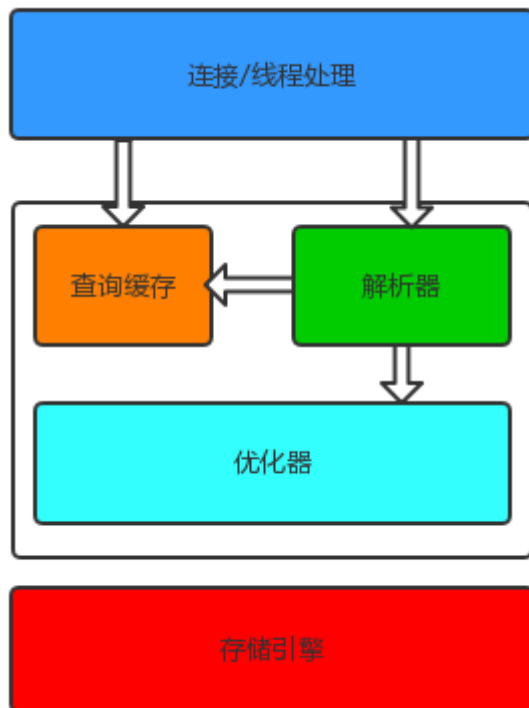


一、MySQL三层逻辑架构

MySQL的存储引擎架构将查询处理与数据的存储/提取相分离。下面是MySQL的逻辑架构图：



1、第一层负责连接管理、授权认证、安全等等。

每个客户端的连接都对应着服务器上的一个线程。服务器上维护了一个线程池，避免为每个连接都创建销毁一个线程。当客户端连接到MySQL服务器时，服务器对其进行认证。可以通过用户名和密码的方式进行认证，也可以通过SSL证书进行认证。登录认证通过后，服务器还会验证该客户端是否有执行某个查询的权限。

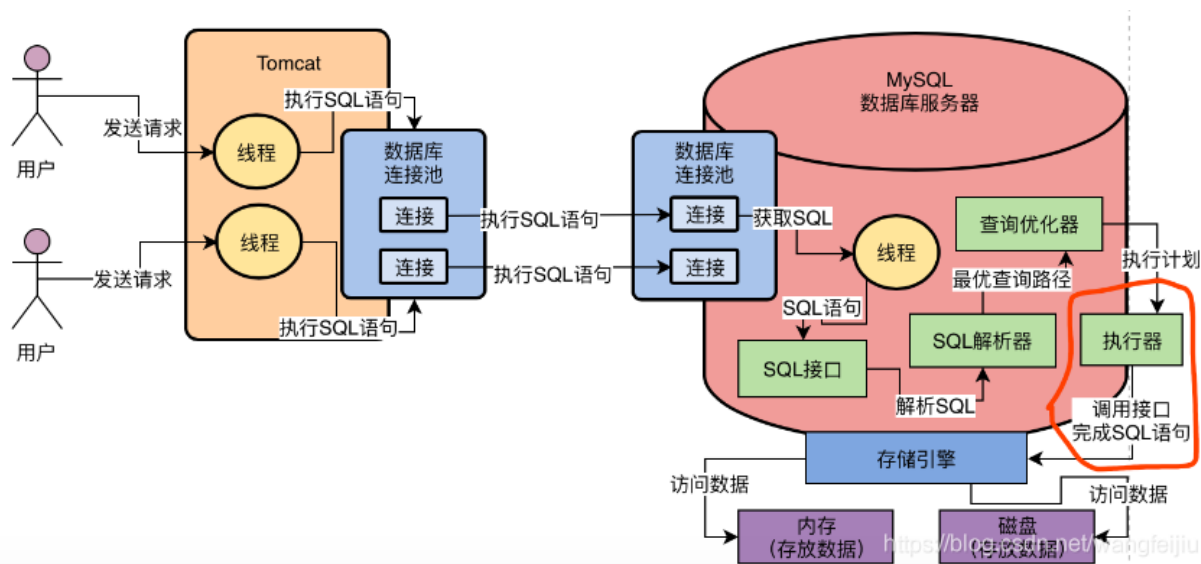
2、第二层负责解析查询

编译SQL，并对其进行优化(如调整表的读取顺序，选择合适的索引等)。对于SELECT语句，在解析查询前，服务器会先检查查询缓存，如果能在其中找到对应的查询结果，则无需再进行查询解析、优化等过程，直接返回查询结果。存储过程、触发器、视图等都在这一层实现。

