# 1 索引优化

原文链接：<https://www.douban.com/note/345871485/>

## 避免在where子句中对字段进行null值判断

因为where子句中对字段进行null值判断，引擎会放弃索引而进行全表扫描。

图1



改动：

1.可以在出现null的字段设置默认值0，确保该字段没有null值。

图2



## 避免在where子句中对字段使用!=或<>不等于操作符

因为where子句中对字段进行!=或<>这些不等于操作符，引擎会放弃索引而进行全表扫描。

图1

select id from t where num <> 1

## where和order by子句中对字段建立索引

where和order by子句中对字段建立索引，避免全表扫描。

## 避免在where子句中使用or连接条件

在where子句中使用or连接条件，引擎会放弃索引而进行全表扫描。

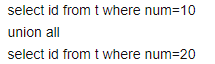
图1



改动：

1. 将or改成union all实现。

图1



## 避免用in和not int

in和not in，不一定会走索引，可能会放弃索引。

（注意：鉴于网络上关于in是否走索引答案不明确，in是可能会放弃索引的）

图1



改动：

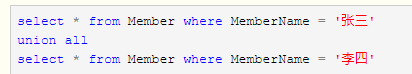
1. 对于连续的值，可以使用between...and代替in。

图2



1. 对于非连续的值，可以使用union或union all。

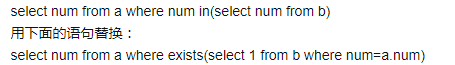
图3



1. 当值不明确，且子表大时，使用exsists代替in。

（注意：子表小时，因为in会使用hash连接将子表载入内存，in速度快；子表大时，in的hash连接载入内存效果不好，但exists使用了子表的索引，exists速度快）

图4



## 4 避免在字段有大量重复值时建立索引

因为当字段比如sex，值male和female各占一半，存在大量重复值，这时候sql查询不会走索引，而且索引还会影响插入和更新的效率。

比如：状态列，这些状态列重复值很多，不适合建立索引。

改动：

1. 使用联合索引。
2. 删除该字段索引。

## 避免使用like ‘%xxx%’

使用like ‘%xxx%’，引擎会放弃索引而进行全表扫描。

图1



改动：

1. like不以%开头，还是可以走索引。

图2



1. like不以%结尾，可以结合reverse函数走索引。

（注意：reverse函数返回字符串值的逆向值）

（注意：reverse函数本身不会走索引，所以要对reverse的值建索引一起使用，注意这不是反向索引，不要混淆）

图3



图4



## 避免在where子句中对字段进行表达式操作

在where子句中对字段进行表达式操作，引擎会放弃索引而进行全表扫描。

图1



改动：

1. 字段处去除表达式

图2



## 避免在where子句中对字段进行函数操作

在where子句中对索引列进行函数操作，引擎会放弃索引而进行全表扫描。

图1



改动：

1. 在索引列处去除函数。

图2



图3（在索引列out\_date处移除函数，注意：current\_date不是索引列，所以加函数没关系）



## 复合索引至少使用到第一个索引字段

如果没有用复合索引的第一个字段，则不会走索引。

## 能使用数字型字段避免使用字符型字段

因为数字型字段只需要比较一次，字符型字段需要一个一个字符比较。

## 避免使用select \*

不要返回用不到的任何字段。

## 临时表数据插入，使用select into避免产生大量日志

临时表数据的导入可以select into或者create table再insert，由于insert产生大量日志，使用select into可以避免产生大量日志，因为产生日志少，所以速度也更快。

## 临时表先truncate再drop避免长时间的系统表锁定

因为临时表的添加和删除数据会导致一部分系统表的锁定，如果要强制drop掉临时表，先truncate可以减少删除数据时锁定临时表的时间。

## 使用前缀索引

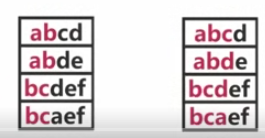
因为MySQL中索引是有大小限制的，不同的引擎限制的大小不同。对于太长的字符串由于索引有大小限制，所以可以建立前缀索引。

