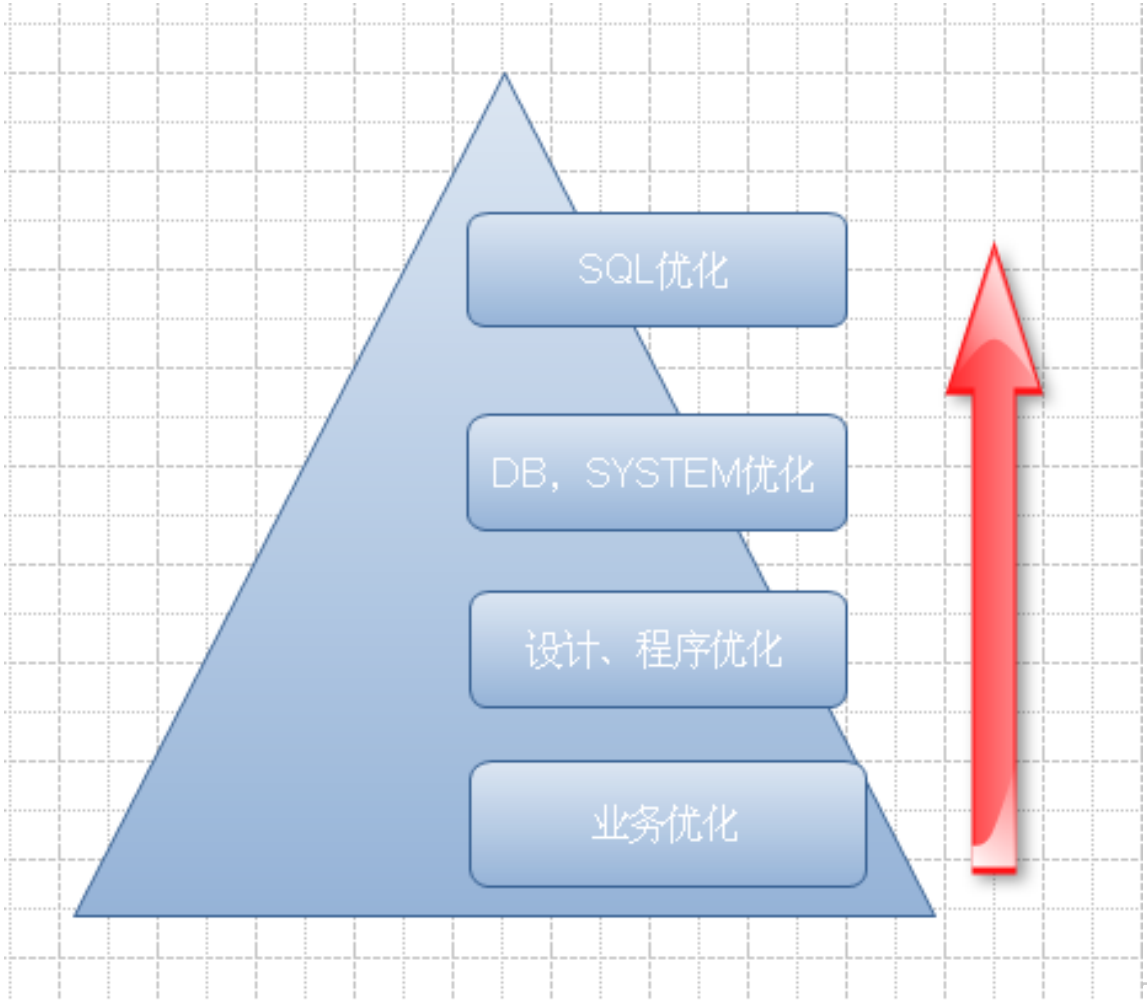
MYSQL 索引与SQL调优

玄惭

主要内容

- Innodb逻辑存储结构
- 加速查询
- 加速插入
- 案例分析

优化顺序

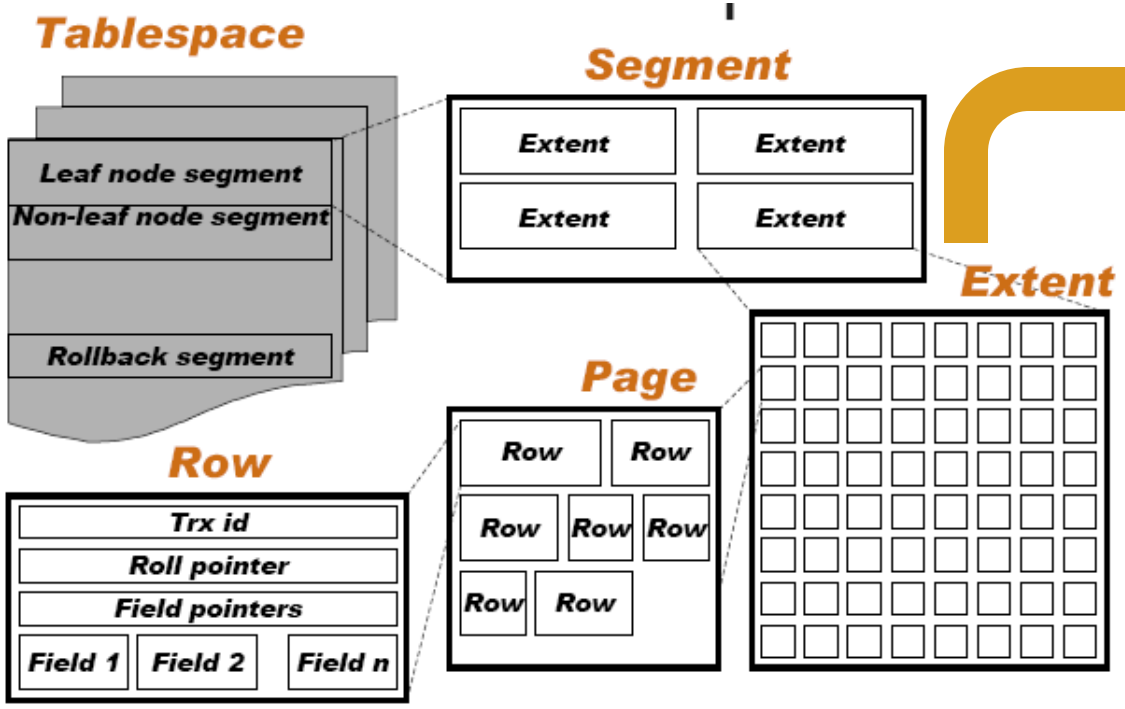


磁盘评估：IOPS

- **IOPS:每秒进行读写（I/O）操作的次数,衡量随机访问的性能;**
- **IOPS类型：顺序读，顺序写，随机读，随机写(顺序读iops最高)**
- **普通SAS：100-200 iops**
- **Intel SSD：2000+ (writes) ，5000+ (reads)**

