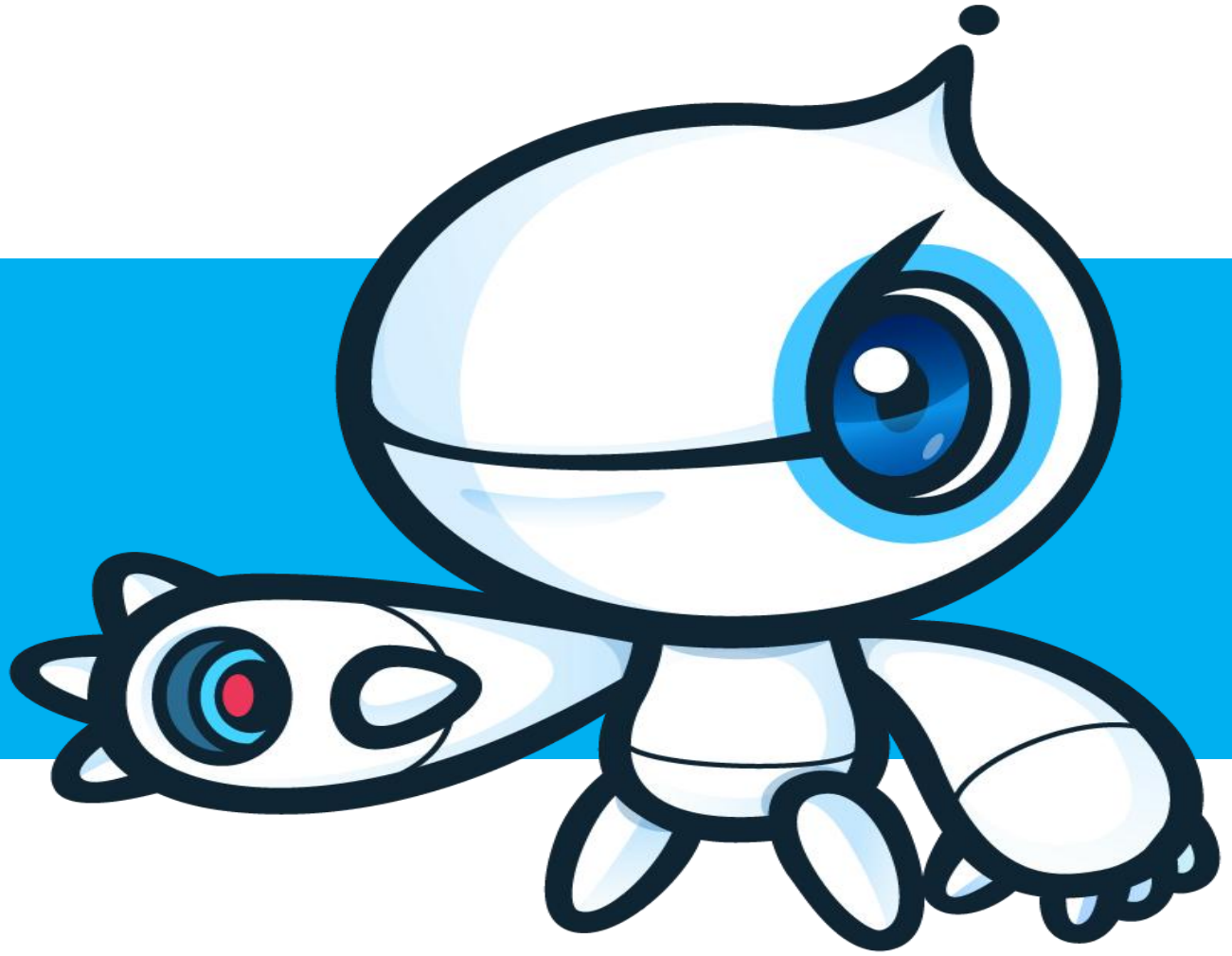
车轮互联
WWW.CHELUN.COM

mysql优化之道

--Mysql为何放弃索引



武林志 2019.1.12

内容

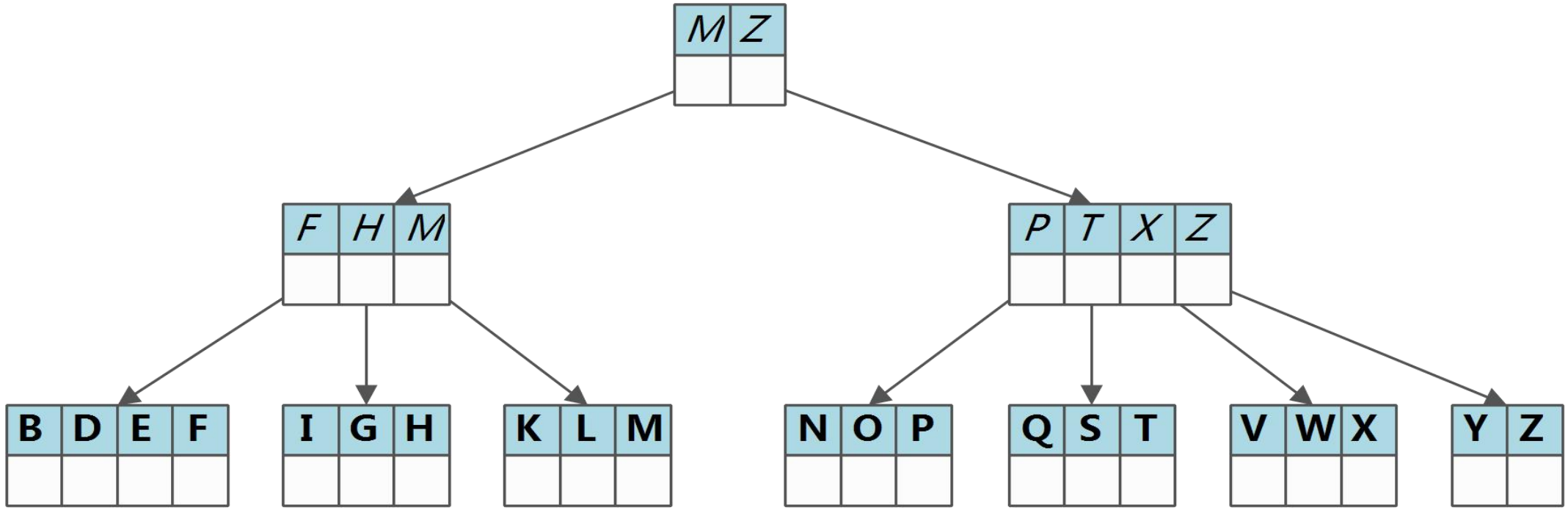
1 概述