（注意：前缀索引选择前缀的宽度时，应该选择不重复的索引值和表的记录数的比值越大的越好）

图1



图2（2位前缀的比值是2/4，3位前缀的比值是4/4，所以应该选择3位作为前缀索引）



## 联合索引的列的顺序的选择

因为联合索引是按列的顺序一个一个进行索引的，如果第一个列索引后剩下的行很少，那么对后面的列肯定效果更好。

1. 经常使用的列优先。
2. 列的值重复性低的列优先。
3. 宽度小的列优先。

# 2 union和union all的区别

union all只是简单的将2个结果合并。

union会对union all合并的结果集进行排序，然后去除重复的记录。

（注意：union和union all不要求字段名称相同，但字段数量，字段类型，字段顺序必须相同）

所以，如果可以确认2个待合并的结果集中不包含重复数据，就用union all速度快。

# 3 varchar和varchar2的区别

1. varchar是标准sql提供的数据类型，varchar2是oracle独有的数据类型。
2. oracle保证varhcar2在varchar2在向上和向下任何版本中兼容，但不保证varchar。所以想与oracle兼容用varchar2，与其他数据库兼容使用varchar。
3. varchar将空串存储为空串，varchar2都把空串当成null处理。
4. varchar存储可变长度的字符数据，最长2000个字符；varchar2存储可变长度的字符数据（注意：实际占用多少，就存储多少，()中的是最大值），最长4000个字符。

# 4 in和exists的区别

1. in的子查询只能select一个字段；exists的子查询可以select多个字段，不过都没意义，exists只在乎子查询是否有记录数。
2. in是hash连接（将小表的连接列(col2)载入内存形成哈希表，然后大表的连接列(col1)与内存中的哈希表使用哈希值连接）；exists是循环外表的每一行连接列的值，对子表进行值查询。
3. exists走子表的索引，in不一定走索引。

# 5 exists

exists：（子查询中能查出记录）则查出主查询的结果；（子查询中能查不出记录）则不查出主查询的结果。

# 6 临时表

## 6.1 oracle临时表

oracle临时表分为：会话级临时表和事务级临时表。

### 6.1.1 会话级

临时表的数据只在会话生命周期存在，会话结束后Oracle自动删除临时表数据。

（注意：会话结束后自动删除的是表数据，而表是不会删除的）

（注意：一个sqlplus或一个plsql的窗口等于一个会话）

图1

**create** **global** **temporary** **table** session\_temp\_tb (col1 **varchar**(20)) **on** **commit** preserve **rows**;

**insert** **into** session\_temp\_tb **values**('test');

**select** \* **from** session\_temp\_tb;

### 6.1.2 事务级临时表

事务结束（包括commit或rollback）时，Oracle自动删除临时表数据。

图1

**create** **global** **temporary** **table** transaction\_temp\_tb (col1 **varchar**(20)) **on** **commit** **delete** **rows**;

**insert** **into** transaction\_temp\_tb **values**('test');

**select** \* **from** transaction\_temp\_tb;

## 6.2 临时表的应用

### 6.2.1 两个大表关联变成小表关联

原文链接：<https://zhidao.baidu.com/question/302606235.html>

两个大表关联的速度非常慢，可以将大表中需要的数据检索出来，放到小表中。然后2个小表关联。

# 7 不适合建立索引的情况

## 7.1 数据唯一性差的字段不要使用索引

数据唯一性差的字段不要使用索引，比如性别，只有男女2种可能，这样索引产生的二叉树级别少，这样的二叉树和全表扫描差不多。

## 7.2 频繁更新的字段不要使用索引

频繁更新的字段不要使用索引，因为索引也改动频繁。

## 7.3 数据量少的表不要使用索引

数据量少的表不要使用索引，改善效果不大。

# 8 MySQL分表

## 8.1 分表的原因

因为MySQL单表承载的数据量有限，一般在1000万以内，当字段多时还少一些。因此当数据量很大时就需要对数据进行拆分。

## 8.2 拆分因子

拆分因子表示按照什么维度拆分，拆分因子和业务强绑定。

比如：1.京东的京豆根据用户维度拆分。

1. 电商的商品信息根据商家维度拆分。
2. 单既属于商家，又属于用户，可以优点根据用户维度拆分。

## 8.3 分表数的计算

1.根据每天数据量和系统可支撑年限，预估系统未来几年后的数据量。比如：每天100W订单，就是100W条记录，系统可支撑5年，5\*365\*100W约等于18.25亿数据。

