数据库技术演进

Malcolm Crowe 荣休教授，计算机科学，苏格兰西部大学，佩斯利，英国
Malcolm.Crowe@uws.ac.uk

Fritz Laux 荣休教授，商务计算，罗伊特林根大学，罗伊特林根，德国
Fritz.Laux@reutlingen-university.de

摘要– 本文回顾了许多研究人员提出的数据库技术变革建议，尤其是致力于不断演进的大数据领域的研究人员。我们探讨了远程数据访问的新方法，以及能够更好地满足商务计算和科学计算等场景中持久性与可审计性需求的标准。我们提出了语言标准可能的演进方向，并在 GitHub 上提供了概念验证实现。

关键词– 大规模实时数据；远程数据；关系型数据库管理系统；SQL；标准。

I. 引言

关系型数据库管理系统（RDBMS）的设计始终着眼于管理结构化且不断演进的数据，例如客户账户和科学研究结果；在这些场景中，共享访问和长期持久性十分重要 [1]。20世纪70年代开发的标准查询语言 SQL 很快成为一项国际标准 [2] ，具备众多功能，其演进也一直为大多数数据库产品所追随。许多研究人员受到启发，致力于为这些产品的实现奠定理论基础，这项工作至今仍在继续 [3][4][5][6]。

在各种形式的演进过程中，一些沿袭下来的方面会随着时间推移变得不合时宜。例如，早期采用的定长字段和有限精度原始类型至今仍存在于数据库存储中，限制了新产品版本的向后兼容性，并影响持久性和可移植性 [7]。包括 PyrrhoDB 在内的一些研究项目则选择采用新的全球化原始类型，以避免依赖机器架构和区域设置 [8]。避免此类依赖有助于数据导入与共享，以及数据仓库的构建 [9]。

数据仓库的发展使人们开始关注元数据和语义，也促使许多系统采用基于文档的NoSQL系统；与此同时，其他研究人员则开发了在关系型系统中纳入语义的方法 [10]。随着这些发展，寻求将关系型数据库管理系统范式调整为能够管理演进中的数据仓库内容（大规模实时数据）的方法，自然成为一种选择 [11]。

数据的演进与持久性之间的张力，一直是关系型数据库管理系统（RDMS）及其相关技术的一个特点。促使 RDBMS 得以发展的用例包括客户账户和库存等商业记录，以及协作科学；在这些场景中，必须支持多人共享访问，同时由负责及时更新数据的许多用户共同使用，这种需求必须与长期存储、一致性和审计的要求相平衡。多年来，随着标准语言 SQL[2], 不断演进，人们通过增加强大的声明式功能和处理功能来完善这类支持；但这种演进也带来了代价：由于并非所有 RDBMS 都实现了同一版本的标准，系统间兼容性会受到影响；而由于 RDBMS 产品也在不断演进，且并非所有 RDBMS 都提供足够的向后兼容性，以便与为早期版本开发的数据库协同工作，数据持久性也会受到影响。正因如此，遗留数据和系统始终是各种商业和科学活动中需要持续关注的问题。

本文的出发点是：数据库管理系统通常应在适当情况下支持企业数据集成，并在有益时支持协作式数据共享。也就是说，数据库管理系统本身应支持（但不强制要求）通过企业扩展数据模型的方法，同时提供相应机制，以便在数据演化的责任归属其他组织时，支持适用于相关情形的应用程序。在这两种情况下，形成的结构都将是一个联邦，允许一定程度的本地管理，并设有委派与责任层级，从而避免一方面过度集中化，或另一方面造成浪费性的重复。本文探讨了若干旨在实现这一目标的数据库管理系统技术改进，同时通过有力的保障措施，确保在共享数据通过系统各部分所支持的活动不断演化时，维持这类复杂系统的一致性。

下一节中，我们将探讨一类重要的用例：人们需要从多个来源收集极具针对性的实时数据，而查询往往会在单台服务器上定位到一个独特的实体。SQL 仍是实现数据库应用程序乃至更通用查询系统的常用方式，理想情况下，任何更改都应尽量符合其最初意图。在本文后续章节中，我们将考察 PyrrhoDB 项目中针对这类用例的一些新颖开源方法。这些方法基于广泛使用的技术，因此有望在未来的大数据发展中发挥作用。PyrrhoDB 本身是一个研究项目，早在 2005 [12] 之前便已启动，并非一款产品；但从创立之初，它就采用了全球化且独立于机器的结构以及国际标准，并始终支持演进与向后兼容性。

在第二节中，我们将探讨最新技术水平，并分析近期研究论文；这些论文指出，数据库支持大型且

不断演进的数据集的需求正在发生变化。本节还为本文其余部分拟定了议程，以探讨并提出对关系数据技术的更改：第三节讨论串行化事务和层次化权限；第四节提出有关数据类型系统和元数据的方案；第五节讨论虚拟数据仓储（基于视图的中介远程访问）；第六节建议使用可共享数据结构构建实现。第七节考察这些方案对查询处理的影响，第八节则提出在常规 SQL 数据访问方法之外增加版本化 API，并将其与其他数据库产品的相关方法进行比较。这些章节都包含示例，此外还在 GitHub 上提供了这些理念的概念验证实现。

II. 大规模实时数据的应用场景

原始科学数据和行政数据对公众而言往往毫无意义，但通常发布在公共网络上，而且通常具有显著的实时性。例如：

- 最新新冠变异株的 DNA 特征（其数据会逐步完善）[13],
- 传感器绘制海啸图谱的最新数据[14],
- 重症患者的治疗史[15],
- 某项流体计算的结果，这项计算耗费超级计算机三天才完成 [16],
- 塔楼中一段钢筋的历史 [17],
- 急诊住院时重症监护设备的可用性 [18],
- 物联网中的某个传感器或执行器[19]。

在某些情况下，建模（或人工智能）可能会提出一些预期，但世界卫生组织、美国国家航空航天局等机构中的许多重要人士希望科学家或研究人员能获得正确的数据。有时，数据（例如来自传感器的数据）是实时的；而在另一些情况下（例如超级计算机的例子），结果可能是根据数值结果生成的高分辨率图像，甚至可能根本没有存储在任何地方。这类需求往往事关生死，为了避免收到近似或过时的信息，人们会求助于电子邮件或电话。

在所有这类情况下，从概念上说，这些数据都属于一个庞大的稀疏数据库，而这个数据库根本不可能有人构建出来。任何单条观测数据都会附带大量依赖元数据（数据溯源、设备特定信息、置信度等）。但是，科学家想提出的问题通常会用数据库术语来表述，例如，考察罕见病患者接受特定治疗后的结果，或某个待更换部件所用钢材的质量等。

