理解 Mysql 索引原理及特性 | 京东物流技术团队

原创 京东云开发者 数据库 2023/12/13 09:46 阅读数 2K

本文被收录于专区
数据库

进入专区参与更多专题讨论 >

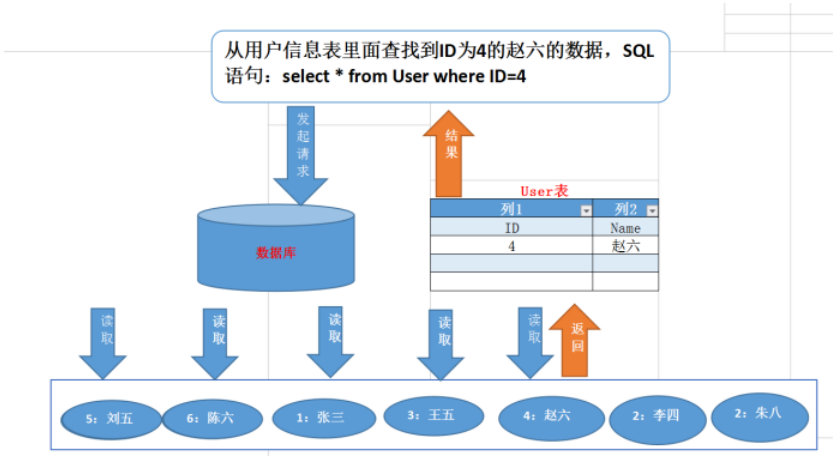
作为开发人员，碰到了执行时间较长的 sql 时，基本上大家都会说" 加个索引吧"。但是索引是什么东西，索引有哪些特性，下面和大家简单讨论一下。

1 索引如何工作，是如何加快查询速度

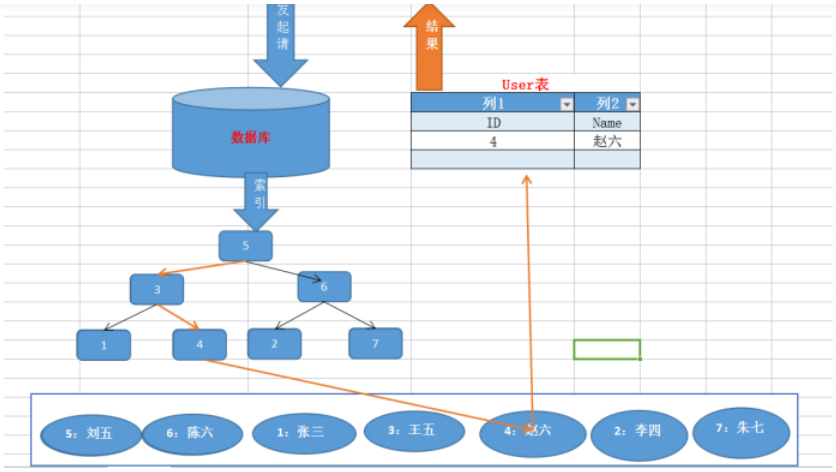
索引就好比书本的目录，提高数据库表数据访问速度的数据库对象。当我们的请求打过来之后，如果有目录，就会快速的定位到章节，再从章节里找到数据。如果没有目录，如大海捞针一般，难度可见一斑。这就是我们经常碰到的罪魁祸首，全表扫描。

一条索引记录中包含的基本信息包括：键值（即你定义索引时指定的所有字段的值）+ 逻辑指针（指向数据页或者另一索引页）。通常状况下，由于索引记录仅包含索引字段值（以及 4-9 字节的指针），索引实体比真实的数据行要小许多，索引页相较数据页来说要密集许多。一个索引页可以存储数量更多的索引记录，这意味着在索引中查找时在 I/O 上占很大的优势，理解这一点有助于从本质上了解使用索引的优势，也是大部分性能优化所需要切入的点。

1）没有索引的情况下访问数据：



2）使用平衡二叉树结构索引的情况下访问数据：



第一张图没有使用索引我们会进行顺序查找，依照数据顺序逐个进行匹配，进行了 5 次寻址才查询出所需数据，第二张图用了一个简单的平衡二叉树索引之后我们只用了 3 次，这还是数据量小的情况下，数据量大了效果更明显，所以总结来说创建索引就是为了加快数据查找速度；

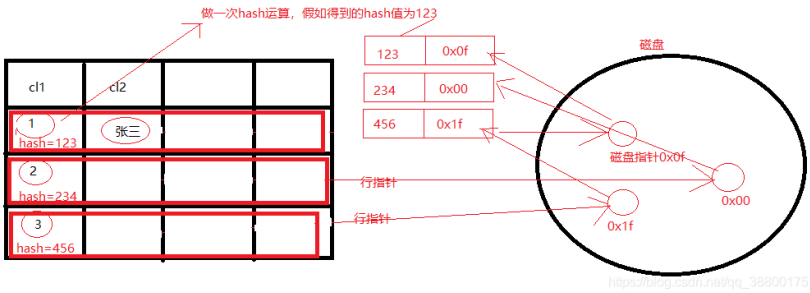
2 索引的组成部分和种类

常见的索引的实现方式有很多种，比如 hash、数组、树，下面为大家介绍下这几种模型使用上有什么区别

2.1 hash

hash 思路简单，就是把我们插入的 key 通过 hash 函数算法 (以前一般是取余数，就好比 hashmap 的计算方式移位异或之类的)，计算出对应的 value，把这个 value 放到一个位置，这个位置叫做哈希槽。对应磁盘位置指针放入 hash 槽里面。一句话总结 hash 索引，就是存储了索引字段的 hash 值和数据所在磁盘文件指针。

