查询的生命周期：从客户端，到服务器，然后到在服务器上进行解析，生成执行计划，并返回结果给客户端。

查询需要在不同的地方花费时间，包括网络，CPU计算，生成统计信息和执行计划、锁等待等操作。

6.2慢查询基础：优化数据访问

对于低效的查询，通过以下两个步骤来分析总是很有效的：

1.确认应用程序是否在检索大量超过需要的数据。这通常意味着访问了太多的行，但有时候也可能是访问了太多的列。

2.确认MySQL服务器是否在分析大量超过需要的数据行。

6.2.1 是否向数据库请求了不需要的数据

一些经典的案例：

1.查询不需要的记录

常见的误区就是：MySQL会只返回需要的数据，实际上MySQL却是先返回全部结果集在进行计算。最简单有效的解决方式是在查询后面加上LIMIT。

2.多表关联返回全部列

3.总是取出全部列

取出全部的列，会放优化器无法完成索引覆盖扫描这类优化，还会为服务器带来额外的I/O、内存和CPU的消耗。

4.重复查询相同的数据

6.2.2 MySQL是否在扫描额外的记录

简单衡量查询开销的三个指标：

1.响应时间

2.扫描的行数

3.返回的行数

理想情况下扫描的行数和返回的行数应该是相同的。

扫描的行数与访问类型也有很大的关系，访问的类型包括扫描表、扫描索引、范围扫描唯一索引查询和常数引用。

一般MySQL能够使用如下三种方式应用where条件，从好到坏一次为：

1.在索引中使用where条件来过滤不匹配的记录。这是在存储引擎层完成的。

2.使用索引覆盖扫描要返回的数据，直接从索引中过滤不需要的记录并返回命中的结果。这是在MySQL服务器层完成的，但无须在回表查询记录。

3.在数据表中返回数据，然后过滤不满足条件的记录，这是在MySQL服务器层完成的，需要先从数据表中读出记录然后过滤。

6.3 重构查询的方式

6.3.1 一个复杂查询还是多个简单查询

6.3.2 切分查询

6.3.2 分解关联查询

1.让缓存效率更高：应用程序更容易缓存单表查询的数据。

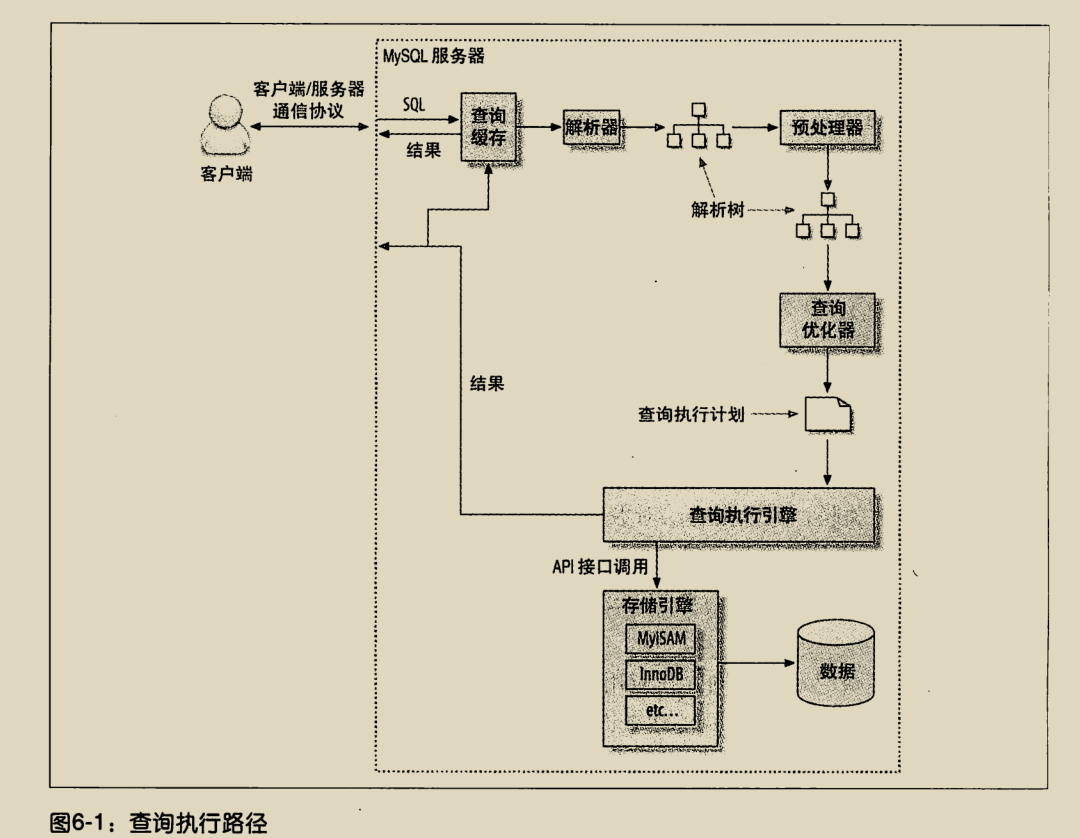
2.将查询分解以后，执行单个查询可以减少锁的竞争。

3.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拓展。

4.查询本身的效率也可能提升。

5.可以减少冗余记录的查询。

6.4 查询执行的基础



6.4.1 MySQL客户端/服务通信协议