2.每张表最多存1000W数据，计算大约182张表。

3.考虑到一定的上浮空间，一般选择2的指数，比如256张表。

4.表的命名，可以采用bountry\_1,bountry\_2...。

## 8.4 定位数据到哪个分表中

可以使用一致性hash算法，对拆分因子求hash值，然后对分表数求余，余数就是哪个分表。

（注意：一致性hash算法一般由应用层实现）

图1

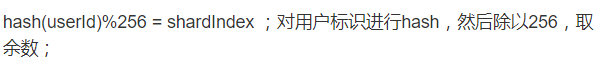


图2（用户id为1到了分表1，以此类推）

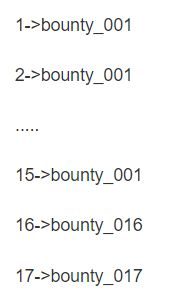
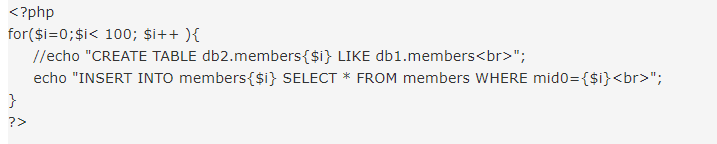


图3



## 8.5 水平分表和垂直分表

水平分表：表数据量很大，需要将数据分摊到若若干个分表中。比如：QQ的用户有100亿，将这个表拆成1000个表，每个表1000万条记录，记为：qq\_1,qq\_2等。

垂直分表：表数据量不大，但某些字段很长，导致表占用空间很大。比如：学生表的id和name很短，但题目和回答字段很长，这时候可以将题目和回答放到单独一个表中，并于原表建立一对一的关系。

## 8.6 分表和分库的区别

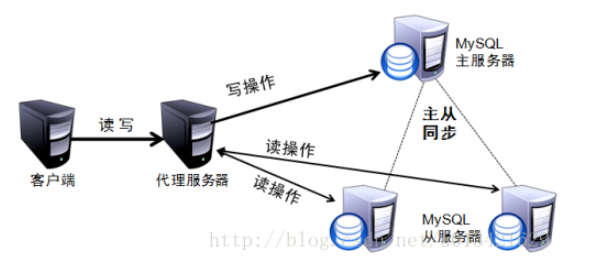
分库能减少单台数据库的并发访问压力。

# 9 MySQL读写分离

读写分离就是在主服务器上写入，将数据同步到从服务器，然后从从服务器读取数据。

（注意：从服务器不能写入）

图1



## 9.1 读写分离的原因

写操作比读操作的耗时大，大并发情况下影响效率。

## 9.2 MySQL支持的复制类型

由于主服务器的数据要复制到从服务器，从而保证主服务器被写入的数据能在从服务器中读取到。MySQL提供的复制类型有3种：

1. 基于语句的复制（默认）：在主服务器上执行sql语句，在从服务器上执行同样的语句，执行效率高。
2. 基于行的复制：将行改变的内容复制到从服务器，而不是让sql在从服务器上执行一遍。
3. 混合类型的复制：默认采用基于语句的复制，当发现基于语句的复制无法精确复制时，再采用基于行的复制。

