阿里云

MySQL 查询优化

SQL诊断调优原则、原理及思路

演讲人

苏坡

关于我

阿里云

- 花名：苏坡
- **云掣科技 MSP**在线服务团队
- 标签：DBA、SQL调优、疑难排障、架构优化改造
- **袋鼠云**成立于2016年，是**国内数据中台领域创导者**
- **云掣**—袋鼠云旗下企业云服务品牌

课程大纲

阿里云

- 优化目的与目标
- 优化流程及思路
- 原理剖析
- MySQL的行为
- 常规优化策略

01 优化目的与目标

为什么要优化

阿里云

- ✓提高资源利用率
- ✓避免短板效应
- ✓提高系统吞吐量
- ✓同时满足更多用户的在线需求

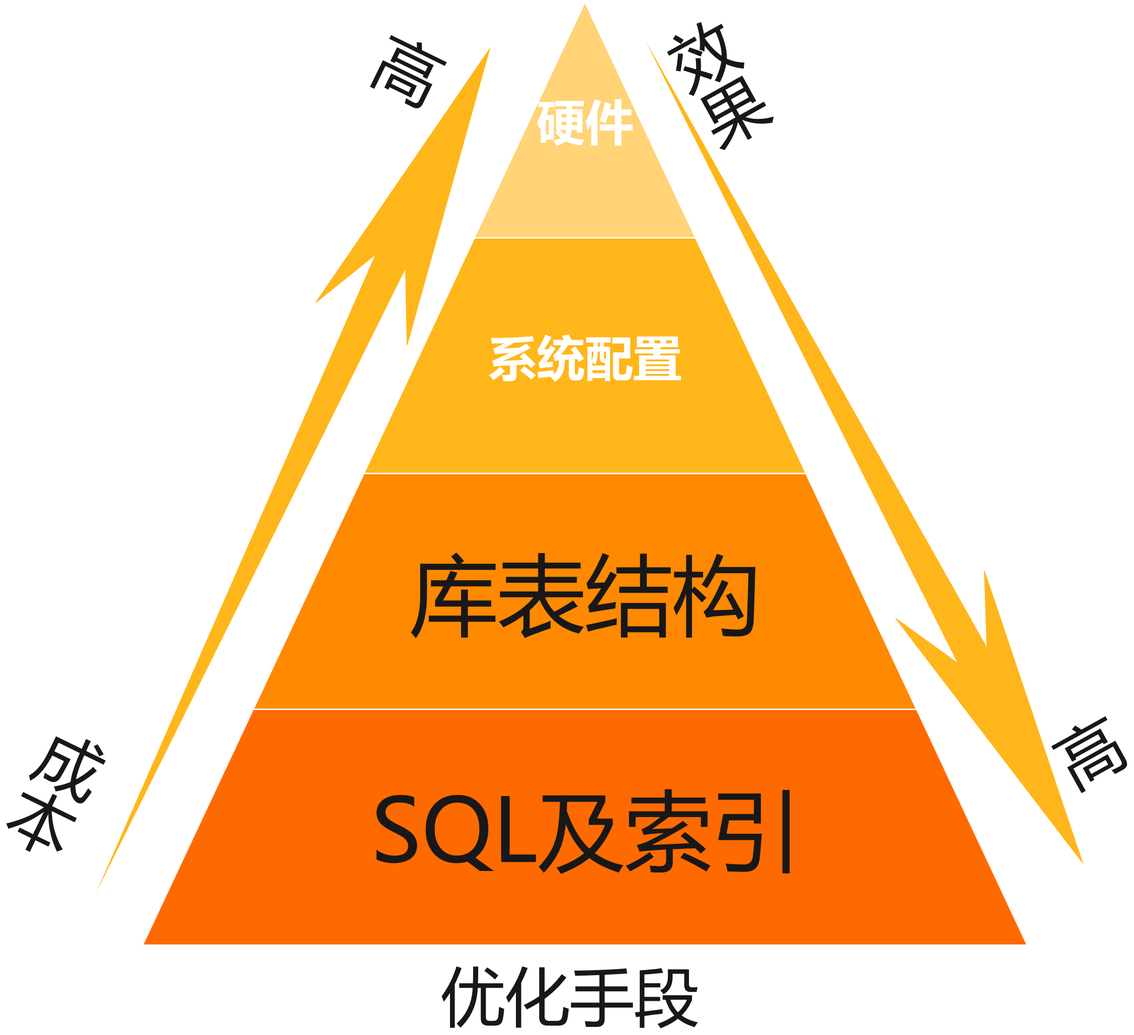
MySQL优化目标

阿里云

✓减少磁盘IO
- 全表扫描
- 磁盘临时表
- 日志、数据块fsync

✓减少网络带宽
- 返回太多数据
- 交互次数过多

✓降低CPU消耗
