目录 /CONTENTS

01

执行计划

EXPLAN

复习一下索引

列无重复值，可以建索引：唯一索引和普通索引
聚集索引和非聚集索引都可以是唯一的。因此，只要列中的数据是唯一的，就可以在同一个表上创建一个唯一的聚集索引和多个唯一的非聚集索引。

建了索引性能得到提高！

需要说明：
唯一索引一定要小心，它带有唯一约束。

需要说明2：
查询区分度
SELECT COUNT(DISTINCT 列_xx)/COUNT(*) FROM 表

大神说过：一天的代码量不宜超过300行。

复习一下索引

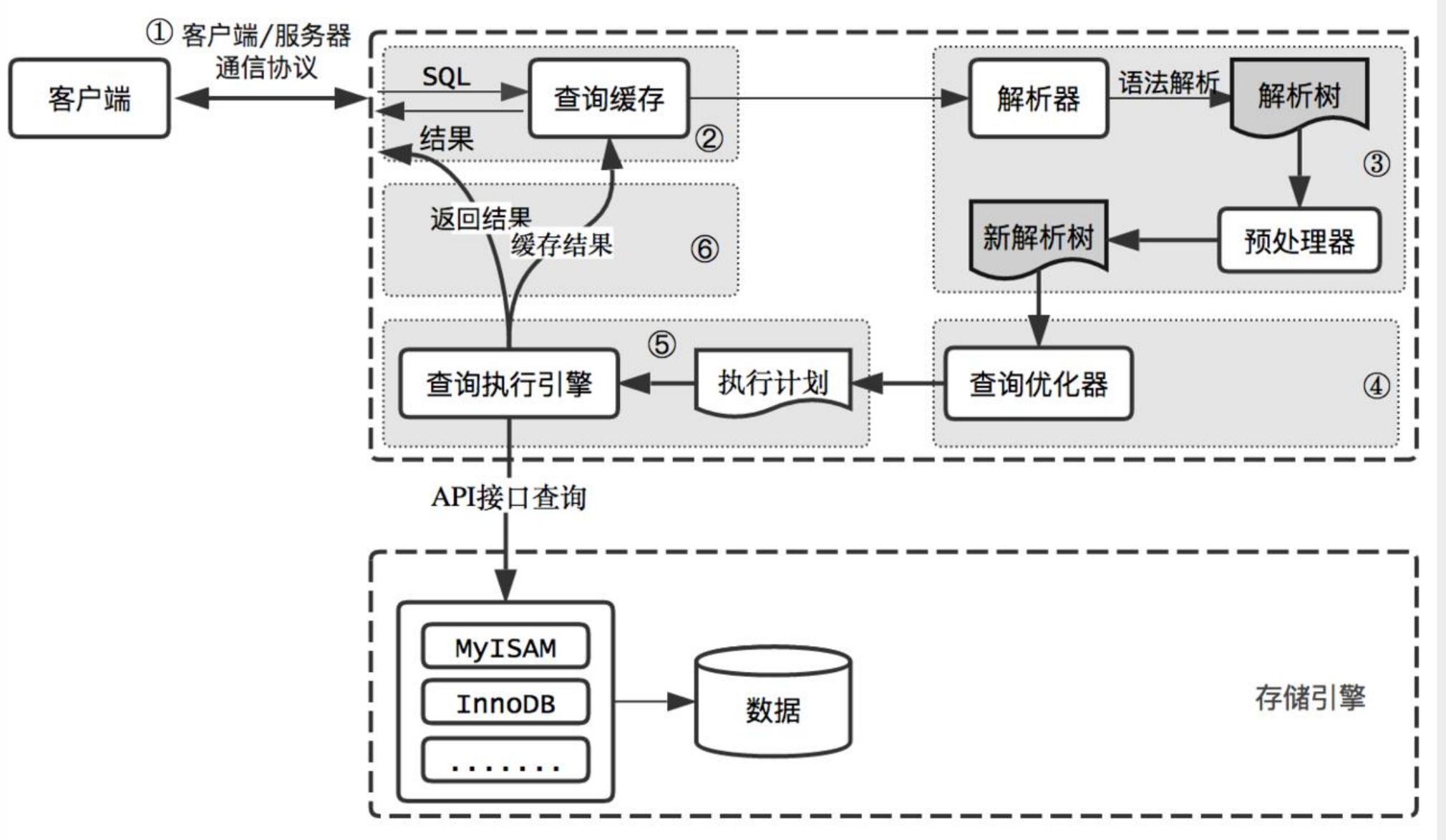
存储引擎及文件格式比较。

	Innodb	Myisam
存储文件	.frm 表定义文件 .ibd 数据文件	.frm 表定义文件 .myd 数据文件 .myi 索引文件
锁	表锁、行锁	表锁
事务	ACID	不支持
CRDU	读、写	读多
count	扫表	专门存储的地方
索引结构	B+ Tree	B+ Tree