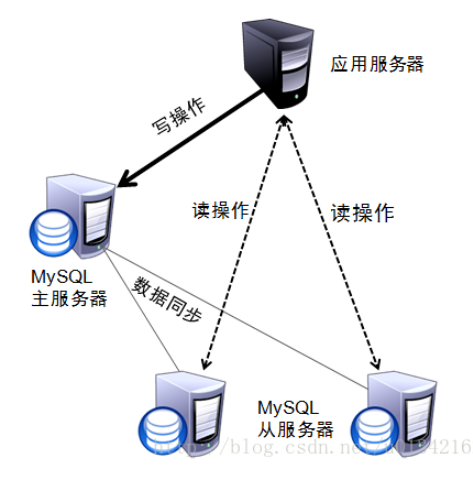
## 9.3 读写分离的实现方式

1.基于应用程序代码实现。

2.基于中间代理层实现。

### 9.3.1 基于应用程序代码实现

图1



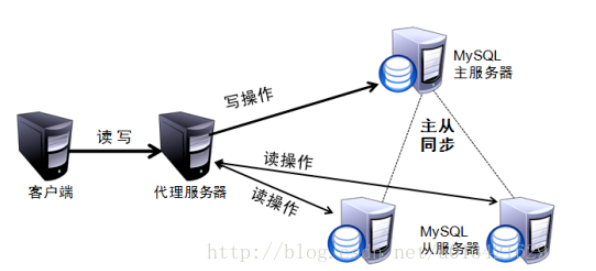
缺点：

1.对代码的改动大。

### 9.3.2 基于中间代理层实现

代理位于应用服务器和数据库服务器之间，代理接收应用服务器的请求后转发到读或写的数据库服务器。

图1



常用的中间代理层有3种：

1. mysql\_proxy：mysql\_proxy是MySQL的一个开源项目，通过自带的lua脚本进行sql判断。
2. Atlas：360在mysql\_proxy 0.8.2的版本上进行了优化，供360内部使用，支持事务和存储过程。
3. Amoeba：阿里巴巴使用java语言开发，供阿里巴巴内部使用，不支持事务和存储过程。

### 9.3.3 mysql\_proxy实现读写分离

原文链接：<https://www.cnblogs.com/phpstudy2015-6/p/6687480.html>

## 9.4 读写分离和分表分库的区别

1.读写分离将写集中在master表，读操作分摊到多个slave表，提高了读操作的效率；但当单表的数据量达到千万或者亿级别，每个master或slave单表的读写都会非常慢，这时候就需要采用分表分库，减少单表的数据量。

# 10 MySQL 常用引擎

原文链接：<https://www.cnblogs.com/sunsky303/p/8274586.html>

MySQL采用不同的技术（不同的存储机制，不同的索引技巧，不同的锁定水平）来将数据存储在文件（或内存）中。这些不同的技术称为MySQL存储引擎。

## 10.1 Innodb引擎

Innodb是一个事务型的存储引擎，支持行级锁和外键约束。

（注意：Innodb默认开启事务，可以把事务停掉）

优点：

1.支持ACID事务（ACID是指事务的原子性，一致性，隔离性，持久性），需要使用事务时Innodb是首选。

2.提供行级锁，锁的粒度小，因为是行级锁所以写操作不会锁表，并发高时Innodb会提高效率。

3.支持外键。

缺点：

1.没有保存表的行数，所以当select count(\*) from table时会全表扫描。

## 10.2 MyIASM引擎

MyIASM是MySQL默认的存储引擎，不支持事务，也不支持行锁和外键约束。

（注意：MyIASM不支持事务不代表业务代码层进行类似事务的控制）

优点：

1. 保存了表的行数，所以当select count(\*) from table时不需要全表扫描。
2. MyIASM在创建表的时候，会创建3个文件，一个是.frm文件保存表的定义，一个是.MYD文件保存表的数据，一个是.MYI文件保存表的索引。因为操作系统对大文件的操作慢，而MyIASM拆成3个文件后，查询数据时只读取.MYD文件可以提高查询效率。

缺点：

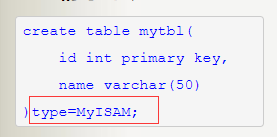
1. 不支持事务。
2. 不支持行级锁。
3. 不支持外键。
4. 因为MyIASM是表锁，所以读锁和写锁是互斥的，即读和写是串行执行的。当存在大量更新操作时，会导致查询操作很难获取读锁，所以MyIASM在存在大量更新操作时会影响读取效率。