但是不可避免的是，无论什么算法，数据量大了之后难免会出现不同的数据被放在一个 hash 槽里面。比如字典上的 “吴” 和” 武” 就是同音，你查字典的时候到这里只能顺序往下去找了。索引的处理也是这样，会拉出一个链表，需要的时候顺序遍历即可。



- 缺点：无序索引，区间查询性能低，因为区间查询会造成多次访问磁盘，多次 io 耗时是很难接受的。
- 优点：insert 迅速，只需往后补就行
- 场景：等值查询， 比如 memcached 。不适用大量重复数据的列，避免 hash 冲突
- 总结：想成 java 的 hashmap 即可

2.2 有序数组

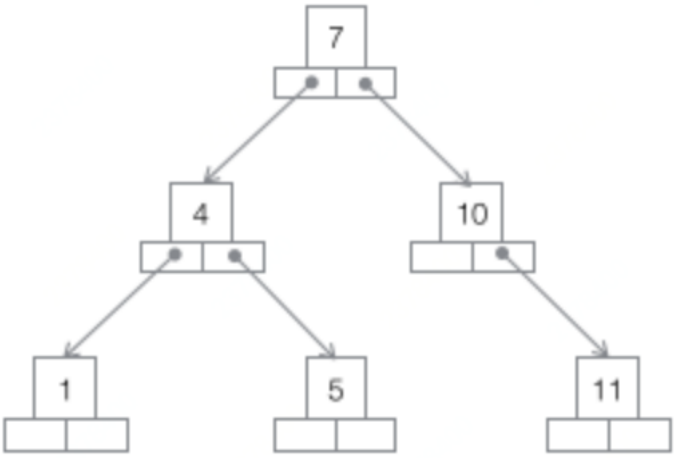
如果我们需要区间查询的时候，hash 索引的性能就不尽如人意了。这个时候有序数组的优势就能体现出来了。

当我们需要从一个有序数组里取 A 和 B 之间的值时，只需要通过二分法定位到 A 的位置，时间复杂度 O (log (N)), 接着从 A 遍历到 B 即可，论速度的话，基本上可以说是最快的了。但是当我们需要更新的时候，需要进行的操作就很多了。如果需要插入一条数据，你需要挪动数据之后的所有数据，浪费性能。所以总结来说，只有不怎么变化的数据适合有序数组结构的索引。

- 场景：归档查询，日志查询等极少变化的
- 总结：就是顺序排的数组

2.3 二叉搜索树

基本原则是树的左节点都小于父节点，右节点都大于父节点



这里我们就能看出来，二叉搜索树的查询效率原则上是 O (log (N))，为了保证是平衡二叉树，更新效率也是 O (log (N))。但是数据很多的情况树的高度会达到很高，过多次访问磁盘，是不可取的。并且极端情况下，树退化成链表，查询的复杂度会被拉成 O (n)。

进化成多叉树，也就是多个子节点的时候，会大大的减少树的高度，降低访问磁盘。

- 缺点：数据量大的时候，树会过高，导致多次访问磁盘
- 优点：进化成多叉树，会降低树高，访问磁盘次数。
- 场景：适用很多场景
- 总结：左小右大的树

2.4 B 树

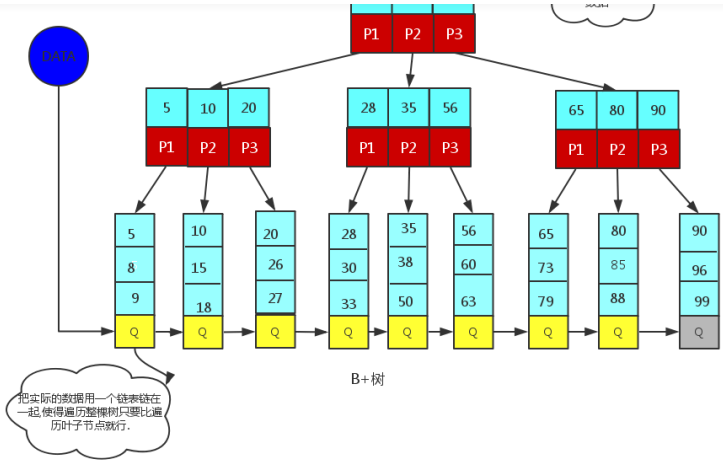
在每个节点存储多个元素，在每个节点尽可能多的存储数据。每个节点可以存储 1000 个索引（16k/16=1000），这样就将二叉树改造成了多叉树，通过增加树的叉树，将树从高瘦变为矮胖。构建 1 百万条数据，树的高度只需要 2 层就可以（1000*1000=1 百万），也就是说只需要 2 次磁盘 IO 就可以查询到数据。磁盘 IO 次数变少了，查询数据的效率也就提高了。

