论骑车穿越美国与实现Postgres的相通之处

迈克尔·斯通布雷克(Michael Stonebraker)
美国麻省理工学院

关键词：数据库　设计与实现　Postgres

编者按：2015年10月22~24日，中国计算机学会在合肥召开了中国计算机大会(CNCC 2015)，现发表图灵奖得主斯通布雷克等8位专家的大会特邀报告，以飨读者。

我今天并行交错地讲两个故事。一个是我和我夫人在1988年夏天骑车穿越美国的旅行；另一个是从1984年到1995年，Postgres的商业版Illustra的设计、实现、商业化以及最后被收购的故事。

Postgres的设计

1988年6月4日，我的夫人贝丝和我从华盛顿州的阿纳科特斯(Anacortes)骑双人自行车出发，向东前往3500英里外的波士顿。虽然我们之前从来没有骑过双人车，骑行也从没有超过5天，也没有骑车爬过高山的经验，何况一路上我们还需要照顾18个月大的女儿，但是我们丝毫没有被这些困难吓倒，哪怕一点点，反而对这次“探险”感到非常兴奋。

图1　从华盛顿州出发骑行前往波士顿

1984年，商业化的Ingres已经4岁了，它的实现代码比学院版的Ingres要好太多，继续在学院版的Ingres上实现我们的一些想法和原型系统已经不现实了，因为它很难比得过商业版的Ingres。于是，我们做了一个痛苦的决定：抛弃所有的Ingres代码，重新设计并实现一个新的数据库系统Postgres。那么，Postgres应该是什么样的呢？

抽象数据类型

Ingres和System R是20世纪80年代初最主要的两个关系数据库原型系统，它们都完全聚焦于对商业数据的处理。然而当时在很多论文中都有这样的结论：“我们在关系数据库模型上实现了X，但是它的效果很差，于是我们在关系模型上加上了Y”。我很清楚，如果在关系模型上加上这些互不相关的扩充，将导致关系模型的消亡。我们需要更好的策

略来迎合非商业数据处理市场。

那个时候，有个地理信息研究小组想要使用Ingres系统，我们以这个为例。假设有一个带有位置范围信息的雇员信息数据库，如表1所示。一个常见的地理信息查询是："找到位置范围与矩形(14, 17, 0, 16)相交的所有雇员"。那么我们需要的SQL查询语句是这样的：

select name where
xmin>14 and xmax<17 and
ymin>0 and ymax<16 。

表1 带地理信息的雇员数据库

name	age	xmin	xmax	ymin	ymax
Sam	38	100	220	46	87

然而，这样的查询语句既复杂又很难优化。为了更好地优化这个查询，我们需要新的数据类型。然而当时的商业数据库只有字符串、浮点数、整数等基本数据类型，我的想法是：这些基本数据类型并不能很好地支持地理信息查询，如果使用一个"location"的属性来替代表示位置范围的4个属性(如表2所示)，那么，我们的SQL语句就可以简化为：

select name
where location overlaps MakeBox(14,17,0,16) 。

表2 使用抽象数据类型的雇员信息库

name	age	location
Sam	38	Internal representation for a box

为了支持这个功能，我们需要用户指定的函数(user defined function)、用户指定的类型(user defined type)和用户指定的操作(user defined operator)。这就是Postgres想要做的事情之一：支持抽象数据类型。虽然这个概念很直观，但是困难总是隐藏在细节中。为了支持抽象数据类型，我们需要新的索引结构来索引新的数据类型，例如R-tree；我们需要告诉查询优化引擎如何处理新的类型；我们需要对"非"操作进行优化等。

在骑行的第三天，我们到达了华盛顿州的温思罗普(Winthrop)。我们从早上5点开始骑行爬山，挣扎了15英里后，终于向上骑行了5000英尺，然而这个时候开始下雪了，我们穿上了所有的衣服依然感觉很冷。虽然如此，我们最终还是翻越了这座山，尽管后面还有很多很多的高山，但我依然非常乐观和高兴，因为翻过当前这座高山，证明我们有能力翻过其他高山。