- 排序分组。order by, group by
- 聚合函数。max,min,sum...
- 逻辑读



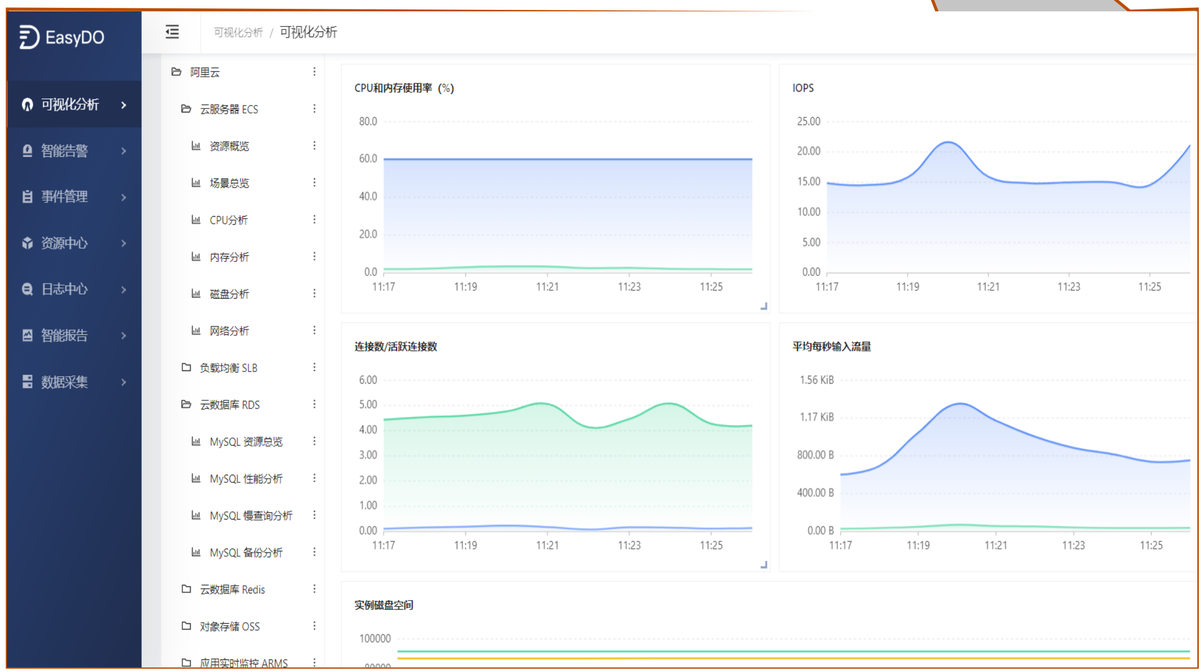
02 优化流程及思路

关注的指标

- CPU使用率
  - sql查询关键资源指标
  - 数据扫描、显式计算
- IOPS
  - 每秒io请求次数
  - 物理读写关键资源指标
- QPS/TPS
  - 吞吐量
  - 业务压力
- 会话数/活跃会话数
  - 应用配置
  - 执行效率
- Innodb逻辑读/物理读
  - 反映整体查询效率的引擎指标
- 临时表
  - 导致SQL执行效率下降的特殊行为

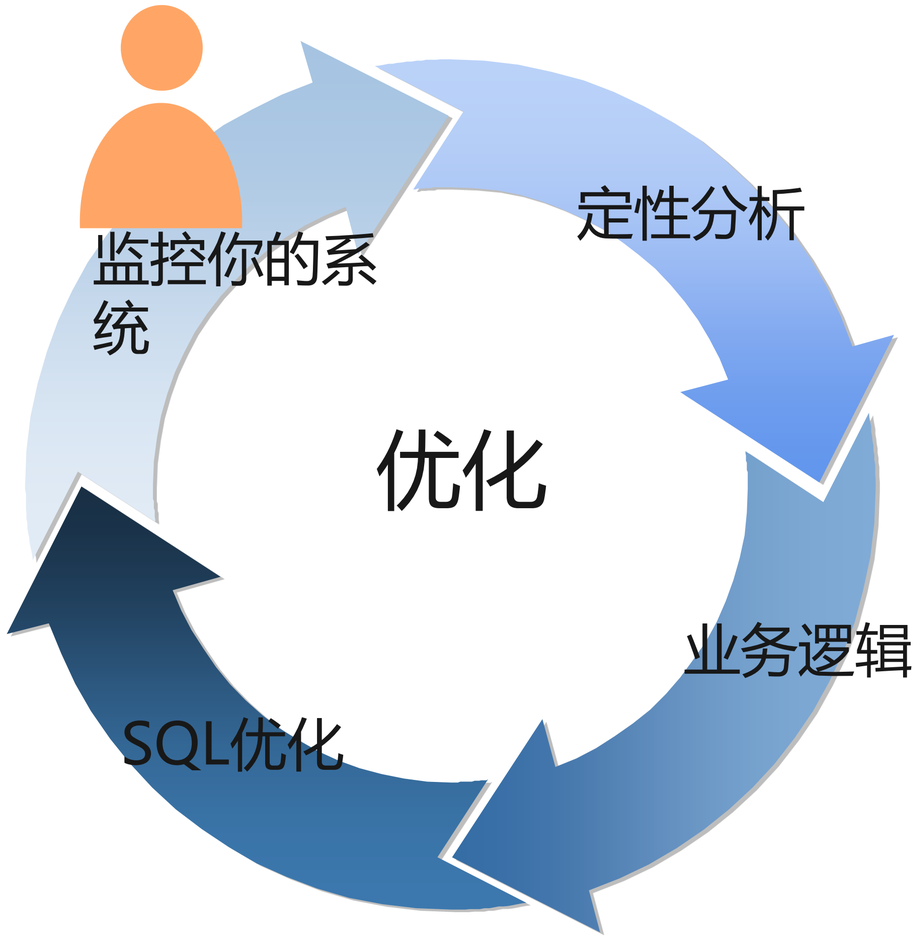
合理监控

top、iostat、sar、dstat
show variables like ...
show status like ...
...