这种数据结构我们称为 B 树，B 树是一种多叉平衡查找树

2.5 B + 树

B + 树和 B 树最主要的区别在于非叶子节点是否存储数据的问题。

- B 树：非叶子节点和叶子节点都会存储数据。
- B + 树：只有叶子节点才会存储数据，非叶子节点至存储键值。叶子节点之间使用双向指针连接，最底层的叶子节点形成了一个双向有序链表。

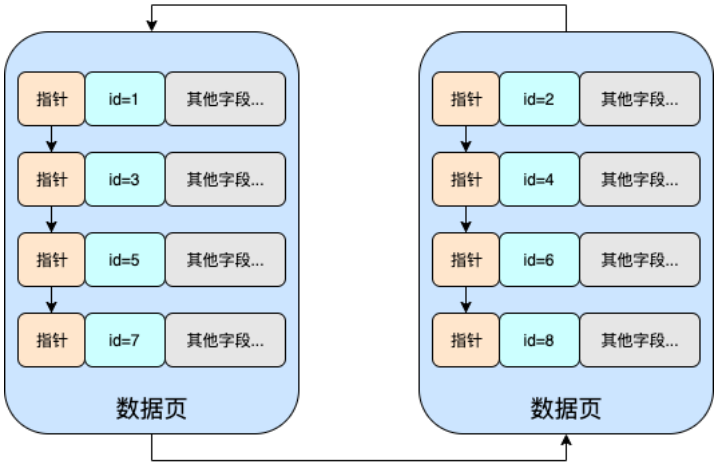


正是因为 B + 树的叶子节点是通过链表连接的，所以找到下限后能很快进行区间查询，比正常的中序遍历快

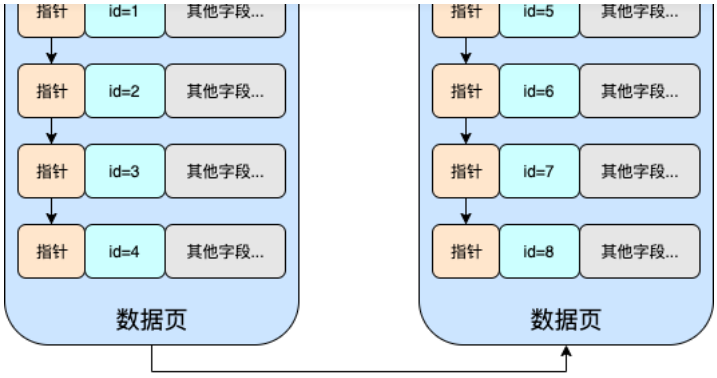
3 索引的维护

当你 insert 一条数据的时候，索引需要做出必要的操作来保证数据的有序型。一般自增数据直接在后面加就行了，特殊情况下如果数据加到了中间，就需要挪动后面所有的数据，这样效率比较受影响。

最糟糕的情况，如果当前的数据页（页是 mysql 存储的最小单位）存满了，需要申请一个新的数据页，这个过程被称为页分裂。如果造成了页分裂的话，势必会造成性能的影响。但是 mysql 并不是无脑的数据分裂，如果你是从中间进行数据分裂的话，对于自增主键，会导致一半的性能浪费。mysql 会根据你的索引的类型，和追踪插入数据的情况决定分裂的方式，一般都存在 mysql 数据页的 head 里面，如果是零散的插入，会从中间分裂。如果是顺序插入，一般是会选择插入点开始分裂，或者插入点往后几行导致的。决定是否从中间分裂，还是从最后分裂。



如果插入的是不规则的数据，没法保证后一个值比前一个大，就会触发上面说的分裂逻辑，最后达到下面的效果



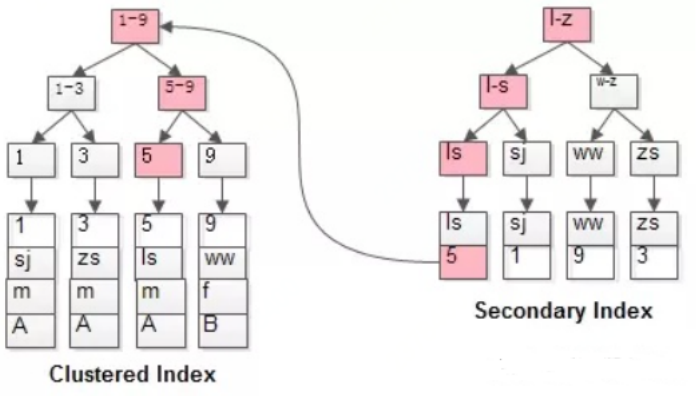
所以绝大多数情况下，我们都需要使用自增索引，除非需要业务自定义主键，最好能保证只有一个索引，且索引是唯一索引。这样可以避免回表，导致查询搜索两棵树。保证数据页的有序性，可以更好的使用索引。

