# 一 基本架构

Server层和存储引擎

server层

连接器,分析器,优化器,执行器,查询缓存

Mysql接收查询后,会先查找缓存,如果能命中缓存则直接返回

存储引擎

Innodb,myisam,memory

# 二 索引

### 多列索引

多个单列索引,在做多条件查询的时候性能会很差,因为涉及到额外的连接操作

多个单列索引在多条件查询时只会生效第一个索引

多个单列索引在多条件查询时优化器会选择最优索引策略，可能只用一个索引，也可能将多个索引全用上！ 但多个单列索引底层会建立多个B+索引树，比较占用空间，也会浪费一定搜索效率，故如果只有多条件联合查询时最好建联合索引！

## 联合索引

联合索引符合左匹配原则,创建了(A,B),再创建(B,A)不属于冗余索引

比如对于(a,b)b只是局部有序的

因此根据最左匹配原则,在遇到范围查询的时候,就会停止匹配

SELECT \* FROM table WHERE a > 1 and b = 2;

这种需要对(b,a)建立索引,因为a是范围查询

SELECT \* FROM table WHERE a = 1 and b = 2 and c = 3;

对于这种查询,mysql会自动优化顺序,所以还是要按照区分度来创建索引

SELECT \* FROM `table` WHERE a = 1 ORDER BY b;

如何建立索引？ 这还需要想？一看就是对(a,b)建索引，当a = 1的时候，b相对有序，可以避免再次排序！ 那么

SELECT \* FROM `table` WHERE a > 1 ORDER BY b;

如何建立索引？ 对(a)建立索引，因为a的值是一个范围，这个范围内b值是无序的，没有必要对(a,b)建立索引。

SELECT \* FROM `table` WHERE a IN (1,2,3) and b > 1;

还是对(a，b)建立索引，因为IN在这里可以视为等值引用，不会中止索引匹配，所以还是(a,b)!

## 聚簇索引

这是一种数据存储方式,存在Innodb引擎中,主键索引为聚簇索引,如果没有主键mysql会选择一个唯一的非空的列作为聚簇索引,如果没有会自己隐式创建一个主键.

索引和数据存放在一起. 只能有一个聚簇索引,其他为二级索引

因此二级索引的回表查找要经历两次,在叶子节点存储了主键值,因此主键不可以过大,否则其他索引数据都会很大

## Index pushdown

减少回表,因此在using index的时候是不会有icp的

在索引存在范围查询的时候出现,例如以下语句

Select first\_name,last\_name,gender from employee where first\_name=’Amer’ and last\_name like ‘B%’

## Count()

两个作用,统计列值的数量或者统计行数.统计列值时不统计NULL

Myisam在做count(\*)的时候是比较快的

Innodb在count(\*) 会访问全部数据量,但是可以using index

## Dependent subquery

在5.7里面已经被优化掉了?

## FirstMatch

explain

select \* from employees where emp\_no in (select emp\_no from salaries where emp\_no in (10001,20001,30001))

在拿外面的记录去匹配里面的记录的时候,如果已经匹配上了,那么不再继续匹配.

这个查询外面并没有全表扫描

## type和extra简要介绍

https://www.cnblogs.com/linjiqin/p/11254247.html

# 三 innodb和myisam差别

1. innodb支持事务,myisam不支持
2. Innodb支持外键,myisam不支持
3. Innodb是聚簇索引,myisam不是. innodb索引和数据存放在一个文件,二myisam索引和数据分开存储,索引保存指向数据文件的指针,myisam主键索引和二级索引是独立的
4. Innodb不保存表的行数,所以count(\*)会很慢,myisam反之很快
5. Innodb的最小锁粒度是行,myisam最小锁粒度是表

# 批量插入

合并数据:在一条sql中集中插入多条数据,可以通过配置修改sql长度. 这样避免多次sql,增加Mysql I\O消耗

一个事务:多次insert,避免创建多次事务 start transaction … commit

有序插入:无序数据会增大索引维护成本

（1）提高[数据库](http://www.111cn.net/list-55/)插入性能中心思想：尽量将数据一次性写入到Data File和减少数据库的checkpoint 操作。这次修改了下面四个配置项：   
1）将 innodb\_flush\_log\_at\_trx\_commit 配置设定为0；按过往经验设定为0，插入速度会有很大提高。

0: Write the log buffer to the log file and flush the log file every second, but do nothing at transaction commit.   
1：the log buffer is written out to the log file at each transaction commit and the flush to disk operation is performed on the log file   
2：the log buffer is written out to the file at each commit, but the flush to disk operation is not performed on it   
2）将 innodb\_autoextend\_increment 配置由于默认8M 调整到 128M

此配置项作用主要是当tablespace 空间已经满了后，需要MySQL系统需要自动扩展多少空间，每次tablespace 扩展都会让各个SQL 处于等待状态。增加自动扩展Size可以减少tablespace自动扩展次数。

