设计和优化专题

mysql的最佳索引攻略

> 本设计和优化专题转自博客园的Mysql的设计和优化专题

Explain优化查询检测

所谓索引就是为特定的mysql字段进行一些特定的算法排序,比如二叉树的算法和哈希算法,哈希算法是通过建立特征值,然后根据特征值来快速查找,而用的最多,并且是mysql默认的就是二叉树算法 BTREE,通过BTREE算法建立索引的字段,比如扫描20行就能得到未使用BTREE前扫描了2^20行的结果,具体的实现方式后续本博客会出一个算法专题里面会有具体的分析讨论;

EXPLAIN可以帮助开发人员分析SQL问题,explain显示了mysql如何使用索引来处理select语句以及连接表,可以帮助选择更好的索引和写出更优化的查询语句.

使用方法,在select语句前加上Explain就可以了：

```
Explain select * from blog where false;
```

mysql在执行一条查询之前,会对发出的每条SQL进行分析,决定是否使用索引或全表扫描如果发送一条 `select * from blog where false` Mysql是不会执行查询操作的,因为经过SQL分析器的分析后MySQL已经清楚不会有任何语句符合操作;

Example

```
mysql> EXPLAIN SELECT `birday` FROM `user` WHERE `birthday` < "1990/2/2";
-- 结果：
id: 1

select_type: SIMPLE -- 查询类型（简单查询,联合查询,子查询）

table: user -- 显示这一行的数据是关于哪张表的

type: range -- 区间索引（在小于1990/2/2区间的数据),这是重要的列,显示连接使用了何种类型。从最好到最差的连接类型为system > const > eq_ref > ref > fulltext > ref_or_null > index_merge > unique_subquery > index_subquery > range > index > ALL,const代表一次就命中,ALL代表扫描了全表才确定结果。一般来说,得保证查询至少达到range级别,最好能达到ref。

possible_keys: birthday  -- 指出MySQL能使用哪个索引在该表中找到行。如果是空的,没有相关的索引。这时要提高性能,可通过检验WHERE子句,看是否引用某些字段,或者检查字段不是适合索引。

key: birthday -- 实际使用到的索引。如果为NULL,则没有使用索引。如果为primary的话,表示使用了主键。

key_len: 4 -- 最长的索引宽度。如果键是NULL,长度就是NULL。在不损失精确性的情况下,长度越短越好

ref: const -- 显示哪个字段或常数与key一起被使用。

rows: 1 -- 这个数表示mysql要遍历多少数据才能找到,在innodb上是不准确的。

Extra: Using where; Using index -- 执行状态说明,这里可以看到的坏的例子是Using temporary和Using
```

select_type

- simple 简单select(不使用union或子查询)
- primary 最外面的select
- union union中的第二个或后面的select语句
- dependent union union中的第二个或后面的select语句,取决于外面的查询
- union result union的结果。
- subquery 子查询中的第一个select
- dependent subquery 子查询中的第一个select,取决于外面的查询
- derived 导出表的select(from子句的子查询)

Extra与type详细说明

- Distinct:一旦MYSQL找到了与行相联合匹配的行,就不再搜索了
- Not exists: MYSQL优化了LEFT JOIN,一旦它找到了匹配LEFT JOIN标准的行,就不再搜索了
- Range checked for each Record（index map:#）:没有找到理想的索引,因此对于从前面表中来的每一个行组合,MYSQL检查使用哪个索引,并用它来从表中返回行。这是使用索引的最慢的连接之一
- Using filesort: **看到这个的时候,查询就需要优化了。**MYSQL需要进行额外的步骤来发现如何对返回的行排序。它根据连接类型以及存储排序键值和匹配条件的全部行的行指针来排序全部行
- Using index: 列数据是从仅仅使用了索引中的信息而没有读取实际的行动的表返回的,这发生在对表的全部的请求列都是同一个索引的部分的时候
- Using temporary **看到这个的时候,查询需要优化了。**这里,MYSQL需要创建一个临时表来存储结果,这通常发生在对不同的列集进行ORDER BY上,而不是GROUP BY上
- Where used 使用了WHERE从句来限制哪些行将与下一张表匹配或者是返回给用户。如果不想返回表中的全部行,并且连接类型ALL或index,这就会发生,或者是查询有问题不同连接类型的解释（按照效率高低的顺序排序
- system 表只有一行：system表。这是const连接类型的特殊情况
- const:表中的一个记录的最大值能够匹配这个查询（索引可以是主键或惟一索引）。因为只有一行,这个值实际就是常数,因为MYSQL先读这个值然后把它当做常数来对待
- eq_ref:在连接中,MYSQL在查询时,从前面的表中,对每一个记录的联合都从表中读取一个记录,它在查询使用了索引为主键或惟一键的全部时使用
- ref:这个连接类型只有在查询使用了不是惟一或主键的键或者是这些类型的部分（比如,利用最左边前缀）时发生。对于之前的表的每一个行联合,全部记录都将从表中读出。这个类型严重依赖于根据索引匹配的记录多少—越少越好+
- range:这个连接类型使用索引返回一个范围中的行,比如使用>或<查找东西时发生的情况+
- index: 这个连接类型对前面的表中的每一个记录联合进行完全扫描（比ALL更好,因为索引一般小于表数据）+
- ALL:这个连接类型对于前面的每一个记录联合进行完全扫描,这一般比较糟糕,应该尽量避免

其中type:

- 如果是Only index,这意味着信息只用索引树中的信息检索出的,这比扫描整个表要快。
- 如果是where used,就是使用上了where限制。
- 如果是impossible where 表示用不着where,一般就是没查出来啥。

- 如果此信息显示Using filesort或者Using temporary的话会很吃力,WHERE和ORDER BY的索引经常无法兼顾,如果按照WHERE来确定索引,那么在ORDER BY时,就必然会引起Using filesort,这就要看是先过滤再排序划算,还是先排序再过滤划算。

索引的类型

UNIQUE唯一索引

不可以出现相同的值,可以有NULL值

INDEX普通索引

允许出现相同的索引内容

PRIMARY KEY主键索引

不允许出现相同的值,且不能为NULL值,一个表只能有一个primary_key索引

fulltext index 全文索引

上述三种索引都是针对列的值发挥作用,但全文索引,可以针对值中的某个单词,比如一篇文章中的某个词,**然而并没有什么卵用,因为只有myisam以及英文支持,并且效率让人不敢恭维,但是可以用coreseek和xunsearch等第三方应用来完成这个需求**

索引的CURD

索引的创建

ALTER TABLE

适用于表创建完毕之后再添加

```
ALTER TABLE 表名 ADD 索引类型 （ unique,primary key,fulltext,index ）[索引名] ( 字段名 )
```

```
ALTER TABLE `table_name` ADD INDEX `index_name` (`column_list`) -- 索引名,可要可不要;如果不要,当前的索引名就是该字段名;
ALTER TABLE `table_name` ADD UNIQUE (`column_list`)
ALTER TABLE `table_name` ADD PRIMARY KEY (`column_list`)
ALTER TABLE `table_name` ADD FULLTEXT KEY (`column_list`)
```

CREATE INDEX

CREATE INDEX可对表增加普通索引或UNIQUE索引

```
--例,只能添加这两种索引;
CREATE INDEX index_name ON table_name (column_list)
CREATE UNIQUE INDEX index_name ON table_name (column_list)
```

另外,还可以在建表时添加

```
CREATE TABLE `test1` (
  `id` smallint(5) UNSIGNED AUTO_INCREMENT NOT NULL, -- 注意,下面创建了主键索引,这里就不用创建了
  `username` varchar(64) NOT NULL COMMENT '用户名',
  `nickname` varchar(50) NOT NULL COMMENT '昵称/姓名',
  `intro` text,
  PRIMARY KEY (`id`),
  UNIQUE KEY `unique1` (`username`), -- 索引名称,可要可不要,不要就是和列名一样
  KEY `index1` (`nickname`),
  FULLTEXT KEY `intro` (`intro`)
) ENGINE=MyISAM AUTO_INCREMENT=4 DEFAULT CHARSET=utf8 COMMENT='后台用户表';
```

索引的删除

```
DROP INDEX `index_name` ON `talbe_name`
ALTER TABLE `table_name` DROP INDEX `index_name`
-- 这两句都是等价的,都是删除掉table_name中的索引index_name;

ALTER TABLE `table_name` DROP PRIMARY KEY -- 删除主键索引,注意主键索引只能用这种方式删除
```

索引的查看

```
show index from tablename \G;
```

索引的更改

更改个毛线,删掉重建一个既可

创建索引的技巧

1.维度高的列创建索引

数据列中**不重复值**出现的个数,这个数量越高,维度就越高

如数据表中存在8行数据a ,b ,c,d,a,b,c,d这个表的维度为4

要为维度高的列创建索引,如性别和年龄,那年龄的维度就高于性别

性别这样的列不适合创建索引,因为维度过低

2.对 **where,on,group by,order by** 中出现的列使用索引

3.对较小的数据列使用索引,这样会使索引文件更小,同时内存中也可以装载更多的索引键

4.为较长的字符串使用前缀索引

5.不要过多创建索引,除了增加额外的磁盘空间外,对于DML操作的速度影响很大,因为其每增删改一次就得

从新建立索引

6.使用组合索引,可以减少文件索引大小,在使用时速度要优于多个单列索引

组合索引与前缀索引
注意,这两种称呼是对建立索引技巧的一种称呼,并非索引的类型;

组合索引
MySQL单列索引和组合索引究竟有何区别呢？

为了形象地对比两者,先建一个表：

```
CREATE TABLE `myIndex` (
  `i_testID` INT NOT NULL AUTO_INCREMENT,
  `vc_Name` VARCHAR(50) NOT NULL,
  `vc_City` VARCHAR(50) NOT NULL,
  `i_Age` INT NOT NULL,
  `i_SchoolID` INT NOT NULL,
  PRIMARY KEY (`i_testID`)
);
```

假设表内已有1000条数据,在这 10000 条记录里面 7 上 8 下地分布了 5 条 vc_Name="erquan" 的记录,只不过 city,age,school 的组合各不相同。
来看这条 T-SQL：

```
SELECT `i_testID` FROM `myIndex` WHERE `vc_Name`='erquan' AND `vc_City`='郑州' AND `i_Age`=25; -- 关联搜索;
```

首先考虑建MySQL单列索引：

在 vc_Name 列上建立了索引。执行 T-SQL 时,MYSQL 很快将目标锁定在了 vc_Name=erquan 的 5 条记录上,取出来放到一中间结果集。在这个结果集里,先排除掉 vc_City 不等于"郑州"的记录,再排除 i_Age 不等于 25 的记录,最后筛选出唯一的符合条件的记录。
虽然在 vc_Name 上建立了索引,查询时MYSQL不用扫描整张表,效率有所提高,但离我们的要求还有一定的距离。同样的,在 vc_City 和 i_Age 分别建立的MySQL单列索引的效率相似。

为了进一步榨取 MySQL 的效率,就要考虑建立组合索引。就是将 vc_Name,vc_City,i_Age 建到一个索引里：

```
ALTER TABLE `myIndex` ADD INDEX `name_city_age` (vc_Name(10),vc_City,i_Age);
```

建表时,vc_Name 长度为 50,这里为什么用 10 呢？这就是下文要说到的前缀索引,因为一般情况下名字的长度不会超过 10,这样会加速索引查询速度,还会减少索引文件的大小,提高 INSERT 的更新速度。

执行 T-SQL 时,MySQL 无须扫描任何记录就到找到唯一的记录！！

如果分别在 vc_Name,vc_City,i_Age 上建立单列索引,让该表有 3 个单列索引,查询时和上述的组合索引效率一样吗？答案是大不一样,远远低于我们的组合索引。虽然此时有了三个索引,**但 MySQL 只能用到其中的那个它认为似乎是最有效率的单列索引,另外两个是用不到的,也就是说还是一个全表扫描的过程。**
建立这样的组合索引,其实是相当于分别建立了

```
vc_Name,vc_City,i_Age
vc_Name,vc_City
vc_Name
```