用"always"关键字实现约束规则

1981年，克里斯·戴特(Chris Date)提出了完整性约束。完整性约束主要是处理主键约束和外键约束的。例如，表3所示的雇员表和部门表，其中雇员的部门属性是部门表的外键。如果我们使用SQL语句删除掉Candy部门，那么会发生什么呢？在Art的部门值上将会有个悬空的外键指针，顺着这个外键指针走将会产生错误。为了避免这种情况，戴特提出了三个合理的解决方案。第一个是级联删除，它将删除所有属于Candy部门的雇员，从而保证数据库的完整性；第二个是把指向Candy部门的外键替换为"null"，这会保证数据库的一致性；第三个是拒绝删除操作，因为这会破坏数据库的一致性。同样，在插入操作中也有对应的三种方案。戴特提出的解决方案在当时被商业数据库广泛采纳。

表3 雇员表和部门表

dname	floor	sq. ft.	budget
Shoe	3	500	40000
Candy	2	800	50000

name	dept	salary	age
Bill	Shoe	2000	40
Art	Candy	3000	35
Sam	Shoe	1500	25
Tom	Shoe	1000	23

在我看来，首先，任何一个新的数据库系统都必须实现外键约束，当然也包括Postgres。其次，在当时，触发器(trigger)引起了很多人的关注。如果用户想确保Sam和Bill的工资是相同的，那么对Sam工资的更新也必须级联到Bill上，这就需要一个触发器。触发器虽然和引用约束不同，但它也只

是更新操作的一个附属操作——在更新操作之后需要完成的。第三是主键约束，例如确保工资不会出现负数，这也是更新操作的一个附属操作。仔细想想就会发现，这仅仅是三个琐碎的规则系统。我的想法是实现一个统一的规则系统，它能处理所有的（至少上述三个）规则。

思考了一年左右，我提出的解决方案是在命令中增加一个“always”关键字来处理这些规则。在更新操作中附上“always”关键字将会保证后面的条件一直成立。例如，为了让 Sam 和 Bill 的工资保持一致，我们使用如下命令：

Update ALWAYS (set salary = E.salary)

Where E.name = ‘Sam’ and name = ‘Bill’。

这是一个比 If-Else 规则系统更加清楚简单的系统。

在第 15 天的时候，我们抵达了蒙大拿州的边境。我们一路下坡，之前的高山和石子路都被我们甩在了背后，一条一直通往芝加哥的平坦马路在我们面前延伸。

无重写存储

Postgres 的第三个计划是无重写存储。20 世纪 70 年代，大家都认识到了处理宕机恢复的重要性。为了确保用户绝对不会丢失他们的数据，我们必须非常认真地对待宕机恢复的问题。在当时，所有人采取的方式都是分开存储数据和日志。每次做更新操作的时候，需要把数据更新前和更新后的样子都写入到日志里。所以有两个数据存储地点，其中一个用来管理数据，另外一个用来管理日志，如图 2 所示。我们必须非常小心地维护这些数据和日志以使它们保持同步。

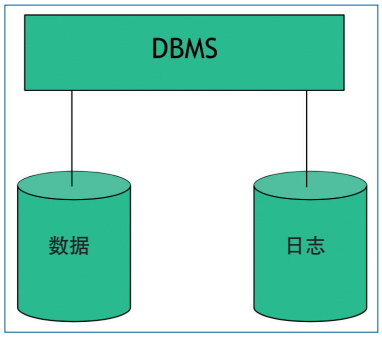

图2 数据与日志分开存储的数据库管理系统

当时我就意识到，处理日志的代码十分复杂而且丑陋，同时还要确保它能正确工作。否则，如果一个非常重要的客户的数据库系统宕机了，而你的日志并不能恢复他们丢失的数据，那你就彻底完蛋了。如果这位客户足够重要，你将会出现在《纽约时报》的头版上。我们必须有更好的方法来解决宕机问题。因此，我们提出统一存储日志和数据（如图 3 所示），并且在更新操作的时候，不覆盖当前的数据，我们做的仅仅是插入一条更新后的数据，并加上时间戳，这样就可以绕过宕机恢复，而且还能顺便实现“时间旅行”的功能。这看起来非常有趣，但是困难总是隐藏在细节当中。传统的数据库对日志的写操作和数据的读操作分别进行了优化，然而在我们的设计中，日志和数据混在一块，这些优化并不能直接套用，如何提高我们的数据库的性能是一个很大的挑战。

