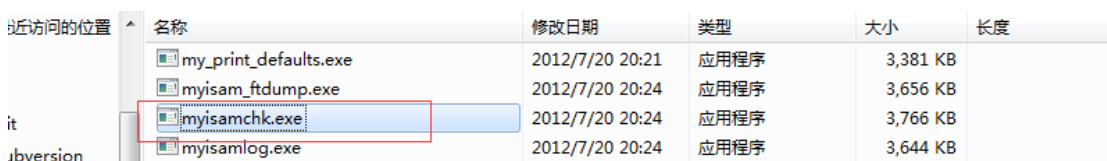
开始压缩 order3 数据表的数据

压缩工具：myisampack.exe 表名



名称	修改日期	类型	大小	长度
my_print_defaults.exe	2012/7/20 20:21	应用程序	3,381 KB	
myisam_ftdump.exe	2012/7/20 20:24	应用程序	3,656 KB	
myisamchk.exe	2012/7/20 20:24	应用程序	3,766 KB	
myisamlog.exe	2012/7/20 20:24	应用程序	3,644 KB	
myisampack.exe	2012/7/20 20:23	应用程序	3,684 KB	
mysql.exe	2012/7/20 20:29	应用程序	3,997 KB	

重建索引：myisamchk.exe -rq 表名



名称	修改日期	类型	大小	长度
my_print_defaults.exe	2012/7/20 20:21	应用程序	3,381 KB	
myisam_ftdump.exe	2012/7/20 20:24	应用程序	3,656 KB	
myisamchk.exe	2012/7/20 20:24	应用程序	3,766 KB	
myisamlog.exe	2012/7/20 20:24	应用程序	3,644 KB	

解压缩工具：myisamchk.exe --unpack 表名

最近访问的位置	名称	修改日期	类型	大小	长度
	my_print_defaults.exe	2012/7/20 20:21	应用程序	3,381 KB	
	myisam_ftdump.exe	2012/7/20 20:24	应用程序	3,656 KB	
Git	myisamchk.exe	2012/7/20 20:24	应用程序	3,766 KB	
	myisamchk.exe	2012/7/20 20:24	应用程序	3,644 KB	

order3 表信息被压缩的 60%的空间:

```
H:\AMP\mysql5\bin>myisampack.exe H:\AMP\mysql5\data\data\shop0407\order3
Compressing H:\AMP\mysql5\data\data\shop0407\order3.MYD: (2097152 records)
- Calculating statistics
- Compressing file
60%
Remember to run myisamchk -rq on compressed tables
H:\AMP\mysql5\bin>
```

order3 数据表有压缩, 但是索引没有了:

库中	名称	修改日期	类型	大小	长度
	order3.MYI	2015/10/7 11:12	MYI 文件	1 KB	
	order3.MYD	2015/10/7 10:49	MYD 文件	16,385 KB	
	order3.frm	2015/10/7 10:18	FRM 文件	9 KB	
	order2.ibd	2015/10/7 10:14	IBD 文件	96 KB	

重建索引:

```
H:\AMP\mysql5\bin>myisamchk.exe -rq H:\AMP\mysql5\data\data\shop0407\order3
- check record delete-chain
- recovering (with sort) MyISAM-table 'H:\AMP\mysql5\data\data\shop0407\order3'
Data records: 2097152
- Fixing index 1
H:\AMP\mysql5\bin>
```

索引果然被重建完毕:

名称	修改日期	类型	大小	长度
order3.MYI	2015/10/7 11:15	MYI 文件	20,853 KB	
order3.MYD	2015/10/7 10:49	MYD 文件	16,385 KB	
order3.frm	2015/10/7 10:18	FRM 文件	9 KB	
order2.ibd	2015/10/7 10:14	IBD 文件	96 KB	

刷新数据表: flush table 表名

```
mysql> flush table order3;
Query OK, 0 rows affected (0.00 sec)
mysql>
```

出现情况:

压缩的数据表是只读表, 不能写信息:

```
mysql> insert into order3 values (null, 'abcdefg');
ERROR 1036 (HY000): Table 'order3' is read only
mysql>
```

压缩的数据表有特点: 不能频繁的写入操作, 只是内容固定的数据表可以做

压缩处理

存储全国地区信息的数据表

收货地址信息数据表

如果必须要写数据：就解压该数据表，写入数据，再压缩

解压 order3 数据表，使得其可以写入数据：

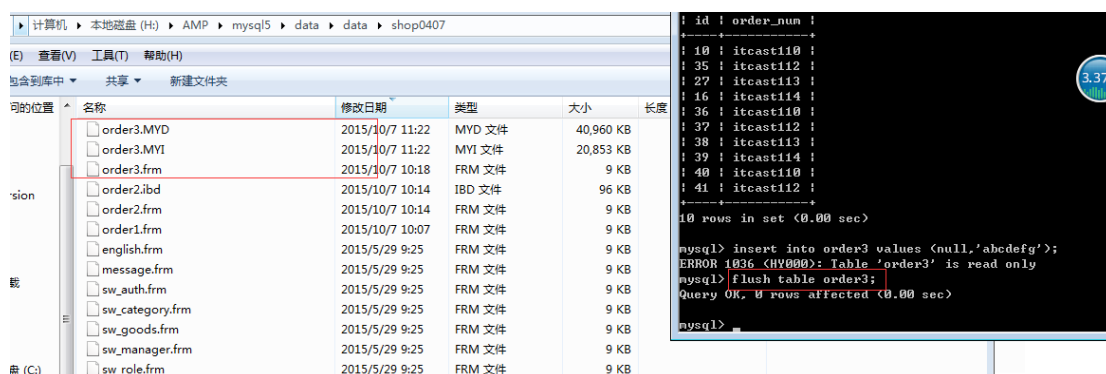
(解压同时索引自动重建)

```
H:\AMP\mysql15\bin>myisamchk --unpack H:\AMP\mysql15\data\data\shop0407\order3
- recovering (with sort) MyISAM-table 'H:\AMP\mysql15\data\data\shop0407\order3'
Data records: 2097152
- Fixing index 1
H:\AMP\mysql15\bin>
```

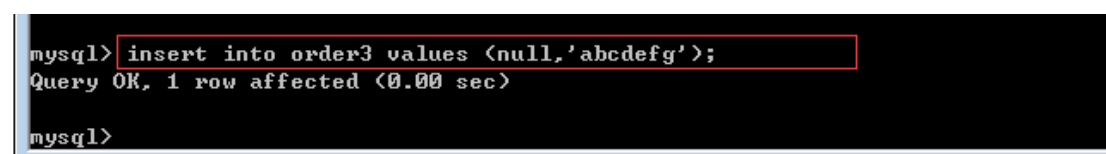
数据解压完毕：

包含到库中 共享 新建文件夹					
访问的位置	名称	修改日期	类型	大小	长度
rsion	order3.MYD	2015/10/7 11:22	MYD 文件	40,960 KB	
	order3.MYI	2015/10/7 11:22	MYI 文件	20,853 KB	
	order3.MYD.00996D46.deleted	2015/10/7 10:49	DELETED 文件	16,385 KB	
	order3.frm	2015/10/7 10:18	FRM 文件	9 KB	
	order2.ibd	2015/10/7 10:14	IBD 文件	96 KB	

执行 flush 操作，更新解压后的数据：flush table 表名；
该操作同时会删除 order3.MYD.00996D46.deleted 的压缩备份文件



此时允许给 order3 继续写入数据：



innodb 存储引擎：适合做修改、删除

Myisam 存储引擎：适合做查询、写入

2.3 Archive

归档型存储引擎，该引擎只有写入、查询操作，没有修改、删除操作
比较适合存储“日志”性质的信息。

2.4 memory

内存型存储引擎，操作速度非常快，比较适合存储临时信息，
服务器断电，给存储引擎的数据立即丢失。

2.5 存储引擎的选择

Myisam 和 innodb

网站大多数情况下“读和写”操作非常多，适合选择 Myisam 类型
例如 dedecms、phpcms 内容管理系统(新闻网站)、discuz 论坛

网站对业务逻辑有一定要求(办公网站、商城)适合选择 innodb

Mysql5.5 默认存储引擎都是 innodb 的

3 字段类型选择

3.1 尽量少的占据存储空间

int 整型

年龄: `tinyint(1)` 0-255 之间

乌龟年龄: `smallint(2)`

`mediumint(3)`

`int(4)`

`bigint(8)`

类型	大小	范围(有符号)	范围(无符号)
TINYINT	1 字节	(-128, 127)	(0, 255)
SMALLINT	2 字节	(-32768, 32767)	(0, 65535)
MEDIUMINT	3 字节	(-8 388 608, 8 388 607)	(0, 16777215)
INT 或 INTEGER	4 字节	(-2147483648, 2147483647)	(0, 4294967295)
BIGINT	8 字节	(-9223372036854775808, 9223372036854775807)	(0, 18446744073709551615)

时间类型 date

`time()` 时分秒

`datetime()` 年月日 时分秒

`year()` 年份

`date()` 年月日

`timestamp()` 时间戳(1970-1-1 到现在经历的秒数)

根据不同时间信息的范围选取不同类型的使用

3.2 数据的整合最好固定长度

`char(长度)`

固定长度, 运行速度快

长度: 255 字符限制

`varchar(长度)`

长度不固定, 内容比较少要进行部位操作, 该类型要保留 1-2 个字节保存当前数据的长度

长度: 65535 字节限制

存储汉字, 例如字符集 `utf8` 的(每个汉字占据 3 个字节), 最多可以存储 65535/3-2 字节

存储手机号码: `char(11)`

3.3 信息最好存储为整型的

时间信息可以存储为整型的(时间戳)

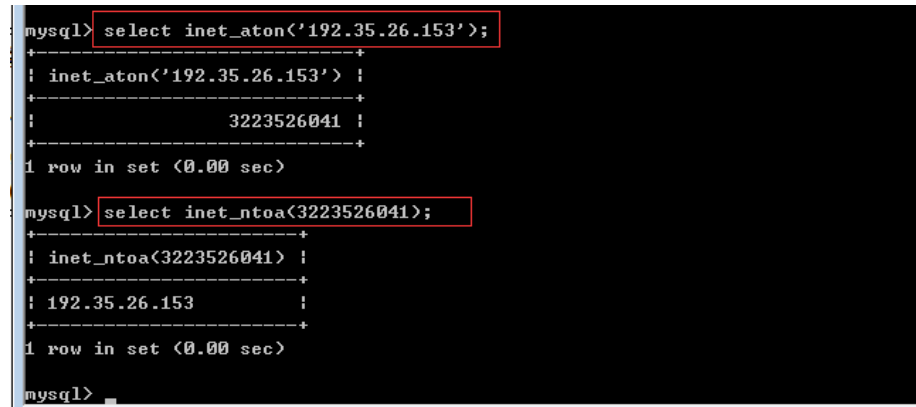
```
select from_unixstamp(时间戳) from 表名
```

set 集合类型 多选: `set('篮球','足球','棒球','乒乓球');`

enum 枚举类型 单选: `enum('男','女','保密');`

推荐使用 set 和 enum 类型, 内部会通过[整型](#)信息参数具体计算、运行。

ip 地址也可以变为整型信息进行存储(mysql 内部有算法, 把 ip 变为数字):



```
mysql> select inet_aton('192.35.26.153');
+-----+
| inet_aton('192.35.26.153') |
+-----+
|          3223526041         |
+-----+
1 row in set (0.00 sec)

mysql> select inet_ntoa(3223526041);
+-----+
| inet_ntoa(3223526041) |
+-----+
| 192.35.26.153        |
+-----+
1 row in set (0.00 sec)

mysql>
```

mysql: `inet_aton(ip)` `inet_ntoa(数字)`

php: `ip2long(ip)` `long2ip(数字)`

总结:

1. 存储引擎

数据存储技术格式

Myisam

innodb

2. 字段类型选择

原则: 占据空间小、数据长度最好固定、数据内容最好为整型的

4 逆范式

4.1 原理

数据库设计需要遵守三范式。

两个数据表：商品表 Goods、分类表 Category

Goods:	id	name	cat_id	price
	101	iphone6s	2003	6000
	204	海尔冰箱	4502	2000

.....

Category:	cat_id	name	goods_num
	2003	手机	
	4502	冰箱	

.....