这样的三个组合索引！为什么没有 vc_City,i_Age 等这样的组合索引呢？这是因为 mysql 组合索引**"最左前缀"**的结果。简单的理解就是只从最左面的开始组合。并不是只要包含这三列的查询都会用到该组合索引,下面的几个 T-SQL 会用到：

```
SELECT * FROM myIndex WHREE vc_Name="erquan" AND vc_City="郑州"
SELECT * FROM myIndex WHREE vc_Name="erquan"
```

而下面几个则不会用到：

```
SELECT * FROM myIndex WHREE i_Age=20 AND vc_City="郑州"
SELECT * FROM myIndex WHREE vc_City="郑州"
```

也就是,`name_city_age` (vc_Name(10),vc_City,i_Age) 从左到右进行索引,如果没有左前索引Mysql不执行索引查询

前缀索引

如果索引列长度过长,这种列索引时将会产生很大的索引文件,不便于操作,可以使用前缀索引方式进行索引
前缀索引应该控制在一个合适的点,控制在0.31黄金值即可(大于这个值就可以创建)

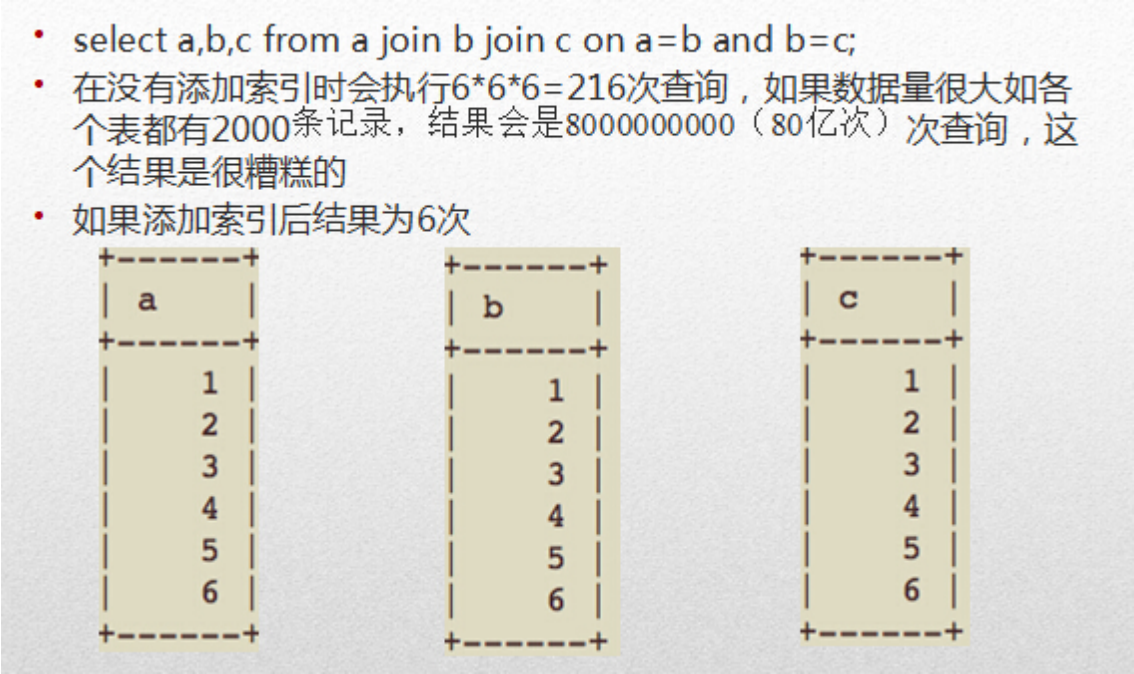
```
SELECT COUNT(DISTINCT(LEFT(`title`,10)))/COUNT(*) FROM Arctic; -- 这个值大于0.31就可以创建前缀索引,Distinct去重复

ALTER TABLE `user` ADD INDEX `uname`(title(10)); -- 增加前缀索引SQL,将人名的索引建立在10,这样可以减少索引文件大小,加快索引查询速度
```

什么样的sql不走索引

要尽量避免这些不走索引的sql

```
SELECT `sname` FROM `stu` WHERE `age`+10=30;-- 不会使用索引,因为所有索引列参与了计算

SELECT `sname` FROM `stu` WHERE LEFT(`date`,4) <1990; -- 不会使用索引,因为使用了函数运算,原理与上面相同

SELECT * FROM `houdunwang` WHERE `uname` LIKE'后盾%' -- 走索引

SELECT * FROM `houdunwang` WHERE `uname` LIKE "%后盾%" -- 不走索引

-- 正则表达式不使用索引,这应该很好理解,所以为什么在SQL中很难看到regexp关键字的原因

-- 字符串与数字比较不使用索引;
CREATE TABLE `a` (`a` char(10));
EXPLAIN SELECT * FROM `a` WHERE `a`="1" -- 走索引
EXPLAIN SELECT * FROM `a` WHERE `a`=1 -- 不走索引

select * from dept where dname='xxx' or loc='xx' or deptno=45 --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

-- 如果mysql估计使用全表扫描要比使用索引快,则不使用索引
```

多表关联时的索引效率



从上图可以看出,所有表的type为all,表示全表索引;也就是6_6_6,共遍历查询了216次;

除第一张表示全表索引(必须的,要以此关联其他表),其余的为range(索引区间获得),也就是6+1+1+1,共遍历查询9次即可;

所以我们建议在多表join的时候尽量少join几张表,因为一不小心就是一个笛卡尔乘积的恐怖扫描,另外,我们还建议尽量使用left join,以少关联多.因为使用join 的话,第一张表是必须的全扫描的,以少关联多就可以减

少这个扫描次数.

索引的弊端

不要盲目的创建索引,只为查询操作频繁的列创建索引,创建索引会使查询操作变得更加快速,但是会降低增加、删除、更新操作的速度,因为执行这些操作的同时会对索引文件进行重新排序或更新;

但是,在互联网应用中,查询的语句远远大于DML的语句,甚至可以占到80%~90%,所以也不要太在意,只是在大数据导入时,可以先删除索引,再批量插入数据,最后再添加索引;

详解慢查询

查询mysql的操作信息

```
show status -- 显示全部mysql操作信息

show status like "com_insert%"; -- 获得mysql的插入次数;

show status like "com_delete%"; -- 获得mysql的删除次数;

show status like "com_select%"; -- 获得mysql的查询次数;

show status like "uptime"; -- 获得mysql服务器运行时间

show status like 'connections'; -- 获得mysql连接次数
```

show [session|global] status like .... 如果你不写 [session|global] 默认是session 会话，只取出当前窗口的执行，如果你想看所有(从mysql 启动到现在，则应该 global)

> 通过查询mysql的读写比例,可以做相应的配置优化;

慢查询

当Mysql性能下降时，通过开启慢查询来获得哪条SQL语句造成的响应过慢，进行分析处理。**当然开启慢查询会带来CPU损耗与日志记录的IO开销，所以我们要间断性的打开慢查询日志来查看Mysql运行状态。**

慢查询能记录下所有执行超过 `long_query_time` 时间的SQL语句, 用于找到执行慢的SQL, 方便我们对这些SQL进行优化.

```
show variables like "%slow%";-- 是否开启慢查询;
show status like "%slow%"; -- 查询慢查询SQL状况;
show variables like "long_query_time"; -- 慢查询时间
```

慢查询开启设置

```
mysql> show variables like 'long_query_time'; -- 默认情况下，mysql认为10秒才是一个慢查询
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+

mysql> set long_query_time=1; -- 修改慢查询时间,只能当前会话有效;
mysql> set global slow_query_log='ON';-- 启用慢查询 ,加上global，不然会报错的;
```

也可以在配置文件中更改

修改mysql配置文件my.ini[windows]/my.cnf[Linux]**加入,注意必须在 [mysqld] 后面加入**

```
slow_query_log = on -- 开启日志;
slow_query_log_file = /data/f/mysql_slow_cw.log -- 记录日志的log文件; 注意:window上必须写绝对路径,比如 D:/wamp/bin/mysql/mysql5.5.16/data/show-slow.log
long_query_time = 2 -- 最长查询的秒数;
log-queries-not-using-indexes -- 表示记录没有使用索引的查询
```

使用慢查询

Example1:

```
mysql> select sleep(3);

mysql> show status like '%slow%';
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| Slow_launch_threads | 0     |
| Slow_queries        | 1     |
+---------------------+-------+
-- Slow_queries 一共有一条慢查询
```

Example2:

利用存储过程构建一个大的数据库来进行测试;

数据准备

```
CREATE TABLE dept(
deptno MEDIUMINT   UNSIGNED  NOT NULL  DEFAULT 0 comment '编号',
dname VARCHAR(20)  NOT NULL  DEFAULT "" comment '名称',
loc VARCHAR(13) NOT NULL DEFAULT "" comment '地点'
) ENGINE=MyISAM DEFAULT CHARSET=utf8 comment '部门表' ;

CREATE TABLE emp
(empno  MEDIUMINT UNSIGNED  NOT NULL  DEFAULT 0,
ename VARCHAR(20) NOT NULL DEFAULT "" comment '名字',
job VARCHAR(9) NOT NULL DEFAULT "" comment '工作',
mgr MEDIUMINT UNSIGNED NOT NULL DEFAULT 0 comment '上级编号',
hiredate DATE NOT NULL comment '入职时间',
sal DECIMAL(7,2)  NOT NULL comment '薪水',
comm DECIMAL(7,2) NOT NULL comment '红利',
deptno MEDIUMINT UNSIGNED NOT NULL DEFAULT 0 comment '部门编号'
)ENGINE=MyISAM DEFAULT CHARSET=utf8 comment '雇员表';
```

```
CREATE TABLE salgrade(
grade MEDIUMINT UNSIGNED NOT NULL DEFAULT 0 comment '等级',
losal DECIMAL(17,2)  NOT NULL comment '最低工资',
hisal DECIMAL(17,2)  NOT NULL comment '最高工资'
)ENGINE=MyISAM DEFAULT CHARSET=utf8 comment '工资级别表';

INSERT INTO salgrade VALUES (1,700,1200);
INSERT INTO salgrade VALUES (2,1201,1400);
INSERT INTO salgrade VALUES (3,1401,2000);
INSERT INTO salgrade VALUES (4,2001,3000);
INSERT INTO salgrade VALUES (5,3001,9999);

delimiter $
create function rand_num()
returns tinyint(6) READS SQL DATA
begin
  declare return_num tinyint(6) default 0;
  set return_num = floor(1+rand()*30);
  return return_num;
end $

delimiter $
create function rand_string(n INT)
returns varchar(255) READS SQL DATA
begin
  declare chars_str varchar(100) default
  'abcdefghijklmnopqrstuvwxyzABCDEFJHIJKLMNOPQRSTUVWXYZ';
  declare return_str varchar(255) default '';
  declare i int default 0;
  while i < n do
    set return_str =concat(return_str,substring(chars_str,floor(1+rand()*52),1));
    set i = i + 1;
  end while;
  return return_str;
end $

delimiter $
create procedure insert_emp(in start int(10),in max_num int(10))
begin
  declare i int default 0;
  #set autocommit =0 把autocommit设置成0,关闭自动提交;
  set autocommit = 0;
  repeat
    set i = i + 1;
    insert into emp values ((start+i) ,rand_string(6),'SALESMAN',0001,curdate(),2000,400,rand_n
um());
    until i = max_num
  end repeat;
  commit;
```

```
  commit;
end $

call insert_emp(1,4000000);
```

```
SELECT * FROM `emp` where ename like '%mQspyv%'; -- 1.163s

# Time: 150530 15:30:58  -- 该查询发生在2015-5-30 15:30:58
# User@Host: root[root] @ localhost [127.0.0.1] -- 是谁,在什么主机上发生的查询
# Query_time: 1.134065  Lock_time: 0.000000 Rows_sent: 8  Rows_examined: 4000000
-- Query_time: 查询总共用了多少时间,Lock_time: 在查询时锁定表的时间,Rows_sent: 返回多少rows数据,Rows_examined: 表扫描了400W行数据才得到的结果;
SET timestamp=1432971058; -- 发生慢查询时的时间戳;
SELECT * FROM `emp` where ename like '%mQspyv%';
```