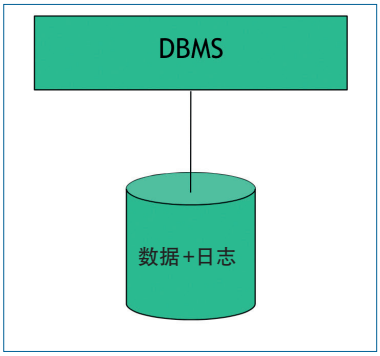

图3 数据与日志统一的无重写存储

以上就是 Postgres 的设计蓝图。抽象数据类型，用“always”关键字来处理规则以及无重写存储。当然我们还有其他计划，例如，复杂对象和继承等，但是前面三个是最重要的。

Postgres的实现

骑行到第 26 天的时候，我们到达了蒙大拿州东侧的北达科他州 (North Dakota)，这里是一望无际的平原，一棵树也看不到。美国中西部夏天的天气很奇怪，呼啸的狂风从东往西一直不停地吹。我们原本打算以 16~18 英里的时速轻松骑行，但事实上我们努力挣扎也只能是每小时 7~8 英里。我们当天只前进了 50 英里，比计划少了 25 英里。距离丘陵地带的明尼苏达州的边界线还有 250 英里，我们感到十分绝望，不知道如何才能走完这段路程。

1986年，C++语言还不成熟，所以最开始我们采用了Lisp语言来实现Postgres。可是事实证明，这是个灾难性的决定，因为它的性能比其他语言要低1个数量级。所以我们抛弃了非常多的代码，把可以转化的代码转化了，并重写了其他不能转化的代码。采用新语言这个实验造成了灾难性后果。

抽象数据类型工作得很好。我是从Ingres的一个用户那里明白这一点的。当时我还是Ingres的首席技术官，我接到一个客户的电话，他告诉我，我们在Ingres上实现的日期和时间的类型有错误。我觉得这不可能，我们是按照专家的指导实现的，即使不同的月份有不同的天数，但我们依然能够正确地处理日期类型。其实这位客户想要实现一个应用系统来计算金融债券利息，然而天知道为什么，华尔街的那帮家伙认为所有月份都是30天，每个月的利息都是一样的，不管这个月实际有多少天。例如，在他们看来3月15日距离2月15日是30天。这位客户想要的仅仅是跳过我们系统设定的日期减法，使用他们自己实现的日期减法。这对Postgres的抽象数据类型来说简直是小菜一碟，然而当时所有的商业数据库系统都不支持这个特性。在这之后，日期还有很多不同的理解方式，例如交易日和工作日等。

“时间旅行”很有趣，但是我们不知道如何才能提高它的性能，这一直困扰着我们。同时，“always”关键字并不能很好地工作。我们不知道如何用它来支持日期提出的全部6种约束，我们又抛弃了大量的代码，重新实现了传统的规则系统。

1987年到1990年的三年时间里，我们的进展非常缓慢。这一段时间我们就像是在泥潭和沼泽里艰难地爬行。我们做了很多修修补补的工作，例如，我们一直想弄清楚复杂对象到底能有多复杂以及如何处理规则系统。这个阶段过得很艰难，在Postgres的实现上和我们的骑行上都是。

Postgres的商业化

然而这个时候，奇迹突然出现了。这个奇迹和狂风无关，狂风依旧呼啸着。这个奇迹是我的兄弟出现了，他决定来陪我们骑行一个星期。他非常强壮，狂风并不能击倒他。他帮助我们破风，就像在自行车竞赛中看到的一样，我们跟在我兄弟的自行车后轮后面大概3英寸的地方骑行。我们终于看到了离开北达科他州的希望。