需求：

计算每个分类下商品的数量是多少？

```
select c.cat_id,c.name,count(c.*) from category as c left
join goods as g on g.cat_id=c.cat_id;
```

上边 sql 语句是一个多表查询，并且还有 count 的聚合计算。

如果这样的需求很多，类似的 sql 语句查询速度没有优势，

如果需要查询速度提升，最好设置为单表查询，并且没有聚合计算。

解决方法是：给 Category 表增加一个商品数量的字段 goods_num

那么优化后的 sql 语句：

```
select cat_id,name,goods_num from category;
```

但是需要维护额外的工作：goods 商品表增加、减少数据都需要维护 goods_num 字段的信息。

以上对经常使用的需求做优化，增加一个 goods_num 字段，该字段的数据其实通过 goods 表做聚合计算也可以获得，该设计不满足三范式，因此成为“逆范式”。

4.2 三范式

① 一范式：原子性，数据不可以再分割

② 二范式：数据没有冗余

order goods

ida 编号 1 下单时间 商品信息 1 商品价格 商品描述 商品产地

idb 编号 1 下单时间 商品信息 2 商品价格 商品描述 商品产地

idb 编号 1 下单时间 商品信息 3 商品价格 商品描述 商品产地

订单表 id 编号 1 下单时间 g1,g2,g3

③ 三范式

数据表每个字段与当前表的主键产生直接关联(非间接关联)

userid name height weight orderid 编号 订单时间

优化：

userid name height weight

userid orderid

orderid 编号 订单时间

5 索引 index

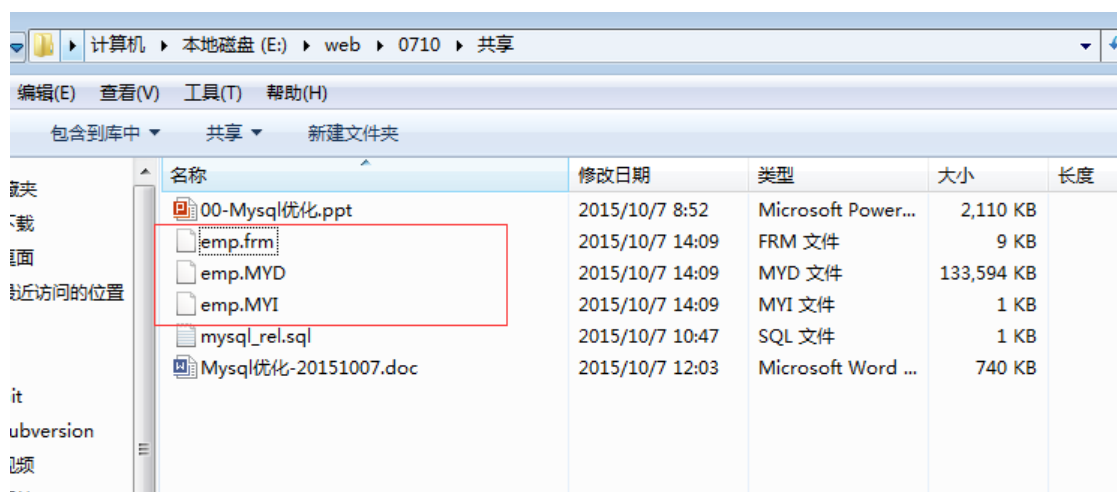
索引是优化数据库设计，提升数据库性能非常显著的技术之一。

各个字段都可以设计为索引，经常使用的索引为主键索引 primary key

索引可以明显提升查询 sql 语句的速度

5.1 是否使用索引速度的差别

直接复制文件到数据库文件目录：



被复制到 shop0407 的数据库文件目录里：

计算机 > 本地磁盘 (H:) > AMP > mysql5 > data > data > shop0407					
编辑(E) 查看(V) 工具(T) 帮助(H)					
打开 新建文件夹					
名称	修改日期	类型	大小	长度	
emp.MYD	2015/10/7 14:09	MYD 文件	133,594 KB		
emp.MYI	2015/10/7 14:09	MYI 文件	1 KB		
emp.frm	2015/10/7 14:09	FRM 文件	9 KB		
order3.MYI	2015/10/7 11:35	MYI 文件	20,853 KB		
order3.MYD	2015/10/7 11:26	MYD 文件	40,961 KB		
order3.frm	2015/10/7 10:18	FRM 文件	9 KB		
order2.ibd	2015/10/7 10:14	IBD 文件	96 KB		
order2.frm	2015/10/7 10:14	FRM 文件	9 KB		

数据库有体现 emp 数据表：

```
mysql> use shop0407;
Database changed
mysql> show tables;
+-----+
| Tables_in_shop0407 |
+-----+
| emp                 |
| english             |
| message             |
| order1              |
| order2              |
| order3              |
| sw_auth             |
| sw_category         |
| sw_goods            |
| sw_manager          |
| sw_role             |
| sw_user             |
| td_book             |
| tencent_qq          |
+-----+
14 rows in set (0.00 sec)
```

对一个没有索引的数据表进行数据查询操作：

```
+-----+
| emp | CREATE TABLE `emp` (
  `empno` int(10) unsigned NOT NULL DEFAULT '0' COMMENT '员工ID',
  `ename` varchar(20) NOT NULL DEFAULT '' COMMENT '姓名',
  `job` varchar(9) NOT NULL DEFAULT '',
  `mgr` mediumint(8) unsigned NOT NULL DEFAULT '0',
  `hiredate` date NOT NULL,
  `sal` decimal(7,2) NOT NULL,
  `comm` decimal(7,2) NOT NULL,
  `deptno` mediumint(8) unsigned NOT NULL DEFAULT '0',
  `password` char(32) DEFAULT ''
) ENGINE=MyISAM DEFAULT CHARSET=utf8
```

没有索引，查询一条记录消耗 1.49s 的时间：


```
mysql> select * from emp where empno=1345615;
+-----+-----+-----+-----+-----+-----+-----+-----+
| empno | ename | job      | mgr | hiredate | sal      | comm      | deptno | e
password |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1345615 | tmdEOg | SALESMAN | 1   | 2015-04-12 | 2000.00 | 400.00 | 158 | a
5641346eb32f30929003358c1d1a988 |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (1.49 sec)
```

一旦设置索引，再做数据查询，时间提升是百倍至千倍级的：

```
mysql> alter table emp add primary key (empno);
Query OK, 1800000 rows affected (19.35 sec)
Records: 1800000 Duplicates: 0 Warnings: 0

mysql> select * from emp where empno=1345615;
+-----+-----+-----+-----+-----+-----+-----+-----+
| empno | ename | job      | mgr | hiredate | sal      | comm      | deptno | e
password |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1345615 | tmdEOg | SALESMAN | 1   | 2015-04-12 | 2000.00 | 400.00 | 158 | a
5641346eb32f30929003358c1d1a988 |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

mysql>
```

5.2 什么是索引

索引本身是一个独立的存储单位，在该单位里边有记录着数据表某个字段和字段对应的物理空间。

索引内部有算法支持，可以使得查询速度非常快。

索引空间(字段--物理地址)					
aobama	0001110093	物理地址	id	名称(索引)	身高
luosifu	00011100C5	0001110032	104	xiaoli	165
xiaoli	0001110032	0001110093	205	aobama	185
yuehan	00011100A4	00011100A4	56	yuehan	170
		00011100C5	89	luosifu	182

有了索引，我们根据索引为条件进行数据查询速度就非常快

- ① 索引本身有“算法”支持，可以快速定位我们要找到的关键字(字段)
 - ② 索引字段与物理地址有直接对应，帮助我们快速定位要找到的信息
- 一个数据表的全部字段都可以设置索引

5.3 执行计划 explain

针对查询语句设置执行计划，当前数据库只有查询语句支持执行计划。

每个 select 查询 sql 语句执行之前，需要把该语句需要用到的各方面资源都计划好

例如：cpu 资源、内存资源、索引支持、涉及到的数据量等资源

查询 sql 语句真实执行之前所有的资源计划就是执行计划。

我们讨论的执行计划，就是看看一个查询 sql 语句是否可以使用上索引。

具体操作：

explain 查询 sql 语句\G;

一条 sql 语句在没有执行之前，可以看一下执行计划。

```
mysql> explain select * from emp where empno=1325467\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: emp
         type: const
possible_keys: PRIMARY
         key: PRIMARY
        key_len: 4
         ref: const
        rows: 1
      Extra:
1 row in set (0.00 sec)
```

可能用到的索引

真实用到的索引

主键索引删除后，该查询语句的执行计划就没有使用索引 (执行速度、效率低)

```
mysql> alter table emp drop primary key;
Query OK, 1800000 rows affected (7.62 sec)
Records: 1800000 Duplicates: 0 Warnings: 0

mysql> explain select * from emp where empno=1325467\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: emp
         type: ALL
possible_keys: NULL
         key: NULL
        key_len: NULL
         ref: NULL
        rows: 1800000
      Extra: Using where
1 row in set (0.00 sec)

mysql>
```

可能索引 和真实 索引 都没有体现

关联180万条记录

半:

5.4 索引类型

四种类型：

① 主键 primary key

auto_increment 必须给主键索引设置

信息内容要求不能为 null，唯一

② 唯一 unique index

信息内容不能重复

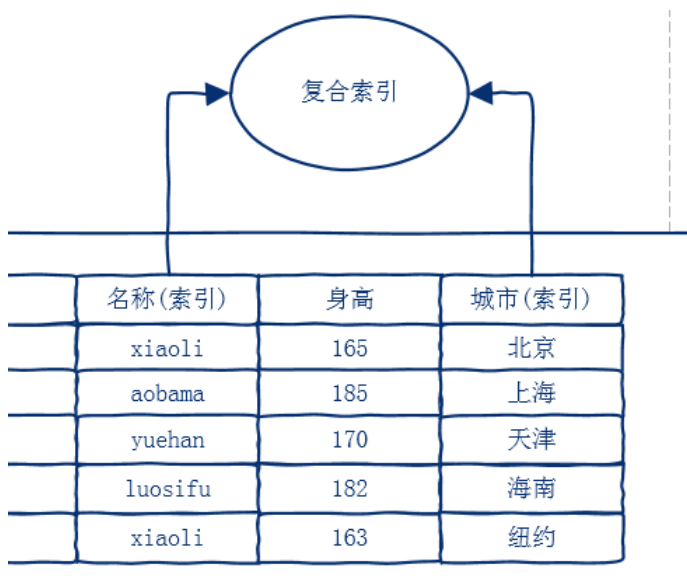
③ 普通 index

没有具体要求

④ 全文 fulltext index

myisam 数据表可以设置该索引

复合索引：索引关联的字段是多个组成的，该索引就是[复合索引](#)。



5.4.1 创建索引

创建：① 创建表时

创建一个 student 数据表，并设置各种索引：

```

39 create table student(
40     id int not null auto_increment comment '主键',
41     name varchar(32) not null default '' comment '名称',
42     height tinyint not null default 0 comment '身高',
43     addr varchar(32) not null default '' comment '地址',
44     school varchar(32) not null default '' comment '学校',
45     intro text comment '简介',
46 --创建索引(主键、唯一、普通、全文)
47     primary key (id),
48 -- 唯一/普通/全文 index [索引名称] (字段) 索引名称不设置就使用字段
49     unique index nm (name),
50     index (height),
51     fulltext index (intro)
52 )engine=myisam charset=utf8;
53
54 create table student(
55     id int not null auto_increment comment '主键',
56     name varchar(32) not null default '' comment '名称',
57     height tinyint not null default 0 comment '身高',
58     addr varchar(32) not null default '' comment '地址',
59     school varchar(32) not null default '' comment '学校',
60     intro text comment '简介',
61     primary key (id),
62     unique index nm (name),
63     index (height),
64     fulltext index (intro)
65 )engine=myisam charset=utf8;

```

查看 student 表结构可以看到各种索引是成功的：

show create table student;

```

+-----+
! student ! CREATE TABLE `student` (
  `id` int(11) NOT NULL AUTO_INCREMENT COMMENT '主键',
  `name` varchar(32) NOT NULL DEFAULT '' COMMENT '名称',
  `height` tinyint(4) NOT NULL DEFAULT '0' COMMENT '身高',
  `addr` varchar(32) NOT NULL DEFAULT '' COMMENT '地址',
  `school` varchar(32) NOT NULL DEFAULT '' COMMENT '学校',
  `intro` text COMMENT '简介',
  PRIMARY KEY (`id`),
  UNIQUE KEY `nm` (`name`),
  KEY `height` (`height`),
  FULLTEXT KEY `intro` (`intro`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8
+-----+

```

② 给现有的数据表添加索引：

```

67 --给已有的数据表设置索引
68 alter table 表名 add primary key (字段);
69 alter table 表名 add unique index [索引名] (字段);
70 alter table 表名 add index [索引名] (字段);
71 alter table 表名 add fulltext index [索引名] (字段);

```

```
mysql> alter table student add primary key <id>;
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table student add unique index <addr>;
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table student add index <school>;
Query OK, 0 rows affected (0.04 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table student add fulltext index <intro>;
Query OK, 0 rows affected (0.02 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
! student ! CREATE TABLE `student` (
  `id` int(11) NOT NULL COMMENT '主键',
  `name` varchar(32) NOT NULL DEFAULT '' COMMENT '名称',
  `height` tinyint(4) NOT NULL DEFAULT '0' COMMENT '身高',
  `addr` varchar(32) NOT NULL DEFAULT '' COMMENT '地址',
  `school` varchar(32) NOT NULL DEFAULT '' COMMENT '学校',
  `intro` text COMMENT '简介',
  PRIMARY KEY (`id`),
  UNIQUE KEY `addr` (`addr`),
  KEY `school` (`school`),
  FULLTEXT KEY `intro` (`intro`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8
```

创建一个复合索引：

索引没有名称，默认把第一个字段取出来当做名称使用。

```
mysql> alter table student add index <name,height>;
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
! student ! CREATE TABLE `student` (
  `id` int(11) NOT NULL COMMENT '主键',
  `name` varchar(32) NOT NULL DEFAULT '' COMMENT '名称',
  `height` tinyint(4) NOT NULL DEFAULT '0' COMMENT '身高',
  `addr` varchar(32) NOT NULL DEFAULT '' COMMENT '地址',
  `school` varchar(32) NOT NULL DEFAULT '' COMMENT '学校',
  `intro` text COMMENT '简介',
  PRIMARY KEY (`id`),
  UNIQUE KEY `addr` (`addr`),
  KEY `school` (`school`),
  KEY `name` (`name`, `height`),
  FULLTEXT KEY `intro` (`intro`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8
```

5.4.2 删除索引

alter table 表名 drop primary key; //删除主键索引

注意：该主键字段如果存在 auto_increment 属性，需要先删除之

alter table 表名 modify 主键 int not null comment '主键';

去除数据表主键字段的 auto_increment 属性：

```
mysql> alter table student modify id int not null comment '主键';
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> show create table student;
+-----+
| student | CREATE TABLE 'student' (
  'id' int(11) NOT NULL COMMENT '主键',
  'name' varchar(32) NOT NULL DEFAULT '' COMMENT '名称',
  'height' tinyint(4) NOT NULL DEFAULT '0' COMMENT '身高',
  'addr' varchar(32) NOT NULL DEFAULT '' COMMENT '地址',
  'school' varchar(32) NOT NULL DEFAULT '' COMMENT '学校',
  'intro' text COMMENT '简介',
  PRIMARY KEY ('id'),
  UNIQUE KEY 'nm' ('name'),
  KEY 'height' ('height'),
  FULLTEXT KEY 'intro' ('intro')
) ENGINE=MyISAM DEFAULT CHARSET=utf8
+-----+
```

禁止删除主键，原因是内部有 auto_increment 属性：

```
mysql> alter table student drop primary key;
ERROR 1075 (42000): Incorrect table definition; there can be only one auto column and it must be defined as a key
```

alter table 表名 drop index 索引名称; //删除其他索引(唯一、普通、全文)