4 回表

通俗的讲就是，如果索引的列在 select 所需获得的列中（因为在 mysql 中索引是根据索引列的值进行排序的，所以索引节点中存在该列中的部分值）或者根据一次索引查询就能获得记录就不需要回表，如果 select 所需获得列中有大量的非索引列，索引就需要先找到主键，再到表中找到相应的列的信息，这就叫回表。

要介绍回表自然就得介绍聚集索引和非聚集索引
InnoDB 聚集索引的叶子节点存储行记录，因此， InnoDB 必须要有，且只有一个聚集索引：

- 如果表定义了主键，则 PK 就是聚集索引；
- 如果表没有定义主键，则第一个非空唯一索引（not NULL unique）列是聚集索引；
- 否则，InnoDB 会创建一个隐藏的 row-id 作为聚集索引；



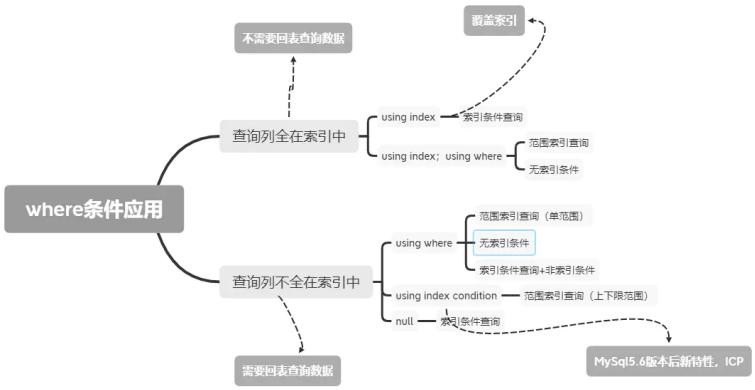
当我们使用普通索引查询方式，则需要先搜索普通索引树，然后得到主键 ID 后，再到 ID 索引树搜索一次。因为非主键索引的叶子节点里面，实际存的是主键的 ID。这个过程虽然用了索引，但实际上底层进行了两次索引查询，这个过程就称为回表。也就是说，基于非主键索引的查询需要多扫描一棵索引树。因此，我们在应用中应该尽量使用主键查询。或者有高频请求时，合理建立联合索引，防止回表。

5 索引覆盖

一句话表达的话，是只需要在一棵索引树上就能获取 SQL 所需的所有列数据，无需回表，速度更快。落实到 sql 上的话，只要执行计划里面的输出结果 Extra 字段为 Using index 时，能够触发索引覆盖。

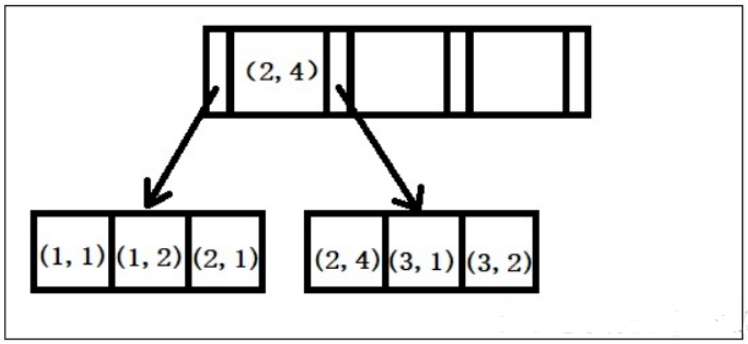
常见的优化手段，就是上面提到的，将查询的字段都建到索引里面，至于 dba 愿不愿意让你建，那就需要你们自己 battle 了。

primary key，也算索引覆盖，无需额外成本。



6 最左匹配原则

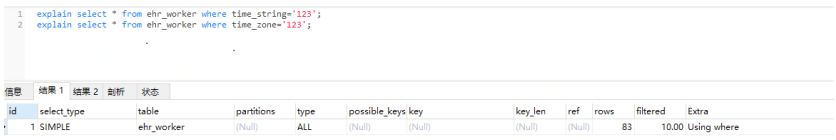
简单来说，就是你使用 'xx%'的时候，符合条件的话也会使用索引。
如果是联合索引的话，我举个例子，创建一个（a,b）的联合索引



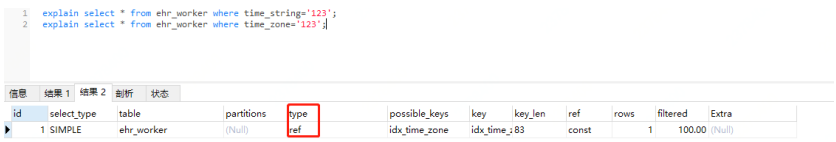
可以看到 a 的值是有顺序的，1，1，2，2，3，3，而 b 的值是没有顺序的 1，2，1，4，1，2。但是我们又可发现 a 在等值的情况下，b 值又是按顺序排列的，但是这种顺序是相对的。这是因为 MySQL 创建联合索引的规则是首先会对联合索引的最左边第一个字段排序，在第一个字段的排序基础上，然后在对第二个字段进行排序。所以 b=2 这种查询条件没有办法利用索引。举个例子，我弄一个索引，

