# 文档历史

|  |  |  |  |
| --- | --- | --- | --- |
| 修订日期 | 修订内容 | 修订版本 | 修订人 |
| 2014/11/12 | 文档创建 | V1.0 | Kerl |
| 2014/12/29 | 文档修改 | V1.1 | Kerl |
|  |  |  |  |

# 目的

说明MYSQL的一些基本优化方法和概念，以便开发的培训和设计参考

# 使用

用户人员培训和工作指导手册

# SQL追踪

开启MYSQL SLOW LOG

SET GLOBAL slow\_query\_log = 1

## 设置long\_query\_time = 0

SET GLOBAL long\_query\_time = 0(如果确认是SESSION级的操作，可不加global)

查看MYSQL的SLOW 位置show global variables like 'slow\_query\_log\_file';

## 跟踪slow\_query\_log\_file文件

# User@Host: s8\_bbs\_user[s8\_bbs\_user] @ [10.1.0.104]

# Thread\_id: 1444568313 Schema: s8\_bbs Last\_errno: 0 Killed: 0

# Query\_time: 3.754404 Lock\_time: 0.000032 Rows\_sent: 10 Rows\_examined: 1380748 Rows\_affected: 0 Rows\_read: 10

# Bytes\_sent: 925 Tmp\_tables: 0 Tmp\_disk\_tables: 0 Tmp\_table\_sizes: 0

# InnoDB\_trx\_id: 54C2454B4

SET timestamp=1415853009;

SELECT subject,tid,fid,lastpost FROM pw\_threads WHERE (fid = '2' ) AND (inspect <> '' ) ORDER BY postdate desc LIMIT 10;

# 单SQL的资源消耗

## 使用set profiling = 1(session 级别)

在SHOW PROFILES结果中查看query\_id， 可在SHOW PROFILE FOR QUERY 1 中 查看到整个SQL在各个部分消耗的时间。

## 分析用的SQL:

SET @query\_id=1;

SELECT STATE, SUM(DURATION) AS Total\_R, ROUND( 100 \* SUM(DURATION) / (SELECT SUM(DURATION) FROM INFORMATION\_SCHEMA.PROFILING WHERE QUERY\_ID = @query\_id ), 2) AS Pct\_R, COUNT(\*) AS Calls, SUM(DURATION) / COUNT(\*) AS "R/Call" FROM INFORMATION\_SCHEMA.PROFILING WHERE QUERY\_ID = @query\_id GROUP BY STATE ORDER BY Total\_R DESC;

可以查询分析, 结果中send data的时间比较难以判断后优化，因为其中包含了很多动作

# MYSQL的索引使用

## MyISAM和InnoDB索引介绍：

* MyISAM indexes 的指向包含匹配数据行的物理存储地址，压缩索引
* InnoDB indexes 的指向包含匹配数据行的主键值，不压缩索引
* MyISAM的主键仅仅是NO NULL UNIQ 索引，按插入顺序排列存储（下图为MyISAM叶子节点存储数据的方式）