MySQL优化流程

- 构建完备的监控体系
  - 细致合理的告警
  - 多维度图形化指标
  - 暴露性能缺陷，掌控大规模资源
- 分析定位问题
  - 异常时间区间
  - System log、DB Error Log
  - Slow Log
  - SQL执行统计
  - session
- 分析业务逻辑
  - 读写需求
  - 事务精简
  - 资源调用关系
- SQL优化
  - explain
  - SQL改写
  - 索引调整
  - 参数调整



SQL优化原则与方法

阿里云

原则

- 减少访问量：数据存取是数据库系统最核心功能，所以IO是数据库系统中最容易出现性能瓶颈，减少SQL访问IO量是SQL优化的第一步；数据块的逻辑读也是产生CPU开销的因素之一。
  - 减少访问量的方法：创建合适的索引、减少不必访问的列、使用索引覆盖、语句改写。

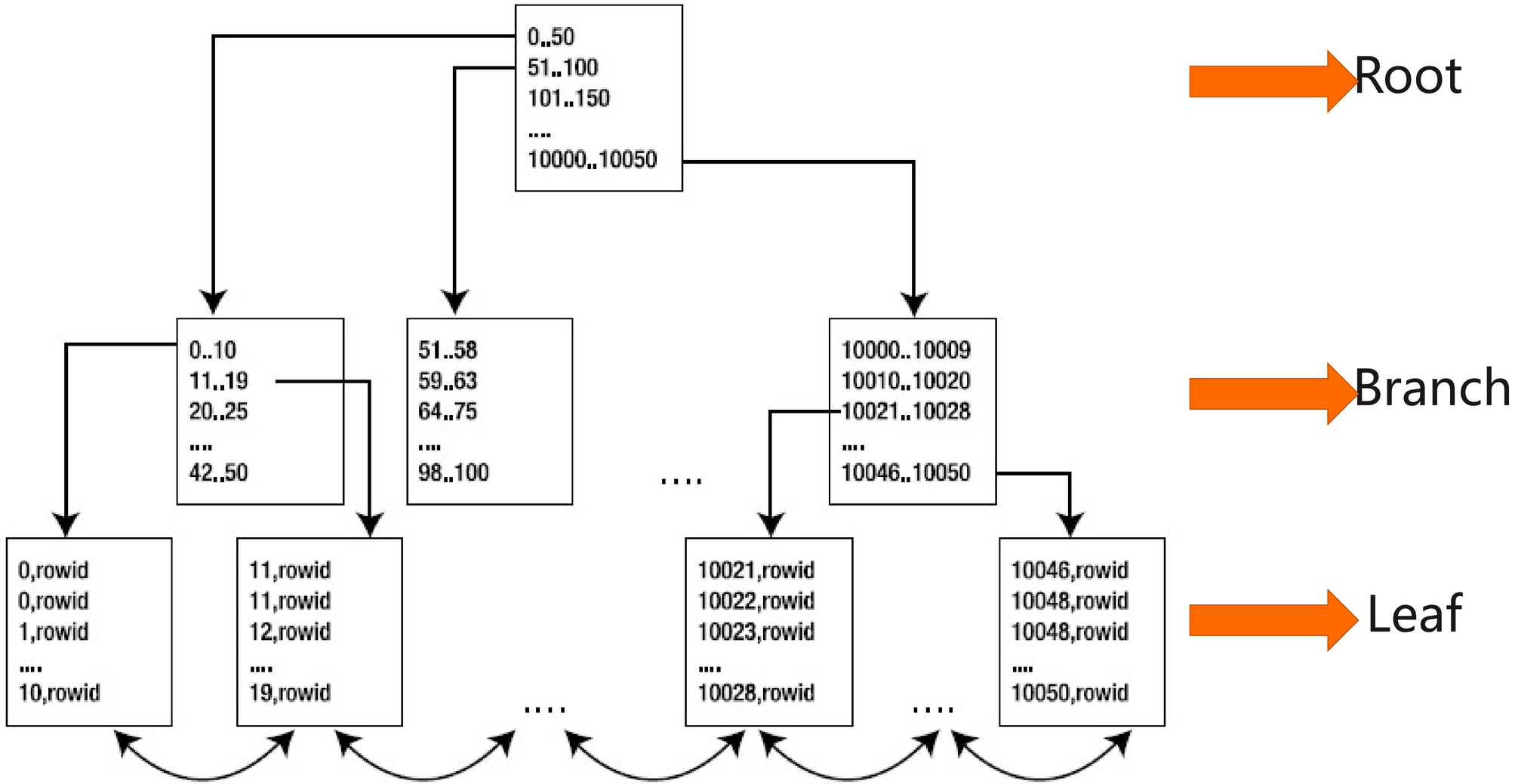
- 减少计算操作：计算操作进行优化也是SQL优化的重要方向。SQL中排序、分组、多表连接操作等计算操作都是CPU消耗的大户。
  - 减少SQL计算操作的方法：排序列加入索引、适当的列冗余、SQL拆分、计算功能拆分。