删除主键：

```
mysql> alter table student drop primary key;
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

删除其他索引：

```
mysql> alter table student drop index nm;
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table student drop index height;
Query OK, 0 rows affected (0.02 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> alter table student drop index intro;
Query OK, 0 rows affected (0.03 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> _ 半:
```

此时数据表没有任何索引:

```
-----+
! student ! CREATE TABLE `student` (
  `id` int(11) NOT NULL COMMENT '主键',
  `name` varchar(32) NOT NULL DEFAULT '' COMMENT '名称',
  `height` tinyint(4) NOT NULL DEFAULT '0' COMMENT '身高',
  `addr` varchar(32) NOT NULL DEFAULT '' COMMENT '地址',
  `school` varchar(32) NOT NULL DEFAULT '' COMMENT '学校',
  `intro` text COMMENT '简介'
) ENGINE=MyISAM DEFAULT CHARSET=utf8
!
-----+
```

5.5 索引适合场景

5.5.1 where 查询条件

where 之后设置的查询条件字段都适合做索引。

5.5.2 排序查询

order by 字段 //排序字段适合做索引

排序字段没有索引，做排序查询就没有使用:

```
mysql> explain select * from emp order by empno limit 50\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: emp
         type: ALL
possible_keys: NULL
          key: NULL
       key_len: NULL
         ref: NULL
        rows: 1800000
    Extra: Using filesort
1 row in set (0.00 sec)

mysql> _ 半:
```

给排序字段设置索引，做排序查询就会使用：

```
mysql> alter table emp add index (empno);
Query OK, 1800000 rows affected (9.81 sec)
Records: 1800000 Duplicates: 0 Warnings: 0

mysql> explain select * from emp order by empno limit 50\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: emp
        type: index
possible_keys: NULL
      key: empno
     key_len: 4
        ref: NULL
       rows: 50
      Extra:
1 row in set (0.00 sec)

mysql>
```

where 和 order by 后边的条件字段都可以适当设置索引

5.5.3 索引覆盖

给 ename 和 job 设置一个复合索引：

```
mysql> alter table emp add index (ename,job);
ERROR 1072 (42000): Key column 'job' doesn't exist in table
mysql> alter table emp add index (ename,job);
Query OK, 1800000 rows affected (41.80 sec)
Records: 1800000 Duplicates: 0 Warnings: 0
```

desc emp

show create table emp

```
hiredate date NOT NULL,
`sal` decimal(7,2) NOT NULL,
`comm` decimal(7,2) NOT NULL,
`deptno` mediumint(8) unsigned NOT NULL DEFAULT '0',
`epassword` char(32) DEFAULT '',
KEY `empno` (`empno`),
KEY `ename` (`ename`,`job`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8
```

```
mysql> explain select ename,job from emp\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: emp
        type: index
possible_keys: NULL
      key: ename
     key_len: 91
        ref: NULL
       rows: 1800000
      Extra: Using index
1 row in set (0.00 sec)

mysql>
```

使用到了复合索引

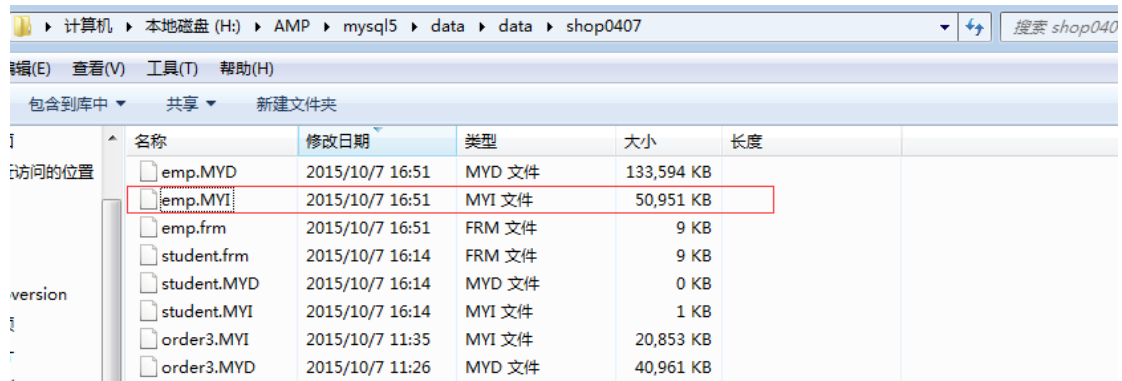
索引覆盖

索引覆盖：我们查询的全部字段 (ename, job) 已经在索引里边存在，就直接获取即可

不用到数据表中再获取了。因此成为“索引覆盖”

该查询速度非常快，效率高，该索引也称为“黄金索引”

索引本身需要消耗资源的 (空间资源、升级维护困难)：



名称	修改日期	类型	大小	长度
emp.MYD	2015/10/7 16:51	MYD 文件	133,594 KB	
emp.MYI	2015/10/7 16:51	MYI 文件	50,951 KB	
emp.frm	2015/10/7 16:51	FRM 文件	9 KB	
student.frm	2015/10/7 16:14	FRM 文件	9 KB	
student.MYD	2015/10/7 16:14	MYD 文件	0 KB	
student.MYI	2015/10/7 16:14	MYI 文件	1 KB	
order3.MYI	2015/10/7 11:35	MYI 文件	20,853 KB	
order3.MYD	2015/10/7 11:26	MYD 文件	40,961 KB	

5.5.4 连接查询

```
join join on  
goods : id name cat_id ...  
category: cat_id name ...
```

在 Goods 数据表中给外键/约束字段 cat_id 设置索引，可以提高联表查询的速度

5.6 索引原则

5.6.1 字段独立原则

`select * from emp where empno=1325467;` //empno 条件字段独立

`select * from emp where empno+2=1325467;` //empno 条件字段不独立

只有独立的条件字段才可以使用索引

独立的条件字段可以使用索引：

```
mysql> explain select * from emp where empno=1324567\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: emp
        type: ref
possible_keys: empno
      key: empno
     key_len: 4
        ref: const
        rows: 1
      Extra:
1 row in set (0.00 sec)

mysql>
```

不独立的条件字段不给使用索引:

```
mysql> explain select * from emp where empno+2=1324567\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: emp
        type: ALL
possible_keys: NULL
      key: NULL
     key_len: NULL
        ref: NULL
        rows: 1800000
      Extra: Using where
1 row in set (0.00 sec)

mysql>
```

```
mysql> select * from emp where empno+2=1324567;
+-----+-----+-----+-----+-----+-----+-----+-----+
| empno | ename | job | mgr | hiredate | sal | comm | deptno | e |
| password |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1324565 | kaysqB | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 376 | 8 |
| 6138de38a19767227aa53df3f314810 |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (1.61 sec)

mysql> select * from emp where empno=1324567;
+-----+-----+-----+-----+-----+-----+-----+-----+
| empno | ename | job | mgr | hiredate | sal | comm | deptno | e |
| password |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1324567 | terwiz | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 491 | 9 |
| c055d785de7a6f5f5000c54e3e5e458 |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

5.6.2 左原则

模糊查询, like % _

 %: 关联多个模糊内容

 _: 关联一个模糊内容

select * from 表名 like "beijing%"; //使用索引

select * from 表名 like "beijing_"; //索引索引

查询条件信息在左边出现, 就给使用索引

XXX% YYY_ 使用索引

%AAA% _ABC_ %UUU 不使用索引

没有使用索引 (中间条件查询):

```
mysql> explain select * from emp where epassword like '%abc%'G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: emp
        type: ALL
possible keys: NULL
      key: NULL
     key_len: NULL
        ref: NULL
        rows: 1800000
    Extra: Using where
1 row in set (0.00 sec)

mysql> _
半:
```

```
mysql> explain select * from emp where epassword like 'abc%'G
***** 1. row *****
      id: 1
    select_type: SIMPLE
        table: emp
        type: ALL
possible keys: NULL
      key: NULL
     key_len: NULL
        ref: NULL
        rows: 1800000
    Extra: Using where
1 row in set (0.00 sec)

mysql>
半:
```

右边条件查询

```
mysql> explain select * from emp where epassword like 'abc%\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: emp
      type: range
possible_keys: epassword
      key: epassword
    key_len: 97
      ref: NULL
     rows: 394
   Extra: Using where
1 row in set (0.00 sec)

mysql>
```

左条件查询

有使用到具体索引

半:

```
mysql> explain select * from emp where epassword like 'abc_'\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: emp
      type: range
possible_keys: epassword
      key: epassword
    key_len: 97
      ref: NULL
     rows: 394
   Extra: Using where
1 row in set (0.00 sec)

mysql>
```

半:

5.6.3 复合索引

ename 复合索引 内部有两个字段 (ename, job)

- ① ename (前者字段) 作为查询条件可以使用复合索引
- ② job (后者字段) 作为查询条件不能使用复合索引

复合索引的**第一个**字段可以使用索引:

```
mysql> explain select * from emp where ename like 'abc%\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: emp
      type: range
possible_keys: ename
      key: ename
    key_len: 62
      ref: NULL
     rows: 93
   Extra: Using where
1 row in set (0.00 sec)
```

复合索引的**其余字段**不能使用索引:

```
mysql> explain select * from emp where job like 'abc%\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: emp
        type: ALL
possible_keys: NULL
  key: NULL
  key_len: NULL
    ref: NULL
   rows: 1800000
  Extra: Using where
1 row in set (0.00 sec)
```

如果第一个字段的内容已经确定好，第二个字段也可以使用索引：

```
mysql> explain select * from emp where ename like 'abc%' and job like 'def%\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: emp
        type: range
possible_keys: ename
  key: ename
  key_len: 91
    ref: NULL
   rows: 93
  Extra: Using where
```

5.6.4 OR 原则

OR 左右的关联条件必须都具备索引 才可以使用索引：

```
! emp ! CREATE TABLE `emp` (
  `empno` int(10) unsigned NOT NULL DEFAULT '0' COMMENT '雇员ID',
  `ename` varchar(20) NOT NULL DEFAULT '' COMMENT '名字',
  `job` varchar(9) NOT NULL DEFAULT '',
  `mgr` mediumint(8) unsigned NOT NULL DEFAULT '0',
  `hiredate` date NOT NULL,
  `sal` decimal(7,2) NOT NULL,
  `comm` decimal(7,2) NOT NULL,
  `deptno` mediumint(8) unsigned NOT NULL DEFAULT '0',
  `epassword` char(32) DEFAULT '',
  KEY `empno` (`empno`),
  KEY `ename` (`ename`,`job`),
  KEY `epassword` (`epassword`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 ;

mysql> explain select * from emp where empno=1345276 or epassword like 'abc%\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: emp
        type: index_merge
possible_keys: empno,epassword
  key: empno,epassword
  key_len: 4,97
    ref: NULL
   rows: 395
  Extra: Using sort_union(empno,epassword); Using where
1 row in set (0.00 sec)
```

有使用到两个真实索引

or 的左右，只有一个有索引，导致整体都没有的使用：

```
mysql> explain select * from emp where empno=1345276 or deptno<10\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
          table: emp
          type: ALL
possible_keys: empno
      key: NULL
     key_len: NULL
         ref: NULL
        rows: 1800000
     Extra: Using where
1 row in set (0.00 sec)

mysql>
```

or的左侧有索引
右侧没有
导致整体都没有使用

半:

5.6.5 索引设计原则

字段内容需要足够花样(重复率不要太高)

性别字段不适合做索引

5.7 索引设计依据

要估算每个数据表全部的查询 sql 语句类型

分析、统计每个 sql 语句的特点(where/order by/or 等等)

原则:

- ① 被频繁执行的 sql 语句要设置
- ② 执行时间比较长的 sql 语句(可以统计)
- ③ 业务逻辑比较重要的 sql 语句(例如支付宝 2 小时内答应返现的业务逻辑)

辑)

5.8 前缀索引

设计索引的字段,不使用全部内容,而只使用该字段前边一部分内容。

如果字段的前边 N 位的信息已经可以足够标识当前记录信息,就可以把前边 N 位信息设置为索引内容,好处:索引占据的物理空间小、运行速度就非常快。

举个例子:

石清清

李德升

许成宝

王伟聪

以上 4 条记录信息,通过前边一个字就可以唯一标识当前记录信息,创建索引的时候就使用前边第一个字即可,节省空间、运行速度快。

具体实现:

① 操作 `alter table 表名 add index (字段(位数))`

② 前边到底取得多少位，才是记录的唯一标识

总记录数目/前 n 位记录数目 = 比值；

`select count(*) from 表名;`

mysql 字符串截取: `substring(字段, 开始位置 1 开始, 长度)`

```
74 --计算一个字段的n位可以唯一标识记录信息
75 select count(*)/count(distinct substring(epassword,1,1)) from emp; //非常大
76 select count(*)/count(distinct substring(epassword,1,2)) from emp;
77 select count(*)/count(distinct substring(epassword,1,3)) from emp;
78 select count(*)/count(distinct substring(epassword,1,4)) from emp;
79 select count(*)/count(distinct substring(epassword,1,5)) from emp; //稍小
80 select count(*)/count(distinct substring(epassword,1,6)) from emp;
81 select count(*)/count(distinct substring(epassword,1,7)) from emp;
82 select count(*)/count(distinct substring(epassword,1,8)) from emp;
83 select count(*)/count(distinct substring(epassword,1,9)) from emp;
84 select count(*)/count(distinct substring(epassword,1,10)) from emp; //直至稳定接近1
85 .....
86 select count(*)/count(distinct substring(epassword,1,n)) from emp;
87 得到的比值稳定或等于1，就是我们需要的n的大小

89 select count(*)/count(distinct substring(epassword,1,n)) from emp;
90 select count(*)/count(distinct epassword) from emp;
91 或者89和90行的sql语句，获得的比值如果相等，n就是唯一标识
```