如果也认为 SQL 查询和安全远程更新值得实现，那么上述用例就指出了一些可能值得实现的功能。排除授权、世界时、国际标准、审计和关联数据等已成为标准的方面，并考虑到并非所有人都需要的功能，

我们可以轻松列出以下 SQL 支持愿望清单：

- 从远程
  数据集
- 允许按元数据（如资源描述框架（RDF））或数据溯源（若可用）进行搜索
- 确保传输的数据附带带时间戳的溯源信息和所有权信息
- 确保远程更新（如获准）由数据所有者直接处理，并完整记录发送者的用户信息
- 直接访问数据的“事务主库”，避免使用二手或过期数据
- 指定服务质量。例如，优先考虑正确性而非可用性，并报告过期数据或离线服务器
- 尽量减少需要从远程服务器获取或预加载的数据量
- 允许在检索过程中进行转换，并在允许更新时提供相应的逆变换
- 确保更改以安全的方式完成交易并被持久记录。

基于上述讨论，接下来我们将遵循以下一般性考量：

- 关注支持遗留数据的需求，应该促使我们将持久数据与易失数据分离。任何单个账户或演进记录的当前状态都需要能够从内存中访问，但与归档一样，持久系统应优先记录并支持对原始数据进行审计，例如特定的输入、更改和删除。下文中，我们将持久存储这一概念专用于此类档案。
- 另一方面，对演进数据共享状态的访问和修改需要有效的事务控制。要获取所需的持久档案，就需要记录此类事务的日志；而证明所记录事务具有可串行化性的最佳方式，是让日志本身以原子方式按提交顺序记录这些事务，并将每次提交的所有步骤完整保存在一起。该日志应采用追加存储方式实现[20]。我们注意到，一些广泛支持的数据库管理系统功能（例如约束、级联和触发器）会使这一要求更加复杂。
- 大多数数据库管理系统对使用互联网持谨慎态度，更倾向于使用定制功能来管理所有网络交互。在我们看来，这如今是一种错误的做法，也忽视了演进中的互联网标准所带来的全球化机遇。更广泛的访问机会应与更完善的数据所有权、数据溯源和责任记录相平衡，这也有助于解决上述顾虑，即在特殊情况下，能够从源获取结果（甚至更新源），而不是从副本获取或更新。我们将证明，更多地采用互联网标准有望减少浪费性的数据复制，尤其是对“实时”数据而言。在考虑数据库管理系统演进的要求时，

因此，我们考虑以下几个方面：

- 事务串行化的验证，考虑到账户涵盖事务的所有副作用，因此，违反约束的事务不应提交；如果由此产生的级联或触发操作会与其他事务冲突，也不应提交。这项要求由国际 SQL 标准 [2] 规定，但商业数据库管理系统很少实现。

- 我们建议采用一种改进的数据库管理系统设计与安全性方案，将数据模型和安全模型置于数据库中，而非应用程序中。SQL 标准几乎提供了实现这一目标所需的全部支持：我们进一步强调预编译代码和约束的定义者角色，并为数据库本身创建元数据功能。我们还提出了一些相应建议，以增强 SQL 的扩展数据类型系统。
- 与美国国防部《橙皮书》一样关于强制访问控制的标准，我们着重强调用户责任和安全性，同时向角色而非用户授予权限。我们的概念验证代码包含为用户和数据库对象实现《橙皮书》B 级和 C 级所需的功能。角色拥有对象的权限，用户则获授角色。不过，我们建议修改 SQL 标准，使用户一次只能使用一个角色。这是一项切实可行的建议，因为用户可以代替生病的同事，但所有操作都会以一种能够识别用户及其当时所声明角色的方式予以记录。

- SQL 编程模型在计算能力上是完备的：我们建议禁止使用外部代码和过程，以便数据库管理系统能够管理所有必要的验证和审计工作。
- 在这些情况下，我们支持通过各种方式改善对 SQL 数据库的远程访问。

本文其余部分将针对上述各方面提出切实可行的方案，同时尽可能少地更改 SQL 标准。这些方案的概念验证代码已在 GitHub 上的 PyrrhoDB 中实现。下文将按以下功能类别介绍详情：串行化事务、数据库管理系统问责与数据所有权、元数据，以及基于视图的远程访问。

III. 串行化事务

根据上述讨论，我们会对所有事务提交执行验证步骤，以确保满足完全串行化事务的要求。因此，ISO 标准中列出的隔离级别（读取_未提交、读取_已提交、可重复_读取、可串行化）已不再适用，因为可能的隔离级别只有一种，可称为串行化 [21],[22], 。这会减少完整性约束可用的操作数量：不允许使用NO ACTION，并限制约束可以延迟的程度。验证步骤可确保事务完全隔离。这意味着事务期间所做的更改绝不会对其他用户可见，但会阻止相互冲突的事务提交。

在事务期间，新记录和数据库对象会暂时分配到内存中的位置，以便它们可以在事务线程中访问并正常工作。提交时，在完成验证步骤后，这些对象会以级联方式迁移到文件中的相应位置，并记录在事务日志中，然后重新安装到内存数据库中。有关此过程的更多详情，请参见 [23]。

在此验证步骤中，事务冲突测试采用以下粒度：(a) 对同一数据库对象（表除外）的更改始终会发生冲突；(b) 对于表，如果其他事务更新了任何已读取的列，我们就报告冲突；但如果仅读取了特定行，则可将验证步骤限定于这些行。即使在并发度很高的情况下，这种粒度的验证也切实可行 [25]。该测试的最新实现（2022年8月）使用两个简单的树结构来记录任何受影响表中的列和行，并且还演示了级联、约束和触发器的正确行为（[26] 中的文件已更新以展示这一点）。

为了最好地实现提交算法中验证步骤所隐含的乐观并发控制，我们主张使用可共享数据结构。在讨论如何共享可修改的数据（例如数组）时，计算机科学教科书通常会对比读时复制和写时复制这两种方法。从我们的角度来看，这两种方法都会浪费时间和资源，而使用可共享数据结构则提供了另一种方法，非常适合数据库技术中常见的各种树状结构。要引出这一概念，一个好方法是考察编程语言中字符串的实现。