Innodb逻辑存储结构

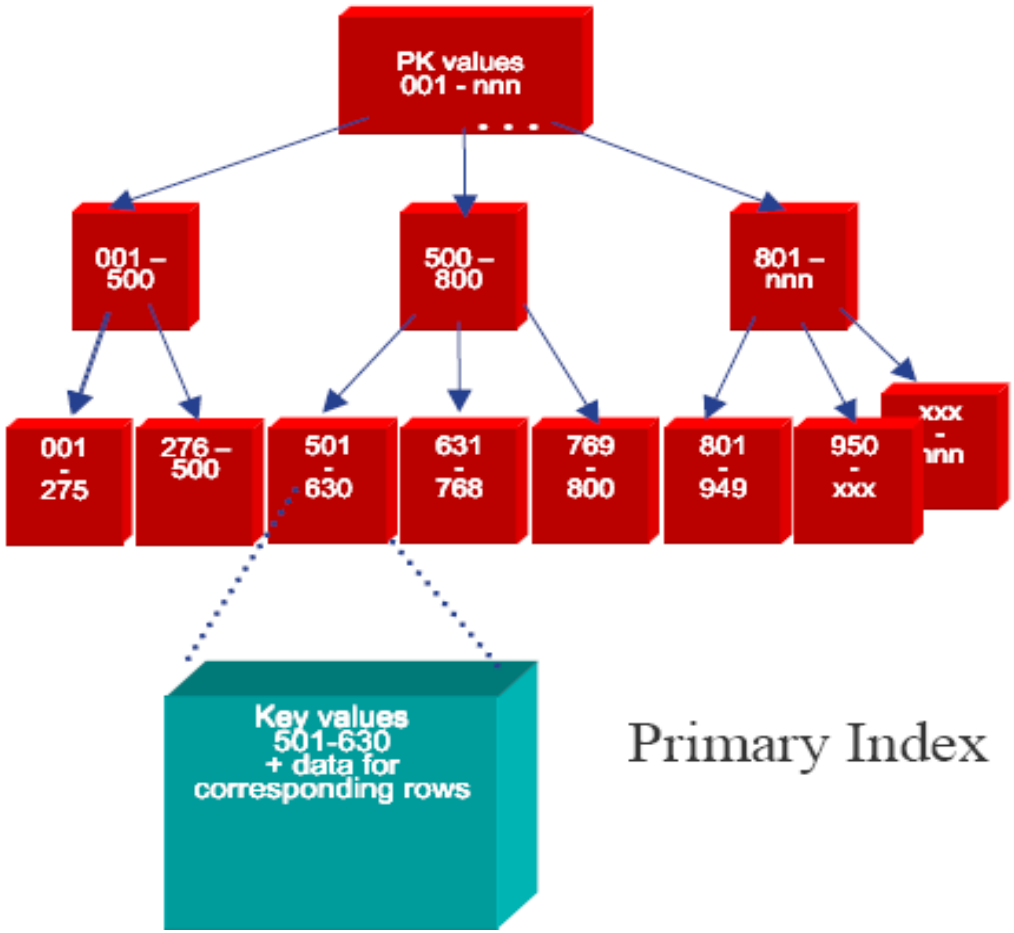
Innodb逻辑存储结构-居民小区结构



古荡新村—第4幢—第5层—第2个单元—3-2号房间
3-1，3-2，3-3三个房间对应的三个row
第2单元对应一个page
第5层楼有2个单元，则对应一个extent
第4幢楼有8个楼层，则对应一个segment
古荡新村所有居民楼对应一个tablespace；

Innodb逻辑存储结构

InnoDB 第一索引(primary index)



1.Innodb存储引擎的表就是索引组织表，表中的数据按照主键顺序存放；
2.索引组织表每张表的主键构造一颗B+树，在叶子节点中存放整张表的行记录；
3.所有叶子节点到根节点的高度H都相同，所以又叫平衡树；
4.叶子节点的数据按照key升序排列，节点间是一个双向链表；

Innodb逻辑存储结构

InnoDB 主键索引访问成本：

- **单行查询成本(SELECT)：**

  S = *h IOPS*

- **单行的更新成本(UPDATE)：**

  U = search cost + rewrite data page = (h + 1) Iops

- **单行的插入成本(INSERT)：**

  I = search cost + rewrite index page + rewrite data page

  I = h + 1 + 1 = (h + 2) Iops

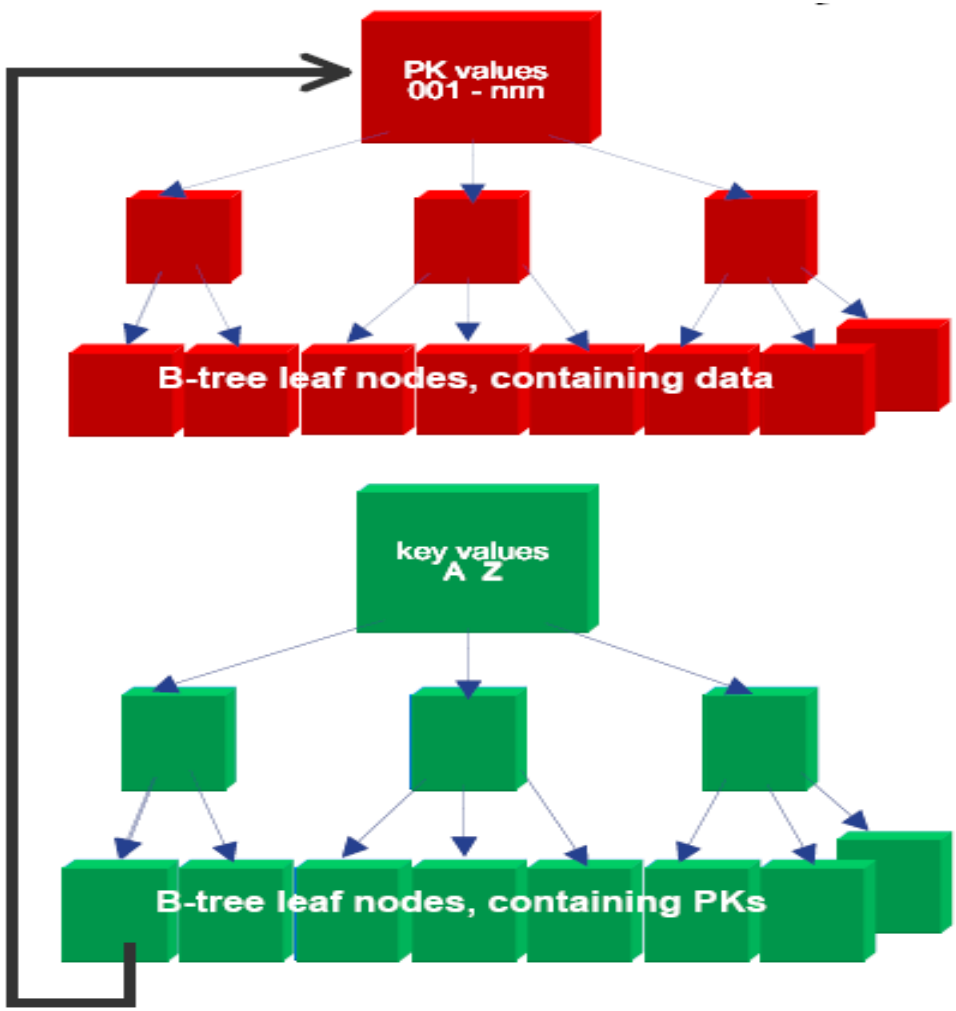
- **单行删除成本(DELETE)：**

  D = search cost + rewrite index page + rewrite data page

  D = h + 1 + 1 = (h + 2) Iops

Innodb逻辑存储结构

- InnoDB 第二索引(secondary index)



1.第二索引（secondary index）同样为B+树，
2.叶子节点包含了索引列key value+相关行对应的主键，通过该主键来访问主表（primary index）
3.第二索引的查找成本：
总共的成本=第二索引查找成本+第一索引查找成本

