课堂主题

Mysql组合索引、索引失效分析、表级锁介绍

课堂目标

掌握索引使用场景

理解组合索引的结构和掌握使用原则

使用explain查看sql执行计划

掌握select_type、type、extra等参数意义

理解索引失效口诀

编写使用索引的sql

掌握MySQL的锁的分类

理解表级锁

一、索引使用场景

哪些情况需要创建索引

1、主键自动建立唯一索引

2、频繁作为查询条件的字段应该创建索引 where

3、多表关联查询中，关联字段应该创建索引 on 两边都要创建索引

4、查询中排序的字段，应该创建索引 B + tree 有顺序

5、覆盖索引 好处是？ 不需要回表 组合索引

6、统计或者分组字段，应该创建索引

哪些情况不需要创建索引

1、表记录太少 索引是要有存储的开销

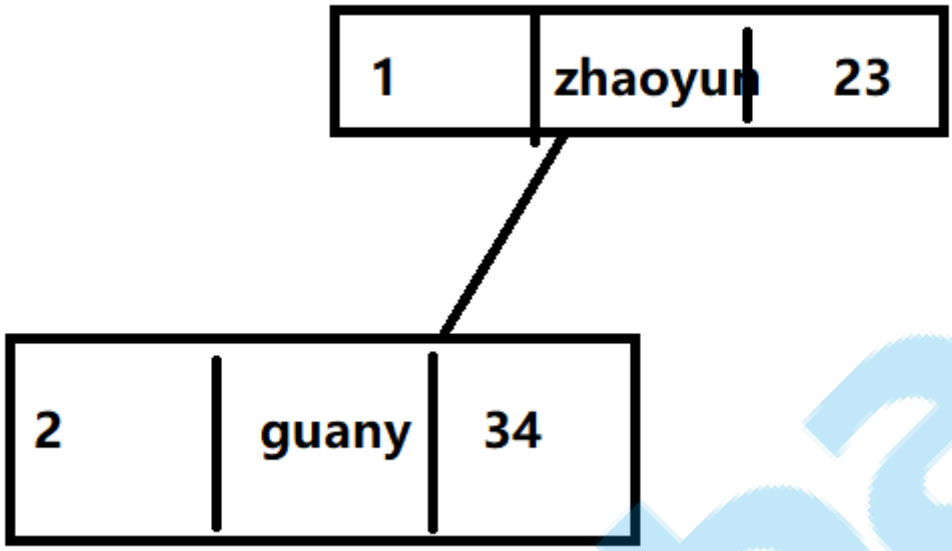
2、频繁更新 索引要维护

3、查询字段使用频率不高

为什么使用组合索引

由多个字段组成的索引 使用顺序就是创建的顺序

```
ALTER TABLE 'table_name' ADD INDEX index_name(col1,col2,col3)
```



在一颗索引树上由多个字段

优势： 效率高、省空间、容易形成覆盖索引

使用：

遵循最左前缀原则

1、前缀索引

like 常量% 使用索引 like %常量 不使用索引

2、最左前缀

从左向右匹配直到遇到范围查询 > < between 索引失效

```
mysql> alter table t1 add index idx_a_b_c_d(a,b,c,d);
Query OK, 0 rows affected (0.56 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> show index from t1;
+-------+------------+-------------+--------------+-------------+-----------+---
----------+----------+--------+------+------------+---------+---------------+
| Table | Non_unique | Key_name    | Seq_in_index | Column_name | Collation |
Cardinality | Sub_part | Packed | Null | Index_type | Comment | Index_comment |
+-------+------------+-------------+--------------+-------------+-----------+---
----------+----------+--------+------+------------+---------+---------------+
| t1    |          0 | PRIMARY     |            1 | id          | A         |
        1 |     NULL | NULL   |      | BTREE      |         |               |
| t1    |          1 | idx_a_b_c_d |            1 | a           | A         |
        1 |     NULL | NULL   | YES  | BTREE      |         |               |
| t1    |          1 | idx_a_b_c_d |            2 | b           | A         |
        1 |     NULL | NULL   | YES  | BTREE      |         |               |
| t1    |          1 | idx_a_b_c_d |            3 | c           | A         |
        1 |     NULL | NULL   | YES  | BTREE      |         |               |
| t1    |          1 | idx_a_b_c_d |            4 | d           | A         |
        1 |     NULL | NULL   | YES  | BTREE      |         |               |
+-------+------------+-------------+--------------+-------------+-----------+---
----------+----------+--------+------+------------+---------+---------------+
5 rows in set (0.00 sec)
```

```
mysql> explain select * from t1 where a=1 and b=1 and c=1 and d=1 ;
+----+-------------+-------+------+---------------+-------------+---------+-----
--------------------+------+-------------+
| id | select_type | table | type | possible_keys | key         | key_len | ref
                    | rows | Extra       |
+----+-------------+-------+------+---------------+-------------+---------+-----
--------------------+------+-------------+
|  1 | SIMPLE      | t1    | ref  | idx_a_b_c_d   | idx_a_b_c_d | 20      |
const,const,const,const |    1 | Using index |
+----+-------------+-------+------+---------------+-------------+---------+-----
--------------------+------+-------------+
1 row in set (0.00 sec)

mysql> explain select * from t1 where a=1 and b=1 and c>1 and d=1 ;
+----+-------------+-------+-------+---------------+-------------+---------+----
--+------+-----------------------+
| id | select_type | table | type  | possible_keys | key         | key_len | ref
| rows | Extra                 |
+----+-------------+-------+-------+---------------+-------------+---------+----
--+------+-----------------------+
|  1 | SIMPLE      | t1    | range | idx_a_b_c_d   | idx_a_b_c_d | 15      |
NULL |    1 | Using index condition |
+----+-------------+-------+-------+---------------+-------------+---------+----
--+------+-----------------------+
1 row in set (0.01 sec)


mysql> explain select * from t1 where a=1 and b=1 and d=1 and  c>1 ;
+----+-------------+-------+------+---------------+-------------+---------+-----
--------+------+-------------------------+
| id | select_type | table | type | possible_keys | key         | key_len | ref
        | rows | Extra                   |
+----+-------------+-------+------+---------------+-------------+---------+-----
--------+------+-------------------------+
|  1 | SIMPLE      | t1    | ref  | idx_a_b_c_d   | idx_a_b_c_d | 15      |
const,const |    1 | Using where; Using index |
+----+-------------+-------+------+---------------+-------------+---------+-----
--------+------+-------------------------+
1 row in set (0.00 sec)

mysql> drop index idx_a_b_c_d on t1;
Query OK, 0 rows affected (0.00 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> alter table t1 add index idx_a_b_c_d(a,b,d,c);
Query OK, 0 rows affected (0.01 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain select * from t1 where a=1 and b=1 and d=1 and  c>1 ;
+----+-------------+-------+------+---------------+-------------+---------+-----
--------------+------+-------------------------+
| id | select_type | table | type | possible_keys | key         | key_len | ref
              | rows | Extra                   |
+----+-------------+-------+------+---------------+-------------+---------+-----
--------------+------+-------------------------+
|  1 | SIMPLE      | t1    | ref  | idx_a_b_c_d   | idx_a_b_c_d | 20      |
const,const,const |    1 | Using where; Using index |
```

```
+----+-------------+-------+------+---------------+-------------+---------+-----
-------------+------+-------------------------+
1 row in set (0.00 sec)
```

二、索引失效

查看执行计划

参数说明

explain出来的信息有10列，分别是

```
id、select_type、table、type、possible_keys、key、key_len、ref、rows、Extra
```

案例表

```
--用户表
create table tuser(
id int primary key,
loginname varchar(100),
name varchar(100),
age int,
sex char(1),
dep int,
address varchar(100)
);
--部门表
create table tdep(
id int primary key,
name varchar(100)
);

--地址表
create table taddr(
id int primary key,
addr varchar(100)
);

--创建普通索引
mysql> alter table tuser add index idx_dep(dep);
--创建唯一索引
mysql> alter table tuser add unique index idx_loginname(loginname);
--创建组合索引
mysql> alter table tuser add index idx_name_age_sex(name,age,sex);
--创建全文索引
mysql> alter table taddr add fulltext ft_addr(addr);
```

id

- 每个 SELECT语句都会自动分配的一个唯一标识符.

