Design and Implementation of Relation Database and Non-Relation Database Unified Query Model

Zheng Xiaoyu, Bian Naizheng

College of Information Science & Engineering, Hunan University, Changsha, Hunan, China (keaiyuzhu@163.com)

***Abstract*—For scenarios that application data stored separately in relational database and non-relational database, an establishment of a unified query abstraction layer for SQL and NoSQL database is proposed, which can query data from different database as from a single data source. By transforming NoSQL data into triples to merge the NoSQL data as a virtual relational into SQL database and the original NoSQL data is rebuilt by series of self-join operations of the triples. For scenes that PostgreSQL and MongoDB are used simultaneously, the above model is implemented and verified. Experimental results show that through an intermediate triple conversion and a series of self-join of the triples, a unified query abstraction model for SQL and NoSQL database is feasible.**

***Keywords*—Relational Data, Non-relational Data, Triple; Self-join, NoSQL query pattern**

SQL 与 NoSQL 数据库的统一查询模型的设计与实现

郑小裕　边耐政

湖南大学信息科学与工程学院，长沙，湖南，中国

摘　要　针对将数据混合存储于关系型数据库和非关系型数据库的应用场景，建立了一个统一查询 SQL 和 NoSQL 数据库的理论模型，使其对以上两种数据库的查询操作对外表现为对单一的数据源进行操作。通过将 NoSQL 类型数据转换为三元组的形式，将这些三元组作为虚拟的关系数据合并到 SQL 数据库中，而原始 NoSQL 数据则通过将这些转换后的三元组进行一系列的自连接操作来重建。针对混合使用 PostgreSQL 和 MongoDB 的场景，对上述统一查询模型进行了实现和验证。实验表明，通过一个中间三元组的转换和对转换后的三元组进行一系列的自连接操作来统一访问 SQL 和 NoSQL 的数据抽象模型是可行的。

关键词　关系型数据；非关系型数据；三元组；自连接；NoSQL 查询模式

1．引言

近年来在 Web 开发领域中，除了传统的关系型数据库比如 MySQL，PostgreSQL 和 SQL Server 仍然被广泛使用，越来越多的非关系型数据库，如 CouchDB，MongoDB，Redis 也成为比较流行的数据存储解决方案[1]。不同于传统的关系型数据库，非关系型数据库采用面向文档、图形、键值对、对象等存储方式，更适合存储某些特定类型的数据，其使用也成为一种趋势和潮流[2]。在本文中，以上提到的非关系型数据库，称为 NoSQL 数据库，传统的关系型数据库则称为 SQL 数据库。

然而，SQL 数据库经过了数十年的研究和性能优化，很多数据都可以通过关系型实体进行很好的建模，SQL 数据库不可能完全被相对不成熟的 NoSQL 产品替代和舍弃[3]。当某个应用的一部分数据非常适合存储于 NoSQL 数据库，而剩下的数据又是典型的关系型数据适合存储于 SQL 数据库时，最适合的解决方案是将数据分离存储。比如在省交通厅十二五规划的交通信息化基础支撑体系工程项目先期建设中，涉及到基于多平台的数据融合技术的研究，在查询车辆实时 GPS 信息时，由于这部分数据更新快，历史累积量庞大，非常适合存储于 NoSQL 数据库中，而具体的交通工具的属性则适合存储在 SQL 数据库中。然而，当两个数据源的数据都要查询并需要整合时，需要开发人员额外的编程来查询和结合不同数据源的两部分数据并给出正确的结果。

因为分离的数据源，开发人员需要访问不同的数据库系统。目前，NoSQL 查询语言没有统一的标准，它是多种

国家自然科学基金项目支持（资助号：61202461）

存储类型的一个统称[4]。总的来说，在一个应用中使用多个数据库系统将增加应用的复杂性并降低其可维护性。为了克服上述缺点，有必要提出一种解决方案允许开发人员在具体应用中使用正确的存储解决方案并有效、方便地管理存储于不同数据源的数据。这既有理论意义，又具有实践价值。