主要内容

- Innodb逻辑存储结构
- 加速查询
- 加速插入
- 案例分析

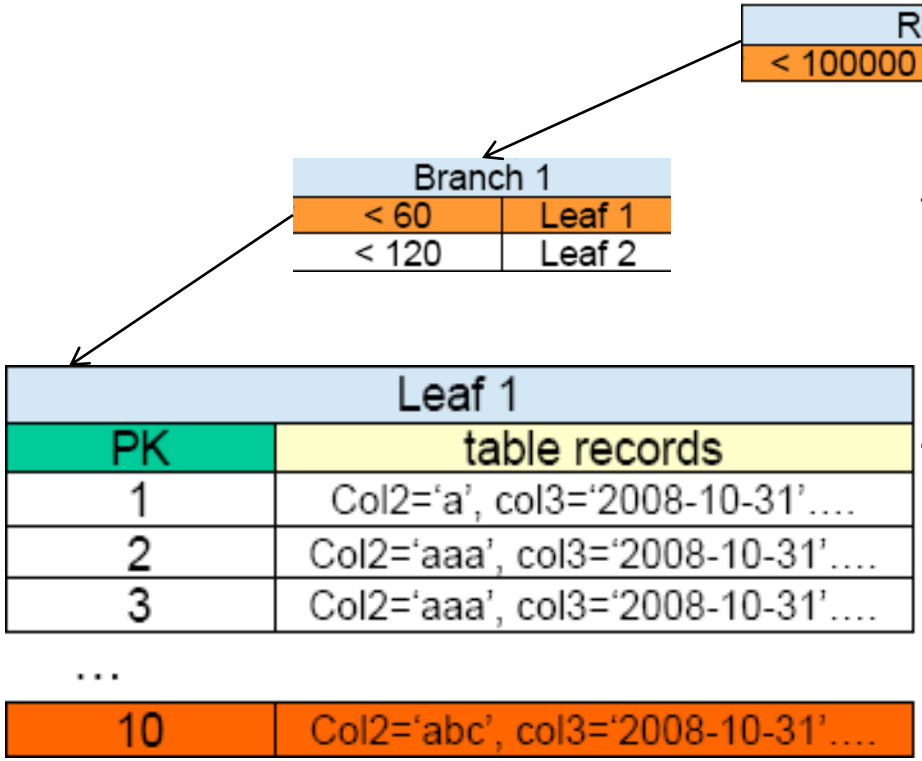
加速查询

- 随机读,顺序读
- 索引范围扫描，覆盖索引扫描，全表扫描

加速查询

InnoDB： 主键查询：

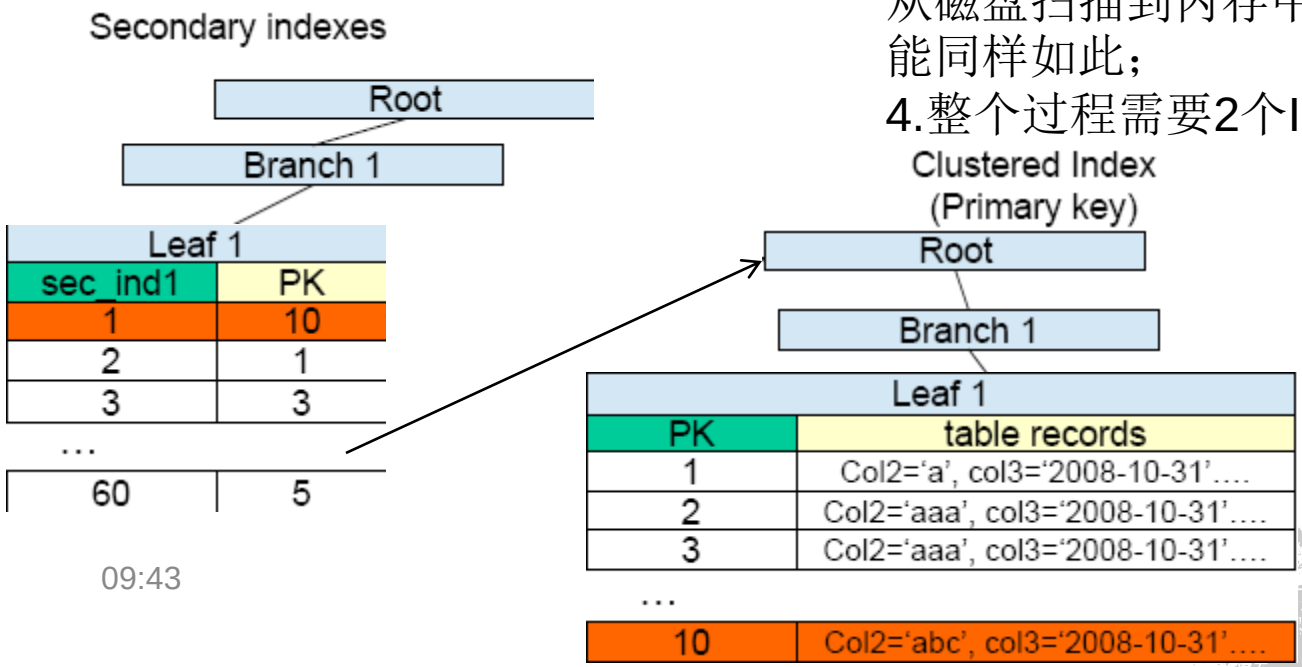
SELECT * FROM tbl WHERE primary_key=10;



1.查询首先从root-》branch-》leaf block-》table records；
2.在leaf block中存放了我们的数据行，并且按照主键pk的顺序进行排序；
3.Root和branch的通常被cache；
4.叶子节点的数据通常需要从磁盘中访问，所以整个过程需要1个IOPS；

InnoDB: 第二索引

```
SELECT * FROM tbl WHERE secondary_index1=1;
```

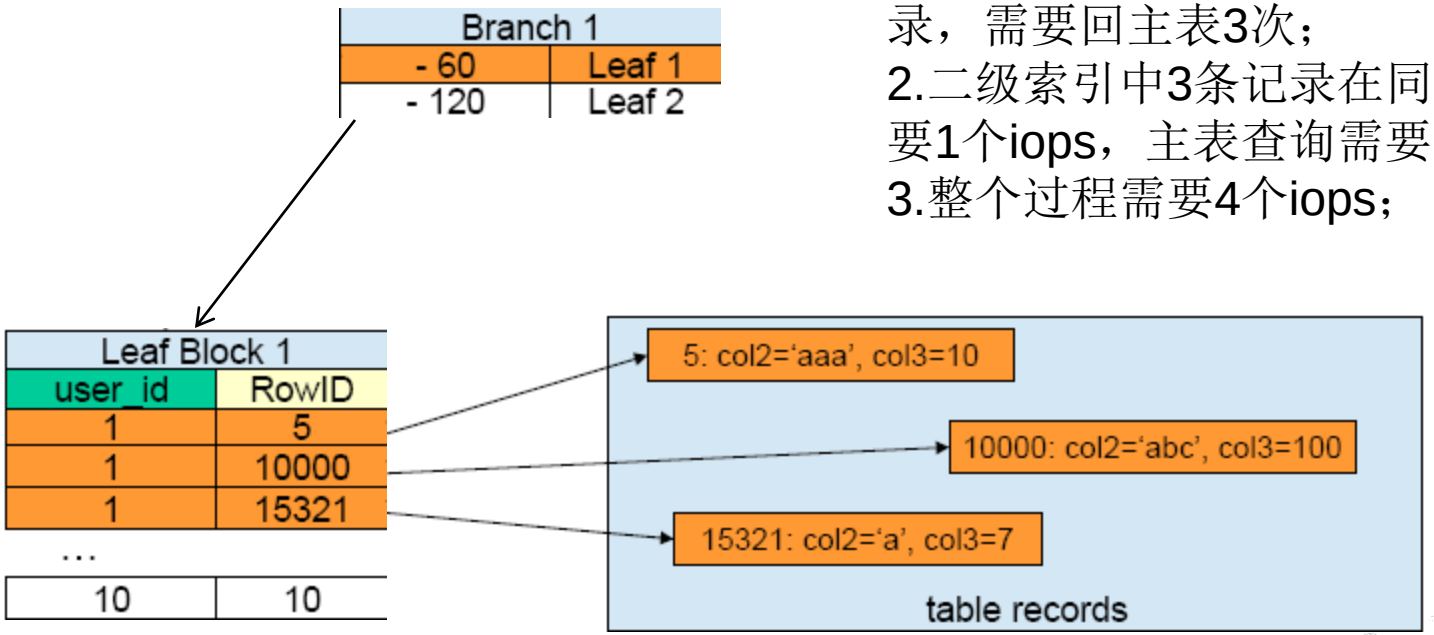


1.查询通过二级索引的root-》branch-》到叶子节点；
2.通过叶子中存放的pk，然后回主表再通过pk定 位到对应的记录；
3.二级索引中的页块可能不在内存中，需要从磁盘扫描到内存中，主表对应的记录也可能同样如此；
4.整个过程需要2个IOPS;

09:43

InnoDB: 非唯一索引

```
SELECT * FROM message_table WHERE user_id =1;
```

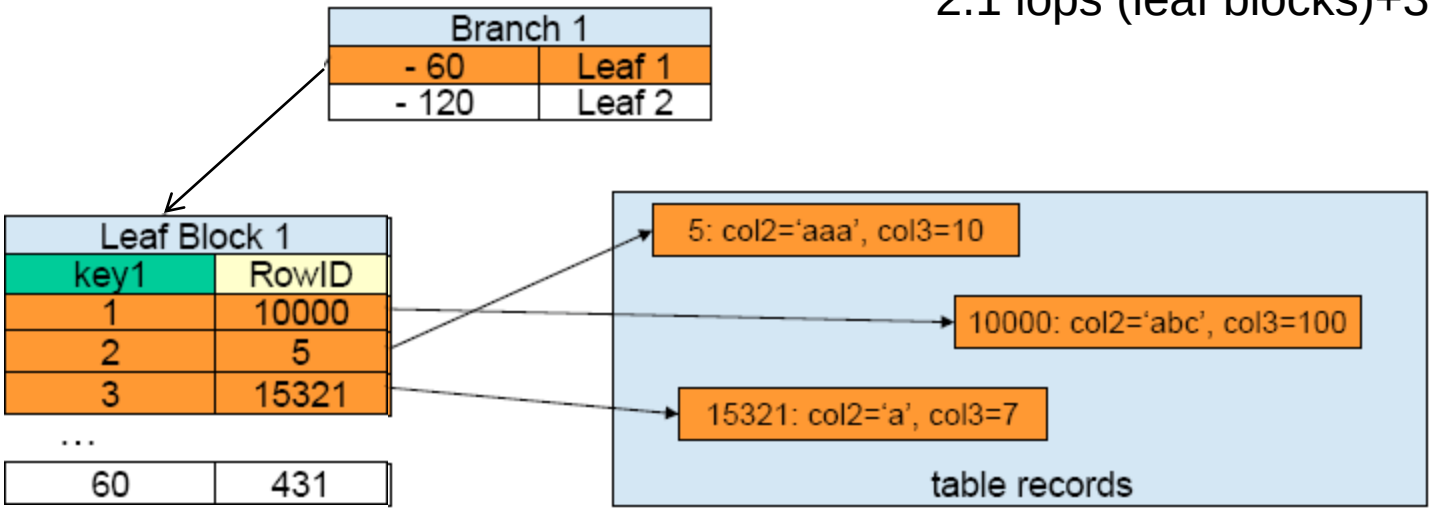


1.一个user_id在二级索引中对应满足3条记录，需要回主表3次；
2.二级索引中3条记录在同一个页块中，需要1个iops，主表查询需要3个iops；
3.整个过程需要4个iops；