## 10.3 修改引擎

（1）修改配置文件my.ini，在[mysqld]后面添加default-storage-engine=InnoDB修改默认引擎，重启服务。

（2）建表时指定该表使用的引擎。

图1



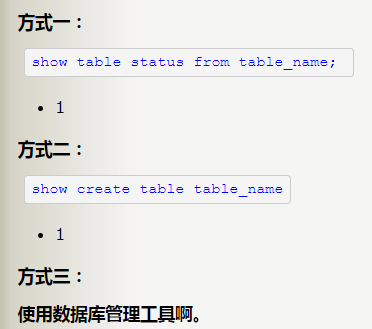
1. 建表后修改该表使用的引擎。

图2



查看修改的引擎：

图3



# 11 MySQL数据类型优化

## 11.1 整数类型

整数类型：tinyint（占8位），smallint（占16位），mediumint（占24位），int（占32位），bigint（占64位），存储值的范围是“-2的n-1次方~2的n-1次方”，n是占用位数。

（注意：整数类型有可选的unsigned属性，表示不允许负值，从而将正数的上限提高一倍，比如tinyint的默认范围是-128~127，tinyint.unsigned范围是0~255）

### 11.1.1 整数的宽度

可以为整数指定宽度，比如：int(11)，但值的存储范围不会因为宽度而改变。它只是规定了MySQL交互工具（比如MySQL命令行客户端）显示字符的个数而已。

（注意：当宽度和zerofill填充0一起使用时，如果实际数值的位数小于宽度的位数，则左边缺少的位数会填充零）

图1（宽度为6显示6位数字）

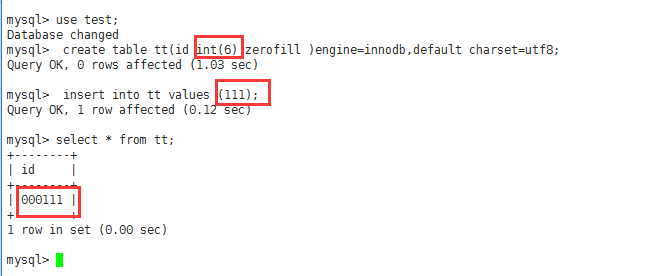
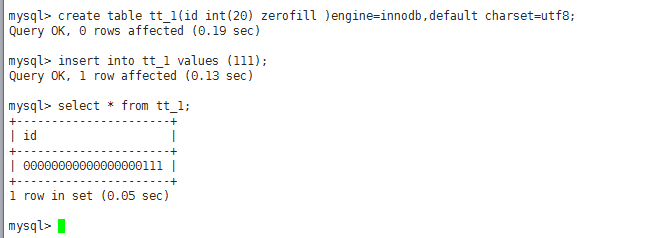


图2（宽度为20显示20位数字）



# 12 MySQL数据类型和Java数据类型映射

## 12.1 bigint

原文链接：<https://blog.csdn.net/u014450159/article/details/79435923>

bigint(20) 有符号（MySQL默认的数据类型是有符号）的数据范围是-9223372036854775808~9223372036854775807，与Java.lang.Long的取值范围一致，说明“bigint(20) 有符号”可以映射成Java.lang.Long，但超出此范围的就只能映射成Java.math.BigInteger。

# 13 自查询

原文链接：<https://blog.csdn.net/wguangliang/article/details/50167283>

图1



# 14 isnull

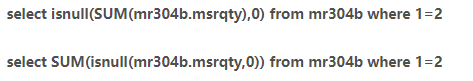
## 14.1 isnull和sum的顺序

原文链接：<https://bbs.csdn.net/topics/391858355?page=1>

isnull在外面是先求和，再判断是否为null；isnull在里面是先判断是否为null，再求和。

（注意：isnull在里面是考虑某条记录的该列值为null的情况；isnull在外面是考虑到空表的情况）

图1



# 15 MySQL版本导致的问题

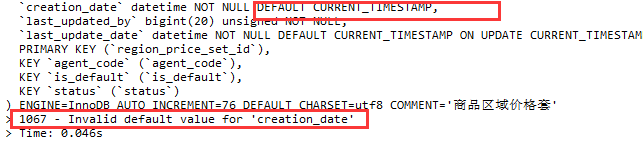
## 15.1 CURRENT\_TIMESTAMP

### 15.1.1 使用CURRENT\_TIMESTAMP赋默认值的问题

原文链接：<https://blog.csdn.net/u011983531/article/details/71305212>

MySQL 5.6之后才可以用CURRENT\_TIMESTAMP为datetime类型赋默认值，否则会抱错。

图1



# 16 有难度的SQL

## 16.1 分组内取前N

原文链接：<https://blog.csdn.net/wzy_1988/article/details/52871636>

法1：利用自关联，同一个部门下（e1.dept\_name = e.dept\_name），自表1的薪水>自表2的薪水（e1.salary > e.salary），取前N。

