PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com

行锁原理分析

谈锁前提

```
select * from t1 where id = 10;

delete from t1 where name = 'zs';
```

前提一：id列是不是主键?

前提二：当前系统的隔离级别是什么?

前提三：id列如果不是主键，那么id列上有索引吗?

简单SQL的加锁分析

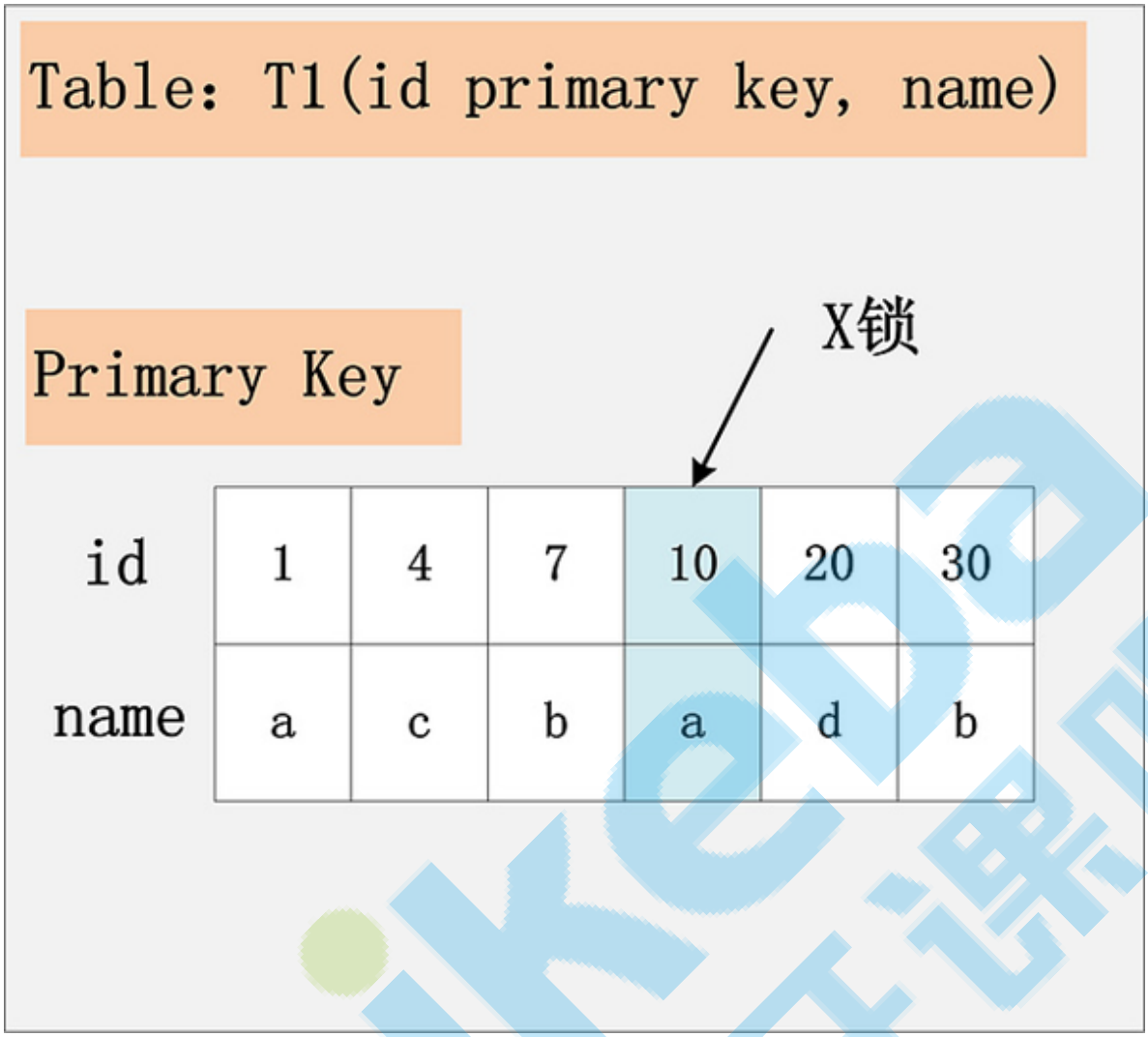
RC隔离级别下

组合一：id主键+RC

这个组合，是最简单，最容易分析的组合。id是主键，Read Committed隔离级别，给定SQL：