复习一下索引

特点	Myisam	NDB	Memory	InnoDB	XtraDB
存储限制	没有	没有	有	64TB	64TB
事务安全		支持		支持	支持
锁机制	表锁	页锁	表锁	行锁	行锁
B树索引	支持	支持	支持	支持	支持
哈希索引			支持	支持	支持
全文索引	支持				
集群索引				支持	支持
数据缓存			支持	支持	支持
索引缓存	支持		支持	支持	支持
数据可压缩	支持				
空间使用	低	低	N/A	高	高
内存使用	低	低	中等	高	高
批量插入的速度	高	高	高	低	低
支持外键				支持	支持

复习一下索引


① 客户端/服务器
通信协议
客户端
SQL
结果
查询缓存
②
解析器
语法解析
解析树
③
返回结果
缓存结果
⑥
新解析树
预处理器
⑤
查询执行引擎
执行计划
查询优化器
④
API接口查询
MyISAM
InnoDB
.....
数据
存储引擎

复习一下索引

建索引的目的

加快查询速度，当然了，使用索引后查询有迹可循。

减少I/O操作，通过索引的路径来检索数据，不是在磁盘中随机检索。

消除磁盘排序，索引是排序的，走完索引就排序完成

复习一下索引

何时使用索引

以下是不需要的情况
查询返回的记录数
排序表<40%
非排序表 <7%
表的碎片较多（频繁增加、删除）

基础表维护时，系统要同时维护索引，不合理的索引将严重影响系统资源，主要表现在CPU和I/O上；

插入、更新、删除数据产生大量db file sequential read锁等待；

复习一下索引

1.频繁更新的字段不适合建立索引

2.where条件中用不到的字段不适合建立索引

3.表数据可以确定比较少的不需要建索引

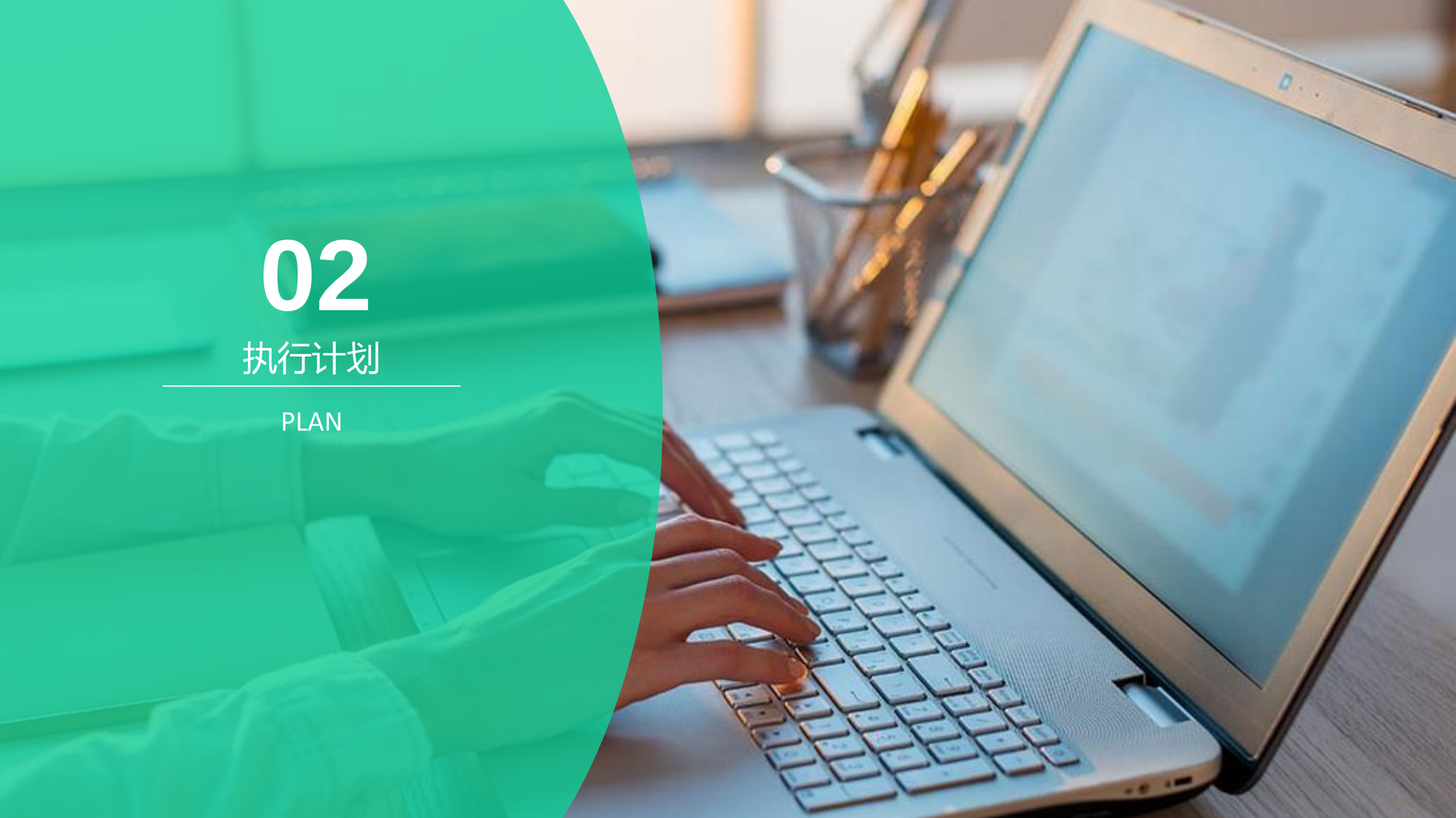
4.数据重复且发布比较均匀的的字段不适合建索引（唯一性太差的字段不适合建立索引），例如性别，真假值