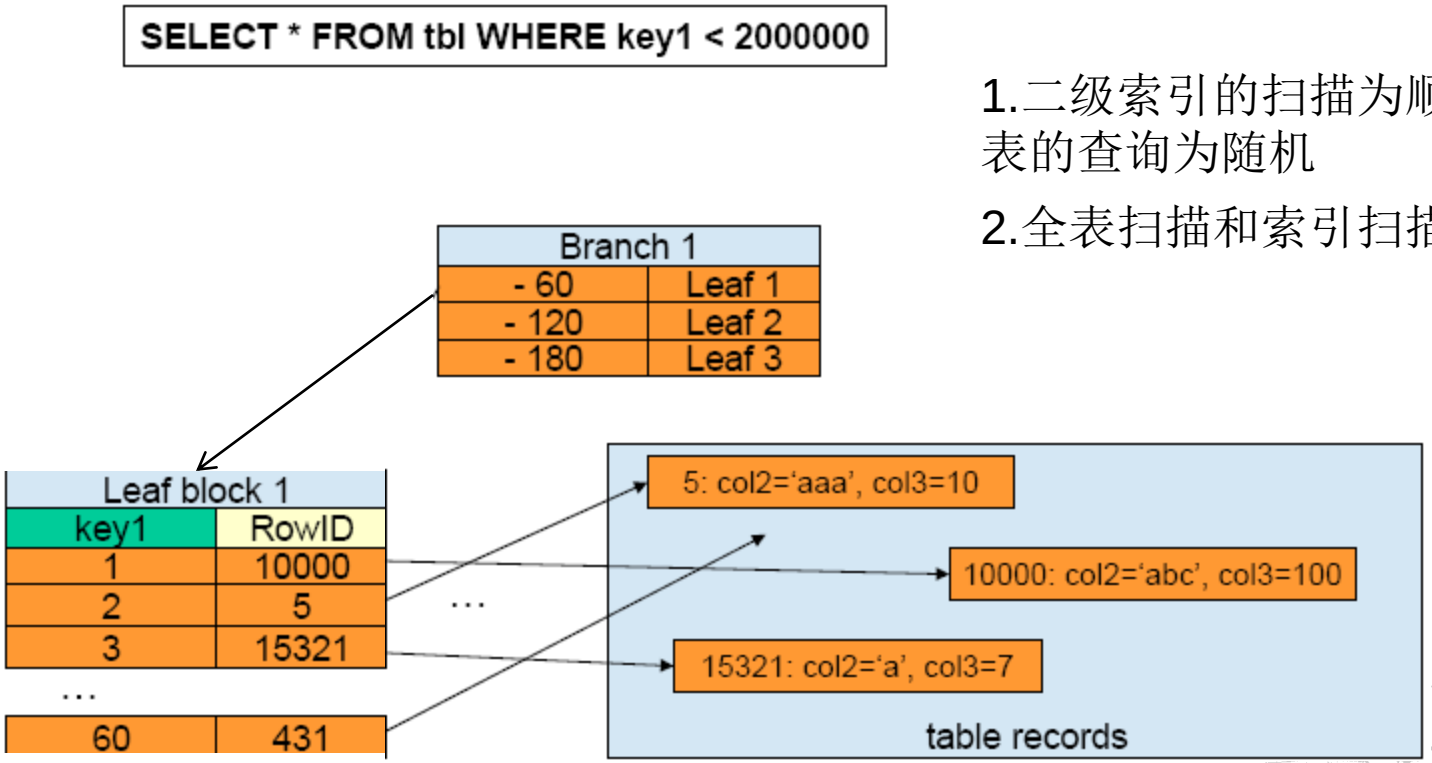
innodb：范围扫描：

```
SELECT * FROM tbl WHERE key1 BETWEEN 1 AND 3;
```

1.二级索引中，满足条件的key在同一页块中，需要1个iops；回主表查询需要3个iops；
2.1 iops (leaf blocks)+3 iops(table records)



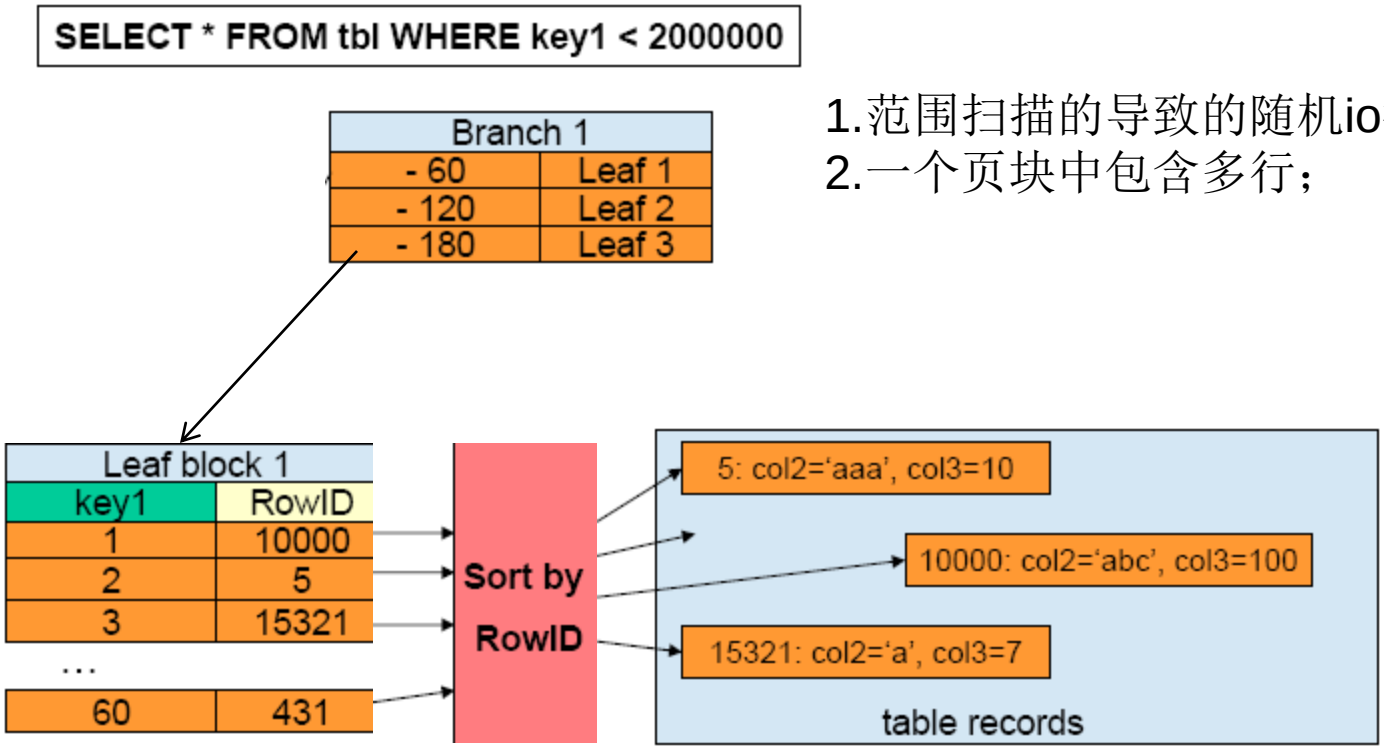
innodb：劣质索引扫描



1.二级索引的扫描为顺序，主表的查询为随机

2.全表扫描和索引扫描

Mysql 6.0：MRR(multi-range-read)