KEY `idx_time_zone` （`time_zone`，`time_string`）USING BTREE

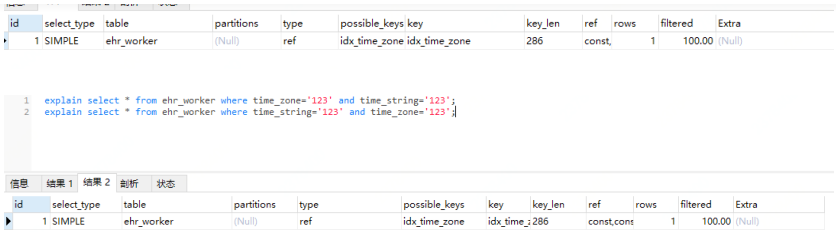
执行第一条 sql, 全表扫描



执行第二条 sql，可以看到使用了索引。



再看两条 sql，建立的索引是 KEY `idx_time_zone` （`time_zone`，`time_string`）USING BTREE



id	select_type	table	partitions	type	possible_keys key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ehr_worker	(Null)	ref	idx_time_zone idx_time_zone	286	const,	1	100.00	(Null)


```
explain select * from ehr_worker where time_zone='123' and time_string='123';
explain select * from ehr_worker where time_string='123' and time_zone='123';
```

信息 结果 1 结果 2 剖析 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ehr_worker	(Null)	ref	idx_time_zone	idx_time_z	286	const,cons	1	100.00	(Null)

按照正常逻辑来说，第二条 sql 是不符合索引字段的顺序的，应该不能使用索引才对，但是实际情况却和我们期望的不太一样，这是为啥呢？

从 mysql 被 oracle 收购以后，mysql 加入了很多 oracle 以前的技术，高版本 mysql 自动优化了 where 条件的先后顺序。简单来说就是查询优化器做了这一步操作，sql 会做预处理，那一条能更好的查询就会使用那种规则。

顺便提一下 mysql 的查询优化器能帮忙干的一些事

6.1 条件转化

例如 where a=b and b=2，可以得到 a=2, 条件传递。最后的 sql 是 a=2 and b=2 > < = like 都可以传递

6.2 无效代码的排除

例如 where 1=1 and a=2, 1=1 永远是正确的，所以最后会优化成 a=2
在比如 where 1=0 永远是 false 的，这样的也会被排除掉，整 sql 无效
或者非空字段 where a is null，这样的也会被排除

6.3 提前计算

包含数学运算的部分，例如 where a= 1+2 会帮你算好，where a=3

6.4 存取类型

当我们评估一个条件表达式，MySQL 判断该表达式的存取类型。下面是一些存取类型，按照从最优到最差的顺序进行排列：

- system 系统表，并且是常量表
- const 常量表
- eq_ref unique/primary 索引，并且使用的是'='进行存取
- ref 索引使用'='进行存取
- ref_or_null 索引使用'='进行存取，并且有可能为 NULL
- range 索引使用 BETWEEN、IN、>=、LIKE 等进行存取
- index 索引全扫描
- ALL 表全扫描

经常看执行计划的，一眼就能看出来这是啥意思，举个例子

where index_col=2 and normal_col =3 这里就会选用 index_col=2 会作为驱动项。驱动项的意思是指一个 sql 选定他的执行计划的时候，可能有多条执行路径，一个是全表扫描，再过滤是否符合索引字段及非索引字段的值。另一种是通过索引字段，键值 = 2 找到对应的索引树，过滤后的结果，再比较是否符合非索引字段的值。一般情况下，走索引都比全表扫描需要读取磁盘的次数少，所以称它为更好的执行路径，也就是通过索引字段，作为其驱动表达式

6.5 范围存取

6.6 索引存取类型

避免使用相同前缀的索引，也就是一个字段不要在多个索引上有相同的前缀。比如一个字段已经建立了唯一索引，这个时候如果再给他建立一个联合索引，会导致优化器并不知道你要使用哪个索引。或者你建了前缀相同的一个单索引，一个联合索引，就算你写上了条件，也不一定能用上联合索引。当然，可以 force，这就另说了。

6.7 转换

简单的表达式可以进行转换，比如 where -2 = a 会自动变成 where a= -2 ，但是如果牵扯到数学运算，就不能转换了 比如 where 2= -a 这时候不会自动转成 where a =-2.

```
explain select * from ehr_worker where 123 = -eos_status;
explain select * from ehr_worker where eos_status = -123;
```

信息 结果 1 结果 2 剖析 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ehr_worker	(Null)	ALL	(Null)	(Null)	(Null)	(Null)	83	100.00	Using where

第二条 sql 就可以使用索引

```
explain select * from ehr_worker where 123 = -eos_status;
explain select * from ehr_worker where eos_status = -123;
```

信息 结果 1 结果 2 剖析 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ehr_worker	(Null)	ref	idx_status	idx_status	5	const	1	100.00	(Null)

