



设计高效 Tableau 工作簿 的最佳做法

Tableau 10 版本

Alan Eldridge
Tableau Software

关于本文档

我再次承认，本文是很多作者所著资料的浓缩精华。我做的大部分工作就是将他们的作品融汇成一个文档，并进行一些结构安排。本文的某些读者会在各个部分中认出这些作者的“手笔”（事实上，有人会发现整段文字都似曾相识）。感谢所有这些作者，如果不是各位持续贡献出色作品且不追究我们侵犯版权，本文将无法成型。

此外，还要感谢审阅此文档的许多人员，是你们帮助确保了文档的准确性和可读性。你们对细节的关注及耐心解释让本文档更加清晰可读，凭我一人之力是绝对无法做到这种程度的。

为了反映 Tableau 10 的功能，已更新本文档。Tableau 以后的发行版将提供新功能，这些功能可能会使本文中的部分建议发生改变。

谨致谢意

Alan Eldridge

2016 年 6 月

太长不看

常有读者反映或建议说，这本白皮书太长。我的回答是，这是因为它需要涵盖范围广泛且具有适当深度的材料。

但我将文档要点总结如下：

- 针对低效工作簿的终结武器是不存在的。首先请查看性能记录器，了解时间花费的去向。查询运行时间过长？查询过多？计算缓慢？呈现方式复杂？利用本文提供的见解，朝着正确方集中向努力。
- 本文仅提供建议。尽管这些建议代表某种程度的最佳做法，但仍需测试其在具体情况下能否提高性能。许多建议可能因数据结构和所用数据源（例如平面文件、RDBMS 和数据提取）而异。
- 若要加快大多数工作簿的运行速度，数据提取是一种简单快捷的方法。
- 数据越清晰，它与问题结构的匹配度越高（即所需准备和操作越少），工作簿的运行速度就越快。
- 多数仪表板运行迟缓的原因是设计不当（尤其是单个仪表板包含多个图表），或者尝试同时显示过多数据。简化设计。允许用户逐步深入详细信息，而非尝试显示所有内容之后再行筛选。
- 仅处理所需数据 - 包括引用的字段和所返回记录的精度。因此，Tableau 可以生成更少、更佳、更快的查询，还可减少需要从数据源移动到 Tableau 引擎的数据量。同时缩小工作簿，使其能够更轻松共享、更快打开。
- 削减数据时，务必有效利用筛选器。
- 处理字符串和日期较慢，处理数字和布尔值较快。

最后，本文中部分建议仅对处理大型和/或复杂数据集有实质性影响。何谓“大型”或“复杂”？视情况而定...但是在所有工作簿中遵循这些建议也无妨，毕竟您不知道数据什么时候会增加。实践出真知。

目录

关于本文档	2
太长不看	3
简介	6
Tableau 有何优点？	6
Tableau 有何缺点？	6
理解高效	8
何谓“高效”工作簿？	8
为何应注重高效？	8
物理定律.....	9
交易工具	11
性能记录器.....	11
日志	13
Tableau Server 性能视图	14
监视和测试.....	15
其他工具.....	15
是不是工作簿设计问题？	17
优秀的仪表板设计.....	17
调整仪表板以提升性能.....	20
优秀的工作表设计.....	25
高效筛选器.....	31
是不是计算问题？	39
计算类型.....	41
分析	45
计算与本机功能.....	45
数据类型的影响.....	45
性能技巧.....	46

这是我的查询吗？	51
自动优化.....	51
联接	57
混合	57
数据集成.....	61
自定义 SQL	61
自定义 SQL 的替代方法.....	63
这是我的数据吗？	65
一般性建议.....	65
数据源	65
数据准备.....	73
数据提取.....	74
数据管理.....	79
是环境吗？	80
升级	80
在服务器上测试 Tableau Desktop.....	80
独立式刷新和交互式工作负荷.....	80
监视和调整服务器.....	80
基础设施.....	81
结语	83

简介

Tableau 有何优点？

在 Tableau，我们力求改变人们查看数据、与数据交互和理解数据的方式。因此，我们无意提供与传统企业 BI 平台类同的体验。在用来创建具有以下特征的工作簿时，Tableau 效果最好：

- **形象直观** – 大量证据表明，人类理解复杂大数据集最高效的方式是通过可视表现形式。Tableau 的默认行为是使用图表、图解和仪表盘来显示数据。表和交叉表自有其用武之地（并受到支持），稍后将详细介绍其最佳使用方法。
- **交互式** – Tableau 文档主要设计为以交互方式传达给用户，无论是在用户桌面上、通过 Web 还是在移动设备上。Tableau 与其他某些 BI 工具不同，这些工具主要生成打印为主的输出（打印到实际纸张或打印到 PDF 之类的文档），而 Tableau 的重心是创建丰富的交互式体验，让用户在业务问题的引导下探索数据。
- **迭代式** – 发现本质上是循环的过程。Tableau 旨在加速从问题到见解再到问题的周期循环，这样用户就能快速形成假设，通过可用数据验证、修改、再验证该假设，如此循环往复。
- **快速** – 从历史上看，BI 过程一直很慢。软件安装和配置很慢；使数据可供分析很慢；设计和实现文档、报表、仪表盘等也很慢。Tableau 可以让用户以前所未有的速度安装、连接和开发文档 – 很多情况下，这将产生答案的时间从数月或数周缩减到几小时或几分钟。
- **简单** – 传统的企业 BI 工具通常超出了大部分业务用户的能力所及，要么成本高昂，要么太过复杂。很多情况下，用户需要 IT 或高级用户的帮助才能创建他们所要的查询和文档。Tableau 为非技术用户提供了直观的界面，他们无需成为数据库或电子表格专家，就能查询和分析复杂的数据。
- **美观** – 人们常说，美丽在观者之眼中，但是在谈到可视化通信之美时，有一些最佳做法可循。通过“智能显示”之类的功能，Tableau 引导非技术用户根据所用的数据，创建高效、可理解的图表。
- **普遍适用** – 用户创建的文档正逐渐不再针对单一交付平台。用户需要查看数据，并与数据交互。这些数据可以桌面上、在 Web 上、在移动设备上、嵌在其他应用程序和文档中，甚至在其他地方。有了 Tableau，只需发布一次文档，就可以在所有这些平台上使用此文档，而无需任何移植或再设计。

Tableau 有何缺点？

Tableau 是一种丰富而强大的工具，但是我们从一开始就要理解，对于某些问题，Tableau 可能并不是最佳的解决方案，这一点很重要。这并不是说 Tableau 不能处理这些问题 – 有人可能想方设法用 Tableau 去执行很多违背 Tableau 设计初衷的任务。我们的意思是，我们开发 Tableau 并不是为了解决这些类型的问题，因此，如果您执意这样做，那么很可能得不偿失，最终得到的解决方案可能性能很差或者缺乏灵活性。

如果遇到以下情况，我们建议您考虑重新审视需求或者另谋他法：

- 您需要的文档不是为屏幕，而是为纸面设计的。这句话的意思是，您是需要控制复杂的页面布局，需要页面页眉/页脚、节页眉/页脚和组页眉/页脚之类的功能，还是需要精确的所见即所得格式。Tableau 可以生成多页报告，但是缺乏专用的分栏式报告工具所提供的格式控制级别。

- 您需要复杂的推送投递机制，通过多种投递模式发送包含个性化内容的文档（也称为“定期投递”）。Tableau Server 包括报告订阅的概念，这允许用户订阅自己（在 Tableau 10 中还可订阅他人），以便通过电子邮件接收报告，尽管客户有时希望使用更灵活的解决方案。Tableau 可用于创建其他形式的推送投递系统，但这不是 Tableau 本身就有的功能。需要开发基于 TABCMD 实用工具生成的自定义解决方案，或者引进第 3 方解决方案，例如 VizAlerts (<http://tabsoft.co/1stldFh>) 或 Metric Insights (<http://bit.ly/1HACxul>) 提供的 *Push Intelligence for Tableau*（推动 Tableau 智能）。
- 读者的主要用例是将数据导出为另一种格式（通常为 CSV 或 Excel 文件）。这通常意味着会得到包含多行详细数据的表格式报告。明确一下，Tableau 确实允许用户将数据从视图或仪表板导出为 Excel – 可以在概要级别或详细信息级别导出。但是，主要用例是导出数据时，这就变成了伪提取-转换-加载 (ETL) 过程。有些解决方案要比报告工具更胜任此类任务。
- 您需要高度复杂的交叉表样式文档，这些文档可能反映了有着复杂小计、交叉引用等的现有电子表格报告。常见示例有损益表、资产负债表等财务报表。此外，可能还需要场景模拟、假设分析，甚至回写假设数据。如果不可获得基础精细数据，或者报告逻辑是基于“单元格引用”，而不是累加记录形成总计，那么可能适合继续使用电子表格提供这种样式的报告。