1.范围扫描的导致的随机io得到缓解
2.一个页块中包含多行；

innodb：全表扫描：

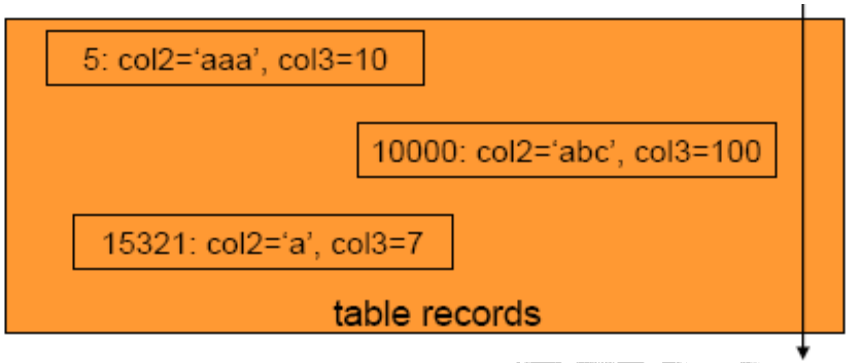
SELECT * FROM tbl

Branch 1	
- 60	Leaf 1
- 120	Leaf 2
- 120	Leaf 3

1.全表扫描为顺序扫描；
2.全表扫描与大范围索引扫描：顺序io<->随机io
→ 单块中有较多行：1个iops访问多行；
→innodb 预读优化：64个块

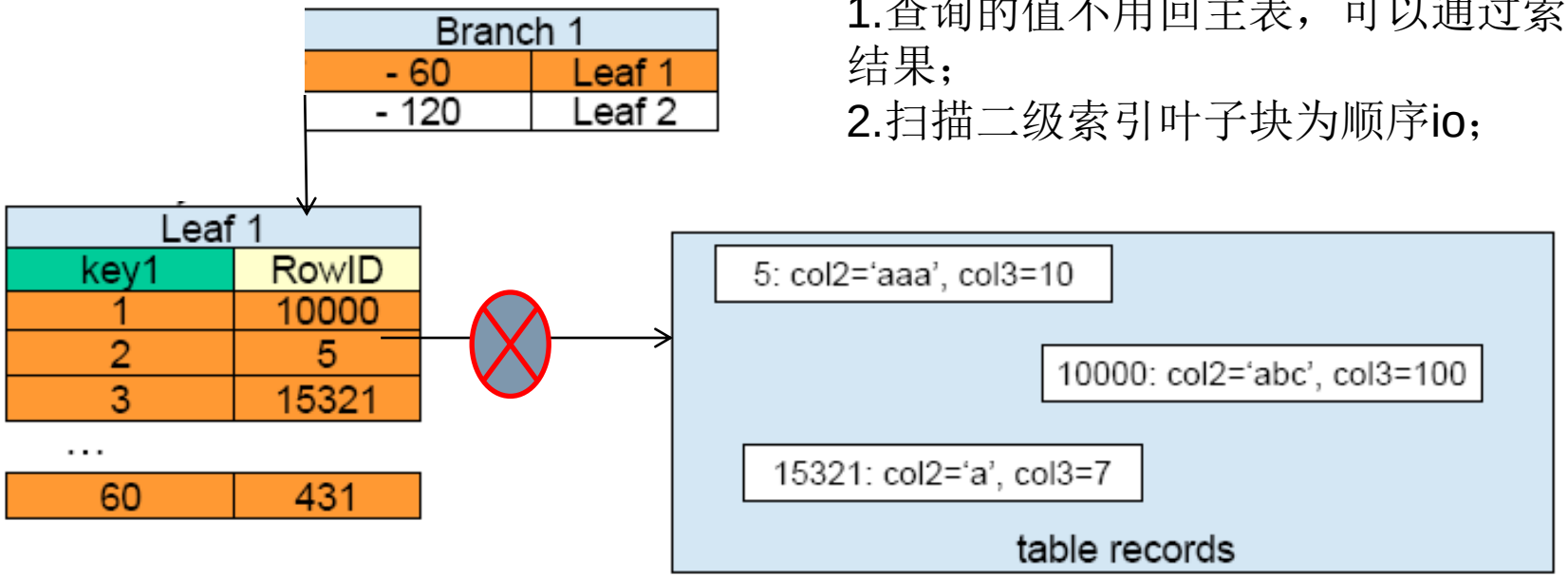
Leaf Block 1	
key1	RowID
1	10000
2	5
3	15321
…	
60	431

…



innodb：覆盖索引扫描：

SELECT key1 FROM tbl WHERE key1 BETWEEN 1 AND 60;

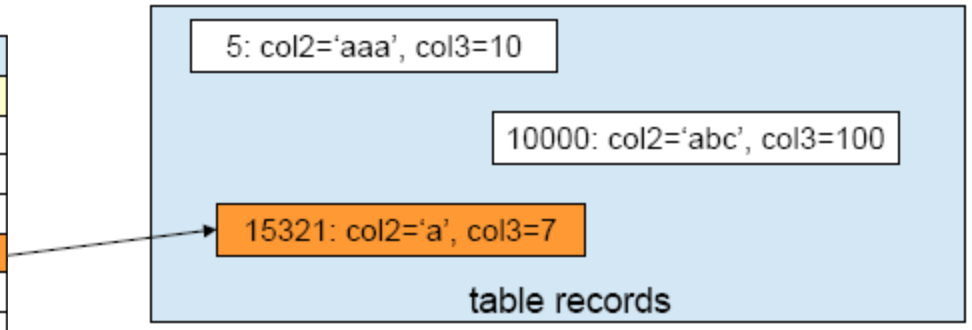


1.查询的值不用回主表，可以通过索引得出结果；
2.扫描二级索引叶子块为顺序io；

innodb：多列索引扫描：

SELECT * FROM tbl WHERE keypart1 = 2 AND keypart2 = 3

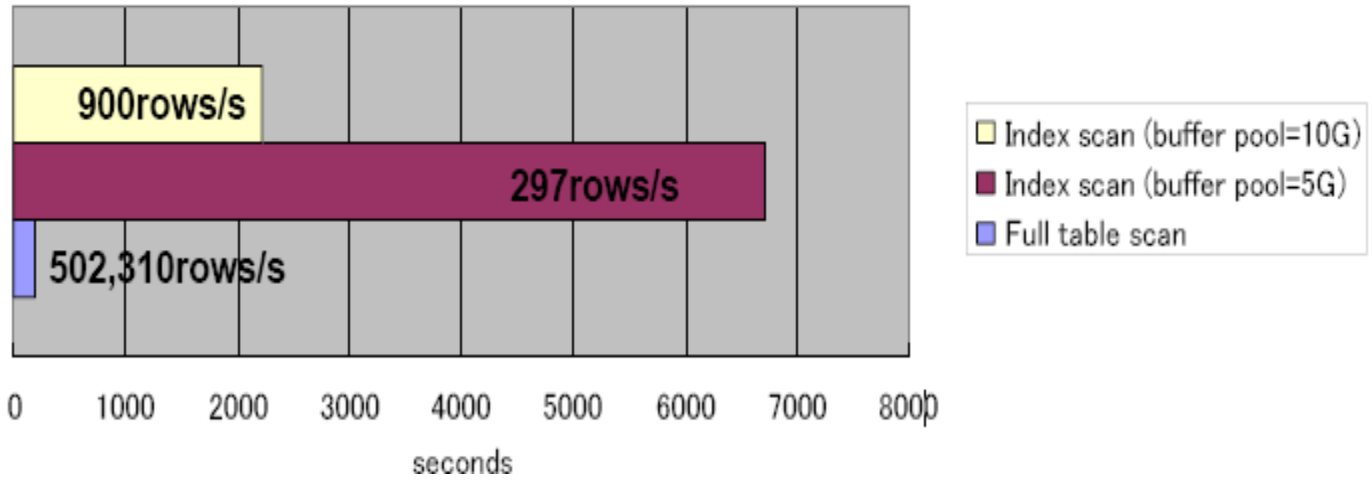
Leaf Block 1		
keypart1	keypart2	RowID
1	5	10000
2	1	5
2	2	4
2	3	15321
3	1	100
3	2	200
3	3	300
4	1	400