delete from t1 where id = 10; 只需要将主键上id = 10的记录加上X锁即可。如下图所示：

PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com

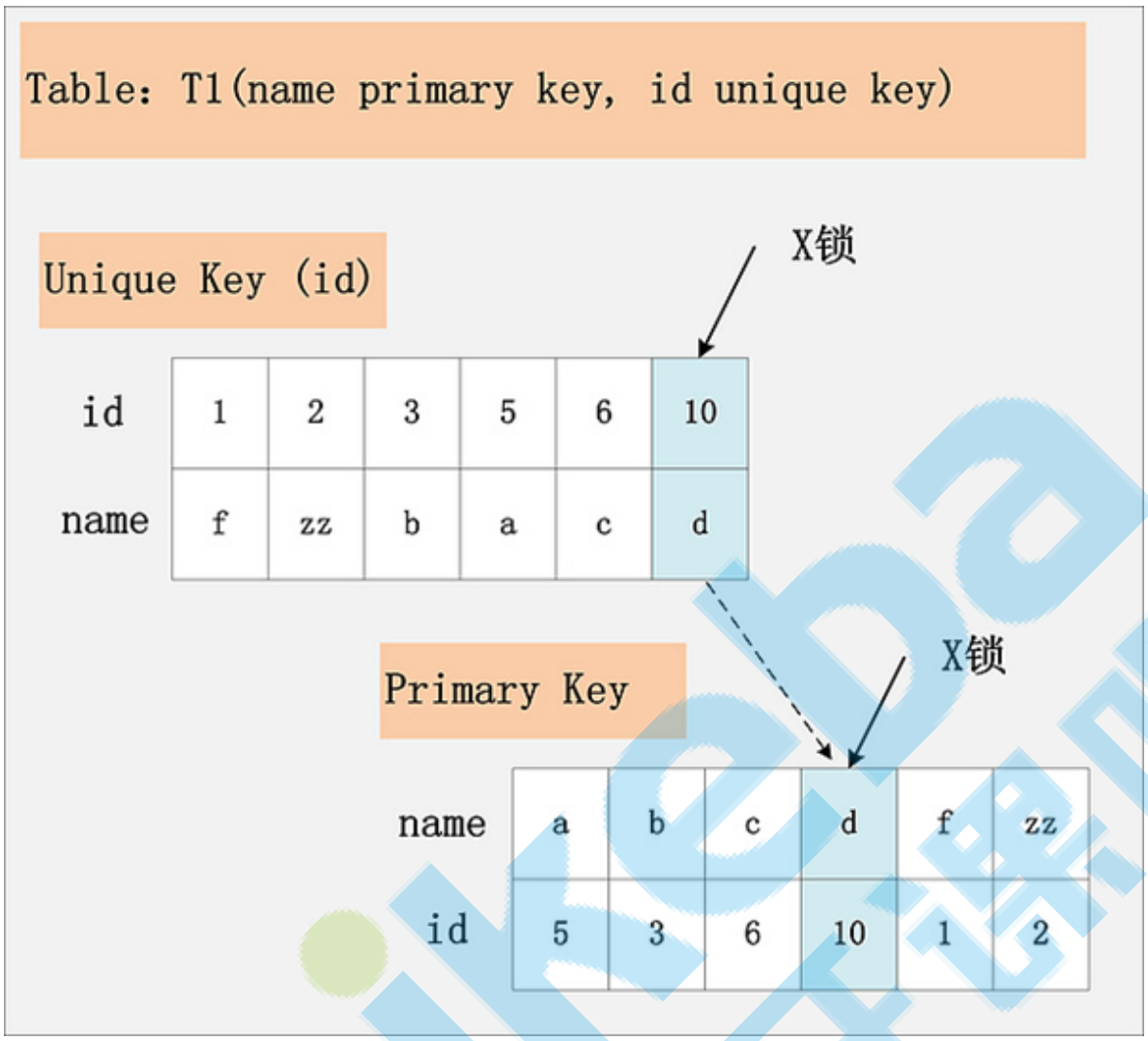


组合二：id唯一索引+RC

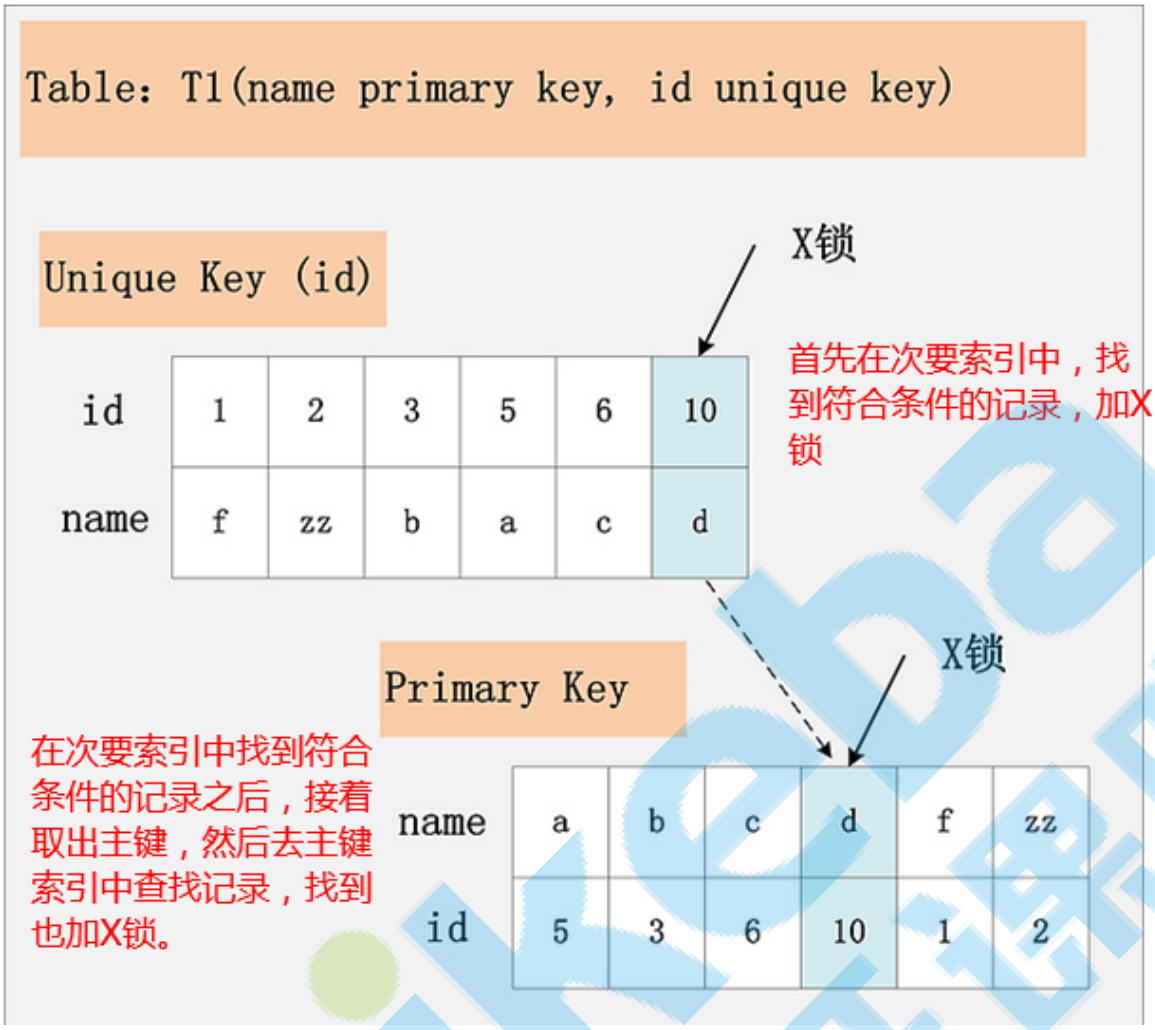
这个组合，id不是主键，而是一个Unique的二级索引键值。那么在RC隔离级别下， 需要加什么锁呢？见下图：

```
delete from t1 where id = 10;
```

PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com

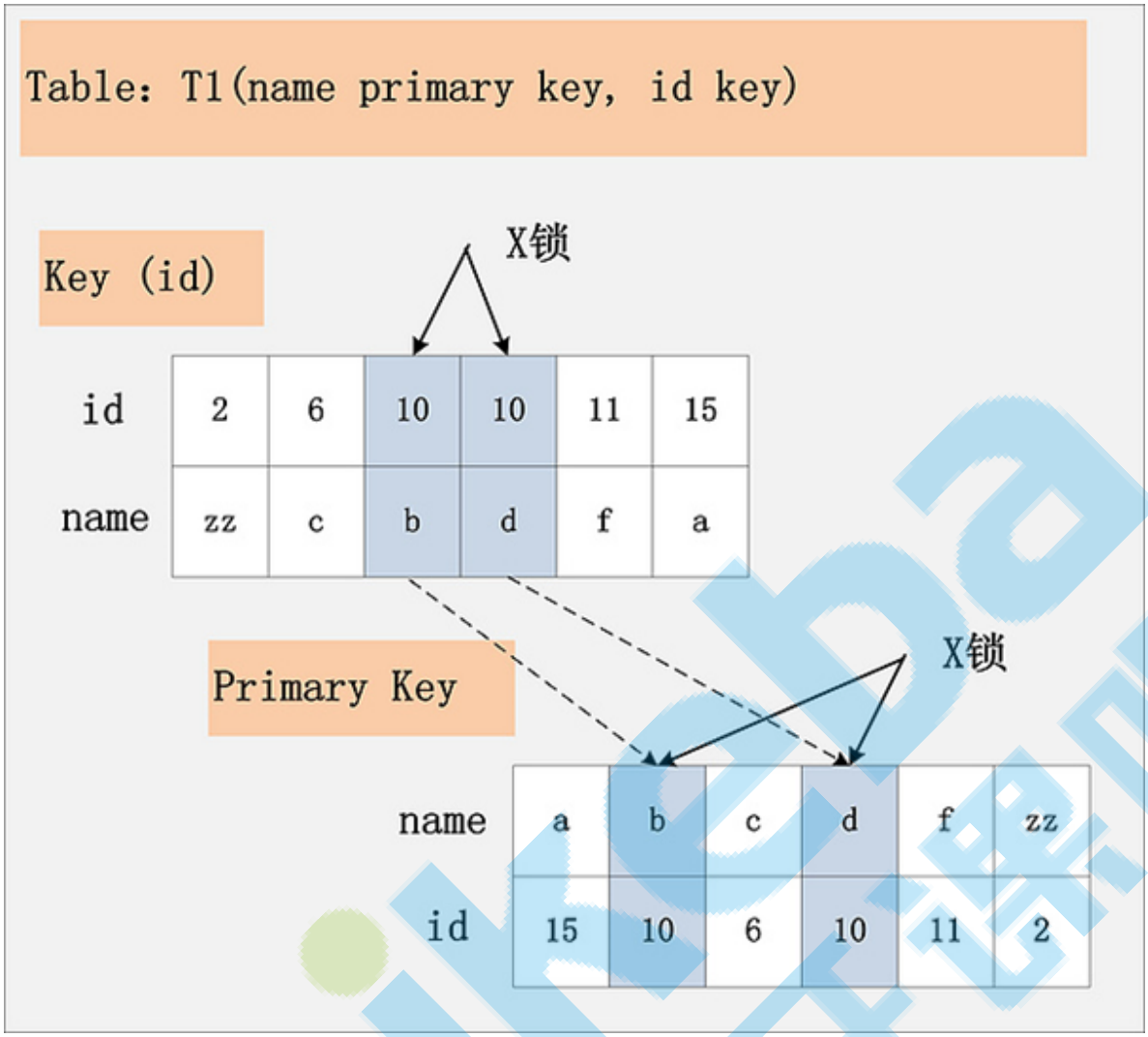


PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com



组合三：id非唯一索引+RC

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

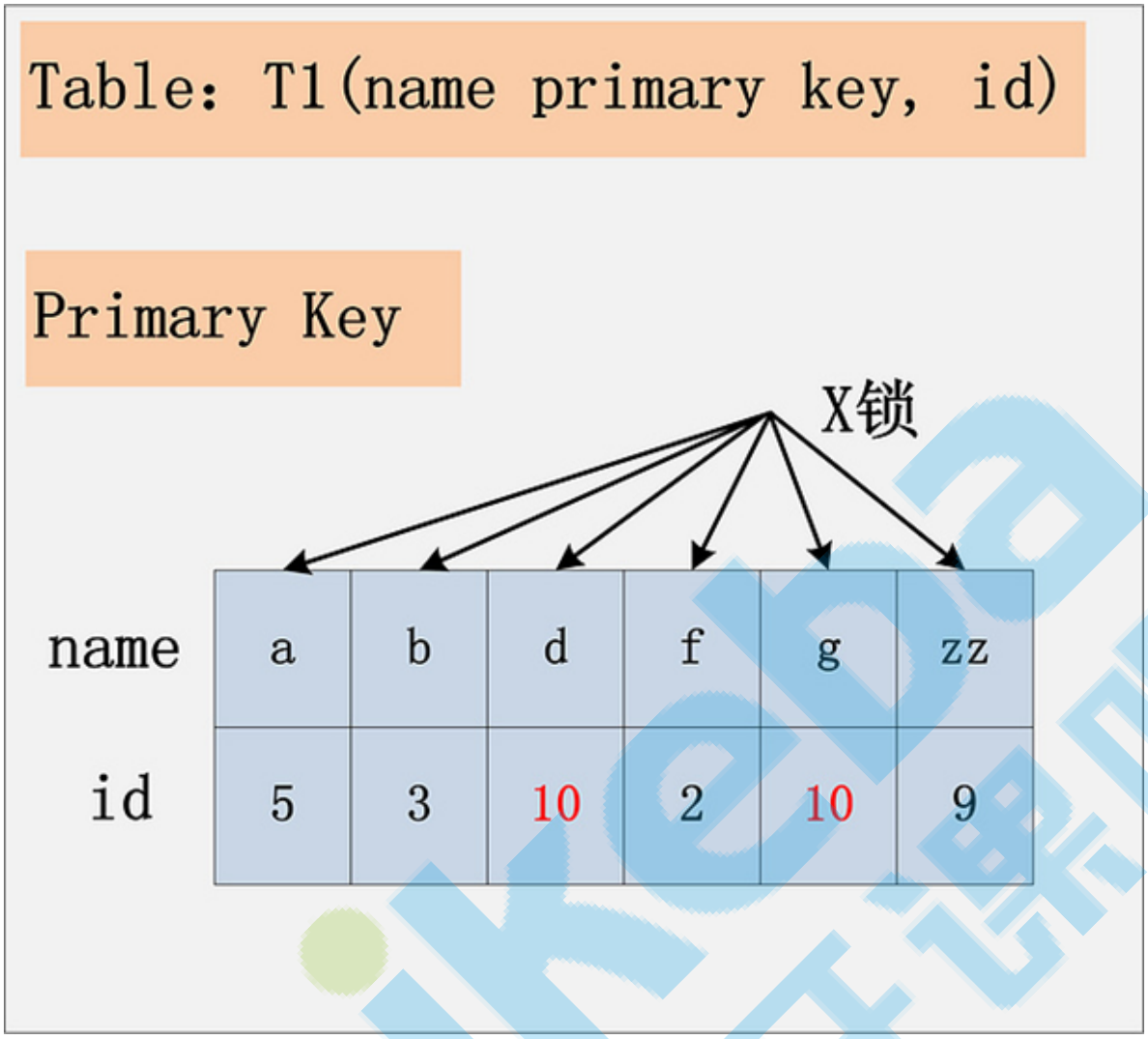


相对于组合一、二，组合三又发生了变化，隔离级别仍旧是RC不变，但是id列上的约束又降低了，id列不再唯一，只有一个普通的索引。假设以下语句，仍旧选择id列上的索引进行过滤where条件，那么此时会持有哪些锁？同样见下图：

组合四：id无索引+RC

id列上没有索引，where id = 10;这个过滤条件，没法通过索引进行过滤，那么**只能走全表扫描做过滤**。对应于这个组合，SQL会加什么锁？或者是换句话说，全表扫描时，会加什么锁？这个答案也有很多：有人说会在表上加X锁；有人说会将聚簇索引上，选择出来的id = 10;的记录加上X锁。那么实际情况呢？请看下图：