1991年，Ingres公司对我的竞业限制正好过期了，我们决定成立一个商业公司来对Postgres进行商业化。这样做的原因是，在我看来，市场是检验一个好想法的最终裁决者。如果你想实现科技转化，那么你有两种选择。第一种是让一个大公司采用你的科技，当时我们接触了Ingres公司，但是大公司有他们自己的日程安排。根据哈佛商学院的一位教授所写的书《创新者的窘境》(*The Innovators Dilemma*)，如果一个销售旧科技的公司要采用新的技术，他们很难避免原有客户的流失，所以对大公司来说，他们多少有些难以接受我的新技术。第二种科技转化的最好方式是建立一个初创企业，这就是我们所做的。我们雇用了一批非常优秀的工程师，他们中有一部分是从学术界过来的。我的朋友出任了临时首席执行官，我们通过融资获得了一笔资金。我们当时的计划是把查询语言从QUEL转化为当时的标准查询语言SQL，优化代码以及提升性能。就像所有的初创企业一样，我们当时非常兴奋。

在第38天的时候，我们穿越了威斯康星州，抵达了密歇根州的拉丁顿(Ludington)。我们距离波士顿只有不到1000英里了，上中西部已经在背后，我们看到了完成这趟探险旅程的希望。

1993年，在几次错误的尝试之后，我们最终把公司命名为Illustra。和所有商业化公司一样，我们需要做的第一件事情就是吸引用户。

我们成功地吸引了几个初始用户并融到了更多资金。最重要的是，我们雇用了一个真正的管理团队，包括首席执行官、主管市场的副总裁以及主管销售的副总裁。我们的公司走上了正轨。

但是好景不长。在骑行第49天的时候，我们到达了位于纽约州和宾夕法尼亚州交界处的埃利科特维尔(Ellicottville)。那天很不幸，我们出发时在酒店

的大理石地面上骑车滑倒，我的膝盖磕破了，我俩只剩下 3 条好腿来完成剩下的旅途。但是最糟糕的是我们遇到了一座高山。从威斯康星到密歇根再到俄亥俄州一路平坦，我们的骑行非常轻松，但是这一切随着这座高山的出现戛然而止。虽然这座山只有 500 英尺高，但是一路都是起起伏伏，我们感到筋疲力尽。

1994 年，我们遇到了 Illustra 的第一个严重危机。我们的很多用户想要使用大型应用商提供的一些抽象数据类型的库，于是我们和那些大型应用商谈判，看是否能把他们的第三方库融合进我们的数据库。然而他们并不关心这些，他们问我，你们现在有多少用户？你们能帮我们卖出去多少许可证？原来他们想要的仅仅是一个销售渠道，而不是根本的技术变革。这样一来，我们陷入了一个死循环：我们希望抓住客户，客户想用大型应用商的库，而大型应用商又要求我们给他带来更多客户。所以当时我们的产品销量并不好，而这又直接导致了另外一个危机。

一个初创企业从零开始融资，每一轮都有个估值，风险投资商会以一定的价格获得这个企业的一些股票。当初创企业的钱用光后需要再进行一轮融资，这时企业的估值通常会更高，股票的价格也会更高。但是如果新的一轮融资的估值变低了，这轮融资就叫做贬值融资。我们就经历了一轮贬值融资，这是比企业破产更糟糕的事情。为什么呢？因为融资合同里通常会有这么一个条款，如果企业遇到了贬值融资，那么之前所有的融资合同里股票的价格都要按照这轮贬值融资的股票价格重新计算。而对于员工来说，他们手上的股票遭到了双重稀释，他们可能会选择跳槽。这是件令人非常尴尬的事情，我绝不想再经历一次。我需要和公司所有的股东进行谈判，这花了我 3 个月的时间，直接导致了我们 3 个月毫无进展。在这之后，我们终于获得了一笔钱，使得 Illustra 能够继续存活一段时间。

