MySQL训练营-MySQL优化总结

本内容属于动脑学院Steven老师首创,如需转载,请注明出处!

Steven QQ : 750954883

动脑学院系列课程咨询: QQ 2729772006

硬件和操作系统的优化

这部分的优化大多数属于我们DBA的主要工作.我们可以作为一个了解

CPU的选择

cpu的两个关键因素：核数、主频

根据不同的业务类型进行选择：

cpu密集型：计算比较多，OLTP 主频很高的cpu、核数还要多

IO密集型：查询比较，OLAP 核数要多，主频不一定高的

更大的内存

我们已经知道Innodb buffer pool会保存大量访问过的数据和索引.

所以我们如果有更大的内存来存储更多的缓存数据和索引信息,我们即可以减少IO的耗时操作

1、OLAP类型数据库，需要更多内存，和数据获取量级有关。

2、OLTP类型数据一般内存是cpu核心数量的2倍到4倍.

更快的磁盘存储

使用磁盘阵列或者SSD 进行数据存储.

优化网络,优化操作系统

关闭DNS的域名解析建议关闭.

将MySQL服务置于核心区 [接入区,DMZ区,核心区]

修改操作系统的网络配置

选择64位的操作系统.

配置足够大的文件句柄打开数量.等等

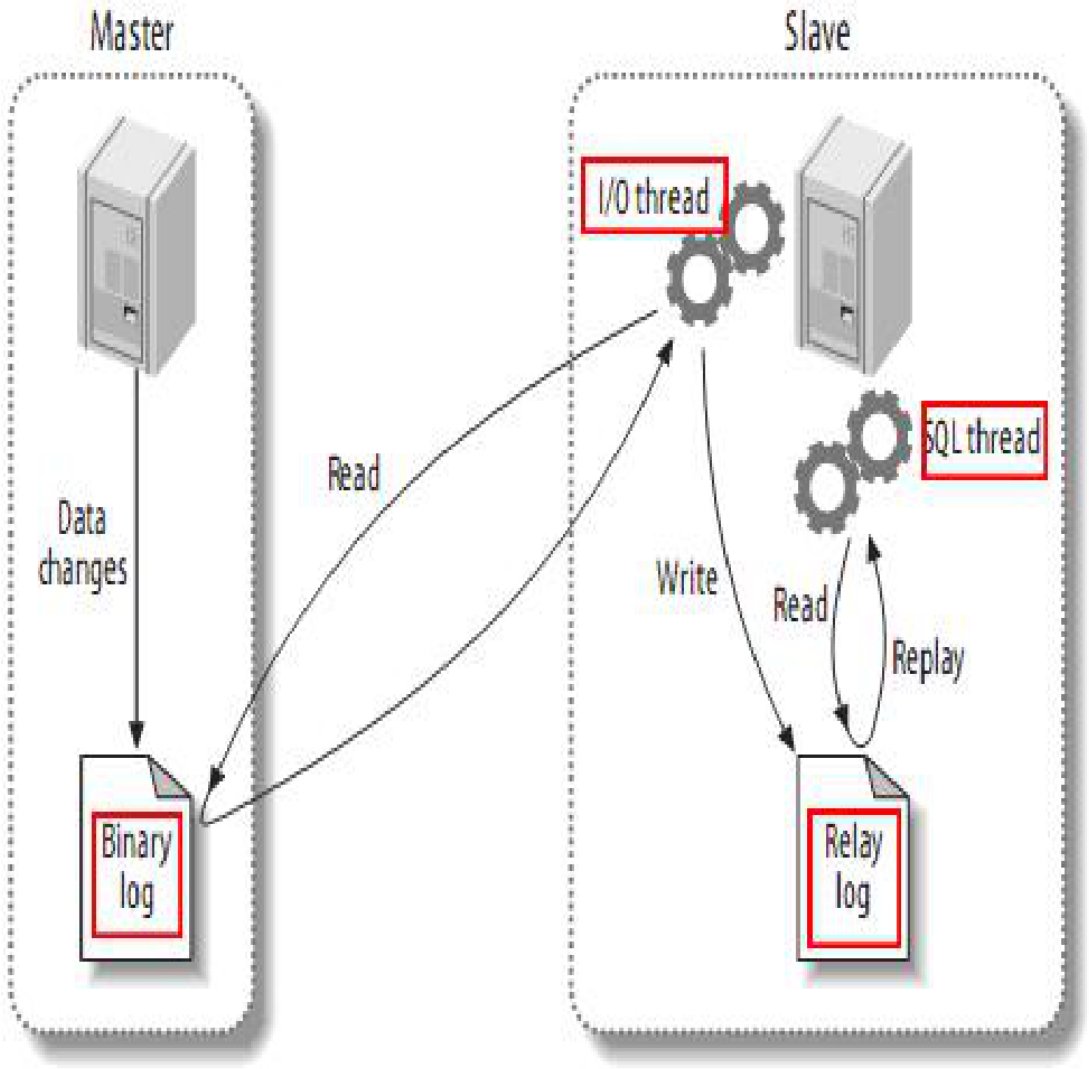
架构的优化

设计缓存服务

我们可以将热点数据进行缓存降低MySQL访问压力.

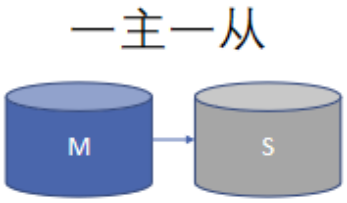
实现读写分析,主从复制

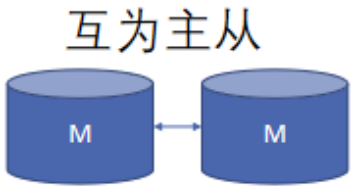
主从复制

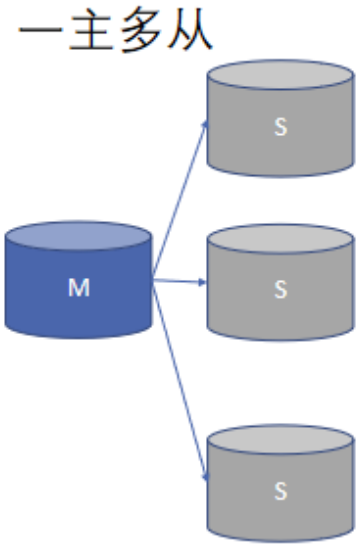


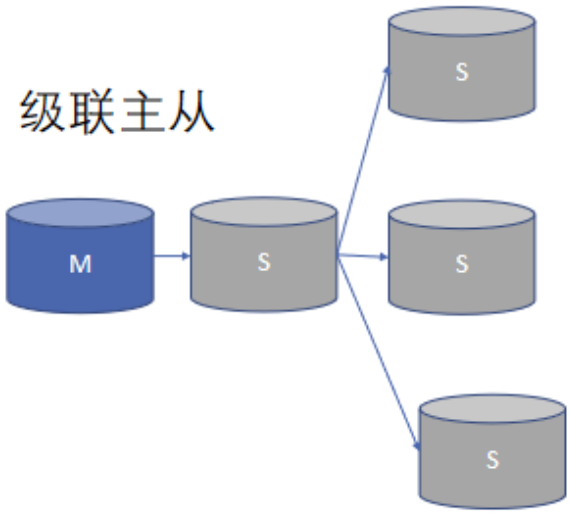
- master将操作记录到二进制日志(binary log)中（这些记录叫做二进制日志事件，binary log events）
- Slave通过I/O Thread异步将master的binary log events拷贝到它的中继日志(relay log)；
- Slave执行relay日志中的事件，匹配自己的配置将需要执行的数据，在slave服务上执行一遍从而达到复制数据的目的。