所以 我们在开发的过程中，需要注意 sql 的写法，自觉写成 where a=-2

6.8 and、union、order by、group by 等

1 ）and

and 条件后，如果都没索引，扫描全表。有一个存取类型更好，见 5.4，会使用存储类型更好的索引，如果都一样，哪个索引先创建，用哪个。

2 ）union

union 每条语句单独优化

```
explain select * from ehr_worker where biz_key ='2022-02-22,UNLOAD'
union all
select * from ehr_worker where  biz_key='2022-02-22,UNLOAD'
```

息 结果 1 剖析 状态

d	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	PRIMARY	ehr_worker	(Null)	ref	idx_type_key_v	idx_type_key_warehouse	514	const	1	100.00	(Null)
2	UNION	ehr_worker	(Null)	ref	idx_type_key_v	idx_type_key_warehouse	514	const	1	100.00	(Null)

这里就会分别执行两条 sql，用到索引，再合并结果集

3 ）order by

order by 会过滤无效排序，比如一个字段本来就有索引

operation_type
PUTAWAY
PUTAWAY
RECEIVING
RECEIVING
RECEIVING
RECEIVING
RECEIVING
RECEIVING
RECEIVING
UNLOAD
UNLOAD

第二条 sql 和第一条的查询效果是一样的

```
12
13  select operation_type from ehr_direct_manhour_record ;
14  select operation_type from ehr_direct_manhour_record order by operation_type
```

信息 结果 1 结果 2 剖析 状态

operation_type
PUTAWAY
PUTAWAY
RECEIVING
RECEIVING
RECEIVING
RECEIVING
RECEIVING
RECEIVING
RECEIVING
UNLOAD
UNLOAD

所以，写 sql 的时候，不要写无用排序，比如 order by 'xxx' 这样没有意义。

4) group by

简单来说 group by 的字段，有索引会走索引，group by a order by a 这里的 order by 等于没写，结果集已经是排序完毕的了，参考 6.8-3 order by
select distinct col_a from table a 等价于 select col_a from a group by col_a

7 索引下推

主要的核心点就在于把数据筛选的过程放在了存储引擎层去处理，而不是像之前一样放到 Server 层去做过滤。

如果在一张表上，name 和 age 都建立索引，查询条件为 where name like 'xx%' and age=11, 在低版本的 mysql (5.6 以下) 的根据索引的最左匹配原则，可以得到放弃了 age，只根据 name 过滤数据。根据 name 拿到所有的 id 之后，再根据 id 回表。

高版本 mysql 里，没有忽略 age 这个属性，带着 age 属性过滤，直接过滤掉了 age 为 11 的数据，假设不根据 age 过滤的数据为 10 条，过滤后只剩 3 条，就少了 7 次回表。减少了 io 会大量减少性能消耗

8 小表驱动大表

小表驱动大表，也是我们听惯了的话了，其含义主要是指小表的数据集驱动大表的数据集，减少连接次数。打个比方:

表 A 有 1w 数据，表 B 有 100w 数据，如果 A 表作为驱动表，处于循环的外层，那么只需要 1w 次的连接即可。如果 B 表在外层，那么则需要循环 100w 次。

下面我们实际测试看看，准备环境 mysql 5.7+

准备两张表，一张表 ib_asn_d 数据 9175， 一张表 bs_itembase_ext_attr 数据 1584115，都在商品编码字段上有索引。

首先小表驱动大表

```
1  select * from ib_asn_d d LEFT JOIN bs_itembase_ext_attr attr on d.item_code = attr.GOODS_NO
```

信息 结果 1 剖析 状态

```
select * from ib_asn_d d LEFT JOIN bs_itembase_ext_attr attr on d.item_code = attr.GOODS_NO
> OK
> 时间: 6.987s
```

多次反复测试，执行时间大概 7 秒。
接下来看看大表驱动小表。

```
1  select * from bs_itembase_ext_attr attr LEFT JOIN ib_asn_d d on attr.GOODS_NO=d.item_code
```

信息 结果 1 剖析 状态

```
select * from bs_itembase_ext_attr attr LEFT JOIN ib_asn_d d on attr.GOODS_NO=d.item_code
> OK
> 时间: 267.964s
```

将近 300 秒，不是一个量级的。
接下来分别分析执行计划，执行计划里第一条就是驱动表。

```
1  explain select * from ib_asn_d d LEFT JOIN bs_itembase_ext_attr attr on attr.GOODS_NO=d.item_code
2
```

信息 结果 1 剖析 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	d	(Null)	ALL	(Null)	(Null)	(Null)	(Null)	8224	100.00	(Null)
1	SIMPLE	attr	(Null)	ref	idx_sku_wareh	idx_sku_warehouse	93	lwms.i	1	100.00	Using where

小表驱动大表，大表用了索引，小表全表扫描，只扫描 8000 多行

```
1  explain select * from bs_itembase_ext_attr attr LEFT JOIN ib_asn_d d on attr.GOODS_NO=d.item_code
```

