2020 week3 总结

1. 为什么查询速度会慢
   1. 查询耗时操作

主要包括：网络、cpu计算、生成统计信息，执行计划，锁等待等操作

1. 慢查询基础：优化数据访问
   1. 分析查询性能低下的步骤

* 确认应用程序是否在检索大量超过需要的数据（是否查询限制条件是不是可以最大程度上限制查询结果集大小）
* 确认mysql服务器是否在分析大量超过需要的数据行
  1. 是否向数据库请求了不需要的数据
* 查询不需要的记录（where、having、limit的正确使用）
* 多表关联时返回全部列（join时select \*）
* 总是取出全部列（select \*）
* 重复查询相同的数据（正确使用缓存）
  1. Mysql是否扫描额外的记录
     1. 衡量查询开销的指标
* 响应时间（服务时间-> 数据库处理查询花费时间和排队时间->等待资源耗费时间）
* 扫描的行数和返回的行数
* 扫描的行数和访问类型（explain type all or ref）
  + 1. mysql where条件使用情况（由好到坏）
* 在索引中使用where条件来过滤不匹配的记录
* 使用索引覆盖扫描（explain extra中出现using index）来返回记录
* 从数据表中返回数据（explain extra中出现using where）来返回记录

1. 重构查询方式
   1. 一个复杂查询还是多个简单查询（视情况而定）

* 传统实现中任务网络通信、查询解析、优化是一件代价高昂的事 🡪 推荐在sql层尽可能多的完成工作
* 由于硬件不断进步 🡪 拆分成多个简单查询，可以减小应用层耦合
  1. 切分查询
* 将大查询切分成小查询
* 定期清除大量数据可能会长时间锁表、占用资源、沾满事务日志、阻塞其他查询🡪切分成每次清除一部分
  1. 分解关联查询
* 让缓存的效率更高
* 查询分解，执行单个查询可以减少锁的竞争
* 在应用层做关联，可以更容易对数据库进行拆分，更容易做到高性能和可扩展
* 查询本身效率也可能会有所提升
* 可以减少冗余记录的查询
* 相当于在应用中实现了哈希关联。而不是使用mysql的嵌套循环关联

1. 查询执行的基础
   1. 查询执行的生命周期

* 客户端发送一条查询指令给服务器
* 服务器先检查查询缓存，如果命中了缓存，则立刻返回存储在缓存中的结果。否则进入下一阶段。
* 服务器端进行SQL解析、预处理，再有优化器生成对应的执行计划。
* MySQL根据优化器生成的执行计划，再电泳存储引擎的API来执行查询
* 将结果返回客户端
  1. 查询状态（Show FULL PROCESSLIST 🡪该命令返回结果中的Command列就表示当前的状态）
* Sleep 线程正在等待客户端发送新的请求
* Query 线程正在执行查询或者正在将结果发送给客户端
* Locked 在mysql服务器层面，该线程正在等待表锁
* ANalyzing and statistics 线程正在收集存储引擎的统计信息，并生成查询的执行计划
* Copying to tmp table [on dist] 线程正在执行查询，并且将其结果集都复制到一个临时表中
* Sorting result 线程正在对结果集进行排序
* Send data 线程可能在多个状态之间传送数据，或者在生成结果集，或者在向客户端返回数据
  1. 查询缓存

