第四章：改进方法

4.1

有许多种存储序列化对象数据的方法，但是大多数方法并不合适于处理通用关系数据库管理系统的操作。一种方法是使数据保持为原来的字符串格式并且存储在单个文本域。这让装载的过程变得简单（因为装载之前并不需要转换）。但是操作这些数据变的开销更大因为系统必须在执行计算前将数据转换为逻辑表达式。

另一种可选的方法是用一个像Apache Avro或Google Protocol Buffers这样使用二进制块而不是用文本来表示数据对象的序列化格式。通常情况下，两种格式都消除了句法上人类可阅读文本表示的冗余，并且提供了比在单个键之间迭代速度更快的技术方法，尤其是通过记录存储序列号数据的模式。然而，两种格式，像JSON，是“顺序的”，这也就意味着在这些原始数据上不支持随机读取。为了从给定的数据中取得单个一个键，应用必须要么将整个给定数据解序列化使之成为逻辑形式并然后将适当的属性解引用；要么每次读取整个序列化数据的一个属性直到获得所需的属性或者到达给定数据的尽头（如果所需的属性不存在）。

因为数据的获取（关系代数中投影）是一个对于SQL查询来说非常常用的操作，我们决定使用一个常规的序列化格式而不是上面说的更关注于数据转换跟平台独立性而不是分析性的两种方法。特别是当我们的格式通过允许对数据的随机读取来减少获得存储在一个序列化对象里的属性时。我们在附录Azhong探索了Sinew’s format,Avro,跟Protocol Buffers在获取操作时的性能差别。

与一个标准的关系型数据库管理系统的元组非常类似，我们的序列化格式有一个包括对象元数据的头部跟一个包含实际数据的主体。特别地，这个头部，也就是在图5种所展示的结构，由一个表示记录中属性属性个数的整数，跟上一串对应于出现的键的属性ID（用于区分类别）的有序整数组成。在属性ID序列之后是第二串表示每个属性在数据中的字节偏移值的整数序列。这题包含用二进制数表示的实际数据。

我们的序列化格式让Sinew能够通过从它的数据中提取文件结构的方式来快速确定一个键的位置或者无需读取整个序列化数据就能确定某个键不存在。当寻找一个键时，Sinew仅仅需要确认属性ID的列表。当最终没有搜索到时，就可以确定这个键不存在于这个文档中。如果它找到了这个键，则会在偏移列表中找出偏移值，然后跳转到数据中正确的位置获得所需类型的值。为了将在头部中属性ID的二进制查找的缓存位置最大化，我们选择从偏移列表中分离键的列表（而不是讲偏移信息放在键的后面）。

键是直接获取的。每当给出一个目标键的时候，查找模式会获取对应的属性ID并从目录的字典中取得类型信息。对于每一个记录，它稍后会对位于头部的属性ID列表进行二进制查找。并且如果ID存在，它会从跟在属性ID后的列表中获取偏移量。模式可以通过这个属性与下一个的偏移信息来计算属性的长度并且以正确的类型获取属性的值。在一个给定的数据项中进行查找的开销是O（logN），因为它为了偏移的引用跟目标值的长度在头部进行了二进制搜索。因为这个，它会明显地笔其他前面提到过的所需开销为O（n）的序列化格式表现出的性能要好。得益于连续存储在缓存的属性ID，属性查找在常数系数上也有更好的表现。

值得注意的是，面向列存储的序列化格式如RCFiles或者Parquet允许随机访问并且可以在以列为对象的数据池中序列化数据。然而，当给定一个将我们的存储解决方案混合设计而成的将一些属性存储到物理列中而其他的序列化在一个列存储的数据池的设计方案时，数据池需要与物理列所取的方向适配。因此，如果Sinew支撑的数据库系统时列存储的，RCFiles或者Parquet可能被用来取代我们常用的序列化格式。然而，如果支撑数据库系统的是行存储，一个列存储的数据格式不能与这样的数据池兼容。并且，按照我们的习惯，应该使用以对象为中心的序列化格式。

4.2

在我们的综合存储模式中，就算他们被具体化，嵌套对象跟数组还是会因为内容集合让优化控制器执行缓慢而引发执行时的阻塞。在这一节，我们演接触一些技术来提升嵌套集在物理上的表现。

对于嵌套的对象，由多种选择。尽管Sinew将会把所有的嵌套对象的子属性分类，也就是将他们具体化并且在需要的情况下标记他们已经被具体化，但是完全平整的数据不能达到最优的结果（在Sec。. 3.1.1中讨论）。因此，当Sinew缺省地选择单个的每行包含一个文件的表的时候（使用一个通用关系模式），系统会让模式上产生这种极端情况下的缓和。如果有可以在在文件集中组成的逻辑组（比如嵌套对象这个例子），用户可以特别地制定这些组放入特殊的表中并且在查询时间中将他们链接起来。

在嵌套数组的情况下，用户可以根据数组句法的重要性挑选一种方法（比如无序集，有序集等等）。缺省情况下，系统将数组作为一个关系型数据库管理系统数组的数据格式来存储。但是如果数组元素的个数是有限的（并且小），系统会将每个位置单独存储为一个列。当数组中的对象是齐次时，可以让Sinew构成一个更加理想的无力模式，从而大大提升性能。

4.3

因为大多数文本索引的实现都包含了可以提供高性能范围查询、部分匹配跟模糊匹配的机制。所以通过对个性数据库管理系统中存储的数据增加一个外部索引，我们可以更加增强Sinew的性能与查询的表达性。

反向索引对虚拟列的查询十分有用。从大的方面来讲，反向文本索引表示输入数据，并将检索项编制在一起，通过使用一组包含某个项的记录相对应的ID值。此外，他能通过强制类型字段来提供编制其中检索组的选择。Sinew利用了这个功能，它将字段同目录中每一个属性结合，并将虚拟列断言重写为文本索引查询。这个结果（匹配的ID记录的集合）可以在之后作为对原始关系的过滤。

尽管我们收录反向索引的主要目的是为了加快虚拟列上含有标准SQL WHERE子句断言的查询的聘雇，但是Sinew将反向索引用来支持整个数据集合的文本查询。用户可以搜索出现在所有列（物理或虚拟）的文本，Sinew将会用反向索引来查找含有目标文本行的集合。

全文本查询性能不仅仅让Sinew可以提供一个可以更加易于表达对半结构化和数据关系断言集合，也可以让Sinew可以处理与此数据一起出现的完全非结构化的数据，这是通过实验简单滴讲非结构化数据存储到一个普通的文本列中病提供全文本索引来存取。不过由于非结构话数据与关系属性无类似之处（与键和属性对应的半结构化数据不同），与这种非结构化数据相连的不是“存粹的”SQL，而是Sinew有一个特殊的函数，这个函数可以在SQL查询的WHERE语句中被调用，同时传入两个参数：查询够格键和字符串

一条简单的查询语句如下：

SELECT \*

FROM webrequests

WHERE matches(‘\*’, "full text query or regex");