理解高效

何谓“高效”工作簿？

使工作簿变得“高效”的因素有很多。有些是技术因素，有些更侧重于用户，但总体而言，高效工作簿有以下特征：

- **简单** - 工作簿是否易于创建,将来是否易于维护？是否利用可视化分析原则来清楚传达作者和数据的信息？
- **灵活** - 工作簿能够解答用户想问的多个问题，还是只能解答一个问题？它给用户带来的是交互式体验还是简单的静态报告？
- **快速** - 对用户而言，工作簿的响应速度是否够快？这可能涉及打开时间、刷新时间或交互响应时间。响应速度是一项主观指标，但一般来说，我们希望工作簿提供信息的初始显示，并在几秒钟内响应用户交互。

仪表板性能受以下因素影响：

- 仪表板和工作表级别的可视化设计 - 例如元素数、数据点数、筛选器和操作的使用等；
- 计算 - 例如计算种类、执行计算的位置等；
- 查询 - 例如返回的数据量、是否为自定义 SQL；
- 数据连接和基础数据源；
- Tableau Desktop 与 Tableau Server 间的某些差异；
- 硬件配置和容量等其他环境因素。

为何应注重高效？

注重高效有几个原因：

- 像分析师或工作簿作者般高效工作可更快获得答案；
- 高效工作有助于随时分析数据。这意味着，您会思考问题和数据，而不是思考如何操作工具取得成果。
- 创建灵活设计的工作簿，可降低创建和维护多个处理类似要求的工作簿的需求。
- 创建简单设计的工作簿，可让他人更轻松地进行您的工作簿并在其基础上进行更多迭代工作。
- 主观响应能力是最终用户查看报告和仪表板时的一项重要成功指标，因此使工作簿运行速度尽可能快有助于提升用户满意度。

就我们的丰富经验来看，客户遇到的多数性能问题均归咎于工作簿设计中的错误。如果我们可以修复这些错误（通过教育从根本上预防错误则更佳），便可解决问题。

如果处理的数据较少，多数建议都无关紧要。尽可以简单粗暴地解决问题。但是，处理数亿条记录、大量工作簿或多个作者时，糟糕工作簿设计带来的影响会被放大，此时，必须深入思考本白皮书中的指南。

当然，请相信实践出真知，建议对所有工作簿都遵循此指南。请记住，针对预期生产数据量测试工作簿后才算完成设计。

重要说明：虽然全文以 Tableau Server 为例，但多数情况下，如果更倾向于使用我们的托管解决方案而不是本地部署，本指南也适用于 Tableau Online。调整/优化服务器配置参数以及在服务器层安装/更新软件时例外 - 软件即服务功能将为您效劳！

物理定律

在我们深入探讨各种功能如何影响工作簿性能的技术细节之前，有一些可帮助您制作高效仪表板和视图的基本原则：

如果在数据源中处理缓慢，则在 Tableau 中也将是缓慢

如果 Tableau 工作簿基于缓慢运行的查询，则工作簿也将是缓慢的。在后面部分，我们将识别几条数据库调优提示，帮助缩短查询运行所用的时间。此外，我们将讨论 Tableau 的快速数据引擎如何用于提高查询性能。

如果在 Tableau Desktop 中处理缓慢，则在 Tableau Server 中也将（几乎始终）是缓慢的。

在 Tableau Desktop 中性能缓慢的工作簿，即使发布到 Tableau Server，性能也不会有丝毫提高。用户常常认为自己的工作簿在 Tableau Server 中运行更快，因为服务器的 CPU/RAM 等比本地 PC 更高。通常，工作簿在 Tableau Server 上的运行速度稍慢，因为：

- 多个用户同时共享服务器资源以生成工作簿（尽管有违直觉地，您可能发现工作簿在开始共享时的响应速度更快，这归功于 Tableau Server 内的缓存机制）；并且，
- 服务器必须执行额外工作才能呈现仪表板和图表，呈现工作并不是在客户端工作站完成的。

开始准备在 Tableau Server 中优化性能之前，应首先努力在 Tableau Desktop 中优化工作簿。

如果 Tableau Desktop 遇到服务器上不存在的资源限制，例如 PC 上的 RAM 不足以支持所分析的数据量，或者服务器连接到数据源的速度更快/延迟更低，则此规则不适用。某些用户在低配的 2G RAM 工作站上处理数据集时，会遇到性能缓慢的问题，甚至是“内存不足”错误，但却发现已发布的工作簿性能速度是可接受的，这是因为服务器的内存和处理能力要强大得多。

越新越好

Tableau 开发团队不懈努力，尽力提升软件性能和可用性。升级到最新版本的 Tableau Desktop 和 Tableau Server，有时无需对工作簿进行任何更改即可大大提升性能。例如，许多客户称仅将 V8 升级到 V9，工作簿性能便提升了 3 倍或以上，而 Tableau 10 及更高版本更将继续关注性能提升。当然，Tableau Online 不存在此问题，因为它始终会在发行最新版本时自动升级。

这适用于维护版本和主要/次要版本。有关 Tableau 发行周期和每个版本具体细节的详细信息，请访问发行说明页面：

<http://www.tableau.com/zh-cn/support/releases>

此外，数据库提供商正致力于改善其产品/服务。还请确保使用相应数据源驱动程序的最新版本，如以下网页中所示：

<http://www.tableau.com/zh-cn/support/drivers>

少即是多

与生活万事一样，物极必反，过犹不及。绝对不要把所有内容都放到一个大而全的工作簿中。虽然 Tableau 工作簿可以有 50 个仪表板，每个仪表板可以有 20 个图表对象，但是如果连接 50 个不同的数据源，那么几乎可以肯定，它的性能会非常缓慢。

如果发现有这样的工作簿，请考虑将其分成多个单独的文件。简单易行 - 仅需复制工作簿之间的仪表板，Tableau 即可调取所有关联的工作表和数据源。如果仪表板过于复杂，请考虑将其简化，并使用交互来引导最终用户在不同报告间切换。请记住，我们的软件并不是按文档数量收费的，所以请尽管将数据分散到多个文档。

可伸缩性与性能不同

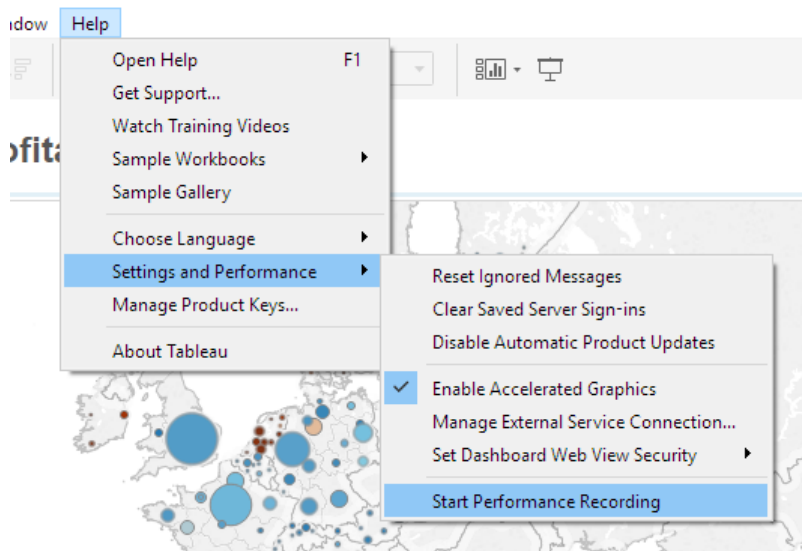
可伸缩性在于确保我们可以支持多个用户查看共享工作簿。性能在于确保单个工作簿尽快运行。尽管本文提供了许多建议，对发布到 Tableau Server 工作簿的可伸缩性会有积极影响，但本文重点在于提升性能。