信息 结果 1 剖析 状态

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	attr	(Null)	ALL	(Null)	(Null)	(Null)	(Null)	1479609	100.00	(Null)
1	SIMPLE	d	(Null)	ref	idx_goods	idx_goods	153	lwms.i	18	100.00	Using where

大表驱动小表，大表全表扫描，需要扫描 147w 行。
经过多次测试得出了结论:

1. 当使用 left join 时，左表是驱动表，右表是被驱动表；
2. 当使用 right join 时，右表是驱动表，左表是被驱动表；
3. 当使用 inner join 时，mysql 会选择数据量比较小的表作为驱动表，大表作为被驱动表；
4. 驱动表索引不生效，非驱动表索引生效

保证小表是驱动表很重要。

9 总结

1. 覆盖索引：如果查询条件使用的是普通索引（或是联合索引的最左原则字段），查询结果是联合索引的字段或是主键，不用回表操作，直接返回结果，减少 IO 磁盘读写读取整行数据，所以高频字段建立联合索引是很有必要的

3. 索引下推：name like 'hello%' and age >10 检索，MySQL 5.6 版本之前，会对匹配的数据进行回表查询。5.6 版本后，会先过滤掉 age<10 的数据，再进行回表查询，减少回表率，提升检索速度

> 作者：京东物流 吴思维
> 来源：京东云开发者社区 转载请注明来源

CodeFuse-MFTCoder提升Qwen-14B代码能力
基于 ACK Fluid 的混合云优化数据访问（二）：搭建弹性计...
LiveVideoStack多媒体技术调研定量收集倒计时三天
突破混合精度训练大模型的局限性
从 13 个企业关心的问题看懂用云范式的改变
MySQL 核心模块揭秘 |《发刊词》
利用预训练模型优化大模型训练
“踩坑”经验分享：Swift语言落地实践
叮，你有一份来自 2023 开放原子开发者大会的邀请函，请...
COSCon'23 主论坛：年度大戏，演职人员已就位
“赋能信创，物联未来” AntDB数据库携高可用解决方案...

全站热门评论

gmg 2024-02-26 22:23
有点好奇为什么发布这种表面看起来吸引眼球的标题。

小而美软件开发 2024-03-05 12:25
你好歹介绍一下这软件的功能

xflcx1991 2024-03-05 13:02
有点意思，solon值得一学

D

dwcz 2024-02-18 22:12
中国法治的水平：二创可以演绎原创，三创不能演绎二创。

osc_25239240 2024-03-05 11:19
先免费在收割，妥妥的在钓鱼骗钱啊。

yl-yue 2024-02-18 13:49
操作也是相当炸裂，原来还可以这样玩，字太多了没读透，我的理解，就算是鸡肋专利也够他们吹嘘了，要是限制性专利，那是相当牛逼，开源行业要炸。

T

transtone 2024-03-05 11:42
可以把 rust 拉进来对比一下。

老盖 2024-02-01 14:11
windows没希望了，一群阿三，越做越差

酷酷的就 2024-03-05 11:12
要看你贡献源码时候,签署的协议, 类似APACHE组织就有很完善的权利要求, 但国内基本为0 .

小而美软件开发 2024-03-05 12:19
滚出去社区

Z生生不息 2024-01-31 18:33
没有人要求你必须要维护必须开源。但是你有权力要求用户“不得以任何形式传播及公开”吗？这是 AGPL 开源协议所允许的吗？如果你连开源协议都搞不清楚，早点把仓库归档吧。整得跟小学生耍流氓一样。😂😂😂

小而美软件开发 2024-03-05 12:28
老黄天天炒热度继续卖他的显卡

闲大赋 2024-02-18 20:49
我对此专利的解读『 https://my.oschina.net/xiandafu/blog/11043929』

M

MrChen89 2024-03-05 11:41
貌似native并不能提升多少性能，对启动速度和第一次初始化有些许帮助而已

划个船 2024-03-05 13:34
配置太拉胯，跑几个ide不得卡死

张先仲 2024-03-05 13:28
RHEL 好像比windows还便宜

CheckStyle 2024-02-21 18:45
关键是什么业务？10个人，2023年，一年，赚2000多万，泼天的富贵啊


小而美软件开发 2024-03-05 12:24
有用吗


魔力猫 2024-02-01 14:19
最近这些事件的主角，初心绝对不是什么为了创造啥，为的只是赚钱。开源只是赚钱手段，而且是觉得自己为了钱放弃很多的那种心态，心理先把自己当成了牺牲者，然后认为所有人都应该补偿自己。不拿钱补偿就是忘恩负义的白眼狼！


osc_94406955 2024-03-01 09:29
预计该问题会在 24 小时内彻底解决..... 今天3月1日了,bug神奇的消失

O
osc_91229770 2024-02-18 12:03
这也申请专利，这个不是正常crud，常规操作吗

K
kylexy 2024-02-26 10:38
大实话。。。。


魔力猫 2024-02-01 09:35
这篇说的好听，但你心里，公有制就是白嫖，你之前公告写着“ioGame21 在线文档依旧采用自愿付费模式策略。简单的说，我们提供了最新在线文档的白嫖方式，如果你打算跟进框架最新版本的，依旧可以选择白嫖在线文档。”，“综上，想继续白嫖的，请跟进最新版本。”不是吗？这是什么理念？我看更像精神分裂的理念，伪君子和真小人之间来回切换，嘴上都不一致。