从结果可以看出，密码的前 9 位就可以唯一标识当前记录信息：

```
mysql> select count(*)/count(distinct substring(epassword,1,7)) from emp;
+-----+
| count(*)/count(distinct substring(epassword,1,7)) |
+-----+
| 1.4807 |
+-----+
1 row in set (12.77 sec)

mysql> select count(*)/count(distinct substring(epassword,1,8)) from emp;
+-----+
| count(*)/count(distinct substring(epassword,1,8)) |
+-----+
| 1.4776 |
+-----+
1 row in set (13.21 sec)

mysql> select count(*)/count(distinct substring(epassword,1,9)) from emp;
+-----+
| count(*)/count(distinct substring(epassword,1,9)) |
+-----+
| 1.4774 |
+-----+
1 row in set (13.12 sec)

mysql> select count(*)/count(distinct substring(epassword,1,10)) from emp;
+-----+
| count(*)/count(distinct substring(epassword,1,10)) |
+-----+
| 1.4774 |
+-----+
```

现在给 `epassword` 创建索引，就可以只取得前 9 位即可：

```
mysql> alter table emp add index (epassword(9));
Query OK, 1800000 rows affected (1 min 18.86 sec)
Records: 1800000 Duplicates: 0 Warnings: 0

mysql> CREATE TABLE `emp` (
  `empno` int(10) unsigned NOT NULL DEFAULT '0' COMMENT '閻囧憫ID',
  `ename` varchar(20) NOT NULL DEFAULT '' COMMENT '錫整砧',
  `job` varchar(9) NOT NULL DEFAULT '',
  `mgr` mediumint(8) unsigned NOT NULL DEFAULT '0',
  `hiredate` date NOT NULL,
  `sal` decimal(7,2) NOT NULL,
  `comm` decimal(7,2) NOT NULL,
  `deptno` mediumint(8) unsigned NOT NULL DEFAULT '0',
  `epassword` char(32) DEFAULT '',
  KEY `empno` (`empno`),
  KEY `ename` (`ename`,`job`),
  KEY `epassword` (`epassword`),
  KEY `epassword_2` (`epassword`(9))
) ENGINE=MyISAM DEFAULT CHARSET=utf8
```

在查找时，也使用对应的前九位进行第一次查找，再使用全部字段进行完全匹配查找。

5.9 全文索引

Mysql5.5 Myisam 存储引擎 支持全文索引

Mysql5.6 Myisam 和 Innodb 存储引擎 都支持全文索引

```
mysql> select version();
+-----+
| version() |
+-----+
| 5.5.27 |
+-----+
1 row in set (0.00 sec)
```

原理:是先定义一个词库，然后在文章中查找每个词条(term)出现的频率和位置，把这样的频率和位置信息按照词库的顺序归纳，这样就相当于对文件建立了一个以词库为目录的索引，这样查找某个词的时候就能很快的定位到该词出现的位置。

问题:目前中文不支持全文索引。在处理英文文档的时候显然这样的方式是非常好的，因为英文自然的被空格分成若干词，只要有足够大的词汇库就能很好的处理。但是亚洲文字因为没有空格作为断词标志，所以就很难判断一个词，而且人们使用的词汇在不断的变化，而维护一个可扩展的词汇库的成本是很高的，所以问题出现了。

解决出现这样的问题使“分词”成为全文索引的关键技术。目前有两种基本的方法

全文索引可以应用在 like '%XXX%' 的操作上边。

物理地址	id	名称(索引)	身高	城市(索引)	签名
0001110032	104	xiaoli	165	北京	每天 都 快快乐乐
0001110093	205	aobama	185	上海	希望 世界 和平
00011100A4	56	yuehan	170	天津	每天 都 想打 棒球
00011100C5	89	luosifu	182	海南	虽然我做轮椅, 但是 我是总统
0001110066	342	xiaoli	163	纽约	希望 非常 漂亮

全文索引(字段—物理地址)	
世界	0001110093
和平	0001110093
棒球	00011100A4
漂亮	0001110066
...	...

创建 articles 数据表，并设置一个单列全文索引：

```

94 --全文索引
95 CREATE TABLE articles (
96     id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
97     title VARCHAR(200),
98     body TEXT
99 )engine=MyISAM charset=utf8;
100 INSERT INTO articles (title,body) VALUES
101     ('MySQL Tutorial','DBMS stands for DataBase ...'),
102     ('How To Use MySQL Well','After you went through a ...'),
103     ('Optimizing MySQL','In this tutorial we will show ...'),
104     ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
105     ('MySQL vs. YourSQL','In the following database comparison ...'),
106     ('MySQL Security','When configured properly, MySQL ...');
107 alter table articles add fulltext index index_content (title);
108

```

```

mysql> explain select * from articles where title like '%MySQL%'\G
***** 1. row *****
      id: 1
  select_type: SIMPLE
        table: articles
         type: ALL
possible_keys: NULL
          key: NULL
        key_len: NULL
         ref: NULL
         rows: 6
   Extra: Using where
1 row in set (0.00 sec)

mysql>

```

普通sql语句模糊查询 不能使用全文索引

需要变形为 match() against() 才可以使用全文索引：

```
mysql> explain select * from articles where match(title) against('ABC')\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: articles
        type: fulltext
possible_keys: index_content
      key: index_content
     key_len: 0
       ref:
      rows: 1
    Extra: Using where
1 row in set (0.00 sec)
```

```
108
109 没有使用全文索引
110 explain select * from articles where title like '%MySQL%\G
111 使用全文索引match() against()
112 explain select * from articles where match(title) against ('MySQL')\G
113
```

复合全文索引的使用：

```
94 --全文索引
95 CREATE TABLE articles (
96     id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
97     title VARCHAR(200),
98     body TEXT
99 )engine=mysam charset=utf8;
100 INSERT INTO articles (title,body) VALUES
101     ('MySQL Tutorial','DBMS stands for DataBase ...'),
102     ('How To Use MySQL Well','After you went through a ...'),
103     ('Optimizing MySQL','In this tutorial we will show ...'),
104     ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
105     ('MySQL vs. YourSQL','In the following database comparison ...'),
106     ('MySQL Security','When configured properly, MySQL ...');
107 alter table articles add fulltext index index_content (title,body);
108
109 没有使用全文索引
110 explain select * from articles where title like '%MySQL%\G
111 使用全文索引match() against()
112 explain select * from articles where match(title,body) against ('MySQL, DataBase')\G
113
```

```
mysql> explain select * from articles where match(title,body) against ('MySQL, Da
taBase')\G
***** 1. row *****
      id: 1
    select_type: SIMPLE
      table: articles
        type: fulltext
possible_keys: index_content
      key: index_content
     key_len: 0
       ref:
      rows: 1
    Extra: Using where
1 row in set (0.00 sec)

mysql>
```

5.10 索引结构(了解)

索引内部有算法，算法可以保证查询速度比较快速。

算法的基础 是 数据结构。

索引的直接称谓就是“**数据结构**”

在 Mysql 数据库中，索引是**存储引擎**层面的技术。
不同的存储引擎使用的数据结构是不一样的。

两种索引结构

- ① 非聚集索引结构 (Myisam)
- ② 聚集索引结构 (Innodb)

5.10.1 Myisam 非聚集索引结构

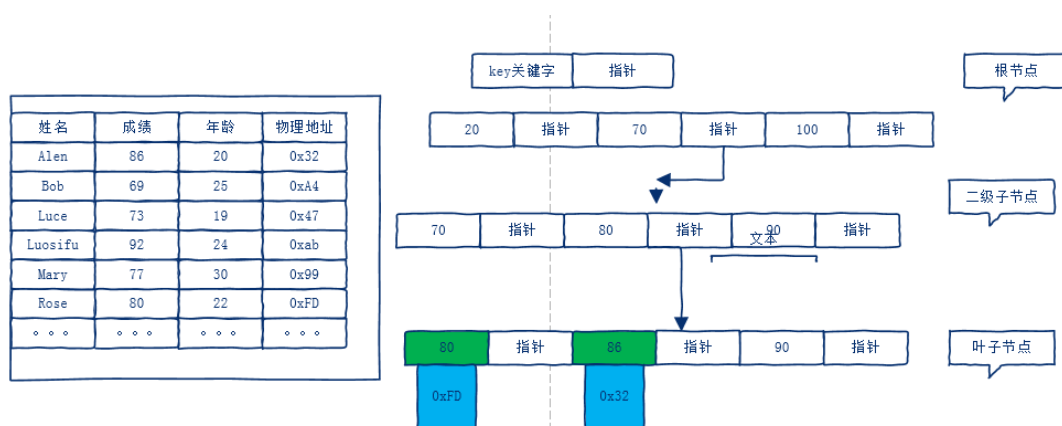
称为：B+Tree 索引结构（索引存储了数据的地址，因此分为两张表也能通过地址找到对应的数据）。

名称	修改日期	类型	大小	长度
articles.frm	2015/10/9 10:36	FRM 文件	9 KB	
articles.MYD	2015/10/9 10:36	MYD 文件	1 KB	
articles.MYI	2015/10/9 10:36	MYI 文件	3 KB	
emp.MYD	2015/10/9 10:02	MYD 文件	133,594 KB	
emp.MYI	2015/10/9 10:02	MYI 文件	120,157 KB	
emp.frm	2015/10/9 10:01	FRM 文件	9 KB	

Myisam表的索引和数据是分离的

Myisam 存储引擎的索引结构为 B+Tree：

即：节点下面还有二级节点。比如，第一级节点范围为 $[20, 70]$ ，其下面二级节点的范围划分为 $[20, 40]$ 、 $[40, 70]$ ，也即是各级节点的范围逐级细分，查找范围逐渐缩小



上图为 B+Tree 索引结构，索引结构内部分为索引节点

节点从左到右 是节点的“宽度”

节点从上到下的层数 是 结构 的“高度”

宽度或高度太大都不适合快速索引查找

宽度 和 高度 的设计会根据数据量的大小做适当的选择 (mysql 底层的算法)

该索引结构“叶子节点” 存储关键字和物理地址，非叶子节点存储关键字和指针，指针用于数据的比较、判断、向下个节点查找。

5.10.2 Innodb 聚集索引结构

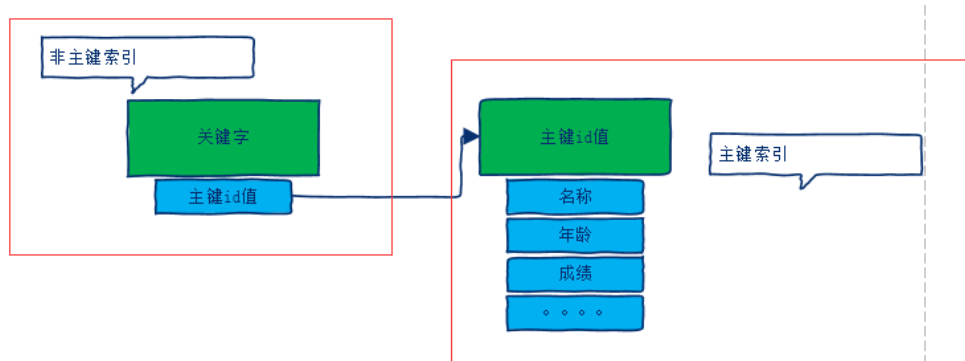
索引结构名称: B+Tree

(1) 主键索引结构

重要一点: 叶子节点的关键字 (主键 id 值) 对应整条记录信息

(2) 非主键索引结构 (唯一、普通等)

叶子节点的关键字 对应 主键 id 值



非主键索引-----innodb 的主键索引-----整条记录
这样我们可以看到: 索引 和 数据 是在一起的

innodb 表物理文件的 索引 和 数据 确实在一起:

(* .ibd 集中存储 order2 数据表的 索引和 数据)

	名称	修改日期	类型	大小	长度
	student.MYD	2015/10/7 16:14	MYD 文件	0 KB	
	student.MYI	2015/10/7 16:14	MYI 文件	1 KB	
	order3.MYI	2015/10/7 11:35	MYI 文件	20,853 KB	
	order3.MYD	2015/10/7 11:26	MYD 文件	40,961 KB	
	order3.frm	2015/10/7 10:18	FRM 文件	9 KB	
	order2.ibd	2015/10/7 10:14	IBD 文件	96 KB	
	order2.frm	2015/10/7 10:14	FRM 文件	9 KB	

概念问题:

B-Tree、B+Tree、 Binary Tree

B+Tree 是 B-Tree 的一个变形

B-Tree 与 B+Tree 的明显区别是: B-Tree 的每个节点的关键字都与“物理地址对应”

Binary Tree 二进制树结构

6 查询缓存设置

一条查询 sql 语句有可能获得很多数据, 并且有一定的时间消耗

如果该 sql 语句被频繁执行获得数据 (这些数据还不经常发生变化), 为了使得每次获得的信息速度较快, 就可以把“**执行结果**”给缓存起来, 供后续的每次使用。

6.1 查看并开启查询缓存

(1) 查看缓存

```
mysql> show variables like 'query_cache%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| query_cache_limit | 1048576 |
| query_cache_min_res_unit | 4096 |
| query_cache_size | 0 |
| query_cache_type | ON |
| query_cache_wlock_invalidate | OFF |
+-----+-----+
5 rows in set (0.00 sec)

mysql>
```

缓存大小为 0, 不能缓存

没有设置缓存之前, 每次查询都消耗 2 多秒时间:

```
mysql> select * from emp where empno=1325467;
+-----+-----+-----+-----+-----+-----+-----+-----+
| empno | ename | job | mgr | hiredate | sal | comm | deptno | e
password |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1325467 | zpahKd | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 136 | 0
4bbf89d07cdad5b61e55779246fc62c |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (2.00 sec)

mysql> select * from emp where empno=1325467;
+-----+-----+-----+-----+-----+-----+-----+-----+
| empno | ename | job | mgr | hiredate | sal | comm | deptno | e
password |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1325467 | zpahKd | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 136 | 0
4bbf89d07cdad5b61e55779246fc62c |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (2.17 sec)

mysql>
```

现在就开启缓存, 设置缓存空间大小为 64M:

```
mysql> set global query_cache_size=64*1024*1024;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

开启缓存后，查询速度有明显的提升：