交易工具

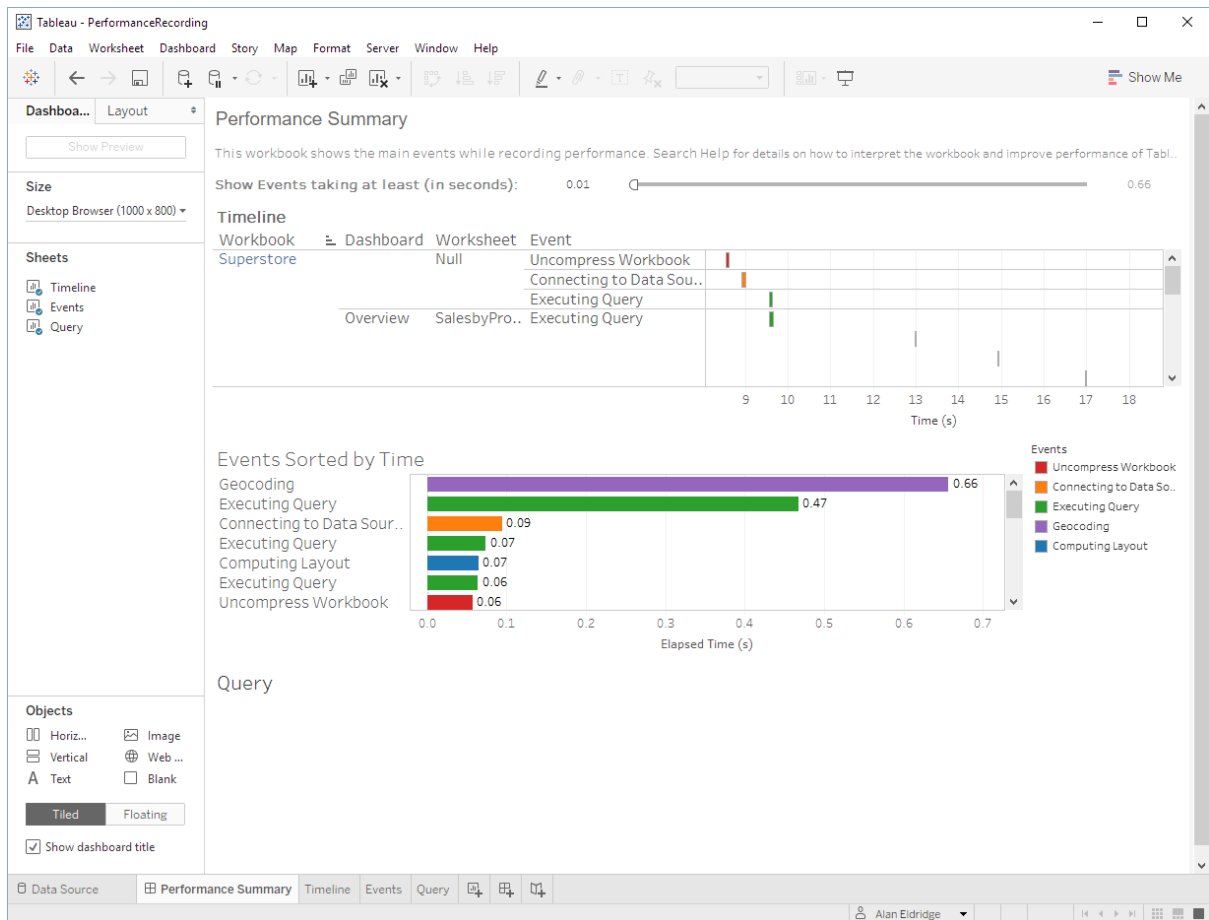
若要了解工作簿的性能，需要了解 a) 正发生的事件 b) 它将花费多长时间。可在多个位置、多个详细级别捕获此信息，具体取决于运行工作簿的位置（即 Tableau Desktop 或 Tableau Server）。本节概述了各个可用选项。

性能记录器

首先应通过 Tableau Desktop 和 Tableau Server 的“性能记录器”功能查找性能信息。在 Tableau Desktop 中，可在“帮助”菜单下启用此功能：



启动 Tableau 之后，启动性能记录，然后打开工作簿（执行此操作时最好不要打开其他工作簿，以免意外争用资源）。像最终用户一样与该工作簿交互。认为收集到足够数据后，请返回帮助菜单，并停止记录。此时会打开另一个 Tableau Desktop 窗口，内含已捕获的数据：



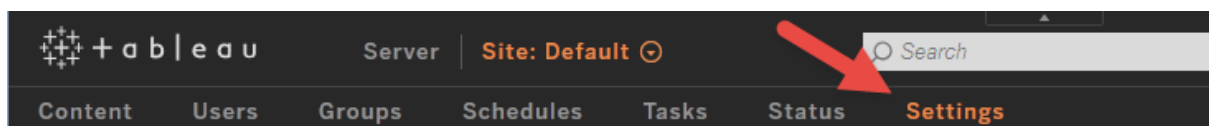
现在可以找出该工作簿中用时最长的操作 - 例如在上图中，选自“时间范围”工作表中的查询耗时 30.66 秒方才完成。单击条形图会显示所执行查询的文本。由于性能记录器的输出是 Tableau 工作簿，还可以创建额外视图，以其他方式浏览此信息。

注意：默认情况下，不会显示耗时小于 0.1 秒的事件。通过调整仪表板顶部的筛选器，可将此类事件加入显示队列，但应重点关注长时间运行的任务。建议将其设置为 1 秒。

还可以在 Tableau Server 上创建性能记录，帮助识别发布工作簿时出现的问题。默认情况下，Tableau Server 不会启用性能记录 - 可基于站点控制此功能。

服务器管理员可以对站点逐一启用性能记录。

1. 导航到要启用性能记录的目标站点。
2. 单击“设置”：



3. 在“工作簿性能指标”下，选择“记录工作簿性能指标”。
4. 单击“保存”。

要创建性能记录时，请执行以下操作：

1. 打开要记录性能的目标视图。打开视图后，Tableau Server 随即在 URL 后附加“:iid=<n>”。这就是会话 ID。例如：

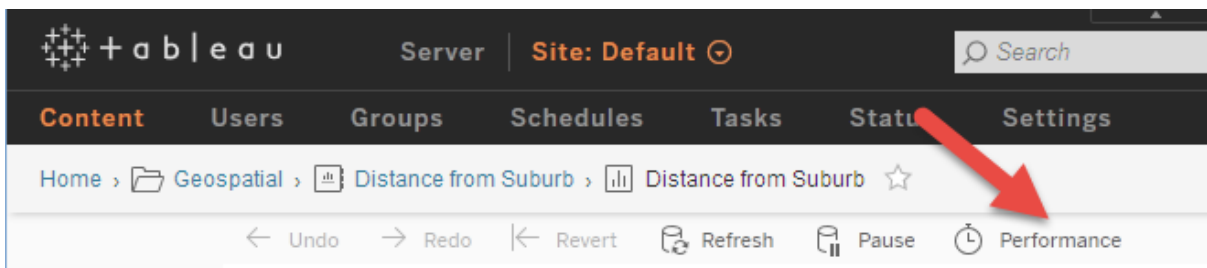
```
http://<tableau_server>/#/views/Coffee_Sales2013/USSalesMarginsByAreaCode?:iid=1
```

2. 在视图 URL 之后、会话 ID 正前键入 :record_performance=yes&。例如：

```
http://<tableau_server>/#/views/Coffee_Sales2013/USSalesMarginsByAreaCode?:record_performance=yes&:iid=1
```

3. 加载该视图。

视图工具栏中的“性能”选项将显示性能记录已启动的可视化确认消息：



完成操作并准备好查看性能记录时，请执行以下操作：

1. 单击“性能”打开性能工作簿。这是性能数据的最新快照。可在继续处理视图时继续获取其他快照；性能数据是可累积的。
2. 移动到其他页面或从 URL 删除 :record_performance=yes 可停止记录。

应使用此信息来确定工作簿中最值得检查的那些部分 - 即，在哪些方面可以最大程度缩短所用的时间？有关如何解读这些记录的更多信息，可以从下面的链接找到：

http://onlinehelp.tableau.com/current/pro/desktop/zh-cn/help.htm#perf_record_create_desktop.html