- 表示查询中操作表的顺序，有三种情况：
  - id相同：执行顺序由上到下
  - id不同：如果是子查询，id号会自增，**id越大，优先级越高。**
  - id相同的不同的同时存在
- id列为null的就表示这是一个结果集，不需要使用它来进行查询。

select_type（重要）

查询类型，主要用于区别**普通查询、联合查询(union、union all)、子查询等复杂查询。**

simple

表示不需要union操作或者不包含子查询的简单select查询。有连接查询时，外层的查询为simple，且只有一个

```
mysql> explain select * from tuser;
+----+-------------+-------+------+---------------+------+---------+------+------
-+-------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra |
+----+-------------+-------+------+---------------+------+---------+------+------
-+-------+
|  1 | SIMPLE      | tuser | ALL  | NULL          | NULL | NULL    | NULL |    1
| NULL  |
+----+-------------+-------+------+---------------+------+---------+------+------
-+-------+
```

primary

一个需要union操作或者含有子查询的select，位于最外层的单位查询的select_type即为primary。且只有一个

```
mysql> explain select (select name from tuser) from tuser ;
+----+-------------+-------+-------+---------------+-----------------+---------
+------+------+-------------+
| id | select_type | table | type  | possible_keys | key             | key_len
| ref  | rows | Extra       |
+----+-------------+-------+-------+---------------+-----------------+---------
+------+------+-------------+
|  1 | PRIMARY     | tuser | index | NULL          | idx_dep         | 5
| NULL |    1 | Using index |
|  2 | SUBQUERY    | tuser | index | NULL          | idx_name_age_sex | 312
| NULL |    1 | Using index |
+----+-------------+-------+-------+---------------+-----------------+---------
+------+------+-------------+
```

subquery

除了from字句中包含的子查询外，其他地方出现的子查询都可能是subquery

```
mysql> explain select * from tuser where  id = (select max(id) from tuser);
                                                    +----+-------------+-------
+-------+--------------+---------+---------+-------+------+--------------------
----------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref   |
rows | Extra                         |
+----+-------------+-------+-------+---------------+---------+---------+-------
+------+-------------------------------+
|  1 | PRIMARY     | tuser | const | PRIMARY       | PRIMARY | 4       | const |
   1 | NULL                          |
|  2 | SUBQUERY    | NULL  | NULL  | NULL          | NULL    | NULL    | NULL  |
NULL | Select tables optimized away |
+----+-------------+-------+-------+---------------+---------+---------+-------
+------+-------------------------------+
```

dependent subquery

与dependent union类似，表示这个subquery的查询要受到外部表查询的影响

```
mysql> explain select id,name,(select name from tdep a where a.id=b.dep) from
tuser b;
+----+--------------------+-------+--------+---------------+---------+---------
+-------------+------+--------+
| id | select_type        | table | type   | possible_keys | key     | key_len |
ref         | rows | Extra |
+----+--------------------+-------+--------+---------------+---------+---------
+-------------+------+--------+
|  1 | PRIMARY            | b     | ALL    | NULL          | NULL    | NULL    |
NULL        |    2 | NULL  |
|  2 | DEPENDENT SUBQUERY | a     | eq_ref | PRIMARY       | PRIMARY | 4       |
demo1.b.dep |    1 | NULL  |
+----+--------------------+-------+--------+---------------+---------+---------
+-------------+------+--------+
```

union

union连接的两个select查询，第一个查询是PRIMARY，除了第一个表外，第二个以后的表select_type都是union

```
mysql> explain select * from tuser where sex='1'  union select * from tuser
where sex='2';
+----+--------------+-----------+------+---------------+------+---------+------
+------+-----------------+
| id | select_type  | table     | type | possible_keys | key  | key_len | ref
| rows | Extra           |
+----+--------------+-----------+------+---------------+------+---------+------
+------+-----------------+
|  1 | PRIMARY      | tuser     | ALL  | NULL          | NULL | NULL    | NULL
|    2 | Using where     |
|  2 | UNION        | tuser     | ALL  | NULL          | NULL | NULL    | NULL
|    2 | Using where     |
| NULL | UNION RESULT | <union1,2> | ALL  | NULL          | NULL | NULL    |
NULL | NULL | Using temporary |
+----+--------------+-----------+------+---------------+------+---------+------
+------+-----------------+
```

dependent union

与union一样，出现在union 或union all语句中，但是这个查询要受到外部查询的影响

```
mysql> explain select * from tuser where sex in (select sex from tuser where
sex='1' union select sex from tuser where sex='2');
+----+--------------------+-----------+------+---------------+---------------
--+---------+------+------+--------------------------+
| id | select_type        | table     | type | possible_keys | key
| key_len | ref  | rows | Extra                    |
+----+--------------------+-----------+------+---------------+---------------
--+---------+------+------+--------------------------+
|  1 | PRIMARY            | tuser     | ALL  | NULL          | NULL
  | NULL    | NULL |    2 | Using where              |
|  2 | DEPENDENT SUBQUERY | tuser     | index | NULL         |
idx_name_age_sex | 312     | NULL |    2 | Using where; Using index |
|  3 | DEPENDENT UNION    | tuser     | index | NULL         |
idx_name_age_sex | 312     | NULL |    2 | Using where; Using index |
| NULL | UNION RESULT       | <union2,3> | ALL  | NULL          | NULL
    | NULL    | NULL | NULL | Using temporary          |
+----+--------------------+-----------+------+---------------+---------------
--+---------+------+------+--------------------------+
```

union result

包含union的结果集，在union和union all语句中,因为它不需要参与查询，所以id字段为null

derived

from字句中出现的子查询，也叫做派生表，其他数据库中可能叫做内联视图或嵌套select

```
mysql> explain select * from (select * from tuser where sex='1') b;
+----+-------------+------------+------+---------------+------+---------+------
+------+-------------+
| id | select_type | table      | type | possible_keys | key  | key_len | ref  |
rows | Extra       |
+----+-------------+------------+------+---------------+------+---------+------
+------+-------------+
|  1 | PRIMARY     | <derived2> | ALL  | NULL          | NULL | NULL    | NULL |
   2 | NULL        |
|  2 | DERIVED     | tuser      | ALL  | NULL          | NULL | NULL    | NULL |
   2 | Using where |
+----+-------------+------------+------+---------------+------+---------+------
+------+-------------+
```

table

- 显示的查询表名，如果查询使用了别名，那么这里显示的是别名
- 如果不涉及对数据表的操作，那么这显示为null
- 如果显示为尖括号括起来的就表示这个是临时表，后边的N就是执行计划中的id，表示结果来自于这个查询产生。
- 如果是尖括号括起来的**<union M,N>**，与类似，也是一个临时表，表示这个结果来自于union查询的id为M,N的结果集。

type（重要）

- 依次从好到差：

```
system，const，eq_ref，ref，fulltext，ref_or_null，unique_subquery，
index_subquery，range，index_merge，index，ALL
```