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com



组合五：id主键+RR

同组合一

主键等值

主键范围 产生 Gap

组合六：id唯一索引+RR

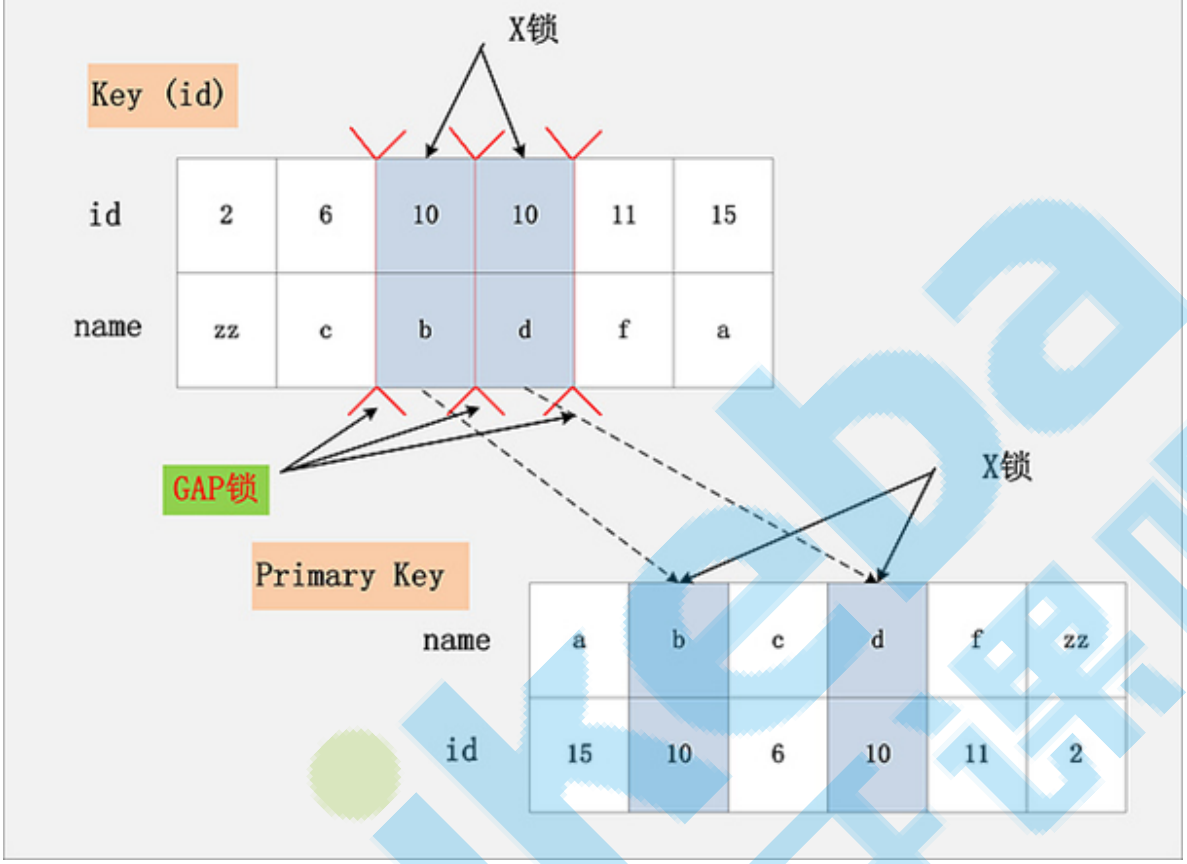
同组合二

组合七：id非唯一索引+RR

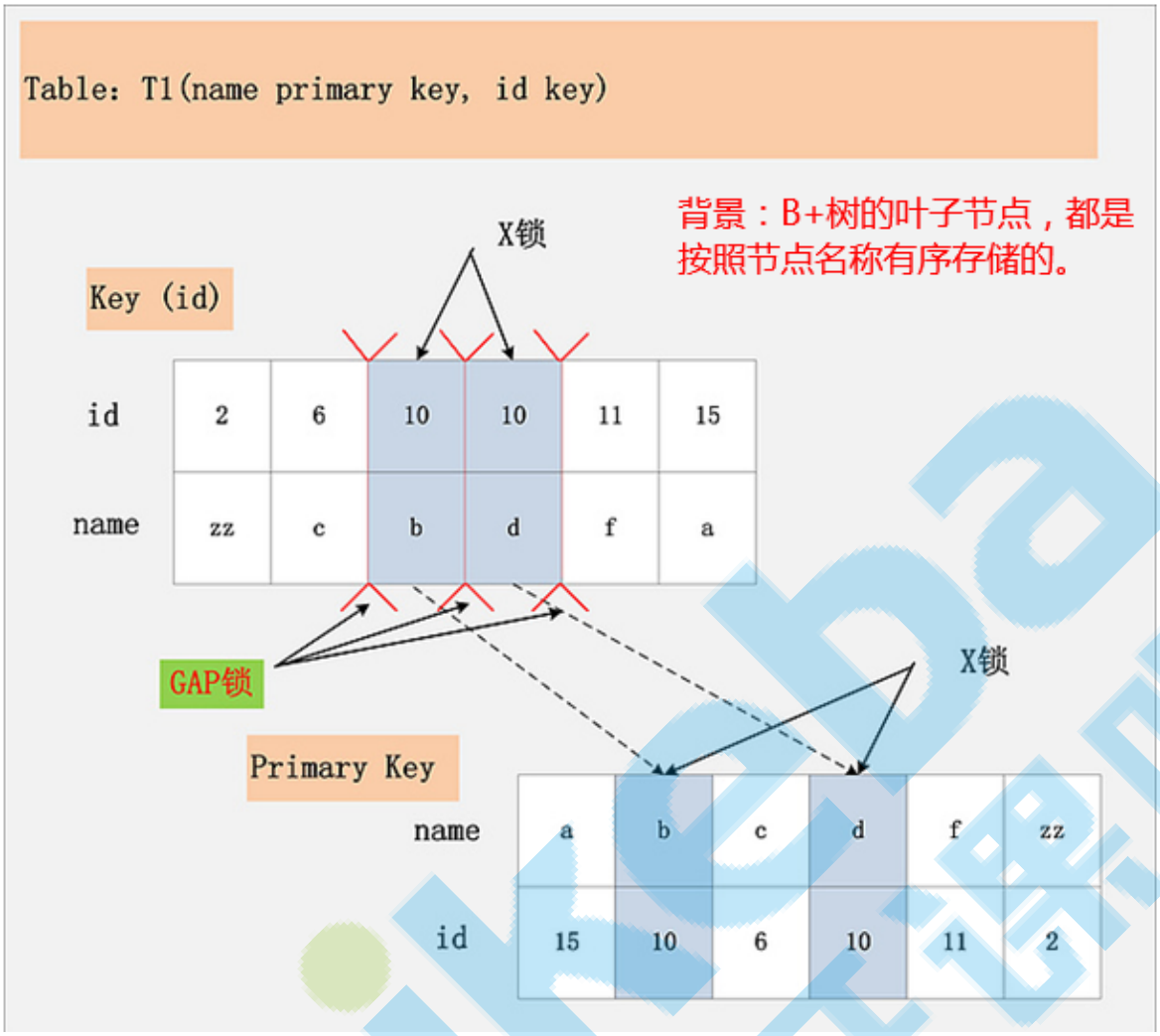
delete from t1 where id = 10;

PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com

Table: T1(name primary key, id key)



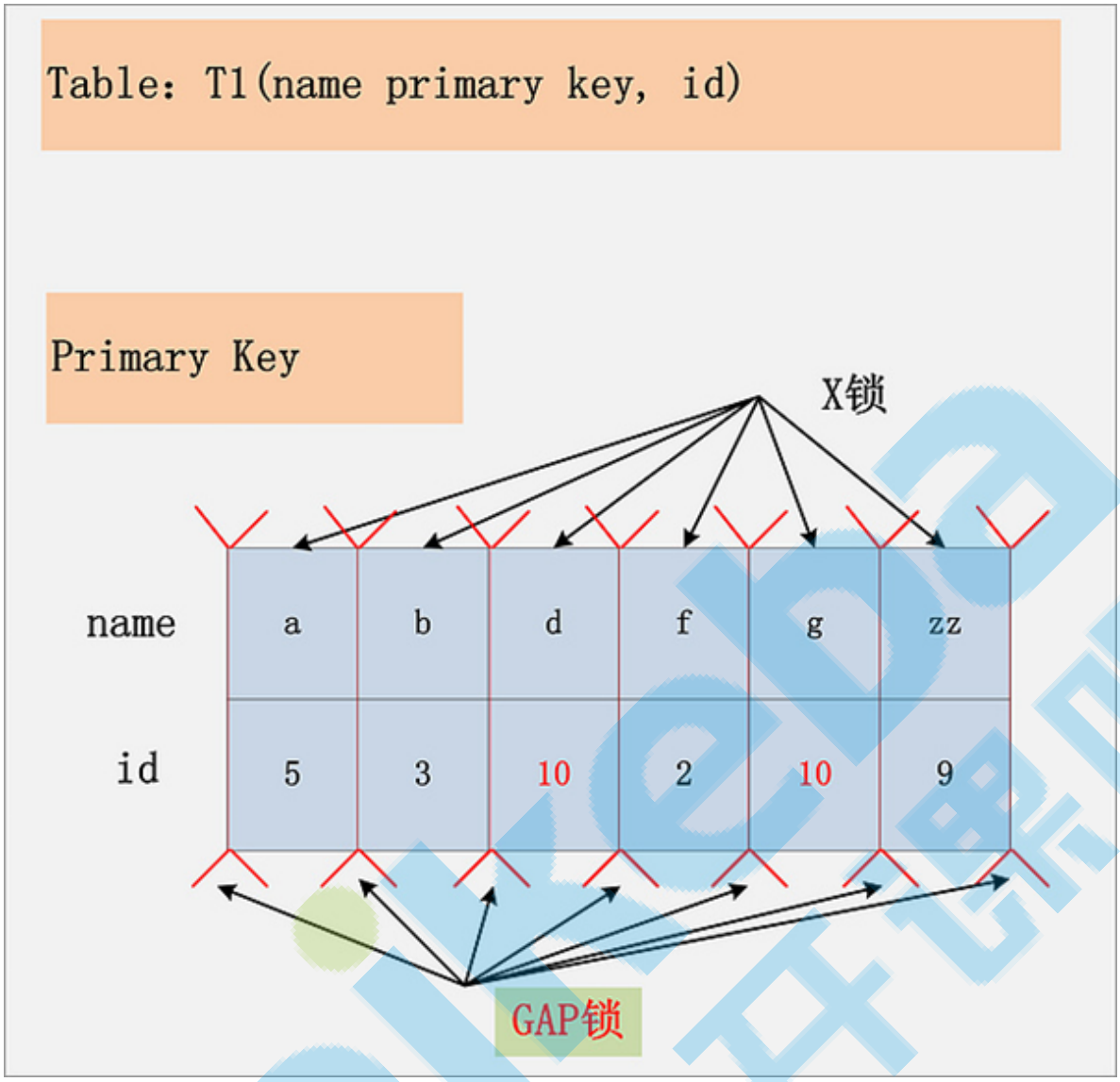
PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com