2 案例分析

3 课后问题

只讲原理，不背理论！

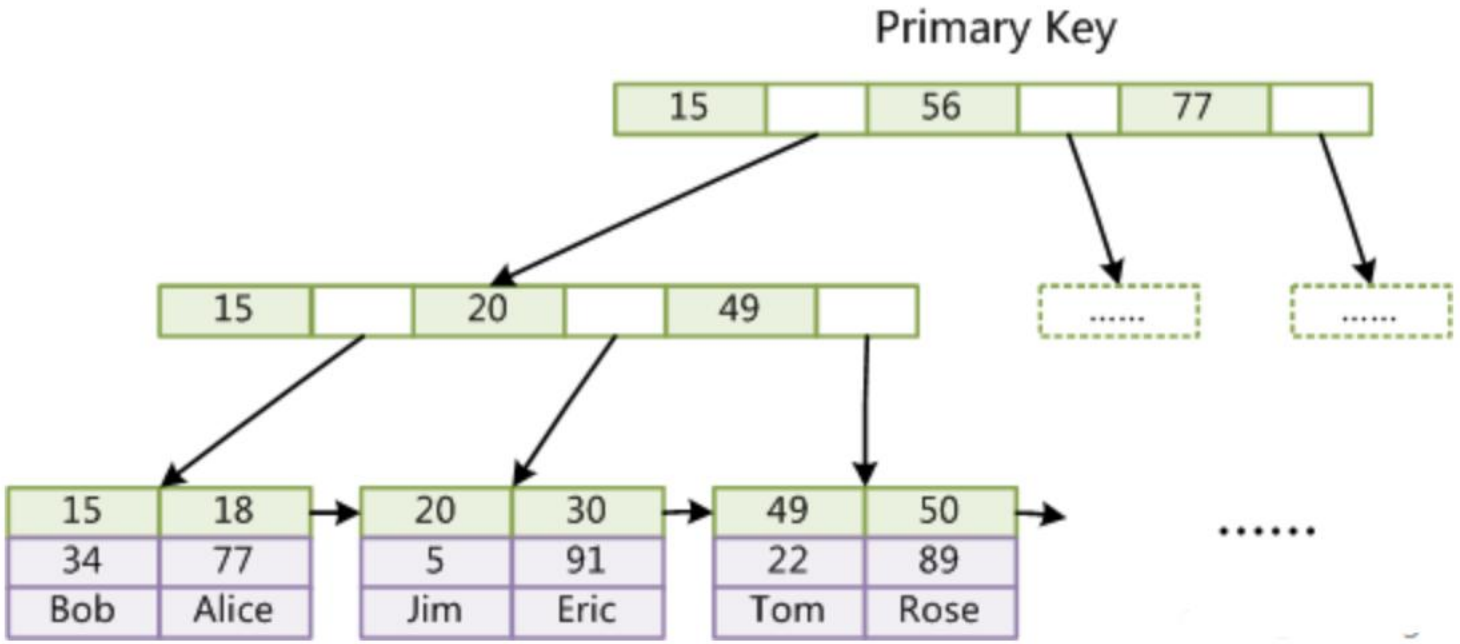
B+树



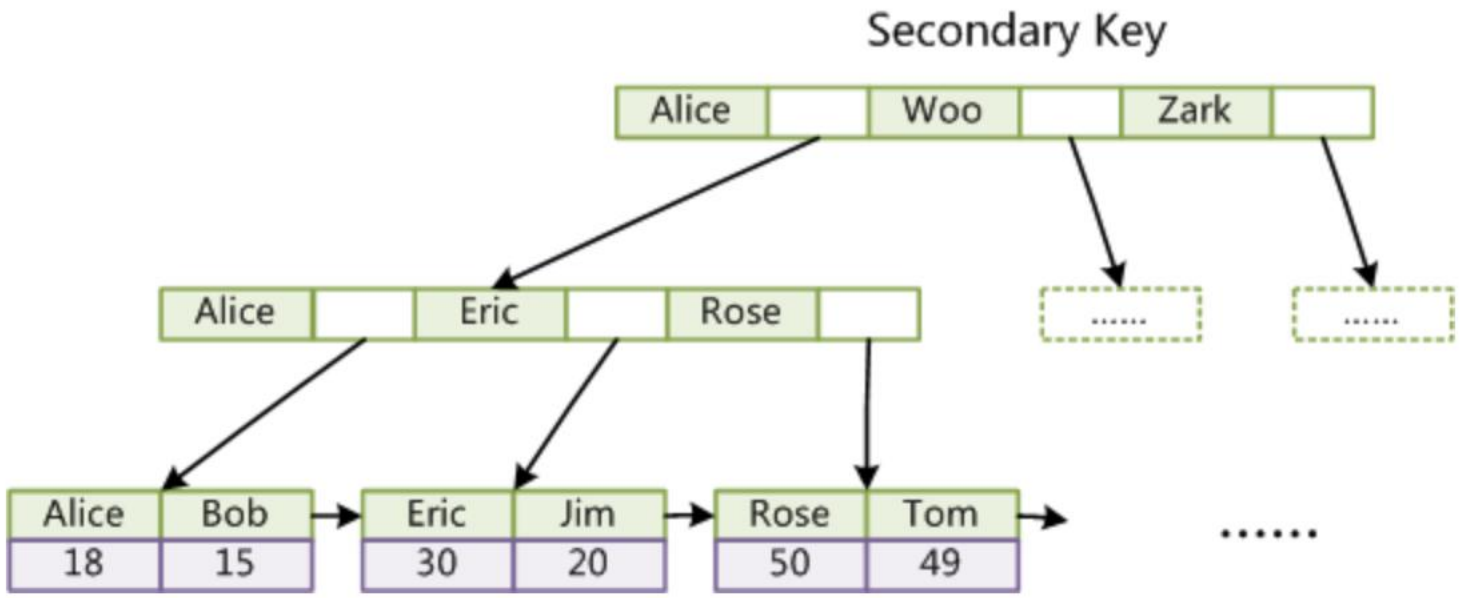
m阶B+树特性：

- 每个节点子树最多为m
- 每个非根节点子树最少为m/2
- 根节点子树子树最少为2（除非只有一个节点）
- 非叶子节点关键字和子树数目相同
- 所有叶节点在同一层上
- 所有数据存放于叶子节点上，并有序链接（链表）
- 节点内关键字有序排列：K[i-1] < K[i]
- P[i] ≤ K[i] < P[i+1]，K为关键字，P为子树关键字集合