本文提出的基于关系型数据库与非关系型数据库的统一查询模型旨在使得在各自的数据存储系统的优势充分发挥的基础上，为 SQL 和 NoSQL 存储方案提供统一的数据访问接口，在 SQL 和 NoSQL 之间架起一座桥梁，屏蔽在需要同时查询不同数据源中的数据时所涉及到数据查询和组合的问题，即为开发人员提供一个屏蔽底层的分离的数据存储源的解决方案。

2．SQL 和 NoSQL 统一查询模型的理论框架

本文给出一个通用的解决方案和理论框架，使之对任意的 SQL 和 NoSQL 数据库都有效可行。

2.1 混合数据库

初步的解决方案是在 SQL 和 NoSQL 数据库上层再构建一个抽象层。通过该抽象层，使用一种查询语言来获得两个数据源的数据。该抽象层负责解析查询，从底层的 SQL 和 NoSQL 数据库获取数据，并将所获取的数据片断组合成一个单一的查询结果返回。如图 1 所示。

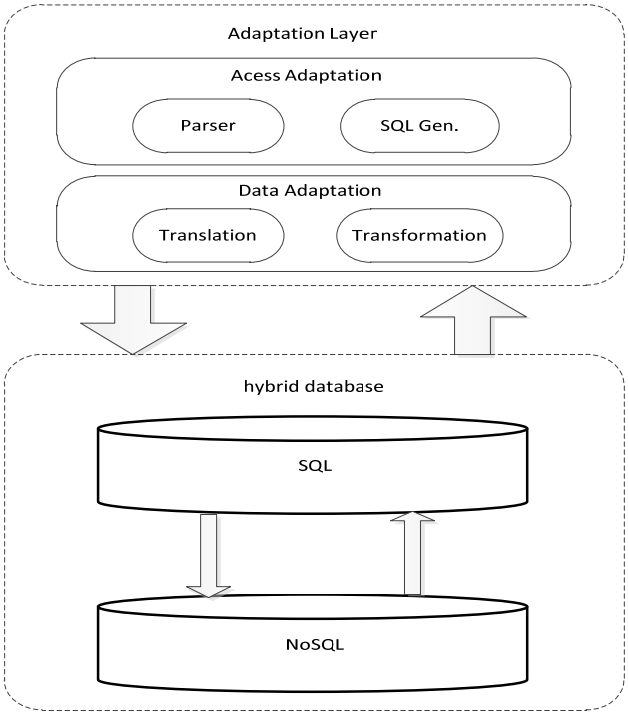

图 1 数据库抽象架构

该抽象层采用的具体的方法是将 NoSQL 数据合并到 SQL 数据库中，此种合并并不是在物理上将 NoSQL 数据完全存入 SQL 数据库中，而是使 NoSQL 数据在 SQL 数据库中虚拟可得，直到查询执行时才去获取相应的 NoSQL 数据。开发人员只需要打开一个连接到 SQL 数据库，通过此连接可以读取 NoSQL 数据。该方法最大的挑战是在 SQL 数据库中，如何表示非结构化的 NoSQL 数据，并在其后使开发人员能够轻松的构建查询来读取 NoSQL 数据源中的数据。

上述将 SQL 数据库作为主数据库来包含 NoSQL 数据的方案有很大的优势。首先，关系数据库和 SQL 是一个既定的数据存储的标准。开发人员对关系型数据库和 SQL 的熟悉度使得将关系型数据库作为混合数据库的主数据库是一个很好的选择，很容易被接受。此外，SQL 数据库有良好的性能和稳定性，许多关系数据库也比较容易扩展，提供了整合外部数据的机制。

2.2 NoSQL 数据的形式化表示

(1) 用三元组来表示 NoSQL 数据

不同的 NoSQL 数据类型采用不同的数据结构。图形数据库用图来表示数据，键-值存储用哈希表来存储数据，而面向文档的数据库则包含嵌套的键-值对集合[5]。然而所有的数据表示方法有一个共同点，即数据实体的每一部分与另一部分以特定的方式相关。

受 RDF[6] (Resource Description Framework)启发，将 NoSQL 数据表示为三元组，通过三元组来描述数据的不同对象如何同另一个对象相关联。给定两个数据对象 s 和 o，可以用一个三元组(s,p,o)来表示对象 s 通过关系 p 关联对象 o。使用三元组的概念可以将非关系型数据转换为适合 SQL 存储的关系型数据的表示形式，即用三元组的 p 用来表示一个属性的名称，o 包含这个属性的值，s 则表明多个三元组彼此联系，是属于相同数据实体的不同属性。在本文中，将(s,p,o)-三元组表示为(id,key,value)-三元组。