不能在间隙 insert 防止幻读 （RR）

组合八：id无索引+RR

PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com



组合九：Serializable （LBCC）

只要有SQL 就锁 而且 无索引 表锁

select 是加锁的

一条复杂SQL的加锁分析

PDF-XChange Editor
Click to BUY NOW!
www.tracker-software.com

Table: t1(id primary key, userid, blogid, pubtime, comment)
Index: idx_t1_pu(puptime,userid)

idx_t1_pu

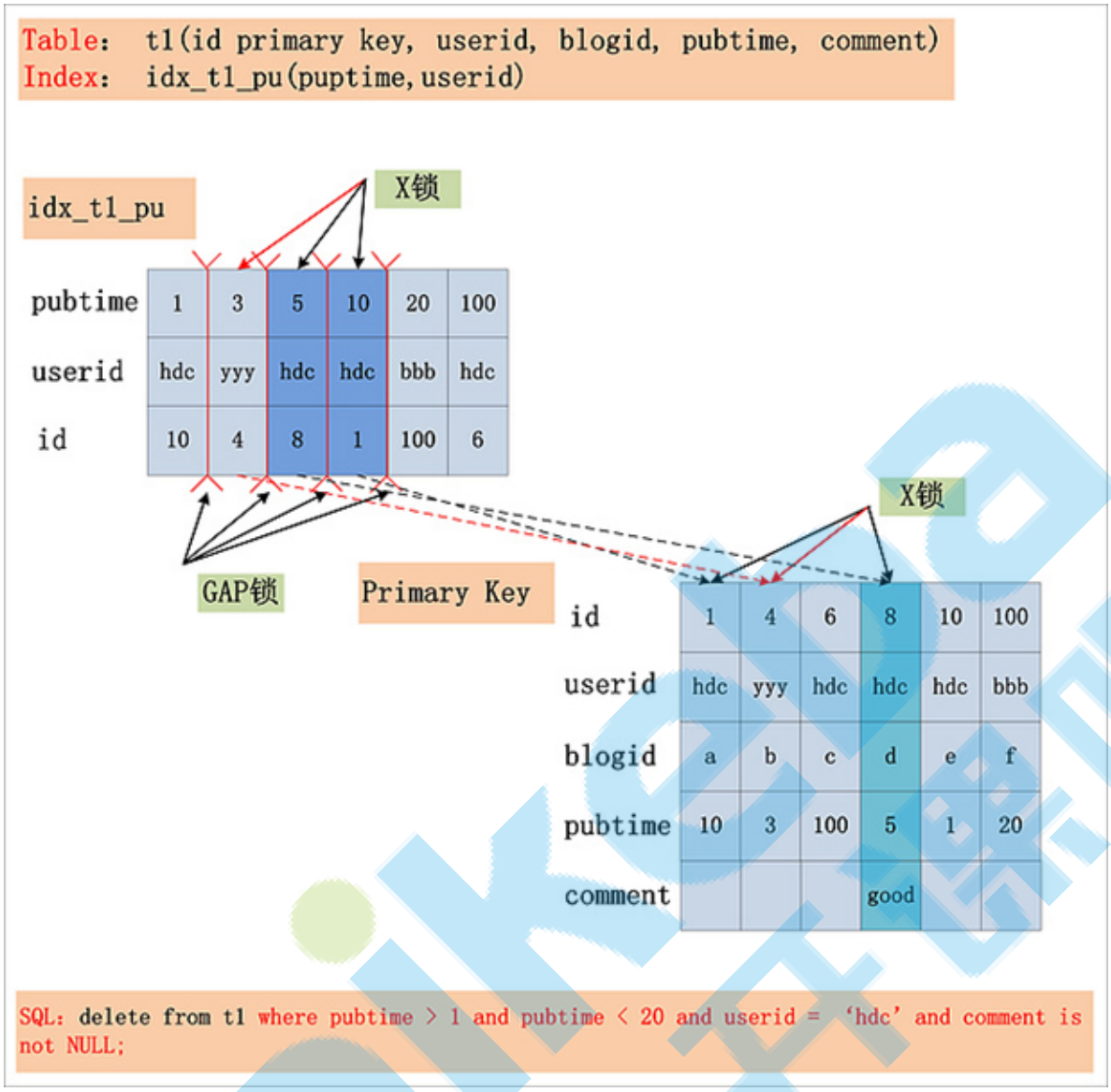
pubtime	1	3	5	10	20	100
userid	hdc	yyy	hdc	hdc	bbb	hdc
id	10	4	8	1	100	6

Primary Key

id	1	4	6	8	10	100
userid	hdc	yyy	hdc	hdc	hdc	bbb
blogid	a	b	c	d	e	f
pubtime	10	3	100	5	1	20
comment				good		

SQL: delete from t1 where pubtime > 1 and pubtime < 20 and userid = 'hdc' and comment is not NULL;

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com



如图中的SQL，会加什么锁？假定在**Repeatable Read**隔离级别下

在详细分析这条SQL的加锁情况前，还需要有一个知识储备，那就是一个SQL中的where条件如何拆分？在这里，我直接给出分析后的结果：

- **Index key：** pubtime > 1 and puptime < 20。此条件，用于确定SQL在idx_t1_pu索引上的查询范围。
- **Index Filter：** userid = 'hdc'。此条件，可以在idx_t1_pu索引上进行过滤，但不属于Index Key。
- **Table Filter：** comment is not NULL。此条件，在idx_t1_pu索引上无法过滤，只能在聚簇索引上过滤。

在where条件过滤时，先过滤index key（索引列为范围查询，起始条件为index First Key，截至条件为index Last key），再过滤Index Filter（索引列），最后过滤Table Filter（非索引列）。在ICP过程中，下推Index Filter。

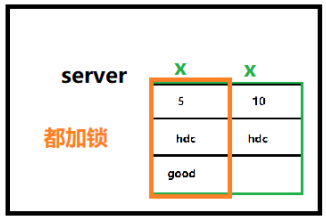
结论：

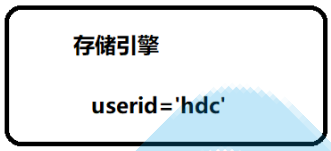
```
在Repeatable Read隔离级别下，针对一个复杂的SQL，首先需要提取其where条件。
 - Index Key确定的范围，需要加上GAP锁；
 - Index Filter过滤条件，视MySQL版本是否支持ICP，若支持ICP，则不满足Index Filter的记录，不加X锁，否则需要X锁；
 - Table Filter过滤条件，无论是否满足，都需要加X锁。 server层
```

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

在存储引擎层
无锁

3	5	10
yyy	hdc	hdc
	good	





explain delete from where xxxx
extra
using index condition 表示索引下推

死锁原理与分析

本文前面的部分，基本上已经涵盖了MySQL/InnoDB所有的加锁规则。深入理解MySQL如何加锁，有两个比较重要的作用：

- 可以根据MySQL的加锁规则，写出不会发生死锁的SQL；
- 可以根据MySQL的加锁规则，定位出线上产生死锁的原因；

案例1（记录锁产生）

```
1、session1: begin;--开启事务未提交
             --手动加行写锁 id=1 ，使用索引
             update mylock set name='m' where id=1;
2、session2: begin;--开启事务未提交
             --手动加行写锁 id=2 ，使用索引
             update mylock set name='m' where id=2;

3、session1: update mylock set name='nn' where id=2; -- 加写锁被阻塞
4、session2: update mylock set name='nn' where id=1; -- 加写锁会死锁，不允许操作
     ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting
transaction
```

案例2（间隙锁产生）

```
1、session1: start  transaction ;
             select * from news where number=6 for update--产生间隙锁
2、session2: start  transaction ;
             select * from news where number=7 for update--产生间隙锁
3、session1: insert into news values(9,7);--阻塞
4、session2: insert into news values(9,7);
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting
transaction


注：可以用 show engine innodb status \G;查看死锁 情况
```

结论：

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

死锁的发生与否，并不在于事务中有多少条SQL语句，【死锁的关键在于】：两个(或以上)的Session【加锁的顺序】不一致。而使用本文上面提到的，分析MySQL每条SQL语句的加锁规则，分析出每条语句的加锁顺序，然后检查多个并发SQL间是否存在以相反的顺序加锁的情况，就可以分析出各种潜在的死锁情况，也可以分析出线上死锁发生的原因。

如何解决死锁呢?