图1

|  |
| --- |
| SELECT  \*  FROM  t\_user AS e  WHERE  (  SELECT  count( DISTINCT ( e1.salary ) )  FROM  t\_user AS e1  WHERE  e1.dept\_name = e.dept\_name  AND e1.salary > e.salary  ) < 2; |

## 16.2 组内排序

组内排序需要先order by再group by。因为group by默认取分组内哪一条记录是随机的，但通过先排序再group by就相当于指定了要取分组内的哪一条记录。

组内排序可以用于获取分组XXX第一条的记录。

图1

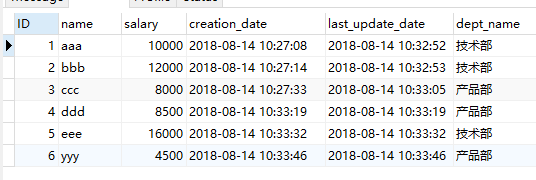


图2（所以这种写法是组外排序，并不是组内排序）

（注意：group by的结果是每个group by字段（dept\_name）只剩下一条记录，至于是该dept\_name的哪一条是随机的）

|  |
| --- |
| select \* from t\_user group by dept\_name order by salary desc |

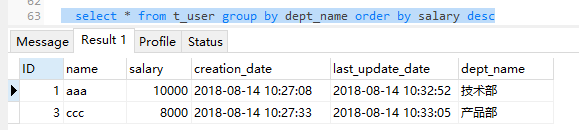
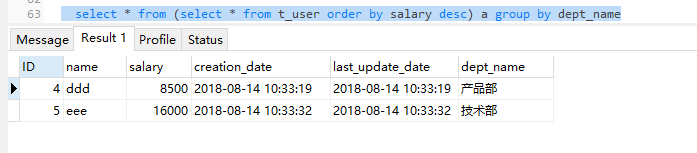


图3（组内排序）

|  |
| --- |
| select \* from (select \* from t\_user order by salary desc) a group by dept\_name |