除了all之外，其他的type都可以使用到索引，除了index_merge之外，其他的type只可以用到一个索引

优化器会选用最优索引 一个

- **注意事项：**

```
最少要索引使用到range级别。
```

system

表中只有一行数据或者是空表。

```
mysql> explain select * from (select * from tuser where id=1) a;
+----+-------------+------------+--------+---------------+---------+---------+--
-----+------+-------+
| id | select_type | table      | type   | possible_keys | key     | key_len |
ref   | rows | Extra |
+----+-------------+------------+--------+---------------+---------+---------+--
-----+------+-------+
|  1 | PRIMARY     | <derived2> | system | NULL          | NULL    | NULL    |
NULL  |    1 | NULL  |
|  2 | DERIVED     | tuser      | const  | PRIMARY       | PRIMARY | 4       |
const |    1 | NULL  |
+----+-------------+------------+--------+---------------+---------+---------+--
-----+------+-------+
```

const（重要）

使用**唯一索引或者主键**，返回记录一定是1行记录的等值where条件时，通常type是const。其他数据库也叫做唯一索引扫描

```
mysql> explain select * from tuser where id=1;
+----+-------------+-------+-------+---------------+---------+---------+-------
+------+-------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref   |
rows | Extra |
+----+-------------+-------+-------+---------------+---------+---------+-------
+------+-------+
|  1 | SIMPLE      | tuser | const | PRIMARY       | PRIMARY | 4       | const |
   1 | NULL  |
+----+-------------+-------+-------+---------------+---------+---------+-------
+------+-------+

mysql> explain select * from tuser where loginname = 'zhy';
+----+-------------+-------+-------+---------------+---------------+---------+--
-----+------+-------+
| id | select_type | table | type  | possible_keys | key           | key_len |
ref   | rows | Extra |
+----+-------------+-------+-------+---------------+---------------+---------+--
-----+------+-------+
|  1 | SIMPLE      | tuser | const | idx_loginname | idx_loginname | 303     |
const |    1 | NULL  |
+----+-------------+-------+-------+---------------+---------------+---------+--
-----+------+-------+
```

eq_ref（重要）

关键字:连接字段**主键或者唯一性索引。**

此类型通常出现在多表的 join 查询, 表示对于前表的每一个结果, **都只能匹配到后表的一行结果**. 并且查询的比较操作通常是 '=', 查询效率较高.

```
mysql> explain select a.id from tuser a left join tdep b on a.dep=b.id;
+----+-------------+-------+--------+---------------+---------+---------+-------
------+------+-------------+
| id | select_type | table | type   | possible_keys | key     | key_len | ref
     | rows | Extra       |
+----+-------------+-------+--------+---------------+---------+---------+-------
------+------+-------------+
|  1 | SIMPLE      | a     | index  | NULL          | idx_dep | 5       | NULL
     |    2 | Using index |
|  1 | SIMPLE      | b     | eq_ref | PRIMARY       | PRIMARY | 4       |
demo1.a.dep |    1 | Using index |
+----+-------------+-------+--------+---------------+---------+---------+-------
------+------+-------------+
```

select * from a,b where a.id=b.id (等值连接)

select * from a where name='zs' (条件查询)

ref（重要）

针对非唯一性索引，使用**等值（=）查询**非主键。或者是使用了**最左前缀规则索引的查询。**

```
--非唯一索引
mysql>  explain select * from tuser where dep=1;
+----+-------------+-------+------+---------------+---------+---------+-------+-
-----+-------+
| id | select_type | table | type | possible_keys | key     | key_len | ref   |
rows | Extra |
+----+-------------+-------+------+---------------+---------+---------+-------+-
-----+-------+
|  1 | SIMPLE      | tuser | ref  | idx_dep       | idx_dep | 5       | const |
   1 | NULL  |
+----+-------------+-------+------+---------------+---------+---------+-------+-
-----+-------+
--等值非主键连接
mysql> explain select a.id from tuser a left join tdep b on a.name=b.name;
+----+-------------+-------+-------+---------------+------------------+---------
+--------------+------+--------------------------+
| id | select_type | table | type  | possible_keys | key              | key_len
| ref          | rows | Extra                    |
+----+-------------+-------+-------+---------------+------------------+---------
+--------------+------+--------------------------+
|  1 | SIMPLE      | a     | index | NULL          | idx_name_age_sex | 312
| NULL         |    2 | Using index              |
|  1 | SIMPLE      | b     | ref   | ind_name      | ind_name         | 72
| demo1.a.name |    1 | Using where; Using index |
+----+-------------+-------+-------+---------------+------------------+---------
+--------------+------+--------------------------+
--最左前缀
mysql> explain select * from tuser where name = 'zhaoyun';
+----+-------------+-------+------+------------------+------------------+-------
--+-------+------+-----------------------+
| id | select_type | table | type | possible_keys    | key              |
key_len | ref   | rows | Extra                 |
```

```
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------+------+-----------------------+
|  1 | SIMPLE      | tuser | ref  | idx_name_age_sex | idx_name_age_sex | 303
  | const |    1 | Using index condition |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------+------+-----------------------+

思考：explain select * from tuser where sex = '1';
```

fulltext

全文索引检索，要注意，全文索引的优先级很高，若全文索引和普通索引同时存在时，mysql不管代价，优先选择使用全文索引

```
mysql> explain select * from taddr where match(addr) against('bei');
+----+-------------+-------+----------+---------------+---------+---------+-----
-+------+-------------+
| id | select_type | table | type     | possible_keys | key     | key_len | ref
| rows | Extra       |
+----+-------------+-------+----------+---------------+---------+---------+-----
-+------+-------------+
|  1 | SIMPLE      | tuser | fulltext | ft_addr       | ft_addr | 0       | NULL
|    1 | Using where |
+----+-------------+-------+----------+---------------+---------+---------+-----
-+------+-------------+
```

ref_or_null

与ref方法类似，只是增加了null值的比较。实际用的不多。

unique_subquery

用于where中的in形式子查询，子查询返回不重复值唯一值

index_subquery

用于in形式子查询使用到了辅助索引或者in常数列表，子查询可能返回重复值，可以使用索引将子查询去重。

range（重要）

索引范围扫描，常见于使用>,<,is null,between ,in ,like等运算符的查询中。

```
mysql> explain select * from tuser where id>1;
+----+-------------+-------+-------+---------------+---------+---------+------+-
-----+-------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  |
rows | Extra       |
+----+-------------+-------+-------+---------------+---------+---------+------+-
-----+-------------+
|  1 | SIMPLE      | tuser | range | PRIMARY       | PRIMARY | 4       | NULL |
   1 | Using where |
+----+-------------+-------+-------+---------------+---------+---------+------+-
-----+-------------+

--like 前缀索引
mysql> explain select * from tuser where name like 'a%';
```

```
+----+-------------+-------+-------+-----------------+-----------------+------
---+------+------+-----------------------+
| id | select_type | table | type  | possible_keys   | key             |
key_len | ref  | rows | Extra                 |
+----+-------------+-------+-------+-----------------+-----------------+------
---+------+------+-----------------------+
|  1 | SIMPLE      | tuser | range | idx_name_age_sex | idx_name_age_sex | 303
  | NULL |    1 | Using index condition |
+----+-------------+-------+-------+-----------------+-----------------+------
---+------+------+-----------------------+
```