日志

在 Tableau 中，查看日志文件可以找到完整查询文本。默认位置是 C:\Users\<username>\Documents\My Tableau Repository\Logs\log.txt。此文件相当冗长且编写为 JSON 编码的文本，因此，建议使用 Notepad++ 或 Sublime 等优秀文本编辑器处理它。如果搜索“开始查询”或“结束查询”，可以找到传递到数据源的查询字符串。查看“结束查询”日志记录还将显示该查询运行所用时间和返回到 Tableau 的记录数：

```
{"ts": "2015-05-24T12:25:41.226", "pid": 6460, "tid": "1674", "sev": "info", "req": "-", "sess": "-", "site": "-", "user": "-", "k": "end-query", "v": {"protocol": "4308fb0", "cols": 4, "query": "SELECT [DimProductCategory].[ProductCategoryName] AS [none:ProductCategoryName:nk], \n [DimProductSubcategory].[ProductSubcategoryName] AS [none:ProductSubcategoryName:nk], \n SUM(CAST([FactSales].[ReturnQuantity]) as BIGINT)) AS [sum:ReturnQuantity:ok], \n SUM([FactSales].[SalesAmount]) AS [sum:SalesAmount:ok] \n FROM [dbo].[FactSales] [FactSales] \n INNER JOIN
```

```

[dbo].[DimProduct] [DimProduct] ON ([FactSales].[ProductKey] =
[DimProduct].[ProductKey])\n INNER JOIN [dbo].[DimProductSubcategory]
[DimProductSubcategory] ON ([DimProduct].[ProductSubcategoryKey] =
[DimProductSubcategory].[ProductSubcategoryKey])\n INNER JOIN
[dbo].[DimProductCategory] [DimProductCategory] ON
([DimProductSubcategory].[ProductCategoryKey] =
[DimProductCategory].[ProductCategoryKey])\nGROUP BY
[DimProductCategory].[ProductCategoryName],\n
[DimProductSubcategory].[ProductSubcategoryName]", "rows":32, "elapsed":0.951}
}

```

如果在 Tableau Server 上查看该日志记录，则日志位于 C:\ProgramData\Tableau\Tableau Server\data\tabsvc\vizqlserver\Logs。

Tableau Server 性能视图

Tableau Server 为管理员提供多个视图，帮助监视 Tableau Server 中的活动。这些视图位于服务器“维护”页面的“分析”表中：

Analysis	
Dashboards that monitor Tableau Server activity.	
Dashboard	Analysis
Traffic to Sheets	Usage and users for published sheets.
Traffic to Data Sources	Usage and users for published data sources.
Actions by All Users	Actions for all users.
Actions by Specific User	Actions for a specific user, including items used.
Actions by Recent Users	Recent actions by users, including last action time and idle time.
Background Tasks for Extracts	Completed and pending extract task details.
Background Tasks for Non Extracts	Completed and pending background task details (non-extract).
Stats for Load Times	Sheet load times and performance history.
Stats for Space Usage	Space used by published workbooks and data sources, including extracts and live connections.
Server Disk Space	Current and historical disk space usage, by server node.
Tableau Desktop License Usage	Summary of usage for Tableau Desktop licenses
Tableau Desktop License Expirations	Expiration information for Tableau Desktop licenses

有关这些视图的更多信息，可访问以下链接：

<http://onlinehelp.tableau.com/current/server/zh-cn/adminview.htm>

此外，还可连接到 Tableau 存储库中的 PostgreSQL 数据库，创建自定义管理视图。相关说明请参阅下方：

http://onlinehelp.tableau.com/current/server/zh-cn/adminview_postgres.htm

监视和测试

TabMon

TabMon 是用于 Tableau Server 的一个开源群集监视器，可用于收集随时间变化的性能统计数据。TabMon 由社区提供支持，我们即将发行经获 MIT 开源许可的完整源代码。

TabMon 是记录系统运行状况和应用程序指标的现成工具。它可收集网络中 Tableau Server 计算机上的内置指标，如 Windows Perfmon、Java Health 和 Java Mbean (JMX) 计数器。使用 TabMon，可以监视物理（CPU、RAM）、网络和硬盘使用情况。还可以跟踪缓存命中率、请求延迟、活动会话数等。它以清晰、统一的结构显示数据，让用户能够在 Tableau Desktop 中轻松地直观显示数据。

使用 TabMon，您可以完全控制所收集指标和所监视计算机，无需编写脚本或代码。只需了解计算机和指标名称即可。TabMon 可以远程或独立运行群集。几乎无需向生产计算机添加任何负载，便可以监视、聚合和分析网络上任何计算机中的群集运行状况。

可在此处找到有关 TabMon 的详细信息：

<https://github.com/tableau/TabMon/releases>

TabJolt

TabJolt 是一款“即点即运行”的负载和性能测试工具，专用于轻松配合 Tableau Server 操作。与传统的负载测试工具不同，TabJolt 可以针对 Tableau Server 自动驱动负载，而无需脚本开发或进行维护。由于 TabJolt 考虑到了 Tableau 的演示模式，所以它可以自动加载可视化内容，并解读测试执行期间可能存在的交互。

因此，可将 TabJolt 指向服务器上的一个或多个工作簿，在 Tableau 视图中自动加载和执行交互。TabJolt 还可收集关键指标（如平均响应时间、吞吐量和第 95 百分位响应时间）并捕获相关 Windows 性能指标。

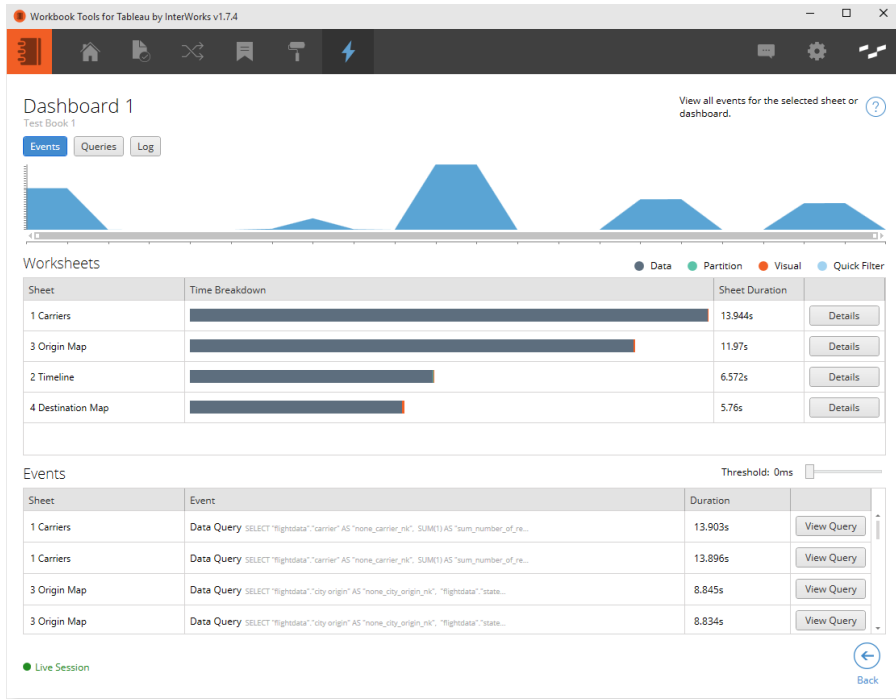
当然，即使使用 TabJolt，用户仍应充分了解 Tableau Server 的体系结构。不建议将 Tableau Server 用作负载测试的黑匣子，否则很可能出现不符合预期的结果。Tabjolt 是一个自动化工具，它并不能轻易复制各种人为交互操作，因此，请认真思索您的 Tabjolt 结果是否反映出实际结果。

可在此处找到有关 TabJolt 的详细信息：

<https://github.com/tableau/tabjolt/releases>

其他工具

还有一些可帮助识别工作簿性能特征的其他第三方工具。其中之一是 Interworks 研发的“Power Tools for Tableau”，它带有性能分析器（类似于内置性能记录器），可用于深入探究并找出耗时最长的表格和查询：



Palette Software 也有一款称为 Palette Insight 的产品，使用此产品，您可以捕获 Tableau Server 的性能信息、执行容量规划、识别占用大量资源的用户和工作簿、执行用户访问审核和生成退款模型。



此外，多数现代 DBMS 平台也包括允许跟踪和分析运行中查询的管理工具。如果性能记录指示查询运行时间是一项关键因素，那么 DBA 可起到很大帮助。

如果确信客户端浏览器与服务器的交互存在问题，还可以使用 Telerik Fiddler 等工具或浏览器的开发人员工具，以便深入了解客户端和服务端之间的流量情况。

是不是工作簿设计问题？

对于许多用户来说，使用 Tableau 是一项新体验，为了创建高效工作簿，这些用户需要学习一些设计技巧和最佳做法。但是，我们发现很多新用户试图对 Tableau 应用老式设计方法，得到的结果自然乏善可陈。本节旨在提出部分可反映最佳做法的设计原则。

优秀的仪表板设计

借助 Tableau，最终用户将能够获得的是交互式体验。Tableau Server 交付的最终结果是可让用户探索数据，而不仅仅是查看数据的交互式应用程序。所以，为了创建高效的 Tableau 仪表盘，就不能再用开发静态报告的思路来看待问题。