3、第三层是存储引擎

存储引擎负责在MySQL中存储数据、提取数据、开启一个事务等等。存储引擎通过API与上层进行通信，这些API屏蔽了不同存储引擎之间的差异，使得这些差异对上层查询过程透明。存储引擎不会去解析SQL。

用户使用mysql查询的一个整体流程如下



4、事务支持

MyISAM: 强调的是性能, 每次查询具有原子性, 其执行速度比InnoDB类型更快, 但是不提供事务支持。

InnoDB: 提供事务支持事务, 外键等高级数据库功能。 具有事务(commit)、回滚(rollback)和崩溃修复能力(crash recovery capabilities)的事务安全(transaction-safe (ACID compliant))型表。

5、 AUTO_INCREMENT

MyISAM: 可以和其他字段一起建立联合索引。引擎的自动增长列必须是索引, 如果是组合索引, 自动增长可以不是第一列, 他可以根据前面几列进行排序后递增。

InnoDB: InnoDB中必须包含只有该字段的索引。引擎的自动增长列必须是索引, 如果是组合索引也必须是组合索引的第一列。

6、 表锁差异

MyISAM: 只支持表级锁, 用户在操作myisam表时, select, update, delete, insert语句都会给表自动加锁, 如果加锁以后的表满足insert并发的情况下, 可以在表的尾部插入新的数据。

InnoDB: 支持事务和行级锁, 是innodb的最大特色。行锁大幅度提高了多用户并发操作的新能。但是InnoDB的行锁, 只是在WHERE的主键是有效的, 非主键的WHERE都会锁全表的。

7、 全文索引

MyISAM: 支持 FULLTEXT类型的全文索引

InnoDB: 不支持FULLTEXT类型的全文索引, 但是innodb可以使用sphinx插件支持全文索引, 并且效果更好。

8、 表主键

MyISAM: 允许没有任何索引和主键的表存在, 索引都是保存行的地址。

InnoDB: 如果没有设定主键或者非空唯一索引, 就会自动生成一个6字节的主键(用户不可见), 数据是主索引的一部分, 附加索引保存的是主索引的值。

9、 表的具体行数

MyISAM: 保存有表的总行数, 如果`select count() from table;`会直接取出该值。

InnoDB: 没有保存表的总行数, 如果使用`select count(*) from table;`就会遍历整个表, 消耗相当大, 但是在加了where条件后, myisam和innodb处理的方式都一样。

10、CRUD操作

MyISAM: 如果执行大量的SELECT, MyISAM是更好的选择。

InnoDB: 如果你的数据执行大量的INSERT或UPDATE, 出于性能方面的考虑, 应该使用InnoDB表。

11、外键

MyISAM: 不支持

InnoDB: 支持

三、sql优化简介

1、什么情况下进行sql优化

性能低、执行时间太长、等待时间太长、连接查询、索引失效。

2、sql语句执行过程

(1) 编写过程

`select distinct ... from ... join ... on ... where ... group by ... having ... order by ... limit ...`

(2) 解析过程

`from ... on ... join ... where ... group by ... having ... select distinct ... order by ... limit ...`

3、sql优化就是优化索引

索引相当于书的目录。

索引的数据结构是B+树。

四、索引

(1) 提高查询效率 (降低IO使用率)

(2) 降低CPU使用率

比如查询order by age desc，因为B+索引树本身就是排好序的，所以再查询如果触发索引，就不用再重新查询了。

2、索引的弊端

- (1) 索引本身很大，可以存放在内存或硬盘上，通常存储在硬盘上。
- (2) 索引不是所有情况都使用，比如①少量数据②频繁变化的字段③很少使用的字段
- (3) 索引会降低增删改的效率