(2) 在 SQL 数据库中 NoSQL 数据的可用性

用三元组表示 NoSQL 数据如上所述，现在进一步说明如何将 NoSQL 数据库中的数据包含在 SQL 数据库中。先不讨论实现细节，假定在 SQL 数据库中存在关系 F(id,key,value)，包含所有代表 NoSQL 数据的三元组。例如，假设 NoSQL 数据集中的车辆集合对象包含速度 velocity 属性，则下面的查询会选择 NoSQL 数据集中所有车辆对象的 velocity 值：

$$\rho_{R(velocity)}(\pi_{value}(\sigma_{key=velocity}(F))) \qquad (1)$$

在 SQL 数据库中关系 F 可以像任何其他关系一样在查询中使用。即可以连接关系 F 到 SQL 数据库其他正常的关系，来实现在单一的查询结果中组合 SQL 和 NoSQL 数据。然而要注意的是关系 F 中的记录是从外部数据源实时获取的，现在需要从 NoSQL 数据库中获取。为了返回代表 NoSQL 数据的关系 F 的记录，SQL 数据库必须从 NoSQL 数据库中以三元组的形式获取其数据。

(3) NoSQL 数据转换为三元组

为了用三元组来填充关系 F，NoSQL 数据必须转换为三元组。NoSQL 数据的表示形式可一般化为嵌套的键-值对结构，其他的 NoSQL 数据形式可以很容易地转换为这种结构。下面介绍嵌套的键-值对结构的转换方法，嵌套的键-值对结构如下所示：

$$\{k_1 : v_1, k_2 : v_2, ..., k_n : v_n\} \quad (2)$$

这里任意 vi 的值可以又是一个键-值对的集合，注意同一嵌套级别上的 key 值必须是唯一的。为了方便起见，将数组形式的 vi 值表示为 key 值为连续整数的键值对的集合。如下所示，将一个数组处理为一个嵌套集合：

$$k_i : [v_{i,1}, v_{i,2}, ..., v_{i,n}] \equiv k_i : \{0 : v_{i,1}, 1 : v_{i,2}, ..., n-1 : v_{i,n}\} \quad (1)$$

为了把嵌套的键-值对结构转换成三元组，必须确保属于同一数据结构 s 的三元组有相同的 id 值，此 id 的具体值并不重要，只要属于相同 NoSQL 数据对象的三元组或有相同的嵌套级别的三元组的 id 值相同即可。嵌套的键-值对的集合需进行递归转换，为了组合嵌套的键-值对，需增加一个三元组关系，它的 value 值是子嵌套键-值对的 id 值。

以下提供转换公式，转换通过两个函数来实现 Φ 和 ψ，s 表示嵌套的键-值对结构，p 表示其中的一个键值对，函数下标表示同一级的三元组有相同的 id 值 i，函数 Φi 用来转换键值对，ψi 用来转换嵌套的键-值对：

$$\phi_i(p) = \begin{cases} \{(i, p_k, p_v)\}, & p_v \text{ 为常量} \\ \{i, p_k, j\} \cup \psi_j(p_v), & p_v \text{ 为集合} \end{cases} \quad (2)$$

$$\psi_i(S) = \bigcup_{p \in S} \phi_i(p) \quad (3)$$

在函数 Φi 中定义了变量 j，表示嵌套子集的 id 值，通过上面两个函数可以将任意嵌套的键值对结构转换为三元组。

2.3 查询处理策略

(1) 数据重建

通过对转换后的三元组进行一系列关系代数的自连接来实现重建 NoSQL 数据，如下例，见表 1 为原始数据：

表 1 普通对象数据表

No.	city	velocity
1	changsha	64
2	zhuzhou	35

表 2 为用转换公式进行转换后的三元组关系数据，如下：

表 2 转换后的三元组表

id	key	value
i1	No.	1
i1	city	changsha
i1	velocity	64
i2	No	2
i2	city	zhuzhou
i2	velocity	35