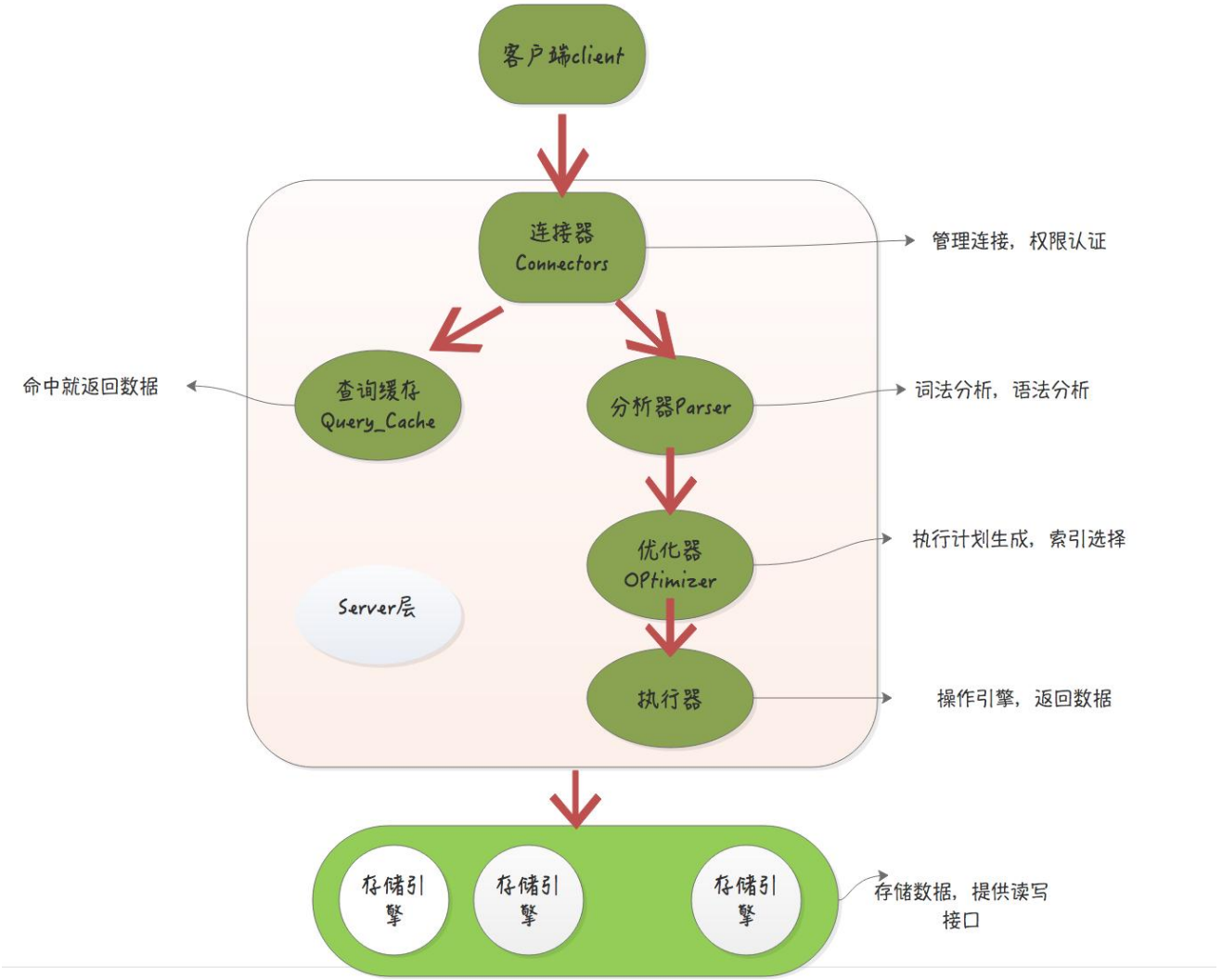
Innodb 主键索引结构

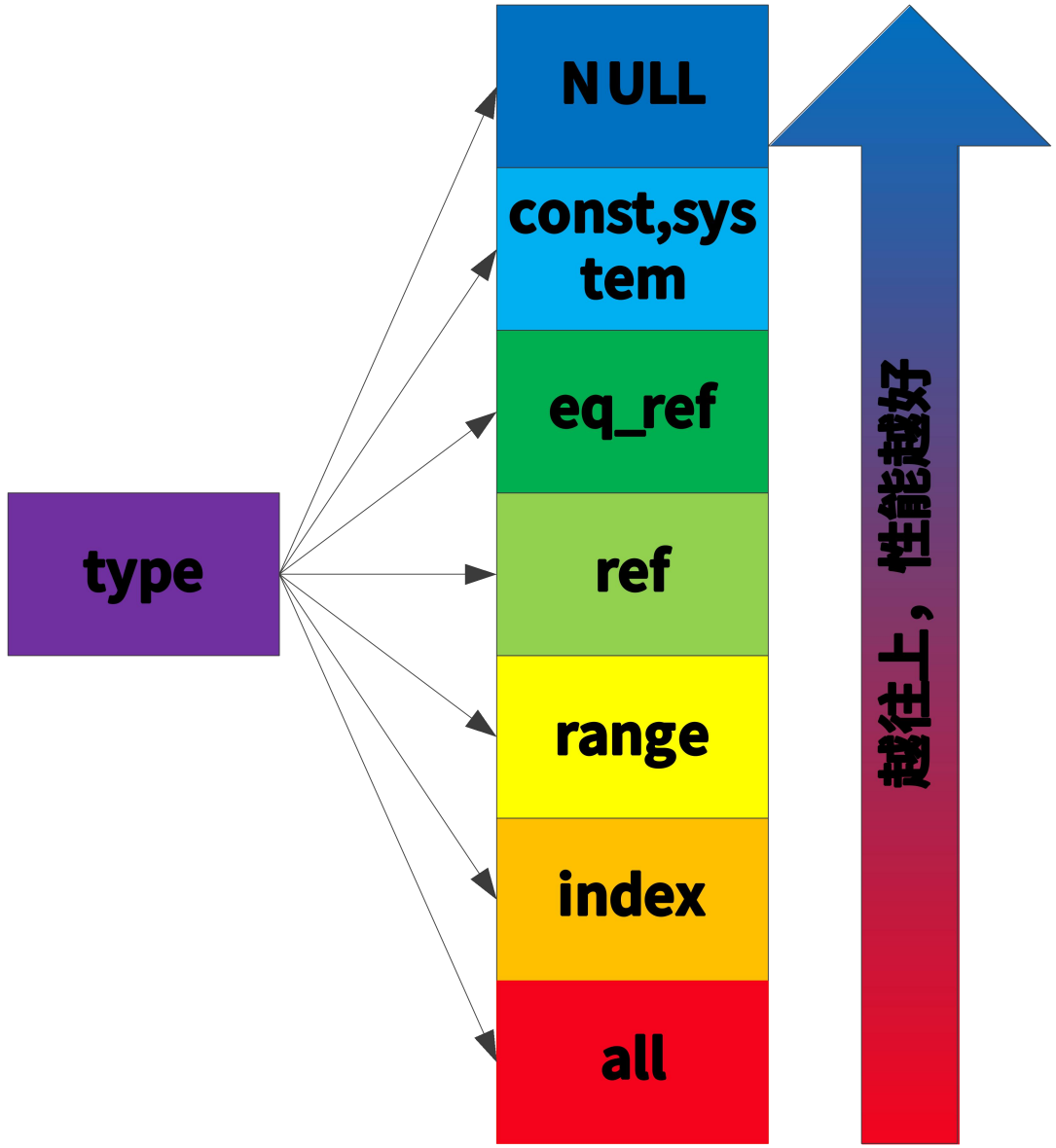


Innodb辅助索引结构



MySQL 的基本架构示意图




type
NULL
const,system
eq_ref
ref
range
index
all
越往上，性能越好

内容

1 概述

2 案例分析

3 课后问题

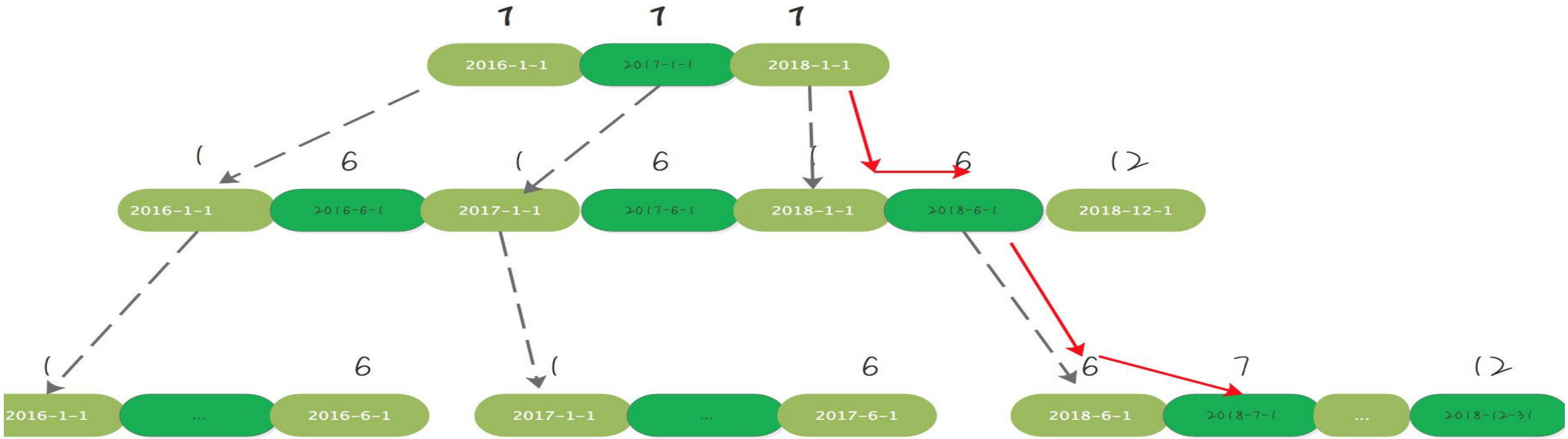
案例一：条件字段函数操作

```
CREATE TABLE `tradelog` (
  `id` int(11) NOT NULL,
  `tradeid` varchar(32) DEFAULT NULL,
  `operator` int(11) DEFAULT NULL,
  `t_modified` datetime DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `idx_tradeid` (`tradeid`),
  KEY `idx_modified` (`t_modified`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

其中交易记录表 tradelog 包含交易流水号（tradeid）、交易员 id（operator）、交易时间（t_modified）等字段

统计发生在所有年份中 7 月份的交易记录总数：

```
mysql> select count(*) from tradelog where month(t_modified)=7;
```



对索引字段做函数操作，可能会破坏索引值的有序性，因此优化器就决定放弃走树搜索功能。