方法

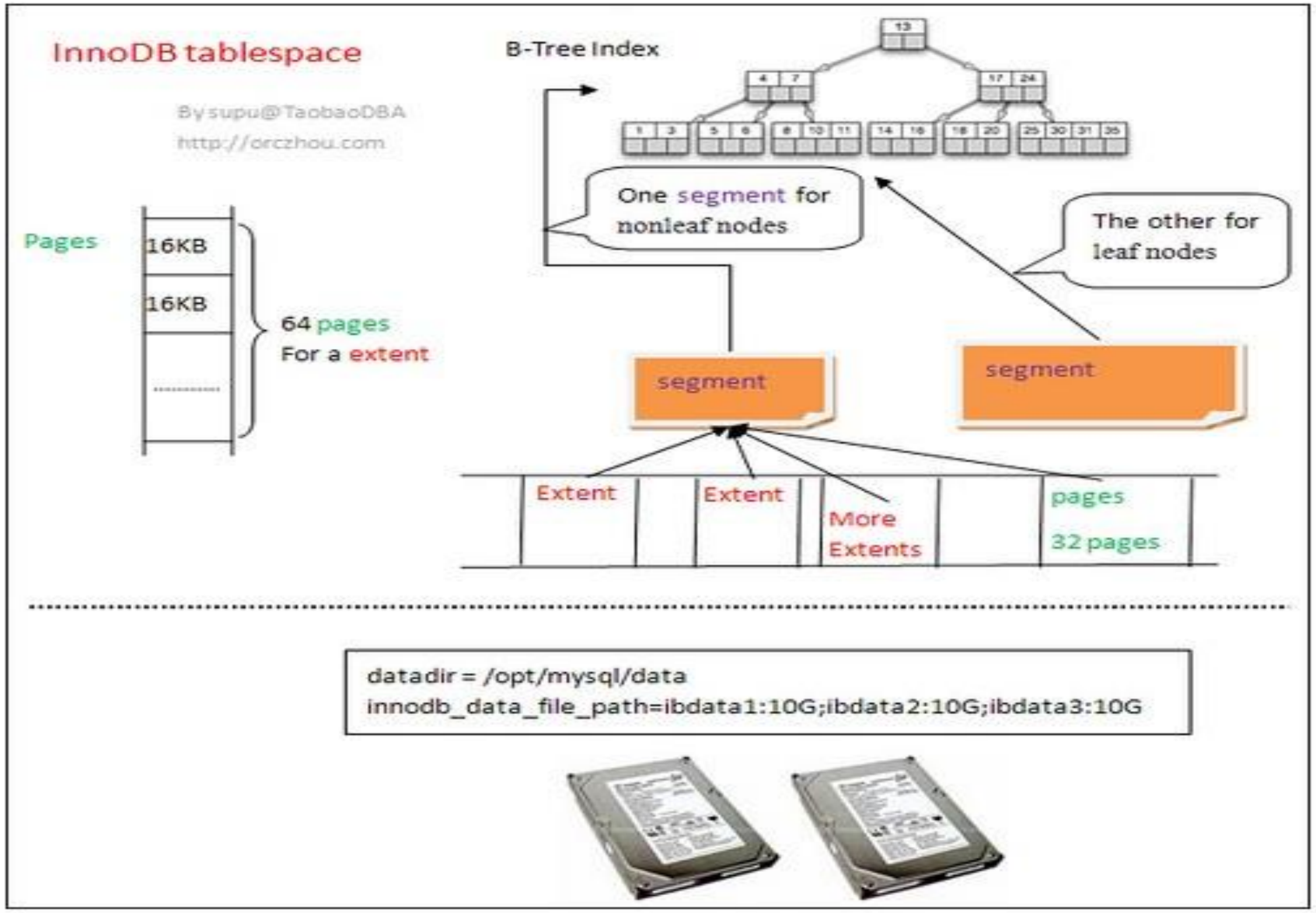
- 创建索引减少扫描量
- 调整索引减少计算量
- 索引覆盖（减少不必访问的列，避免回表查询）
- SQL改写
- 干预执行计划

03 原理剖析

B+ Tree index


0..50
51..100
101..150
....
10000..10050
0..10
11..19
20..25
....
42..50
51..58
59..63
64..75
....
98..100
10000..10009
10010..10020
10021..10028
....
10046..10050
0,rowid
0,rowid
1,rowid
....
10,rowid
11,rowid
11,rowid
12,rowid
....
19,rowid
10021,rowid
10022,rowid
10023,rowid
....
10028,rowid
10046,rowid
10048,rowid
10048,rowid
....
10050,rowid
Root
Branch
Leaf