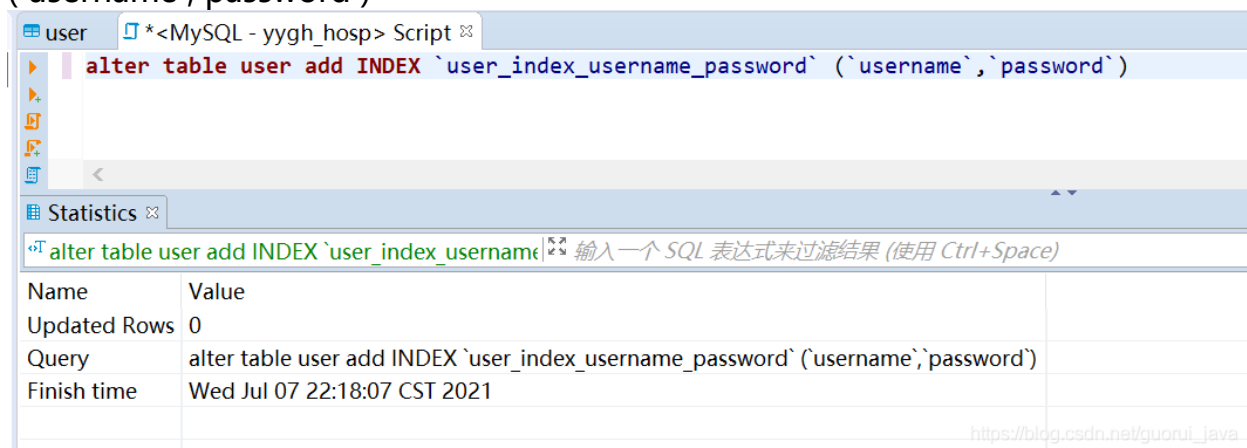
3、索引的分类

- (1) 单值索引
- (2) 唯一索引
- (3) 联合索引
- (4) 主键索引

备注：唯一索引和主键索引唯一的区别：主键索引不能为null

4、创建索引

```
alter table user add INDEX `user_index_username_password`  
(`username`,`password`)
```



5、MySQL索引原理 -> B+树

MySQL索引的底层数据结构是B+树

B+Tree是在B-Tree基础上的一种优化，使其更适合实现外存储索引结构，InnoDB存储引擎就是用B+Tree实现其索引结构。

B-Tree结构图中每个节点中不仅包含数据的key值，还有data值。而每一个页的存储空间是有限的，如果data数据较大时将会导致每个节点（即一个页）能存储的key的数量很小，当

存储的数据量很大时同样会导致B-Tree的深度较大，增大查询时的磁盘I/O次数，进而影响查询效率。在B+Tree中，所有数据记录节点都是按照键值大小顺序存放在同一层的叶子节点上，而非叶子节点上只存储key值信息，这样可以大大加大每个节点存储的key值数量，降低B+Tree的高度。

一般来说，索引本身也很大，不可能全部存储在内存中，因此索引往往以索引文件的形式存储的磁盘上。这样的话，索引查找过程中就要产生磁盘I/O消耗，相对于内存存取，I/O存取的消耗要高几个数量级，所以评价一个数据结构作为索引的优劣最重要的指标就是在查找过程中磁盘I/O操作次数的渐进复杂度。