开启慢查询后每天都有可能有好几G的慢查询日志,这个时候去人工的分析明显是不实际的;

慢查询分析工具:

mysqldumpslow

该工具是慢查询自带的分析慢查询工具，一般只要安装了mysql，就会有该工具；

```
Usage: mysqldumpslow [ OPTS... ] [ LOGS... ]  -- 后跟参数以及log文件的绝对地址;

  -s            what to sort by (al, at, ar, c, l, r, t), 'at' is default
                al: average lock time
                ar: average rows sent
                at: average query time
                c: count
                l: lock time
                r: rows sent
                t: query time

  -r           reverse the sort order (largest last instead of first)
  -t NUM       just show the top n queries
  -a           don't abstract all numbers to N and strings to 'S'
  -n NUM       abstract numbers with at least n digits within names
  -g PATTERN   grep: only consider stmts that include this string
  -h HOSTNAME  hostname of db server for *-slow.log filename (can be wildcard),
               default is '*', i.e. match all
  -i NAME      name of server instance (if using mysql.server startup script)
  -l           don't subtract lock time from total time
```

常见用法

```
mysqldumpslow -s c -t 10 /var/run/mysqld/mysqld-slow.log # 取出使用最多的10条慢查询

mysqldumpslow -s t -t 3 /var/run/mysqld/mysqld-slow.log # 取出查询时间最慢的3条慢查询

mysqldumpslow -s t -t 10 -g “left join” /database/mysql/slow-log # 得到按照时间排序的前10条里面含有左连接的查询语句

 mysqldumpslow -s r -t 10 -g 'left join' /var/run/mysqld/mysqld-slow.log # 按照扫描行数最多的
```

> 注意: 使用mysqldumpslow的分析结果不会显示具体完整的sql语句,只会显示sql的组成结构;

假如: SELECT * FROM sms_send WHERE service_id=10 GROUP BY content LIMIT 0, 1000;
mysqldumpslow来显示

```
Count: 1  Time=1.91s (1s)  Lock=0.00s (0s)  Rows=1000.0 (1000), vgos_dba[vgos_dba]@[10.130.229.196]
SELECT * FROM sms_send WHERE service_id=N GROUP BY content LIMIT N, N;
```

pt-query-digest

说明

pt-query-digest是用于分析mysql慢查询的一个工具，它可以分析binlog、General log、slowlog，也可以通过SHOWPROCESSLIST或者通过tcpdump抓取的MySQL协议数据来进行分析。可以把分析结果输出到文件中，分析过程是先对查询语句的条件进行参数化，然后对参数化以后的查询进行分组统计，统计出各查询的执行时间、次数、占比等，可以借助分析结果找出问题进行优化。
pt-query-digest是一个perl脚本，只需下载并赋权即可执行。

安装

```
wget http://www.percona.com/get/pt-query-digest
chmod +x pt-query-digest
# 注意这是一个Linux脚本,要指明绝对或相对路径来使用

--或者下载整套工具

wget percona.com/get/percona-toolkit.rpm
rpm -ivh percona-toolkit-2.2.13-1.noarch.rpm

wget percona.com/get/percona-toolkit.tar.gz
tar -zxvf percona-toolkit-2.2.13.tar.gz
cd percona-toolkit-2.2.13
perl Makefile.PL
make && make install
```

语法及重要选项

pt-query-digest [OPTIONS] [FILES] [DSN]

--create-review-table 当使用--review参数把分析结果输出到表中时，如果没有表就自动创建。
--create-history-table 当使用--history参数把分析结果输出到表中时，如果没有表就自动创建。
--filter 对输入的慢查询按指定的字符串进行匹配过滤后再进行分析
--limit限制输出结果百分比或数量，默认值是20,即将最慢的20条语句输出，如果是50%则按总响应时间占比从大到小排序，输出到总和达到50%位置截止。
--host mysql服务器地址
--user mysql用户名
--password mysql用户密码
--history 将分析结果保存到表中，分析结果比较详细，下次再使用--history时，如果存在相同的语句，且查询所在的时间区间和历史表中的不同，则会记录到数据表中，可以通过查询同一CHECKSUM来比较某类型查询的历史变化。
--review 将分析结果保存到表中，这个分析只是对查询条件进行参数化，一个类型的查询一条记录，比较简单。当下次使用--review时，如果存在相同的语句分析，就不会记录到数据表中。
--output 分析结果输出类型，值可以是report(标准分析报告)、slowlog(Mysql slow log)、json、json-anon，一般使用report，以便于阅读。
--since 从什么时间开始分析，值为字符串，可以是指定的某个”yyyy-mm-dd [hh:mm:ss]”格式的时间点，也可以是简单的一个时间值：s(秒)、h(小时)、m(分钟)、d(天)，如12h就表示从12小时前开始统计。
--until 截止时间，配合—since可以分析一段时间内的慢查询。

第一部分：总体统计结果:

标准分析报告解释

```
# 200ms user time, 20ms system time, 14.83M rss, 24.04M vsz
# Current date: Sat May 30 17:46:10 2015
# Hostname: semple
# Files: /var/run/mysqld/mysqld-slow.log
# Overall: 6 total, 4 unique, 0.01 QPS, 0.09x concurrency _________________
# Time range: 2015-05-30 16:15:39 to 16:23:58
# Attribute          total     min     max     avg     95%  stddev  median
# ============     ======= ======= ======= ======= ======= ======= =======
# Exec time            45s      1s     38s      8s     37s     13s      2s
# Lock time          553us    63us   179us    92us   176us    39us    82us
# Rows sent        910.77k       0 910.74k 151.79k 871.90k 324.93k   10.84
# Rows examine      38.15M       7  26.70M   6.36M  25.91M   8.98M   3.68M
# Query size           559      47     136   93.17  130.47   34.90  121.58
```

Overall: 总共有多少条查询，上例为总共266个查询。

Time range: 查询执行的时间范围。

unique: 唯一查询数量，即对查询条件进行参数化以后，总共有多少个不同的查询，该例为4。

total: 总计 min:最小 max: 最大 avg:平均

95%: 把所有值从小到大排列，位置位于95%的那个数，这个数一般最具有参考价值。

median: 中位数，把所有值从小到大排列，位置位于中间那个数。

第二部分: 查询分组统计结果:

```
# Profile
# Rank Query ID           Response time Calls R/Call  V/M   Item
# ==== ================== ============= ===== ======= ===== ==============
#    1 0xD44CBA69E70299D2 38.2327 84.4%     1 38.2327  0.00 SELECT dept emp
#    2 0xC80DFDD0ECFF51EC  3.1713  7.0%     2  1.5856  0.00 SELECT emp
#    3 0x9C8070F604A66CAE  2.3101  5.1%     2  1.1551  0.00 SELECT dept emp
# MISC 0xMISC              1.6026  3.5%     1  1.6026   0.0 <1 ITEMS>
```

这部分对查询进行参数化并分组，然后对各类查询的执行情况进行分析，结果按总执行时长，从大到小排序。

Response: 总的响应时间。

time: 该查询在本次分析中总的时间占比。

calls: 执行次数，即本次分析总共有多少条这种类型的查询语句。

R/Call: 平均每次执行的响应时间。

Item : 查询对象

第三部分：每一种查询的详细统计结果:

```
# Query 1: 0 QPS, 0x concurrency, ID 0xD44CBA69E70299D2 at byte 891 ______
# This item is included in the report because it matches --limit.
# Scores: V/M = 0.00
# Time range: all events occurred at 2015-05-30 16:22:03
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count         16       1
# Exec time     84     38s     38s     38s     38s     38s       0     38s
# Lock time     16    91us    91us    91us    91us    91us       0    91us
# Rows sent     99 910.74k 910.74k 910.74k 910.74k 910.74k       0 910.74k
# Rows examine  69  26.70M  26.70M  26.70M  26.70M  26.70M       0  26.70M
# Query size    20     113     113     113     113     113       0     113
# String:
# Databases    test
# Hosts        192.168.42.233
# Users        root
# Query_time distribution
#   1us
#  10us
# 100us
#   1ms
#  10ms
# 100ms
#    1s
#  10s+  ################################################################
# Tables
#    SHOW TABLE STATUS FROM `test` LIKE 'dept'\G
#    SHOW CREATE TABLE `test`.`dept`\G
#    SHOW TABLE STATUS FROM `test` LIKE 'dept'\G
#    SHOW CREATE TABLE `test`.`dept`\G
#    SHOW TABLE STATUS FROM `test` LIKE 'emp'\G
#    SHOW CREATE TABLE `test`.`emp`\G
# EXPLAIN /*!50100 PARTITIONS*/
select * from dept as d LEFT JOIN emp as e on d.deptno = e.deptno\G
```

由上图可见,1号查询的详细统计结果，最上面的表格列出了执行次数、最大、最小、平均、95%等各项目的统计。

Databases: 库名

Users: 各个用户执行的次数（占比）
Query_time distribution : 查询时间分布, 长短体现区间占比，本例中1s-10s之间查询数量没有,全部集中在10S里面。
Tables: 查询中涉及到的表
Explain: 该条查询的示例

用法示例
(1)直接分析慢查询文件:

```
pt-query-digest  slow.log > slow_report.log
```

(2)分析最近12小时内的查询：

```
pt-query-digest  --since=12h  slow.log > slow_report2.log
```

(3)分析指定时间范围内的查询：

```
pt-query-digest slow.log --since '2014-05-17 09:30:00' --until '2014-06-17 10:00:00'> > slow_report3.log
```

(4)分析只含有select语句的慢查询

```
pt-query-digest --filter '$event->{fingerprint} =~ m/^select/i' slow.log> slow_report4.log
```

(5) 针对某个用户的慢查询

```
pt-query-digest --filter '($event->{user} || "") =~ m/^root/i' slow.log> slow_report5.log
```

(6) 查询所有所有的全表扫描或full join的慢查询

```
pt-query-digest --filter '(($event->{Full_scan} || "") eq "yes") ||(($event->{Full_join} || "") eq "yes")' slow.log> slow_report6.log
```

(7)把查询保存到test数据库的query_review表,如果没有的话会自动创建;

```
pt-query-digest  --user=root –password=abc123 --review  h=localhost,D=test,t=query_review  --create-review-table  slow.log
```

(8)把查询保存到query_history表

```
pt-query-digest  --user=root –password=abc123 --review  h=localhost,D=test,t=query_ history --create-review-table  slow.log_20140401
```

(9)通过tcpdump抓取mysql的tcp协议数据，然后再分析

```
tcpdump -s 65535 -x -nn -q -tttt -i any -c 1000 port 3306 > mysql.tcp.txt
pt-query-digest --type tcpdump mysql.tcp.txt> slow_report9.log
```

(10)分析binlog

```
mysqlbinlog mysql-bin.000093 > mysql-bin000093.sql
pt-query-digest  --type=binlog  mysql-bin000093.sql > slow_report10.log
```

(11)分析general log

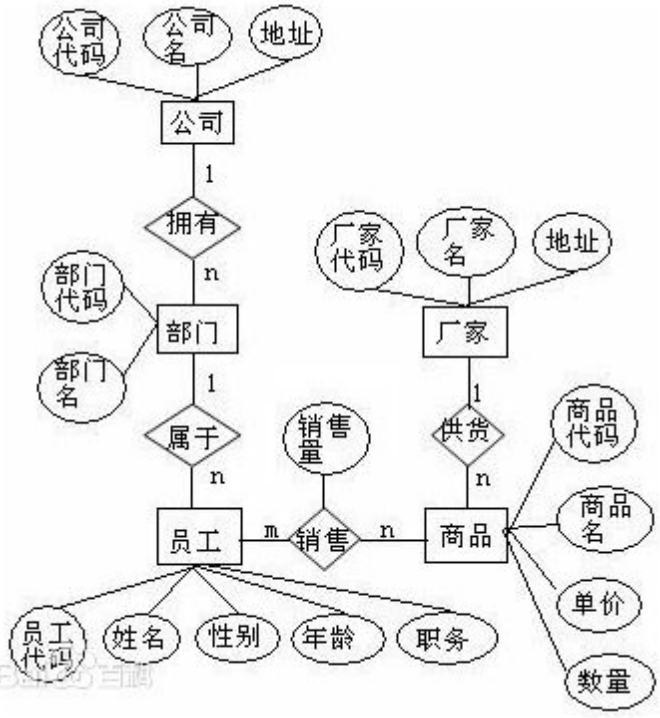
```
pt-query-digest  --type=genlog  localhost.log > slow_report11.log
```