Innodb Table



IOT
有序存储

索引检索过程

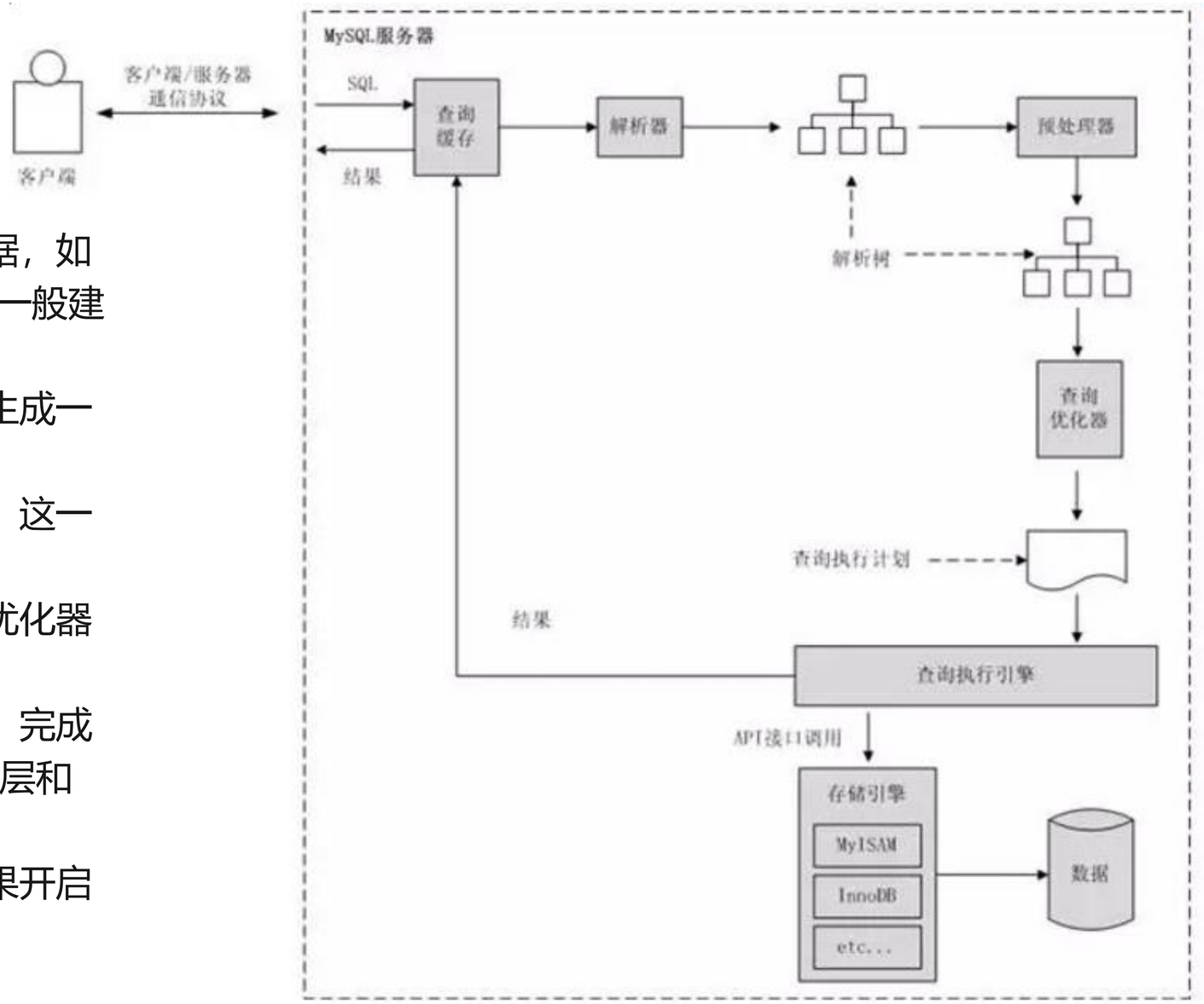
非主键查询

主键查询

覆盖索引

04 MySQL的行为

MySQL SQL执行过程



1.客户提交一条语句
2.先在查询缓存查看是否存在对应的缓存数据，如有则直接返回(一般有的可能性极小，因此一般建议关闭查询缓存)。
3.交给解析器处理，解析器会将提交的语句生成一个解析树。
4.预处理器会处理解析树，形成新的解析树。这一阶段存在一些SQL改写的过程。
5.改写后的解析树提交给查询优化器。查询优化器生成执行计划。
6.执行计划交由执行引擎调用存储引擎接口，完成执行过程。这里要注意，MySQL的Server层和Engine层是分离的。
7.最终的结果由执行引擎返回给客户端，如果开启查询缓存的话，则会缓存。

MySQL SQL执行过程

阿里云

SQL执行顺序：

(8) SELECT (9) DISTINCT <select_list>
(1) FROM <left_table>
(3) <join_type> JOIN <right_table>
(2) ON <join_condition>
(4) WHERE <where_condition>
(5) GROUP <group_by_list>
(6) WITH {CUBE|ROLLUP}
(7) HAVING <having_condition>
(10) ORDER BY <order_by_list>
(11) LIMIT <limit_number>