1.一次leaf blcok访问，一次数据块访问
2.keypart=2 不在索引中，将会由3次数据块访问

innodb：全表扫描VS索引扫描：

Index scan for 2mil rows vs Full scan for 100mil rows



主要内容

- Innodb逻辑存储结构
- 加速查询
- 加速插入
- 案例分析

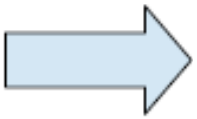
加速插入

插入的时候做了哪些？

```
INSERT INTO tbl (key1) VALUES (61)
```

Leaf Block 1	
key1	RowID
1	10000
2	5
3	15321
…	
60	431

Leaf block is (almost) full



Leaf Block 1	
key1	RowID
1	10000
2	5
3	15321
…	
60	431

Leaf Block 2	
key1	RowID
61	15322
Empty	

A new block is allocated

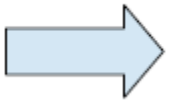
叶子节点分裂

加速插入

顺序插入

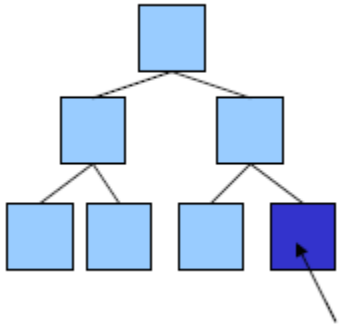
INSERT INTO tbl (key1) VALUES (current_date())

Leaf Block 1	
key1	RowID
2008-08-01	1
2008-08-02	2
2008-08-03	3
…	
2008-10-29	60



Leaf Block 1	
key1	RowID
2008-08-01	1
2008-08-02	2
2008-08-03	3
…	
2008-10-29	60

Leaf Block 2	
key1	RowID
2008-10-29	61
Empty	



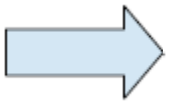
1.插入的顺序为递增（id，datetime）；
2.插入在叶子块最后追加，没有碎片；
3.插入数据在同一块内，插入命中，性能高；

所有新插入的条目都在改叶子块内，插入命中

随机插入

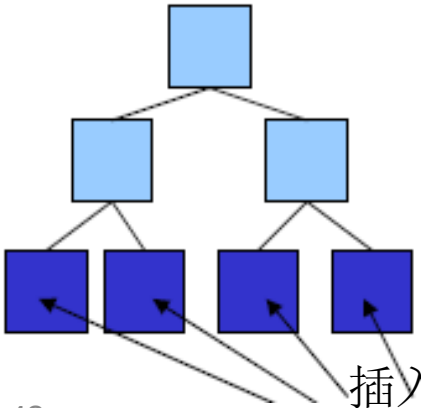
INSERT INTO message_table (user_id) VALUES (31)

Leaf Block 1	
user_id	RowID
1	10000
2	5
3	15321
…	
60	431



Leaf Block 1	
user_id	RowID
1	10000
…	
30	333
Empty	

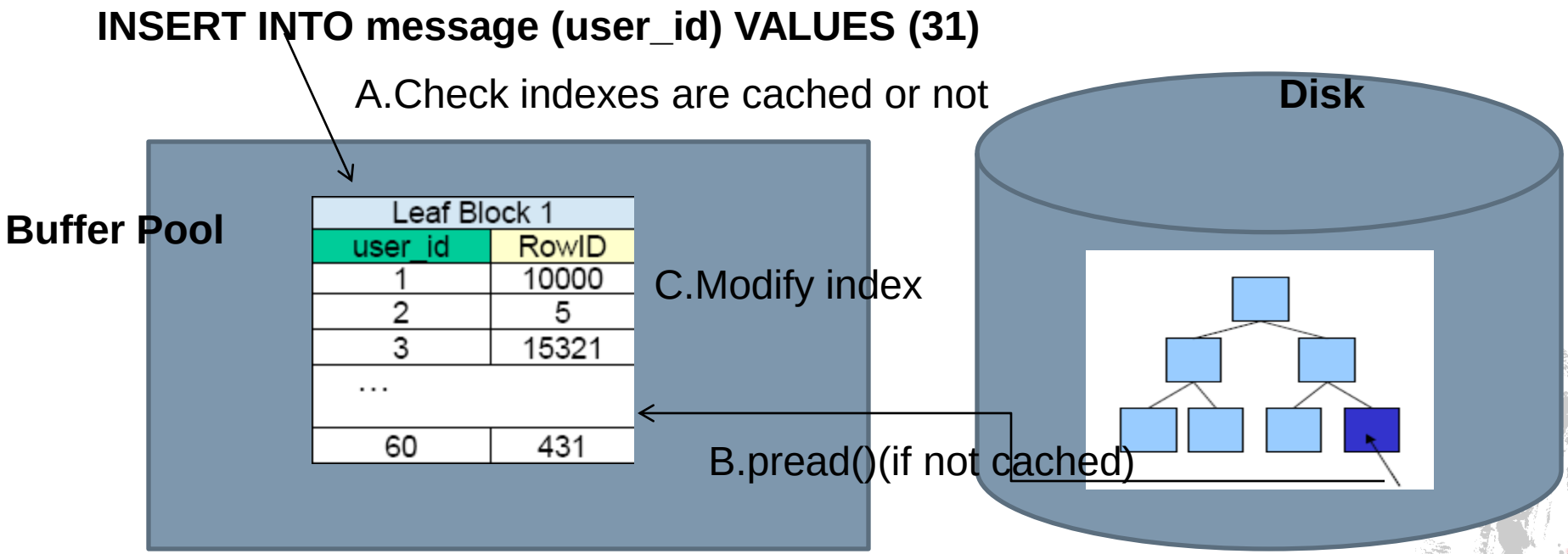
Leaf Block 2	
user_id	RowID
31	345
…	
60	431
Empty	