MySQL默认会主动探知死锁，并回滚某一个影响最小的事务。等另一事务执行完成之后，再重新执行该事务。

如何避免死锁

1、注意程序的逻辑

根本的原因是程序逻辑的顺序，最常见的是交差更新

Transaction 1: 更新表A -> 更新表B

Transaction 2: 更新表B -> 更新表A

Transaction获得两个资源

2、保持事务的轻量

越是轻量的事务，占有越少的锁资源，这样发生死锁的几率就越小

3、提高运行的速度

避免使用子查询，尽量使用主键等等

4、尽量快提交事务，减少持有锁的时间

越早提交事务，锁就越早释放

课堂主题

MySQL性能分析和性能优化

课堂目标

会使用和分析慢查询日志

会使用和分析profile

理解服务器层面优化思路

掌握表设计层面优化

掌握SQL层面优化技术

性能分析和性能优化篇

性能分析的思路

1. 首先需要使用【慢查询日志】功能，去获取所有查询时间比较长的SQL语句
2. 其次【查看执行计划】查看有问题的SQL的执行计划 explain
3. 最后可以使用【show profile[s]】 查看有问题的SQL的性能使用情况

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

慢查询日志

开启:

```
slow_query_log=ON
long_query_time=3
slow_query_log_file=/var/lib/mysql/slow-log.log
```

慢查询日志介绍

开启慢查询功能

```
mysql> show variables like '%slow_query%';
+---------------------+-----------------------------------+
| Variable_name       | Value                             |
+---------------------+-----------------------------------+
| slow_query_log      | OFF                               |
| slow_query_log_file | /var/lib/mysql/localhost-slow.log |
+---------------------+-----------------------------------+
2 rows in set (0.00 sec)

mysql> SHOW VARIABLES LIKE 'long_query_time%';
+-----------------+-----------+
| Variable_name   | Value     |
+-----------------+-----------+
| long_query_time | 10.000000 |
+-----------------+-----------+
1 row in set (0.00 sec)
```

慢查询日志格式

```
# User@Host: root[root] @ localhost []  Id:     2
# Query_time: 3.120275  Lock_time: 0.000110 Rows_sent: 1  Rows_examined: 10000000
SET timestamp=1564990057;
select count(*) from tuser;
# Time: 190805  0:27:48
# User@Host: root[root] @ localhost []  Id:     2
# Query_time: 3.375150  Lock_time: 0.000112 Rows_sent: 1  Rows_examined: 10000000
SET timestamp=1564990068;
select count(*) from tuser order by id;
# Time: 190805  0:27:54
# User@Host: root[root] @ localhost []  Id:     2
# Query_time: 3.411974  Lock_time: 0.000089 Rows_sent: 1  Rows_examined: 10000000
SET timestamp=1564990074;
select count(*) from tuser order by id;
# Time: 190805  0:28:48
# User@Host: root[root] @ localhost []  Id:     2
# Query_time: 10.052493  Lock_time: 0.000135 Rows_sent: 1  Rows_examined: 10000000
SET timestamp=1564990128;
select count(name) from tuser;
# Time: 190805  0:29:00
# User@Host: root[root] @ localhost []  Id:     2
# Query_time: 3.636800  Lock_time: 0.000971 Rows_sent: 1  Rows_examined: 10000000
SET timestamp=1564990140;
select count(id) from tuser;
# Time: 190805  0:31:21
# User@Host: root[root] @ localhost []  Id:     2
# Query_time: 11.080994  Lock_time: 0.000143 Rows_sent: 1  Rows_examined: 10000000
SET timestamp=1564990281;
select * from tuser order by id limit 9999999 ,1;
# Time: 190805  0:31:51
# User@Host: root[root] @ localhost []  Id:     2
# Query_time: 13.223564  Lock_time: 0.001181 Rows_sent: 1  Rows_examined: 10000000
SET timestamp=1564990311;
select * from tuser order by id limit 9999999 ,1;
```

分析慢查询日志的工具

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

使用mysqldumpslow工具

mysqldumpslow是MySQL自带的慢查询日志工具。

可以使用mysqldumpslow工具搜索慢查询日志中的SQL语句。

得到按照时间排序的前10条里面含有左连接的查询语句：

```
[root@localhost mysql]# mysqldumpslow -s t -t 10 -g “left join”
/var/log/mysql/slow.log
```

常用参数说明：

-s：是表示按照何种方式排序

c：访问计数

l：锁定时间

r：返回记录

t：查询时间

al：平均锁定时间

ar：平均返回记录数

at：平均查询时间

-t：是top n的意思，即为返回前面多少条的数据

-g：后边可以写一个正则匹配模式，大小写不敏感的

使用percona-toolkit工具

percona-toolkit是一组高级命令行工具的集合，可以查看当前服务的摘要信息，磁盘检测，分析慢查询日志，查找重复索引，实现表同步等等。

- 下载

https://www.percona.com/downloads/percona-toolkit/3.0.11/binary/tarball/percona-toolkit-3.0.11_x86_64.tar.gz

```
wget https://www.percona.com/downloads/percona-
toolkit/3.0.11/binary/tarball/percona-toolkit-3.0.11_x86_64.tar.gz
```

- 安装

```
tar -xf percona-toolkit-3.0.11_x86_64.tar.gz
cd percona-toolkit-3.0.11
perl Makefile.PL
make
make install
```

- 调错

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

Can't locate ExtUtils/MakeMaker.pm in @INC 错误的解决方式:

先执行再安装

```
yum install -y perl-ExtUtils-CBuilder perl-ExtUtils-MakeMaker
```

Can't locate Time/HiRes.pm in @INC

```
yum install -y perl-Time-HiRes
```

Can't locate Digest/MD5.pm in @INC

```
yum install perl-Digest-MD5.x86_64
```

- 使用pt-query-digest查看慢查询日志

```
pt-query-digest /var/lib/mysql/localhost-slow.log
```

pt-query-digest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。
```

分析pt-query-digest输出结果

- 第一部分：总体统计结果 Overall：总共有多少条查询 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 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

```
# 该工具执行日志分析的用户时间，系统时间，物理内存占用大小，虚拟内存占用大小
# 340ms user time, 140ms system time, 23.99M rss, 203.11M vsz
# 工具执行时间
# Current date: Fri Nov 25 02:37:18 2016
# 运行分析工具的主机名
# Hostname: localhost.localdomain
```

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

```
# 被分析的文件名
# Files: slow.log
# 语句总数量，唯一的语句数量，QPS，并发数
# Overall: 2 total, 2 unique, 0.01 QPS, 0.01x concurrency _________________
# 日志记录的时间范围
# Time range: 2016-11-22 06:06:18 to 06:11:40
# 属性                 总计       最小     最大     平均     95%  标准     中等
# Attribute          total     min     max     avg     95%  stddev  median
# ============     ======= ======= ======= ======= ======= ======= =======
# 语句执行时间
# Exec time             3s   640ms      2s      1s      2s   999ms      1s
# 锁占用时间
# Lock time            1ms       0     1ms   723us     1ms     1ms   723us
# 发送到客户端的行数
# Rows sent              5       1       4    2.50       4    2.12    2.50
# select语句扫描行数
# Rows examine     186.17k       0 186.17k  93.09k 186.17k 131.64k  93.09k
# 查询的字符数
# Query size           455      15     440  227.50     440  300.52  227.50
```

- 第二部分：查询分组统计结果 Rank：所有语句的排名，默认按查询时间降序排列，通过--order-by指定 Query ID：语句的ID，（去掉多余空格和文本字符，计算hash值） 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 V/M：响应时间Variance-to-mean的比率 Item：查询对象