在 Unix 中，传统上，字符串（char *）是可变的：任何能够访问字符串的人都可以修改其中的单个字符。在 Java、C# 和 Python 中，字符串是可共享的：修改单个字符的唯一方法是创建一个新字符串，因此，如果两个线程共享一个字符串，那么一个线程对该字符串所做的任何更改，除非显式地将新版本提供给另一个线程，否则另一个线程都无法看到。

除了字符串之外，数据库技术中最常用的数据结构是B树。除根节点外，每个节点至少有n个子节点，最多不超过2n个子节点，其中n>1，信息存储在叶节点中。因此，要使数据库结构可共享，关键步骤是使用一种可共享的B树。该模型可追溯至1982年[26], ，下方图1所示的示意图表明，当对树的某个叶节点进行更改时，我们会得到一个新的根节点，而这项更改需要$O(\log_n N)$个新节点，其中N是叶节点的数量。

这意味着新旧版本会继续共享该结构中的大多数节点。稍加思考便可看出，与上述任何一种方法（字符串实现、读时复制、写时复制）相比，这种方式更节省存储空间，但会加重内存分配和垃圾回收的负担。不过，至关重要的是，这种方式是安全的；如果我们在数据库中用这种结构实现所有索引和列表，那么许多数据库操作（例如启动新事务）就会简单得多 [27]。我们将在下文第七节中进一步讨论这些方面。

数据库管理系统应为允许本地管理的安全模型指定并提供审计支持。SQL 标准有机会在这一领域倡导良好实践。PyrrhoDB 已为本地数据库采取了以下实用措施：

A. 将完整事务日志作为唯一存入非易失性内存的记录。

1972 年，将易失性信息存入非易失性存储有其充分理由，但这些理由如今已不再成立。可以理解的是，如果数据库占用大量物理存储空间，数据库管理员可能会认为事务日志所需的额外存储是一种奢侈。PyrrhoDB 的完整事务日志也采用序列化方式，因此可以明显看出，并发事务得到了正确处理。即使在并发程度很高的情况下，本文提出的算法和解决方案也已证明切实可行 [21]。

如果写入磁盘的唯一数据是插入或更新的记录，或表示某条记录已被删除的标记，那么数据库流量所需的磁盘活动会大幅减少，尤其是在数据库索引存储于磁盘上的情况下 [12]。

B. 记录每次数据库变更对应的用户和角色

这项功能相对容易实现，但数据库专业人员和会计师对此强烈抵制，因为他们不愿在自己管理的数据库或编制的客户账户中留下自己的痕迹。然而，这需要对 SQL 标准作出若干偏离 [2]：其特性 F771 和 F321 允许在查询语言中声明“当前用户”，而不是由操作系统予以保证；此外，它也不要求用户设置单一角色。出于取证目的，并为了允许员工因病等原因临时担任不同角色，识别用户和声明的角色都很重要。如果按照上文第 III.A 节的建议维护事务日志，这件事就很简单。

为了使角色和用户信息对取证分析有用，应弃用向角色授予对象所有权和角色使用权限的做法，并弃用向用户授予这些权限以外任何权限的做法。

如第III.C节所述，应允许使用对象的定义者角色，将所有权授予另一位用户。

C. 数据库对象只能由其
所有者修改，且所有执行都应使用定义者角色

根据III.B，在定义对象时存在一个当前角色：即定义者角色，且该角色必须是用户获准使用的角色之一。该角色和所有者的身份会成为对象的属性，并可通过授权进行修改。新定义的细节会在解析期间以及对象后续每次执行期间进行检查。

SQL 标准为过程调用规定了上下文栈，因此，将这种栈的用途扩展到访问表列和视图源，以及执行约束和触发器，相对而言也很容易。

随后，执行引擎只需将被调用上下文的当前角色设为相应表、视图、过程、约束或触发器的定义者角色；根据 III.B 的规定，执行引擎能够确定该角色。调用者仍需要具备适当的权限，才能启动这一流程（通过访问或修改表或视图，或调用过程），并访问任何表的列或行结果。

标准中的规范使得为数据库操作创建一套可用的权限变得非常困难，因为用户需要对每种数据类型和每一列都拥有使用权限。

建议进行两项额外简化：这样一来，标准中的 REFERENCES权限就变得多余，因为它与SELECT的作用相同；此外，如果所有数据类型都可供PUBLIC 使用（不过，对其字段的访问可能会受到限制），安全模型也会更简单。按照此处所述使用定义者角色，并配合这些更改，安全模型将更易于管理。新对象可以由定义它们的用户拥有（并将该用户声明的角色设为定义者角色），而授予对象权限时无需考虑数据类型或依赖定义。因此，即使数据库对象数量庞大，也更容易维护一套可用的权限配置。

采取这些措施后，Pyrrho 的安全模型更易于管理和验证；不过，执行速度自然会有所降低：检查对单个对象的访问权限，需要访问一次该对象的属性树，其深度通常为 3（见下文）。

我们认为，这比 Oracle [28] 和 PostgreSQL [29]采用的安排有所改进。这些产品对定义者角色所作的警示是正确的，因为它们会安装本机外部过程。数据库服务器执行操作是安全的，因为它可以在访问对象时检查所有对象权限。

通过使用上文讨论的角色声明模型，关系数据库的所有安全设置都可以并且应该由数据库自身管理，而不是由数据库应用程序管理。标准 SQL 模型允许对角色和权限的管理进行层级委派，并将其与用户身份验证分开。

例如，请看下面这个乒乓球俱乐部的简单数据库。它允许对所示的两张表进行 SELECT 访问，但普通成员对数据库所做的更改必须借助提供的两个过程完成：

创建表 成员 (id int 主键, firstname 字符) [创建表 对局记录 (id int 主键, 获胜者 int 引用 成员, 失败者 int 引用 成员, agreed 布尔值)] 授予查询权限 on 成员 to PUBLIC 授予查询权限 on 对局记录 to PUBLIC [创建过程 claim(won int, beat int) 插入 into 对局记录(获胜者, 失败者) VALUES(claim.won, claim. beat)

] 创建过程 agree(p int) 更 [

新 对局记录 set agreed=true where 获胜者=agree.p and 失败者 in (select m.id from 成员 m where 用户 like '%'| |firstname escape '^')] 创建角色 管理员 创建角色 membergames [授权 execute on procedure claim(int, int) to role

membergames]
grant 将过程 agree(int) 授权给角色 membergames，将 membergames 授权给 PUBLIC
执行

要使用这些过程，获准登录系统的公众成员应将自己的角色设为 membergames。

IV. 类型系统与元数据

企业数据集成和数据协作的一大难点，是定义一种数据模型，以支持企业不同部门的应用程序开发。我们认为，数据库应尽可能为数据语义提供支持，同时尽可能保留本地开发的灵活性。