MySQL客户端与服务器端之间的通信协议是“半双工”，即任何一个时刻，要么是由服务器端向客户端发送数据，要么是客户端向服务器端发送数据。因此无法将一个消息切成小块独立来发送。

查询的状态

任何一个时刻都可以查询MySQL连接的状态，最简单的是使用：

SHOW FULL PROCESSLIST。

这些状态值包括：

1.Sleep:线程正在等待客户端发送新的请求。

2.Query:线程正在执行查询或者正在将结果发送给客户端。

3.Locked:在MySQL服务层，该线程正在等待表锁。

4.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。

5.Copying to tmp table[on disk]:

线程正在执行查询，并且将其结果都复制到一个临时表中，这种状态要么是在做GROUP BY操作，要么是在做文件排序操作，或者是UNION操作。如果这个状态后面还有“on disk”标记，那表示MySQL正在将一个内存临时表放在磁盘上。

6.Sorting result:线程正在对结果集进行排序。

7.Sending data：线程可能在多个状态之间传递数据，或者在生成结果集，或者在向客户端返回数据。

6.4.2 查询缓存：这个检查是通过一个对大小写敏感的哈希查找实现的。

6.4.3 查询优化处理

1.语法解析器和预处理

2.查询优化器

一条查询可以有很多种执行方式，最后都返回相同的结果。优化器的作用就是找到这其中最好的执行计划。

查询优化策略可以分为两种：

1.静态优化-编译时优化

a.静态优化可以直接对解析树进行分析，并完成优化。

b.静态优化不依赖于特别的数值。

c.静态优化在第一次完成以后就一直有效，即使使用不同的参数重复执行也不会发生变化。

2.动态优化-运行时优化

a.动态优化则和查询的上下文有关。

造成MySQL优化器选择错误的执行计划的原因：

1.统计信息不准确。

2.执行计划中的成本估算不等于实际执行的成本。

3.MySQL的最优可能和你想象中的最优不一样。你可能希望执行时间尽可能的短，但是MySQL只是基于其成本模型选择最优的执行计划，而有些时候这并不是最快的执行方式。故根据执行成本来选择执行计划并不是完美的模型。

4.MySQL从不考虑其他并发执行的查询。

5.MySQL也并不是任何时候都是基于成本的优化。

6.MySQL不会考虑不受其控制的操作的成本。

MySQL能够处理的优化类型：

1.重新定义关联表的顺序

2.将外连接转换为内连接

3.使用等值变化规则

4.优化count()、min()和max()