```
mysql> select * from emp where empno=1325467;
+-----+-----+-----+-----+-----+-----+-----+-----+
| empno | ename | job      | mgr | hiredate | sal      | comm      | deptno | e
password |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1325467 | zpahKd | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 136 | 0
4bbf89d07cdad5b61e55779246fc62c |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (1.38 sec)

mysql> select * from emp where empno=1325467;
+-----+-----+-----+-----+-----+-----+-----+-----+
| empno | ename | job      | mgr | hiredate | sal      | comm      | deptno | e
password |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1325467 | zpahKd | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 136 | 0
4bbf89d07cdad5b61e55779246fc62c |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)
```

(2) 缓存失效

数据表或数据有变动(增加、减少、修改)，会引起缓存失效。

重新执行该语句时，缓存会重新生效。

```
mysql> insert into emp (empno,hiredate,sal,comm) values(3000000,'1995-9-2',8000,
500);
Query OK, 1 row affected (0.00 sec)

mysql> select * from emp where empno=1325467;
+-----+-----+-----+-----+-----+-----+-----+-----+
| empno | ename | job      | mgr | hiredate | sal      | comm      | deptno | e
password |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 1325467 | zpahKd | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 136 | 0
4bbf89d07cdad5b61e55779246fc62c |
+-----+-----+-----+-----+-----+-----+-----+-----+
1 row in set (1.45 sec)
```

6.2 什么情况下不会使用缓存

(1) sql 语句中有变动的信息，就不使用缓存

例如：时间信息、随机数

有时间信息的不给缓存：

```
mysql> select *,now() from emp where empno=1325467;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno	password	now()
1325467	zpahKd	SALESMAN	1	2015-04-12	2000.00	400.00	136	04bbf89d07cdad5b61e55779246fc62c	2015-10-09 11:50:23

1 row in set (1.27 sec)

```
mysql> select *,now() from emp where empno=1325467;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno	password	now()
1325467	zpahKd	SALESMAN	1	2015-04-12	2000.00	400.00	136	04bbf89d07cdad5b61e55779246fc62c	2015-10-09 11:50:31

1 row in set (1.33 sec)

```
mysql>
```

(2) 有随机数的也不给缓存:

```
mysql> select * from emp order by rand() limit 10;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno	password
214743	etERuA	SALESMAN	1	2015-04-12	2000.00	400.00	172	0b4d9f51cfedf56862a2cc871d26f18c
278093	eWtgya	SALESMAN	1	2015-04-12	2000.00	400.00	301	433af18d41da71b5b54cbbcce749c7e2
732249	wmUUJM	SALESMAN	1	2015-04-12	2000.00	400.00	353	482aebdc853d43677587272bda5f8e6b
1152066	SEujMg	SALESMAN	1	2015-04-12	2000.00	400.00	234	78daff661c441982352131500188af85
581822	ftENei	SALESMAN	1	2015-04-12	2000.00	400.00	260	7fb1d2f76ab3c029345e9dd05edd1b05
288388	gBpUKK	SALESMAN	1	2015-04-12	2000.00	400.00	232	ef854ff6bf20cbbd9630d5dd765fc967
410847	DlwziS	SALESMAN	1	2015-04-12	2000.00	400.00	141	4318184fad9b9c0985db9ffd0fe71285
578657	yUcDLW	SALESMAN	1	2015-04-12	2000.00	400.00	223	b15302e6d06798e7e341afb1a53f0808
1183635	nWUJJt	SALESMAN	1	2015-04-12	2000.00	400.00	415	569f4bca1900d8757994cea078f9fef5
1449447	SOrqFc	SALESMAN	1	2015-04-12	2000.00	400.00	228	89ae5fbae676eb40271ce7207b86c99a

10 rows in set (6.52 sec)

```
mysql>
```


6.5 查看缓存空间状态

```
mysql> show status like 'Qcache%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Qcache_free_blocks | 1 |
| Qcache_free_memory | 67094920 |
| Qcache_hits | 13 |
| Qcache_inserts | 5 |
| Qcache_lowmem_prunes | 0 |
| Qcache_not_cached | 8 |
| Qcache_queries_in_cache | 4 |
| Qcache_total_blocks | 10 |
+-----+-----+
8 rows in set (0.00 sec)

mysql>
```

缓存被使用后，空间有变小：

```
mysql> show status like 'Qcache%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| Qcache_free_blocks | 1 |
| Qcache_free_memory | 67091896 |
| Qcache_hits | 16 |
| Qcache_inserts | 7 |
| Qcache_lowmem_prunes | 0 |
| Qcache_not_cached | 8 |
| Qcache_queries_in_cache | 6 |
| Qcache_total_blocks | 14 |
+-----+-----+
8 rows in set (0.00 sec)

mysql>
```

7 分表/分区

一个数据表里边可以存储许多记录信息，如果一个数据表里边存储的数据非常多（例如 淘宝商城 的商品表），这样该商品表的相关工作量就很多（数据的增、删、改、查），负载（工作量）高到一定程度，会造成把表锁死的情况发生。为了降低商品表的负载/工作量，可以给该表拆分为多个数据表。这样每个数据表的工作量会有多降低。

Mysql5.1 版本之后就支持分表分区的设计。宏观拆分可以如下：

Goods 数据表需要拆分：Goods_1 Goods_2
Goods_3.....Goods_10

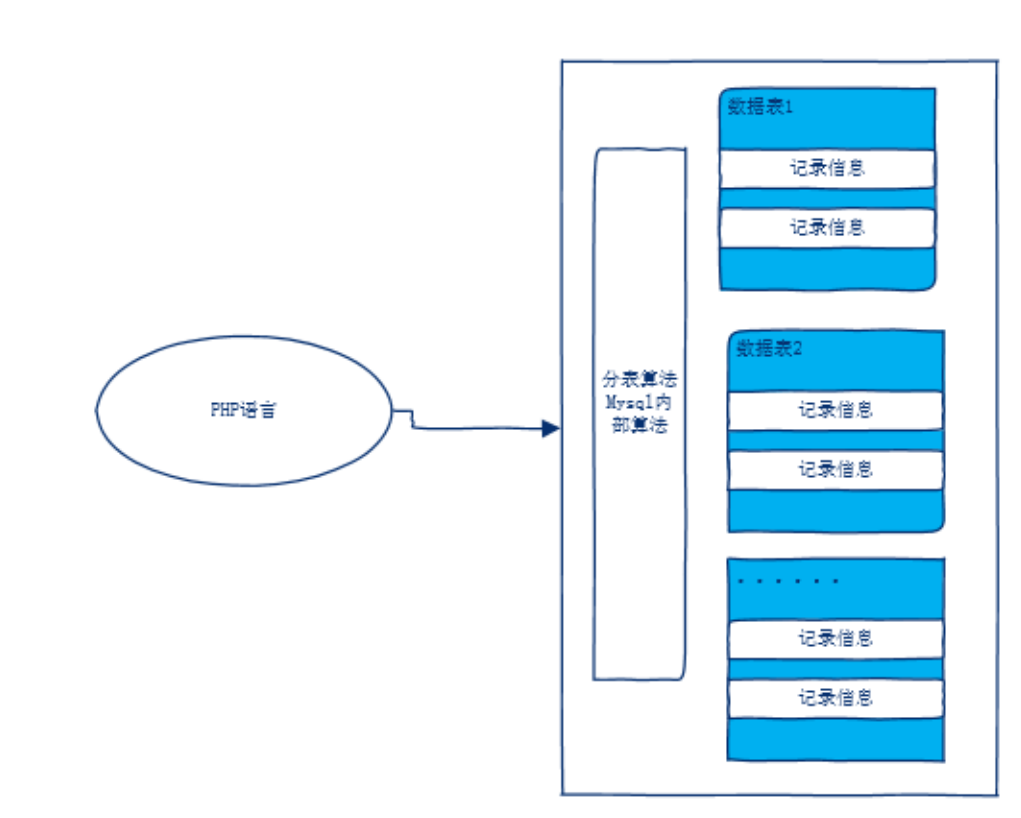
数据表拆分为以后，需要考虑 php 如何操作这些数据表。

php-----（[手动/mysql]算法）-----数据表（分表）

➤ 手动算法：需要在 php 语言里边设计操作逻辑，增加 php 语言的代码工作量

➤ mysql 算法：php 语言不需要做额外操作就可以像以往一样操作同一个

数据表的不同分区，是 mysql 分表推荐的方式



7.1 创建一个“分表/分区”数据表

Myisam 和 innodb 数据表都可以做分表设计，推荐使用 Myisam。

设计分区的字段，需要是主键的一部分。创建一个有 10 个分区的 goods 数据表：

```
115 --分表/分区
116 create table goods(
117     id int auto_increment,
118     name varchar(32) not null default '',
119     price int not null default 0,
120     pubdate datetime not null default '0000-00-00',
121     primary key (id)
122 ) engine=Myisam charset=utf8
123 partition by key(id) partitions 10;
124
125
```

可以看到 goods 数据表有 10 个分区：

名称	修改日期	类型	大小	长度
goods#p#p0.MYD	2015/10/9 14:54	MYD 文件	0 KB	
goods#p#p0.MYI	2015/10/9 14:54	MYI 文件	1 KB	
goods#p#p1.MYD	2015/10/9 14:54	MYD 文件	0 KB	
goods#p#p1.MYI	2015/10/9 14:54	MYI 文件	1 KB	
goods#p#p2.MYD	2015/10/9 14:54	MYD 文件	0 KB	
goods#p#p2.MYI	2015/10/9 14:54	MYI 文件	1 KB	
goods#p#p3.MYD	2015/10/9 14:54	MYD 文件	0 KB	
goods#p#p3.MYI	2015/10/9 14:54	MYI 文件	1 KB	
goods#p#p4.MYD	2015/10/9 14:54	MYD 文件	0 KB	
goods#p#p4.MYI	2015/10/9 14:54	MYI 文件	1 KB	
goods#p#p5.MYD	2015/10/9 14:54	MYD 文件	0 KB	
goods#p#p5.MYI	2015/10/9 14:54	MYI 文件	1 KB	
goods#p#p6.MYD	2015/10/9 14:54	MYD 文件	0 KB	
goods#p#p6.MYI	2015/10/9 14:54	MYI 文件	1 KB	
goods#p#p7.MYD	2015/10/9 14:54	MYD 文件	0 KB	
goods#p#p7.MYI	2015/10/9 14:54	MYI 文件	1 KB	
goods#p#p8.MYD	2015/10/9 14:54	MYD 文件	0 KB	
goods#p#p8.MYI	2015/10/9 14:54	MYI 文件	1 KB	
goods#p#p9.MYD	2015/10/9 14:54	MYD 文件	0 KB	
goods#p#p9.MYI	2015/10/9 14:54	MYI 文件	1 KB	
goods.frm	2015/10/9 14:54	FRM 文件	9 KB	
goods.par	2015/10/9 14:54	PAR 文件	1 KB	

上图每个分区表 都独立的*.MYD 数据文件和*.MYI 索引文件，给该表存放信息，信息会平均分摊到各个数据表里边。

7.2 四种分表分区算法

各种分区设计关联的**字段**必须是主键的一部分或者是主键本身、或者是复合主键索引的从属主键部分

求余：

key 根据指定的**字段**进行分区设计

hash 根据指定的**表达式**进行分区设计

条件：

range 字段/表达式 符合某个**条件范围**的分区设计

list 字段/表达式 符合某个**列表范围**的分区设计

7.2.1 key 的分区算法

```

116 --分表/分区
117 create table goods(
118     id int auto_increment,
119     name varchar(32) not null default '',
120     price int not null default 0,
121     pubdate datetime not null default '0000-00-00',
122     primary key (id)
123 ) engine=Myisam charset=utf8
124
125 partition by key(id) partitions 10;
126

```

7.2.2 hash 分区算法

根据指定的**表达式**进行分区设计

设计分区的时候，分区字段必须是主键的一部分：

```
mysql> create table goods_HH(  
-> id int auto_increment,  
-> name varchar(32) not null default '',  
-> price int not null default 0,  
-> pubdate datetime not null default '0000-00-00',  
-> primary key (id)  
-> ) engine=Myisam charset=utf8  
-> partition by hash(month(pubdate)) partitions 12;  
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function  
mysql>
```

```
125  
126 --分表/分区  
127 --hash 表达式分区  
128 --month()函数可以获得时间信息的“月份”信息  
129 create table goods_HH(  
130 id int auto_increment,  
131 name varchar(32) not null default '',  
132 price int not null default 0,  
133 pubdate datetime not null default '0000-00-00',  
134 primary key (id, pubdate)  
135 ) engine=Myisam charset=utf8  
136 partition by hash(month(pubdate)) partitions 12;  
137  
138
```

计算机 > 本地磁盘 (H:) > AMP > mysql5 > data > data > shop0407

编辑(E) 查看(V) 工具(T) 帮助(H)

包含到库中 共享 新建文件夹

	名称	修改日期	类型	大小	长度
方便的位置	goods_hh#p0.MYD	2015/10/9 15:09	MYD 文件	0 KB	
	goods_hh#p0.MYI	2015/10/9 15:09	MYI 文件	1 KB	
	goods_hh#p1.MYD	2015/10/9 15:09	MYD 文件	0 KB	
	goods_hh#p1.MYI	2015/10/9 15:09	MYI 文件	1 KB	
version	goods_hh#p2.MYD	2015/10/9 15:09	MYD 文件	0 KB	
	goods_hh#p2.MYI	2015/10/9 15:09	MYI 文件	1 KB	
	goods_hh#p3.MYD	2015/10/9 15:09	MYD 文件	0 KB	
	goods_hh#p3.MYI	2015/10/9 15:09	MYI 文件	1 KB	
下载	goods_hh#p4.MYD	2015/10/9 15:09	MYD 文件	0 KB	
	goods_hh#p4.MYI	2015/10/9 15:09	MYI 文件	1 KB	
	goods_hh#p5.MYD	2015/10/9 15:09	MYD 文件	0 KB	
	goods_hh#p5.MYI	2015/10/9 15:09	MYI 文件	1 KB	
磁盘 (C:)	goods_hh#p6.MYD	2015/10/9 15:09	MYD 文件	0 KB	
	goods_hh#p6.MYI	2015/10/9 15:09	MYI 文件	1 KB	
	goods_hh#p7.MYD	2015/10/9 15:09	MYD 文件	0 KB	
	goods_hh#p7.MYI	2015/10/9 15:09	MYI 文件	1 KB	
磁盘 (D:)	goods_hh#p8.MYD	2015/10/9 15:09	MYD 文件	0 KB	
磁盘 (E:)	goods_hh#p8.MYI	2015/10/9 15:09	MYI 文件	1 KB	
磁盘 (F:)	goods_hh#p9.MYD	2015/10/9 15:09	MYD 文件	0 KB	
磁盘 (G:)	goods_hh#p9.MYI	2015/10/9 15:09	MYI 文件	1 KB	
磁盘 (H:)	goods_hh#p10.MYD	2015/10/9 15:09	MYD 文件	0 KB	
	goods_hh#p10.MYI	2015/10/9 15:09	MYI 文件	1 KB	
	goods_hh#p11.MYD	2015/10/9 15:09	MYD 文件	0 KB	
	goods_hh#p11.MYI	2015/10/9 15:09	MYI 文件	1 KB	