```
mysql> explain select count(*) from tradelog where month(t_modified)=7;
+----+-------------+----------+------------+-------+---------------+-----------+---------+------+--------+----------+--------------------------+
| id | select_type | table    | partitions | type  | possible_keys | key       | key_len | ref  | rows   | filtered | Extra                    |
+----+-------------+----------+------------+-------+---------------+-----------+---------+------+--------+----------+--------------------------+
|  1 | SIMPLE      | tradelog | NULL       | index | NULL          | t_modified | 6       | NULL | 100335 |   100.00 | Using where; Using index |
+----+-------------+----------+------------+-------+---------------+-----------+---------+------+--------+----------+--------------------------+
```

需要注意的是，优化器并不是要放弃使用这个索引。

在这个例子里，放弃了树搜索功能，优化器可以选择遍历主键索引，也可以选择遍历索引 t_modified，优化器对比索引大小后发现，索引 t_modified 更小，遍历这个索引比遍历主键索引来得更快。因此最终还是会选择索引 t_modified。

```
mysql> select count(*) from tradelog where
    -> (t_modified >= '2016-7-1' and t_modified<'2016-8-1') or
    -> (t_modified >= '2017-7-1' and t_modified<'2017-8-1') or
    -> (t_modified >= '2018-7-1' and t_modified<'2018-8-1');
```

思考：

```
select * from tradelog where id + 1 = 10000
```

使用了不改变有序性的函数，会不会使用索引快速定位到9999这一行？

案例二：隐式类型转换

```
mysql> select * from tradelog where tradeid=110717;
```

```
root@localhost:mysql_10120.sock [cwz_client]>explain select * from qcp_trade_order where telephone=12345678909;
+----+-------------+----------------+------------+------+----------------+------+---------+------+--------+----------+-------------+
| id | select_type | table          | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra       |
+----+-------------+----------------+------------+------+----------------+------+---------+------+--------+----------+-------------+
|  1 | SIMPLE      | qcp_trade_order | NULL       | ALL  | ind_telephone | NULL | NULL    | NULL | 136269 |    10.00 | Using where |
+----+-------------+----------------+------------+------+----------------+------+---------+------+--------+----------+-------------+
1 row in set, 3 warnings (0.01 sec)

root@localhost:mysql_10120.sock [cwz_client]>root@localhost:mysql_10120.sock [cwz_client]>explain select * from qcp_trade_order where telephone='12345678909';
+----+-------------+----------------+------------+------+----------------+----------------+---------+-------+------+----------+-------+
| id | select_type | table          | partitions | type | possible_keys | key            | key_len | ref   | rows | filtered | Extra |
+----+-------------+----------------+------------+------+----------------+----------------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | qcp_trade_order | NULL       | ref  | ind_telephone | ind_telephone | 152     | const |    1 |   100.00 | NULL  |
+----+-------------+----------------+------------+------+----------------+----------------+---------+-------+------+----------+-------+
1 row in set, 1 warning (0.00 sec)
```

问题？

数据类型转换的规则是什么？

为什么有数据类型转换，就需要走全索引扫描？

https://dev.mysql.com/doc/refman/5.7/en/type-conversion.html

做个实验

```
mysql> select "10" > 9;
+-----------+
| "10" > 9 |
+-----------+
|         1 |
+-----------+
```

从图中可知，select “10” ı 9 返回的是 1，
所以你就能确认 MySQL 里的转换规则了：
在 MySQL 中，字符串和数字做比较的话，
是将字符串转换成数字。