注： like '%a' 不使用索引

```
mysql> explain select * from tuser where loginname like  'a%';
+----+-------------+-------+-------+---------------+--------------+---------+--
----+------+-----------------------+
| id | select_type | table | type  | possible_keys | key          | key_len |
ref  | rows | Extra                 |
+----+-------------+-------+-------+---------------+--------------+---------+--
----+------+-----------------------+
|  1 | SIMPLE      | tuser | range | idx_loginname | idx_loginname | 303     |
NULL |    1 | Using index condition |
+----+-------------+-------+-------+---------------+--------------+---------+--
----+------+-----------------------+
1 row in set (0.00 sec)
```

index_merge

表示查询使用了两个以上的索引，最后取交集或者并集，常见and，or的条件使用了不同的索引，官方排序这个在ref_or_null之后，但是实际上由于要读取所个索引，性能可能大部分时间都不如range

index（重要）

关键字：条件是出现在索引树中的节点的。可能没有完全匹配索引。

索引全表扫描，把索引从头到尾扫一遍，常见于使用索引列就可以处理不需要读取数据文件的查询、可以使用索引排序或者分组的查询。

```
--单索引
mysql> explain select loginname from tuser;
+----+-------------+-------+-------+---------------+--------------+---------+--
----+------+-------------+
| id | select_type | table | type  | possible_keys | key          | key_len |
ref  | rows | Extra       |
+----+-------------+-------+-------+---------------+--------------+---------+--
----+------+-------------+
|  1 | SIMPLE      | tuser | index | NULL          | idx_loginname | 303     |
NULL |    2 | Using index |
+----+-------------+-------+-------+---------------+--------------+---------+--
----+------+-------------+
--组合索引
mysql> explain select age from tuser;
+----+-------------+-------+-------+---------------+-----------------+---------
+------+------+-------------+
| id | select_type | table | type  | possible_keys | key             | key_len
| ref  | rows | Extra       |
```

```
+----+-------------+-------+-------+---------------+-----------------+---------
+------+------+-------------+
|  1 | SIMPLE      | tuser | index | NULL          | idx_name_age_sex | 312
| NULL |    2 | Using index |
+----+-------------+-------+-------+---------------+-----------------+---------
+------+------+-------------+

思考：explain select loginname,age from tuser;

覆盖索引
```

all（重要）

这个就是全表扫描数据文件，然后再**在server层进行过滤**返回符合要求的记录。

```
mysql>  explain select * from tuser;
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------+
|  1 | SIMPLE      | tuser | ALL  | NULL          | NULL | NULL    | NULL |    2
| NULL  |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------+

回表查询
思考：如何使用索引？
```

主键不要uuid

why？ 无序、辅助索引要存 太长

主键： 自增

分布式主键： 雪花算法（sharding jdbc）、 redis生成

possible_keys

此次查询中可能选用的索引，一个或多个

key

查询真正使用到的索引，select_type为index_merge时，这里可能出现两个以上的索引，其他的select_type这里只会出现一个。

key_len

- 用于处理查询的索引长度，如果是单列索引，那就整个索引长度算进去，如果是多列索引，那么查询不一定都能使用到所有的列，具体使用到了多少个列的索引，这里就会计算进去，没有使用到的列，这里不会计算进去。
- 留意下这个列的值，算一下你的多列索引总长度就知道有没有使用到所有的列了。

- *另外，key_len只计算where条件用到的索引长度，而排序和分组就算用到了索引，也不会计算到key_len中。*

看组合索引的使用情况

ref

- 如果是使用的常数等值查询，这里会显示const
- 如果是连接查询，被驱动表的执行计划这里会显示驱动表的关联字段
- 如果是条件使用了表达式或者函数，或者条件列发生了内部隐式转换，这里可能显示为func

rows

这里是执行计划中估算的扫描行数，不是精确值（InnoDB不是精确的值，MyISAM是精确的值，主要原因是InnoDB里面使用了MVCC并发机制）

extra（重要）

这个列包含不适合在其他列中显示单十分重要的额外的信息，这个列可以显示的信息非常多，有几十种，常用的有

no tables used

不带from字句的查询或者From dual查询

使用not in()形式子查询或not exists运算符的连接查询，这种叫做反连接

即，一般连接查询是先查询内表，再查询外表，反连接就是先查询外表，再查询内表。

using filesort（重要）

- 排序时无法使用到索引时，就会出现这个。常见于order by和group by语句中
- 说明MySQL会使用一个外部的索引排序，而不是按照索引顺序进行读取。
- MySQL中无法利用索引完成的排序操作称为“文件排序”

```
mysql> explain select * from tuser order by address;
+----+-------------+-------+------+---------------+------+---------+------+-----
-+----------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra          |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+----------------+
|  1 | SIMPLE      | tuser | ALL  | NULL          | NULL | NULL    | NULL |    2
| Using filesort |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+----------------+
```

using index（重要）

查询时**不需要回表查询**，直接通过索引就可以获取查询的数据。

- 表示相应的SELECT查询中使用到了**覆盖索引（Covering Index）**，避免访问表的数据行，效率不错！

- 如果同时出现Using Where， 说明索引被用来执行查找索引键值
- 如果没有同时出现Using Where， 表明索引用来读取数据而非执行查找动作。

```
mysql>  explain select name,age,sex from tuser ;
+----+-------------+-------+-------+---------------+-----------------+---------
+------+------+-------------+
| id | select_type | table | type  | possible_keys | key             | key_len
| ref  | rows | Extra       |
+----+-------------+-------+-------+---------------+-----------------+---------
+------+------+-------------+
|  1 | SIMPLE      | tuser | index | NULL          | idx_name_age_sex | 312
| NULL |    2 | Using index |
+----+-------------+-------+-------+---------------+-----------------+---------
+------+------+-------------+
全值匹配 覆盖索引
```

using temporary

- 表示使用了临时表存储中间结果。
- MySQL在对查询结果**order by和group by**时使用临时表
- 临时表可以是内存临时表和磁盘临时表，执行计划中看不出来，需要查看status变量，used_tmp_table，used_tmp_disk_table才能看出来。
- **distinct**