1.user_id的插入是随机的，这样造成的结果就是：
碎片，每个页块中的行数变少→更多的块，更多的存储空间，内存的命中率降低

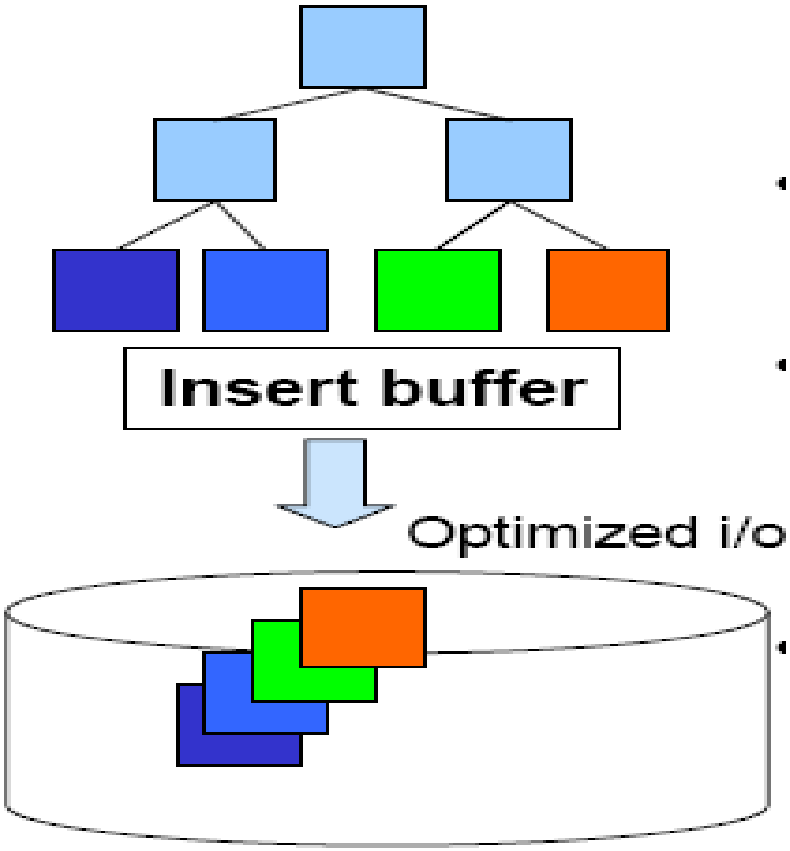
插入发生在许多随机的叶子块中，插入不命中

随机插入读取索引



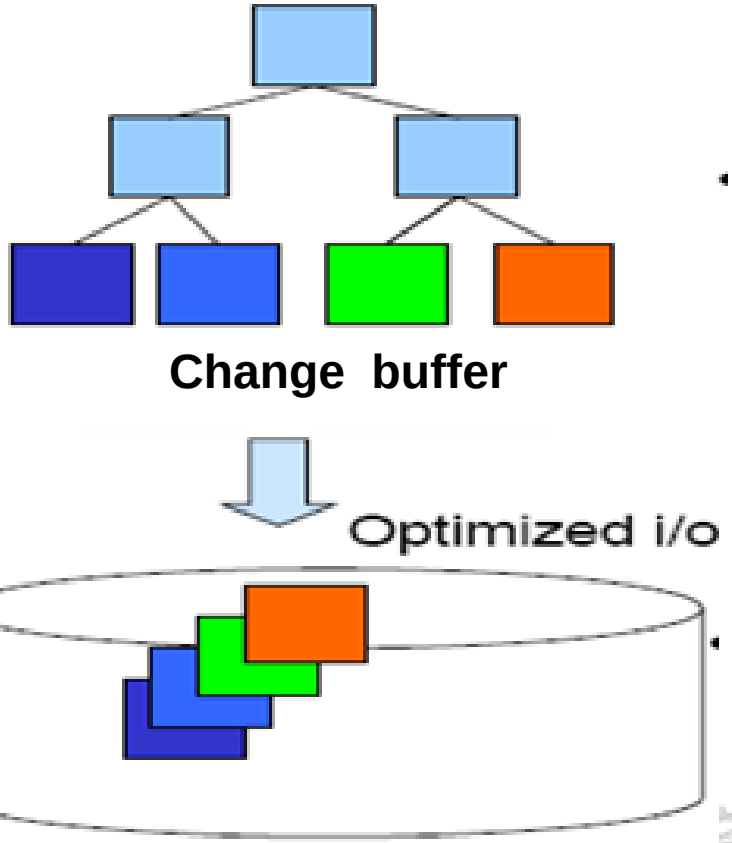
1.在修改的索引块前，必须将索引块读入内存中；
2.如果索引块不在内存中，则需要调用pread将其读入内存；
3.使用更大的内存，或者采用ssd来提升性能；

5.1 Insert buffer



- 1.当innodb插入数据的时候，非唯一的第二索引的索引块没有在内存中时，插入到insert buffer中，以避免随机io；
- 2.Insert buffer 间隔一段时间merger到第二索引中；
- 3.对于update，delete操作，还是不能避免随机io操作；

5.5 Change buffer



1.二级索引上所有的操作都在change buffer中完成:insert,update,delete

主要内容

- Innodb逻辑存储结构
- 加速查询
- 加速插入
- 案例分析

案例分析--T=S/V

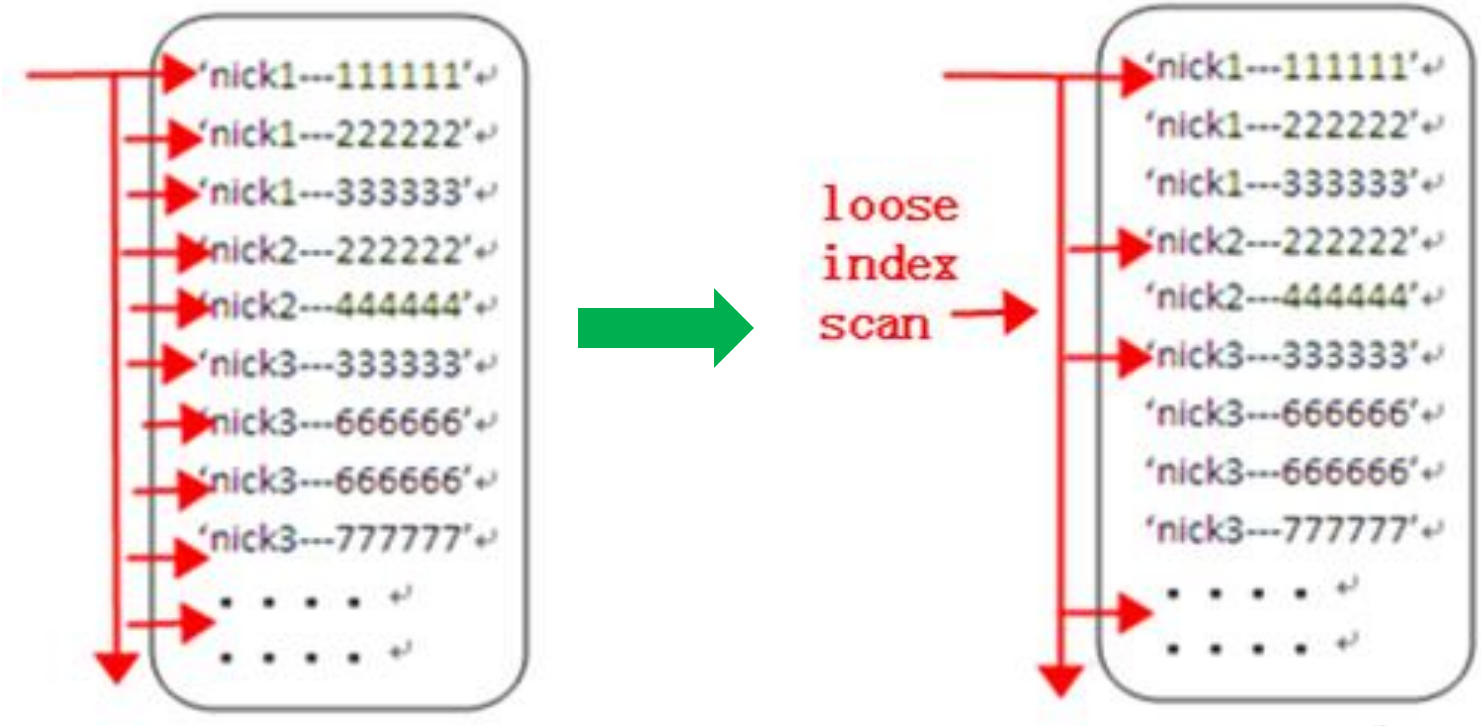
案例分析—T=S/V(时间=路程/速度)

+日志系统UV统计优化(优化S)

+TC读库的回表（优化V）

日志系统UV统计优化—'loose index scan':

select count(distinct nick) from XX;



52.78 s

5.81 s

日志系统UV统计优化—’loose index scan’:

select count(*) from (select distinct(nick) from xx);

```
root@db>select count(*) from ( select
distinct(nick) from xx)t ;
+----------+
| count(*) |
+----------+
|   806934 |
+----------+
1 row in set (5.81 sec)
```

```
root@db>select count(distinct nick) from xx;
+---------------------+
| count(distinct nick) |
+---------------------+
|              806934 |
+---------------------+
1 row in set (52.78 sec)
```

案例分析--T=S/V

日志系统UV统计优化—’loose index scan’:

select count(*) from (select distinct(nick) from xx);