3）将 innodb\_log\_buffer\_size 配置由于默认1M 调整到 16M

此配置项作用设定innodb 数据库引擎写日志缓存区；将此缓存段增大可以减少数据库写数据文件次数。

4）将 innodb\_log\_file\_size 配置由于默认 8M 调整到 128M

此配置项作用设定innodb 数据库引擎UNDO日志的大小；从而减少数据库checkpoint操作。

经过以上调整，系统插入速度由于原来10分钟几万条提升至1秒1W左右；注：以上参数调整，需要根据不同机器来进行实际调整。特别是 innodb\_flush\_log\_at\_trx\_commit、innodb\_log\_buffer\_size和 innodb\_log\_file\_size 需要谨慎调整；因为涉及MySQL本身的容灾处理。

# group by

index (first\_name,last\_name)

explain select first\_name from employees group by first\_name

'1', 'SIMPLE', 'employees', NULL, 'range', 'inx\_f\_l', 'inx\_f\_l', '16', NULL, '1289', '100.00', 'Using index for group-by'

explain select last\_name from employees group by last\_name

'1', 'SIMPLE', 'employees', NULL, 'index', 'inx\_f\_l', 'inx\_f\_l', '34', NULL, '299379', '100.00', 'Using index; Using temporary; Using filesort'

# limit

limit 100,50 会导致扫描抛弃大量不需要的数据

延迟关联策略,先用索引查出数据,然后再做一次关联

采用已知的位置 使用between and

# select type

primary 最外层的查询

simple 就一个select或者比较简单的查询

explain select \* from (select emp\_no from salaries where emp\_no<10002) a

subquery

explain select \* from employees where emp\_no=(select emp\_no from employees where emp\_no=10001)

union

带有union的查询

# explain

* id：执行编号，标识 select 所属的行
* select\_type：select 查询的类型
* table：查询的是哪个表
* partitions：匹配的分区
* type：关联类型，或者访问类型
* possible\_keys：该查询可以选用的索引
* key：该查询选用的索引
* key\_len：索引中使用的字节数
* ref：显示上述表的连接匹配条件，即哪些列或常量被用于查询索引列上的值
* rows：估计为了找到所需行而要读取的行数
* filtered：按表条件过滤的行的百分比
* Extra：额外的信息

select\_type

该列指明了查询的类型，以下为常见的取值

SIMPLLE：简单查询，该查询不包含 UNION 或子查询

PRIMARY：如果查询包含 UNION 或子查询，则最外层的查询被标识为 PRIMARY

UNION：表示此查询是 UNION 中的第二个或者随后的查询

DEPENDENT：UNION 满足 UNION 中的第二个或者随后的查询，其次取决于外面的查询

UNION RESULT：UNION 的结果

SUBQUERY：子查询中的第一个 select 语句

DEPENDENT SUBQUERY：子查询中的 第一个 select，同时取决于外面的查询

DERIVED：派生表 select，包含在 from 字句的子查询中的查询

UNCACHEABLE SUBQUERY：满足是子查询中的第一个 select 语句，同时意味着 select 中的某些特性阻止结果被缓存于一个 Item\_cache 中

UNCACHEABLE UNION：满足此查询是 UNION 中的第二个或者随后的查询，同时意味着 select 中的某些特性阻止结果被缓存于一个 Item\_cache 中

# type

const、eq\_ref、ref、range、index、ALL（从左到右，性能从好到差）

ref:使用非唯一索引扫描或者唯一索引的前缀扫描,最终返回一行

一种索引访问，也称索引查找，它返回所有匹配某个单个值的行。此类型通常出现在多表的 join 查询, 针对于非唯一或非主键索引, 或者是使用了最左前缀规则索引的查询。

explain select \* from employees where first\_name='Amer'

eq\_ref:在关联中使用唯一索引

多出现在多表join中,对于前表的每一个结果,都只能匹配到后表的一行,并且比较操作是=。在使用唯一性索引或主键查找时会出现该值，非常高效

explain select \* from employees where emp\_no in (select emp\_no from employees where emp\_no>10001)

and emp\_no<10006

index:全索引扫描

explain select last\_name from employees where last\_name='Amer' 一般索引覆盖了

# key

key列显示mysql实际使用的索引

# extra

Using index：使用覆盖索引，表示查询索引就可查到所需数据，不用扫描表数据文件，往往说明性能不错。

Using Where：在存储引擎检索行后再进行过滤，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。

Using temporary：在查询结果排序时会使用一个临时表，一般出现于排序、分组和多表 join 的情况，查询效率不高，建议优化。

Using filesort：对结果使用一个外部索引排序，而不是按索引次序从表里读取行，一般有出现该值，都建议优化去掉，因为这样的查询 CPU 资源消耗大。

# 介绍

<https://thinkwon.blog.csdn.net/article/details/104778621>

尽量避免在where字句中使用null判断

explain

select emp\_no from employees where emp\_no IS NOT NULL

这个会导致全表扫描

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

Or不一定会导致全表扫描

explain

select emp\_no from employees where emp\_no=10004 or emp\_no=10009 这个使用主键只查询2行

In 不一定会导致全表扫描

explain

select emp\_no from employees where emp\_no in (10004,10009)

explain

select count(distinct birth\_date) from employees 使用索引加速

1, SIMPLE, employees, , range, idx\_b, idx\_b, 3, , 4736, 100.00, Using index for group-by

隔离级别和锁

https://tech.meituan.com/2014/08/20/innodb-lock.html