CREATE TABLE layout\_test (

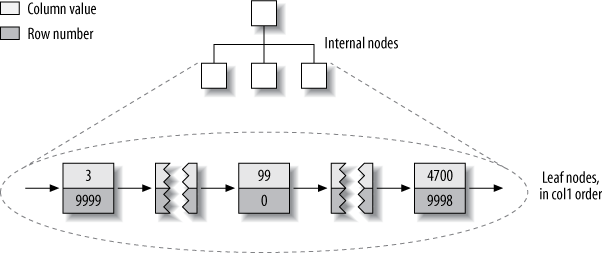
col1 int NOT NULL,

col2 int NOT NULL,

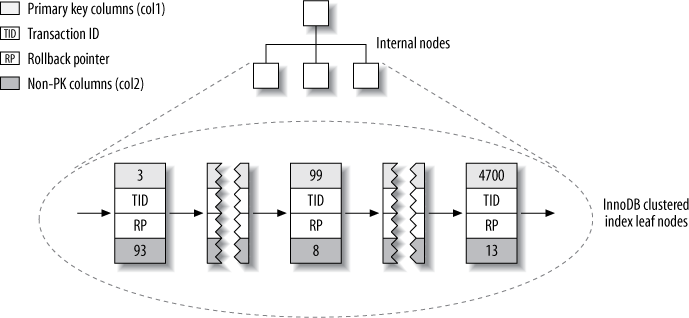
PRIMARY KEY(col1),

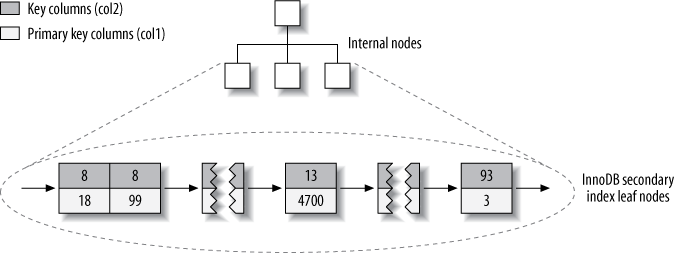
KEY(col2)

);

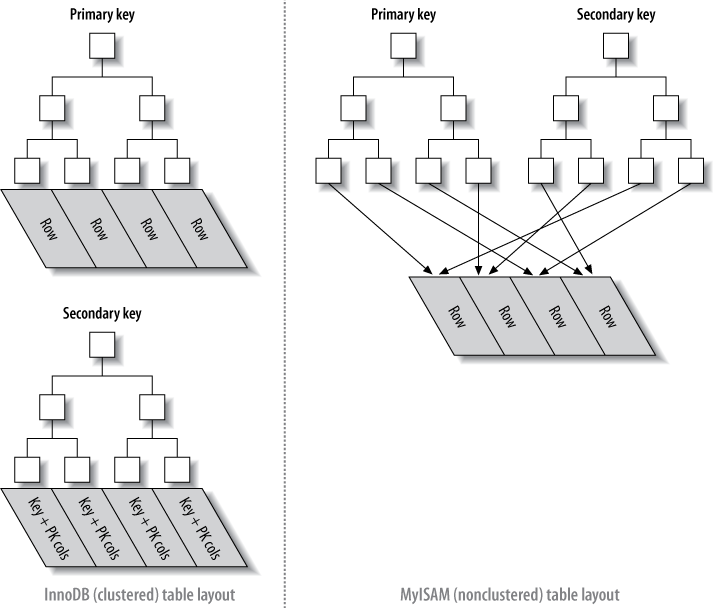


* InnoDB的主键（唯一键）会被选择为Clustered Indexes， 按主键值得序列排列存储，包含整行数据，（下图为InnoDB叶子节点存储数据的方式）





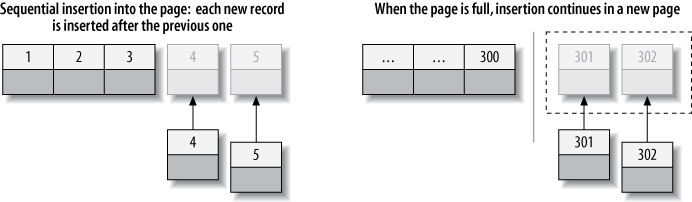
* 两种ENGINE的键值存储结构



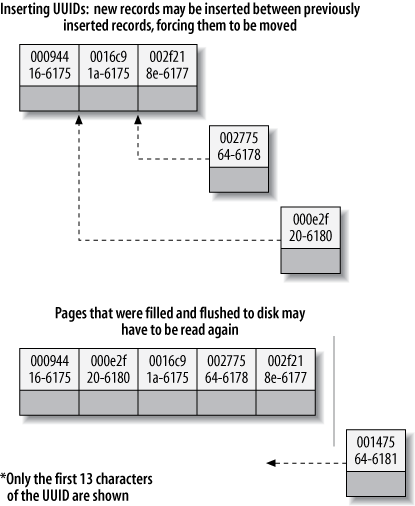
## InnoDB主键有序的重要性

无序主键会导致INSET 和 JOIN 的效率大打折扣，因每次插入数据INNODB都会重新寻找序列，导致PAGE结构杂乱，并伴随大量 split pages

* 有序主键（自增）的数据存储结构



* 无序主键（UUID）的数据存储结构



## B-Tree 索引的使用: 自左向右的原则

Mysql> explain select \* from pw\_pinglog where fid = 158 and tid like '37694%'\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. Row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