阿里云

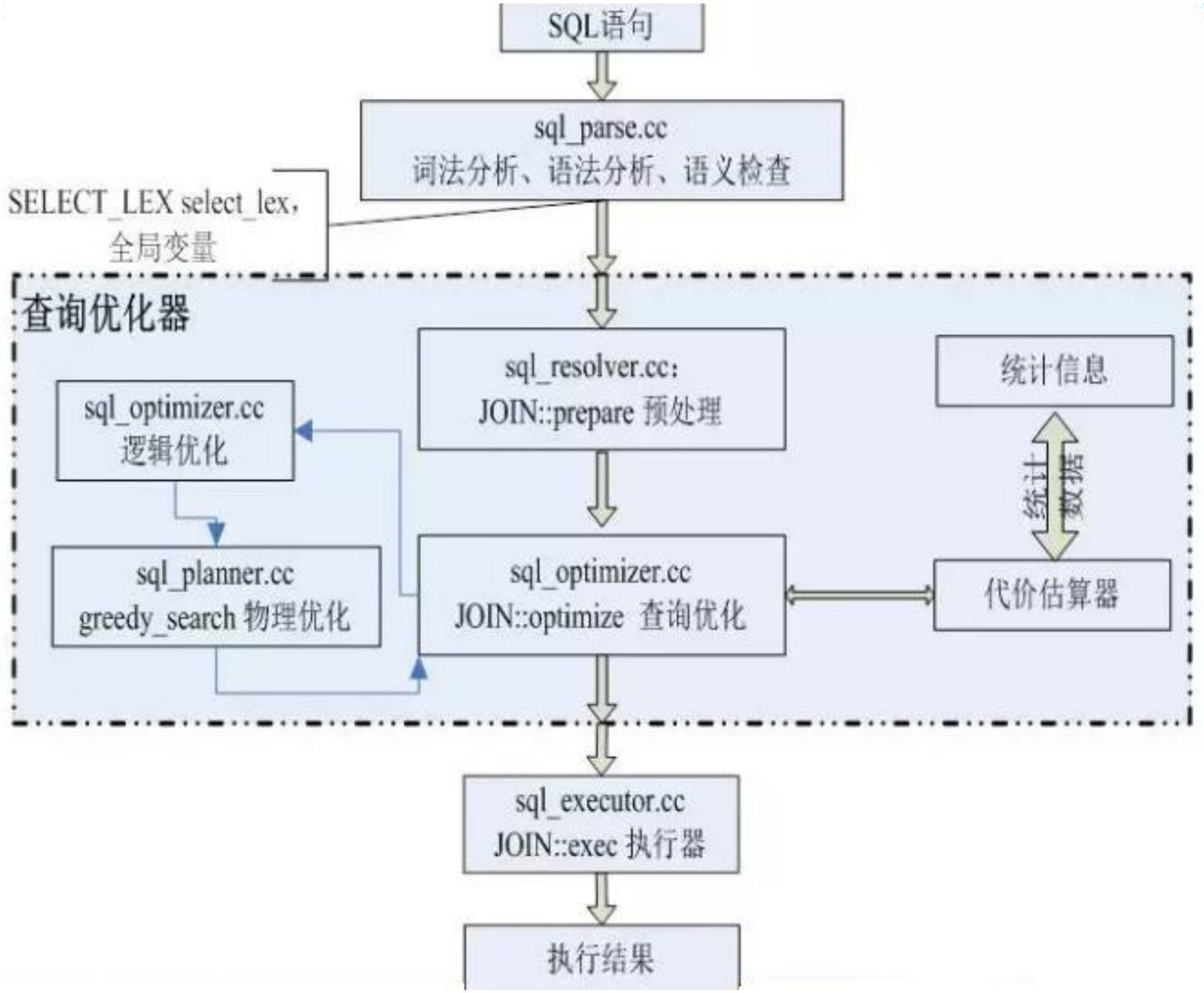
查询优化器

- 负责生成 SQL 语句的有效执行计划的数据库组件
- 优化器是数据库的核心价值所在，它是数据库的“大脑”
- 优化SQL，某种意义上就是理解优化器的行为
- 优化的依据是执行成本（CBO）
- 优化器工作的前提是了解数据，工作的目的是解析SQL，生成执行计划

MySQL优化器与执行计划

查询优化器工作过程

1.词法分析、语法分析、语义检查
2.预处理阶段(查询改写等)
3.查询优化阶段(可详细划分为逻辑优化、物理优化两部分)
4.查询优化器优化依据，来自于代价估算器估算结果(它会调用统计信息作为计算依据)
5.交由执行器执行



查看和干预执行计划

阿里云

执行计划：

- explain [extended] SQL_Statement

```
mysql> explain select * from payment where rental_id > 1000 order by payment_date;
+----+-------------+---------+------+--------------------+------+---------+------+-------+-----------------------------+
| id | select_type | table   | type | possible_keys      | key  | key_len | ref  | rows  | Extra                       |
+----+-------------+---------+------+--------------------+------+---------+------+-------+-----------------------------+
|  1 | SIMPLE      | payment | ALL  | fk_payment_rental | NULL | NULL    | NULL | 15511 | Using where; Using filesort |
+----+-------------+---------+------+--------------------+------+---------+------+-------+-----------------------------+
1 row in set (0.00 sec)
```

优化器开关：

- show variables like 'optimizer_switch'

```
mysql> show variables like 'optimizer_switch'\G
*************************** 1. row ***************************
Variable_name: optimizer_switch
        Value: index_merge=on,index_merge_union=on,index_merge_sort_union=on,index_merge_intersection=on,engine
_condition_pushdown=on,index_condition_pushdown=on,mrr=on,mrr_cost_based=on,block_nested_loop=on,batched_key_ac
cess=off,materialization=on,semijoin=on,loosescan=on,firstmatch=on,duplicateweedout=on,subquery_materialization
_cost_based=on,use_index_extensions=on,condition_fanout_filter=on,derived_merge=on
1 row in set (0.01 sec)
```

processlist

- show [full] processlist
- information_schema.processlist

copy to tmp table：出现在某些alter table语句的copy table操作
Copying to tmp table on disk：由于临时结果集大于tmp_table_size，正在将临时表从内存存储转为磁盘存储以此节省内存
converting HEAP to MyISAM：线程正在转换内部MEMORY临时表到磁盘MyISAM临时表
Creating sort index：正在使用内部临时表处理select查询
Sorting index：磁盘排序操作的一个过程
Sending data：正在处理SELECT查询的记录，同时正在把结果发送给客户端
Waiting for table metadata lock: 等待元数据锁
...

05 常规优化策略

SELECT优化-order by

order by查询的两种情况：

- Using index。MySQL直接通过索引返回有序记录，不需要额外的排序操作，操作效率较高
- Using filesort。无法只通过索引获取有序结果集，需要额外的排序，某些特殊情况下，会出现Using temporary

优化目标：尽量通过索引来避免额外的排序，减少CPU资源的消耗

✓where条件和order by使用相同的索引

✓order by的顺序和索引顺序相同

✓order by的字段同为升序或降序

注：当where条件中的过滤字段为覆盖索引的前缀列，而order by字段是第二个索引列时，只有where条件是const匹配时，才可以通过索引消除排序，而between...and或>?、<?这种range匹配都无法避免filesort操作

SELECT优化-order by

阿里云

当无法避免filesort操作时，优化思路就是让filesort的操作更快

排序算法：

- 两次扫描算法。两次访问数据，第一步获取排序字段的行指针信息，在内存中排序，第二步根据行指针获取记录
- 一次扫描算法。一次性取出满足条件的所有记录，在排序区中排序后输出结果集。是采用空间换时间的方式

注：需要排序的字段总长度越小，越趋向于第二种扫描算法，MySQL通过max_length_for_sort_data参数的值来进行参考选择

优化策略：

1、适当调大max_length_for_sort_data这个参数的值，让优化器更倾向于选择第二种扫描算法

2、只使用必要的字段，不要使用select *的写法

3、适当加大sort_buffer_size这个参数的值，避免磁盘排序的出现（线程参数，不要设置过大）

SELECT优化-Subquery

阿里云

- 子查询会用到临时表，需尽量避免
- 可以使用效率更高的join查询来替代

优化策略：

等价改写、反嵌套

如下SQL：

```
select * from customer where customer_id not in (select customer_id from payment)
```

改写形式：

```
select * from customer a left join payment b on a.customer_id=b.customer_id where b.customer_id is null
```

SELECT优化-limit

阿里云

- 分页查询，就是将过多的结果在有限的界面上分好多页来显示。
- 其实质是每次查询只返回有限行，翻页一次执行一次。

优化目标：

1、消除排序

2、避免扫描到大量不需要的记录

SQL场景（film_id为主键）：

```
select film_id,description from film order by title limit 10000,20
```