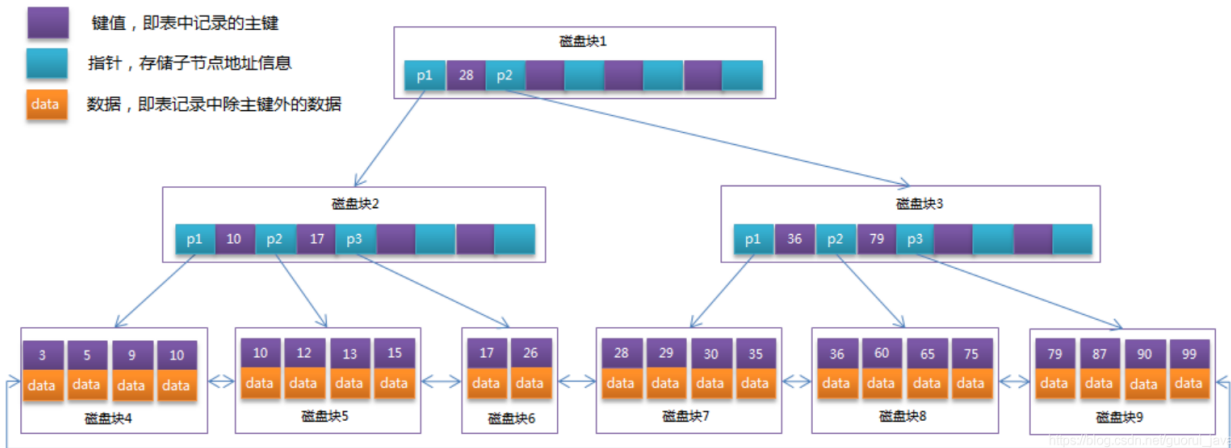
B+Tree相对于B-Tree有几点不同：

非叶子节点只存储键值信息。

所有叶子节点之间都有一个链指针。

数据记录都存放在叶子节点中。

将上一节中的B-Tree优化，由于B+Tree的非叶子节点只存储键值信息，假设每个磁盘块能存储4个键值及指针信息，则变成B+Tree后其结构如下图所示：



通常在B+Tree上有两个头指针，一个指向根节点，另一个指向关键字最小的叶子节点，而且所有叶子节点（即数据节点）之间是一种链式环结构。因此可以对B+Tree进行两种查找运算：一种是对主键的范围查找和分页查找，另一种是从根节点开始，进行随机查找。

可能上面例子中只有22条数据记录，看不出B+Tree的优点，下面做一个推算：

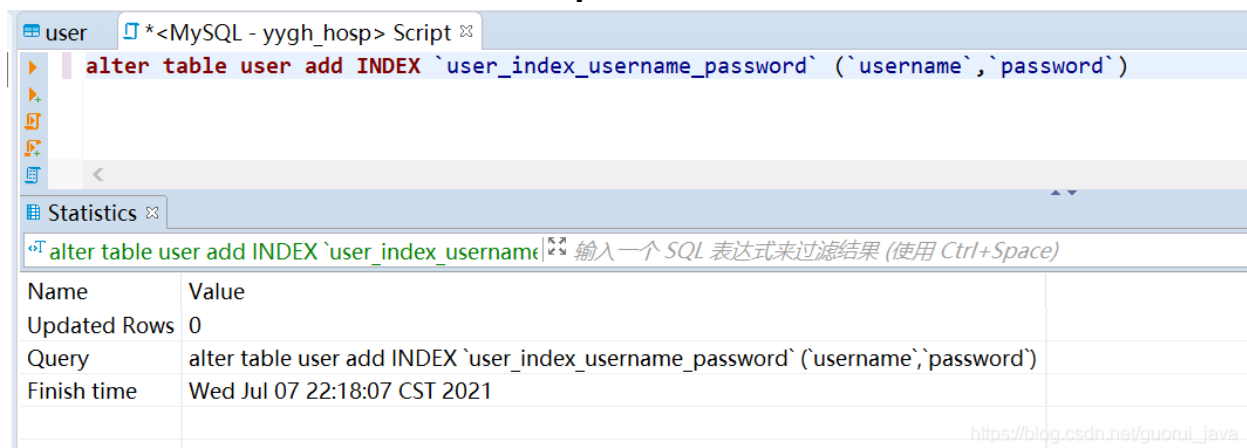
InnoDB存储引擎中页的大小为16KB，一般表的主键类型为INT（占用4个字节）或BIGINT（占用8个字节），指针类型也一般为4或8个字节，也就是说一个页（B+Tree中的一个节点）中大概存储 $16KB / (8B + 8B) = 1K$ 个键值（因为是估值，为方便计算，这里的K取值为 $\lfloor 10 \rfloor^3$ ）。也就是说一个深度为3的B+Tree索引可以维护 $10^3 * 10^3 * 10^3 = 10^9$ 条记录。