5. 参与列计算的列不适合建索引

02

执行计划

PLAN



预备知识-事务

ACID

•原子性：一个事务（transaction）中的所有操作，要么全部完成，要么全部不完成，不会结束在中间某个环节。事务在执行过程中发生错误，会被恢复（Rollback）到事务开始前的状态，就像这个事务从来没有执行过一样。

•一致性：在事务开始之前和事务结束以后，数据库的完整性没有被破坏。这表示写入的资料必须完全符合所有的预设规则，这包含资料的精确度、串联性以及后续数据库可以自发性地完成预定的工作。

•隔离性：数据库允许多个并发事务同时对其数据进行读写和修改的能力，隔离性可以防止多个事务并发执行时由于交叉执行而导致数据的不一致。事务隔离分为不同级别，包括读未提交（Read uncommitted）、读提交（read committed）、可重复读（repeatable read）和串行化（Serializable）。

•持久性：事务处理结束后，对数据的修改就是永久的，即便系统故障也不会丢失。

预备知识-锁

1.按照对数据操作的类型分

读锁：也称为共享锁，针对同一资源，多个读操作是可以并行进行的，并且互不影响。

写锁：也称排它锁。 当前线程写数据的时候，会阻断其他线程来读数据和写数据

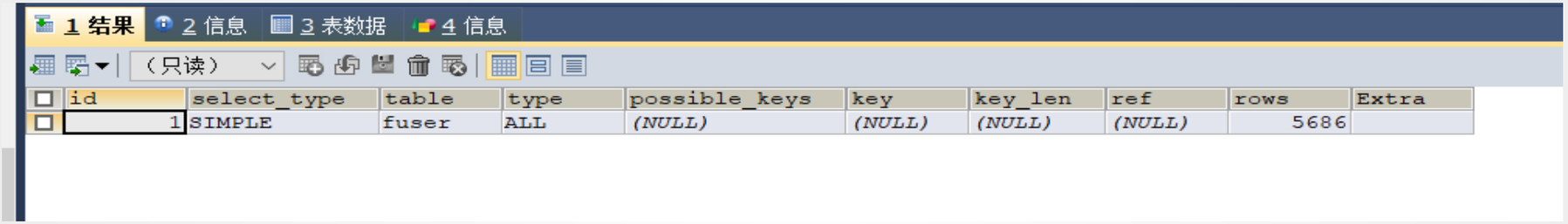
2.按照 粒度来分

表锁：就是锁整个表（myisam）

行锁：就是锁单独某个表中的某一行(innodb)