此时MySQL排序出前10020条记录后仅仅需要返回第10001到10020条记录，前10000条记录造成额外的代价消耗

SELECT优化-limit

阿里云

优化策略一：

覆盖索引

```
alter table film add index idx_lmtest(title,description);
```

- 记录直接从索引中获取，效率最高
- 仅适合查询字段较少的情况

优化策略二：

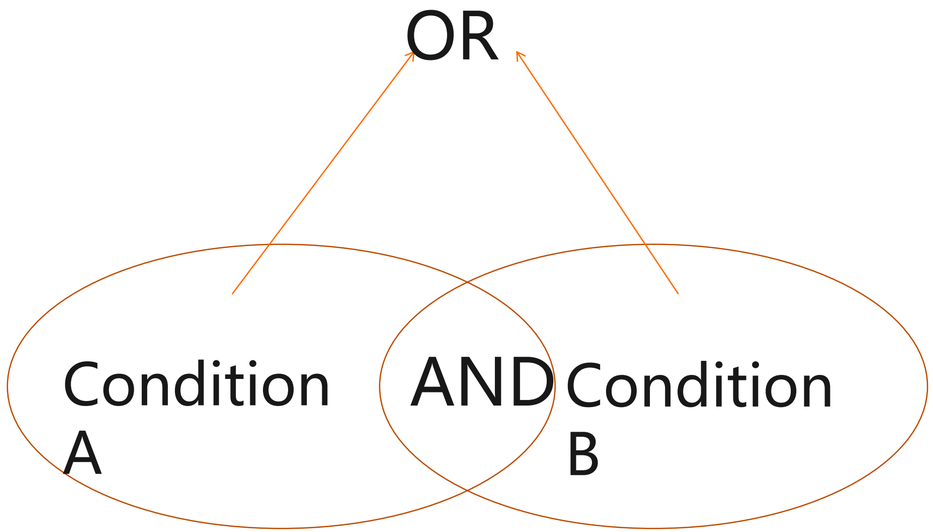
SQL改写

```
select a.film_id,a.description from film a inner join (select film_id from film
order by title limit 1000,20) b on a.film_id=b.film_id;
```

- 优化的前提是title字段有索引
- 思路是从索引中取出20条满足条件记录的主键值，然后回表获取记录

SELECT优化-or/and condition

- and结果集为关键字前后过滤结果的交集
- or结果集为关键字前后分别查询的并集
- and条件可以在前一个条件过滤基础上过滤
- or条件被处理为UNION，相当于两个单独条件的查询
- 复合索引对于or条件相当于一个单列索引



处理策略：

✓and子句多个条件中拥有一个过滤性较高的索引即可

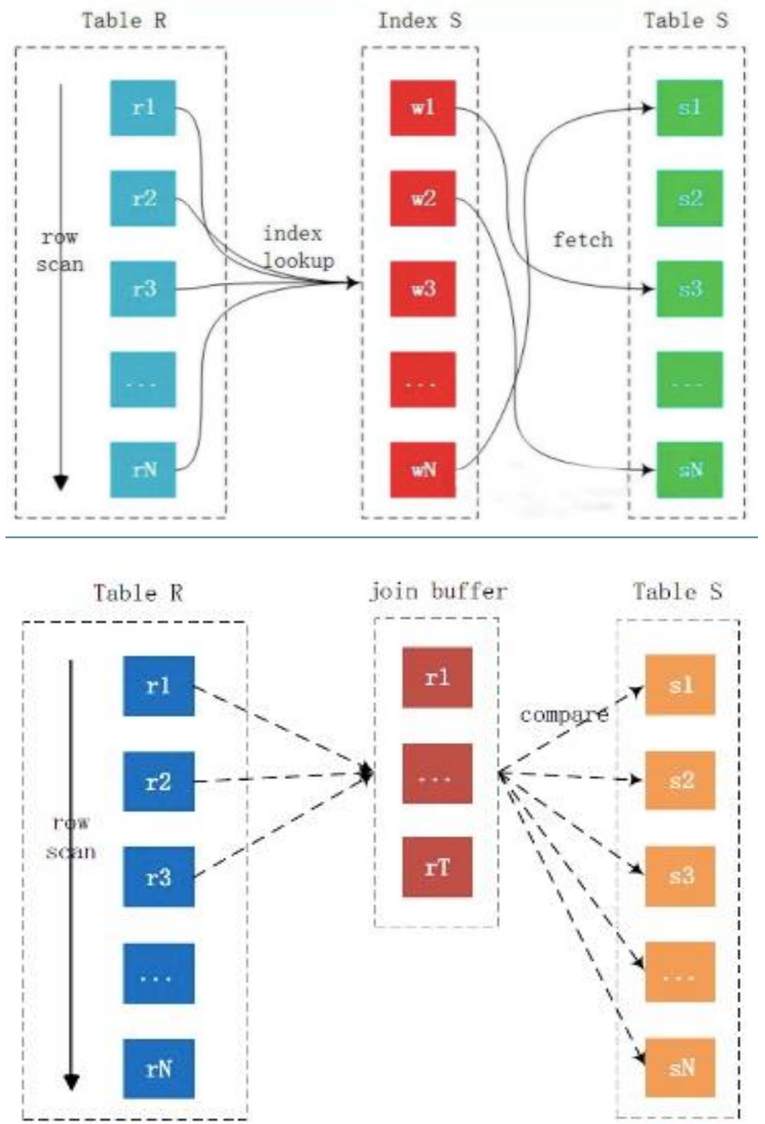
✓or条件前后字段均要创建索引

✓为最常用的and组合条件创建复合索引

join优化

- Nested-Loop Join算法

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions, send to
client
    }
  }
}
```



- 关联字段索引：每层内部循环仅获取需要关心的数据
  - 引申算法：Bloack Nested-Loop
- 小表驱动原则：减少循环次数
  - 小表：返回结果集较少的表

join优化

阿里云

- 关联字段索引的必要性