主从复制的常见形态



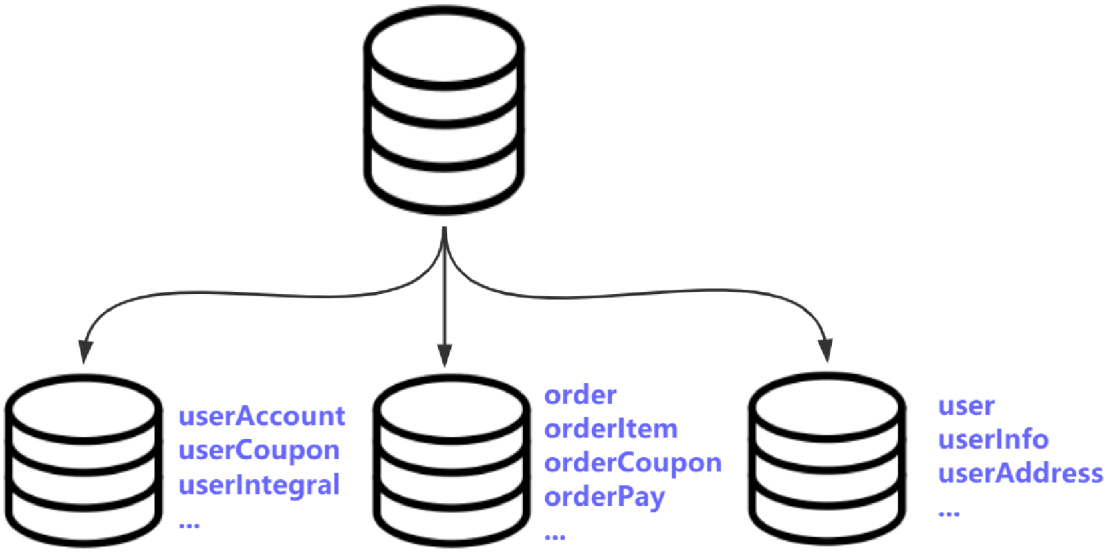






合理的业务拆分,分库分表

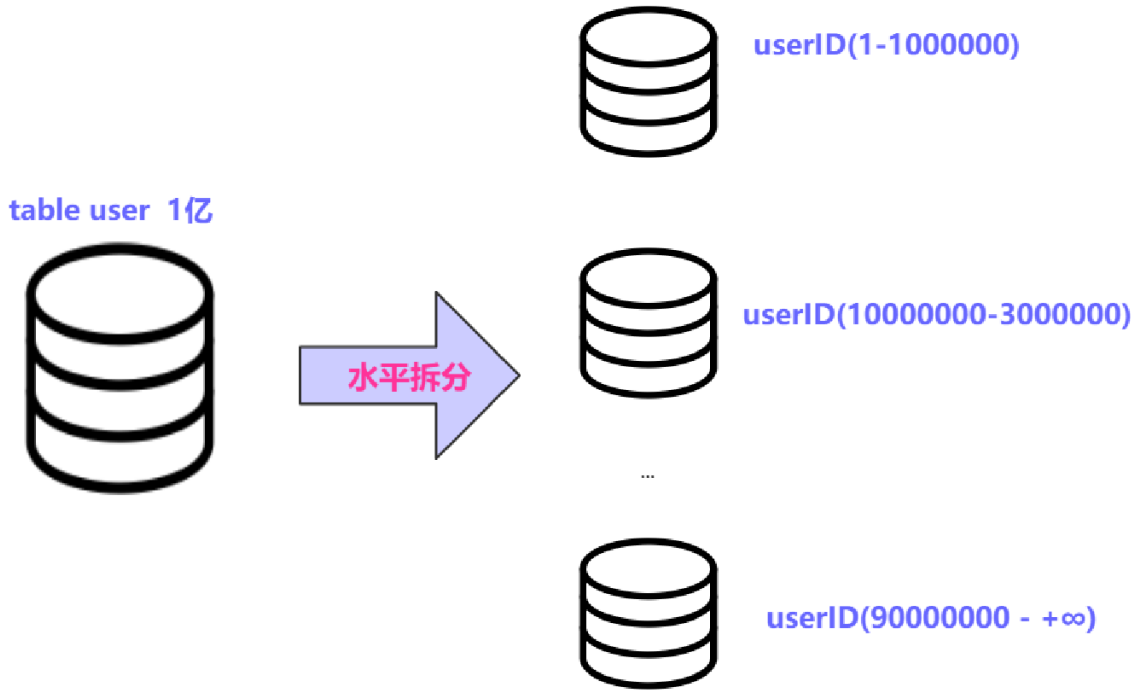
垂直拆库



将一个大库按照一个的业务关联度和模块分解划分成若干个小库

缓解单库应对多个业务系统的压力.

水平分表



针对单表数据量大解决单表性能瓶颈.

MySQL的配置优化

MySQL的配置概览

https://dev.mysql.com/doc/refman/5.7/en/server-system-variables.html

参见官方文档.