```
# Profile
# Rank Query ID           Response time Calls R/Call V/M   Item
# ==== ================== ============= ===== ====== ===== ===============
#    1 0xF9A57DD5A41825CA  2.0529 76.2%     1 2.0529  0.00 SELECT
#    2 0x4194D8F83F4F9365  0.6401 23.8%     1 0.6401  0.00 SELECT wx_member_base
```

- 第三部分：每一种查询的详细统计结果 由下面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。 ID：查询的ID号，和上图的Query ID对应 Databases：数据库名 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之间查询数量是10s以上的两倍。 Tables：查询中涉及到的表 Explain：SQL语句

```
# Query 1: 0 QPS, 0x concurrency, ID 0xF9A57DD5A41825CA at byte 802 ______
# This item is included in the report because it matches --limit.
# Scores: V/M = 0.00
# Time range: all events occurred at 2016-11-22 06:11:40
# Attribute    pct   total     min     max     avg     95%  stddev  median
# ============ === ======= ======= ======= ======= ======= ======= =======
# Count         50       1
# Exec time     76      2s      2s      2s      2s      2s       0      2s
# Lock time      0       0       0       0       0       0       0       0
# Rows sent     20       1       1       1       1       1       0       1
# Rows examine   0       0       0       0       0       0       0       0
# Query size     3      15      15      15      15      15       0      15
# String:
# Databases    test
# Hosts        192.168.8.1
# Users        mysql
# Query_time distribution
```

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

```
#   1us
#  10us
# 100us
#   1ms
#  10ms
# 100ms
#    1s  ########################################################################
#  10s+
# EXPLAIN /*!50100 PARTITIONS*/
select sleep(2)\G
```

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 '2017-01-07 09:30:00' --until '2017-01-07
10:00:00'> > slow_report3.log
```

4.分析指含有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query_review表

```
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_0001
pt-query-digest  --user=root -password=abc123 --review
h=localhost,D=test,t=query_history--create-review-table  slow.log_0002
```

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

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
```

---

pt-query-digest语法及重要选项

profile分析语句

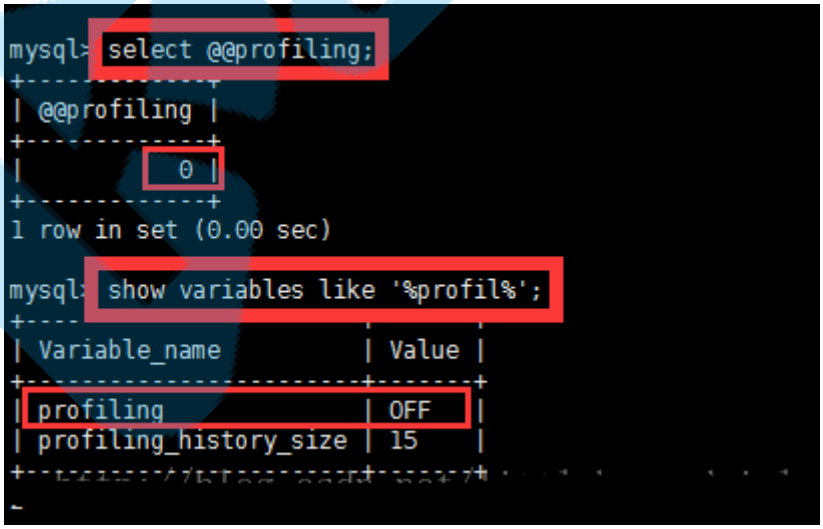
Query Profiler是MySQL自带的一种**query诊断分析工具**，通过它可以分析出一条SQL语句的**硬件性能瓶颈**在什么地方。比如CPU，IO等，以及该SQL执行所耗费的时间等。不过该工具只有在MySQL 5.0.37以及以上版本中才有实现。默认的情况下，MYSQL的该功能没有打开，需要自己手动启动。

开启Profile功能

- Profile 功能由MySQL会话变量 : **profiling**控制,默认是**OFF**关闭状态。
- 查看是否开启了Profile功能:

```
select @@profiling;

show variables like '%profil%';
```



- 开启profile功能

```
set profiling=1; --1是开启、0是关闭
```

示例

```
```

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

```
mysql> select @@profiling;
+-------------+
| @@profiling |
+-------------+
|           0 |
+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> set profiling=1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> select @@profiling;
+-------------+
| @@profiling |
+-------------+
|           1 |
+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> select count(id) from tuser;
ERROR 1046 (3D000): No database selected
mysql> use kkb_2;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> select count(id) from tuser;
+-----------+
| count(id) |
+-----------+
|  10000000 |
+-----------+
1 row in set (4.62 sec)

mysql> show profiles;
+----------+------------+----------------------------+
| Query_ID | Duration   | Query                      |
+----------+------------+----------------------------+
|        1 | 0.00016275 | select @@profiling         |
|        2 | 0.00009200 | select count(id) from tuser |
|        3 | 0.00014875 | SELECT DATABASE()          |
|        4 | 0.00066875 | show databases             |
|        5 | 0.00021050 | show tables                |
|        6 | 4.61513875 | select count(id) from tuser |
+----------+------------+----------------------------+
6 rows in set, 1 warning (0.13 sec)

mysql> show profile for query 6;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| starting             | 0.000228 |
| checking permissions | 0.000018 |
| Opening tables       | 0.000035 |
| init                 | 0.000204 |
| System lock          | 0.000071 |
| optimizing           | 0.000013 |
| statistics           | 0.000067 |
```

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

```
| preparing              | 0.000027 |
| executing              | 0.000004 |
| Sending data           | 4.614239 |
| end                    | 0.000045 |
| query end              | 0.000009 |
| closing tables         | 0.000026 |
| freeing items          | 0.000019 |
| logging slow query     | 0.000124 |
| cleaning up            | 0.000011 |
+----------------------+----------+
16 rows in set, 1 warning (0.00 sec)

mysql> show profile cpu,block io,swaps for query 6;
+----------------------+----------+----------+------------+--------------+------
--------+-------+
| Status               | Duration | CPU_user | CPU_system | Block_ops_in |
Block_ops_out | Swaps |
+----------------------+----------+----------+------------+--------------+------
--------+-------+
| starting             | 0.000228 | 0.000361 |   0.000000 |            0 |
      0 |     0 |
| checking permissions | 0.000018 | 0.000000 |   0.000000 |            0 |
      0 |     0 |
| Opening tables       | 0.000035 | 0.000000 |   0.000000 |            0 |
      0 |     0 |
| init                 | 0.000204 | 0.000224 |   0.000000 |            0 |
      0 |     0 |
| System lock          | 0.000071 | 0.000000 |   0.000000 |            0 |
      0 |     0 |
| optimizing           | 0.000013 | 0.000000 |   0.000000 |            0 |
      0 |     0 |
| statistics           | 0.000067 | 0.000131 |   0.000000 |            0 |
      0 |     0 |
| preparing            | 0.000027 | 0.000000 |   0.000000 |            0 |
      0 |     0 |
| executing            | 0.000004 | 0.000000 |   0.000000 |            0 |
      0 |     0 |
| Sending data         | 4.614239 | 3.648639 |   0.543410 |        55280 |
      0 |     0 |
| end                  | 0.000045 | 0.000202 |   0.000000 |            0 |
      0 |     0 |
| query end            | 0.000009 | 0.000000 |   0.000000 |            0 |
      0 |     0 |
| closing tables       | 0.000026 | 0.000000 |   0.000000 |            0 |
      0 |     0 |
| freeing items        | 0.000019 | 0.000000 |   0.000000 |            0 |
      0 |     0 |
| logging slow query   | 0.000124 | 0.000155 |   0.000000 |            0 |
      8 |     0 |
| cleaning up          | 0.000011 | 0.000000 |   0.000000 |            0 |
      0 |     0 |
+----------------------+----------+----------+------------+--------------+------
--------+-------+
```

服务器层面优化

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

将数据保存在内存中，保证从内存读取数据

buffer pool 默认128M

扩大buffer pool 理论上内存的3/4或4/5

怎样确定 innodb_buffer_pool_size 足够大。数据是从内存读取而不是硬盘?