给数据表写入数据，数据会根据“月份”添加到对应的分区表中：

```

137
138 insert into goods_HH values (null,'apple',5000,'2015-9-23');
139 insert into goods_HH values (null,'apple',5000,'2015-1-23');
140 insert into goods_HH values (null,'apple',5000,'2015-2-23');
141 insert into goods_HH values (null,'apple',5000,'2015-3-23');
142 insert into goods_HH values (null,'apple',5000,'2015-12-23');
143

```

名称	修改日期	类型	大小	长度
goods_hh#p#p0.MYD	2015/10/9 15:12	MYD 文件	1 KB	
goods_hh#p#p0.MYI	2015/10/9 15:12	MYI 文件	2 KB	
goods_hh#p#p1.MYD	2015/10/9 15:12	MYD 文件	1 KB	
goods_hh#p#p1.MYI	2015/10/9 15:12	MYI 文件	2 KB	
goods_hh#p#p2.MYD	2015/10/9 15:12	MYD 文件	1 KB	
goods_hh#p#p2.MYI	2015/10/9 15:12	MYI 文件	2 KB	
goods_hh#p#p3.MYD	2015/10/9 15:12	MYD 文件	1 KB	
goods_hh#p#p3.MYI	2015/10/9 15:12	MYI 文件	2 KB	
goods_hh#p#p9.MYI	2015/10/9 15:11	MYI 文件	2 KB	
goods_hh#p#p9.MYD	2015/10/9 15:11	MYD 文件	1 KB	
goods_hh#p#p4.MYD	2015/10/9 15:09	MYD 文件	0 KB	

7.2.3 range() 分区算法

```

144 --分表/分区
145 --range 条件范围分区
146 --根据年代给数据包进行分区设计
147 --range(字段/表达式)
148 partition by range(字段/表达式)(
149     partition 分区名字 values less than (常量),
150 )
151 create table goods_RR(
152     id int auto_increment,
153     name varchar(32) not null default '',
154     price int not null default 0,
155     pubdate datetime not null default '0000-00-00',
156     primary key (id,pubdate)
157 ) engine=Myisam charset=utf8
158 partition by range(year(pubdate))(
159     partition hou70 values less than(1980),
160     partition hou80 values less than(1990),
161     partition hou90 values less than(2000),
162     partition hou00 values less than(2010)
163 );
164

```

名称	修改日期	类型	大小	长度
goods_rr#p#hou00.MYD	2015/10/9 15:21	MYD 文件	0 KB	
goods_rr#p#hou00.MYI	2015/10/9 15:21	MYI 文件	1 KB	
goods_rr#p#hou70.MYD	2015/10/9 15:21	MYD 文件	0 KB	
goods_rr#p#hou70.MYI	2015/10/9 15:21	MYI 文件	1 KB	
goods_rr#p#hou80.MYD	2015/10/9 15:21	MYD 文件	0 KB	
goods_rr#p#hou80.MYI	2015/10/9 15:21	MYI 文件	1 KB	
goods_rr#p#hou90.MYD	2015/10/9 15:21	MYD 文件	0 KB	
goods_rr#p#hou90.MYI	2015/10/9 15:21	MYI 文件	1 KB	
goods_rr.frm	2015/10/9 15:21	FRM 文件	9 KB	

数据添加的时候根据年份，写入到对应的分区：

164	
165	insert into goods_RR values (null,'apple',5000,'1975-3-23');
166	insert into goods_RR values (null,'apple',5000,'1998-12-23');

名称	修改日期	类型	大小	长度
goods_rr#P#hou70.MYD	2015/10/9 15:22	MYD 文件	1 KB	
goods_rr#P#hou70.MYI	2015/10/9 15:22	MYI 文件	2 KB	
goods_rr#P#hou90.MYD	2015/10/9 15:22	MYD 文件	1 KB	
goods_rr#P#hou90.MYI	2015/10/9 15:22	MYI 文件	2 KB	
goods_rr#P#hou00.MYD	2015/10/9 15:21	MYD 文件	0 KB	
goods_rr#P#hou00.MYI	2015/10/9 15:21	MYI 文件	1 KB	

7.2.4 list 分区算法

168	--分表/分区
169	--list 列表范围分区
170	--根据 月份所属季节分区设计
171	partition by range(字段/表达式) (
172	partition 分区名字 values in (n1,n2,n3..),
173)
174	create table goods_LL(
175	id int auto_increment,
176	name varchar(32) not null default '',
177	price int not null default 0,
178	pubdate datetime not null default '0000-00-00',
179	primary key (id,pubdate)
180) engine=Myisam charset=utf8
181	partition by list(month(pubdate)) (
182	partition spring values in(3,4,5),
183	partition summer values in(6,7,8),
184	partition autumn values in(9,10,11),
185	partition winter values in(12,1,2)
186);

名称	修改日期	类型	大小	长度
goods_ll#P#autumn.MYD	2015/10/9 15:28	MYD 文件	0 KB	
goods_ll#P#autumn.MYI	2015/10/9 15:28	MYI 文件	1 KB	
goods_ll#P#spring.MYD	2015/10/9 15:28	MYD 文件	0 KB	
goods_ll#P#spring.MYI	2015/10/9 15:28	MYI 文件	1 KB	
goods_ll#P#summer.MYD	2015/10/9 15:28	MYD 文件	0 KB	
goods_ll#P#summer.MYI	2015/10/9 15:28	MYI 文件	1 KB	
goods_ll#P#winter.MYD	2015/10/9 15:28	MYD 文件	0 KB	
goods_ll#P#winter.MYI	2015/10/9 15:28	MYI 文件	1 KB	
goods_ll.frm	2015/10/9 15:28	FRM 文件	9 KB	
goods_ll.par	2015/10/9 15:28	PAR 文件	1 KB	

key: 该方式区分不明显 (不一定会严格平均给分区分配数据), 但是大方向明显
hash/range/list: 会根据业务特点把数据写入到对应的分区表里边。

7.3 管理分区

增加、减少分区

7.3.1 求余 (key、hash) 算法管理

增加分区: alter table 表名 add partition partitions 数量;

减少分区: alter table 表名 coalesce partition 数量;

减少分区数据要丢失

The screenshot shows a MySQL command line session and a file explorer view. The command line shows the creation of a table 'goods' with 10 partitions, followed by adding 2 more partitions and then coalescing 5 partitions. The file explorer shows the resulting files in the 'goods' directory, with annotations indicating the number of partitions remaining at each step.

```
mysql> create table goods(  
-> id int auto_increment,  
-> name varchar(32) not null default '',  
-> price int not null default 0,  
-> pubdate datetime not null default '0000-00-00',  
-> primary key (id)  
-> > engine=Myisam charset=utf8  
-> partition by key(id) partitions 10;  
Query OK, 0 rows affected (0.03 sec)  
  
mysql> alter table goods add partition partitions 2;  
Query OK, 0 rows affected (0.26 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
  
mysql>
```

增加分区

名称	修改日期	文件类型	大小
goods#P#p6.MYD	2015/10/9 15:50	MYD 文件	0 KB
goods#P#p6.MYI	2015/10/9 15:50	MYI 文件	1 KB
goods#P#p7.MYD	2015/10/9 15:50	MYD 文件	0 KB
goods#P#p7.MYI	2015/10/9 15:50	MYI 文件	1 KB
goods#P#p8.MYD	2015/10/9 15:50	MYD 文件	0 KB
goods#P#p8.MYI	2015/10/9 15:50	MYI 文件	1 KB
goods#P#p9.MYD	2015/10/9 15:50	MYD 文件	0 KB
goods#P#p9.MYI	2015/10/9 15:50	MYI 文件	1 KB
goods#P#p10.MYD	2015/10/9 15:50	MYD 文件	0 KB
goods#P#p10.MYI	2015/10/9 15:50	MYI 文件	1 KB
goods#P#p11.MYD	2015/10/9 15:50	MYD 文件	0 KB
goods#P#p11.MYI	2015/10/9 15:50	MYI 文件	1 KB

增加的两个分区

```
mysql> drop table goods 11;  
Query OK, 0 rows affected (0.01 sec)  
  
mysql> create table goods(  
-> id int auto_increment,  
-> name varchar(32) not null default '',  
-> price int not null default 0,  
-> pubdate datetime not null default '0000-00-00',  
-> primary key (id)  
-> > engine=Myisam charset=utf8  
-> partition by key(id) partitions 10;  
Query OK, 0 rows affected (0.03 sec)  
  
mysql> alter table goods add partition partitions 2;  
Query OK, 0 rows affected (0.26 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
  
mysql> alter table goods coalesce partition 5;  
Query OK, 0 rows affected (0.15 sec)  
Records: 0 Duplicates: 0 Warnings: 0  
  
mysql>
```

减少5个分区

还剩余7个分区

7.3.2 条件 (range、list) 算法管理

增加分区:

```
alter table 表名 add partition(字段, 表达式) (  
partition 分区名 values less than[in] (常量[列表]),  
partition 分区名 values less than[in] (常量[列表]),  
....  
)
```

减少分区:

alter table 表名 drop partition 分区名称;

减少分区, 会丢失对应分区的数据。

增加:

The screenshot shows a file explorer on the left with a list of files including 'goods_rr#P#hou10.MYD', 'goods_rr#P#hou10.MYI', 'goods_rr#P#hou20.MYD', 'goods_rr#P#hou20.MYI', 'goods_rr.frm', 'goods_rr.par', 'goods_rr#P#hou00.MYD', 'goods_rr#P#hou00.MYI', 'goods_rr#P#hou70.MYD', 'goods_rr#P#hou70.MYI', 'goods_rr#P#hou80.MYD', 'goods_rr#P#hou80.MYI', 'goods_rr#P#hou90.MYD', 'goods_rr#P#hou90.MYI', 'emp.MYD', 'emp.MYI', and 'emp.frm'. A red box highlights the files for 'hou10' and 'hou20'. A red arrow points from this box to a MySQL terminal window on the right. The terminal shows the following commands and output:

```
mysql> alter table goods_RR add partition range(year(pubdate))<
-> partition hou10 values less than (2020),
-> partition' at line 1
mysql>
mysql>
mysql> alter table goods_RR add partition range(year(pubdate))<
-> partition hou10 values less than (2020),
-> partition hou20 values less than (2030)
-> );
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual
corresponds to your MySQL server version for the right syntax to use near '
(year(pubdate))<
partition hou10 values less than (2020),
partition' at line 1
mysql>
mysql> alter table goods_RR add partition<
-> partition hou10 values less than (2020),
-> partition hou20 values less than (2030)
-> );
Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql>
```

A red arrow points from the text '增加两个range分区表' to the terminal output.

减少:

The screenshot shows a file explorer on the left with a list of files including 'goods_rr.frm', 'goods_rr.par', 'goods_rr#P#hou10.MYD', 'goods_rr#P#hou10.MYI', 'goods_rr#P#hou20.MYD', 'goods_rr#P#hou20.MYI', 'goods_rr#P#hou00.MYD', 'goods_rr#P#hou00.MYI', 'goods_rr#P#hou70.MYD', 'goods_rr#P#hou70.MYI', 'goods_rr#P#hou90.MYD', 'goods_rr#P#hou90.MYI', 'emp.MYD', 'emp.MYI', and 'emp.frm'. A red box highlights the files for 'hou70' and 'hou90'. A red arrow points from this box to a MySQL terminal window on the right. The terminal shows the following commands and output:

```
mysql> alter table goods_RR add partition range(year(pubdate))<
-> partition hou10 values less than (2020),
-> partition hou20 values less than (2030)
-> );
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual th
corresponds to your MySQL server version for the right syntax to use near 'ran
(year(pubdate))<
partition hou10 values less than (2020),
partition' at line 1
mysql>
mysql>
mysql> alter table goods_RR add partition<
-> partition hou10 values less than (2020),
-> partition hou20 values less than (2030)
-> );
Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql> alter table goods_rr drop partition hou80;
Query OK, 0 rows affected (0.04 sec)
Records: 0 Duplicates: 0 Warnings: 0
mysql>
```

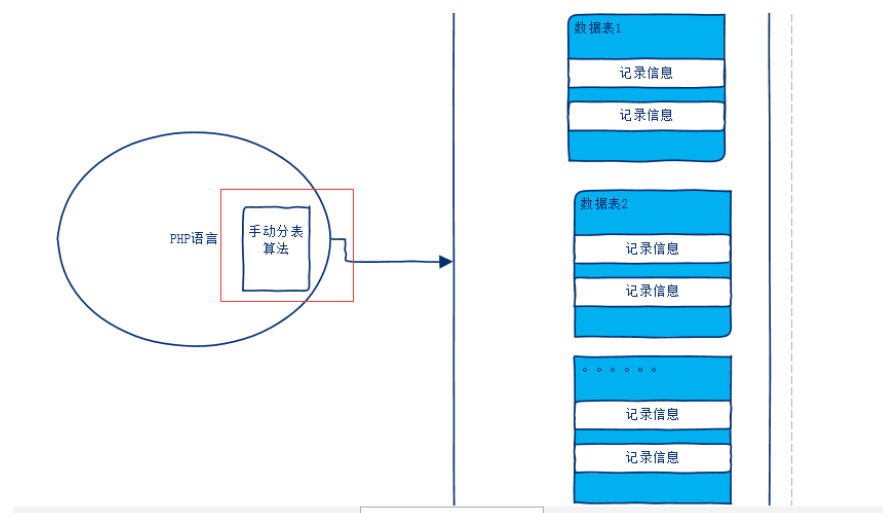
A red arrow points from the text '减少hou80的分区' to the terminal output.