在解析一个查询语句之前，如果查询是打开的，那么MySQL会优化检查这个查询是否命中查询缓存中的数据。

* 1. 查询优化处理

查询的什么周期的下一步是将一个SQL转换成一个执行计划，MySQL再依照这个执行计划何存储引擎进行交互。包括：解析SQL、预处理、优化SQL执行计划

* + 1. 语法解析器何预处理

通过关键字将SQL预计进行解析，并生成一个对应的解析树

* + 1. 查询优化器

一条SQL可能有很多种执行方式，优化器的作用是找到这其中最好的执行计划

* + 1. 优化策略
* 静态优化 直接对解析树进行分析，并完成优化
* 动态优化 与查询的上下文有关，每次查询的时候都需要重新评估，运行时优化
  + 1. MySQL能处理的优化类型
* 重新定义关联表的顺序 --> 数据表的关联并不总是按照查询中指定的顺序进行。（没太理解）
* 将外连接转换成内连接 --> 并不是所有的 OUTER JOIN 都必须以外连接的范式执行。（没太理解）
* 使用等价变换规则 --> 可以合并何减少一些比较，还可以一处一些恒成立和恒不成立的条件判断。例如：(5=5 and a>5) ==> (a>5);( a<b and b=c ) and a= 5 ==> (b>5 and b=c and a=5)
* 优化 count()、min()、max() --> 存在索引的话只需要找到最左端记录就可以优化min()、最右端记录优化max();
* 预估并转化为常数表达式 --> MySQL检测到一个表达式可以转化成常数的时候，就会一直吧该表达式作为常数进行优化处理。例如：在索引列上执行min()。
* 覆盖索引扫描 --> 但索引的列包含所有查询中需要使用到的列的时候，MySQL就可以使用索引返回需要的数据，而无需查询对应的数据行。
* 子查询优化 --> 在某些特定情况下可以将子长县转换成一种效率更高的形式，从而检查多个查询多次对数据进行访问
* 提前终止查询 --> 发现已满足查询需求时，MySQL总能立刻终止查询。例如：limit
* 等值传播 --> 两个列的值通过等式关联，那么MySQL能够吧其中一个列的where条件传递到另一列上。例如：select film.fim\_id from file inner join file\_actor using(file\_id) where film.file\_id > 500 ==> where file.file\_id > 500 and file\_actor.id > 500
* 列表in()的比较 --> MySQL会将in()列表中的数据先进行排序，然后通过二分查找的范式来确定列表中的值是否满足条件。
  + 1. 关联查询优化器
* 它决定了多个表关联时的顺序。
* 通常多表关联的时候，可以有多种不同的关联顺序来获得相同的执行结果。
* 关联查询优化器则通过评估不同顺序的成本来选择代价最小的那个。
* straight join重写查询，优化器会根据自己手写的关联顺序来执行
  + 1. 排序优化

排序是一个成本很高的操作，从性能角度考虑，应尽可能避免排序或者尽可能避免对大量数据进行排序。

* 索引排序-->Mysql可以通过索引排序
* 文件排序-->当不能使用索引生成排序结果的时候，Mysql需要自己进行拍摄，如果数据量小则在内存中进行，如果数据量大则需要在磁盘中排序。mysql在文件排序的所需临时存储空间将会非常大。对每个记录都会分配一个足够长的定长空间进行存放，这个定长空间必须足够长以容纳其中最长的字符串。

例如：如果是varchar列需要分配其最长长度，UTF-8字符集，mysql还将对每个字符预留三个字节。

* + - 1. mysql排序算法
         1. 两次传输排序（旧版本使用）
* 读取行指针何需要排序的字段，对其进行排序，然后再根据排序结果读取所需要的数据行。
* 需要从数据表中读取两次数据，第二次读取数据的时候，因为时读取排序列进行排序后的所有记录，这回产生大量随机I/O，成本较高。
* 优点：在排序的时候存储尽可能少的数据，让“排序缓冲区”中尽可能容纳尽可能多的数据行进行排序。
  + - * 1. 单次传输排序（新版本使用）
* 先读取查询所需要的所有列，再根据给定列进行排序，最后直接返回排序结果。
* 优点：不需要从数据表读取两次数据，对于I/O密集型应用，会提高效率；这种算法只需要一次顺序I/O速度所有数据，而无需任何的随机I/O。
* 缺点：如果返回的数据列非常多，非常大，会额外占用大量的空间。
  + - 1. 关联查询两种排序方式
* order by的所有列都来四关联的第一个表，mysql会在处理关联的第一个表的时候进行文件排序。explain using filesort；
* 除第一种情况外，mysql会将关联结果存放到一个临时表中，关联结束后在进行排序。explan using temporary；using filesort；
  1. 查询执行引擎