# 17 select语句的正确执行顺序

原文链接：<https://blog.csdn.net/wzy_1988/article/details/52871636>

先group by，再select，最后再order by。

（注意：因为order by执行顺序在group by之后，所以order by是对group by执行后的全部行进行的排序）

图1（原SQL格式）

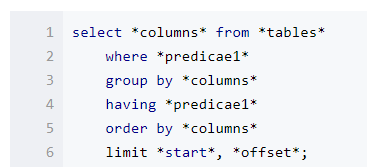


图2（正确执行顺序）

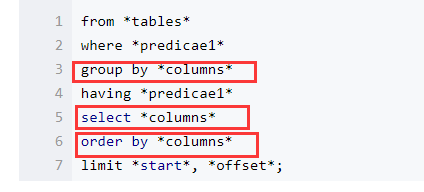


图3（所以这种写法是组外排序，并不是组内排序）



# 18 范式

## 18.1 第一范式

第一范式：一列不能存2个值。

（注意：关系型数据库一定满足第一范式）

图1（错误，联系方式那一列存了2个值）

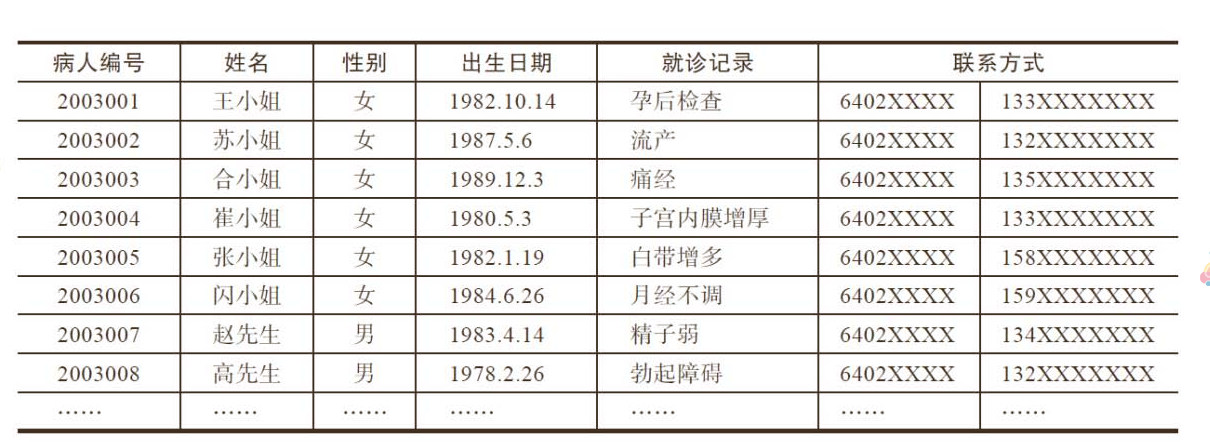
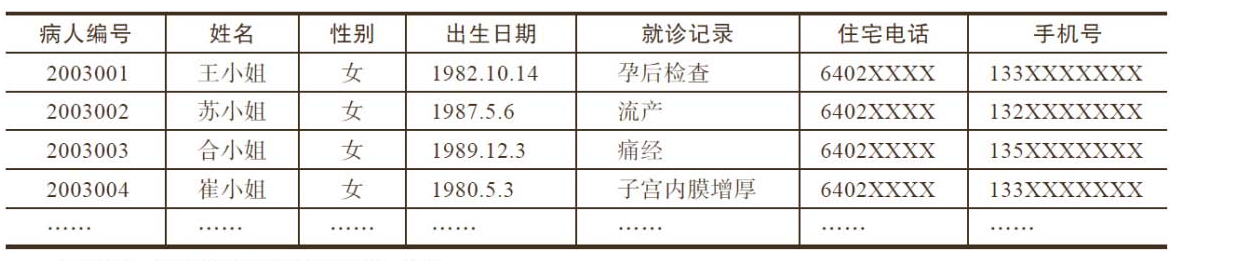


图2（正确）



## 18.2 第二范式

非主键列必须依赖主键。

（注意：第二范式是可以非主键列传递依赖主键）

（注意：不能依赖主键的一部分）

## 18.3 第三范式

非主键列必须直接依赖主键。

（注意：第三范式不能传递依赖主键，即不能：非主键列A依赖于非主键列B，非主键列B依赖于主键列）

图1

（错误，非主键列科室名称依赖于非主键列科室编号，非主键列科室编号依赖于主键医生编号）



图2（正确，非主键列科室名称直接依赖于主键列科室编号）



### 18.3.1 第三范式的缺点

第三范式的缺点是查询时要join很多表，导致查询效率很低。所以有时候出于性能考虑，需要有意违反第三范式。

# 19 字段类型的选择

字段类型选择的一般原则是：能用占用字节小的字段类型，就不用占用字节大的字段类型。因为小字段即提高查询效率（定位4字节字段比定位32字节字段快），也节省存储空间。

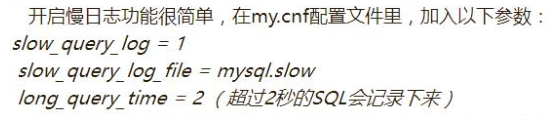
## 19.1 enum字段类型的缺点

enum字段类型不便于拓展。比如用户在线状态：0表示离线，1表示在线，2表示离开，3表示忙碌，4表示隐身，此时如果用enum字段类型没问题。但如果后期拓展了：5表示请勿打扰，6表示开会中，7表示隐身对好友可见，但这时候就需要更改字段类型。而用tinyint就不需要更改字段类型。

# 20 开启慢日志

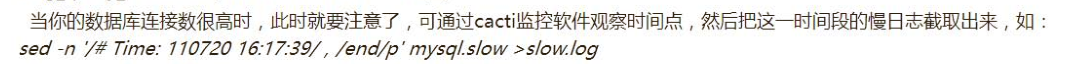
1. 开启慢日志功能，并将慢日志写入指定文件中。

图1



1. 截图某个时间段的慢日志。

图2



1. 取出耗时最长的前10条慢SQL进行分析。

图3