```
mysql>  select count(*) from film2 a  join film_category2 b on a.film_id=b.film_id;

+----------+
| count(*) |
+----------+
|   512000 |
+----------+
1 row in set (1 min 50.26 sec)

mysql>
mysql> explain  select count(*) from film2 a  join film_category2 b on a.film_id=b.film_id;
+----+-------------+-------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------------+
| id | select_type | table | partitions | type | possible_keys | key  | key_len | ref  | rows   | filtered | Extra                                              |
+----+-------------+-------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------------+
|  1 | SIMPLE      | b     | NULL       | ALL  | NULL          | NULL | NULL    | NULL |   4000 |   100.00 | NULL                                               |
|  1 | SIMPLE      | a     | NULL       | ALL  | NULL          | NULL | NULL    | NULL | 127042 |    10.00 | Using where; Using join buffer (Block Nested Loop) |
+----+-------------+-------+------------+------+---------------+------+---------+------+--------+----------+----------------------------------------------------+
2 rows in set, 1 warning (0.00 sec)

mysql>
```

join优化

阿里云

- 关联字段索引的必要性

```
mysql>
mysql> alter table film2 add index(film_id);
Query OK, 0 rows affected (0.32 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> explain  select count(*) from film2 a  join film_category2 b on a.film_id=b.film_id;
+----+-------------+-------+------------+------+---------------+---------+---------+------------------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key     | key_len | ref              | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+---------+---------+------------------+------+----------+-------------+
|  1 | SIMPLE      | b     | NULL       | ALL  | NULL          | NULL    | NULL    | NULL             | 4000 |   100.00 | NULL        |
|  1 | SIMPLE      | a     | NULL       | ref  | film_id       | film_id | 2       | sakila.b.film_id |  125 |   100.00 | Using index |
+----+-------------+-------+------------+------+---------------+---------+---------+------------------+------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)

mysql>  select count(*) from film2 a  join film_category2 b on a.film_id=b.film_id;
+----------+
| count(*) |
+----------+
|   512000 |
+----------+
1 row in set (0.31 sec)
```

join优化

阿里云

- 小表驱动原则

忽略b表的索引，使b表作为驱动表：

```
mysql> explain  select count(*) from film2 a  join film_category2 b ignore index(film_id) on a.film_id=b.film_id;
+----+-------------+-------+------------+------+---------------+---------+---------+------------------+------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key     | key_len | ref              | rows | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+---------+---------+------------------+------+----------+-------------+
|  1 | SIMPLE      | b     | NULL       | ALL  | NULL          | NULL    | NULL    | NULL             | 4000 |   100.00 | NULL        |
|  1 | SIMPLE      | a     | NULL       | ref  | film_id       | film_id | 2       | sakila.b.film_id |  125 |   100.00 | Using index |
+----+-------------+-------+------------+------+---------------+---------+---------+------------------+------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)

mysql>
mysql>   select count(*) from film2 a  join film_category2 b ignore index(film_id) on a.film_id=b.film_id;
+----------+
| count(*) |
+----------+
|   512000 |
+----------+
1 row in set (0.31 sec)
```

join优化

阿里云

- 小表驱动原则

忽略a表的索引，使a表作为驱动表：

```
mysql>
mysql> explain   select count(*) from film2 a ignore index(film_id)  join film_category2 b on a.film_id=b.film_id;
+----+-------------+-------+------------+------+---------------+---------+---------+------------------+--------+----------+-------------+
| id | select_type | table | partitions | type | possible_keys | key     | key_len | ref              | rows   | filtered | Extra       |
+----+-------------+-------+------------+------+---------------+---------+---------+------------------+--------+----------+-------------+
|  1 | SIMPLE      | a     | NULL       | ALL  | NULL          | NULL    | NULL    | NULL             | 127042 |   100.00 | NULL        |
|  1 | SIMPLE      | b     | NULL       | ref  | film_id       | film_id | 2       | sakila.a.film_id |      4 |   100.00 | Using index |
+----+-------------+-------+------------+------+---------------+---------+---------+------------------+--------+----------+-------------+
2 rows in set, 1 warning (0.00 sec)

mysql>
mysql>   select count(*) from film2 a ignore index(film_id)  join film_category2 b on a.film_id=b.film_id;
+----------+
| count(*) |
+----------+
|   512000 |
+----------+
1 row in set (0.53 sec)
```

```
mysql>   select count(*) from film2;
+----------+
| count(*) |
+----------+
|   128000 |
+----------+
1 row in set (0.05 sec)

mysql>   select count(*) from film_category2;
+----------+
| count(*) |
+----------+
|     4000 |
+----------+
1 row in set (0.00 sec)
```

insert优化

阿里云

优化策略一：

减少交互次数

如批量插入语句：

```
insert into test values(1,2,3);
insert into test values(4,5,6);
insert into test values(7,8,9);
...
```

可改写为如下形式：

```
insert into test values(1,2,3),(4,5,6),(7,8,9) ...;
```

优化策略二：

文本装载方式

通过LOAD DATA INFILE句式，从文本装载数据，通常比insert语句快20倍

06 小结

MySQL查询优化

阿里云

✓优化的目的是让资源发挥价值
✓SQL和索引是调优的关键，往往可以起到“四两拨千斤”的效果

✓充分了解核心指标，并构建完备的监控体系，这是优化工作的前提
✓SQL优化的原则是减少数据访问及计算
✓常用的优化方法主要是调整索引、改写SQL、干预执行计划

✓innodb的表是典型的IOT，数据本身是B+ tree索引的叶节点
✓扫描二级索引可以直接获取数据，或者返回主键id
✓优化器是数据库的大脑，我们要了解优化器，并观测以及干预MySQL的行为