5.预估并转化为常数表达式

6.覆盖索引扫描

7.子查询优化

8.提前终止查询

9.等值传播：如果两个列的值通过等式关联，那么MySQL能够把其中一个列的where条件传递到领一个列上。

10.列表的IN()的比较

3.数据和索引的统计信息

这些统计信息包括：每个表或者索引有多少个页面、每个表的每个索引的基数是多少、数据行和索引长度、索引的分布信息等。

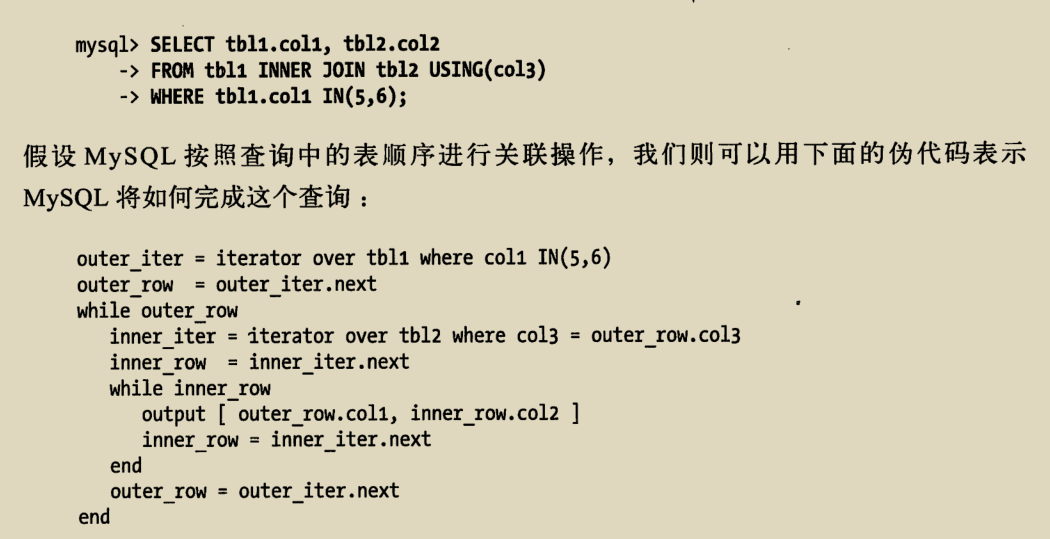
4.MySQL如何执行关联查询

MySQL认为任何一个查询都是一次“关联”，而不仅仅是一个查询需要用到两个表匹配才叫关联。故在MySQL中，每一个查询，每一个片段（包括子查询、单表的select）都可能是关联。

MySQL执行关联的策略-嵌套循环关联操作

MySQL先在一个表中循环取出单条数据，然后再嵌套循环到下一个表中寻找匹配的行，依次下去，直到找到所有表中匹配的行为止。然后根据各表匹配的行，返回查询中需要的各个列。MySQL会尝试在最后一个关联表中找到所有匹配的行，如果最后一个关联表无法找到更多的行，MySQL返回到上一层关联表，看是否能够找到更多的匹配记录，依次类推迭代执行。

内连接：



外连接：



5.执行计划

MySQL生成查询的一棵指令树，然后通过存储引擎执行完成这棵指令树并返回结果。最终的执行计划包含了重构查询的全部信息。

6.关联查询优化器

关联查询优化器通过评估不同顺序时的成本来选择一个代价最小的关联顺序。

7.排序优化

当不能使用索引排序时，MySQL需要自己进行排序，如果数据量小则在内存中进行，如果数据量大则需要使用磁盘排序。MySQL将这个过程统一称为文件排序（filesort）。

如果需要排序的数据量小于“排序缓冲区”，MySQL使用内存进行“快速排序”操作。如果内存排序不够，那么那么MySQL会将数据分块，对每个独立的快使用“快速排序”进行排序，并将各个块的排序结果放在磁盘上，然后将各个排好序的块进行合并，然后返回排序结果。