由同一 NoSQL 对象转换后的三元组必须要通过组合相关的记录来重建原始数据。如前所述，具有相同 id 值的记录是属于同一 NoSQL 对象的属性，见表 2，id 值为 i1 和 i2，为了组合 NoSQL 对象的 No.，city，velocity 信息，需要进行两次连续的自连接，连接时要保证其 id 值相同。下面通过重命名和 θ 连接来得到下面三个关系 T_1，T_2，T_3：

表 3 T1

No.
1
2

表 4 T2

No.	city
1	changsha
2	zhuzhou

表 5 T3

No.	city	velocity
1	changsha	64
2	zhuzhou	35

关系 T_1，T_2，T_3 分别如表 3、表 4、表 5 所示，是表 2 中的三元组通过关系代数表达式重建的。表达式通过重命名和适当的 θ 连接来重建数据并将原始数据以简单的关系数据的形式来输出，下面给出连接对应的关系代数表达式：

$$T_1 = \rho_{T_1(No.)}(\pi_{v_i}(\sigma_{k_i=No.}(\rho_{T_i(i,k_i,v_i)}(T)))) \quad (4)$$

$$T_2 = \rho_{T_2(No.,city)}(\pi_{v_i,v_n}(\sigma_{k_i=No.\wedge k_n=city}(\rho_{T_i(i,k_i,v_i)}(T) \bowtie \rho_{T_n(i,k_n,v_n)}(T)))) \quad (5)$$

$$T_3 = \rho_{T_3(No.,city,date)}(\pi_{v_i,v_n,v_o}(\sigma_{k_i=No.\wedge k_n=city\wedge k_o=date}(\rho_{T_i(i,k_i,v_i)}(T) \bowtie \rho_{T_n(i,k_n,v_n)}(T) \bowtie \rho_{T_o(i,k_o,v_o)}(T)))) \quad (6)$$

注意关系 T3 即是表 1 中的原始数据。本例中，由转换后的三元组重建原始数据对象对其进行了两次自连接。一般来说，重建有 n 个属性的数据需要进行 n-1 次自连接，模型会将这个过程自动化。

(2) 复合查询语言

要将重建 NoSQL 数据的过程自动化，需要对查询语言进行扩展。正如前面受 RDF 启发将 NoSQL 数据表示为三元组，同理，参考 RDF 的查询语言 SPARQL 来描述如何查询三元组。参照 SPARQL 中变量绑定的概念和图形模式的概念[7]，本文中相应的将复合查询语句中描述 NoSQL 数据的查询部分叫做 NoSQL 查询模式，它描述查询哪部分的 NoSQL 数据，剩余部分则是普通的 SQL 语句，为了更方便的把查询语句中的 NoSQL 查询部分转换为等价的 SQL 查询，将查询语句中的 NoSQL 查询部分独立为一个关系，如下例：

s = {number : 1, location : [121.56, 31.36]}

将数组转换成键值对后，此结构等价于下面的嵌套键值对结构：

s = {number: 1, location: {0 : 121.56, 1 : 31.36}}

将 s 由转换公式 ψi(s)转换后，结果如下表：

表 6 s 的三元组关系

id	key	value
i1	number	1
i1	location	i2
i2	0	121.56
i2	1	31.36

此结构 s 的 NoSQL 查询模式为：

```
(
  number: ? number,
  location: (
    0: ?f,
    1: ?s
  )
)
```

以每个括号为标志，为一个 NoSQL 查询模式。在连接三元组时，处于同一 NoSQL 查询模式的键值对被看作为有相同的 id 值的三元组，可进行组合，比如本例中的键值 number 和 location。注意 NoSQL 查询模式的概念和嵌套的键-值对结构的概念很相似，相对于集合，这里使用括号，并允许 key 或 value 值被变量代替。相对于将 NoSQL 数据转换为三元组，对于查询语言来说，是一个相反的过程，即使用嵌套的键-值对结构来描述想查询哪些三元组和它们如何关联。

NoSQL 查询模式描述要查询哪部分 NoSQL 数据。复合查询语言中通过使用变量绑定，允许 NoSQL 数据中的变量在查询的其他部分被使用，即在查询的其余部分可引用 NoSQL 查询模式中的变量。一个简单的查询语言如下：

```
SELECT
  r. number
FROM
  NoSQL(
    number: ? number
    location: (
      0: ?f,
      1:?s
    )
  ) As r
WHERE
  r.f = 121.56
```