叶锁：他是鉴于表锁和行数之间的一种粒度

Explain



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	fuser	ALL	(NULL)	(NULL)	(NULL)	(NULL)	5686	

1.Id，SQL执行的顺利的标识,SQL从大到小的执行.
2. select_type，就是select类型,可以有以下几种
3.Table，显示这一行的数据是关于哪张表的.
4.Type，这列很重要,显示了连接使用了哪种类别,有无使用索引.从最好到最差的连接类型为const、eq_reg、ref、range、indexhe和ALL
5.possible_keys，possible_keys列指出MySQL能使用哪个索引在该表中找到行。
6. Key，key列显示MySQL实际决定使用的键（索引）。
7.key_len，使用的索引的长度。在不损失精确性的情况下，长度越短越好
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,下面详细.

Explain

select_type
1、SIMPLE，简单查询
2、PRIMARY，主查询（多个表关联时）
3、UNION，联合查询
4、DEPENDENT UNION，子查询中的联合查询
5、UNION RESULT，联合的结果集
6、SUBQUERY，第一个子查询
7、 DEPENDENT SUBQUERY，子查询中第一句
8、DERIVED，派生表

type
1、system, const联接类型的一个特例。表仅有一行满足条件
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表很快，因为它们只读取一次！
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
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。
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。

Explain

Extra，这个列可以显示的信息非常多，有几十种。常用如下：
(1).Distinct, 一旦MYSQL找到了与行相联合匹配的行，就不再搜索了
(2).Not exists ， 使用了反连接，先查询外表，再查询内表
(3).Range checked for each Record（index map:#）
没有找到理想的索引，因此对于从前面表中来的每一个行组合，MYSQL检查使用哪个索引，并用它来从表中返回行。这是使用索引的最慢的连接之一
(4).Using filesort
看到这个的时候，查询需要优化。MYSQL需要进行额外的步骤来发现如何对返回的行排序。它根据连接类型以及存储排序键值和匹配条件的全部行的行指针来排序全部行
(5).Using index
列数据是从仅仅使用了索引中的信息而没有读取实际的行动的表返回的，这发生在对表的全部的请求列都是同一个索引的部分的时候
(6).Using temporary
看到这个的时候，查询需要优化。这里，MYSQL需要创建一个临时表来存储结果，这通常发生在对不同的列集进行ORDER BY上，而不是GROUP BY上
(7).Using where
使用了WHERE从句来限制哪些行将与下一张表匹配或者是返回给用户。如果不想返回表中的全部行，并且连接类型ALL或index，这就会发生，或者是查询有问题
(8).firstmatch(tb_name)：5.6.x开始引入的优化子查询的新特性之一，常见于where字句含有in()类型的子查询。如果内表的数据量比较大，就可能出现这个.
(9).loosescan(m..n)：5.6.x之后引入的优化子查询的新特性之一，在in()类型的子查询中，子查询返回的可能有重复记录时，就可能出现这个

Explain

MySQL执行计划的局限

- EXPLAIN不会告诉你关于触发器、存储过程的信息或用户自定义函数对查询的影响情况
- EXPLAIN不考虑各种Cache
- EXPLAIN不能显示MySQL在执行查询时所作的优化工作
- 部分统计信息是估算的，并非精确值
- EXPALIN只能解释SELECT操作，其他操作要重写为SELECT后查看执行计划

03

几个场景

SCOPE

分页特殊处理

```
Select * from fentrust e
Inner join (select fid from fentrust limit 4100000, 10) a on a.fid = e.fid
```

```
Select * from fentrust e limit 4100000, 10
```

```
Select * from fentrust e where fid in(select fid from (select fid from fentrust limit 4100000, 10) a )
```