MySQL有两种排序算法：

两次传输排序(旧版本使用)：

读取行指针和需要排序的字段，对其进行排序，然后在根据排序结果读取所需要的数据行。这需要进行两次数据传输，即需要从数据库读取两次数据，第二次读取数据的时候，因为是读取排序列进行排序后的所有记录，会产生大量的随机I/O,传输成本很高。

单次传输排序(新版本使用MySQL4.1)：

先读取查询所需要的所有列，然后根据给定列进行排序，最后直接返回排序结果。缺点是，如果需要返回的列非常多、非常大，会额外占用大量的空间。

关联查询排序，MySQL会分两种情况排序：

a.如果ORDER BY子句的所有列都来自关联的第一个表，那么MySQL在关联处理第一个表的时候就进行文件排序。Extra字段列会有“Using filesort”。

b.除a以外的所有情况，MySQL都会先将关联的结果存放在一个临时表中，然后在所有的关联都结束以后，再进行文件排序。Extra字段列会有“Using temporyary ,Using filesort”。

6.4.4 查询执行引擎

6.4.5 返回结果给客户端

6.5 MySQL查询优化器的局限性

6.5.1 关联子查询

MySQL的子查询实现的非常糟糕。最糟糕的一类查询就是where条件中包含IN()的子查询语句。转换为内连接或者使用EXISTS()来等效改写获取更高的效率。

IN()子查询：

EXPLAIN SELECT

\*

FROM

t\_role role

WHERE

role\_id IN (

SELECT

role\_id

FROM

t\_user\_role WHERE user\_id = 1

);

内连接：

EXPLAIN SELECT

\*

FROM

t\_role role

INNER JOIN t\_user\_role ur ON role.role\_id = ur.role\_id

WHERE

ur.user\_id = 1;

EXISTS()：

EXPLAIN SELECT

\*

FROM

t\_role role

WHERE EXISTS(

SELECT \* from t\_user\_role ur WHERE role.role\_id = ur.role\_id and ur.user\_id = 1

);

exists:存在，后面一般都是子查询，当子查询返回行数时，exists返回true。

6.5.2 UNION的限制

有时，MySQL无法将限制条件从外层下推到内层，这使得原本能够限制部分返回的结果条件无法应用到内层查询的优化上。

如果希望UNION的各个子句能够根据LIMIT只取部分结果，或者希望能够先排好序再合并结果集的话，就需要在UNION的各个子句中分别使用这些子句。

6.5.3 索引合并并优化

6.5.4 等值传递

6.5.5 并行执行

6.5.6 哈希关联

6.5.7 松散索引扫描

6.5.8 最大值和最小值

6.5.9 在同一个表上查询和更新

6.6 查询优化器的提示（hint）

6.7 优化特定类型的查询

6．7.1 优化count()查询

Count两种不同的作用：

1.统计某个列的数量，在统计列值时要求列值时非空。

2.统计结果集的行数，最好使用count(\*)。

6.7.2 优化关联查询

1.确保ON或者USING子句中的列上有索引。一般来说，除非有其他理由，否则只需要在关联顺序中的第二个表的相应列上创建索引。

2.确保任何的GROUP BY和ORDER BY中的表达式只涉及到一个表中的列，这样MySQL才有可能使用索引来优化这个过程。

6.7.3 优化子查询

6.7.4 优化 GROUP BY和DISTINCT

6.75 优化LIMIT分页

1.采用延迟关联

7 MySQL高级特性

水平分区：将同一表中不同行的记录分配到不同的物理文件中。

垂直分区：将同一表中不同列的记录分配到不同的物理文件中。

模糊查询的正确方式：

select userId，name from user where userId like '123%';

字段类型是字符串，where时一定用引号括起来，否则索引失效

select \* from user where userid ='123';