> 另外,还有一款 Query-digest-UI 监控慢可视化查询应用，后续再玩；

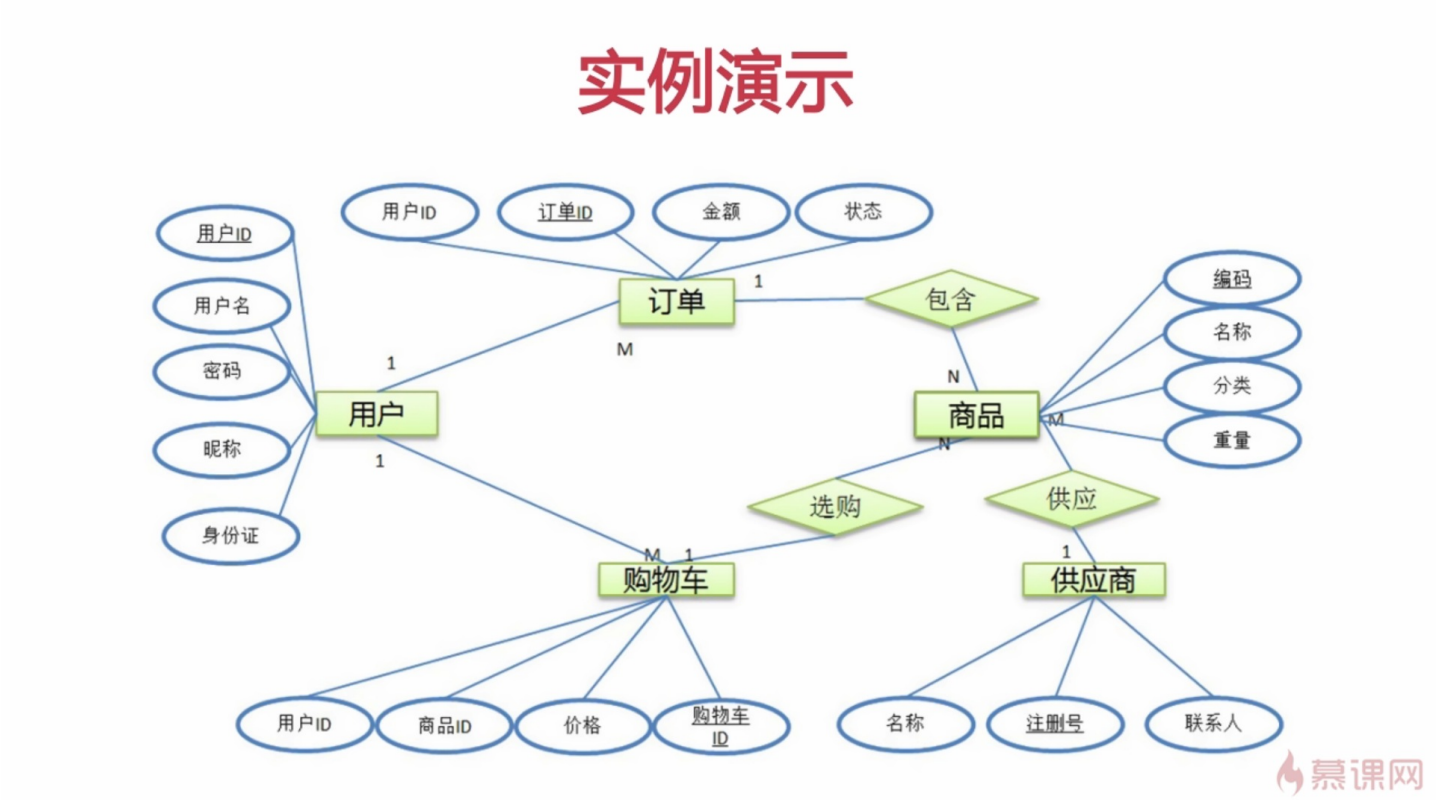
ER图，数据建模与数据字典

> 需求分析是做项目中的极为重要的一环,而作为整个项目中的'血液'--数据,更是重中之重。
> viso，workbench，phpmyadmin等软件可以帮我们更好的处理数据分析问题。

ER图



E-R方法是“实体-联系方法”（Entity-Relationship Approach）的简称。它是描述现实世界概念结构模型的有效方法。是表示概念模型的一种方式，**用矩形表示实体型，矩形框内写明实体名；用椭圆表示实体的属性，并用无向边将其与相应的实体型连接起来,属性如果有下划线的话,就表示该属性为主键属性；用菱形表示实体型之间的联系，在菱形框内写明联系名(实体和实体之间的关系)，并用无向边分别与有关实体型连接起来，同时在无向边旁标上联系的类型（1:1,1:n或m:n）**



实体之间联系

联系可分为以下 3 种类型：

(1) 一对一联系(1 ：1)

例如，一个部门有一个经理，而每个经理只在一个部门任职，则部门与经理的联系是一对一的。

(2) 一对多联系(1 ：N)

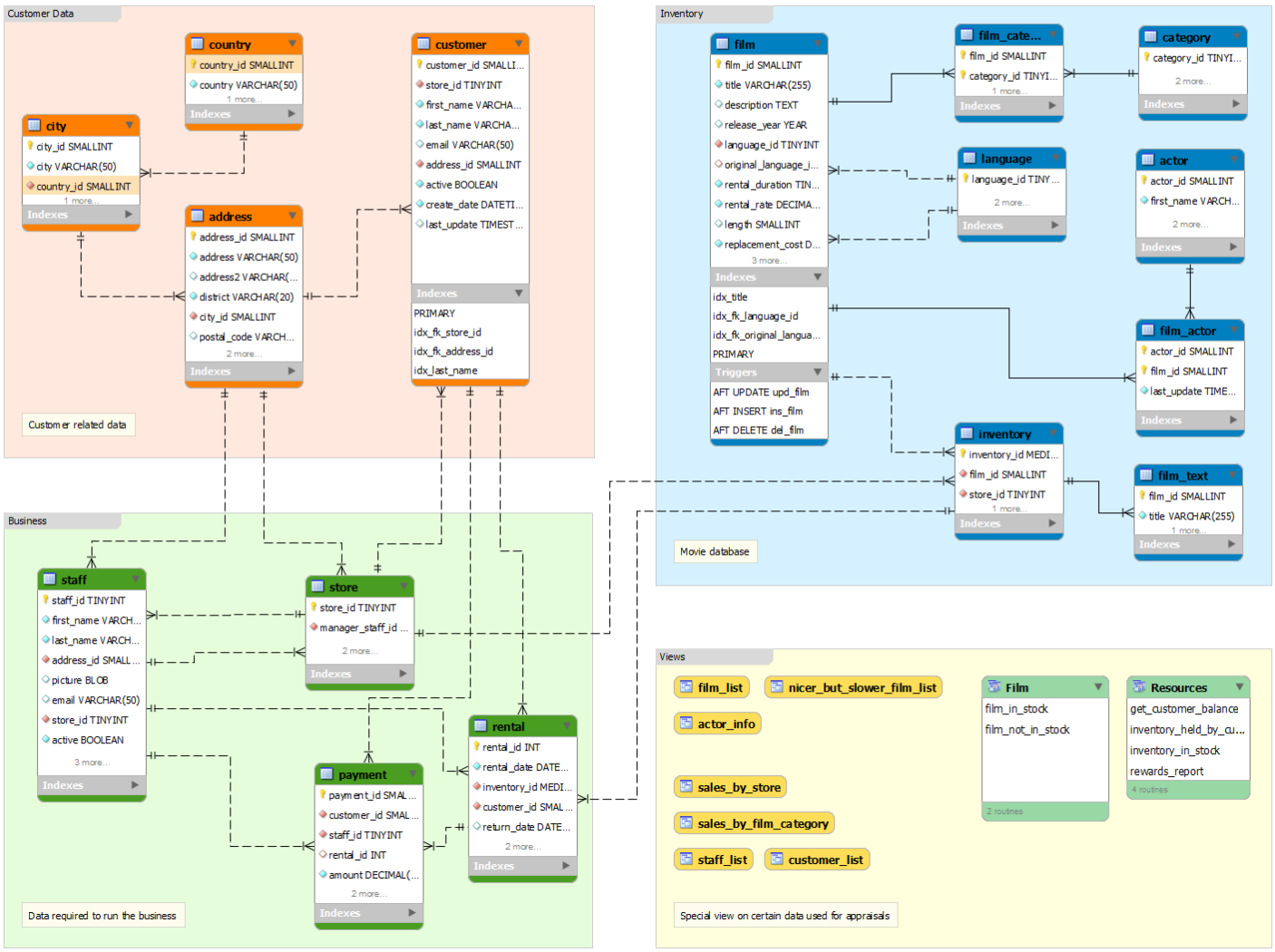
例如，某校教师与课程之间存在一对多的联系“教”，即每位教师可以教多门课程，但是每门课程只能由一位教师来教

(3) 多对多联系(M ：N)

例如，图1表示学生与课程间的联系(“学”)是多对多的，即一个学生可以学多门课程，而每门课程可以有多个学生来学。联系也可能有属性。例如，学生“学” 某门课程所取得的成绩，既不是学生的属性也不是课程的属性。由于“成绩” 既依赖于某名特定的学生又依赖于某门特定的课程，所以它是学生与课程之间的联系“学”的属性。