  在select部分使用了distinct关键字 （索引字段）

- ```
  mysql> explain select distinct a.id from tuser a,tdep b where a.dep=b.id;

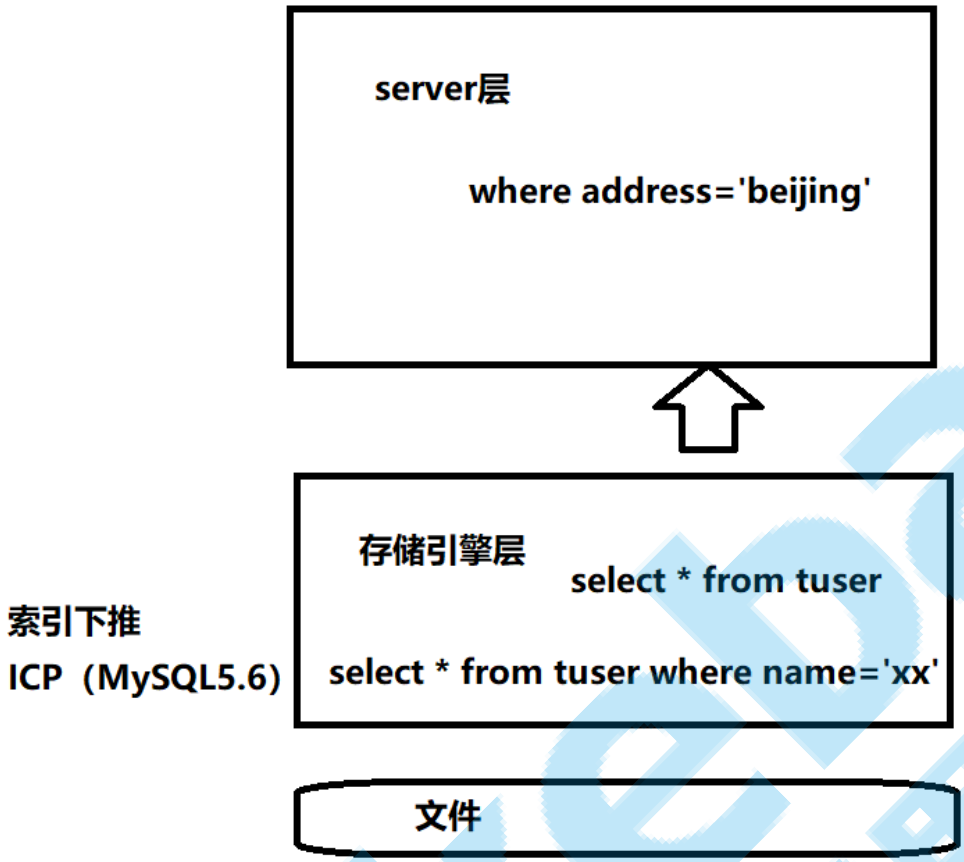
  +----+-------------+-------+--------+-----------------------------------
  ---------+---------+---------+------------+------+-------------------------
  ----------------+
  | id | select_type | table | type   | possible_keys
          | key     | key_len | ref        | rows | Extra
                  |
  +----+-------------+-------+--------+-----------------------------------
  ---------+---------+---------+------------+------+-------------------------
  ----------------+
  |  1 | SIMPLE      | a     | index  |
  PRIMARY,idx_loginname,idx_name_age_sex,idx_dep | idx_dep | 5       | NULL
      |    2 | Using where; Using index; Using temporary |
  |  1 | SIMPLE      | b     | eq_ref | PRIMARY
          | PRIMARY | 4       | kkb2.a.dep |    1 | Using index; Distinct
                  |
  +----+-------------+-------+--------+-----------------------------------
  ---------+---------+---------+------------+------+-------------------------
  ----------------+
  ```

using where（重要）
  ```

- 表示存储引擎返回的记录并不是所有的都满足查询条件，需要在server层进行过滤。

```
--查询条件无索引
mysql> explain select * from tuser where address='beijing';
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
|  1 | SIMPLE      | tuser | ALL  | NULL          | NULL | NULL    | NULL |    2
| Using where |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
--索引失效
mysql>  explain select * from tuser where age=1;
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
|  1 | SIMPLE      | tuser | ALL  | NULL          | NULL | NULL    | NULL |    2
| Using where |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
mysql> explain select * from tuser where  id in(1,2);
+----+-------------+-------+-------+---------------+---------+---------+------+-
-----+-------------+
| id | select_type | table | type  | possible_keys | key     | key_len | ref  |
rows | Extra       |
+----+-------------+-------+-------+---------------+---------+---------+------+-
-----+-------------+
|  1 | SIMPLE      | tuser | range | PRIMARY       | PRIMARY | 4       | NULL |
   2 | Using where |
+----+-------------+-------+-------+---------------+---------+---------+------+-
-----+-------------+
```

- 查询条件中分为限制条件和检查条件，5.6之前，存储引擎只能根据限制条件扫描数据并返回，然后server层根据检查条件进行过滤再返回真正符合查询的数据。5.6.x之后支持**ICP**特性，可以把检查条件也下推到存储引擎层，不符合检查条件和限制条件的数据，直接不读取，这样就大大减少了存储引擎扫描的记录数量。extra列显示**using index condition**




```
mysql> explain select * from tuser where name='asd';
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------+------+-----------------------+
| id | select_type | table | type | possible_keys   | key             |
key_len | ref   | rows | Extra                 |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------+------+-----------------------+
|  1 | SIMPLE      | tuser | ref  | idx_name_age_sex | idx_name_age_sex | 303
   | const |    1 | Using index condition |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------+------+-----------------------+
```

索引失效分析

1. 全值匹配我最爱
2. 最佳左前缀法则
3. 不在索引列上做任何操作（计算、函数、(自动or手动)类型转换），会导致索引失效而转向全表扫描
4. 存储引擎不能使用索引中范围条件右边的列
5. 尽量使用覆盖索引(只访问索引的查询(索引列和查询列一致))，减少select *
6. mysql 在使用不等于(!= 或者<>)的时候无法使用索引会导致全表扫描
7. is null ,is not null 也无法使用索引
8. like以通配符开头('%abc...')mysql索引失效会变成全表扫描的操作
9. 字符串不加单引号索引失效
10. 少用or,用它来连接时会索引失效

1.全值匹配我最爱

```
mysql> explain select * from tuser where name='zhaoyun' and age=1 and sex='1';
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------------------+------+-----------------------+
| id | select_type | table | type | possible_keys    | key             |
key_len | ref               | rows | Extra                 |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------------------+------+-----------------------+
|  1 | SIMPLE      | tuser | ref  | idx_name_age_sex | idx_name_age_sex | 312
  | const,const,const |    1 | Using index condition |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------------------+------+-----------------------+
```

条件与索引一一对应

2.最佳左前缀法则

组合索引

带头索引不能死，中间索引不能断

如果索引了多个列，要遵守最佳左前缀法则。指的是查询从索引的最左前列开始 并且不跳过索引中的列。 正确的示例参考上图。

错误的示例：

带头索引死：

```
mysql> explain select * from tuser where age=23;
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
|  1 | SIMPLE      | tuser | ALL  | NULL          | NULL | NULL    | NULL |    2
| Using where |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
```

中间索引断（带头索引生效，其他索引失效）：

```
mysql> explain select * from tuser where name='aa' and sex='1';
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------+------+-----------------------+
| id | select_type | table | type | possible_keys    | key             |
key_len | ref   | rows | Extra                 |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------+------+-----------------------+
|  1 | SIMPLE      | tuser | ref  | idx_name_age_sex | idx_name_age_sex | 303
  | const |    1 | Using index condition |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------+------+-----------------------+
```

比较

```
mysql>  explain select * from tuser where name='aa' and sex='1' and age=23;

+----+-------------+-------+------+-----------------+-----------------+-------
--+-------------------+------+-----------------------+
| id | select_type | table | type | possible_keys    | key             |
key_len | ref               | rows | Extra                 |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------------------+------+-----------------------+
|  1 | SIMPLE      | tuser | ref  | idx_name_age_sex | idx_name_age_sex | 312
  | const,const,const |    1 | Using index condition |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------------------+------+-----------------------

比较
mysql> explain select * from tuser where name='aa' and sex=1 and age=23;
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------------+------+-----------------------+
| id | select_type | table | type | possible_keys    | key             |
key_len | ref         | rows | Extra                 |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------------+------+-----------------------+
|  1 | SIMPLE      | tuser | ref  | idx_name_age_sex | idx_name_age_sex | 308
  | const,const |    1 | Using index condition |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------------+------+-----------------------+
```

3.不要在索引上做计算

