3.系统结构

从根本上说,Sinew与之前的工作不同在于，这是一个可以查询和操作多结构数据而不要求用户在查询过程任何点定义模式的系统。﻿ 此外，它通过利用现有的数据库功能和委托查询优化和执行对一个潜在的RDBMS尽可能提供高性能的读和写。

虽然我们在设计中严重依赖RDBMS，我们限制设计不要求改变RDBMS代码。﻿ 这大大扩展了Sinew的适用性，允许它被用来和大部分现有的关系数据库系统（最好是对象关系系统）合作而不要求转换到一个新的数据库系统。Sinew因此应被认为是一个上面放着一个数据库系统的层。

Sinew的结构如图1所示。﻿ 在最高层，系统包含以下内容，这部分将在本章剩下部分详述：

•有“混合”物理模式的RDBMS(3.1.1节)

•一个目录(3.1.2节)

•一个数据库模式分析器(3.1.3节)

•一个列实现器(3.1.4节)

•一个加载器(3.2.1节)

•一个查询重写器(3.2.2节)

•一个(可选)原文索引(4.3节)

为方便讨论我们将假设数据以JSON格式输入Sinew。﻿ 通常，任何数据可以支持被表示为一个需要的，可选的，嵌套的和重复的字段，即使相同名字中的种类不完全相同。

3.1存储层

3.1.1混合模式

给定一个输入数据由文档形式包含一些潜在的嵌套键值对的集合，Sinew自动生成一个逻辑的，面向用户的视图和独立于RDBMS关系数据的底层的模式来维护它。﻿ 我们先讨论逻辑视图(一个基于加载目前为止的数据创造的发展模式，不利于用户发出SQL查询)然后讨论它的物理实现。

在逻辑视图，Sinew提供了一个普遍的关系，每个文档对应一行。﻿ 任何文档中的每对独一的顶层键会在通常模式中用SQL提供的传统获得模式作为一个列开给客户。﻿ 因此，对文档中的每一对键值对，值会被(逻辑上)储存在文档对应的行和(逻辑上)键对应的列。﻿ 如果文档包含一个嵌套对象，它的子键也被引用为以父对象的键之前的子键的点号分隔的名字不同的列。﻿ 与其他数据库原语如字符串和整数，嵌套对象保持可被最初的键引用。﻿ (数组没有这么直接且将在4.2节讨论)

例如，给定图2给定的数据集，用户将有图3展示的视图，和下列查询：

SELECT url FROM webrequests WHERE hits > 20;

将返回所有在多于20“hit”的对象中与键“url”的存在有联系的值。

有两种方式储存RDBMS逻辑模式的属性，我们将用物理和虚拟来讨论两种方式。﻿ 一个物理列是既在逻辑视图的列也在数据系统中储存为物理列。﻿ 相比之下，一个虚拟列是逻辑视图的列而在数据库系统中不存在物理列。﻿ 相反，它可以从原始键值对的序列化表示中(在4.1节介绍)被运行时取出。

因此，有两种极端从逻辑模式映射到物理模式：﻿ (1)将所有列储存为物理列或(2)将所有列储存为虚拟的。﻿ 在(1)中，我们有一个与上面介绍的逻辑模式相同的物理数据库模式(即一个包含对于每一个数据集合中的唯一键的列的宽表)，而在(2)，我们对每一个对象序列化(文本或二进制)有一个单独列的表并储存在该列(每行一个对象)。

所有物理列选项提供一个更简单的系统设计但能对许多、潜在的稀疏属性或那些大型嵌套对象运行问题解决数据集。﻿ 例如，行获得的RDBMS对每表中每行的每一个模式属性分配至少1位储存空间。﻿ 这个空间存留在元组头或者与其他数据在元组的身体。﻿ 这种对每个属性预分配的空间(是否属性的非空值存在)会导致储存稀疏数据变得膨胀，会显著的降低性能。

为更好理解全物理获得的问题，考虑现有版本的推特API。这API规定推文有13个可为空的、顶层属性，当完全展开时将扩展为23个键。加入嵌套用户对象(可选包含推文)、标签、符号、url、用户评论和媒体后，原始推文展开后的版本会包含至多150个可选属性。﻿ 如果我们试图在InnoDb，一个MySQL受欢迎的储存引擎，保存这些表现，每条记录将达到至少300个字节的额外储存开销(InnoDB标题的每项属性包含2个字节)。对一个最小推文(只有文字而没有额外的项目或元数据)，这个标题开销会比数据本身还大。﻿ 甚至在有高效NULL表达的RDBMS(例如Postgres，一个使用位图来表示NULL的存在)，有个不可忽略的系统消耗来对稀疏系统保持追踪。﻿ 因此，作为一个实际的限制，和简化目录的设计，大部分行获得RDBMS占据了一个模式可能声明的列中的一个硬限制。

列获得数据库系统对宽的、稀疏的数据没有相同的储存膨胀问题且能扩展数据至比行储存更大的程度。﻿ 然而，一个完全展开的物理表示通常不是最佳的。﻿ 再次提到推特的API，一个普遍的查询模式可能检索发送了一个给定推文的“用户”(推特嵌套一个完全的用户对象作为推文的一个属性)。  ﻿ 在一个完全展开的表达式，嵌套键的父对象不在明确存在。﻿ 相反，他们是合并所有属性名是所需父对象扩展的列的结果。﻿ 为了返回父对象，系统必须首先计算列的恰当集然后支付合并他们的消耗。对于频繁访问的嵌套键，最后把他们储存为单独的集合而不是独立的元素。

下面给出“全物理列”的稀疏开销方法，一种可能会去另一个极端，上述“全虚拟列”的方法。﻿ 虽然从列中取出键值的消耗包括序列化数据会保持较小(见4.1节和附录B)，用一个单独列保存所有的序列化属性会降低RDBMS优化器生成高效查询计划的能力。﻿ 这是因为，我们规定不修改底层数据库代码，优化器不能维护属性层的数据。对优化器而言，虚拟列并不存在。

为了演示查询计划潜在的不同，我们列出了表1来自1000万条来自推特推文的查询列表。﻿ 每条推文有上述介绍的属性，稀疏度的范围从1%到100%。﻿ (见第6节我们的实验设置)。﻿ 这个查询计划通过满足表2展示条件的优化器来优化。﻿ 这些计划包含不同的用于UNIQUE和GROUP BY查询的操作器，也和JOIN里的合取查询不同。﻿ 这些差异可以归结于这样的事实：优化器为虚拟列的查询假设一个固定的选择度(实验中1000万数据中的200行)。﻿ 因此当选择度事实上远远更低的时候()，得到的查询计划会是次优的，这可能带来巨大性能影响，取决于数据集和系统配置。﻿ 例如，当查询一个物理模式对比一个虚拟的时自合取看到一个数量级的提高，最初的50分钟查询时间仅在4分钟内完成。

为了发挥每属性一列映射的性能优势和空间高效和单列序列化的可扩展性，我们合并两种映射。﻿ 在我们的混合模式，我们为一些属性创造列并将剩余储存在一个特别的序列化列，我们称为列储存所。﻿ 这允许我们在物理模式更有效的时候实现在数据库系统利用他们的好处，同时保持稀疏和至少频繁获得的键为稀疏列。