例如，这里有一个仪表盘类型，我们发现很多新手作者都会创建这种类型，尤其是如果他们以前使用过 Excel 或 Access 之类的工具，或者有过使用“传统”报告工具的背景。我们先从一个“什么都显示”的表格式报告以及一系列筛选器说起，这些筛选器允许用户细化表，直至表显示他们感兴趣的那几条记录：

A Bad Design								
Continent	Country	State/Province	Product Category	Product Subcate..	Product Name	Sales Qty	Total Cost	Sales Amou..
Asia	Turkmenistan	Ahal Province	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	64	\$5,547.52	\$11,790.68
North America	United States	Alaska	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	62	\$5,200.80	\$11,536.20
North America	Canada	Alberta	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	40	\$3,467.20	\$7,540.00
Europe	France	Alpes-Maritim.	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	31	\$2,687.08	\$5,734.17
Asia	Armenia	Armenia	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	19	\$1,560.24	\$3,468.40
Europe	France	Bas-Rhin	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	36	\$3,033.80	\$6,710.60
Europe	Germany	Bavaria	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	60	\$5,114.12	\$10,819.90
Asia	China	Beijing	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	2,276	\$194,683.28	\$421,825.30
Europe	Germany	Berlin	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	1,193	\$102,022.36	\$220,330.11
Europe	Switzerland	Bern	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	38	\$3,207.16	\$6,936.80
North America	Canada	British Colum.	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	107	\$9,274.76	\$20,075.25
Europe	Romania	Bucuresti	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	10	\$866.80	\$1,885.00
Europe	Greece	Central Greec.	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	18	\$1,560.24	\$3,317.60
Asia	Japan	Chubu	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	13	\$1,040.16	\$2,337.40
Asia	Kyrgyzstan	Chuy Province	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	66	\$5,547.52	\$12,280.78
North America	United States	Colorado	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	699	\$59,635.84	\$130,093.28
North America	United States	Connecticut	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	293	\$25,050.52	\$54,533.05
Asia	Syria	Damascus	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	104	\$8,928.04	\$19,311.83
Europe	United Kingdo.	England	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	478	\$40,912.96	\$88,702.45
North America	United States	Florida	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	423	\$35,885.52	\$78,745.88
Europe	Germany	Hesse	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	26	\$2,253.68	\$4,674.80
Asia	Japan	Hokkaido	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	22	\$1,906.96	\$4,109.30
Asia	China	Hong Kong	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	111	\$9,448.12	\$20,574.78
Asia	Pakistan	Islamabad Ca.	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	63	\$5,287.48	\$11,724.70
Asia	Japan	Kansai	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	119	\$10,228.24	\$22,156.18
Asia	Japan	Kanto	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	176	\$15,255.68	\$32,648.20
Asia	Thailand	Krung Thep	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	89	\$7,541.16	\$16,522.03
Europe	Ireland	Leinster	Cameras and cam.	Digital Cameras	A. Datum Advanced Digital Ca..	31	\$2,687.08	\$5,824.65

Continent

Asia

Europe

North America

Country

Armenia

Australia

Bhutan

Canada

City

Null

Albany

Alexandria

Amsterdam

Product Category

Audio

Cameras and camcorders

Cell phones

Computers

Product Subcategory

Air Conditioners

Bluetooth Headphones

Boxed Games

Camcorders

Product Name

A. Datum Advanced Digit.

A. Datum Advanced Digit.

A. Datum Advanced Digit.

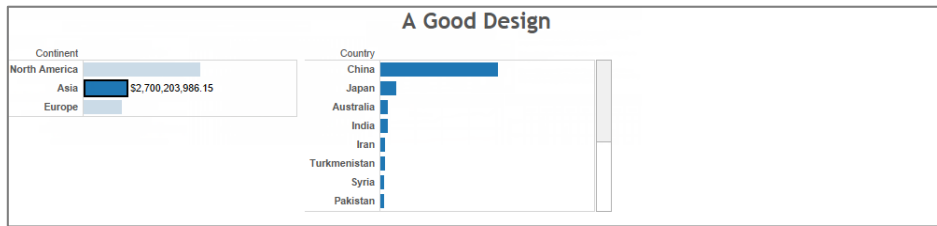
A. Datum Advanced Digit.

这不是一个“优秀的” Tableau 仪表盘（实际上，甚至根本不是“像样”的仪表盘）。从坏处看，这是一个徒有虚名的数据提取过程，因为用户要做的是将数据提取到另一个工具（如 Excel）作进一步分析以及绘制图表。而从好的方面看，它暴露出我们其实不了解最终用户希望如何探索数据，所以我们采取的做法是“根据用户的起始条件，把一切都丢给用户...然后再提供几个筛选器对象，让他们自己进一步细化结果集，找到他们真正关心的东西”。

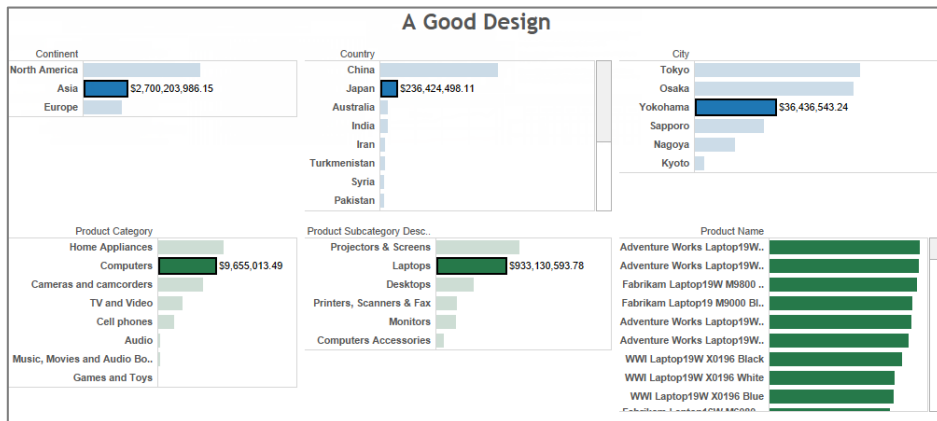
现在请考虑下面重做的结果 – 所用的数据完全一样。我们先从最高的聚合级别开始：



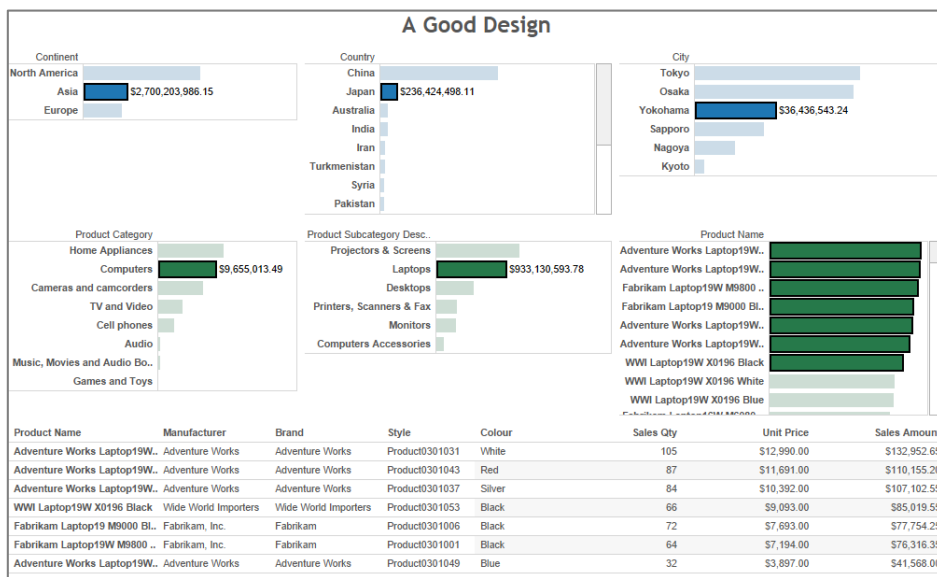
选择一个或多个元素后，将显示下一个详细级别：



我们不断这样选择，每次都会显示更深一层的详细级别：



直至我们最后揭示出最终级别 – 也就是上面的交叉表仪表板中显示的同样的数据。



不要太注重数据的表现形式（虽然数据表现形式也很重要，但这是后话）。而是应该考虑使用此仪表板的体验。请注意它是如何按自然路径流动的，从左到右、从上至下。这个示例背后可能有很多底层数据，但是仪表板引导最终用户逐渐下钻，直至发现他们要找的重点详细记录集。

这两个示例的关键区别是，它们如何引导最终用户完成分析过程。第一个示例开始时涉及面很广（显示您可能要查看的所有记录），然后让最终用户应用筛选器来减少显示的记录数。此做法存在一些本身固有的问题：