实际情况中每个节点可能不能填满，因此在数据库中，B+Tree的高度一般都在2~4层。MySQL的InnoDB存储引擎在设计时是将根节点常驻内存的，也就是说查找某一键值的行记录时最多只需要1~3次磁盘I/O操作。

数据库中的B+Tree索引可以分为聚集索引（clustered index）和辅助索引（secondary index）。上面的B+Tree示例图在数据库中的实现即为聚集索引，聚集索引的B+Tree中的叶子节点存放的是整张表的行记录数据。辅助索引与聚集索引的区别在于辅助索引的叶子节点并不包含行记录的全部数据，而是存储相应行数据的聚集索引键，即主键。当通过辅助索引来查询数据时，InnoDB存储引擎会遍历辅助索引找到主键，然后再通过主键在聚集索引中找到完整的行记录数据。

五、如何触发联合索引

1、对user表建立联合索引username、password




The screenshot shows the MySQL Workbench interface. The top pane contains the SQL command: `alter table user add INDEX `user_index_username_password` (`username`,`password`)`. The bottom pane shows the execution statistics for this command.

Name	Value
Updated Rows	0
Query	alter table user add INDEX `user_index_username_password` (`username`,`password`)
Finish time	Wed Jul 07 22:18:07 CST 2021

2、触发联合索引

（1）使用联合索引的全部索引键可触发联合索引



The screenshot shows the MySQL Workbench interface. The top pane contains the SQL query: `explain select id, username, age from user where username = 'zs' and password = '123456'`. The bottom pane shows the execution plan for this query.

触发联合索引

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	user	[NULL]	ref	user_index_username_password	user_index_username_password	424	const/co	1	100	[NULL]

（2）使用联合索引的全部索引键，但是用or连接的，不可触发联合索引

user *MySQL - yygh_hosp> Script *MySQL - yygh_hosp> Script-1 department id id

```

explain
select
  id,
  username,
  age
from
  user
where
  username = 'zs' or password = '123456'

```

当使用or关联时，不触发索引

Result

explain select id, username, age from user where username = 'zs' or password = '123456' 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	1	SIMPLE	user	[NULL]	ALL	user_index_username_password	[NULL]	[NULL]	3	55.56	Using where

(3) 单独使用联合索引的左边第一个字段时，可触发联合索引

user *MySQL - yygh_hosp> Script *MySQL - yygh_hosp> Script-1 department id id

```

explain
select
  id,
  username,
  age
from
  user
where
  username = 'zs'

```

单独使用联合索引的左边第一个字段，可触发联合索引

Result

explain select id, username, age from user where username = 'zs' 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	1	SIMPLE	user	[NULL]	ref	user_index_username_password	302	const	1	100	Using where

(4) 单独使用联合索引的其它字段时，不可触发联合索引

user *MySQL - yygh_hosp> Script *MySQL - yygh_hosp> Script-1 department id id

```

explain
select
  id,
  username,
  age
from
  user
where
  password = '123456'

```

单独使用非左边第一个字段时，不可触发索引

Result

explain select id, username, age from user where password = '123456' 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	1	SIMPLE	user	[NULL]	ALL	[NULL]	[NULL]	[NULL]	3	33.33	Using where

六、分析sql的执行计划---explain

explain可以模拟sql优化，执行sql语句。explain显示了mysql如何使用索引来处理select语句以及连接表。可以帮助选择更好的索引和写出更优化的查询语句。

1、explan使用简介

(1) 用户表

user

select * from 'user' 输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

id	username	password	age	sex	telephone	address	create_date	update_date	deleted	version	dept
1	zs	123456	18	0	10086	北京市	2021-07-07	2021-07-07	0	1	1
2	ls	7823456	20	0	110	上海市	2021-07-07	2021-07-07	0	1	1
3	ww	root	30	1	10010	大连市	2021-07-07	2021-07-07	0	1	2