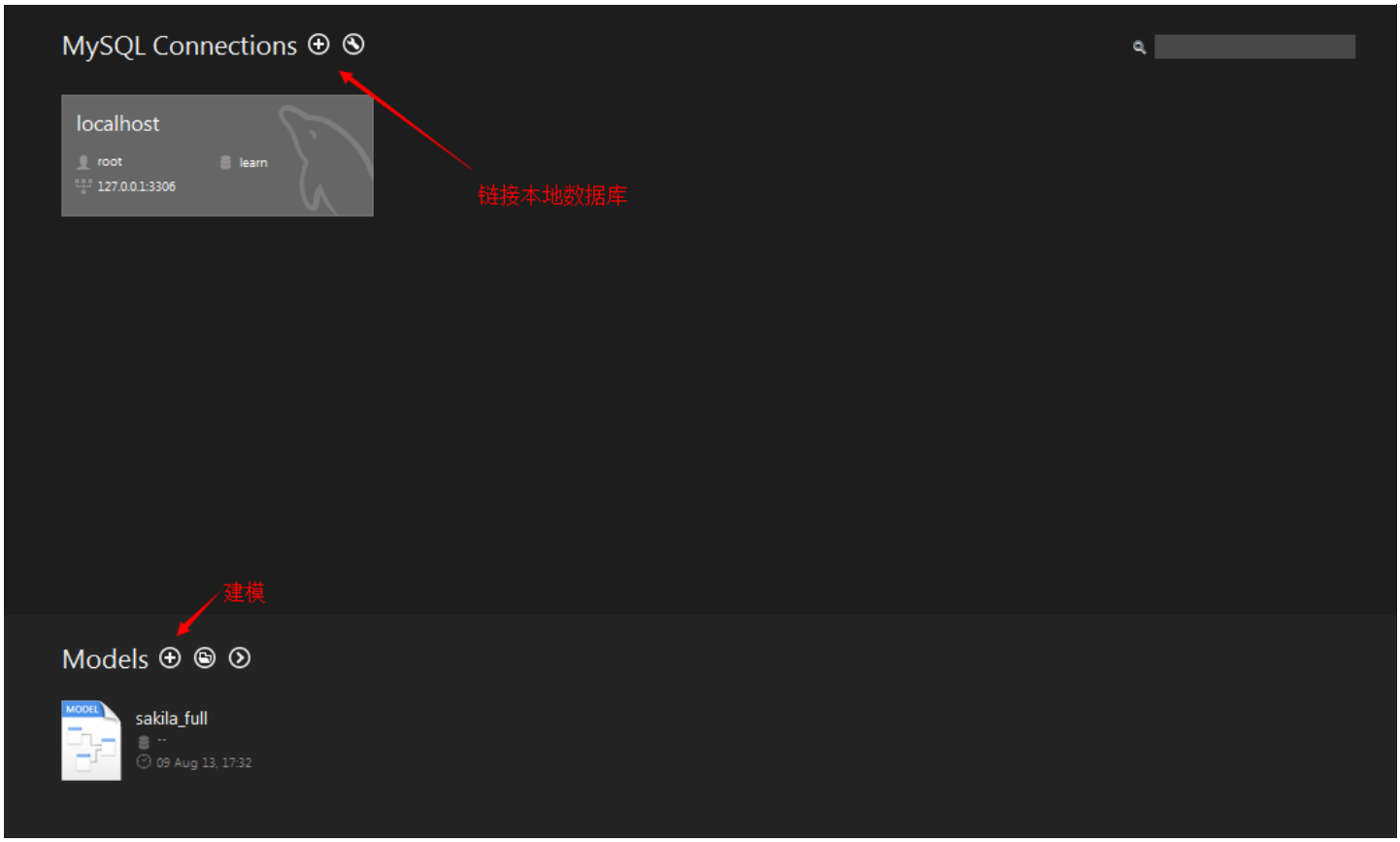
> 推荐使 **亿图图示专家或viso** 来画ER图

数据建模

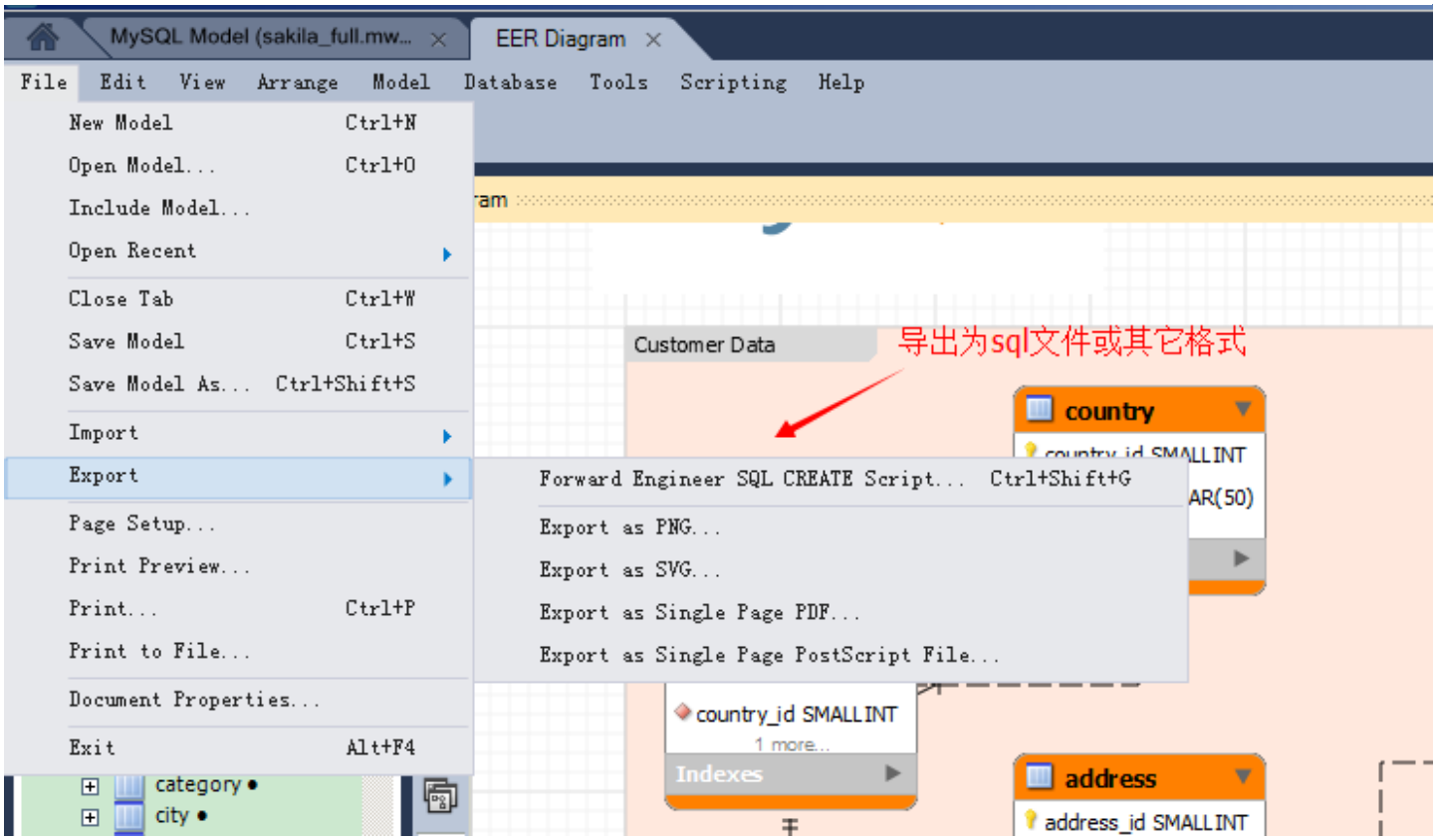


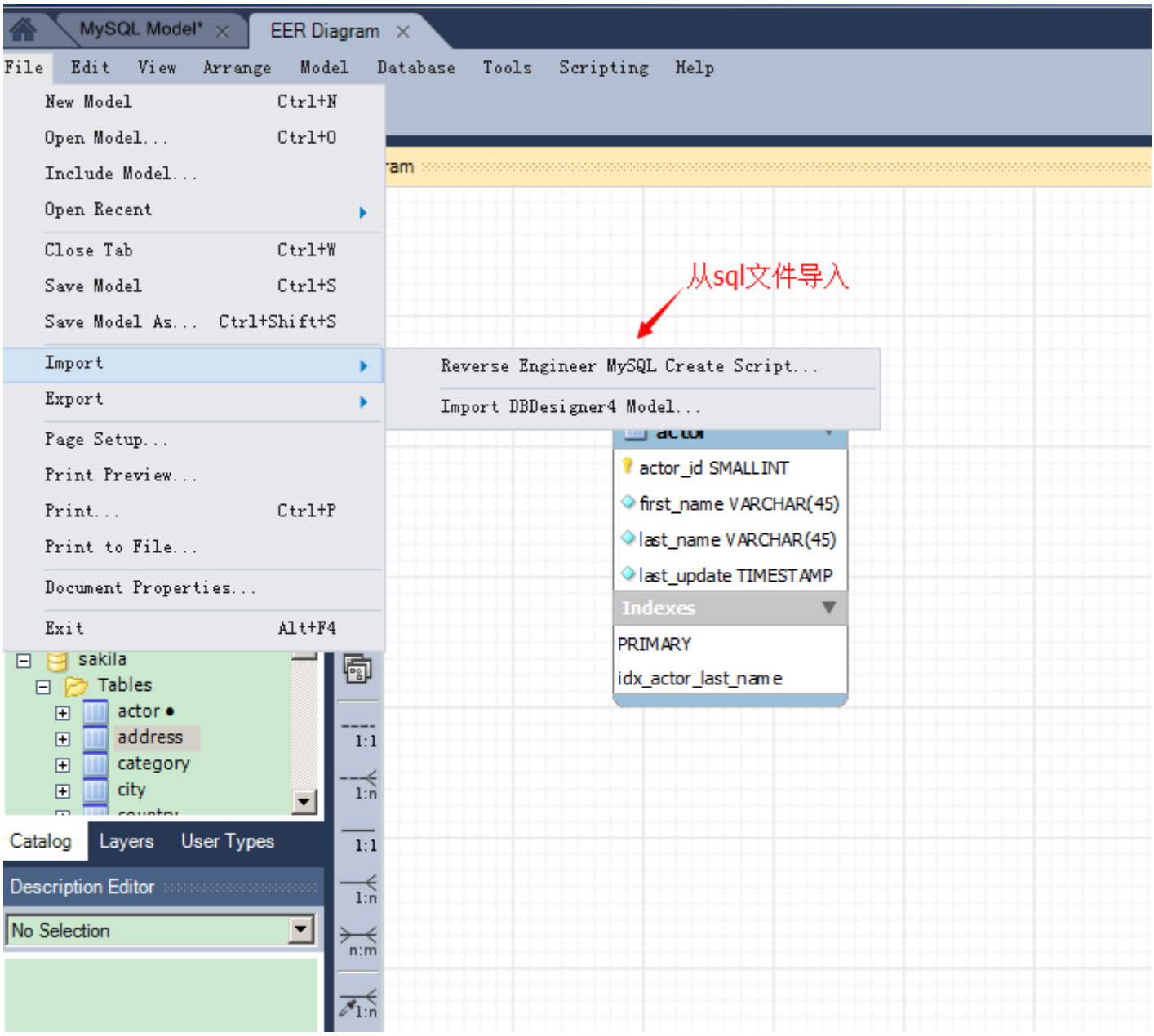
使用workbench软件可以很方便的建立数据模型,当然workbench不仅仅可以用来建模,还可以用来管理数据库.但通常我们只用来建模,管理数据库用navcate等更为方便的工具;
软件很简单,只不过是英文版本的,貌似市面上还没有出现中文版的,其实软件能用英文版的尽量使用英文版的