```
mysql> show global status like 'innodb_buffer_pool_pages_%';
+----------------------------------+-------+
| Variable_name                    | Value |
+----------------------------------+-------+
| Innodb_buffer_pool_pages_data    | 8190  |
| Innodb_buffer_pool_pages_dirty   | 0     |
| Innodb_buffer_pool_pages_flushed | 12646 |
| Innodb_buffer_pool_pages_free    | 0     |    0 表示已经被用光
| Innodb_buffer_pool_pages_misc    | 1     |
| Innodb_buffer_pool_pages_total   | 8191  |
+----------------------------------+-------+
```

修改 my.cnf

innodb_buffer_pool_size = 750M

内存预热

```
mysql> select count(id) from tuser;
+-----------+
| count(id) |
+-----------+
|  10000000 |
+-----------+
1 row in set (5.03 sec)

mysql> select count(id) from tuser;
+-----------+
| count(id) |
+-----------+
|  10000000 |
+-----------+
1 row in set (2.85 sec)
```

降低磁盘写入次数

1、redolog 大 落盘次数少

innodb_log_file_size 设置成 innodb_buffer_pool_size * 0.25

2、通用查询日志、慢查询日志 可以不开 bin-log 开

3、写redolog策略 innodb_flush_log_at_trx_commit 0 1 2

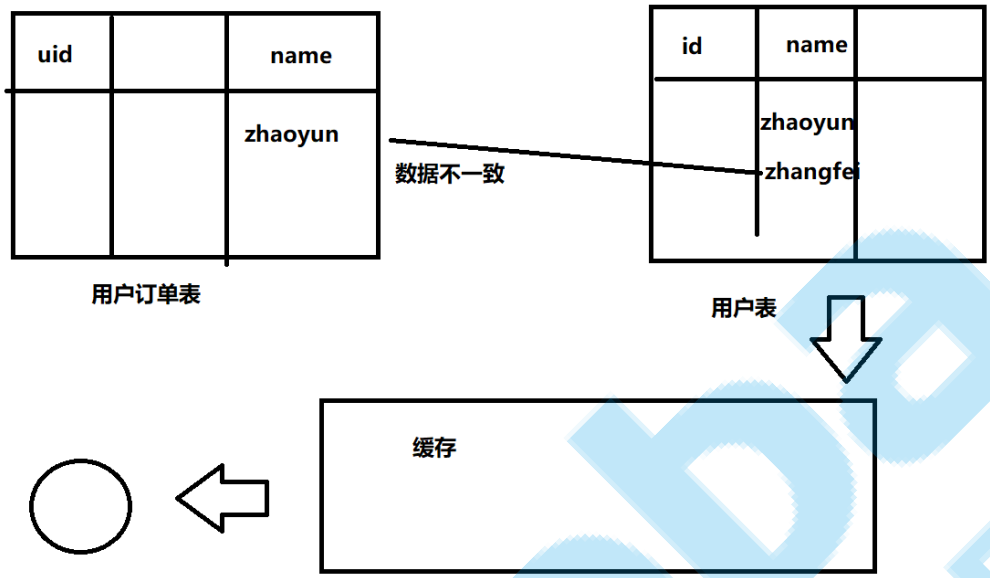
提高磁盘读写

SSD

SQL设计层面优化

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

- 设计中间表，**一般针对于**统计分析功能，或者实时性不高的需求（OLTP、OLAP）
- 为减少关联查询，创建合理的**冗余字段**（考虑数据库的三范式和查询性能的取舍，创建冗余字段还需要注意**数据一致性问题**）



- 对于字段太多的大表，考虑**拆表**（比如一个表有100多个字段） 人和身份证
- 对于表中经常不被使用的字段或者存储数据比较多的字段，考虑拆表（比如商品表中会存储商品介绍，此时可以将商品介绍字段单独拆解到另一个表中，使用商品ID关联）
- 每张表建议都要有一个主键（**主键索引**），而且主键类型最好是**int类型**，建议自增主键（**不考虑分布式系统的情况下**）。

SQL语句优化

索引优化

where 字段 、组合索引 （最左前缀） 、 索引下推 （非选择行 不加锁） 、索引覆盖（不回表）

on 两边 排序 分组统计

不要用 *

LIMIT优化

原SQL

```
mysql> select * from tuser limit 9999999,1;
+----------+-----------+-----------------+------+------+------+---------+
| id       | loginname | name            | age  | sex  | dep  | address |
+----------+-----------+-----------------+------+------+------+---------+
| 10000000 | zy10000000 | zhaoyun10000000 |   23 | 1    |    1 | beijing |
+----------+------------+-----------------+------+------+------+---------+
1 row in set (15.70 sec)
```

优化：

limit 是可以停止全表扫描的

PDF-XChange Editor Click to BUY NOW! www.tracker-software.com

```
mysql> select * from tuser order by id desc limit 1;
+----------+------------+-----------------+------+------+------+---------+
| id       | loginname  | name            | age  | sex  | dep  | address |
+----------+------------+-----------------+------+------+------+---------+
| 10000000 | zy10000000 | zhaoyun10000000 |   23 | 1    |    1 | beijing |
+----------+------------+-----------------+------+------+------+---------+
1 row in set (0.10 sec)
```

定位

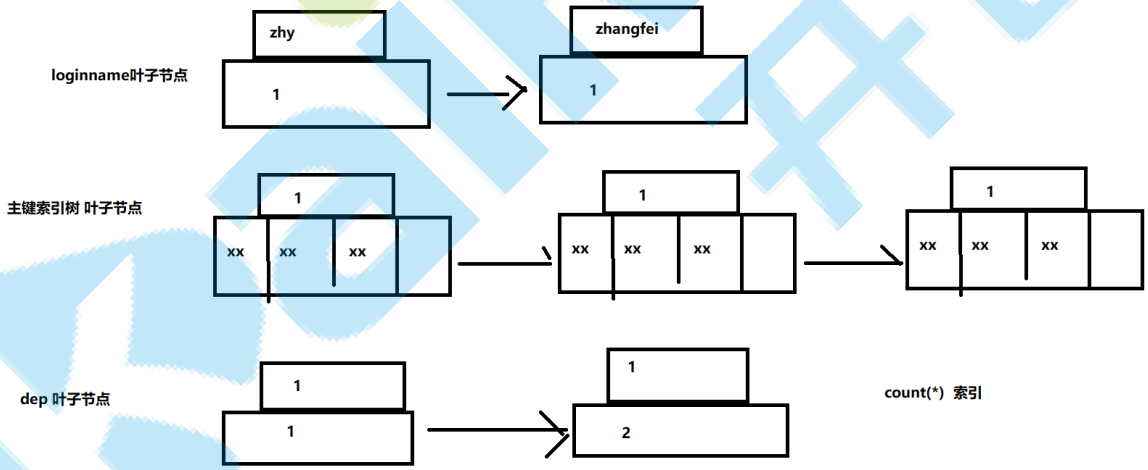
```
mysql> select * from tuser where id>9999999 limit 1;
                                                    +----------+------------+--
---------------+------+------+------+---------+
| id       | loginname  | name            | age  | sex  | dep  | address |
+----------+------------+-----------------+------+------+------+---------+
| 10000000 | zy10000000 | zhaoyun10000000 |   23 | 1    |    1 | beijing |
+----------+------------+-----------------+------+------+------+---------+
1 row in set (0.00 sec)
```

其他优化

count (*) 找普通索引 ,找到最小的那棵树来遍历 包含空值

count（字段） 走缓存 不包含空值

count(1) 忽略字段 包含空值



不用 MySQL 内置的函数，因为内置函数不会建立查询缓存。

SELECT * FROM user where birthday = now();