           id: 1

  select\_type: SIMPLE

        table: pw\_pinglog

         type: ref

possible\_keys: tid,fid

          key: fid

      key\_len: 2

          ref: const

         rows: 50304

        Extra: Using where

mysql> explain select \* from pw\_pinglog where fid = 158 and tid like '37694%' and pid = 0\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

           id: 1

  select\_type: SIMPLE

        table: pw\_pinglog

         type: ref

possible\_keys: tid,pid,fid

          key: fid

      key\_len: 2

          ref: const

         rows: 50304

        Extra: Using where

## MYSQL  order by limit 的优化

* 一定使用索引

    SELECT \* FROM table WHERE A = ? AND B = ? ORDER BY  C

    KEY(A,B,……，C）

* 避免使用表达式order by
* 在JOIN中，尽量使用主表order by
* order by 多个字段时要保持相同的ASC/DESC
* limit 大值时，使用反向sort来减少扫描区间： SELECT \* FROM table ORDER BY  date ASC limit 1000,20改写成SELECT \* FROM table ORDER BY  date desc limit 0,20
* 必要时使用FORCE INDEX, MYSQL的索引优化策略是：Selectivity优先

## Index Selectivity

对于B-TREE索引，最高的Selectivity为1，即索引字段值唯一。每次查询扫描行数 < 全表行数的15%为佳

## Prefix Indexes

对长字段做部分索引达到一样的Selectivity

mysql> SELECT COUNT(\*) AS cnt, city  FROM sakila.city\_demo GROUP BY city ORDER BY cnt DESC LIMIT 10;

+-----+----------------+

| cnt | city |

+-----+----------------+

| 65 | London |

| 49 | Hiroshima |

| 48 | Teboksary |

| 48 | Pak Kret |

| 48 | Yaound |

| 47 | Tel Aviv-Jaffa |

| 47 | Shimoga |

| 45 | Cabuyao |

| 45 | Callao |

| 45 | Bislig |

+-----+----------------+

mysql> SELECT COUNT(\*) AS cnt, LEFT(city, 3) AS pref FROM sakila.city\_demo GROUP BY pref ORDER BY cnt DESC LIMIT 10;

+-----+------+

| cnt | pref |

+-----+------+

| 483 | San |

| 195 | Cha |

| 177 | Tan |

| 167 | Sou |

| 163 | al- |

| 163 | Sal |

| 146 | Shi |

| 136 | Hal |

| 130 | Val |

| 129 | Bat |

+-----+------+

mysql> SELECT COUNT(DISTINCT city)/COUNT(\*) FROM sakila.city\_demo;

+-------------------------------+

| COUNT(DISTINCT city)/COUNT(\*) |

+-------------------------------+

| 0.0312 |

+-------------------------------+

mysql> SELECT COUNT(\*) AS cnt, LEFT(city, 7) AS pref FROM sakila.city\_demo GROUP BY pref ORDER BY cnt DESC LIMIT 10;

+-----+---------+

| cnt | pref |

+-----+---------+

| 70 | Santiag |

| 68 | San Fel |

| 65 | London |

| 61 | Valle d |

| 49 | Hiroshi |

| 48 | Teboksa |

| 48 | Pak Kre |

| 48 | Yaound |

| 47 | Tel Avi |

| 47 | Shimoga |

+-----+---------+

mysql> SELECT COUNT(DISTINCT LEFT(city, 3))/COUNT(\*) AS sel3, COUNT(DISTINCT LEFT(city, 4))/COUNT(\*) AS sel4, COUNT(DISTINCT LEFT(city, 5))/COUNT(\*) AS sel5, COUNT(DISTINCT LEFT(city, 6))/COUNT(\*) AS sel6, COUNT(DISTINCT LEFT(city, 7))/COUNT(\*) AS sel7 FROM sakila.city\_demo;

+--------+--------+--------+--------+--------+

| sel3 | sel4 | sel5 | sel6 | sel7 |

+--------+--------+--------+--------+--------+

| 0.0239 | 0.0293 | 0.0305 | 0.0309 | 0.0310 |

## 索引的长度控制

767 bytes (up to 3072 bytes)，可调节但不建议

## Covering Indexes（INNODB）

查询的结果值包含在索引中，避免再次的数据抓取(因为Innodb索引都包含主键的值，查询结果里包含主键也可使用这种优化)