- 必须先运行初始查询，然后最终用户才能看到内容，而这实际上是您可提出的最大查询，也就是“给我所有记录”。对任何现实数据集来说，这需要大量时间来执行并回落到 Tableau 引擎。“首次接触”体验对于确定最终用户对该解决方案的看法至关重要，如果过了好几秒，用户还是没有感觉到任何变化，那这无疑负面体验。
- 创建包含几十万甚至数百万个标记（交叉表中的每一个单元格称为一个标记）的视图需要大量占用 CPU 和内存。而且这还需要很多时间 – 从而进一步增加对系统响应度的负面看法。在 Tableau Server 上，让很多人全都生成大型交叉表可能导致性能缓慢，在最差的情况下，系统可能耗尽内存。这可能引起服务器稳定性问题、错误和各种最终用户感觉到不愉快的体验。当然，可以向服务器添加更多内存来尽可能减少这种情况，但这是治标不治本。
- 最后，用户没有联系前后关系的指导，因此无从知道他们最初的筛选器组合是条件过宽还是过窄。报告用户怎么会知道，如果他们检查所有可用的类别，他们的初始查询将返回几万条记录，并耗尽服务器的所有可用 RAM？除非有过痛苦的经历，否则他们不会知道。

与此相反，在第二种方法中，我们的初始查询只显示最高的聚合级别：

- 必须运行的初始查询高度聚合，因此只返回少量记录。对设计良好的数据库来说，这是非常高效的，因此“首次接触”响应时间很快，从而产生对系统的正面看法。随着我们不断下钻，每一个后续查询都是根据更高一级的选择项进行聚合，并局限在这些选择项中。它们仍然可以快速执行并返回到 Tableau 引擎。
- 虽然在仪表板完全完成后，我们得到了更多视图，但是每个视图只显示少量标记。生成每个视图所需的资源（即使在很多最终用户正在使用系统时）微不足道，因此现在系统耗尽内存的可能性大大降低。
- 最后，可以看到，对于更高的“导航”级别，我们利用机会显示了每个类别的销售量。这给了用户一些背景信息，使其可以知道这样的选择是包含很多记录还是很少记录。我们还使用颜色来指出每个类别的盈利能力。现在这就变得意义非常重大了，因为您能看到哪些具体方面需要予以关注，而不是盲目导航。

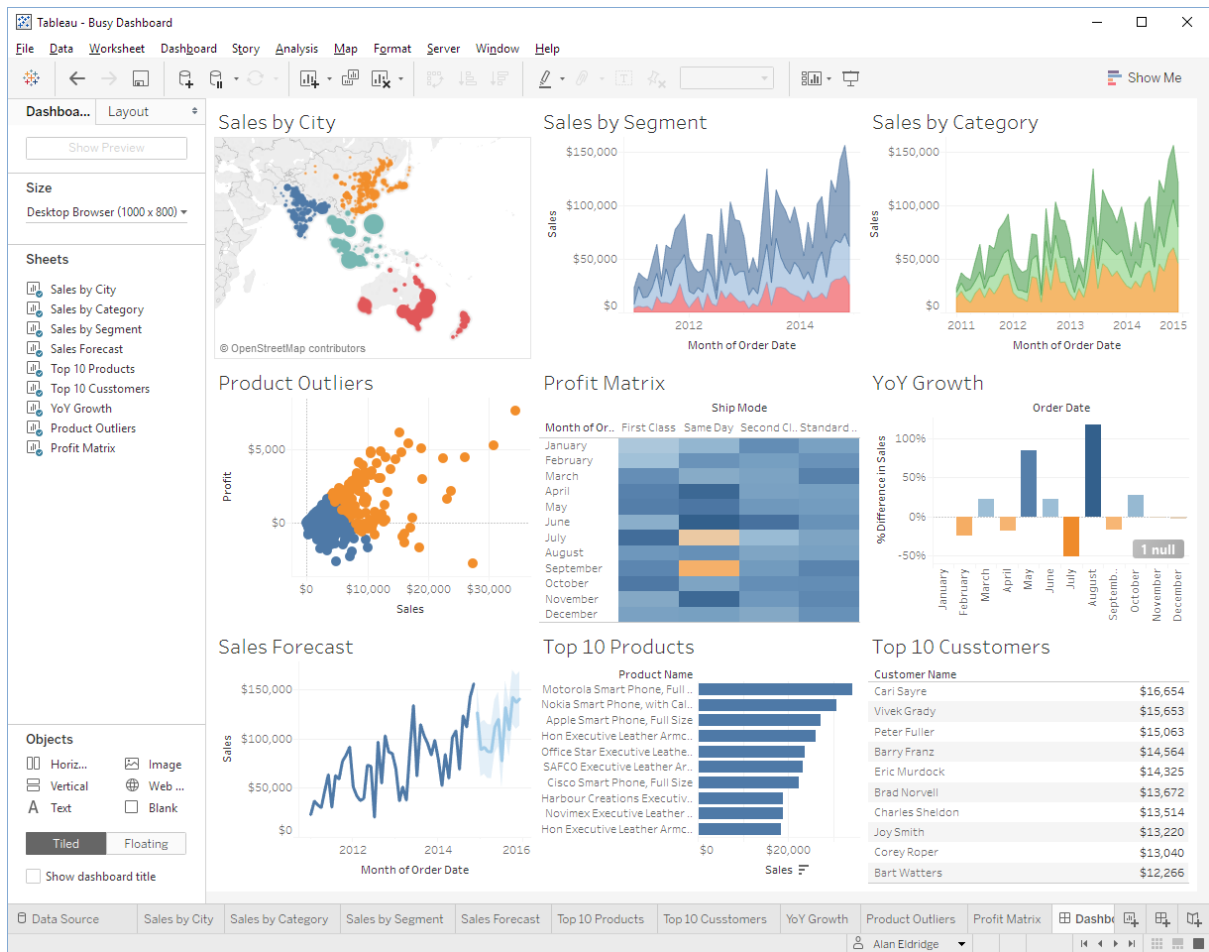
简化设计

新用户常犯的错误是创建过于“复杂”的仪表板。他们可能尝试重新创建曾通过其他工具中使用过的文档，还可能尝试创建专用于打印报告的某些内容。最终生成运行缓慢且低效的工作簿。

以下几点将导致复杂：

每个仪表板包含过多工作表

新用户常犯的错误是尝试在单个仪表板上放置大量图表/工作表。



请注意，每个工作表将针对数据源运行一个（或者更多）查询，因此，工作表越多，呈现仪表板所需时间就越长。Tableau 旨在向最终用户提供交互式仪表板，因此请将数据分散到多个仪表板/页面。

筛选卡过多

筛选卡是 Tableau 的一项非常强大的功能，可用于为最终用户创建丰富的交互式仪表板。但是，每个筛选器可以请求一个查询以枚举选项，因此，向仪表板添加过多筛选器可能意外导致仪表板呈现耗时变长。此外，在筛选器上使用“显示相关值”时，每次更改其他筛选器均需要一个查询才能更新显示值。请谨慎使用此功能。