```
不要进行这些操作：计算、函数、自动/手动类型转换，不然会导致索引失效而转向全表扫描

mysql> explain select * from tuser where loginname='zy';
+----+-------------+-------+-------+---------------+---------------+---------+--
-----+------+-------+
| id | select_type | table | type  | possible_keys | key           | key_len |
ref   | rows | Extra |
+----+-------------+-------+-------+---------------+---------------+---------+--
-----+------+-------+
|  1 | SIMPLE      | tuser | const | idx_loginname | idx_loginname | 303     |
const |    1 | NULL  |
+----+-------------+-------+-------+---------------+---------------+---------+--
-----+------+-------+
1 row in set (0.00 sec)

mysql> explain select * from tuser where left(loginname,1)='zy';
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
|  1 | SIMPLE      | tuser | ALL  | NULL          | NULL | NULL    | NULL |    2
| Using where |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
```

4.范围条件右边的列失效

```
不能继续使用索引中范围条件（bettween、<、>、in等）右边的列
mysql> explain select * from tuser where name='asd' and age>20 and sex='1';
+----+-------------+-------+-------+-----------------+-----------------+------
---+------+------+-----------------------+
| id | select_type | table | type  | possible_keys   | key             |
key_len | ref  | rows | Extra                 |
+----+-------------+-------+-------+-----------------+-----------------+------
---+------+------+-----------------------+
|  1 | SIMPLE      | tuser | range | idx_name_age_sex | idx_name_age_sex | 308
  | NULL |    1 | Using index condition |
+----+-------------+-------+-------+-----------------+-----------------+------
---+------+------+-----------------------+
```

5.尽量使用覆盖索引

```
尽量使用覆盖索引（只查询索引的列），也就是索引列和查询列一致，减少select *

mysql> explain select *  from tuser ;
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------+
|  1 | SIMPLE      | tuser | ALL  | NULL          | NULL | NULL    | NULL |    2
| NULL  |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------+
1 row in set (0.00 sec)

mysql> explain select name,loginname  from tuser ;
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------+
|  1 | SIMPLE      | tuser | ALL  | NULL          | NULL | NULL    | NULL |    2
| NULL  |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------+
1 row in set (0.01 sec)

mysql> explain select name,age,sex  from tuser ;
+----+-------------+-------+-------+---------------+-----------------+---------
+------+------+-------------+
| id | select_type | table | type  | possible_keys | key             | key_len
| ref  | rows | Extra       |
```

```
+----+-------------+-------+-------+---------------+-----------------+---------
+------+------+-------------+
|  1 | SIMPLE      | tuser | index | NULL          | idx_name_age_sex | 312
| NULL |    2 | Using index |
+----+-------------+-------+-------+---------------+-----------------+---------
+------+------+-------------+
1 row in set (0.00 sec)

mysql> explain select loginname  from tuser ;
+----+-------------+-------+-------+---------------+---------------+---------+--
----+------+-------------+
| id | select_type | table | type  | possible_keys | key           | key_len |
ref  | rows | Extra       |
+----+-------------+-------+-------+---------------+---------------+---------+--
----+------+-------------+
|  1 | SIMPLE      | tuser | index | NULL          | idx_loginname | 303     |
NULL |    2 | Using index |
+----+-------------+-------+-------+---------------+---------------+---------+--
----+------+-------------+
1 row in set (0.00 sec)
```

6.索引字段上不要使用不等

```
索引字段上使用（！= 或者 < >）判断时，会导致索引失效而转向全表扫描
注：主键索引会使用范围索引，辅助索引会失效
mysql> explain select * from tuser where loginname='zhy';
+----+-------------+-------+-------+---------------+---------------+---------+--
-----+------+-------+
| id | select_type | table | type  | possible_keys | key           | key_len |
ref   | rows | Extra |
+----+-------------+-------+-------+---------------+---------------+---------+--
-----+------+-------+
|  1 | SIMPLE      | tuser | const | idx_loginname | idx_loginname | 303     |
const |    1 | NULL  |
+----+-------------+-------+-------+---------------+---------------+---------+--
-----+------+-------+
1 row in set (0.00 sec)

mysql> explain select * from tuser where loginname!='zhy';
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
|  1 | SIMPLE      | tuser | ALL  | idx_loginname | NULL | NULL    | NULL |    1
| Using where |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
```

7.主键索引字段上不可以判断null

主键字段上不可以使用 null
索引字段上使用 is null 判断时，可使用索引

```
mysql>  explain select * from tuser where name is  null;
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------+------+-----------------------+
| id | select_type | table | type | possible_keys    | key             |
key_len | ref   | rows | Extra                 |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------+------+-----------------------+
|  1 | SIMPLE      | tuser | ref  | idx_name_age_sex | idx_name_age_sex | 303
  | const |    1 | Using index condition |
+----+-------------+-------+------+-----------------+-----------------+-------
--+-------+------+-----------------------+
1 row in set (0.00 sec)


mysql>  explain select * from tuser where loginname is  null;
+----+-------------+-------+------+--------------+--------------+---------+---
----+------+-----------------------+
| id | select_type | table | type | possible_keys | key          | key_len |
ref   | rows | Extra                 |
+----+-------------+-------+------+--------------+--------------+---------+---
----+------+-----------------------+
|  1 | SIMPLE      | tuser | ref  | idx_loginname | idx_loginname | 303     |
const |    1 | Using index condition |
+----+-------------+-------+------+--------------+--------------+---------+---
----+------+-----------------------+
1 row in set (0.00 sec)


1 row in set (0.00 sec)


主键非空  不使用索引
mysql> explain select * from tuser where id is not null;
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
|  1 | SIMPLE      | tuser | ALL  | PRIMARY       | NULL | NULL    | NULL |    2
| Using where |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
```

8.索引字段使用like不以通配符开头

索引字段使用like以通配符开头（‘%字符串’）时，会导致索引失效而转向全表扫描

```
mysql> explain select * from tuser  where  name like 'a%';
+----+-------------+-------+-------+-----------------+-----------------+------
---+------+------+-----------------------+
| id | select_type | table | type  | possible_keys    | key             |
key_len | ref   | rows | Extra                 |
```

```
+----+-------------+-------+-------+-----------------+-----------------+------
---+------+------+-----------------------+
|  1 | SIMPLE      | tuser | range | idx_name_age_sex | idx_name_age_sex | 303
  | NULL |    1 | Using index condition |
+----+-------------+-------+-------+-----------------+-----------------+------
---+------+------+-----------------------+

mysql>  explain select * from tuser  where  name like '%a';
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
| id | select_type | table | type | possible_keys | key  | key_len | ref  | rows
| Extra       |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
|  1 | SIMPLE      | tuser | ALL  | NULL          | NULL | NULL    | NULL |    2
| Using where |
+----+-------------+-------+------+---------------+------+---------+------+-----
-+-------------+
```

由结果可知，like以通配符结束相当于范围查找，索引不会失效。与范围条件（bettween、<、>、in等）不同的是：不会导致右边的索引失效。

问题：解决like '%字符串%'时，索引失效问题的方法？ 使用覆盖索引可以解决。

```
mysql> explain select name , age,sex from tuser where name like '%a%';
+----+-------------+-------+-------+---------------+-----------------+---------
+---------+------+--------------------------+
| id | select_type | table | type  | possible_keys | key             | key_len
| ref  | rows | Extra                    |
+----+-------------+-------+-------+---------------+-----------------+---------
+---------+------+--------------------------+
|  1 | SIMPLE      | tuser | index | NULL          | idx_name_age_sex | 312
| NULL |    2 | Using where; Using index |
+----+-------------+-------+-------+---------------+-----------------+---------
+---------+------+--------------------------+
```

9.索引字段字符串要加单引号