CREATE TABLE `pw\_threads` (

  PRIMARY KEY (`tid`),

  KEY `fid\_topped\_lastpost\_postdate\_key` (`fid`,`topped`,`lastpost`,`postdate`)

) ENGINE=InnoDB AUTO\_INCREMENT=4469374 DEFAULT CHARSET=utf8

mysql> explain select tid,`topped`,`lastpost`,`postdate` from pw\_threads where fid = 13\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

           id: 1

          select\_type: SIMPLE

          table: pw\_threads

          type: ref

          key: fid\_topped\_lastpost\_postdate\_key

          key\_len: 2

          ref: const

         rows: 2255

        Extra: Using index

# MyISAM和InnoDB的优劣

## MyISAM缓存的是KEY，不缓存数据

## InnoDB缓存的是数据

## MyISAM更适合变更操作较少的数据

例如日志表，并且MyISAM的批量插入效率要比InnoDB高

## InnoDB使用recode lock+gap lock+next lock 的混合方式来控制行级别事务

## WHERE 语句后条件字段的独立性

# 数据类型的选择

## 数值类型在MYSQL中不需要指定长度，对空间无影响

## 对已知长度的字符型数据用char比varchar有优势

空间的节省（N+2），高效的索引

## 日期类型在MYSQL中的自动初始化和变更

（5.6以后datetime 类型也同样适用此方法）

* col\_name TIMESTAMP DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP
* col\_name TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

## IP数据用INT类型存储，使用函数

        mysql> SELECT INET\_ATON('10.0.5.9'); -> 167773449

        mysql> SELECT INET\_NTOA(167773449); -> '10.0.5.9'

# 技巧和问题

## 对于字符型的数据查询使用CRC32 函数进行加速

例如用户登录名，邮箱等

## Optimizing SQL\_CALC\_FOUND\_ROWS

MYSQL的分页逻辑是查找到索要行后即停止，但加入SQL\_CALC\_FOUND\_ROWS后迫使MYSQL对结果集外的数据做计算

* 使用limit N+1 方式判断下一页是否存在（即时查询，忽略总页数，数据变更频率很高）
* 先使用COUNT(1)预设总行数，在此范围内查询（对即时要求不高，数据变更频率较低）

## 计数表的设计

牺牲SELECT换高效UPDATE，减少mutex（ 0 <= RAND() < 1.0）（ R取值范围为：i <= R < j 的公式： FLOOR(i + RAND() \* (j – i))）

* 计数表结构：

mysql> CREATE TABLE daily\_hit\_counter (

 day date not null,

 slot tinyint unsigned not null,

 cnt int unsigned not null,

 primary key(day, slot)

) ENGINE=InnoDB;

* 计数的方式：

INSERT INTO daily\_hit\_counter(day, slot, cnt) VALUES(CURRENT\_DATE, RAND() \* 100, 1)  ON DUPLICATE KEY UPDATE cnt = cnt + 1;

* 统计的方式：

SELECT SUM(cnt) FROM hit\_counter;

## group\_concat 函数的使用

列转行

mysql> SELECT column\_name from information\_schema.columns where table\_name = 'pw\_threads';

+-------------+

| column\_name |

+-------------+

| tid |

| fid |

| icon |

| titlefont |

| author |

| authorid |

| subject |

| toolinfo |

| toolfield |

| ifcheck |

| type |

| postdate |

| lastpost |

| lastposter |

| hits |

| replies |

| favors |

| modelid |

| shares |

| topped |

| topreplays |

| locked |

| digest |

| special |

| state |

| ifupload |

| ifmail |

| ifmark |

| ifshield |

| anonymous |

| dig |

| fight |

| ptable |

| ifmagic |

| ifhide |

| inspect |

| tpcstatus |

| ding |

+-------------+

38 rows in set (0.02 sec)

mysql> SELECT GROUP\_CONCAT(column\_name) from information\_schema.columns where table\_name = 'pw\_threads'\G

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

GROUP\_CONCAT(column\_name): tid,fid,icon,titlefont,author,authorid,subject,toolinfo,toolfield,ifcheck,type,postdate,lastpost,lastposter,hits,replies,favors,modelid,shares,topped,topreplays,locked,digest,special,state,ifupload,ifmail,ifmark,ifshield,anonymous,dig,fight,ptable,ifmagic,ifhide,inspect,tpcstatus,ding

1 row in set (0.00 sec)