关系模型的实质

关系模型的实质

每当我批评关系式数据库，就会有人说，SQL和关系式数据库的设计，其实偏离了E.F.Codd最初的关系式理论。关系式理论和关系式模型，本身还是很好的，只不过被人实现的时候搞砸了。如果你看透了本质，就会发现这只是一个托词。关系式数据库的问题是根源性的，这个问题其实源自关系式理论本身，而不只是具体的实现。

人们总是喜欢制造这些概念上的壁垒，用以防止自己的理论受到攻击。把过错推到SQL或者IBM身上，是关系式数据库领域常见的托词，用以掩盖其本质上的空洞和设计上的失误。在下面的讨论里为了方便，我会使用少量SQL来表示关系模型里面对应的概念，但这并不削弱我对关系模型的批评，因为它们表示的是关系式模型里面的核心概念。

关系式模型与数据结构

很多人把关系式理论和数据库独立开来，认为它们是完全不同的领域。而其实，数据结构的理论，可以很容易的解释所有关系式数据库里面的操作。

关系模型的每一个“关系”或者“行”（row），表示的不过是一个普通语言里的“结构”，就像C语言的struct。一个表（table），其实不过是某种结构的数组。举个例子，以下SQL语句构造的数据库表：

```
CREATE TABLE Students ( sid CHAR(20),
                        name CHAR(20),
                        login CHAR(20),
                        age INTEGER,
                        gpa REAL )
```

其实相当于以下C语言的结构数组：

```
struct student {
  char* sid;
  char* name;
  char* login;
  int age;
  double gpa;
}
```

每一个“foreign key”，其实就是一个指针。每一个join操作，本质上就是对指针的“访问”，找到它所指向的对象。在实现上，join跟指针引用有一定差别，因为 join需要查“索引”（index），所以它比指针引用要慢。

所谓的查询（query），本质上就是函数式语言里面的filter, map等操作。只不过关系式代数更加笨拙，组合能力很弱。比如，以下的SQL语句

```
SELECT Book.title
 FROM Book
 WHERE price > 100
```

其实相当于以下的Lisp代码：

```
(map .title
     (filter (lambda (b) (> (.price b) 100)) Book)
```

关系式模型的局限性

所以关系模型所能表达的东西，其实不会超过普通程序语言所用的数据结构，然而关系模型，却具有比数据结构更多的局限。由于“行”只能有固定的宽度，所以导致了你没法在里面放进任何“变长”的对象。比如，如果你有一个数组，那你是不能把它放在一个行里的。你需要把数组拿出来，旋转90度，做成另一个表B。从原来的表A，用一个“foreign key”指向B。这样在表B的每一行，这个key都要被重复一次，产生大量冗余。这种做法通常被叫做normalization。

这种方法虽然可行，其实它只不过是绕过了关系式模型无须有的限制。类似这样的操作，导致了关系式数据库的繁琐。说白了，normalization就是在手动做一些比C语言的手动内存管理还要低级的工作。连C这么低级的语言，都允许你在结构里面嵌套数组，而在关系式模型里面你却不能。很多宝贵的人力，就是在构造，释放，连接这些“中间表格”的

工作中消磨掉了。

另外有一些人（比如这篇文章）通过关系模型与其它数据模型（比如网状模型之类）的对比，以支持关系模型存在的必要性，然而如果你理解了这小节的所有细节就会发现，使用基本的数据结构，其实可以完全的表示关系模型以及被它所“超越”的那些数据模型。这些所谓“数据模型”其实全都是故弄玄虚，无中生有。数据模型可以完全被普通的数据结构所表示，然而它们却不可能表达数据结构带有的所有信息。这些模型之所以流行，是因为它们让人误以为知道了所谓的“一对一”，“一对多”等冠冕堂皇的概念，就可以取代设计数据结构所需要的技能，所以我认为数据模型本身就属于技术上的“减肥药”。

NoSQL

SQL和关系模型所引起的一系列无须有的问题，终究引发了所谓“NoSQL运动”。很多人认为NoSQL是划时代的革命，然而在我看来，它最多可以被称为“不再愚蠢”。大多数NoSQL数据库的设计者，并没有看到上述的问题，或者他们其实也想故弄玄虚，所以NoSQL数据库的设计，并没有完全摆脱关系模型，以及SQL所带来的思维枷锁。

最早试图冲破关系模型和SQL限制的一种技术，叫做“列模式数据库”（column-based database），比如Vertica, HBase等。这种数据库其实就是针对了我刚刚提到的，关系模型无法保存变长结构的问题。它们所谓的“列压缩”，其实不过是在“行结构”里面增加了对“数组”的表示和实现。很显然，每一个数组需要一个字段来表示它的长度N，剩下的空间用来依次保存每一个元素，这样你只需要一个key就可以找到数组里所有的元素，而不需要把key重复N遍。

这种“巧妙”的实现，也就是你在列模式数据库的广告里看到的，只不过那些广告把它说得天花烂坠，貌似与众不同而已。在列模式数据库里，你不需要进行normalization，也不需要重复很多key。这种对数组的表示，是一开始就应该有的，却被关系模型排除在外。然而，很多列模式数据库并没有看到这一实质。它们经常设定一些无端的限制，比如给变长数组的嵌套层数作出限制，等等。所以，列模式数据库，其实没能完全逃脱关系式数据库的思想枷锁。如此明显的事情，数据库专家们最开头恁是看不到。到后来改来改去改得六成对，还美其名曰“优化”和“压缩”。

最新的一些NoSQL数据库，比如Neo4j, MongoDB等，部分的改善了SQL的表达力问题。Neo4j设计了个古怪的查询语言叫Cypher，不但语法古怪，表达力弱，而且效率出奇的低，以至于几乎任何有用的操作，你都必须使用Java写“扩展”（extension）来完成。MongoDB使用JSON来表示查询，本质就是手写编译器里的语法树（AST），非常奇葩和苦逼。现在看来，数据库的主要问题，其实是语言设计的问题。NoSQL数据库的领域，由于缺乏经过正规教育的程序语言专家，而且由于利益驱使，不尊重事实，所以会在很长一段时间之内处于混沌之中。

其实数据库的问题哪有那么困难，它其实跟“远过程调用”（RPC）没什么两样。只要你有一个程序语言，你就可以发送这语言的代码，到一个“数据服务器”。服务器接受并执行这代码，对数据进行索引，查询和重构，最后返回结果给客户端。如果你看清了SQL的实质，就会发现这样的“过程式设计”，其实并不会损失SQL的“描述”能力。反而由于过程式语言的简单，直接和普遍，使得开发效率大大提高。NoSQL数据库比起SQL和关系式数据库，存在某种优势，也就是因为它们在朦胧中朝着这个“RPC”的方向发展。