首先，我们为 JSON 值引入原始 Document 类型，并允许在 SQL 中使用花括号“{”和“}”界定行值，使用方括号“[”和“]”作为 Document 值的字符串下标；此外，还引入内置的、返回 Document 值的函数 HTTP，其参数为 HTTP方法和 URL，第三个可选参数为表示提交的数据的 Document。

许多数据库管理系统发现，有必要以各种方式扩展其数据访问方法和数据库应用程序：

- 控制查询的 XML 和 JSON 输出，以确定表列是作为表的属性/字段输出，还是作为其子项/子文档输出。
- 用于数据可视化，例如图表
- 实体数据模型：在数据库应用程序中声明与数据库基表相对应的类，并将派生类引用与外键、查找函数等关联起来。

我们认为，将所有此类元数据纳入数据库设计是良好的实践，并且应按角色进行设置，以便针对不同业务用途提供一系列数据库应用程序。

在 PyrrhoDB 中，我们整理出了一份实用的元数据标识符列表。

元数据 = 标题 | 图例 | X | Y | ((直方图 | 折线图 | 饼图 | 数据点) ['(' id ',' id ')'])| ([URL | MIME | SQL 代理 | 用户 | 密码] 字符串)| JSON | CSV | 实体标签 | 毫
| 单调 | ((反演 |格式化) id)| 属性 | 实体 | ((后缀 |前缀) id) | 国际化资源标识符。

此语法是 Pyrrho 扩展，几乎任何 DDL 命令都可以向数据库对象添加（或删除）元数据。大多数选项会影响 Pyrrho Web 服务中某个角色的查询输出。上面的列表将这些关键字粗略分为四组：(1) 针对特定表和视图的数据可视化，(2) 支持与远程数据协作，(3) 支持适配器函数，以及 (4) 支持本地数据模型。列中若存在 属性，则表示包含该列的表优先采用 XML 输出。直方图、图例、折线图、数据点、饼图（用于表、视图或函数元数据）、标题、X 和 Y（用于列或子对象元数据）指定添加到 HTML 输出中的 JavaScript，用于绘制指定的数据可视化内容。此语法允许使用字符串作为描述。对于 反演，id 应为被反演函数的名称，而对于格式化，id 是一种类型。前缀和后缀定义添加到客户端输出字符串中的 id，并在 SQL 触发器中定义该类型的默认构造函数，正如本节末尾的货币示例所述。

Pyrrho 可借助上述第 (1) 组关键词定义数据可视化，并通过简单的 URL 映射 HTTP 服务提供支持，如下例所示：

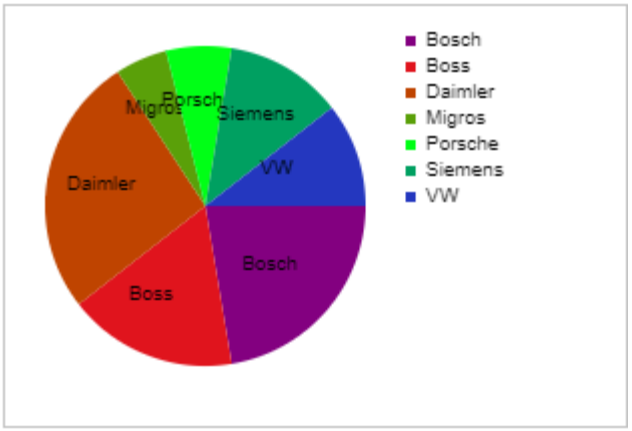
使用以下语句创建的数据库 E

[create table sales (cust char(12) primary key, custSales numeric(
8,2])) insert into sales values ('Bosch' ,
[
17000.00),('Boss' , 13000 00 20000 00] .),
[
('Daimler', .) insert into sales values ('Sie
mens',9000.00),('Porsche', 5000.00), ('VW', 8000.00), ('Migros' ,
4000.00)] create role E grant E to "用户*machine*/用户名"

数据可视化输出采用 HTML，并返回给客户端应用程序或直接显示在页面上。在这里，如果浏览器请求

http://localhost:8180/E/E/SALES/?PIE(CUST,CUSTSALES)LEGEND

浏览器将显示 PyrrhoDB 服务器返回的以下内容：



我们将在下文回到这个示例。

用户定义类型可以在UNDER子句中指定一种原始类型，这对于区分从不同子组织导入或使用的数据很有用。SQL 标准已经提供了用于选择某类型值的OF谓词、用于为标量值指定子类型的TREAT函数，以及用于指定行类型的“创建类型表”机制。Pyrrho 还增加了为VALUES指定子类型的功能。

例如，若要展示由此产生的语法，假设我们定义：

[创建类型 货币，其属性为（amt 数值型，unit 字符），方法 exchange（tounit 字符）返回货币，方法 tonumeric（）返回数值型]

这里的 exchange 方法将通过上述 HTTP 函数在数据库中实现。

互联网上有许多货币转换器，例如

[创建方法 exchange(tounit char) returns currency for currency begin if unit=tounit return this; declare rates document; declare roe numeric; set rates=http('post', 'http://www.floatrates.com/daily/'||unit||'.json'); set roe= rates[lower(tounit)]['rate']; return currency(amt*roe,tounit) end]

那么我们就可以这样写：

[创建类型 dollars，基于 currency，检查条件为 unit='USD')，构造方法 dollars(x numeric)，构造方法 (x currency)，前缀 "$"] 创建构造方法 dollars(x [numeric) begin set amt = =] x; [set unit 'USD' end 创建构造方法 dollars(x currency) begin set amt= x.exchange('USD').amt; set unit='USD' end]

如果我们对欧元也进行类似的声明，就可以把代码写得非常简洁，例如：

选择 euros("$" 10) 创建表 money (cur currency) 向 money 插入值 ("$" 34), ("€" 212.7) 从 money 中选择所有内容，其中 cur 的类型为 dollars

```
SQL> select euros("$" 10)
|-----------------|
|EUROS            |
|-----------------|
|€9.6397693015117|
|-----------------|
SQL> create table money (cur currency)
SQL> insert into money values ("$" 34), ("€" 212.7)
2 records affected in rb
SQL> select * from money where cur is of(dollars)
|---|
|CUR|
|---|
|$34|
|---|
SQL>
```