7.4 物理分表设计

该分表是纯“物理分表”---即通过手工分表, 需要手动创建多个相同的表:

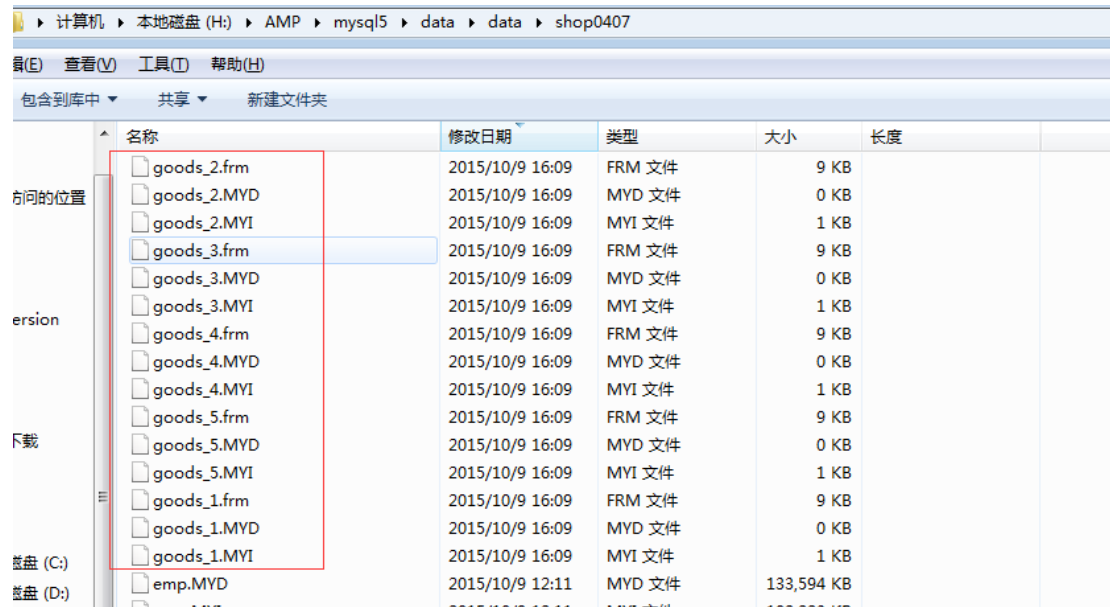
Goods: Goods_1、Goods_2、Goods_3、Goods_4、Goods_5

该物理分表需要通过 php 算法, 实现数据平均分配给每个表存储。



创建 5 张物理表 (表内部结构完全一致)：

```
198 --物理分表
199 create table goods_1(
200     id int auto_increment,
201     name varchar(32) not null default '',
202     price int not null default 0,
203     pubdate datetime not null default '0000-00-00',
204     primary key (id)
205 ) engine=Myisam charset=utf8;
206 create table goods_2(
207     id int auto_increment,
208     name varchar(32) not null default '',
209     price int not null default 0,
210     pubdate datetime not null default '0000-00-00',
211     primary key (id)
212 ) engine=Myisam charset=utf8;
213 create table goods_3(
214     id int auto_increment,
215     name varchar(32) not null default '',
216     price int not null default 0,
217     pubdate datetime not null default '0000-00-00',
218     primary key (id)
219 ) engine=Myisam charset=utf8;
```



名称	修改日期	类型	大小	长度
goods_2.frm	2015/10/9 16:09	FRM 文件	9 KB	
goods_2.MYD	2015/10/9 16:09	MYD 文件	0 KB	
goods_2.MYI	2015/10/9 16:09	MYI 文件	1 KB	
goods_3.frm	2015/10/9 16:09	FRM 文件	9 KB	
goods_3.MYD	2015/10/9 16:09	MYD 文件	0 KB	
goods_3.MYI	2015/10/9 16:09	MYI 文件	1 KB	
goods_4.frm	2015/10/9 16:09	FRM 文件	9 KB	
goods_4.MYD	2015/10/9 16:09	MYD 文件	0 KB	
goods_4.MYI	2015/10/9 16:09	MYI 文件	1 KB	
goods_5.frm	2015/10/9 16:09	FRM 文件	9 KB	
goods_5.MYD	2015/10/9 16:09	MYD 文件	0 KB	
goods_5.MYI	2015/10/9 16:09	MYI 文件	1 KB	
goods_1.frm	2015/10/9 16:09	FRM 文件	9 KB	
goods_1.MYD	2015/10/9 16:09	MYD 文件	0 KB	
goods_1.MYI	2015/10/9 16:09	MYI 文件	1 KB	
emp.MYD	2015/10/9 12:11	MYD 文件	133,594 KB	
emp.MYI	2015/10/9 12:11	MYI 文件	102,222 KB	

通过 php 的算法，实现给分表进行数据的增、删、改、查操作：

```

238
239 --① 根据id获得指定的一条记录信息
240 --到指定的分表获得指定的记录信息
241 $yu = $id%5;
242 $sql = "select * from goods_$yu where id=$id";
243 --以上可以根据$id获得指定的分表, 进而操作该$id代表的记录信息
244 --那么修改、删除一样可以操作
245
246
247 --② 给指定的分表写入指定的记录信息
248 $maxId = "select max(id) from goods_fu";
249 $trueid = $maxId+1;
250 $yu = $trueid%5;
251 $sql = "insert into goods_$yu values (.....)";
252 $sql = "insert into goods_fu values ($trueid)";--注意辅助要把最大的id信息给维护起来
253 --不清楚数据写入到那个分表里边
254 --答: 获得数据表内部最大的主键id值(获得全部分表最大的主键id值, 取其中最大的)
255
256 -- 另一种解决: 维护一张辅助表goods_fu(id一个字段), 只要往任何一个分表写入一条数据
257 -- 该辅助表id都累加一
258
259
260
261
262
263
264
265
266
267 --② 给指定的分表写入指定的记录信息
268 $sql = "insert into goods_fu values (null)";--注意辅助要把最大的id信息给维护起来
269 $maxId = "select last_insert_id()";--获得刚刚insert语句的生成的主键id
270
271 $yu = $maxId%5;
272 $sql = "insert into goods_$yu values ($maxId,.....)";
273
274 --不清楚数据写入到那个分表里边
275 --答: 获得数据表内部最大的主键id值(获得全部分表最大的主键id值, 取其中最大的)
276
277 -- 另一种解决: 维护一张辅助表goods_fu(id一个字段), 只要往任何一个分表写入一条数据
278 -- 该辅助表id都累加一
279
280
281
282
283

```

7.5 垂直分表

对记录进行分割并存储到许多不同的表, 称为“水平分表”

对字段进行分割并存储到许多不同表, 称为“垂直分表”

一个数据表, 内部有许多字段, 有的字段频繁被操作, 有的字段很少被操作。这样当操作数据表中一些字段的时候, 没有直接业务关系的字段也需要给其分配相应的资源, 这样速度会稍慢, 还要消耗系统额外的工作量。

例如数据表 (student) 有如下字段:

id 登录名 名称 密码 生日 身高 体重 手机号码 qq 号码
简介 城市

student_zhu: id 登录名 密码 手机号码

student_fu: id 名称 生日 身高 体重 qq 号码 简介 城市

以上两个数据表 (student_zhu、student_fu), 其中 zhu 表需要被频繁操作。

后期数据维护需要同时维护两个数据表

insert into student_zhu values (.....)

生成一个 zhu 的 id 信息 \$zhuid

那么就: insert into student_fu values (\$zhuid)

8 架构(集群)设计

一个 mysql 服务器的操作分为：增、删、改、查

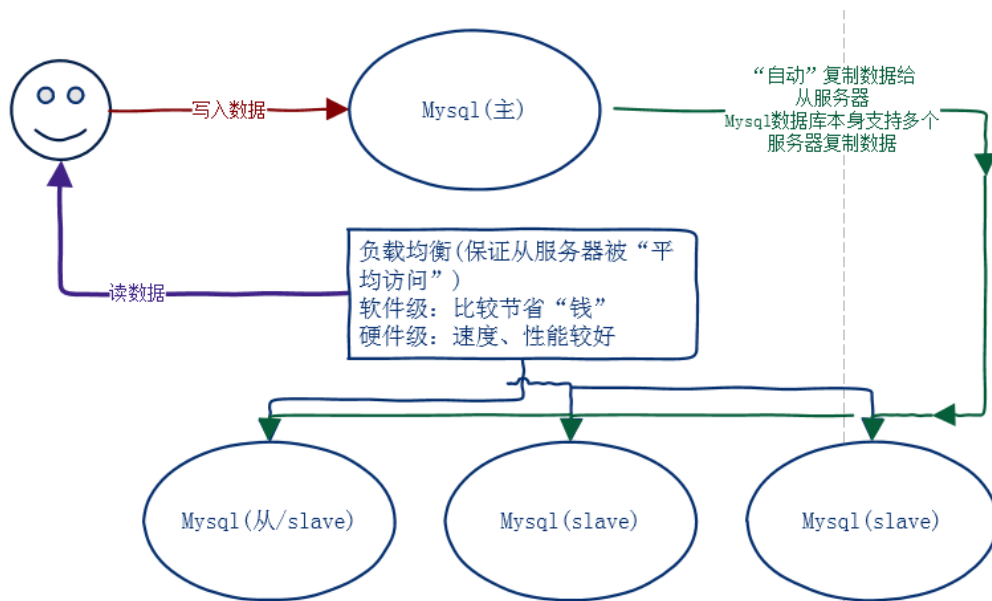
其中查询操作最为**频繁**：查询/写入 = 7/1

查询操作本身还最**消耗**资源

架构设计：原先有一个 mysql 服务器做的工作现在平摊给多个 mysql 服务器实现。

多个数据库设计与 Redis 的分布式设计雷同：

主从模式（一主多从/读写分离）



安装多个服务器（多个 mysql 服务器）

负载均衡 软件

主 mysql 给 从 mysql 同步数据

9 慢查询日志

系统运行起来，内部需要执行许多 sql 语句

此时要把查询**速度很慢**的 sql 语句给统计出来，并做优化设计。

设定一个**时间阈值**，超过该时间，就说明 sql 语句很慢。

```
mysql> show variables like 'slow_query%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| slow_query_log | OFF   |
| slow_query_log_file | H:\AMP\mysql5\data\Data\jinnan-PC-slow.log |
+-----+-----+
2 rows in set (0.00 sec)
```

慢查询sql语句存储的日志文件

开启慢查询日志:

```
mysql> set global slow_query_log = 1;
Query OK, 0 rows affected (0.01 sec)

mysql> show variables like 'slow_query%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| slow_query_log | ON    |
| slow_query_log_file | H:\AMP\mysql5\data\Data\jinnan-PC-slow.log |
+-----+-----+
2 rows in set (0.00 sec)
```

```
mysql> show variables like 'long_query_time%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| long_query_time | 10.000000 |
+-----+-----+
1 row in set (0.00 sec)
```

查看慢查询的时间阈值

设置时间阈值 (set 后边没有 global):

```
mysql> set long_query_time=2;
Query OK, 0 rows affected (0.00 sec)

mysql> show variables like 'long_query_time%';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| long_query_time | 2.000000 |
+-----+-----+
1 row in set (0.00 sec)
```

设置时间阈值为2秒

```
P:\mysql5\data\data\jinnan-PC-slow.log - Sublime Text 2 (UNREGISTERED)
Selection Find View Goto Tools Project Preferences Help

10.php x qq.sql x mysql-1.sql x mysql_rel.sql x jinnan-PC-slow.log x 101.php x 102.php x

1 H:\AMP\mysql5\bin\mysqld, Version: 5.5.27 (MySQL Community Server (GPL)). started with:
2 TCP Port: 3306, Named Pipe: MySQL
3 Time Id Command Argument
4 # Time: 151008 22:23:23
5 # User@Host: root[root] @ localhost [127.0.0.1]
6 # Query_time: 5.213298 Lock_time: 0.001000 Rows_sent: 5 Rows_examined: 3600007
7 use test;
8 SET timestamp=1444314203;
9 select * from emp order by rand() limit 5;
10 H:\AMP\mysql5\bin\mysqld, Version: 5.5.27 (MySQL Community Server (GPL)). started with:
11 TCP Port: 3306, Named Pipe: MySQL
12 Time Id Command Argument
13 # Time: 151009 17:29:33
14 # User@Host: root[root] @ localhost [127.0.0.1]
15 # Query_time: 7.572433 Lock_time: 0.000000 Rows_sent: 10 Rows_examined: 2800011
16 use shop0407;
17 SET timestamp=1444382973;
18 select * from emp order by empno limit 1000000,10;
19
```

执行超过2秒的慢查询sql语句被记录

10 大量写入记录信息

保证数据[非常快](#)地写入到数据库中

```
insert into 表名 values (),(),(),();
```

以上一个 insert 语句可以同时写入多条记录信息，但是不要写入太多，避免意外情况发生。

可以一次少写一些，例如每次写入 1000 条，这样 100 万的记录信息，执行 1000 次 insert 语句就可以了。[分批分时间](#)把数据写入到数据库中。

以上设计写入大量数据的方法损耗的时间：

写入数据 (1000 条) -----> 为 1000 条数据维护索引

写入数据 (1000 条) -----> 为第 2 个 1000 条数据维护索引

.....