原理

索引覆盖最快

善用子查询

```
SELECT wu.fuid, wu.fwid, v.fvi_fid,
v.fvi2_fid, SUM(l.fcount), SUM(l.famount),
SUM(l.famount / v.famount * v.ffees),
v.fentrustType, 0, 2018-01-01', NOW(),
SUM(v.fleftCount), SUM(v.fleftfees), 0
FROM fentrustlog_vcoin
INNER JOIN fentrust_vcoin v ON l.fen_fid =
v.fid
INNER JOIN fwebsite_user wu ON wu.fuid =
v.fus_fid
WHERE l.fid NOT IN(SELECT l2.fid FROM
fentrustlog_vcoin l2, fentrust_vcoin v2
WHERE l2.fen_fid = v2.fid AND l2.fprize =
l.fprize AND l2.fcount = l.fcount AND
l2.fcreateTime = l.fcreateTime
AND l2.fid <> l.fid AND v2.fus_fid =
v.fus_fid and wu.fwid=1)
GROUP BY wu.fuid, wu.fwid, v.fvi_fid,
v.fvi2_fid, v.fentrustType
```

```
SELECT v.fus_fid, 1, v.fvi_fid,
v.fvi2_fid, SUM(v.fcount - v.fleftCount),
SUM(v.fsuccessAmount),SUM(v.ffees -
v.fleftfees), v.fentrustType, 0, '2018-
01-01', NOW(), SUM(v.fleftCount),
SUM(v.fleftfees), 0
FROM fentrust_vcoin v
where v.fstatus > 1 and v.FUs_fId in
(select fuid from fwebsite_user wu where
wu.fwid = 1)
and v.fVi2_fId in (select fvid from
fwebsite_coin where fwebsite_id = 1)
GROUP BY v.fentrustType, v.fvi_fid,
v.fvi2_fid, v.fus_fid
```

原理

子查询比join快，虽然规律不绝对，但对大表多数有效

Where条件顺序

```
SELECT * FROM `fentrustlog` e WHERE e.fcount > 1000 and e.famount > 300000
```

e.fcount > 1000：48万行

e.famount > 300000： 24行

谁先谁后？

结果是不太有效

大事务问题

尽量避免大事务操作，提高系统并发能力。

有时无法避免，改用定时器延迟处理

不走索引的情况

SELECT ` famount ` FROM ` fentrust ` WHERE ` famount `+10=30;-- 不会使用索引,因为所有索引列参与了计算

SELECT `famount` FROM `fentrust` WHERE LEFT(`fcreateTime`,4) <1990; -- 不会使用索引,因为使用了函数运算,原理与上面相同

SELECT * FROM ` fuser` WHERE `floginname` LIKE ‘138%' -- 走索引

SELECT * FROM ` fuser ` WHERE ` floginname ` LIKE "%7488%" -- 不走索引 -- 正则表达式不使用索引,这应该很好理解,所以为什么在SQL中很难看到regexp关键字的原因 -- 字符串与数字比较不使用索引;

EXPLAIN SELECT * FROM `a` WHERE `a`=1 -- 不走索引

select * from fuser where floginname='xxx' or femail='xx' or fstatus=1 --如果条件中有or,即使其中有条件带索引也不会使用。换言之,就是要求使用的所有字段,都必须建立索引, 我们建议大家尽量避免使用or 关键字

如果mysql估计使用全表扫描要比使用索引快,则不使用索引

04

千万级数据表优化

BIG DATA

大数据表优化

1.对查询进行优化，应尽量避免全表扫描，首先应考虑在 where 及 order by 涉及的列上建立索引。

2.应尽量避免在 where 子句中对字段进行 null 值判断，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num is null可以在num上设置默认值0，确保表中num列没有null值，然后这样查询：select id from t where num=0

3.应尽量避免在 where 子句中使用!=或<>操作符，否则引擎将放弃使用索引而进行全表扫描。

4.应尽量避免在 where 子句中使用or 来连接条件，否则将导致引擎放弃使用索引而进行全表扫描，如：select id from t where num=10 or num=20可以这样查询：select id from t where num=10 union all select id from t where num=20

5.in 和 not in 也要慎用，否则会导致全表扫描，如：select id from t where num in(1,2,3) 对于连续的数值，能用 between 就不要用 in 了：select id from t where num between 1 and 3

6.下面的查询也将导致全表扫描：select id from t where name like '李%'若要提高效率，可以考虑全文检索。

7. 如果在 where 子句中使用参数，也会导致全表扫描。因为SQL只有在运行时才会解析局部变量，但优化程序不能将访问计划的选择推迟到运行时；它必须在编译时进行选择。然 而，如果在编译时建立访问计划，变量的值还是未知的，因而无法作为索引选择的输入项。如下面语句将进行全表扫描：select id from t where num=@num可以改为强制查询使用索引：select id from t with(index(索引名)) where num=@num

8.应尽量避免在 where 子句中对字段进行表达式操作，这将导致引擎放弃使用索引而进行全表扫描。如：select id from t where num/2=100应改为:select id from t where num=100*2

大数据表优化

9.应尽量避免在where子句中对字段进行函数操作，这将导致引擎放弃使用索引而进行全表扫描。如：select id from t where substring(name,1,3)='abc' ， name以abc开头的id
应改为:
select id from t where name like 'abc%'

10.不要在 where 子句中的“=”左边进行函数、算术运算或其他表达式运算，否则系统将可能无法正确使用索引。

11.在使用索引字段作为条件时，如果该索引是复合索引，那么必须使用到该索引中的第一个字段作为条件时才能保证系统使用该索引，否则该索引将不会被使用，并且应尽可能的让字段顺序与索引顺序相一致。

12.不要写一些没有意义的查询，如需要生成一个空表结构：select col1,col2 into #t from t where 1=0
这类代码不会返回任何结果集，但是会消耗系统资源的，应改成这样：
create table #t(...)