我们在下文第八节中给出一个使用数据模型元数据指令 实体 的示例。

V. 视图中介的远程访问

数据仓储涉及创建中央数据存储库（使用抽取-转换-加载技术），以便对组合数据集进行分析处理。在某些情况下，这并不可取，例如，中央存储库承担的数据保护责任过重，数据变化频繁，实时维护所有集中存储的数据成本高昂，或者更适合将数据留在其源头，由源头承担相应责任 [10]。借助数据库技术，视图（如果已定义但未实体化）可以访问其他位置定义的数据。虚拟数据仓库这一概念利用了这一点，并致力于避免数据在中央端累积。Pyrrho 使用 HTTP，通过简单的 REST 接口 [22], 从远程数据库管理系统收集数据，因此这里由此产生的技术称为 RESTView。

因此，借助 RESTView，Pyrrho 数据库可以定义视图，其中的数据存储在远程数据库管理系统中，并可通过 HTTP 发送 SQL 语句访问，返回 JSON 响应。Pyrrho 本身提供此类 HTTP 服务，发行版还包含合适的接口服务器（ RestIf ），以便为远程 MySQL 和 SqlServer 数据库管理系统提供此类服务。该实现支持以远程数据库管理系统普通客户端的身份进行身份验证，其管理员可以授权访问定义恰当的视图。

HTTP 访问使用贡献者数据库所有者在 MySQL 中为此目的设置的用户名/密码组合。在此讨论的用例中，查询 Q 引用了 RESTView V；我们假设：(a) 出于某些法律原因或所需数据量过大，不宜通过提取—转换—加载的方式物化 V；(b) 我们对贡献者数据库的内部细节一无所知。每个 RESTView 都由一条远程 SELECT 语句定义：与贡献者达成的协议并未规定任何复杂协议，因此，对于任意给定的 Q，我们希望对任何一个贡献者至多发起一次查询，并且该查询符合贡献者授予我们的权限，即授予对 RESTView 各列的查询权限。

不过，关键是，对于给定的任意 Q，我们都希望尽可能减少传输数据量 D。我们可以考察 Q 需要多少数据才能计算出结果，并重写查询，使 D 尽可能小。显然，许多此类查询（例如显而易见的 select * from V）都需要全部数据。另一种极端情况是，如果 Q 仅引用本地数据（不涉及 RESTViews），那么 D 始终为零，因此所有这些分析都只适用于 RESTView 技术。

在查询处理期间，视图会被其定义替换，因此整个查询变成从其所引用的表中进行选择。该过程通过为每个表引用添加唯一标识符，来处理同一张表在多个位置被引用的情况。

筛选器会应用于查询的最低层级（例如，直接应用于远程表），而遍历远程表会向指定 URL 发出 REST 服务往返请求。返回结果的 JSON 表示会经过少许扩展，加入用于计算任何远程聚合的寄存器[23]。

语法如下：

视图定义 = [视图规范] AS (查询表达式 | GET [USING 表标识符]) {元数据}。

竖线所示的选项取决于视图由单个贡献者提供，还是由多个远程数据库提供。此处的 QueryExpression 选项是定义视图的常规语法。两个 REST 选项都包含 GET 关键字。最简单的 RESTView 类型定义为从元数据中指定的 URL 执行 GET。列的类型需要使用稍作扩展的视图规范语法指定。如果有多个远程数据库，则可以使用 GET USING table_id 选项。此表中的行描述远程贡献：最后一列

提供贡献者的元数据，包括 URL；其他列中的数据（如果有）则会直接复制到视图中。例如：

```
SQL> create view VV of (E int,F char) as get 'http://localhost:8180/DB/DB/t'
SQL> create view WW of (E int, D char, K int, F char) as get using VU
SQL> select * from ww where e=5
|-|-|-|----|
|E|D|K|F   |
|-|-|-|----|
|5|C|1|Five|
|-|-|-|----|
SQL> table vu
|-|-|---------------------------|
|D|K|U                          |
|-|-|---------------------------|
|B|4|http://localhost:8180/DB/DB/t|
|C|1|http://localhost:8180/DC/DC/u|
|-|-|---------------------------|
SQL>
```

根据远程贡献的定义方式，RESTViews 可能是可更新的，并且可能支持插入和删除操作。

这些理念的实现已在[23]中展示。

在这些安排下，需要考虑多主机场景中的事务要求。根本难点是所谓的双军问题：所有数据都需要一个事务主库。每个事务都在一个数据库上启动（将其服务器所在的主机称为本地主机），然后通过上述类型的视图定义访问远程数据。假设事务提供了访问该数据库所需的适当凭据，它便可以提交本地服务器上的更改，并且最多执行一次远程服务器更新。提交按以下机制进行：(a) 锁定本地数据库；(b) 验证本地更改；(c) 使用 HTTP 1.1 执行单次远程更新（采用 RFC7232 机制）；(d) 随后即可完成本地提交并解除锁定。只有一次远程更新时，该机制是安全的，且发生任何异常时都可以回滚。

可以设想运用这些技术实现异构数据库管理系统之间的互操作，因此，务必继续采用 REST 服务的行业标准方法。许多系统已经实现了 URL 和 XML/JSON 到数据库的映射，并且可以利用 RFC7232 [24] 中的 ETag 机制来提供事务功能 [20]。目前，Pyrrho 提供了几种实现方式，由上文列出的元数据标志 URL和 ETAG决定。

再次考虑第四节中的销售数据库 E，它会随着时间推移积累大量销售记录。假设 E 向角色 rs_V 提供一个数据视图，其中包含将当前累计销售份额计算为介于 0 和 1 之间的数值：

[创建视图 sales¯V(cust, custSales, runningSalesShare) 作为 select cust, custSales, (select sum(custSales) from sales where custSales >= u.custSales) / (select sum(custSales) from sales) from sales as u] 创建角色 rs V¯授权 rs_V 给 "user\machine"

随后，便可以通过指定机器上的仪表板式查询访问此视图，根据当前累计销售份额将客户归类为 A、B 或 C，而无需获知所有单笔销售记录。

[select case when runningSalesShare <= 0.5 then 'A' when runningSalesShare > 0.5 and runningSalesShare <= 0.85 then 'B' when runningSalesShare > 0.85 then 'C' else null end as 类别, cust, custSales, cast(cast(custSales / (select sum(custSales) from sales _ 100 V) * as 十进制数(6, 2)) as 字符(6)) || ' %' as 份额 from sales V¯order by custSales desc]