MySQL的配置参数特点

- 作用域范围
  - global
  - session
- 是否可热加载

MySQL配置文件

课通过`mysql --help`查看帮助文档.

Default options are read from the following files in the given order: /etc/my.cnf /etc/mysql/my.cnf /usr/etc/my.cnf ~/.my.cnf

核心常见配置说明

`max_connections` 客户端的最大连接数

`port` MySQL监听的服务端口

`datadir` 数据存放目录

`bin-log` binlog日志文件存放路径

`tmp_table_size=16M` 临时表大小限定

`bind-address` =0.0.0.0 限制访问IP

`sort_buffer_size` 排序缓冲区大小

建议256K(默认值)-> 2M之内 当查询语句中有需要文件排序功能时，马上为connection分配配置的内存大小

`join_buffer_size` 关联查询缓冲区大小

建议256K(默认值)-> 1M之内 当查询语句中有关联查询时，马上分配配置大小的内存用这个关联查询，所以有可能在一个查询语句中会分配很多个关联查询缓冲区

`Innodb_buffer_pool_size` innodb缓冲区大小

大的缓冲池可以减小多次磁盘I/O访问相同的表数据以提高性能

`innodb_write_io_threads` `innodb_read_io_threads`

innodb使用后台线程处理innodb缓冲区数据页上的读写 I/O(输入输出)线程数量.

库表设计优化及SQL的优化

数据库表设计

数据库的三范式

第一范式（1NF）： 字段具有原子性,不可再分。 所有关系型数据库系统都满足第一范式）数据库表中的字段都是单一属性的， 不可再分；

第二范式（2NF）： 要求实体的属性完全依赖于主键。 所谓完全依赖是指不能存在仅依赖主键一部分的属性， 如果存在， 那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体， 新实体与原实体之间是一对多的关系。为实现区分通常需要为表加上一个列，以存储各个实例的惟一标识。 简而言之， 第二范式就是属性完全依赖主键。

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信息。

三大范式简单理解

1. 每一列只有一个单一的值，不可再拆分
2. 每一行都有主键能进行区分
3. 利用表关联的方式解决数据关联问题

强制依赖满足三大范式的问题

- 充分的满足第一范式设计将为表建立太量的列 数据从磁盘到缓冲区，缓冲区脏页到磁盘进行持久的过程中，列的数量过多会导致性能下降。过多的列影响转换和持久的性能

- 过分的满足第三范式化造成了太多的表关联 表的关联操作将带来额外的内存和性能开销
- 使用innodb引擎的外键关系进行数据的完整性保证 外键表中数据的修改会导致Innodb引擎对外键约束进行检查，就带来了额外的开销

```
SELECT
    *
FROM
     consume_detail
WHERE
    (convert((price_deal * 100 - price_cost * 100) , SIGNED) -
convert(coupon_price*100,SIGNED))  > 0
```

上面的案例的主体意思是,查询售出没有亏损的商品.

这一条SQL.语句如果我们在表中,创建price_deal price_cost coupon_price 都没有效果,在函数中使用索引字段并不会使用索引扫描.

所以我们可以在表设计阶段,设计一个独立的字段如 profit .然后再商品成交时将此字段的值计算出来,存储进去.我们在查询时,既可以在profit字段上建立索引.加速查询.

SQL优化三步曲

慢SQL的定位和排查

慢SQL的定位方法

- 业务驱动
- 测试驱动
- 慢查询日志

慢查询日志配置

```
show variables  like 'slow_query_log'

set global slow_query_log = on

set global slow_query_log_file = '/var/lib/mysql/dnedu-slow.log'

set global log_queries_not_using_indexes = on

set global long_query_time  = 0.1  (秒)
```

```
Time                 Id Command    Argument
# Time: 2020-03-26T09:12:13.445118Z
# User@Host: root[root] @ localhost []  Id:    20
# Query_time: 0.064851  Lock_time: 0.000219 Rows_sent: 100001  Rows_examined: 100001
use db_user;
SET timestamp=1585213933;
select * from user_innodb;
# Time: 2020-03-26T09:12:17.312422Z
# User@Host: root[root] @ localhost []  Id:    20
# Query_time: 0.000871  Lock_time: 0.000056 Rows_sent: 2000  Rows_examined: 2000
SET timestamp=1585213937;
select * from user_innodb limit 2000;
```