(2) 部门表

department				
select * from department 输入一个 SQL 表达式				
	id	name	comment	
1	1	开发部	负责开发工作	
2	2	财务部	负责财务工作	

(3) 未触发索引

<

(4) 触发索引

```
explain select
  user.id,
  user.username,
  department.name
from
  user,
  department
where
  department.id = 1
order by
  department.id desc
```

order by 驱动表的字段，会触发索引

Result

explain select user.id, user.username, department.name

输入一个 SQL 表达式来过滤结果 (使用 Ctrl+Space)

	id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	1	SIMPLE	department	[NULL]	const	PRIMARY	PRIMARY	4	const	1	100	[NULL]
2	1	SIMPLE	user	[NULL]	index	[NULL]	user_index	424	[NULL]	3	100	Using index

<https://www.guorui-jia.com>

(5) 结果分析

explain中第一行出现的表是驱动表。

- 指定了联接条件时，满足查询条件的记录行数少的表为[驱动表]
- 未指定联接条件时，行数少的表为[驱动表]

对驱动表直接进行排序就会触发索引，对非驱动表进行排序不会触发索引。

2、explain查询结果简介

(1) id: SELECT识别符。这是SELECT的查询序列号。

(2) select_type: SELECT类型:

1. SIMPLE: 简单SELECT(不使用UNION或子查询)
2. PRIMARY: 最外面的SELECT
3. UNION: UNION中的第二个或后面的SELECT语句
4. DEPENDENT UNION: UNION中的第二个或后面的SELECT语句, 取决于外面的查询
5. UNION RESULT: UNION的结果
6. SUBQUERY: 子查询中的第一个SELECT
7. DEPENDENT SUBQUERY: 子查询中的第一个SELECT, 取决于外面的查询
8. DERIVED: 导出表的SELECT(FROM子句的子查询)

(3) table: 表名

(4) type: 联接类型

1. system: 表仅有一行(=系统表)。这是const联接类型的一个特例。
2. const: 表最多有一个匹配行, 它将在查询开始时被读取。因为仅有一行, 在这行的列值可被优化器剩余部分认为是常数。const用于用常数值比较PRIMARY KEY或UNIQUE索引的所有部分时。
3. eq_ref: 对于每个来自于前面的表的行组合, 从该表中读取一行。这可能是最好的联接类型, 除了const类型。它用于在一个索引的所有部分被联接使用并且索引是UNIQUE或PRIMARY KEY。eq_ref可以用于使用= 操作符比较的带索引的列。比较值可以为常量或一个使用在该表前面所读取的表的列的表达式。
4. ref: 对于每个来自于前面的表的行组合, 所有有匹配索引值的行将从这张表中读取。如果联接只使用键的最左边的前缀, 或如果键不是UNIQUE或PRIMARY KEY(换句话说, 如果联接不能基于关键字选择单个行的话), 则使用ref。如果使用的键仅仅匹配少量行, 该联接类型是不错的。ref可以用于使用=或<=>操作符的带索引的列。
5. ref_or_null: 该联接类型如同ref, 但是添加了MySQL可以专门搜索包含NULL值的行。在解决子查询中经常使用该联接类型的优化。
6. index_merge: 该联接类型表示使用了索引合并优化方法。在这种情况下, key列包含了使用的索引的清单, key_len包含了使用的索引的最长的关键元素。
7. unique_subquery: 该类型替换了下面形式的IN子查询的ref: value IN (SELECT primary_key FROM single_table WHERE some_expr); unique_subquery是一个索引查找函数, 可以完全替换子查询, 效率更高。

8. `index_subquery`: 该联接类型类似于`unique_subquery`。可以替换IN子查询, 但只适合下列形式的子查询中的非唯一索引: `value IN (SELECT key_column FROM single_table WHERE some_expr)`