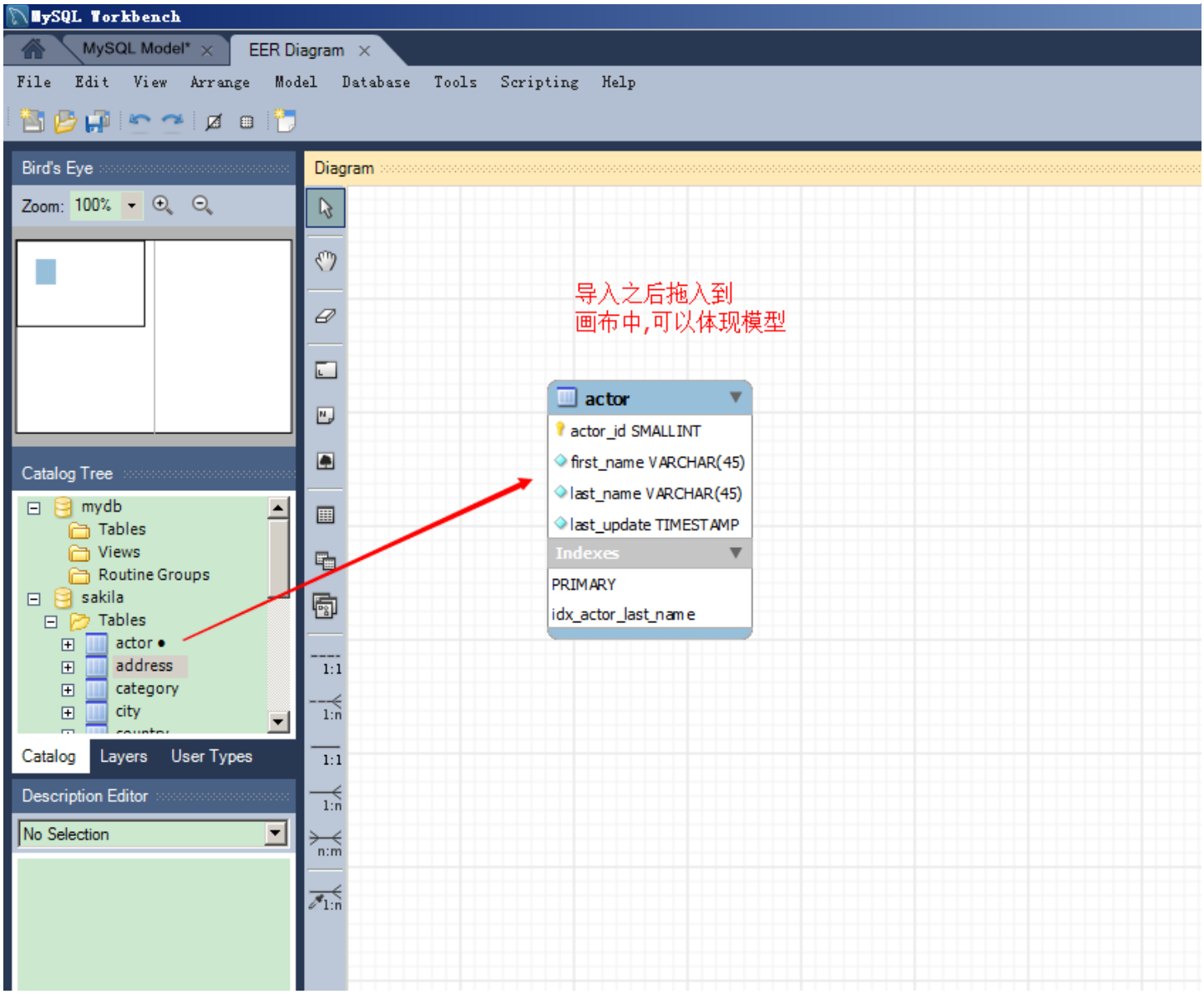
简单的图示使用说明

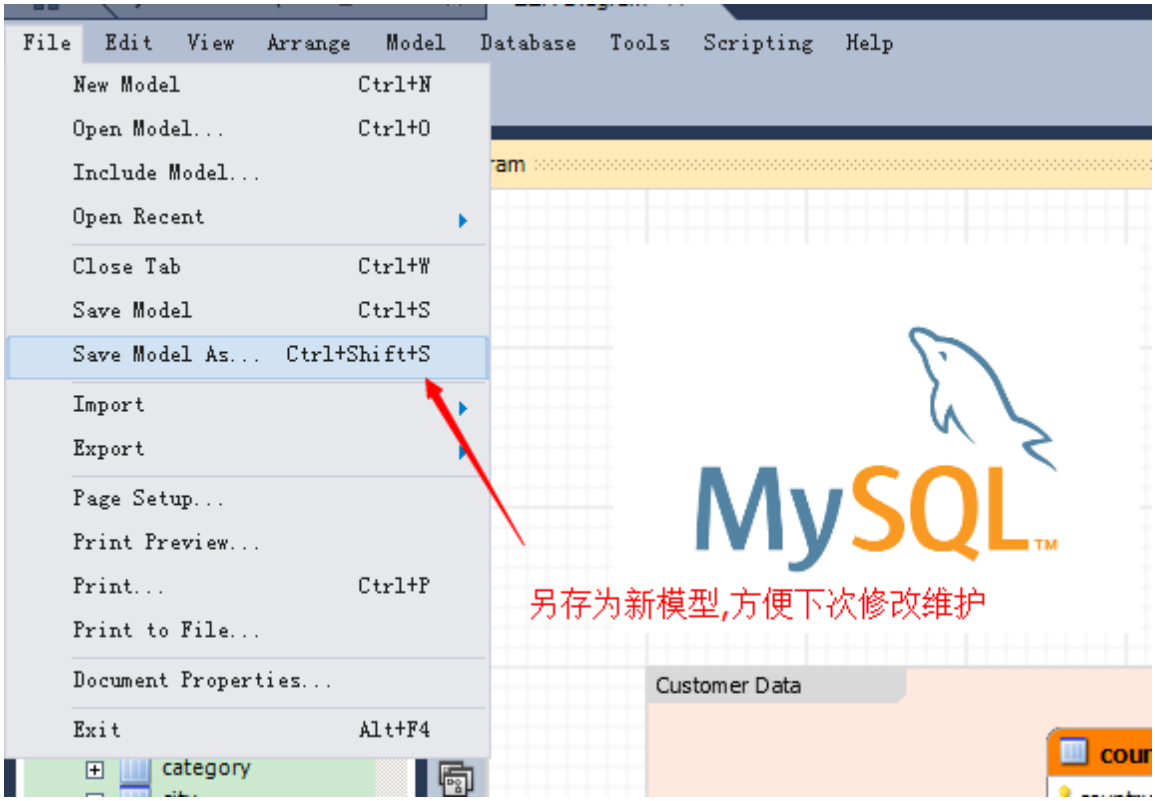

MySQL Connections
localhost
root
learn
127.0.0.1:3306
链接本地数据库
建模
Models
MODEL
sakila_full
09 Aug 13, 17:32


Model Overview
Add Diagram
Physical Schemas
mydb
MySQL Schema
创建新图表
Tables (0 items)
Add Table
Views (0 items)
Add View
Routines (0 items)
Add Routine
Routine Groups (0 items)
Add Group
Schema Privileges
SQL Scripts
Model Notes


MySQL Model (sakila_full.mw...
EER Diagram
File Edit View Arrange Model Database Tools Scripting Help
New Model Ctrl+N
Open Model... Ctrl+O
Include Model...
Open Recent
Close Tab Ctrl+W
Save Model Ctrl+S
Save Model As... Ctrl+Shift+S
Import
Export
Page Setup...
Print Preview...
Print... Ctrl+P
Print to File...
Document Properties...
Exit Alt+F4
Forward Engineer SQL CREATE Script... Ctrl+Shift+G
Export as PNG...
Export as SVG...
Export as Single Page PDF...
Export as Single Page PostScript File...
Customer Data
导出为sql文件或其它格式
country
country_id SMALLINT
1 more...
Indexes
address
address_id SMALLINT
category
city


MySQL Model*
EER Diagram
File Edit View Arrange Model Database Tools Scripting Help
New Model Ctrl+N
Open Model... Ctrl+O
Include Model...
Open Recent
Close Tab Ctrl+W
Save Model Ctrl+S
Save Model As... Ctrl+Shift+S
Import
Export
Page Setup...
Print Preview...
Print... Ctrl+P
Print to File...
Document Properties...
Exit Alt+F4
Reverse Engineer MySQL Create Script...
Import DBDesigner4 Model...
从sql文件导入
actor_id SMALLINT
first_name VARCHAR(45)
last_name VARCHAR(45)
last_update TIMESTAMP
Indexes
PRIMARY
idx_actor_last_name
sakila
Tables
actor
address
category
city
Catalog
Layers
User Types
Description Editor
No Selection
1:1
1:n
1:1
1:n
n:m
1:n


MySQL Workbench
MySQL Model*
EER Diagram
File Edit View Arrange Model Database Tools Scripting Help
Bird's Eye
Zoom: 100%
Catalog Tree
mydb
Tables
Views
Routine Groups
sakila
Tables
actor
address
category
city
Catalog Layers User Types
Description Editor
No Selection
Diagram
导入之后拖入到
画布中,可以体现模型
actor
actor_id SMALLINT
first_name VARCHAR(45)
last_name VARCHAR(45)
last_update TIMESTAMP
Indexes
PRIMARY
idx_actor_last_name



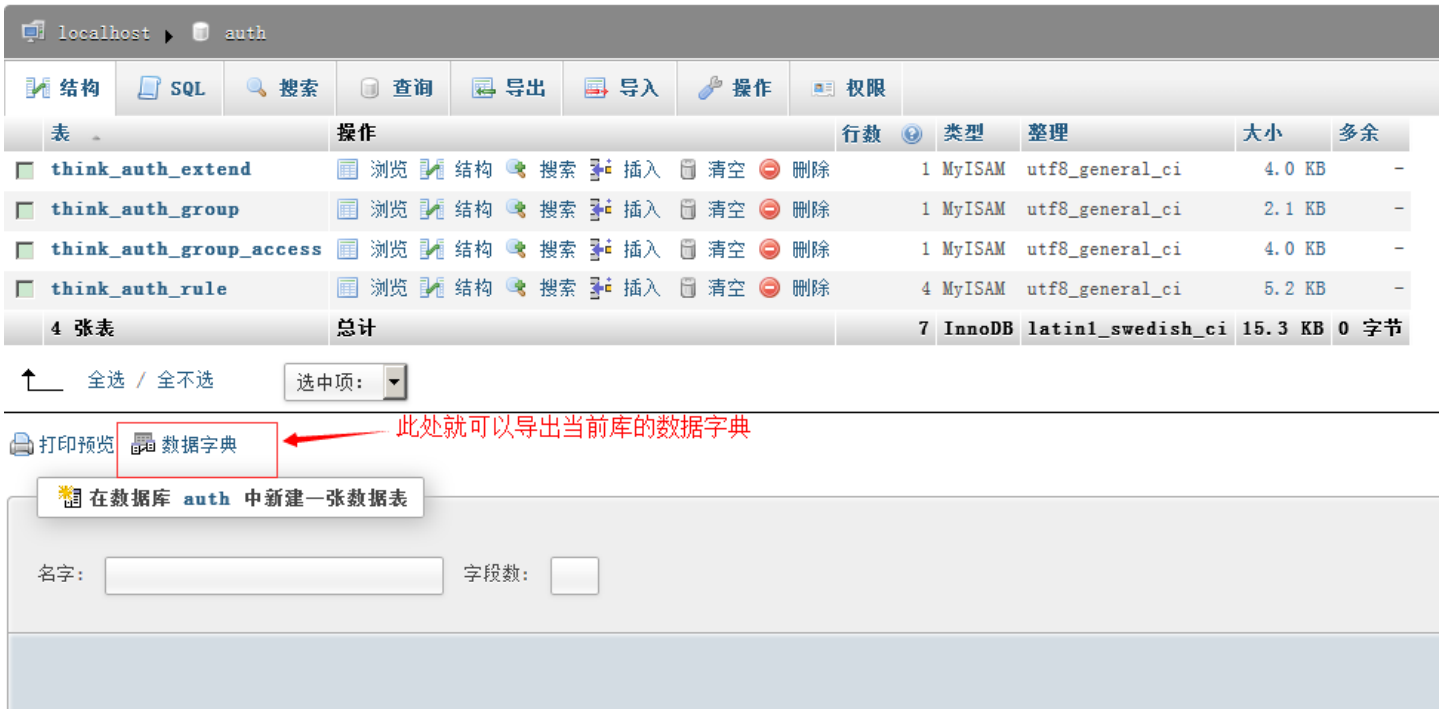
数据字典

在使用数据字典前,要保证sql的注释务必要详情

```
CREATE TABLE `sc_role` (
  `id` smallint(6) unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(20) NOT NULL DEFAULT '角色名称',
  `parentid` smallint(6) NOT NULL COMMENT '父角色ID',
  `status` tinyint(1) unsigned NOT NULL COMMENT '状态',
  `remark` varchar(255) NOT NULL COMMENT '备注',
  `create_time` int(11) unsigned NOT NULL COMMENT '创建时间',
  `update_time` int(11) unsigned NOT NULL COMMENT '更新时间',
  `listorder` int(3) NOT NULL DEFAULT '0' COMMENT '排序字段',
  PRIMARY KEY (`id`),
  KEY `parentId` (`parentid`),
  KEY `status` (`status`)
) ENGINE=MyISAM AUTO_INCREMENT=5 DEFAULT CHARSET=utf8 COMMENT='角色信息列表'
;
```

然后可以通过phpmyadmin来导出数据字典

phpAdmin是一个用php语言写的B/S架构,其配置文件在其应用的根目录config.inc.php;在该文件中可以设置数据库链接的一些信息



> ER图,数据模型,数据字典是对分析数据结构和维护数据库非常有帮助的,千万不要怕麻烦

数据中设计中的范式与反范式

> 设计关系数据库时，遵从不同的规范要求，设计出合理的关系型数据库，这些不同的规范要求被称为不同的范式，各种范式呈递次规范，越高的范式数据库冗余越小。但是有些时候一昧的追求范式减少冗余，反而会降低数据读写的效率，这个时候就要反范式，利用空间来换时间。

目前关系数据库有六种范式：第一范式（1NF）、第二范式（2NF）、第三范式（3NF）、巴斯-科德范式（BCNF）、第四范式(4NF）和第五范式（5NF，又称完美范式）。满足最低要求的范式是第一范式（1NF）。在第一范式的基础上进一步满足更多规范要求的称为第二范式（2NF），其余范式以次类推。一般说来，数据库只需满足第三范式(3NF）就行了。