```
mysqlı select * from tradelog where tradeid=110717;
```

```
mysqlı select * from tradelog where  CAST(tradeid as  signed int)=110717;
```

问题：	select * from tradelog where id=‘83126‘;

案例三：隐式字符编码转换

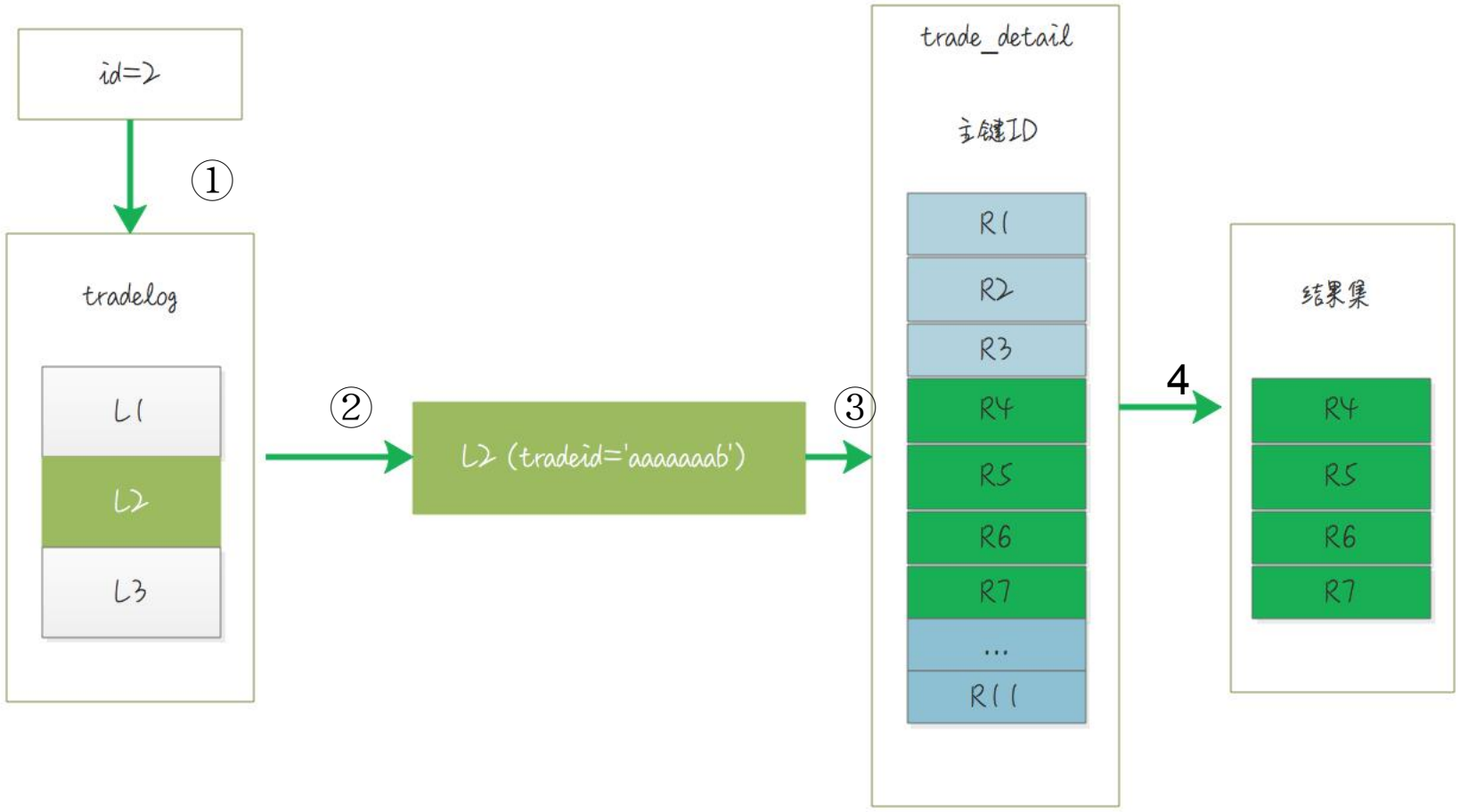
```
CREATE TABLE `trade_detail` (
  `id` int(11) NOT NULL,
  `tradeid` varchar(32) DEFAULT NULL,
  `trade_step` int(11) DEFAULT NULL, /* 操作步骤 */
  `step_info` varchar(32) DEFAULT NULL, /* 步骤信息 */
  PRIMARY KEY (`id`),
  KEY `tradeid` (`tradeid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;

insert into tradelog values(1, 'aaaaaaaa', 1000, now());
insert into tradelog values(2, 'aaaaaaab', 1000, now());
insert into tradelog values(3, 'aaaaaaac', 1000, now());
insert into trade_detail values(1, 'aaaaaaaa', 1, 'add');
insert into trade_detail values(2, 'aaaaaaaa', 2, 'update');
insert into trade_detail values(3, 'aaaaaaaa', 3, 'commit');
insert into trade_detail values(4, 'aaaaaaab', 1, 'add');
insert into trade_detail values(5, 'aaaaaaab', 2, 'update');
insert into trade_detail values(6, 'aaaaaaab', 3, 'update again');
insert into trade_detail values(7, 'aaaaaaab', 4, 'commit');
insert into trade_detail values(8, 'aaaaaaac', 1, 'add');
insert into trade_detail values(9, 'aaaaaaac', 2, 'update');
insert into trade_detail values(10, 'aaaaaaac', 3, 'update again');
insert into trade_detail values(11, 'aaaaaaac', 4, 'commit');
```

```
mysql> select d.* from tradelog l, trade_detail d where d.tradeid=l.tradeid and l.id=2;
```

```
mysql> explain select d.* from tradelog l , trade_detail d where d.tradeid=l.tradeid and l.id=2;
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra       |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------------+
|  1 | SIMPLE      | l     | NULL       | const | PRIMARY,tradeid | PRIMARY | 4       | const |    1 |   100.00 | NULL        |
|  1 | SIMPLE      | d     | NULL       | ALL   | NULL          | NULL    | NULL    | NULL  |   11 |   100.00 | Using where |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------------+
```

执行过程


id=2
①
tradelog
L1
L2
L3
②
L2 (tradeid='aaaaaaab')
③
trade_detail
主键ID
R1
R2
R3
R4
R5
R6
R7
...
R11
4
结果集
R4
R5
R6
R7

字符集转换

第三步：