Postgres被收购

在第 56 天的时候，我们抵达了马萨诸塞州的边境，我们绕过了那座连绵起伏的大山，横在我们面前的是抵达目的地前的最后一座山，再往后的路都十分好走。

1995 年，奇迹突然出现。互联网蓬勃发展，网络公司受到大家的热烈追捧。此时，我们的产品不再仅仅是一个拥有扩展数据类型的数据库，而是一个真正意义上的互联网数据库，可以存储互联网上所有类型的东西，例如文本、图像和音频。当时一个互联网巨擎考虑的是把我们的产品融入到他们的系统中，这会促成一笔交易。他们首先需要测试我们的产品，很遗憾他们并没有测试我们的产品检索文本或者地理信息的性能，他们要在一个测试事务处理性能的标准数据集上测试我们的产品，因为他们认为互联网上的每一个操作都对应一个事务，所以我们的产品需要高效的事务处理能力。然而事务处理并不是我们的系统所擅长的，如果要大幅度提升事务处理的性能，我们需要重写大部分的代码。

在第 59 天的时候，我们抵达了波士顿，我们把双人车骑行到大西洋边的海滩来结束我们的旅程。

1996 年 2 月，奇迹突然出现了。我们的一个竞争对手同时也是一家大公司，提出要收购 Illustra。这直接解决了我们前面的两个问题，首先解决了我们的一些客户需要大型应用商的第三方库的问题，因为大型应用商很乐意和这家用户量巨大的大公司合作。其次，这家大公司有非常高效的事务处理引擎，这解决了我们的产品事务处理的性能问题。被

收购之后，我们成功地把 Illustra 的很多特性移植到了这家大公司的系统中。

结论

我为什么要讲关于自行车骑行的故事呢？首先，我要证明我是会写代码的。这是骑行穿越美国的算法伪代码：

```
直到（大西洋）{
    起床；
    向东骑行；
    克服遇到的任何困难；
}
```

我们稍微泛化一下这个伪代码：

```
直到（目标完成）{
    起床；
    行动；
    克服遇到的任何困难；
}
```

我们把这段代码定义为一个宏：Make It Happen（动手实现）。

第二个要说明的是，为什么一个正常的人想要骑车穿越美国？更别说我们两个是拥有高等计算机学位的人。毕竟骑车穿越美国是件单调无聊，时常令人沮丧，偶尔让人兴奋，远距离且困难的事情。如果你看看我的简历，你就能找到答案。我花了 5 年的时间获得博士学位，这是一个“动手实现”的过程，因为要写一篇能让导师签字通过的毕业论文并不容易，我甚至在博士资格考试中失败过一次。之后我在加州大学伯克利分校做助理教授，花了 5 年时间获得终身教职，也是“动手实现”的过程。另外，花两个月的时间骑车穿越美国也是“动手实现”的过程。

第三是因为骑行故事是设计实现大型软件系统的一个隐喻。要设计实现 Postgres 系统，首先你需要有几个好主意，然后花费 5 年时间实现一个原型系统，这是一个“动手实现”的过程。在这期间别忘了，什么时候扔掉所有代码重新来过都不晚。在这之后你就可以创立一家公司，雇用一些绝顶聪明的人，完成你的产品，这也是一个“动手实现”的过程。这期间就像在一个长长的沼泽里爬行一样。简而言之，设计实现 Postgres 只需要一个好主意并“动手实现”。可是这花费了我 10 年的时间，期间起起落落，甚至比获得终身教职还难。有人可能会问既然如此之难，那你为什么还要设计实现 Postgres 呢？这和骑行穿越美国是一样的，那是我要做的 (That’s what I do)。

骑车穿越美国与设计和实现 Postgres 有什么相通之处呢？第一，动手实现；第二，有突然出现的奇迹。■

（本文根据 CNCC 2015 特邀报告整理而成）

作　者：

迈克尔·斯通布雷克
(Michael Stonebraker)

美国工程院院士，2014 ACM图灵奖获得者。美国麻省理工学院教授。主要研究方向为数据库研究和开发。

整　理：

邓　栋

清华大学博士生。主要研究方向为数据质量以及数据融合。
dd11@mails.tsinghua.edu.cn

李国良

CCF高级会员、本刊编委，CCF青年科学家奖获得者。清华大学副教授。主要研究方向为大数据管理、数据清洗与融合、移动数据管理等。
liguoliang@tsinghua.edu.cn