三范式

第一范式（1NF）

即表的列的具有原子性,不可再分解，即列的信息，不能分解, 只要数据库是关系型数据库(mysql/oracle/db2/informix/sysbase/sql server)，就自动的满足1NF。

用户ID	用户名	密码	姓名	电话
1	zhang3	*******	张三	138888888

用户ID	用户名	密码	用户信息	
			姓名	电话
1	Zhang3	******	张三	1388888

> 关系型数据库: mysql/oracle/db2/informix/sysbase/sql server
> 非关系型数据库: (特点: 面向对象或者集合)
> NoSql数据库: MongoDB/redis(特点是面向文档)

第二范式（2NF）

第二范式（2NF）是在第一范式（1NF）的基础上建立起来的，即满足第二范式（2NF）必须先满足第一范式（1NF）。第二范式（2NF）要求数据库表中的每个实例或行必须可以被惟一地区分。**为实现区分通常需要我们设计一个主键来实现(这里的主键不包含业务逻辑)**

第三范式（3NF）

满足第三范式（3NF）必须先满足第二范式（2NF）。简而言之，第三范式（3NF）要求一个数据库表中不包含已在其它表中已包含的非主键字段。就是说，表的信息，如果能够被推导出来，就不应该单独的设计一个字段来存放(能尽量外键join就用外键join)。很多时候，我们为了满足第三范式往往会把一张表分成多张表

商品名称	价格	商品描述	重量	有效期	分类	分类描述
可乐	3.00		250ml	2014.6	酒水饮料	碳酸饮料
苹果	8.00		500g		生鲜食品	水果

商品ID	商品名称	价格	商品描述	重量	有效期
1	可乐	3.00		250ml	2014.6

分类ID	分类	分类描述
1	酒水饮料	碳酸饮料

分类ID	商品ID
1	1

反三范式

没有冗余的数据库未必是最好的数据库，有时为了提高运行效率，就必须降低范式标准，适当保留冗余数据。具体做法是： 在概念数据模型设计时遵守第三范式，降低范式标准的工作放到物理数据模型设计时考虑。降低范式就是增加字段，减少了查询时的关联，提高查询效率，因为在数据库的操作中查询的比例要远远大于DML的比例。但是反范式化一定要适度，并且在原本已满足三范式的基础上再做调整的。

字段类型与合理的选择字段类型

> 本篇博客稍微有点长,它实际上包括两个内容:一是mysql字段类型的介绍,二是在mysql建表过程中是如何正确选择这些字段类型;

字段类型

数值

MySQL 的数值数据类型可以大致划分为两个类别，一个是整数，另一个是浮点数或小数。
许多不同的子类型对这些类别中的每一个都是可用的，每个子类型支持不同大小的数据，并且 MySQL 允许我们指定数值字段中的值是否有正负之分(UNSIGNED)或者用零填补(ZEROFILL)。

1bit 位　　1 字节=8bit　　1k=1024 字节　　1 兆=1024k　　1G=1023M　　1T=1024G

类型	大小	范围（有符号）	范围（UNSIGNED）	用途
TINYINT	1 字节	(-128，127)	(0，255)	小整数值
SMALLINT	2 字节	(-32 768，32 767)	(0，65 535)	大整数值
MEDIUMINT	3 字节	(-8 388 608，8 388 607)	(0，16 777 215)	大整数值
INT 或 INTEGER	4 字节	(-2 147 483 648，2 147 483 647)	(0，4 294 967 295)	大整数值
BIGINT	8 字节	(-9 233 372 036 854 775 808，9 223 372 036 854 775 807)	(0，18 446 744 073 709 551 615)	极大整数值
FLOAT	4 字节	(-3.402 823 466 E+38，1.175 494 351 E-38)，0，(1.175 494 351 E-38，3.402 823 466 351 E+38)	0,(1.175 494 351 E-38,3.402 823 466 E+38)	单精度浮点数值
DOUBLE	8 字节	(1.797 693 134 862 315 7 E+308，2.225 073 858 507 201 4 E-308)，0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	0，(2.225 073 858 507 201 4 E-308，1.797 693 134 862 315 7 E+308)	双精度浮点数值
DECIMAL	对 DECIMAL(M,D) ，如果 M>D，为 M+2 否则为 D+2	依赖于 M 和 D 的值	依赖于 M 和 D 的值	小数值

- INT
  在 MySQL 中支持的 5 个主要整数类型是 TINYINT，SMALLINT，MEDIUMINT，INT 和 BIGINT。这些类型在很大程度上是相同的，只有它们存储的值的大小是不相同的。

MySQL 以一个可选的显示宽度指示器的形式对 SQL 标准进行扩展(如 INT(6),6即是其宽度指示器,**该宽度指示器并不会影响int列存储字段的大小,也就是说,超过6位它不会自动截取,依然会存储,只有超过它本身的**

存储范围才会截取;此处宽度指示器的作用在于该字段是否有zerofill,如果有就未满足6位的部分就会用0来填充)，这样当从数据库检索一个值时，可以把这个值加长到指定的长度。例如，指定一个字段的类型为 INT(6)，就可以保证所包含数字少于 6 个的值从数据库中检索出来时能够自动地用空格填充。需要注意的是，使用一个宽度指示器不会影响字段的大小和它可以存储的值的范围。

万一我们需要对一个字段存储一个超出许可范围的数字，MySQL 会根据允许范围最接近它的一端截短后再进行存储。还有一个比较特别的地方是，MySQL 会在不合规定的值插入表前自动修改为 0。

- unsigned 和 zerofill
  UNSIGNED 修饰符规定字段只保存正值,即无符号,而mysql字段默认是有符号的。因为不需要保存数字的正、负符号，可以在储时节约一个"位"的空间(即翻一倍)。从而增大这个字段可以存储的值的范围。注意这个修饰符要紧跟在数值类型后面;
  ZEROFILL 修饰符规定 0（不是空格）可以用来真补输出的值。使用这个修饰符可以阻止 MySQL 数据库存储负值,如果某列设置为zerofill,那它自动就unsigned。这个值要配合int,tinyint,smallint,midiumint等字段的宽度指示器来用;XXint(M),如果没有zerofill,这个M的宽度指示器是没有意义的.(注意,测试前导0的时候,还是去黑窗口测试;)

> 为什么mysql存储的值要分有符号和无符号呢?因为一个字节,占8bit;也就1个bit有0和1两种可能,8个bit就是2^8 = 256种可能,也就是0~255;但如果是有符号的话,就得拿一个1bit来存储这个负号,本来8bit只剩7bit,2^7 = 128,也就是-128~127(正数部分包含一个0);

- FLOAT、DOUBLE 和 DECIMAL 类型
  MySQL 支持的三个浮点类型是 FLOAT、DOUBLE 和 DECIMAL 类型。FLOAT 数值类型用于表示单精度浮点数值，而 DOUBLE 数值类型用于表示双精度浮点数值。
  与整数一样，这些类型也带有附加参数：一个显示宽度指示器和一个小数点指示器(必须要带有指示器,要不然会查不到结果,**并且宽度指示器和XXint类型的宽度指示器不同,这里是有实际限制宽度的**)。比如语句 FLOAT(7,3) 规定显示的值不会超过 7 位数字(包括小数位)，小数点后面带有 3 位数字。对于小数点后面的位数超过允许范围的值，MySQL 会自动将它四舍五入为最接近它的值，再插入它。
  DECIMAL 数据类型用于精度要求非常高的计算中，这种类型允许指定数值的精度和计数方法作为选择参数。精度在这里指为这个值保存的有效数字的总个数，而计数方法表示小数点后数字的位数。比如语句 DECIMAL(7,3) 规定了存储的值不会超过 7 位数字，并且小数点后不超过 3 位。
  FLOAT 类型在长度比较高比如 float(10,2)和 decimal(10,2)同时插入一个符合(10,2)宽度的数值,float就会出现最后小数点出现一些出入;
  UNSIGNED 和 ZEROFILL 修饰符也可以被 FLOAT、DOUBLE 和 DECIMAL 数据类型使用。并且效果与 INT 数据类型相同。

> 关于flaot和double
> 在这里我建议,干脆忘记mysql有double这个数据类型。至于why？就不要管它了

字符串类型

MySQL 提供了 8 个基本的字符串类型，可以存储的范围从简单的一个字符到巨大的文本块或二进制字符串数据。

1 英文字符 占用 1 字节

1 个中文字符 占用 2 个字节

类型	大小	用途
CHAR	0-255 字节	定长字符串
VARCHAR	0-255 字节	变长字符串
TINYBLOB	0-255 字节	不超过 255 个字符的二进制字符串
TINYTEXT	0-255 字节	短文本字符串
BLOB	0-65 535 字节=64k	二进制形式的长文本数据
TEXT	0-65 535 字节	长文本数据
MEDIUMBLOB	0-16 777 215 字节=16M	二进制形式的中等长度文本数据
MEDIUMTEXT	0-16 777 215 字节	中等长度文本数据
LOGNGBLOB	0-4 294 967 295 字节=4G	二进制形式的极大文本数据
LONGTEXT	0-4 294 967 295 字节	极大文本数据

- BINARY
  BINARY不是函数，是类型转换运算符，它用来强制它后面的字符串为一个二进制字符串，可以理解为在字符串比较的时候区分大小写

  ```
  SELECT BINARY 'ABCD' = 'abcd' as COM1,'ABCD' = 'abcd' as COM2; -- COM1输出为0,COM2输出为1;
  ```

- CHAR 和 VARCHAR 类型
  CHAR 类型用于定长字符串，并且必须在圆括号内用一个大小修饰符来定义。这个大小修饰符的范围从 0-255。**比指定长度大的值将被截短，而比指定长度小的值将会用空格作填补。**

CHAR 类型可以使用 BINARY 修饰符。当用于比较运算时，这个修饰符使 CHAR 以二进制方式参于运算，而不是以传统的区分大小写的方式。

CHAR 类型的一个变体是 VARCHAR 类型。它是一种可变长度的字符串类型，并且也必须带有一个范围在 0-255 之间的指示器。

CHAR 和 VARCHGAR 不同之处在于 MYSQL 数据库处理这个指示器的方式：CHAR 把这个大小视为值的大小，不长度不足的情况下就用空格补足。而 VARCHAR 类型把它视为最大值并且只使用存储字符串实际需要的长度（增加一个额外字节来存储字符串本身的长度）来存储值。所以短于指示器长度的 VARCHAR 类型不会被空格填补，但长于指示器的值仍然会被截短。

因为 VARCHAR 类型可以根据实际内容动态改变存储值的长度，所以在不能确定字段需要多少字符时使用

VARCHAR 类型可以大大地节约磁盘空间、提高存储效率。**但如果确切知道字符串长度,比如就在50~55之间,那就用 CHAR 因为 CHAR 类型由于本身定长的特性使其性能要高于 VARCHAR**;

VARCHAR 类型在使用 BINARY 修饰符时与 CHAR 类型完全相同。

- TEXT 和 BLOB 类型
  对于字段长度要求超过 255 个的情况下，MySQL 提供了 TEXT 和 BLOB 两种类型。根据存储数据的大小，它们都有不同的子类型。这些大型的数据用于存储文本块或图像、声音文件等二进制数据类型。

TEXT 和 BLOB 类型在分类和比较上存在区别。BLOB 类型区分大小写，而 TEXT 不区分大小写。大小修饰符不用于各种 BLOB 和 TEXT 子类型。比指定类型支持的最大范围大的值将被自动截短。

时间类型

在处理日期和时间类型的值时，MySQL 带有 5 个不同的数据类型可供选择。

类型	大小 (字节)	范围	格式	用途
DATE	3	1000-01-01/9999-12-31	YYYY-MM-DD	日期值
TIME	3	'-838:59:59'/'838:59:59'	HH:MM:SS	时间值或持续时间
YEAR	1	1901/2155	YYYY	年份值
DATETIME	8	1000-01-01 00:00:00/9999-12-31 23:59:59	YYYY-MM-DD HH:MM:SS	混合日期和时间值
TIMESTAMP	8	1970-01-01 00:00:00/2037 年某时	YYYYMMDD HHMMSS	混合日期和时间值，时间戳