其中 NoSQL 查询模式以 NoSQL 关键字来标识，其中的变量可以像 SQL 查询中关系的属性一样被引用，如上例中的 r. number 和 r. f。

(3) 查询转换的代数实现

NoSQL 查询模式是一个嵌套的键值对集合，将此结构表示如下：

$$NoSQL(k_1:v_1, k_2:v_2, \ldots, k_r:(k_{r,1}:v_{r,1}, k_{r,2}:v_{r,2}, \ldots, k_{r,m}:v_{r,m}) \ldots, k_n:v_n)$$

如转换中提到键值对 t，则是指具体的键值对 $k_t : v_t$。因在转换中需要对三元组关系 F 进行多次自连接来重建 NoSQL 数据，所以关系 F 会被拷贝多次。因此引入下标 t 来区分关系 F 的不同实例，如下：

$$F_t = \rho_{F_t(i_t,k_t,v_t)}(F)$$

这避免了在用关系 F 进行 θ 连接时由于重叠的关系名和属性名所引起的名字冲突，用关系 F 的每一个拷贝 F_t 来代表一个单独的键值对 t。根据键值对的 key 值和 value 值的类型，在 SQL 语句中，可以加一个选择操作来排除 F_t 上的不必要的三元组。即在进行其他关系代数运算之前，

过滤掉关系数据库中多余的三元组。对常量 key 值可以加上条件 key=k_t。类似的，如果 v_t 是常量，则可以在 F_t 上加上 value=v_t 的选择条件。对每个嵌套结构下的键值对 $k_{t,r}$:$v_{t,r}$，用关系 $F_{t,r}$ 来代表，同样，如果 $k_{t,r}$ 或 $v_{t,r}$ 是个常量，可以增加选择条件。注意因为 NoSQL 查询模式中的 key 值在每个嵌套的键值对中唯一，所以每个 F_t 通过逐层展开父节点的 key 名能够被唯一命名，因此通过三元组中唯一的 id 值和 key 值能正确的重建 NoSQL 数据。

下面使用关系代数来描述 NoSQL 查询模式的代数转换，此转换适用于任意 NoSQL 片断：

1）非嵌套模式

在考虑更复杂的 NoSQL 片断之前，先来关注带有 n 个 key 值的非嵌套的 NoSQL 模式：

$$\mathrm{NoSQL}\left(k_1:v_1,k_2:v_2,...,k_n:v_n\right)\mathrm{AS\ r}$$

为每个键值对 t 引进一个关系 F_t 意味着要连接这些关系：F_1，F_2，…，F_n。它们是属于同一 NoSQL 对象的三元组，这些关系的 id 属性有相同值，因为 NoSQL 片断非嵌套，且来自同一 NoSQL 对象的关系 F 有相同的 id 值。用关系代数来表示即为：

$$F_1\infty_{i_1=i_2}F_2\infty_{i_2=i_3}\cdots\infty_{i_{n-1}=i_n}F_n \qquad (7)$$

2) 嵌套模式

关于包含嵌套结构的 NoSQL 片断转换方法，整个过程递归进行即可，只需要正确的连接嵌套三元组关系和其他关系。考虑如下包含嵌套结构的例子：

$$\mathrm{NoSQL}\left(...,k_r:\left(k_{r,1}:v_{r,1},k_{r,2}:v_{r,2},...,k_{r,n}:v_{r,n}\right)...\right)$$

考虑将 NoSQL 数据转换为三元组的过程，这里只需为嵌套的集合增加一个连接条件，即嵌套数据的 value 属性和其子结构的 id 值相同：

$$\cdots\infty_{i_{r-1}=i_r}F_r\infty_{v_r=i_{r,1}}\left(F_{r,1}\infty_{i_{r,1}=i_{r,2}}\cdots\right)\infty_{i_r=i_{r+1}}\cdots \qquad (8)$$

省略号表示其余部分采用非嵌套模式的转换方法。上式与非嵌套结构唯一不同的是多了一个连接条件 vr=ir,1。注意因为嵌套关系 Fr,t 有相同的 id 值，所以增加这一个连接条件已经足够。