Time： 日志记录的时间

User@Host：执行的用户及主机

Query_time：查询耗费时间 Lock_time 锁表时间 Rows_sent 发送给请求方的记录条数 Rows_examined 语句扫描的记录条数

SET timestamp 语句执行的时间点

select .... 执行的具体语句

mysqldumpslow -t 10 -s at xxx.log

```
[root@gupaoedu ~]# mysqldumpslow --help
Usage: mysqldumpslow [ OPTS... ] [ LOGS... ]

Parse and summarize the MySQL slow query log. Options are

  --verbose    verbose
  --debug      debug
  --help       write this text to standard output

  -v           verbose
  -d           debug
  -s ORDER     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
```

mysqldumpslow -t 10 -s at {slow}.log

```
[root@localhost mysql]# cat /var/lib/mysql/dnedu-slow.log
/usr/sbin/mysqld, Version: 5.7.26 (MySQL Community Server (GPL)). started with:
Tcp port: 3306  Unix socket: /var/lib/mysql/mysql.sock
Time                 Id Command    Argument
# Time: 2020-03-26T09:12:13.445118Z
# User@Host: root[root] @ localhost []  Id:    20
# Query_time: 0.064851  Lock_time: 0.000219 Rows_sent: 100001  Rows_examined: 100001
use db_user;
SET timestamp=1585213933;
select * from user_innodb;
# Time: 2020-03-26T09:12:17.312422Z
# User@Host: root[root] @ localhost []  Id:    20
# Query_time: 0.000871  Lock_time: 0.000056 Rows_sent: 2000  Rows_examined: 2000
SET timestamp=1585213937;
select * from user_innodb limit 2000;
[root@localhost mysql]# mysqldumpslow -t 10 -s at /var/lib/mysql/dnedu-slow.log

Reading mysql slow query log from /var/lib/mysql/dnedu-slow.log
Count: 1  Time=0.06s (0s)  Lock=0.00s (0s)  Rows=100001.0 (100001), root[root]@localhost
  select * from user_innodb

Count: 1  Time=0.00s (0s)  Lock=0.00s (0s)  Rows=2000.0 (2000), root[root]@localhost
  select * from user_innodb limit N

Died at /usr/bin/mysqldumpslow line 161, <> chunk 2.
```

show profile工具

Show Profile是MySQL提供的可以用来分析当前会话中SQL语句执行的资源消耗情况的工具，可用于SQL调优的测量。在当前会话中.默认情况下处于show profile是关闭状态，打开之后保存最近15次的运行结果

`show variables like 'profiling'` 查看当前会话状态

`set profiling = on;` 打开当前会话的show profile功能.

可通过`show profiles;` 查看最近15条sql的运行情况

```
mysql> show profiles;
+----------+------------+-----------------------------------------------------------------+
| Query_ID | Duration   | Query                                                           |
+----------+------------+-----------------------------------------------------------------+
|        5 | 0.00074800 | select count(*) from user_innodb where name like 'hello-12%'    |
|        6 | 0.00588725 | select count(*) from user_innodb where name like 'hello-1%'     |
|        7 | 0.04411125 | select count(*) from user_innodb where name like 'hello-%'      |
|        8 | 0.00050525 | select count(*) from user_innodb where name like 'hello-1234%'  |
|        9 | 0.03647900 | select count(*) from user_innodb where name like 'hello-%'      |
|       10 | 0.00029625 | select count(*) from user_innodb where name like 'hello-12321%' |
|       11 | 0.00475625 | select count(*) from user_innodb where name like 'hello-1%'     |
|       12 | 0.00541275 | select count(*) from user_innodb where name like 'hello-2%'     |
|       13 | 0.00420025 | select count(*) from user_innodb where name like 'hello-3%'     |
|       14 | 0.00444300 | select count(*) from user_innodb where name like 'hello-4%'     |
|       15 | 0.01978050 | select count(*) from user_innodb limit 1000                     |
|       16 | 0.00110925 | select * from user_innodb limit 1000                            |
|       17 | 0.00094550 | select * from user_innodb limit 1000                            |
|       18 | 0.00095575 | select * from user_innodb limit 1002                            |
|       19 | 0.00023800 | select * from user_innodb limit 103                             |
+----------+------------+-----------------------------------------------------------------+
15 rows in set, 1 warning (0.00 sec)
```

可通过查询具体某一个查询的ID 进行详细信息查看.