mysql的查询执行引擎会更加执行计划来完成整个查询。知识简单的根据执行计划给出的指令逐步执行。

* 1. 返回结果给客户端

查询执行的最后一个阶段就是将结果返回客户端。

mysql将结果集返回客户端是一个增量，逐步返回的过程。服务器处理完最后一个关联表，开始生成第一条结果时，mysql姐可以开始向客户端逐步返回结果集了。

服务的无需存储过度结果，内存消耗低。

客户端可以第一时间获取返回的结果。

1. 查询优化器的局限性
   1. 关联子查询

mysql的子查询实现的非常糟糕，最糟糕的一类时where天降中包含In（）的子查询。

例：

select \* from film

where film id in(

select film\_id from file\_actor where actor\_id = 1

)

MySQL会见相关的外层表压倒子查询中，它任务这样可以更高效率的查到数据行

==>

select \* from film

where exist (

select \* from file\_actor where actor\_id = 1 and file\_actor.film\_id = film.film\_id

)

改进方案一

select \* from film

inner join file\_actor using(film\_id)

where actor\_id = 1

改进方案二

select \* from film

where exist(

select \* from file\_actor where actor\_id = 1

and file\_actor.film\_id = film.film\_id

)

* 1. 如何用好关联子查询

一般使用left outer join 重新查询。具体还得根据测试结果来判断。

* 1. Union的限制

(select \* from film\_1) union (select \* from file\_2) limit 20;

此时会吧表一和表二的信息放在同一个临时表中，然后再取出前20条记录

改进方案

(select \* from film\_1 limit 20) union (select \* from file\_2 limit 20) limit 20;

* 1. 索引合并优化
  2. 等值传递 （很少会碰到）
  3. 并行执行 （暂时mysql做不到，不支持）
  4. 哈希关联 （MariaDB已经实现了哈希关联）
  5. 松散索引扫描（不支持）
  6. 最大值最小值优化

前提id自增长

select min(id) from film where name = "aini"

改进方案

select id from film use index(primary) where name = "aini" limit 1;

* 1. 在同一个表上查询何更新

update tb1 as outer\_tb1

set cnt = (

select count(1) from tb1 as inner\_tb1

where inner\_tb1.type = outer\_tb1.type

)

error 1093:you cannot sprcify target table 'outer\_tb1' for update in from clause

可以通过使用生成表的形式绕过限制

update tb1

inner join(

select type,count(1) as cnt

from tb1

group by type

) as der using(type)

set tb1.cnt = der.cnt;

1. 查询优化器的提示（hint）
   1. 如果对优化器选择的执行计划不满意，可以使用优化器提供的几个提示来控制最终的执行计划。
      1. HIGH\_PRIORITY 和 LOW\_PRIORITY

HIGH\_PRIORITY 用于select预计的时候，Mysql会将此select语句重新调度到所有正在等待表锁一级修改数据的语句前面。

LOW\_PRIORITY 会让该语句一直处于等待状态，只要队列中还有需要访问通过一个表的语句-即使是王提交到服务器的语句。

只对使用表锁的存储引擎有效，不要再innoDB或者其他有细粒度锁机制和并发控制的引擎中使用，这两个提示胡导致并发插入被禁用，严重影响性能。

* + 1. DELAYED

对Insert 和 replace 有效。Mysql会将使用该提示的语句立即返回给客户端，并将插入的行数据放入到缓冲区，然后再表空闲时批量将数据写入。

并不是所有存储引起都支持这样的做法；

并且该提示会导致函数last\_insert\_id()无法正常工作

* + 1. staraght\_join

这个提示可以防止再select语句的select关键字的后面。可以让查询中所有的表按照语句中出现的顺序进行关联

也可以放在任意两个关联的名字之间。固定前后两个表的关联顺序。

当你确定无论怎样的where条件，某个固定的关联顺序始终时最佳的时候，使用这个提示可以大大提高优化器效率。

升级Mysql版本时需要重新考察下这类查询，某些新的优化特性可能会因为该提示而失败。