输出及由此生成的饼图如图2所示。

VI. 使用可共享数据结构的实现

本节将介绍上述功能实现的一些细节。该实现遵循前文阐述的理念，采用全局化且独立于体系结构的数据格式。PyrrhoDB 使用 64 位唯一标识符来标识所有数据库对象、日志条目和表行。它为可变长度的基本类型提供了一种表示方式：整数（最多 2040 位）、实数（整数尾数、整数刻度）和字符（最多 2^{60} 字节的 Unicode 字符串）。数据库对象（数据类型除外）的命名按角色区分。

如上所述，数据库在磁盘上以事务日志的形式表示，该日志由一系列“物理记录”组成：大约有 70 种物理记录格式，其中一种用于事务详情，另一种用于表标识符，还有一种用于列详情等。详情请参阅 GitHub 发行版中的 Pyrrho 手册。事务日志采用追加存储。

服务器首次访问时，会读取数据库的整个事务日志，并在内存中构建活动数据库对象。

在 StrongDBMS [21] 于高并发演示中成功执行可串行化事务之后，PyrrhoDB 已重新实现，全面采用可共享数据结构。可共享数据结构不能更新或修改，因此任何变更都需要创建一个新实例。可共享数据结构的示例包括基本数据类型，如整数或浮点数；C#、Java 或 Python 中的字符串类型；以及字段全部为只读可共享数据结构的类。借助一些简单的可共享构建块（BList<V> 和 BTree<K,V>），可以轻松构建表示表、索引乃至数据库的可共享数据结构。

Domain类指定一种基类型及许多其他属性。如果它包含列（例如，用户定义类型或基表），Domain 还会指定一个列唯一标识符列表，以及一棵用于给出各字段所属 Domain 的树。原始类型具有由系统分配的（负数）唯一标识符，而其他任何 Domain 都会作为物理对象创建在定义它的数据库中。

表行由类型值组成：类型值由一个 Domain 和一种可共享数据结构定义。

上文介绍了 BTree<K,V> 实现：它是一种不平衡的 B树，插入、更改或删除节点的最坏情况性能为 O(logN)。这些更改中的任何一项都会创建一个新的根节点，以及通往新叶节点的新内部节点，最多产生 logN 个新节点，而其余节点则由树的新旧版本共享。因此，这种实现的效率出奇地高。

BList<V> 就没那么巧妙了。它以 BTree 的形式实现，但会将节点重新编号为 0、1、2……，导致最坏情况性能为 O(N)。

只要内容（所有 K 和 V 对象）可共享，BList 和 BTree 就是可共享数据结构。遍历 BTree 和 BList 时，不使用 Java 和 C# 中的枚举器，而是使用同样可共享的数据结构“书签”：可以双向遍历，而且遍历会从指定的根节点继续进行，因此不会受到被遍历树发生更改的影响。

构成数据库对象的所有类都可以共享。例如，行集本质上是由行组成的 BTree，这些行是由游标遍历的类型值；游标是前文提到的书签类的子类。

每个连接到数据库的连接都会启动一个新的服务器线程。协议请求通常会创建一个事务，用于查询或修改数据库，或读取查询结果中的下一组数据；这些操作都限定在该连接线程中。创建事务很简单：每个事务都从数据库根节点的副本（快照）开始。回滚或断开连接时，可以直接丢弃该事务，因为其他线程都未曾看到它。

VII. 查询处理与编译对象

解析过程中，会为生成的表达式以及任何可能作为新数据库对象提交的内容分配唯一标识符。根据需要分配上文所述范围内（目前）4×2^{60} 的唯一标识符：根据其生命周期是当前会话、当前事务还是当前词法输入等因素，这些唯一标识符分属若干个范围。所有 SQL 表达式在解析时都会确定其域；所有行集也都有指定其列的域，因此在查询处理过程中会根据需要构造临时域。由于查询可能通过一个表行集多次引用同一个表源，因此表行集的列唯一标识符必须针对这类引用分别设置，并在堆范围（7×2^{60}以上）内分配：在实现中，这一过程称为实例化。视图也可能引用多个数据源，因此同样适用实例化过程。表值函数也有类似的需求。

在本节中，我们还将探讨如何实现本地数据管理这一概念。如第III.C节所述，服务器应继续掌控存储过程、触发器和约束的执行，因此这些功能应以 SQL 编写。由于编译对象的定义者通常拥有与执行查询或更新的用户不同的权限，因此确保可执行代码预先编译至关重要。出于向前和向后兼容性的考虑，数据库文件仅包含存储过程、约束、触发器等的 SQL 源代码。数据库在服务器中加载时（冷启动后）会构建已编译的组件。

如上所述，许多数据库对象对应于事务日志中的永久物理记录，因此其定义位置是固定的，并且可以由此数据库的所有事务共享。服务器在编译期间构造的对象（实际上也可以共享）没有物理文件位置，因此会被分配专用范围内的唯一标识符（目前为 6×2^{60} .. 7×2^{60}-1），并组成一个集合，与编译对象的内存版本一同存储。大多数编译对象包含可执行代码，但此机制也用于基表或视图的域。这些编译对象实际分配到的唯一标识符取决于日志中物理对象的顺序，也取决于服务器的当前版本。在实例化期间，列唯一标识符将通过级联方式分配，因为许多编译对象会包含对正在实例化的列的引用。

在 RowSet 审查过程中也会使用级联：只要存在编译时不可用的索引和筛选器，就会尽可能简化 RowSet 管道。

VIII. 版本化库和数据模型

以上讨论描述了如何在数据库实现中表示数据库的数据模型。将数据模型置于数据库中的真正好处，是让应用程序员能够使用它，从而使面向同一数据库的所有应用程序都能对其数据的结构和语义达成一致。目前，除了提供一种线程安全的 Command/ExecuteReader/Read 编程接口（该接口为 ADO.NET 和 JDBC 用户所熟悉）之外，Pyrrho 还为使用 C#、Java 和 Python 编写的应用程序提供了这类支持。

当前实现受到 Microsoft 的 Entity Framework [30] 和 Java Persistence Architecture [31]的启发，但与它们有一个关键区别：应用程序员应从服务器生成（并在运行时由服务器检查）的类定义着手，而不是以注解或代码属性的形式编写自己的模型版本。

以下示例展示了可提供的支持类型。假设数据库 ABC 中包含一个角色“Sales”，该角色定义了以下表：

```
[创建表 "Customer"(id int 主键, "NAME" 字符 唯一)
                                ] 创建表 "Order"(id int 主键, cust int 引用 "
[
Customer", "OrderDate" 日期, "Total" 数值型(6,2))]
```