```
索引字段是字符串，但查询时不加单引号，会导致索引失效而转向全表扫描
mysql> explain select * from tuser where name=123;
+----+-------------+-------+------+-----------------+------+---------+------+--
----+-------------+
| id | select_type | table | type | possible_keys    | key  | key_len | ref  |
rows | Extra       |
+----+-------------+-------+------+-----------------+------+---------+------+--
----+-------------+
|  1 | SIMPLE      | tuser | ALL  | idx_name_age_sex | NULL | NULL    | NULL |
 2 | Using where |
+----+-------------+-------+------+-----------------+------+---------+------+--
----+-------------+
```

10.索引字段不要使用or

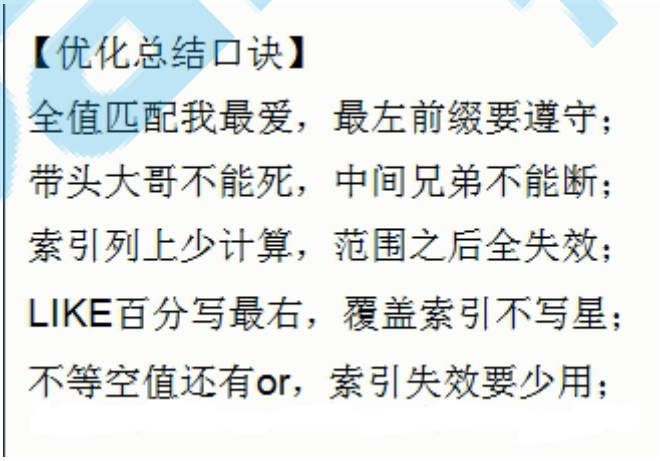
索引字段使用 or 时，会导致索引失效而转向全表扫描

```
mysql> explain select * from tuser where name='asd' or age=23;
+----+-------------+-------+------+-----------------+------+---------+------+------+-------------+
| id | select_type | table | type | possible_keys   | key  | key_len | ref  | rows | Extra       |
+----+-------------+-------+------+-----------------+------+---------+------+------+-------------+
|  1 | SIMPLE      | tuser | ALL  | idx_name_age_sex | NULL | NULL    | NULL |    2 | Using where |
+----+-------------+-------+------+-----------------+------+---------+------+------+-------------+
```

总结

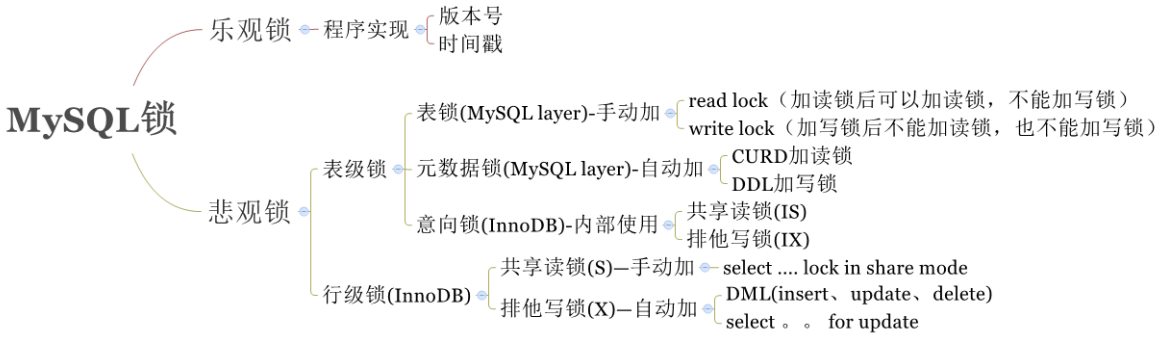
假设index(a,b,c)

Where语句	索引是否被使用
where a = 3	Y,使用到a
where a = 3 and b = 5	Y,使用到a，b
where a = 3 and b = 5 and c = 4	Y,使用到a,b,c
where b = 3 或者 where b = 3 and c = 4 或者 where c = 4	N
where a = 3 and c = 5	使用到a， 但是c不可以，b中间断了
where a = 3 and b > 4 and c = 5	使用到a和b， c不能用在范围之后，b断了
where a = 3 and b like 'kk%' and c = 4	Y,使用到a,b,c
where a = 3 and b like '%kk' and c = 4	Y,只用到a
where a = 3 and b like '%kk%' and c = 4	Y,只用到a
where a = 3 and b like 'k%kk%' and c = 4	Y,使用到a,b,c



三、MySQL锁篇

MySQL锁介绍



MySQL表级锁

表级锁介绍

由MySQL SQL layer层实现

- MySQL的表级锁有两种：

```
一种是表锁。
一种是元数据锁（meta data lock，MDL)。
```

- MySQL 实现的表级锁定的争用状态变量：

```
mysql> show status like 'table%';
```

```
mysql> show status like 'table%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Table_locks_immediate      | 113   |
| Table_locks_waited         | 0     |
| Table_open_cache_hits      | 5     |
| Table_open_cache_misses    | 1     |
| Table_open_cache_overflows | 0     |
+----------------------------+-------+
```

```
- table_locks_immediate：产生表级锁定的次数；
- table_locks_waited：出现表级锁定争用而发生等待的次数；
```

表锁介绍

- 表锁有两种表现形式：