2.4 框架概览

图 2 描述了完整的查询流程，其包含前面小节中的所有步骤。

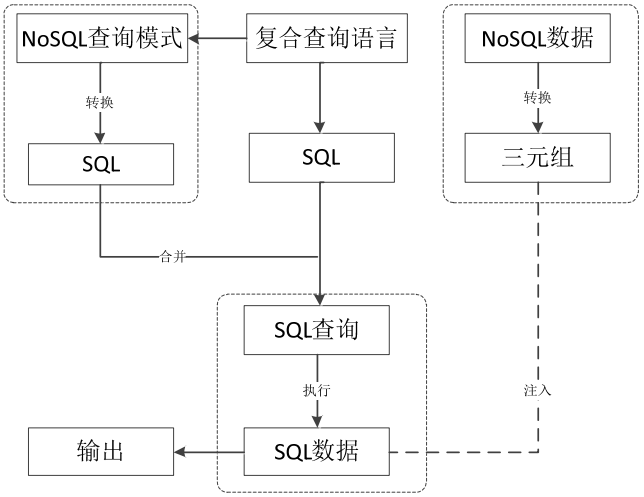

图 2 统一查询模型的工作处理流程

最终转换后的 SQL 所查询的结果是用户查询的最终结果，接下来描述理论框架的具体实现。

3．理论框架的原型实现

有了完整的理论框架和具体的查询处理策略，接下来进行具体的实现。

3.1 NoSQL 数据转换为三元组

本文选择 PostgreSQL 和 MongoDB 作为原型实现的 SQL 和 NoSQL 数据库。接下来要实现从 MongoDB 获取 NoSQL 数据，并将其转换为三元组关系 F 输出到 PostgreSQL 中，为实现这一转换，采用 PostgreSQL 的外部数据包装器机制，最终使用 MULTICORN 这一具体实现来完成理论原型中的三元组关系 F 的转换。

(1) 外部数据包装器

一种将外部数据加载到 PostgreSQL 中的方法是使用外部表[8]。从用户的角度来看外部表像关系数据库中的其他关系表一样。它有一个模式，包含数据记录，可以没有任何限制地被用于选择查询中，唯一的不同是外部表不可以被写。技术上，外表部是通过外部数据包装器来实现的[9]。PostgreSQL 的社区有志愿者写了很多外部数据包装器作为 PostgreSQL 的扩展[10]。

在本理论框架中需要 MongoDB 的外部数据包装器。主要的任务是将外部数据源通过外部表表示为一个关系数据库中的关系，这样就可以通过查询外部表来访问外部数据。在创建外部表的时候可以向其传递选项参数，允许用户为多个外部表重用一个外部数据包装器。大部分选项通常是设置连接参数的，比如外部数据源的网络位置等。

(2) Multicorn

一个具体的外部数据包装器的实现是 Multicorn，它将一个 PYTHON 脚本的输出作为它的外部数据源，

PYTHON 脚本可以通过函数实现来构建它的输出。即我们的工作是写一个 PYTHON 脚本来访问外部数据并将处理后的结果记录输出到 Multicorn，Multicorn 再将数据转发到 PostgreSQL 中。Multicorn 可作为一个 PostgreSQL 的普通扩展来安装，它具体使用哪个 PYTHON 脚本是在创建外部表的时候通过选项参数来指定的。

(3) 用 Multicorn 实现转换公式

下面用 Multicorn 来实现在 1.2 节中提出的将 NoSQL 数据转换为三元组的递归转换公式。在包含必要的库函数并初始化 ForeignDataWrapper 后，实现函数 ϕ 和 ψ 的关键代码如下：

```
# ϕ 函数
def process_dict(self, i, d):
    return reduce(lambda x, y: x + y, map(lambda (k, v): self.process(i, str(k), v), d. iteritems()), [])

# 处理列表
def process_list(self, i, l):
    return reduce(lambda x, y: x + y, map(lambda (k, v): self.process(i, str(k), v), enumerate(l)), [])

# ψ 函数
def process(self, i, k, v):
    row = {
        'id': '_'.join(i),
        'key': k
    }

    # 产生新的唯一 id
    j = i + [k]

    # 分类处理
    if type(v) is dict:
        # 处理嵌套集合
        row['value'] = '_'.join(j)
        return [row] + self.process_dict(j, v)
    elif type(v) is list:
        #处理列表
        row['value'] = '_'.join(j)
        return [row] + self.process_list(j, v)
    else:
        #处理非嵌套集合
        row['value'] = v
        return [row]
```