接下来，系统表 "Role$ClassValue" 将提供类似以下内容的代码片段：

```
使用 System;使
用 Pyrrho;

/// <摘要>/// 来自数据库 ABC、角色 Sales 的 Custo
mer 类 // 主键(ID) // 唯一(NAME)
```

```
/// < > /摘要 Table( )] public class Customer : 版本 [ 23,122
化 {[ ] Field(Pyrrho 数据类型.
[
Integer) 自动键 ] public Int64? ID;

[ ] Field(Pyrrho 数据类型.String)

public 字符串? NAME; public Order [] =>
< >
orders conn.FindWith Order(("CUST",ID)); } /// <摘要> /// 来自数据
库 ABC、角色 Sales 的 Order 类 // 主键(ID) // 外键, 限制更新, 级联删除
(CUST) /// < > /摘要 Table( )] public class Order :
[ 175,362
版本化 {[ ] Field(Pyrrho 数据类型.
[
Integer) 自动键 ] public Int64? ID;

[ ] Field(Pyrrho 数据类型.Integer)
64
public Int ? CUST; [ ] Field(Pyrrho 数据
类型.Date) public 日期? OrderDate; [Field(Pyrrho 数据类型.Decimal,"
NUMERIC 域 Prec=6 刻度 =2 ] ") public 十进制数? Total;

public Customer customer =>
< >
conn.FindOne Customer ((“ID”,CUST)); }
```

数字 23 和 175 是这些对象在数据库中的定义位置引用，其他数字则是架构键，应用程序运行时服务器会检查这些键，以确保表定义未发生变化。我们可以看到，这些类中定义的列均可公开访问（服务器会在访问时检查用户和角色），而这些列的 Pyrrho 数据类型则以属性的形式提供。

重要的是，表之间的外键关系使上述类中增加了两个“导航”字段，从而可以快速访问订单对应的客户，以及客户对应的订单。主键和唯一键声明也支持快速访问。

使用上述类定义的简单程序可以这样开头：

```
静态 void 主程序 {

conn = new PyrrhoConnect("Files=Demo;Role=Sales");
conn.Open(); try { // 获取所有订单列表并显示客户名称 var aa =
conn.FindAll<Order>(); foreach (var a in aa)
Console.WriteLine(a.ID + ": " + a.customer.NAME); if
(aa.Length == 0) { Console.WriteLine("Order 表为空"); goto
skip; } // 更改第一笔订单的客户名称 //（更新导航属性）var j =
aa[0].customer;
j.NAME = "Johnny"; j.Put(); // 添加一位新客户
（此处使用自动键） var g = new Customer() { NAME =
"Greta" }; conn.Post(g); // 为 Mary 创建一笔新订单 //
（二级索引；此处可省略单引号！） var m =
conn.FindOne<Customer>(("NAME","Mary")); var o =
new Order() { CUST = (long)m.ID,
OrderDate = new Date(DateTime.Now) };
conn.Post(o);
```

上面的 Versioned 基类使用 ETag，使库能够将代码中的对象引用关联到数据库中的行。这不仅使上面的示例程序可以使用简写表示法，还能在提交显式事务时自动进行事务验证。该事务使用类似于 ADO.NET 和 JDBC 的 API 启动。更多详情请参阅 Pyrrho 手册 [32]。

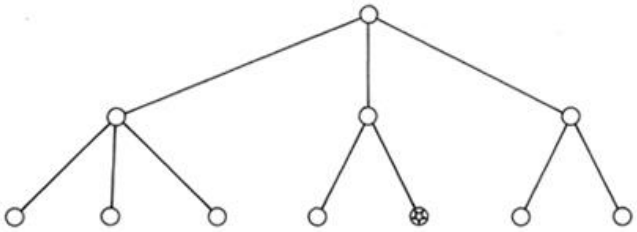
IX. 结论

本文回顾了近期文献中提出的关系数据库模型的一系列改进，并概述了 ShareableDataStructures 项目在 GitHub [32] 上提供的这些改进的实现。PyrrhoDB v7 的实现目前处于阿尔法阶段，欢迎各界对这些构想提出意见。作者衷心感谢在本项目期间收到的许多支持与鼓励。

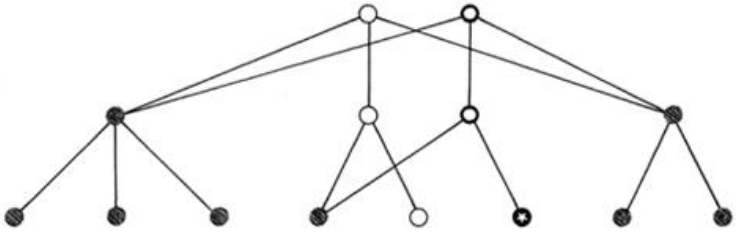
参考文献

[1] Crowe, M. K. 和 Laux, F.：《数据演化与持久性：PyrrhoDB 实验的经验教训》，主题演讲，ICSEA 2022，葡萄牙里斯本，2022年10月，https://www.iaria.org/conferences2022/filesICSEA22/Keynote_MalcolmCrowe_FritzLaux_DataEvolutionAndDurability.pdf（访问于2022年12月4日）。[2] ISO.（2016）。IEC 9075:2016：信息技术：数据库语言：SQL，国际标准化组织。[3] Schüle, M. E., Kemper, A. 和 Neumann, T.（2022年7月）。用于数据挖掘的递归 SQL。载于第34届科学与统计数据库管理国际会议（第1-4页）。[4] Stockinger, K., Bundi, N., Heitz, J. 和 Breymann, W.（2019）。大数据金融分析的可扩展架构：用户定义函数与 SQL。大数据期刊，6(1)，1-24。[5] Khine, P. P. 和 Wang, Z.（2019）。大数据领域多语言持久化综述。《Information》，10(4)，141。[6] Aluko, V. 和 Sakr, S.（2019）。大数据 SQL 系统：实验评估。《集群计算》，22(4)，1347-1377。[7] Antonopoulos, P., Budovski, A.,Diaconu, C., Hernandez Saenz, A., Hu, J...（2019年6月）。Socrates：云端的新型 SQL Server。载于2019年数据管理国际会议论文集（第1743-1756页）。[8] Crowe, M. K.（2015）。《Pyrrho之书》，苏格兰西部大学，ISBN 978-1-903978-50-4。[9] Alam, F. 和 Kamal, N.（2019）：传统数据仓库到实时数据仓库综述及实时数据对社会的影响，《国际计算机应用杂志》177.9，第20-24页。

