关系式模型的实质

每当我调侃关系式数据库，就会有人说，SQL 和关系式数据库的设计偏离了 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先进的，只不过实现的时候被人搞砸了。

我很悲哀，因为如果你看透了关系式理论（模型/代数）本身，就会发现关系式数据库的问题是根源性的：关系式理论本身就是空洞而虚浮的，它是一个披着”数学”外衣的噱头，是潜伏在大学计算机系课程里几十年之久的无稽之谈。

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身上，说 SQL 没有忠实的实现关系式理论的精髓，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。在下面的讨论里为了方便，我会使用少量 SQL 来表示关系式模型里面对应的概念，但这并不削弱我对关系式模型的批评，因为它们表示的是关系式模型里面等价的核心概念。

关系式模型与数据结构

很多人把关系式理论和数据结构（data structure）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。

关系式模型的每一个“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，或者 Java 的 class。一个表（table），不过就是某种结构的数组（比如 `Student[]`）。举个例子，以下 SQL 语句构造的表：

```
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;
}
```

每一个数据库的“key”，本质和 C 语言的指针是一回事，就像 `char* p`。所谓“join”操作，就是对指针的“访问”（dereference），得到指针指向的对象，就像 C 语言里写 `*p`。在实现上，join 跟指针访问有一定差别，因为 join 需要用软件查“索引”（index），所以它比指针访问要慢很多。

数据库所谓的查询（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
```

表达的东西相当于以下的 Lisp 代码：

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

但 SQL 的嵌套组合能力和一致性都要比 Lisp 差很多。很多你认为应该自然可以表达的查询，SQL 表达不了，折腾很久才发现得用很蹩脚的方式表达。嵌套的查询经常是个问题，需要扩展 SQL 的语法才能实现，而 Lisp 天生可以优雅地表达任意的嵌套和组合。

不可否认，某些 SQL 底层实现对基本查询的实现或许更加高效，然而其实 Lisp 的底层运行系统也可以采用类似的高效实现。我们不应该把“底层实现”和“上层概念”混淆起来。

一个糟糕的概念可以被实现得很快，然而概念本身仍然是糟糕的，用起来痛苦，莫名其妙。一个优雅的设计也许被实现得很低效很慢，但聪明人看到了它概念上的优势，可以改变底层实现，做出很高效的系统。实际上已经有人实现了这样的数据库系统，它用类似这里的 Lisp 方式来表达查询。

关系式模型的局限性

所以关系式模型所能表达的东西，不会超过普通数据结构，然而关系式模型却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，如果你有一个动态长度的数组，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。更傻的是，在表 B 的每一行，这个 key 都要被重复一次。数组有多长，这个 key 就需要重复多少次，占用大量不必要的空间。这种从数据结构角度看来极其愚蠢的做法，在数据库领域却被起了一个高深莫测的名字，叫做“normalization” ;)

类似这样的操作，组合在一起，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。许多宝贵的人力，耗费在构造，释放，连接这些“中间表格”的工作中。

另外有一些人（比如这篇文章）采用五十步笑一百步的做法，通过关系式模型与其它数据模型（Data Model，比如网状模型之类）的对比，以支持关系式模型存在的必要性。你说我关系式模型不好，看哪，还有更差的！如果你理解了这小节的所有细节就会发现，你完全可以使用基本的数据结构，表示关系式模型以及被它所“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，然而它们却不可能简单而完整的表达数据结构带有的信息。这些数据模型之所以流行，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”，告诉你要吃好几个疗程才会见效，最后还是不见效，那肯定是你自己什么地方操作错了 ;)

与其寄希望于这些贴着精美“数学”标签的减肥药，你不如去隔壁二流大学旁听一堂基础的数据结构课程 ;)

NoSQL

所以 E. F. Codd 的关系式理论（关系式模型，关系式代数）是这一切麻烦的祸根，而 SQL 只是它的一个小喽啰。人们用数据库遇到麻烦，一般都拿小喽啰开刀，骂 SQL，却给关系式理论制造各种托词。他们畏惧“代数”这样的术语。一个概念被冠以“关系式代数”这样的称呼，你是不敢骂它的，否则别人会说你不懂，学识太浅，理解不了“数学” ;)

关系式理论和它的小喽啰 SQL 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式模型以及 SQL 所带来的思维枷锁。

最早试图冲破关系式模型和 SQL 限制的一种技术，叫做“列存储数据库”（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，关系式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，一个数组放在存储设备里，需要一个字段来表示它的长度 N，剩下的空间依次保存每个元素。这样你只需要一个 key 就可以找到数组里所有的元素，而不需要进行 normalization，把 key 重复 N 遍。

这是每个初学编程的人存储数组的时候都会想到的做法，却被关系式模型排除在外。列存储数据库只不过是纠正了一个历史遗留的愚蠢错误，却把自己说成是重大的突破。 甚至很多列存储数据库也没有看到这一实质。它们经常存在一些无端的限制，比如给变长数组的嵌套层数作出限制，等等。所以，列存储数据库其实也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，以至于几乎任何实际的操作，你都必须使用 Java 写“扩展”（extension）来完成。MongoDB 等使用 JSON 来表示查询，本质就是手写编译器里的语法树（AST），不直观又容易出错。

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，由于缺乏负责的程序语言专家，而且由于利益驱使，急功近利，所以会在很长一段时间之内处于混沌之中，给使用者造成痛苦。

其实数据库的问题哪有那么困难，它跟“远过程调用”（RPC）没什么两样。只要你有一个程序语言，几乎任何程序语言，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”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优势，也就是因为它们在朦胧中朝着这个“RPC”的方向发展。

有些人说你这样直接编程不好，因为外存的管理，索引数据结构，都是很容易出错的代码，还是不如用数据库。可是谁告诉你一定要自己写外存管理和索引代码呢？你完全可以使用经过千锤百炼的代码库，把它们放在服务器上面做成一个“存储索引系统”，你的“查询代码”只需要发送过去调用这些代码库就可以了。

所以到现在，我的脑子里早已不存在“数据库”，“关系式”，“NoSQL”这样的概念，因为它们带来的更多是困扰，它们把本来简单的问题复杂化。在我的脑子里，只有更通用而简单的数据结构，以及针对它们的高效存储处理方式。