写入数据 (1000 条) -----> 为第 1000 个 1000 条数据维护索引

以上设计写入 100 万条记录信息，时间主要都被“维护索引”给占据了

如果做优化：就可以减少索引的维护，达到整体运行时间变少。

(索引维护不需要做 1000 次，就想做一次)

解决：

先把索引给停掉，专门把数据先写入到数据库中，最后在一次性维护索引

10.1 Myisam 数据表

(1) 数据表中已经[存在](#)数据 (索引已经存在一部分)

```
alter table 表名 disable keys;
```

大量写入数据

```
alter table 表名 enable keys; //最后统一维护索引
```

(2) 数据表中没有数据(索引内部没有东西)

```
alter table 表名 drop primary key ,drop index 索引名称(唯一/普通/全文);
```

大量写入数据

```
alter table 表名 add primary key(id),(唯一/全文)index 索引名 (字段);
```

10.2 Innodb 数据表

该存储引擎支持“事务”

该特性使得我们可以一次性写入大量 sql 语句

具体操作:

```
start transaction;
```

大量数据写入(100 万条记录信息 insert 被执行 1000 次)

事务内部执行的 insert 的时候,数据还没有写入到数据库

只有数据真实写入到数据库才会执行“索引”维护

```
commit;
```

commit 执行完毕后最后会自动维护一次“索引”;

11 单表、多表查询

数据库操作有的时候设计到 联表查询、子查询操作。

复合查询一般要涉及到多个数据表,

多个数据表一起做查询好处: sql 语句逻辑清晰、简单

其中不妥当的地方是:消耗资源比较多、时间长

不利于数据表的并发处理,因为需要长时间锁住多个表

例如:

查询每个品牌下商品的总数量 (Goods/Brand)

```
Goods:id name bd_id
```

```
Brand: bd_id name
```

```
select b.bd_id,b.name,count(g.*) from Brand b join Goods  
g on b.bd_id=g.bd_id group by b.bd_id;
```

以上 sql 语句总运行时间是 5s

但是业务要求是数据库的并发性要高,就需要把“多个查询”变为“单表查询”

步骤:

```
① select bd_id,count(*) from Goods group by bd_id; //
```

查询每个品牌的商品数量 //3s

② `select bd_id,name from Brand; //3s`

③ 在 php 通过逻辑代码整合① 和 ② //1s

12 . limit 使用

数据分页使用 limit;

limit 偏移量, 长度 (每页条数);

偏移量: (当前页码-1) * 每页条数

分页实现:

每页获得 10 条信息:

`limit 0,10;`

`limit 10,10;`

`limit 20,10;`

`limit 30,10;`

`limit 990,10; //第 100 页`

`limit 9990,10; //第 1000 页`

`limit 99990,10; //第 10000 页`

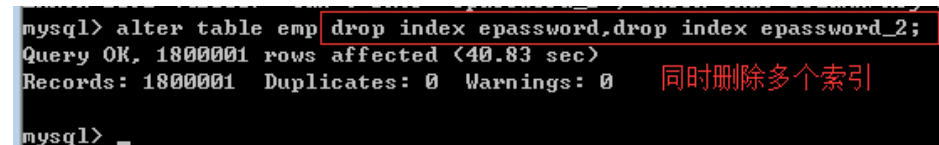
`limit 999990,10; //第 100000 页`

`limit 1499990,10; //第 150000 页`

`limit 1500000,10; //第 150001 页`

`select * from emp limit 1500000,10; //慢 1 秒多时间`

`select * from emp where empno>1600001 limit 10; //快 0.00 秒级`



```
mysql> alter table emp drop index epassword,drop index epassword_2;
Query OK, 1800001 rows affected (40.83 sec)
Records: 1800001  Duplicates: 0  Warnings: 0
mysql>
```

数据表目前有 empno [主键索引](#):

```

+-----+
| emp | CREATE TABLE `emp` (
| `empno` int(10) unsigned NOT NULL DEFAULT '0' COMMENT '员工ID',
| `ename` varchar(20) NOT NULL DEFAULT '' COMMENT '姓名',
| `job` varchar(9) NOT NULL DEFAULT '',
| `mgr` mediumint(8) unsigned NOT NULL DEFAULT '0',
| `hiredate` date NOT NULL,
| `sal` decimal(7,2) NOT NULL,
| `comm` decimal(7,2) NOT NULL,
| `deptno` mediumint(8) unsigned NOT NULL DEFAULT '0',
| `password` char(32) DEFAULT '',
| PRIMARY KEY (`empno`),
| KEY `ename` (`ename`,`job`)
| ) ENGINE=MyISAM DEFAULT CHARSET=utf8
+-----+

```

limit 偏移量, 长度; 运行时间较长:

```

mysql> select * from emp limit 1500000,10;
+-----+
| empno | ename | job | mgr | hiredate | sal | comm | deptno | password |
+-----+
| 1600002 | QITouM | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 288 | 96ea84a8bda5f3466dd9c7e2ec6cfbc6 |
| 1600003 | CcEQrj | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 460 | 1795cff4d62222024dae6081dd6fbc9f |
| 1600004 | YiUZpx | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 222 | 3d684dd92cd338468ba809c1b81decff |
| 1600005 | MxcShh | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 150 | 7ab13ada6057d9a233a36d5d79269b9e |
| 1600006 | YdsJbm | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 74 | eca26b58ec04b4b701376cfc41b4b08e |
| 1600007 | WnCUXZ | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 66 | 68b82f4593a7182ea417737d86b0746f |
| 1600008 | EJrOWq | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 404 | b4e448b6ccffac9cd88a80d5abd49db9 |
| 1600009 | ZEZgLi | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 355 | 0e5be2722f7403b6361cc1bcbabc88f5 |
| 1600010 | XLJqzB | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 90 | f871057d4f79e47c04162d5ea624cc0a |
| 1600011 | lJyyXq | SALESMAN | 1 | 2015-04-12 | 2000.00 | 400.00 | 368 | deb953900c34d6a59e86b0952bc3156e |
+-----+
10 rows in set (1.03 sec)

```

单纯运行 limit 运行时间比较长, 内部没有使用索引, 翻页效果 之前页码的信息给获得出来, 但是“越”过去, 因此比较浪费时间

现在对获得相同页码信息的 sql 语句进行优化

由单纯 limit 变为 where 和 limit 的组合:

执行速度明显加快, 因为其有使用 where 条件字段的索引


```
mysql> select * from emp where empno>1600001 limit 10;
```

empno	ename	job	mgr	hiredate	sal	comm	deptno	e password
1600002	QITouM	SALESMAN	1	2015-04-12	2000.00	400.00	288	96ea84a8bda5f3466dd9c7e2ec6cfbc6
1600003	CcEQrj	SALESMAN	1	2015-04-12	2000.00	400.00	460	1795cff4d62222024dae6081dd6fbc9f
1600004	YiUZpx	SALESMAN	1	2015-04-12	2000.00	400.00	222	3d684dd92cd338468ba809c1b81decff
1600005	MxcShh	SALESMAN	1	2015-04-12	2000.00	400.00	150	7ab13ada6057d9a233a36d5d79269b9e
1600006	YdsJbm	SALESMAN	1	2015-04-12	2000.00	400.00	74	eca26b58ec04b4b701376cfc41b4b08e
1600007	WnCUXZ	SALESMAN	1	2015-04-12	2000.00	400.00	66	8b82f4593a7182ea417737d86b0746f
1600008	EJrOWq	SALESMAN	1	2015-04-12	2000.00	400.00	404	4e448b6ccffac9cd88a80d5abd49db9
1600009	ZEZgLi	SALESMAN	1	2015-04-12	2000.00	400.00	355	0e5be2722f7403b6361cc1bcbab88f5
1600010	XLJqzB	SALESMAN	1	2015-04-12	2000.00	400.00	90	f871057d4f79e47c04162d5ea624cc0a
1600011	lJyyXq	SALESMAN	1	2015-04-12	2000.00	400.00	368	deb953900c34d6a59e86b0952bc3156e

```
10 rows in set (0.00 sec)
```

```
mysql>
```

```
mysql> explain select * from emp limit 1500000,10\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: emp
type: ALL
possible_keys: NULL
key: NULL
key_len: NULL
ref: NULL
rows: 1800001
Extra:
1 row in set (0.00 sec)
```

单纯limit没有“索引”可用

```
mysql> explain select * from emp where empno>1600001 limit 10\G
***** 1. row *****
id: 1
select_type: SIMPLE
table: emp
type: range
possible_keys: PRIMARY
key: PRIMARY
key_len: 4
ref: NULL
rows: 248159
Extra: Using where
1 row in set (0.00 sec)
```

```
mysql>
```

有索引使用“主键索引”

13. order by null

强制不排序

有的 sql 语句在执行的时候，本身默认会有排序效果

但是有的时候我们的业务不需要排序效果，就可以进行强制限制，进而“节省默认排序”的资源。

group by 字段；

获得的结果默认情况会根据“分组字段”进行排序：

```
mysql> select goods_category_id,count(*) from sw_goods group by goods_category_id;
+-----+-----+
| goods_category_id | count(*) |
+-----+-----+
| 0 | 76 |
| 2 | 1 |
| 3 | 14 |
| 4 | 3 |
| 5 | 1 |
| 8 | 1 |
| 13 | 2 |
| 14 | 2 |
| 15 | 1 |
+-----+-----+
9 rows in set (0.00 sec)
```

order by null 强制不排序，节省对应资源：

```
mysql> select goods_category_id,count(*) from sw_goods group by goods_category_id order by null;
+-----+-----+
| goods_category_id | count(*) |
+-----+-----+
| 4 | 3 |
| 0 | 76 |
| 8 | 1 |
| 3 | 14 |
| 2 | 1 |
| 5 | 1 |
| 13 | 2 |
| 15 | 1 |
| 14 | 2 |
+-----+-----+
9 rows in set (0.00 sec)
```

14 总结

14.1 Myisam 和 Innodb 存储引擎特点

Myisam

表结构、数据、索引 分别有对应的存储文件

写入数据非常快，安装自然顺序写入数据

数据稳定后可以压缩数据信息

支持全文索引

并发性：少低，锁表操作

Innodb

表结构有单独存储文件，数据和索引共享同一个存储文件 (ibdata1、*.ibd)

ibdata1 是全部 innodb 表的数据和索引的存储文件

*.ibd 是每个 innodb 表的数据和索引的存储文件

支持事务和外键的

并发性：好，操作数据表时锁定记录 (行)

Mysql5.6 版本有支持全文索引

14.2 字段类型选择

1) 给数据分配的空间要尽量小

tinyint smallint mediumint int bigint

时间：datetime date year time timestamp

2) 数据最好整合为固定长度的信息

3) 数据最好变为整型信息存储 (set enum 时间戳 ip)

14.3 逆范式

获得每个分类下商品总数量 (连表查询 Goods Category)

为了查询速度快，给 Category 维护一个商品总数量的字段，可以使得查询变为一条 sql 的执行

14.4 索引

(四种类型、创建和删除、执行计划、使用场合、索引原则)

索引就是把数据表的某个字段获得出来，该字段作为关键字与记录物理地址进行对应，以便快速定位记录信息 (索引内部有算法，可以快速定义信息)

四种类型：主键 (auto_increment)、唯一、普通、全文

14.4.1 创建：

```
create table 表名 (
    .....
    primary key (字段),
    unique index 索引名 (字段),
    index 索引名 (字段),
    fulltext 索引名 (字段)
)
alter table 表名 add primary key (id);
alter table 表名 add unique index 名称 (字段);
```

```
alter table 表名 add index 名称 (字段);
alter table 表名 add fulltext index 名称 (字段);
```

删除:

```
alter table 表名 drop primary key ;
alter table 表名 drop index 索引名 ; //唯一、普通、全文
```

执行计划: explain

针对查询语句起作用

查询语句在没有执行之前可以看下该语句消耗资源情况, 例如 sql 语句是否使用索引

```
explain 查询语句\G
```

索引使用场合:

where order by 索引覆盖 (复合索引) 连接查询

索引使用原则:

列独立 左原则 复合索引 OR 原则

14.4.2 索引设计依据

与数据表有关系的 sql 语句都统计出来

where order by or 等等条件的字段适当做索引

原则:

频率高的 sql 语句

执行时间长的 sql 语句

业务逻辑重要的 sql 语句

什么样子字段不适合做索引?

内容比较单调的字段不适合做索引

14.4.3 前缀索引

一个字段只取前边的几位内容做索引

好处: 索引空间比较少、运行速度快

前 n 位做索引, 前 n 位要具备唯一标识当前记录的特点

14.4.4 全文索引

Mysql5.5 只 Myisam 存储引擎可以实现

Mysql5.6 Myisam 和 Innodb 存储引擎都可以实现

```
fulltext index 索引名称 (字段, 字段)
```

```
select * from 表名 where 字段 like '%内容%' or 字段 like '%内容%';
```

```
select * from 表名 match(字段, 字段) against(“内容 1, 内容 2”);
```

```
match(字段, 字段) against(“内容 1, 内容 2”)
```

14.4.5 索引结构

Mysql 的索引结构是 B+Tree 结构

索引就是数据结构(自然有算法)，算法可以保证数据非常快速被找到

非聚集(Myisam)

叶子节点的关键字(索引字段内容) 与 记录的物理地址对应

聚集(Innodb)

主(键)索引：叶子节点的关键字 与 整条记录对应

非主(唯一/普通/全文)索引：叶子节点的关键字 与 主键关键字对应

14.5 查询缓存

开启缓存，开辟缓存空间(64MB)

缓存失效：表 或 数据 内容改变

不使用缓存：sql 语句有变化的信息，例如当前时间、随机数

同一个业务逻辑的多个 sql 语句，有不同结构(空格变化、大小写变哈)的变化，每个样子的 sql 语句会分别设置缓存

14.6 分区、分表设计

分区表算法(Mysql)：key hash range list
(php 代码不会发生变化)

分区增加或减少：

减少：hash/range/list 类型算法会丢失对应的数据

14.7 垂直分表

把一个数据表的多个字段进行拆分，分别分配到不同的数据表中

涉及的算法是 php 层面的

14.8 架构设计

主从模式 (读写分离、一主多从)

主服务器负责 “写” 数据，从服务器负责 “读” 数据

“主” 会 自动 给“从” 同步数据 (mysql 本身技术)

通过 “负载均衡” 可以平均地从 从服务器 获得数据

14.9 慢查询日志设置

```
show variables like 'slow_query_log%';
```

开启慢查询日志开关

设置时间阈值

15 Mysql 中常用语句

查看数据库引擎: `show engines`

查看数据库版本: `select version()`