[10] Interlandi, M., Ekmekji, A., Shah, K., Gulzar, M. A., Tetali, S. D. 等（2018）。为 Apache Spark 添加数据溯源支持。The VLDB Journal，27(5)，595-615。[11] Crowe, M. K., Begg, C. E. 和 Laux, F.（2017），实时大数据的数据验证，载于 DBKDA 2017，第九届数据库、知识与数据应用进展国际会议，巴塞罗[illegible]58-6（第30-36页）[12] Crowe, M. K.（2005）：Pyrrho 数据库引擎中的事务，载于 Hamza, M. H.（编）：DBA 2005，IASTED 数据库与应用国际会议论文集，因斯布鲁克，ISBN：0-88986-460-8（第71-76页）[13] de Moura, M. C., Davalos, V., Planas-Serra, L., Alvarez-Errico, D., Arribas 等（2021）。COVID-19重症及呼吸衰竭的表观基因组范围关联研究。EBioMedicine，66，103339。[14] Parsi, M. 和 Akbarpour Jannat, M. R.（2021）。使用物联网的海啸预警系统。《海洋学杂志》，11(44)，1-17。[15] Soriano, A., Carmeli, Y., Omrani, A. S., Moore, L. S., Tawadrous, M. 和 Irani, P.（2021）。头孢他啶-阿维巴坦用于治疗治疗选择有限的严重革兰氏阴性菌感染：系统性文献综述。《传染病与治疗》，10(4)，1989-2034。[16] Gorobets, A. 和 Bakhvalov, P.（2022）。异构中央处理器+图形处理器并行化：在混合型超级计算机上对可压缩湍流进行高精度尺度解析模拟。《计算物理通讯》，271，108231。[17] Bai, Y., Ma, Y., Yang, Q., Florez-Lopez, J., Li, X. 和 Biondini, F.（2021）。地震诱发损伤更新，用于钢框架子结构系统的剩余寿命评估。《机械系统与信号处理》，159，107782。[18] Komenda, M., Černý, V., Šnajdárek, P., Karolyi, M., Hejný, M., Panoška, P. 等（2022）。重症监护控制中心：COVID-19疫情期间实现有效协调、实时监测和战略规划的工具。《医学互联网研究杂志》，24(2)，e33149。[19] Meana Llorián, D., González García, C., Pelayo García-Bustelo, B. C. 和 Cueva Lovelle, J. M.（2021）。BILROST：通过推文和 Twitter，使用领域专用语言处理物联网执行器。《国际交互式多媒体与人工智能杂志》。[20] Finlayson, R 和 Cheriton, D.“日志文件：一种利用一次写入存储的扩展文件服务。”《ACM SIGOPS 操作系统评论》21.5（1987）：139-148。[21] Crowe, M. K. 和 Fyffe, C.（2019）：StrongDBMS 基准测试，（主题演讲），发表于 DBKDA 2019，第十一届数据库、知识与数据应用进展国际会议 https://www.iaria.org/conferences2019/filesDBKDA19/MalcolmCrowe_CallumFyffee_Keynote_BenchmarkingStrongDBMS.pdf（访问于2022年12月4日）[22] Crowe, M. K. and Laux, F.：实现真正的可串行化事务，教程视频，DBKDA 2021，第十三届数据库、知识与数据应用进展国际会议，西班牙瓦伦西亚。https://www.youtube.com/watch?v=t4h-zPBPtSw&t=39s，（访问于2022年12月4日）[23] Crowe, M. K. and Laux, F.：实现真正的可串行化事务，教程文件，DBKDA 2021，第十三届数据库、知识与数据应用进展国际会议，西班牙瓦伦西亚。https://www.iaria.org/conferences2021/filesDBKDA21/[24] Fielding, R. T. and Reschke, J（编）（2014）：RFC 7232：超文本传输协议（HTTP/1.1）：条件请求，IETF.org[25] Crowe, M. K. and Laux, F.：重新审视关系型数据库管理系统的乐观算法，DBKDA 2020，第十二届数据库、知识与数据应用进展国际会议，葡萄牙里斯本[26] Krijnen, T. and Meertens, G. L. T.：“让B树适用于B”。阿姆斯特丹：数学中心基金会，1982年，技术报告 IW 219/83[27] Crowe, M. K. and Matalonga, S.（2019）：StrongDBMS：由不可变组件构建，载于 DBKDA 2019，第十一届数据库、知识与数据应用进展国际会议，雅典，ISBN 978-1- 61208-715-3（第11-16页）[28] Oracle.com 产品文档：http[illegible]le.com/en/database/oracle/oracle-database/19/dbseg/managing-security-for-definers-rights-and-invokers-rights.html（访问于2022年12月4日）[29] PostgreSQL 产品文档 https://www.postgresql.org/docs/current/sql-createfunction.html（访问于2022年12月4日）[30] Microsoft 在 GitHub 上的产品软件：https://github.com/dotnet/efcore（访问于2022年12月4日）[31] Oracle 产品文档 https://docs.oracle.com/javaee/7/tutorial/persistence- intro.htm#BNBPZ（访问于2022年12月4日）[32] Crowe, M. K.：GitHub 上的 PyrrhoDB 手册；https://github.com/MalcolmCrowe/ShareableDataStructures/blob/master/PyrrhoV7alpha/doc/Pyrrho.pdf（访问于2022年12月4日）



(a) the original shared tree,
the position of modification is marked



(b) the path to the position of modification is "deshared",
new nodes are thicker, shared nodes are shaded

图1。 B树的运作 [26]

CATEGORY	CUST	CUSTSALES	SHARE
A	Daimler	40000	52.63 %
A	Bosch	34000	44.74 %
B	Boss	26000	34.21 %
B	Siemens	18000	23.68 %
C	VW	16000	21.05 %
C	Porsche	10000	13.16 %
C	Migros	8000	10.53 %

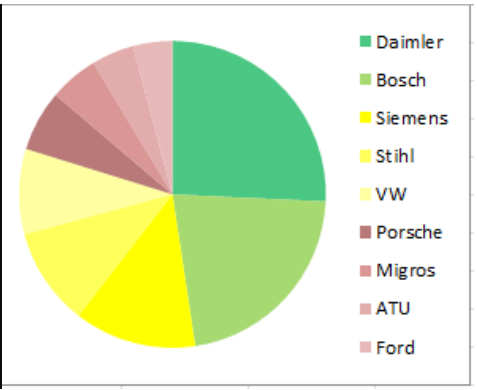


图2: 第VI节示例中的ABC分析（由PyrrhoDB客户端和服务器输出）