T
tedx53 2024-02-27 09:42
高考状元的试卷给我抄，我也能轻松上清华


dantezhu 2024-02-28 11:20
那，这就叫专业。


小而美软件开发 2024-03-05 12:25
用uni啊


zhangjinsongok 2024-02-01 16:29
开源世界的孤胆英雄。


高排量低炭烧 2024-02-26 21:29
鸿蒙只是人家现学的，人家本来薪水就这么高，而不是新手培训完就值这个数


朋克 2024-02-28 11:36
这才是正常的盈利模式


大风起兮9527 2024-02-06 08:57
这事没啥好讨论的啊，代码如果开始的时候是开源，那么你有闭源的权利，但是要从声明之后的版本开始，不能回溯。文档也是如此，但是有一点，如果文档虽然没有声明开源，但是自己在公开场合发布过，你可以建议请求不要随意传播，但是不能强硬的禁止。关于白嫖，这个说法过分了，对于这些想要商业化的项目，开源伊始的初心，无非是面推广费、免费测试。各有所图，不要互相指责。


梅子酒好吃 2024-03-05 11:51
对内存，有很大帮助！


zoujiaqing 2024-03-05 13:04
8G内存，还没手机的大

U
unameuname 2024-02-19 14:02
第一次钓鱼，别TM给我鱼竿，我只要鱼。


小而美软件开发 2024-03-05 12:29
哈哈


天朝八阿哥 2024-02-29 10:32
虽然不懂，但表示很赞，比随便就冠以“国产”“自主研发”之类的让人舒心太多了

无法选中 2024-03-05 10:44

这没用native image吧

记得小蘋初见123 2024-02-29 16:53

百小僧，出列

梦踏溪渔丶 2024-03-05 11:28

一公开，背后消耗不就被算得明明白白了，投资和收益概率不就被分析清楚了，然后还咋忽悠投资人。

买房也用券 2024-01-31 14:25

这几天OSC还真热闹,看了半天,对于个人和小厂开源总结出一个规律: 用的人少了就用"开源,永久免费"诸如此类的卖吆喝(当然,真正能做到的人也是有的,这个不能一杆子都打死); 用的人多了就开始各种套路割韭菜,割不到了就说作者要生存,你们都是白嫖党,不尊重作者的劳动成果; 反观大厂的开源作品倒是活的好好的:Druid,fastjson,dubbo,Doris 总之就是国内个人和小厂开源的作品少用,慎用或不用,真心没必要给自己找麻烦;

无法选中 2024-03-05 11:55

比内存占用吖

小而美软件开发 2024-03-05 12:03

为什么每次点更新都是直接下载一个压缩包也没有安装界面

爱吃生梨 2024-01-31 15:35

百小僧果然是刷新中国开源底线里程碑式的人物，这不追随者来了。而且从.net扩散到java

xflcx1991 2024-03-05 13:03

之前还有一个抛弃了servlet的java框架 actframework

2

2cong 2024-02-26 11:21

如果让我抄，我就会！

红白机 2024-03-05 13:38

用它做了一个项目，结果现在停止更了。惨。。。

张先仲 2024-03-05 13:23

7 万亿, 去做深空探索, 实现资源自由吧, 之后你想做什么就能做什么

梅子酒好吃 2024-03-05 11:51

好。下次来个 java, go, rust 昆战！

osc_25239240 2024-03-05 11:07

只有把这个卓卓公司关闭了，才可以避免更多企业及人心受伤。

小肥侠 2024-02-19 17:51

所以将开源软件打包到应用商店，是真的有市场。

小而美软件开发 2024-03-05 12:20

不能举报吗

魔力猫 2024-02-01 09:48

开源不是不可以收费，基本上也没人认为开源项目里有收费项目是什么十恶不赦的罪过。问题在于，你要合理合法，要符合开源的道德。最近的事件，要么是某和尚绑票，不合理不合法更没道德，绑票拿赎金。要么就是这位，明明合法的事情，偏偏发个歧视公告，张嘴白嫖闭嘴白嫖，过嘴瘾有意思吗？开源生意，哪怕你心理一万个不愿意，但是既然你做了这个生意，伪君子人设好歹你不干的时候再扔呀，一会儿伪君子一会儿真小人，觉得不找骂才怪。

梅子酒好吃 2024-03-05 10:45

下次我用native image再测测看

monkey_cici 2024-02-26 11:39

开源系统还是要看民企的深度统信和华为欧拉...

C

cassan 2024-03-01 22:19

开源了，我们国内的公司又可以申请知识产权了

橘子皮皮 2024-03-05 11:10

已经不错啦，加油。

西迷岛主 2024-01-31 13:33

技术员的耻辱

Yoona520 2024-02-24 17:44

国外那个P站的技术水准可不低，毕竟服务全世界除CN之外的人