13.很多时候用 exists 代替 in 是一个好的选择：select num from a where num in(select num from b)
用下面的语句替换：
select num from a where exists(select 1 from b where num=a.num)

14.并不是所有索引对查询都有效，SQL是根据表中数据来进行查询优化的，当索引列有大量数据重复时，SQL查询可能不会去利用索引，如一表中有字段sex，male、female几乎各一半，那么即使在sex上建了索引也对查询效率起不了作用。

15. 索引并不是越多越好，索引固然可 以提高相应的 select 的效率，但同时也降低了 insert 及 update 的效率，因为 insert 或 update 时有可能会重建索引，所以怎样建索引需要慎重考虑，视具体情况而定。一个表的索引数最好不要超过6个，若太多则应考虑一些不常使用到的列上建的索引是否有 必要。

大数据表优化

16. 应尽可能的避免更新 clustered 索引数据列，因为 clustered 索引数据列的顺序就是表记录的物理存储顺序，一旦该列值改变将导致整个表记录的顺序的调整，会耗费相当大的资源。若应用系统需要频繁更新 clustered 索引数据列，那么需要考虑是否应将该索引建为 clustered 索引。

17.尽量使用数字型字段，若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

18.尽可能的使用 varchar/nvarchar 代替 char/nchar ，因为首先变长字段存储空间小，可以节省存储空间，其次对于查询来说，在一个相对较小的字段内搜索效率显然要高些。

19.任何地方都不要使用 select * from t ，用具体的字段列表代替“*”，不要返回用不到的任何字段。

20.尽量使用表变量来代替临时表。如果表变量包含大量数据，请注意索引非常有限（只有主键索引）。

21.避免频繁创建和删除临时表，以减少系统表资源的消耗。

22.临时表并不是不可使用，适当地使用它们可以使某些例程更有效，例如，当需要重复引用大型表或常用表中的某个数据集时。但是，对于一次性事件，最好使用导出表。

大数据表优化

23.在新建临时表时，如果一次性插入数据量很大，那么可以使用 select into 代替 create table，避免造成大量 log ，以提高速度；如果数据量不大，为了缓和系统表的资源，应先create table，然后insert。

24.如果使用到了临时表，在存储过程的最后务必将所有的临时表显式删除，先 truncate table ，然后 drop table ，这样可以避免系统表的较长时间锁定。

25.尽量避免使用游标，因为游标的效率较差，如果游标操作的数据超过1万行，那么就应该考虑改写。

26.使用基于游标的方法或临时表方法之前，应先寻找基于集的解决方案来解决问题，基于集的方法通常更有效。

27. 与临时表一样，游标并不是不可使 用。对小型数据集使用 FAST_FORWARD 游标通常要优于其他逐行处理方法，尤其是在必须引用几个表才能获得所需的数据时。在结果集中包括“合计”的例程通常要比使用游标执行的速度快。如果开发时 间允许，基于游标的方法和基于集的方法都可以尝试一下，看哪一种方法的效果更好。

28.在所有的存储过程和触发器的开始处设置 SET NOCOUNT ON ，在结束时设置 SET NOCOUNT OFF 。无需在执行存储过程和触发器的每个语句后向客户端发送DONE_IN_PROC 消息。

29.尽量避免大事务操作，提高系统并发能力。

30.尽量避免向客户端返回大数据量，若数据量过大，应该考虑相应需求是否合理。

批量删除，而不一次性

```
while(true){

        //每次只做1000条
        “delete from logs where log_date <= ’ 2012-11-01’ limit 1000”;
        if(mysql_affected_rows == 0){

                //删除完成，退出！
                break；
       }

      //每次暂停一段时间，释放表让其他进程/线程访问。
      Thread.sleep(5000L)

}
```

大数据表优化

建立汇总表

建立流水表

分表分库