3.2 NoSQL 查询模式转换为 SQL

前面 1.3 节中描述了复合查询语言中 NoSQL 查询模式转换为纯 SQL 的代数实现，接下来通过编程语言将这一转换过程自动化。主要思想是，在 PostgreSQL 执行最终的 SQL 查询语言之前加入处理 NoSQL 查询模式的逻辑层。如果 NoSQL 查询模式中包含嵌套结构中的键值对共有 n 个键值对，则需要对三元组关系进行 n 次拷贝和 n-1 次连接，为防止转换后的结果中关系 F 的拷贝出现命名冲突，关系 F 的拷贝以键名来作为标识，如果遇到嵌套键值对，则键名遵循嵌套结构按层次叠加，即可保证关系 F 的拷贝命名的唯一性，关系 F 的拷贝相关的属性重命名也遵循此原则。下面是实现转换的关键程序逻辑：

(1) 预处理复合查询语言中的 NoSQL 查询模式，将查询模式中所有键值对保存于 php 数组$nosqlArray 中，该数组中键值对的结构与 NoSQL 查询模式中的键值对结构保持一致。

(2) 将$nosqlArray 数组和重命名关系 F 的前缀作为转换函数 process 的参数传入进行处理，process($nosqlArray,$pre)函数处理流程如下：

Step 1：逐一处理$nosqlArray 数组中的键值对

```
foreach ($nosqlArray as $key => $item)
{
    if ($item 为数组)
    A. 将此键值对的$key 值与前缀$pre 相连接，作为对应关系 F 的拷贝的重命名
    B. 加入嵌套模式的连接条件
    C. 递归调用 process($item,$pre.$key.'_')函数处理$item
    else ($item 非数组)
    将此键值对的$key 值与前缀$pre 相连接，作为对应关系 F 的拷贝的重命名
}
```

Step 2：增加同一嵌套级别的连接条件，即同层级的键值对对应的关系 F 的拷贝的 id 属性值相等

Step 3：按关系 F 的重命名规则重命名各关系 F 的拷贝对应的属性

Step 4：连接经处理后的各 sql 子句并返回

经过上述步骤的处理，最终将 NoSQL 查询模式转换为进行多次三元组关系 F 的 θ 连接的 SQL 语句，process 函数最后返回 NoSQL 查询模式经转换后的 SQL 语句。

4 模型验证

为了验证统一查询模型理论框架及其实现的正确性和有效性，下面用该原型进行实验，观察分析实验结果。

4.1 实验环境

表 7 标明了该实验选用的操作系统，关系型数据库和非关系型数据库及扩展库的版本情况。

表 7 软件版本表

操作系统	ubuntu12.04-server
关系型数据库	postgreSQL9.1
非关型数据库	mongodb-linux-x86_64-2.2.0
外部数据包装器	multicorn0.0.9

基于湖南省交通厅十二五规划的交通信息化基础支撑体系工程项目，实验数据通过对湖南省某市某时段 10000 多辆常规浮动车 GPS 信息进行建模采集。浮动车向交通信息中心传输的数据主要包括：车载终端 ID 号、经纬度坐标、瞬时速度、方向、回传时间、车辆运行状态等字段，这部分数据增长快，数据量大，存于 MongoDB 中。而对应的车辆信息、驾驶员信息等，因其增长慢，结构稳定，存于 PostgreSQL 中。

4.2 实验分析

对分别存储在 MongoDB 中的 NoSQL 数据车辆实时 GPS 信息和存储在 PostgreSQL 中的 SQL 数据车辆信息通过现有模型进行复合查询，注查询针对需同时获取两部分数据源的情况进行。