9. `range`: 只检索给定范围的行, 使用一个索引来选择行。`key`列显示使用了哪个索引。`key_len`包含所使用索引的最长关键元素。在该类型中`ref`列为NULL。当使用`=`、`<>`、`>`、`>=`、`<`、`<=`、`IS NULL`、`<=>`、`BETWEEN`或者IN操作符, 用常量比较关键字列时, 可以使用`range`

10. `index`: 该联接类型与ALL相同, 除了只有索引树被扫描。这通常比ALL快, 因为索引文件通常比数据文件小。

11. `all`: 对于每个来自于先前的表的行组合, 进行完整的表扫描。如果表是第一个没标记`const`的表, 这通常不好, 并且通常在它情况下很差。通常可以增加更多的索引而不要使用ALL, 使得行能基于前面的表中的常数值或列值被检索出。

(5) `possible_keys`: `possible_keys`列指出MySQL能使用哪个索引在该表中找到行。注意, 该列完全独立于EXPLAIN输出所示的表的次序。这意味着在`possible_keys`中的某些键实际上不能按生成的表次序使用。

(6) `key`: `key`列显示MySQL实际决定使用的键(索引)。如果没有选择索引, 键是NULL。要想强制MySQL使用或忽视`possible_keys`列中的索引, 在查询中使用`FORCE INDEX`、`USE INDEX`或者`IGNORE INDEX`。

(7) `key_len`: `key_len`列显示MySQL决定使用的键长度。如果键是NULL, 则长度为NULL。注意通过`key_len`值我们可以确定MySQL将实际使用一个多部关键字的几个部分。

(8) `ref`: `ref`列显示使用哪个列或常数与`key`一起从表中选择行。

(9) `rows`: `rows`列显示MySQL认为它执行查询时必须检查的行数。

(10) `Extra`: 该列包含MySQL解决查询的详细信息。

1. `Distinct`: MySQL发现第1个匹配行后, 停止为当前的行组合搜索更多的行。

2. `Not exists`: MySQL能够对查询进行LEFT JOIN优化, 发现1个匹配LEFT JOIN标准的行后, 不再为前面的的行组合在该表内检查更多的行。

3. `range checked for each record (index map: #)`: MySQL没有发现好的可以使用的索引, 但发现如果来自前面的表的列值已知, 可能部分索引可以使用。对前面的

表的每个行组合，MySQL检查是否可以使用range或index_merge访问方法来索取行。

4. Using filesort: MySQL需要额外的一次传递，以找出如何按排序顺序检索行。通过根据联接类型浏览所有行并为所有匹配WHERE子句的行保存排序关键字和行的指针来完成排序。然后关键字被排序，并按排序顺序检索行。

5. Using index: 从只使用索引树中的信息而不需要进一步搜索读取实际的行来检索表中的列信息。当查询只使用作为单一索引一部分的列时，可以使用该策略。

6. Using temporary: 为了解决查询，MySQL需要创建一个临时表来容纳结果。典型情况如查询包含可以按不同情况列出列的GROUP BY和ORDER BY子句时。

7. Using where: WHERE子句用于限制哪一个行匹配下一个表或发送到客户。除非你专门从表中索取或检查所有行，如果Extra值不为Using where并且表联接类型为ALL或index，查询可能会有一些错误。

8. Using sort_union(...), Using union(...), Using intersect(...): 这些函数说明如何为index_merge联接类型合并索引扫描。

9. Using index for group-by: 类似于访问表的Using index方式，Using index for group-by表示MySQL发现了一个索引，可以用来查询GROUP BY或DISTINCT查询的所有列，而不要额外搜索硬盘访问实际的表。并且，按最有效的方式使用索引，以便对于每个组，只读取少量索引条目。

通过相乘EXPLAIN输出的rows列的所有值，你能得到一个关于一个联接如何的提示。这应该粗略地告诉你MySQL必须检查多少行以执行查询。当你使用max_join_size变量限制查询时，也用这个乘积来确定执行哪个多表SELECT语句。