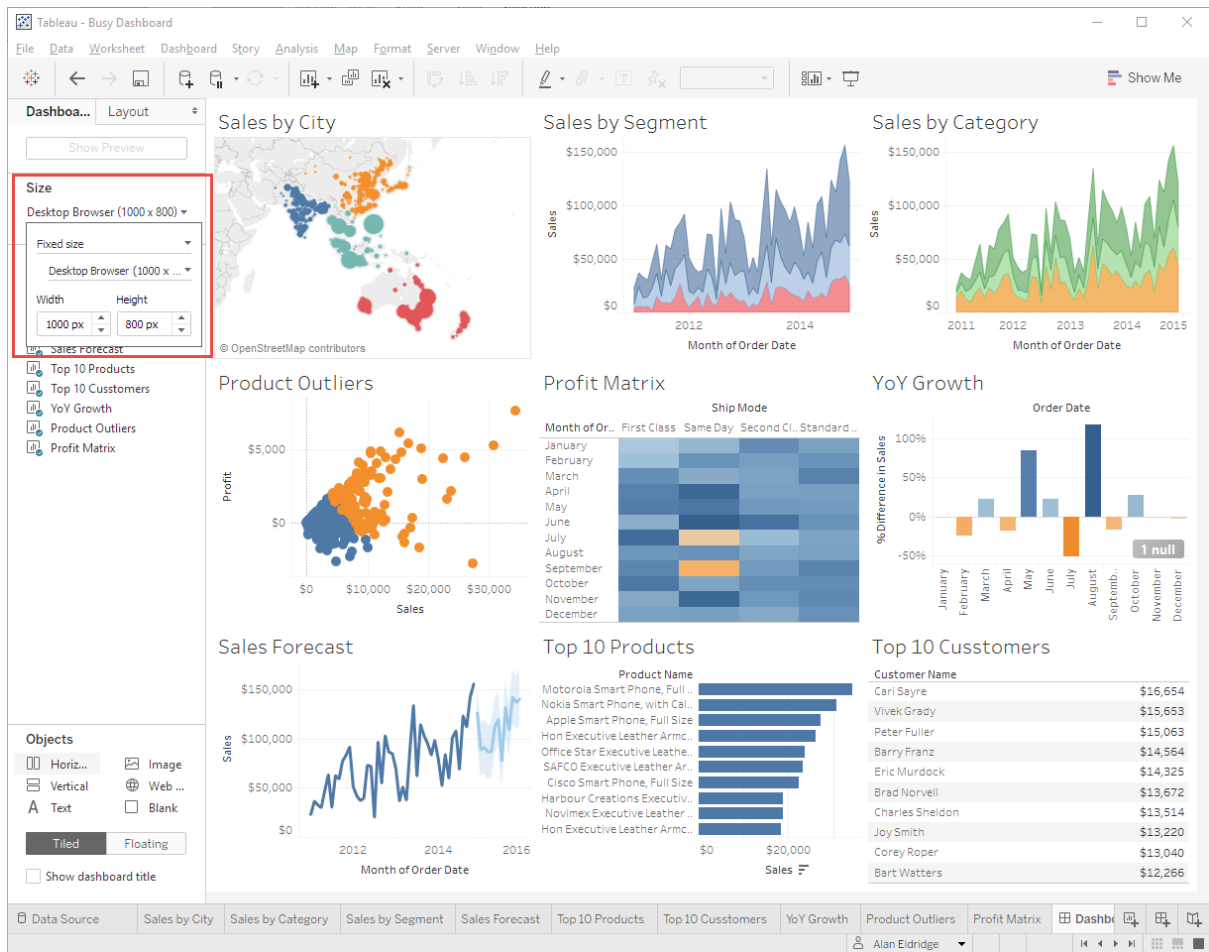
此外，如果将筛选器应用于多个工作表，请注意每次更改都将触发多个查询，因为所有受影响的可见工作表都将更新（不会运行不可见工作表）。如果需要几秒钟方才完成，这将带来糟糕的用户体验。如果期望用户对多筛选器类型进行多处更改，建议显示“应用”按钮，以使用户完成更改其所选内容时可以触发更新。

调整仪表板以提升性能

确保仪表板尽可能简单后，我们可以调整设计来利用缓存，从而进一步提升性能。

固定大小的仪表板

若要提升性能，可执行的最简单做法之一是检查仪表板是否为固定大小。



在 Tableau 中，呈现过程的一部分是创建布局 - 要显示的小型序列图和交叉表的行数/列数；要绘制的轴刻度/网格线数目和间隔；要显示的标记标签数和位置等。这取决于显示仪表板的窗口大小。

如果拥有针对相同仪表板，但源自不同大小窗口的多个请求，则需为每个请求生成布局。通过将仪表板布局设置为固定大小，可确保我们仅需创建所有请求均可重复使用的单个布局。这对服务器端呈现甚至更为重要，因为固定大小的仪表板意味着我们还可以缓存和共享服务器上呈现的位图，从而提升性能和可伸缩性。

设备特定仪表板

在 Tableau 10 中，我们引入了一项称为“设备特定仪表板”的新功能。通过此功能，可以创建基于所用设备自动选择的自定义仪表板布局。

我们可基于屏幕大小选择要使用的布局：

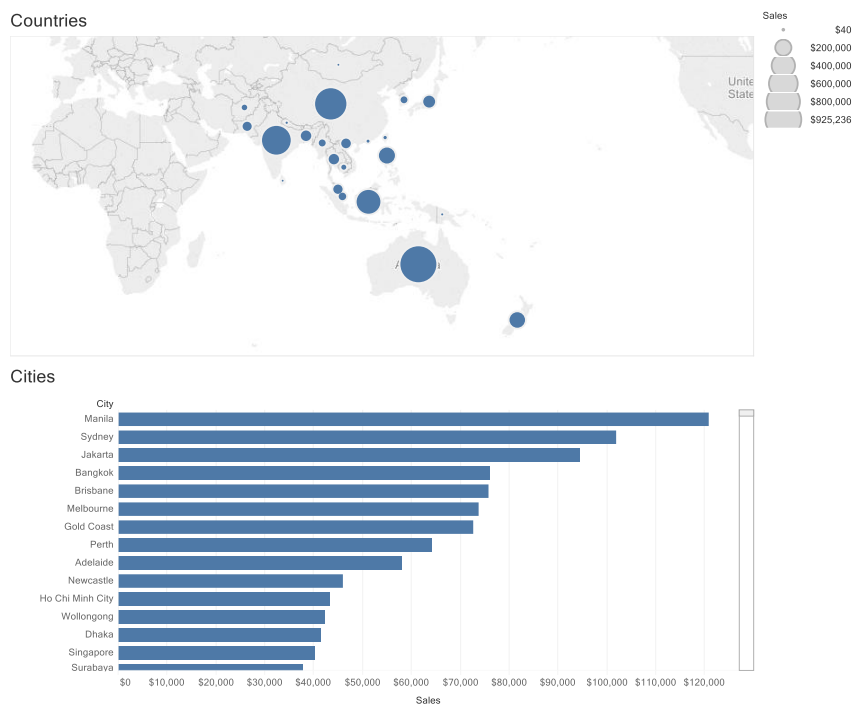
- 最小轴小于或等于 500px - 手机
- 最小轴小于或等于 800px - 平板电脑
- 大于 800px - 桌面

考虑到不同设备具有这些范围内的不同屏幕大小且设备可以旋转，因此通常需要将手机/平板电脑布局设置为自动调整大小。这可以实现不同设备间的最佳视觉体验，但会影响缓存重复使用（演示模式缓存和用于服务器端呈现的图像磁贴缓存）。通常，针对设备适当调整大小的优

点可抵消对缓存带来的影响，但仍需考虑到这种影响。请注意，一旦用户开始使用工作簿，您最终需要为最常见屏幕大小填充模型和位图，以此提升性能。

使用可视化详细级别减少查询

尽管常规的最佳做法是仅使用每个工作表所需的字段，但通过提取某一工作表上的更多信息来避免其他工作表上的查询，有时也可以提升性能。请考虑以下仪表板：



生成预期仪表板，执行计划将出现两个查询 - 分别对应于每个工作表：

Performance Summary

This workbook shows the main events while recording performance. Search Help for details on how to interpret the workbook and improve performance of Tableau.

Show Events taking at least (in seconds): 0.10 2.11

Timeline

Workbook	Dashboard	Worksheet	Event	Time (s)
Book1	Dashboard 1	Countries	Executing Query	807 - 808
		Cities	Executing Query	813 - 815

```
SELECT [Superstore APAC].[City] AS [City],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
GROUP BY [Superstore APAC].[City]
```

```
SELECT [Superstore APAC].[Country] AS [Country],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
GROUP BY [Superstore APAC].[Country]
```

如果更改仪表板设计，并将“Country (国家/地区)”添加到(“详细信息”功能区上的)“City (城市)”工作表，Tableau 仅通过单个查询便可完成仪表板。Tableau 足够智能，它会先运行“City (城市)”工作表的查询，然后使用查询结果缓存数据提供给“Country (国家/地区)”工作表。此功能称为“查询批处理”。

Performance Summary

This workbook shows the main events while recording performance. Search Help for details on how to interpret the workbook and improve performance of Tableau.