`show profile for query 15;`

```
mysql> show profile for query 15
    -> ;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| starting             | 0.000153 |
| checking permissions | 0.000010 |
| Opening tables       | 0.000013 |
| init                 | 0.000011 |
| System lock          | 0.000006 |
| optimizing           | 0.000004 |
| statistics           | 0.000010 |
| preparing            | 0.000009 |
| executing            | 0.000003 |
| Sending data         | 0.019377 |
| end                  | 0.000016 |
| query end            | 0.000008 |
| closing tables       | 0.000007 |
| freeing items        | 0.000017 |
| cleaning up          | 0.000138 |
+----------------------+----------+
15 rows in set, 1 warning (0.00 sec)
```

`show profile cpu,block io for query 15;`

```
mysql> show profile cpu,block io for query 15;
+----------------------+----------+----------+------------+--------------+---------------+
| Status               | Duration | CPU_user | CPU_system | Block_ops_in | Block_ops_out |
+----------------------+----------+----------+------------+--------------+---------------+
| starting             | 0.000153 | 0.000054 |   0.000095 |            0 |             0 |
| checking permissions | 0.000010 | 0.000003 |   0.000005 |            0 |             0 |
| Opening tables       | 0.000013 | 0.000004 |   0.000008 |            0 |             0 |
| init                 | 0.000011 | 0.000004 |   0.000007 |            0 |             0 |
| System lock          | 0.000006 | 0.000003 |   0.000004 |            0 |             0 |
| optimizing           | 0.000004 | 0.000001 |   0.000002 |            0 |             0 |
| statistics           | 0.000010 | 0.000004 |   0.000007 |            0 |             0 |
| preparing            | 0.000009 | 0.000003 |   0.000005 |            0 |             0 |
| executing            | 0.000003 | 0.000001 |   0.000002 |            0 |             0 |
| Sending data         | 0.019377 | 0.018289 |   0.000000 |            0 |             0 |
| end                  | 0.000016 | 0.000009 |   0.000000 |            0 |             0 |
| query end            | 0.000008 | 0.000008 |   0.000000 |            0 |             0 |
| closing tables       | 0.000007 | 0.000007 |   0.000000 |            0 |             0 |
| freeing items        | 0.000017 | 0.000017 |   0.000000 |            0 |             0 |
| cleaning up          | 0.000138 | 0.000140 |   0.000000 |            0 |             0 |
+----------------------+----------+----------+------------+--------------+---------------+
15 rows in set, 1 warning (0.00 sec)
```

show profile的常用查询参数。

- ALL：显示所有的开销信息。
- BLOCK IO：显示块IO开销。
- CONTEXT SWITCHES：上下文切换开销。
- CPU：显示CPU开销信息。
- IPC：显示发送和接收开销信息。
- MEMORY：显示内存开销信息。
- PAGE FAULTS：显示页面错误开销信息。
- SOURCE：显示和Source_function，Source_file，Source_line相关的开销信息。
- SWAPS：显示交换次数开销信息。

执行计划分析

分析SQL的执行计划,优化SQL语句

https://dev.mysql.com/doc/refman/5.7/en/select-optimization.html 具体的优化过程.

常见SQL优化的规则

- SQL的查询一定要基于索引完成SQL的结果集扫描
- 避免索引列上使用函数或者运算,这样会导致索引失效
- where 字句中like %号,尽量放置在右边
- 使用索引扫描,联合索引中的列从左往右,命中越多越好.
- 尽可能使用SQL语句用到的索引完成排序,避免使用 文件排序的方式
- 查询有效的列信息即可.少用 * 代替列信息
- 永远用小结果集驱动大结果集

  以小结果集驱动能减少循环次数，从而减少对被驱动结果集的访问，从而减少被驱动表的锁定

  执行时会以最左边的表为基础表循环与右边表数据做笛卡尔积

  ```
  SELECT * FROM a JOIN b ON a.id=b.id LEFT JOIN c ON c.id=a.id
  ```