```
表共享读锁（Table Read Lock）
表独占写锁（Table Write Lock）
```

- 手动增加表锁

```
lock table 表名称 read(write),表名称2 read(write), 其他;
```

- 查看表锁情况

```
show open tables;
```

- 删除表锁

```
unlock tables;
```

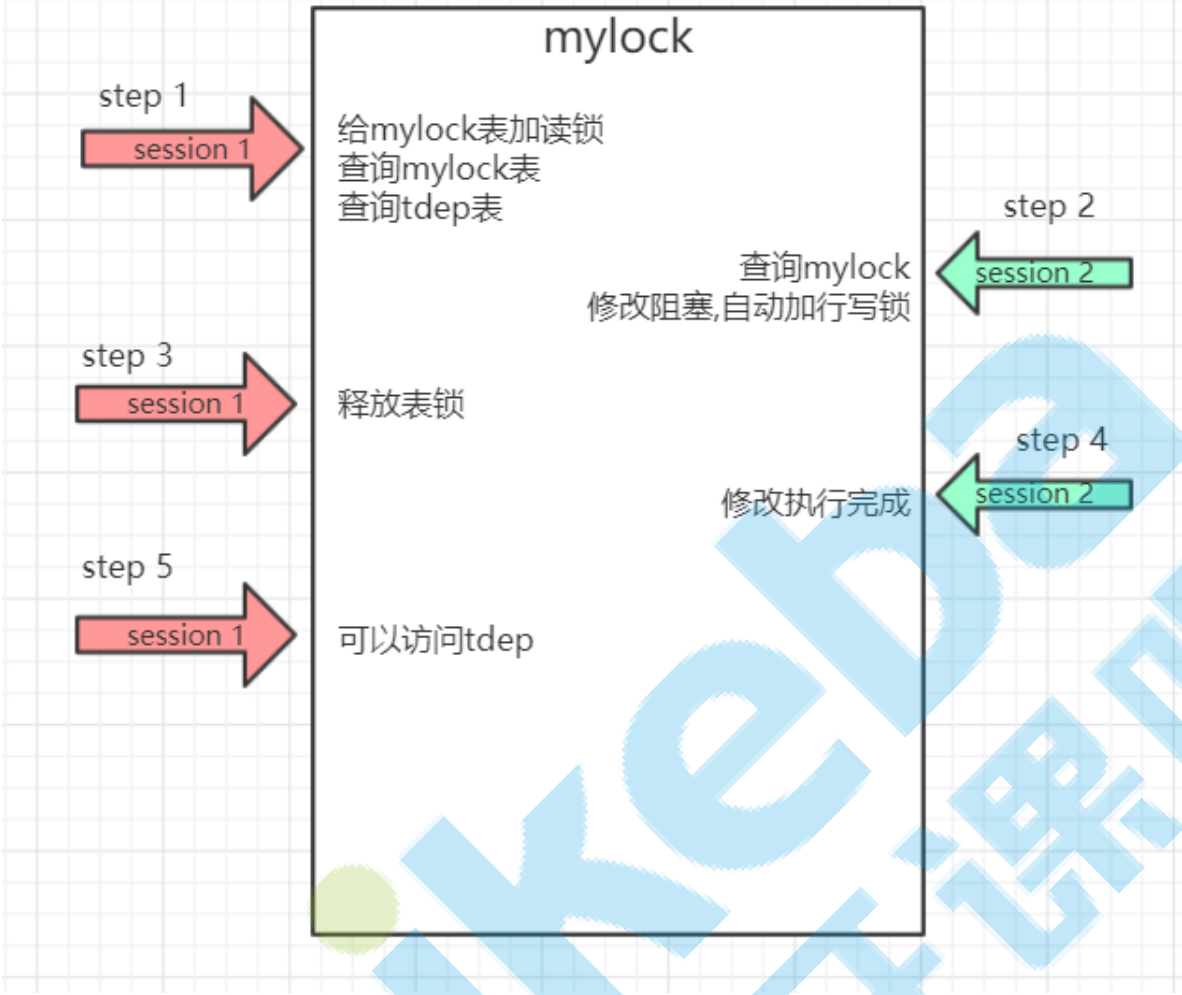
表锁演示

环境准备

```
--新建表
CREATE TABLE mylock (
id int(11) NOT NULL AUTO_INCREMENT,
NAME varchar(20) DEFAULT NULL,
PRIMARY KEY (id)
);
INSERT INTO mylock (id,NAME) VALUES (1, 'a');
INSERT INTO mylock (id,NAME) VALUES (2, 'b');
INSERT INTO mylock (id,NAME) VALUES (3, 'c');
INSERT INTO mylock (id,NAME) VALUES (4, 'd');
```

读锁演示

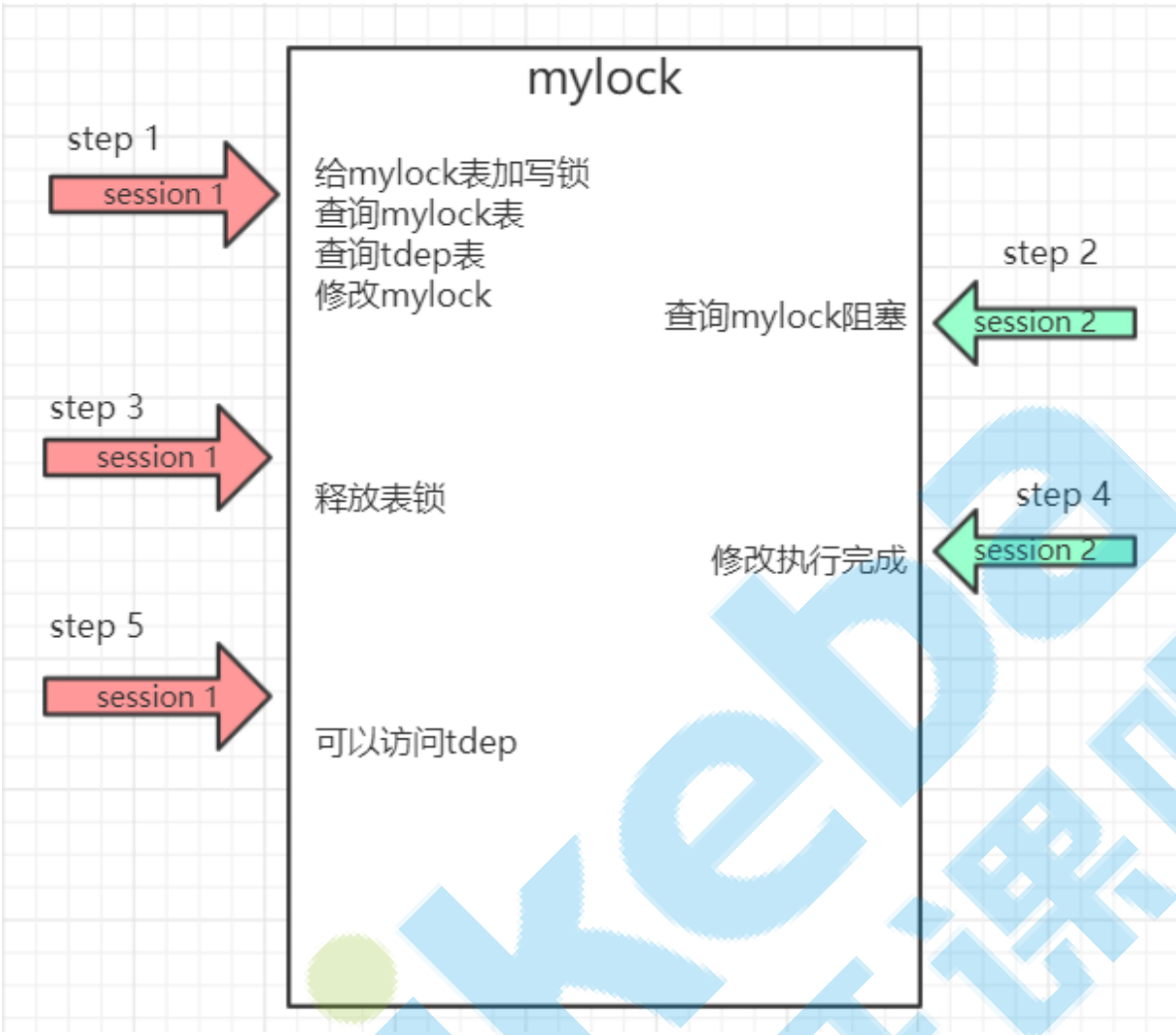
1、表读锁



session1（Navicat）、session2（mysql）

```
1、session1: lock table mylock read; -- 给mylock表加读锁
2、session1: select * from mylock; -- 可以查询
3、session1: select * from tdep; --不能访问非锁定表
4、session2: select * from mylock; -- 可以查询 没有锁
5、session2: update mylock set name='x' where id=2; -- 修改阻塞,自动加行写锁
6、session1: unlock tables; -- 释放表锁
7、session2: Rows matched: 1  Changed: 1  Warnings: 0 -- 修改执行完成
8、session1: select * from tdep; --可以访问
```

2、表写锁



session1（Navicat）、session2（mysql）

```
1、session1: lock table mylock write; -- 给mylock表加写锁
2、session1: select * from mylock; -- 可以查询
3、session1: select * from tdep; --不能访问非锁定表
4、session1: update mylock set name='y' where id=2; --可以执行
5、session2: select * from mylock; -- 查询阻塞
6、session1: unlock tables; -- 释放表锁
7、session2: 4 rows in set (22.57 sec) -- 查询执行完成
8、session1: select * from tdep; --可以访问
```