(1) 实验数据

表 8 中的 n 表示存储在 MongoDB 中表示车辆实时 GPS 信息的文档数，取 l,m,n 三个值。

表 8 nosql 数据规模

n	nosql 文档
l	100000
m	200000
h	400000

表 9 中的 s 表示存储在 PostgreSQL 中表示车辆实体的记录数，取 l,m,n 三个值。

表 9 sql 数据规模

s	sql 记录
l	1000
m	5000
h	10000

S(n,s)表示对 nosql 数据规模为 n, sql 数据规模为 s 的数据进行组合查询，即实验数据大小为 n 个 nosql 文档和与之关联的 s 条 sql 记录。

(2) 实验结果

对 S(n,s)的 9 种数据规模的数据集合的查询性能，以及复合查询语言进行转换所需要的转换时间指标进行分析对比。

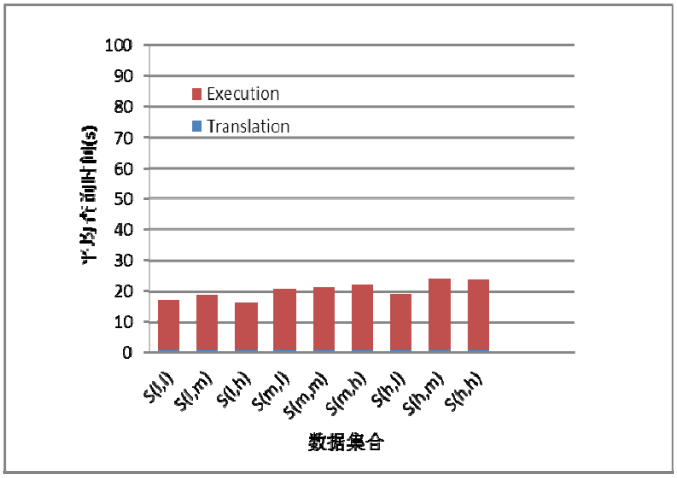

图 3 每个数据集合的查询性能

由图 3 可见，首先对每种数据组合而言，用户查询的转换时间相对于查询处理时间，只占很小一部分，且耗时比较稳定。模型处理的大部分时间都是用在获取结果数据上。其次，每个数据集合的平均查询时间相差不大，说明在一定的数据规模内，该查询模型的查询时间较稳定。

5 结束语

本文提出了一种基于 SQL 数据库和 NoSQL 数据库的统一查询模型，该查询模型理论上适用于任一种 SQL 数据库和 NoSQL 数据库，并选定 PostgreSQL 和 MongoDB 对理论框架进行了完整实现，通过理论框架提出的嵌入 NoSQL 查询模式的类 SQL 语句在复合语言查询分析器中的执行，实现了统一查询 MongoDB 中的 NoSQL 数据和 PostgreSQL 中的 SQL 数据，并且不同数据源的数据可以进行任意连接、组合。但面对海量数据的查询，该查询模型的查询效率还需进一步改进和优化。总的来说，该统一查询模型提出的理论框架是可行的，达到了预期的效果。如果遇到本文没有涉及的论文格式问题，作者可首先参阅本届大会网页提供的“英文论文格式”要求。若“英文论文格式”也没有涉

及该问题，作者可以采纳其它学术论文的惯例。

参考文献(References)

[1] Stonebraker M. SQL databases v. NoSQL databases. Communications of the ACM, 2010, 53(4): 10-11.

[2] Rick Cattell. Scalable SQL and NoSQL Data Stores.ACM Special Interest Group on Management of Data. New York, 2010: 12-27.

[3] Brynko B. NuoDB: Reinventing the Database. Information Today, 2012, 29(9): 9-9.

[4] Stonebraker M. New sql: An alternative to nosql and old sql for new oltp apps. 2011.

[5] Rick Cattell.Scalable SQL and NoSQL Data Stores.SIGMOD record,2010,39(4):12-27.

[6] Valer H, Sauer C, Härder T. XQuery processing over NoSQL stores. Liebe Teilnehmerinnen und Teilnehmer, 2013.

[7] Roijackers J. Bridging SQL and NoSQL. MSc Thesis, Eindhoven University of Technology, 2012.

[8] Wu-Chun Chung, Hung-Pin Lin, Shih-Chang Chen, Mon-Fong Jiang, Yeh-Ching Chung. JackHare: a framework for SQL to NoSQL translation using MapReduce. Automated Software Engineering,2013(9): 153-170.