- DATE、TIME 和 YEAR 类型
  MySQL 用 DATE 和 YEAR 类型存储简单的日期值，使用 TIME 类型存储时间值。这些类型可以描述为字符串或不带分隔符的整数序列。如果描述为字符串，DATE 类型的值应该使用连字号作为分隔符分开，而 TIME 类型的值应该使用冒号作为分隔符分开。

需要注意的是，没有冒号分隔符的 TIME 类型值，将会被 MySQL 理解为持续的时间，而不是时间戳。

MySQL 还对日期的年份中的两个数字的值，或是 SQL 语句中为 YEAR 类型输入的两个数字进行最大限度的通译。因为所有 YEAR 类型的值必须用 4 个数字存储。MySQL 试图将 2 个数字的年份转换为 4 个数字的值。把在 00-69 范围内的值转换到 2000-2069 范围内。把 70-99 范围内的值转换到 1970-1979 之内。如果 MySQL 自动转换后的值并不符合我们的需要，请输入 4 个数字表示的年份。

- DATETIME 和 TIMESTAMP 类型
  除了日期和时间数据类型，MySQL 还支持 DATETIME 和 TIMESTAMP 这两种混合类型。它们可以把日期和时间作为单个的值进行存储。这两种类型通常用于自动存储包含当前日期和时间的时间戳，并可在需要执行大量数据库事务和需要建立一个调试和审查用途的审计跟踪的应用程序中发挥良好作

用。

> 如果我们对 TIMESTAMP 类型的字段没有明确赋值，或是被赋与了 null 值。MySQL 会自动使用系统当前的日期和时间来填充它。

复合类型

MySQL 还支持两种复合数据类型 ENUM 和 SET，它们扩展了 SQL 规范。虽然这些类型在技术上是字符串类型，但是可以被视为不同的数据类型。一个 ENUM 类型只允许从一个集合中取得一个值；而 SET 类型允许从一个集合中取得任意多个值。

- ENUM 类型
  ENUM 类型因为只允许在集合中取得一个值，有点类似于单选项。在处理相互排拆的数据时容易让人理解，比如人类的性别。ENUM 类型字段可以从集合中取得一个值或使用 null 值，除此之外的输入将会使 MySQL 在这个字段中插入一个空字符串。另外如果插入值的大小写与集合中值的大小写不匹配，MySQL 会自动使用插入值的大小写转换成与集合中大小写一致的值。

ENUM 类型在系统内部可以存储为数字，并且从 1 开始用数字做索引。一个 ENUM 类型最多可以包含 65536 个元素，其中一个元素被 MySQL 保留，用来存储错误信息，这个错误值用索引 0 或者一个空字符串表示。

MySQL 认为 ENUM 类型集合中出现的值是合法输入，除此之外其它任何输入都将失败。这说明通过搜索包含空字符串或对应数字索引为 0 的行就可以很容易地找到错误记录的位置。

- SET 类型
  SET 类型与 ENUM 类型相似但不相同。SET 类型可以从预定义的集合中取得任意数量的值。并且与 ENUM 类型相同的是任何试图在 SET 类型字段中插入非预定义的值都会使 MySQL 插入一个空字符串。**如果插入一个即有合法的元素又有非法的元素的记录，MySQL 将会保留合法的元素，除去非法的元素。**
  一个 SET 类型最多可以包含 64 项元素。还去除了重复的元素，所以 SET 类型中不可能包含两个相同的元素。
  希望从 SET 类型字段中找出非法的记录只需查找包含空字符串或二进制值为 0 的行。

字段类型总结

1. 虽然上面列出了很多字段类型,但最常用也就是varchar(255),char(255),text,tinyint(4),smallint(6),mediumint,int(11)几种。
2. 复合类型我们一般用tinyint,更快的时间更省的空间以及更容易扩展
3. 关于手机号，推荐用char(11),char(11)在查询上更有效率，因为手机号是一个活跃字段参与逻辑会很多。
4. 一些常用字段举例

姓名：char(20)
价格：DECIMAL(7, 3)
产品序列号：SMALLINT(5) unsigned
文章内容: TEXT
MD5: CHAR(32)
ip: char(15)
time: int(10)
email char(32)

合理的选择数据类型

- 选择合理范围内最小的
  我们应该选择最小的数据范围，因为这样可以大大减少磁盘空间及磁盘I/0读写开销，减少内存占用，减少CPU的占用率。

- 选择相对简单的数据类型
  数字类型相对字符串类型要简单的多，尤其是在比较运算时，所以我们应该选择最简单的数据类型，*比如说在保存时间时，因为PHP可以良好的处理LINUX时间戳所以我们可以将日期存为int(10)要方便、合适、快速的多*。

> 但是，工作中随着项目越做越多，业务逻辑的处理越来越难以后，我发现时间类型还是用时间类型本身的字段类型要好一些，因为mysql有着丰富的时间函数供我使用，方便我完成很多与时间相关的逻辑，比如月排行榜，周排行榜，当日热门，生日多少天等等逻辑

- 不要使用null
  为什么这么说呢，因为MYSQL对NULL字段索引优化不佳，增加更多的计算难度，同时在保存与处理NULL类形时，也会做更多的工作，所以从效率上来说，不建议用过多的NULL。有些值他确实有可能没有值，怎么办呢？解决方法是数值弄用整数0，字符串用空来定义默认值即可。

- 字符串类型的使用
  字符串数据类型是一个万能数据类型，可以储存数值、字符串、日期等。
  保存数值类型最好不要用字符串数据类型，这样存储的空间显然是会更大，而且在排序时字符串的9是大于22的，其实如果进行运算时mysql会将字符串转换为数值类型，大大降低效果，而且这种转换是不会走原有的索引的。
  如果明确数据在一个完整的集合中如男，女，那么可以使用set或enum数据类型，这种数据类型在运算及储存时以数值方式操作，所以效率要比字符串更好，同时空间占用更少。

- VARCHAR与CHAR
  VARCHAR是可变长度字符串类型，那么即然长度是可变的就会使用1，2个字节来保存字符的长度，如果长度在255内使用1个字节来保存字符长度，否则使用2个字符来保存长度。由于varchar是根据储存的值来保存数据，所以可以大大节约磁盘空间。
  如果数据经常被执行更新操作，由于VARCHAR是根据内容来进行储存的，所以mysql将做更多的工作

来完成更新操作，如果新数据长度大于老数据长度一些存储引擎会进行拆分操作处理。同时varchar会完全保留内部所有数据，最典型的说明就是尾部的空格。
CHAR固定长度的字符串保存类型，CHAR会去掉尾部的空格。在数据长度相近时使用char类型比较合适，比如md5加密的密码用户名等。
如果数据经常进行更新修改操作，那么CHAR更好些，因为char长度固定，性能上要快。

- 数值类型的选择
  数值数据类型要比字符串执行更快，区间小的数据类型占用空间更少，处理速度更快，如tinyint可比bigint要快的多

  选择数据类型时要考虑内容长度，比如是保存毫米单位还是米而选择不同的数值类型

  整数

  整数类型很多比如tinyint、int、smallint、bigint等，那么我们要根据自己需要存储的数据长度决定使用的类型，同时tinyint(10)与tinyint(100)在储存与计算上并无任何差别，区别只是显示层面上，但是我们也要选择适合合适的数据类型长度。可以通过指定zerofill属性查看显示时区别。

  浮点数与精度数值

  浮点数float在储存空间及运行效率上要优于精度数值类型decimal，但float与double会有舍入错误而decimal则可以提供更加准确的小数级精确运算不会有错误产生计算更精确，适用于金融类型数据的存储。

表的垂直拆分和水平拆分

垂直拆分

垂直拆分是指数据表**列的拆分**，把一张列比较多的表拆分为多张表

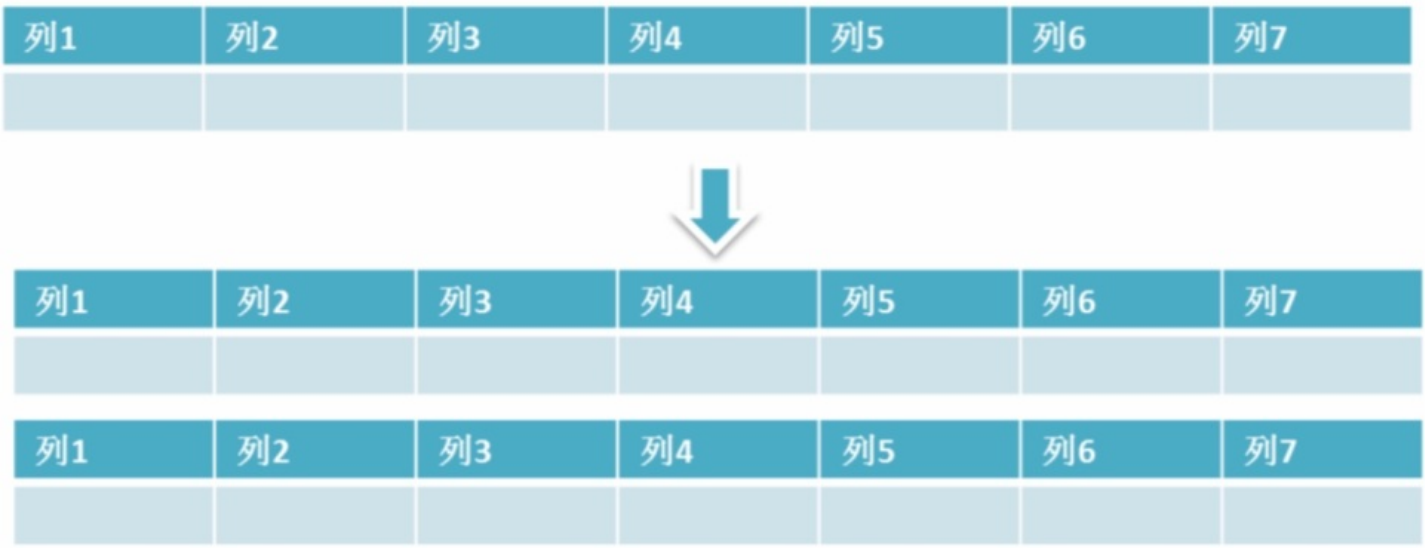
通常我们按以下原则进行垂直拆分:

1. 把不常用的字段单独放在一张表;
2. 把text，blob等大字段拆分出来放在附表中;
3. 经常组合查询的列放在一张表中;

> 垂直拆分更多时候就应该在数据表设计之初就执行的步骤，然后查询的时候用jion关键起来即可;

水平拆分

水平拆分是指数据表**行的拆分**，表的行数超过200万行时，就会变慢，这时可以把一张的表的数据拆成多张表来存放。



水平拆分的一些技巧

1. 拆分原则

通常情况下，我们使用取模的方式来进行表的拆分;比如一张有400W的用户表 `users` ，为提高其查询效率

我们把其分成4张表 users1，users2，users3，users4
通过用ID取模的方法把数据分散到四张表内 Id%4+1 = [1,2,3,4]
然后查询,更新,删除也是通过取模的方法来查询

```
$_GET['id'] = 17,
17%4 + 1 = 2,
$tableName = 'users'.'2'
Select * from users2 where id = 17;
```

在insert时还需要一张临时表 uid_temp 来提供自增的ID,该表的唯一用处就是提供自增的ID;

```
insert into uid_temp values(null);
```

得到自增的ID后,又通过取模法进行分表插入;

> 注意,进行水平拆分后的表,字段的列和类型和原表应该是相同的,但是要记得去掉auto_increment自增长

另外

- 部分业务逻辑也可以通过地区，年份等字段来进行归档拆分;
- 进行拆分后的表，只能满足部分查询的高效查询需求，这时我们就要在产品策划上，从界面上约束用户查询行为。比如我们是按年来进行归档拆分的,这个时候在页面设计上就约束用户必须要先选择年,然后才能进行查询;
- 在做分析或者统计时，由于是自己人的需求,多点等待其实是没关系的,并且并发很低,这个时候可以用union把所有表都组合成一张视图来进行查询,然后再进行查询;

```
Create view users as select from users1 union select from users2 union.....
```