原始写法：

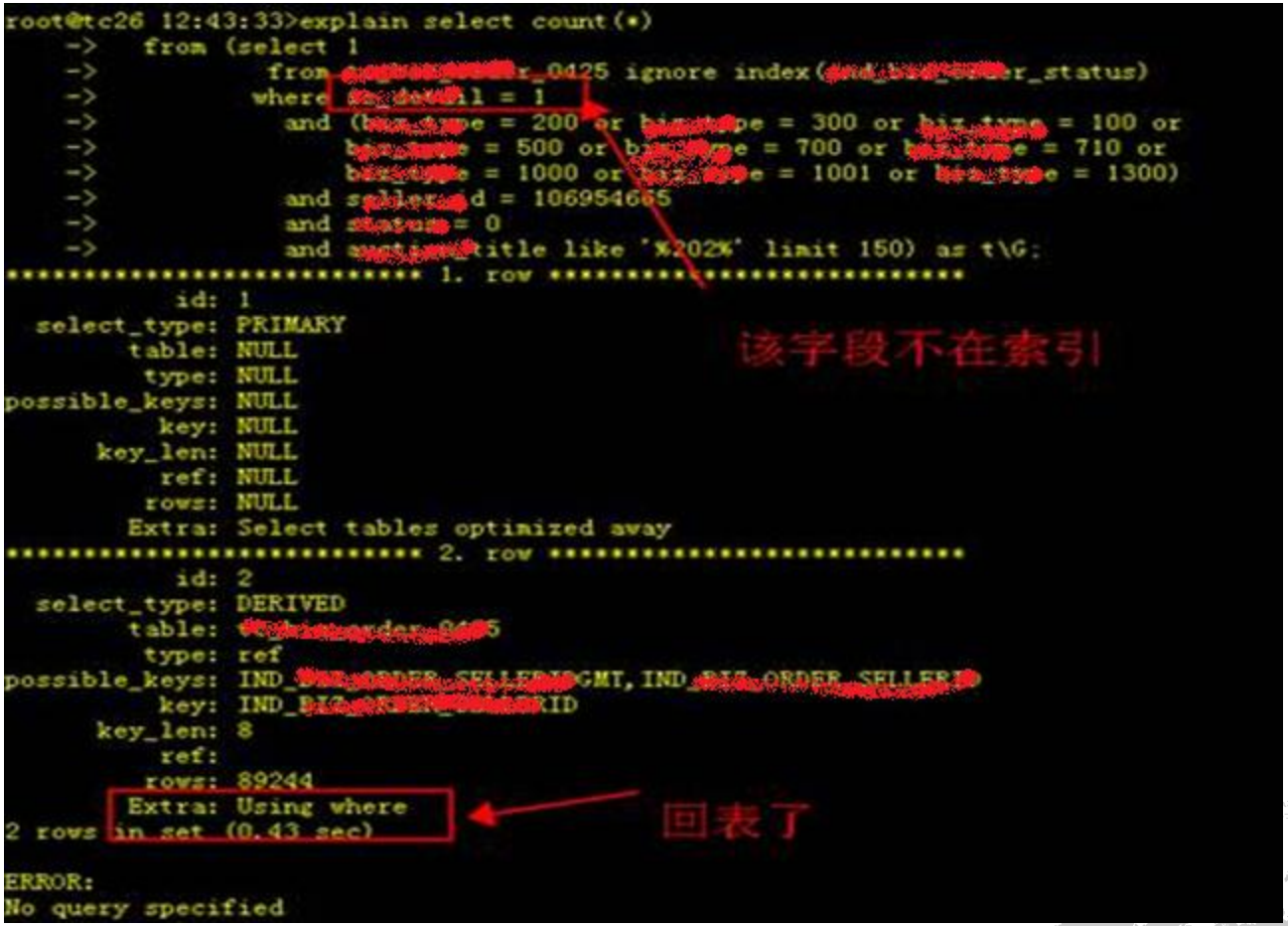
```
root@db>explain select count(distinct nick) from xx;
+----+-------------+-------+------+---------------+---------+---------+------+----------+-------------+
| id | select_type | table | type | possible_keys | key     | key_len | ref  | rows     | Extra       |
+----+-------------+-------+------+---------------+---------+---------+------+----------+-------------+
|  1 | SIMPLE      | xx    | index| NULL          | ind_nick| 67      | NULL | 19546123 | Using index |
+----+-------------+-------+------+---------------+---------+---------+------+----------+-------------+

root@db>explain select distinct(nick) from xx ;
+----+-------------+-------+------+---------------+---------+---------+------+---------+--------------------------+
| id | select_type | table | type | possible_keys | key     | key_len | ref  | rows    | Extra                    |
+----+-------------+-------+------+---------------+---------+---------+------+---------+--------------------------+
|  1 | SIMPLE      | xx    | range| NULL          | ind_nick| 67      | NULL | 2124695 | Using index for group-by |
+----+-------------+-------+------+---------------+---------+---------+------+---------+--------------------------+
```

案例分析--T=S/V

TC读库的回表—随机io与顺序io:

大量的数据需要回表取得，造成大量的随机io



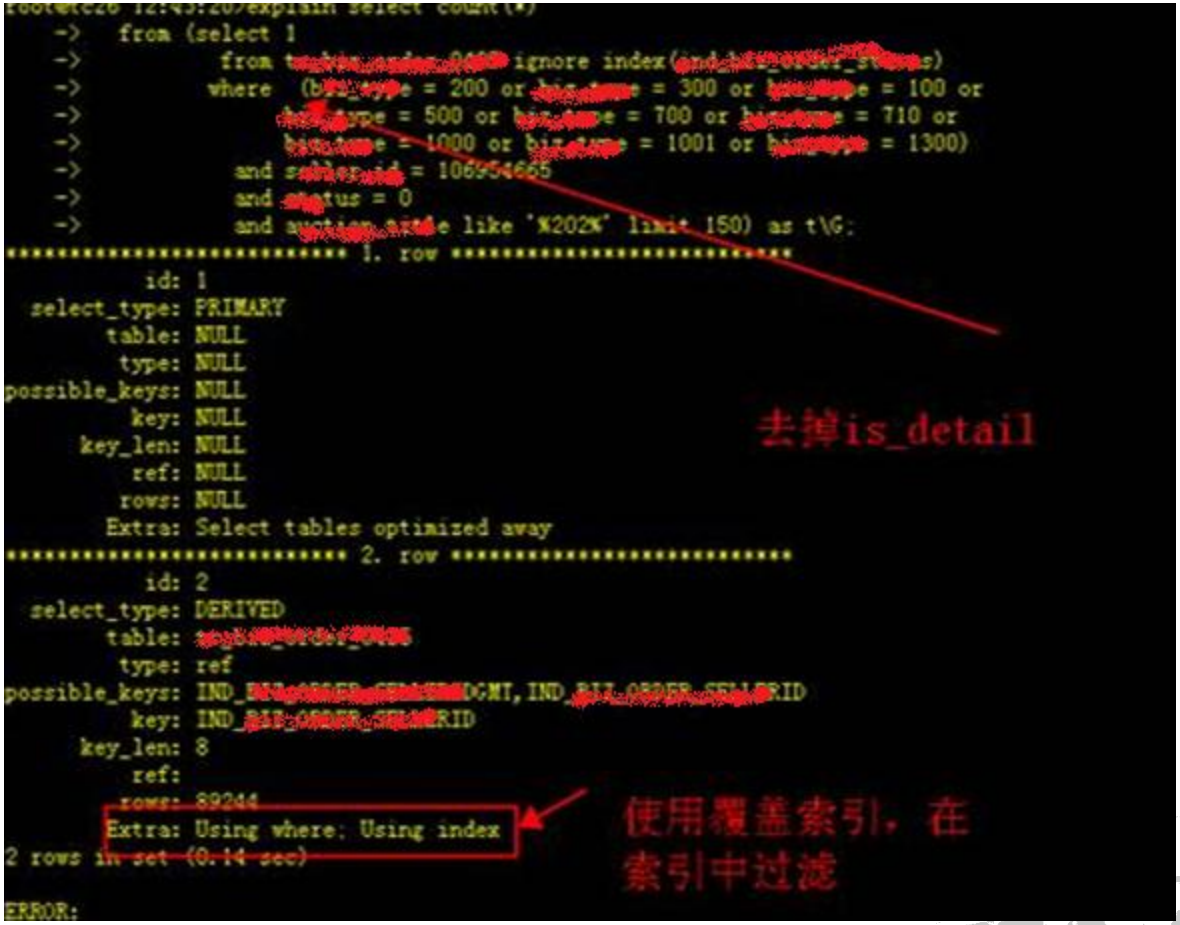

```
root@tc26 12:43:33>explain select count(*)
    ->   from (select 1
    ->           from [redacted]r_0425 ignore index([redacted]r_status)
    ->          where [redacted]il = 1
    ->            and ([redacted]pe = 200 or [redacted]pe = 300 or [redacted]e = 100 or
    ->                 [redacted]e = 500 or [redacted]e = 700 or [redacted]e = 710 or
    ->                 [redacted]e = 1000 or [redacted]e = 1001 or [redacted]e = 1300)
    ->            and [redacted]d = 106954665
    ->            and [redacted] = 0
    ->            and [redacted]itle like '%202%' limit 150) as t\G;
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Select tables optimized away
*************************** 2. row ***************************
           id: 2
  select_type: DERIVED
        table: [redacted]
         type: ref
possible_keys: IND_[redacted]CMT,IND_[redacted]
          key: IND_[redacted]
      key_len: 8
          ref:
         rows: 89244
        Extra: Using where
2 rows in set (0.43 sec)

ERROR:
No query specified
```

案例分析--T=S/V

TC读库的回表—随机io与顺序io:

索引中包含了所需查询的字段，不用回表，随机io转为顺序io



参考

参考：

- **Mastering the Art of Indexing Presentation---** *Yoshinori Matsunobu*
- **MySQL技术内幕InnoDB存储引擎---**姜承尧
- **innodb internal--- http://www.innodb.com/**
- **Hidba.net—玄惭**

Thank You