```
mysql> select * from trade_detail where tradeid=‘ L2.tradeid.value;
```

```
select * from trade_detail  where CONVERT(traideid USING
utf8mb4)=‘ L2.tradeid.value;
```

这就再次触发了我们上面说到的原则：对索引字段做函数操作，优化器会放弃走树搜索功能。

对比

```
mysql>select l.operator from tradelog l , trade_detail d where
d.tradeid=l.tradeid and d.id=4;
```

```
mysql> explain select l.operator from tradelog l , trade_detail d where d.tradeid=l.tradeid and d.id=4;
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
| id | select_type | table | partitions | type  | possible_keys | key     | key_len | ref   | rows | filtered | Extra |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
|  1 | SIMPLE      | d     | NULL       | const | PRIMARY       | PRIMARY | 4       | const |    1 |   100.00 | NULL  |
|  1 | SIMPLE      | l     | NULL       | ref   | tradeid       | tradeid | 131     | const |    1 |   100.00 | NULL  |
+----+-------------+-------+------------+-------+---------------+---------+---------+-------+------+----------+-------+
2 rows in set, 1 warning (0.00 sec)
```

```
select operator from tradelog  where traideid =CONVERT(‘  R4.tradeid.value
USING utf8mb4);
```

优化

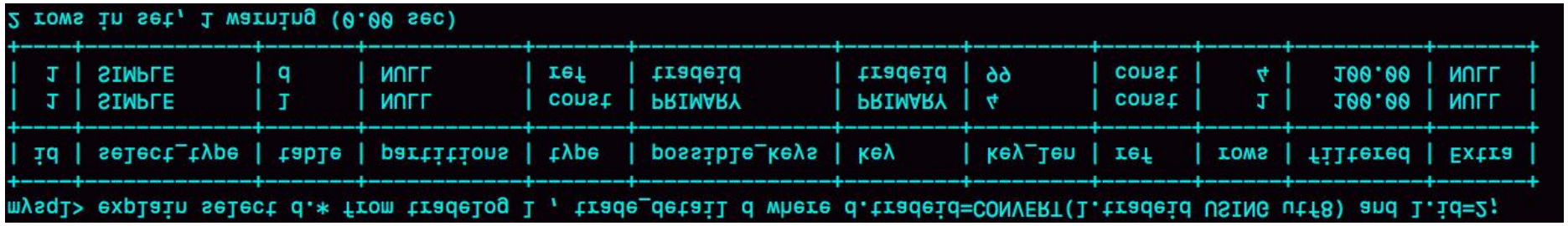
```
select d.* from tradelog l, trade_detail d where d.tradeid=l.tradeid and l.id=2;
```

1、比较常见的： 修改trade_detai表tradeid字段为utf8mb4:

```
alter table trade_detail modify tradeid varchar(32) CHARACTER SET utf8mb4 default null;
```

2、如果数据量大，业务上不允许这个ddl操作，改写sql：

```
mysql> select d.* from tradelog l , trade_detail d where d.tradeid=CONVERT(l.tradeid USING utf8) and l.id=2;
```



课后问题1:

```
语句1: select count(*) from tradelog ;          ---count(*)计算总行数
语句2: select count(1) from tradelog ;          ---count(1)计算总行数
语句3: select count(id) from tradelog ;         ---count(主键)计算总行数
语句4: select count(tradeid) from tradelog ; --count(辅助索引) 计算总行数
```

以上四条语句计算tradelog表的总行数，有什么差别？性能区别有多大，原理是什么？

课后问题2:

表：

```
CREATE TABLE `t` (
  `id` int(11) NOT NULL,
  `a` int(11) DEFAULT NULL,
  `b` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `a` (`a`),
  KEY `b` (`b`)
) ENGINE=InnoDB ;
==========================
```

往表 t 中插入 10 万行记录，取值按整数递增，即：(1,1,1)，(2,2,2)，(3,3,3) 直到 (100000,100000,10000)

```
mysql> explain select * from t where (a between 1 and 1000) and (b between 50000 and 100000) order by b limit 1;
```

为什么会选用b索引而放弃代价更小的a索引？

```
mysql> explain select * from t where (a between 1 and 1000)  and (b between 50000 and 100000) order by b limit 1;
+----+-------------+-------+------------+-------+---------------+------+---------+------+-------+----------+-----------------------------------------------------+
| id | select_type | table | partitions | type  | possible_keys | key  | key_len | ref  | rows  | filtered | Extra                                               |
+----+-------------+-------+------------+-------+---------------+------+---------+------+-------+----------+-----------------------------------------------------+
|  1 | SIMPLE      | t     | NULL       | range | a,b           | b    | 5       | NULL | 50198 |     1.00 | Using index condition; Using where |
+----+-------------+-------+------------+-------+---------------+------+---------+------+-------+----------+-----------------------------------------------------+
```

答疑

Q&A

Q&A

Thank You

完