Show Events taking at least (in seconds): 0.10 3.02

Timeline

Workbook	Dashboard	Worksheet	Event
Book1	Dashboard 1	Cities	Executing Query

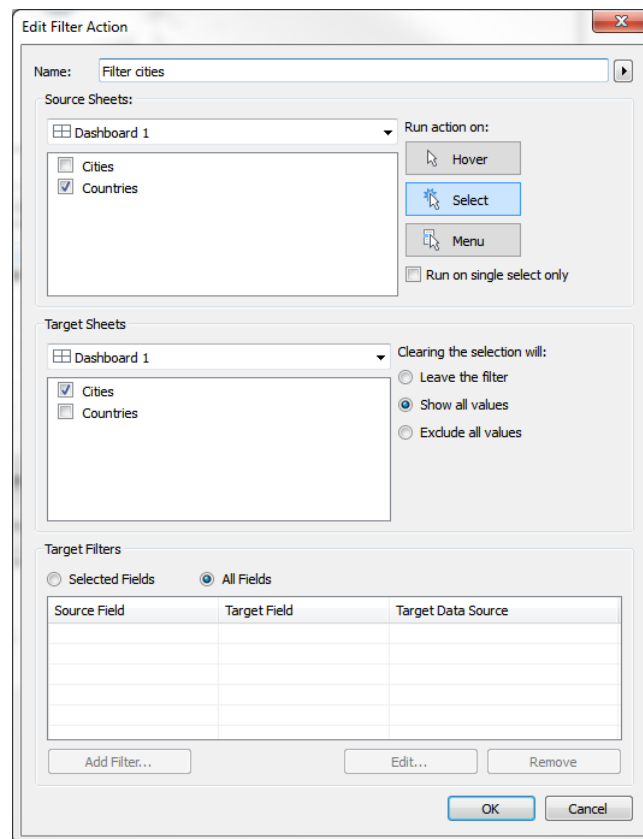
126.5 127.0 127.5 128.0 128.5 129.0 129.5
Time (s)

```
SELECT [Superstore APAC].[City] AS [City],
       [Superstore APAC].[Country] AS [Country],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
GROUP BY [Superstore APAC].[City],
         [Superstore APAC].[Country]
```

很明显，并非所有情况下都可执行此操作，因为向可视化添加维度会更改详细级别，进而可能导致显示更多标记。但是，这在数据具有如上例所示分层关系时非常有用，因为它不会影响可见的详细级别。

使用可视化详细级别优化操作

我们可以使用类似方法通过操作减少需要运行的查询数。考虑以上（优化之前）仪表板，但现在将筛选动作从“Country（国家/地区）”工作表添加到“City（城市）”工作表。



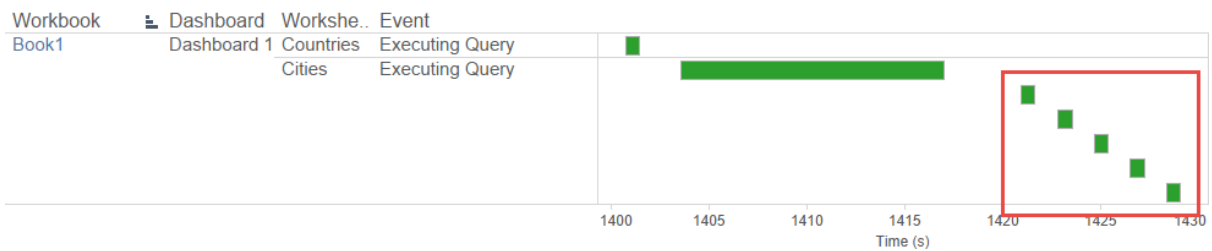
通过单击地图上的标记触发此操作时，我们会看到 Tableau 需要运行查询才能确定“City（城市）”工作表中的值。这是因为城市-国家/地区关系的查询结果缓存中不存在任何数据。

Performance Summary

This workbook shows the main events while recording performance. Search Help for details on how to interpret the workbook and improve performance of Tableau.

Show Events taking at least (in seconds): 0.10  13.47

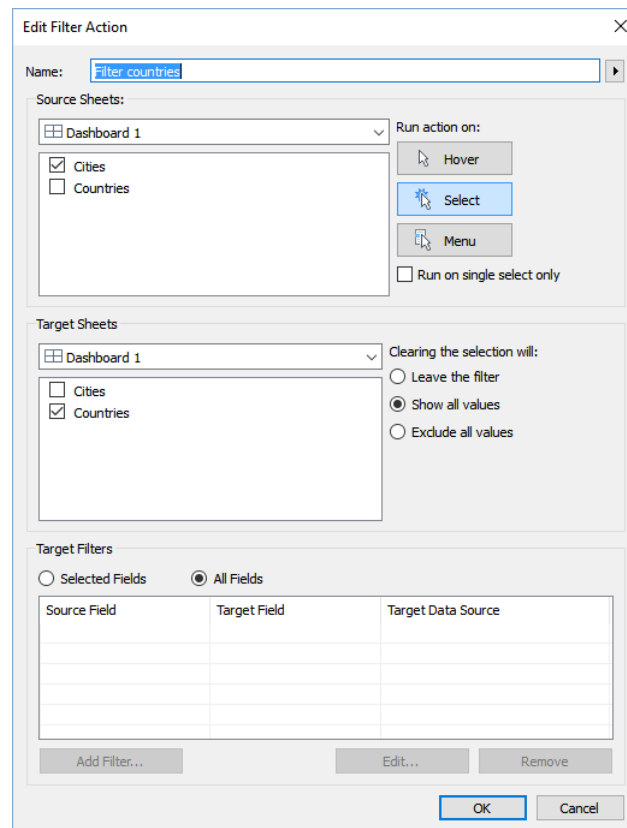
Timeline



```
SELECT [Superstore APAC].[City] AS [City],  
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]  
FROM [dbo].[Superstore APAC] [Superstore APAC]  
WHERE ([Superstore APAC].[Country] = 'Australia')  
GROUP BY [Superstore APAC].[City]
```

如果将“Country (国家/地区)”添加到“City (城市)”工作表，查询结果缓存中现在就有的足够信息，因此无需返回数据源即可执行这些筛选。

当源工作表比目标工作表更详细时，可以进行类似的优化。如果采用默认操作定义，使用“全部字段”进行筛选：



Dialog Box: Edit Filter Action

Name: filter countries

Source Sheets:

- Dashboard 1
- Cities
- Countries

Run action on:

-
-
-

Run on single select only

Target Sheets:

- Dashboard 1
- Cities
- Countries

Clearing the selection will:

- Leave the filter
- Show all values
- Exclude all values

Target Filters:

- Selected Fields
- All Fields

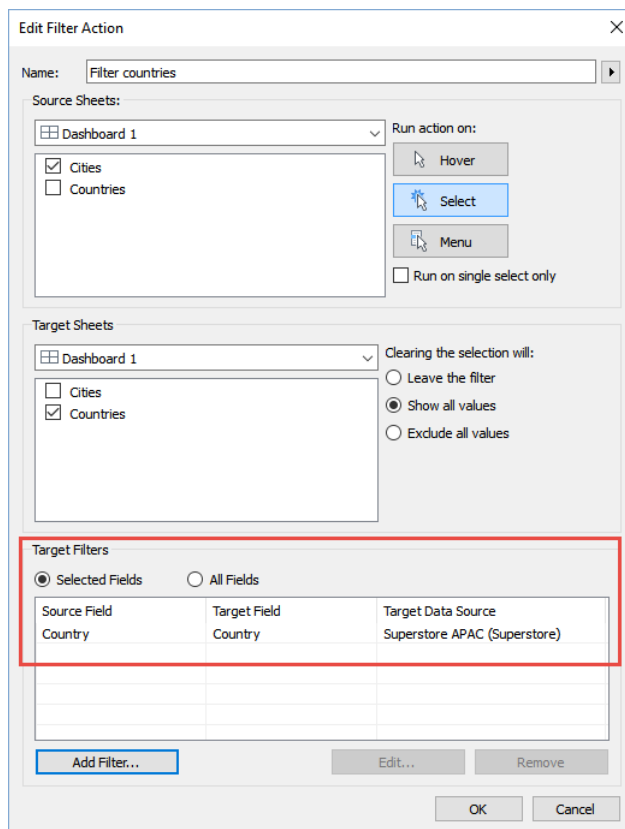
Source Field	Target Field	Target Data Source

Buttons: Add Filter..., Edit..., Remove, OK, Cancel

这会导致工作簿为每个操作运行一个查询，因为筛选子句将引用 “Country (国家/地区)” 和 “City (城市)” ，而后者无法从 “Country (国家/地区)” 工作表的查询结果缓存中获取。

```
SELECT [Superstore APAC].[Country] AS [Country],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
WHERE (([Superstore APAC].[City] = 'Sydney') AND ([Superstore APAC].[Country] = 'Australia'))
GROUP BY [Superstore APAC].[Country]
```

如果将操作更改为仅基于 “Country (国家/地区)” 进行筛选：



现在从查询结果缓存即可完成此筛选，因此不需要返回查询数据源。如上所述，需要考虑更改详细级别是否会影响工作表设计，如不影响，则可使用此技巧。

优秀的工作表设计

仪表板的下一级是工作表。在 Tableau 中，工作表的设计本质上与运行的查询相关。每个工作表将生成一个或多个查询，在此阶段我们将尝试确保生成可能的最优查询。

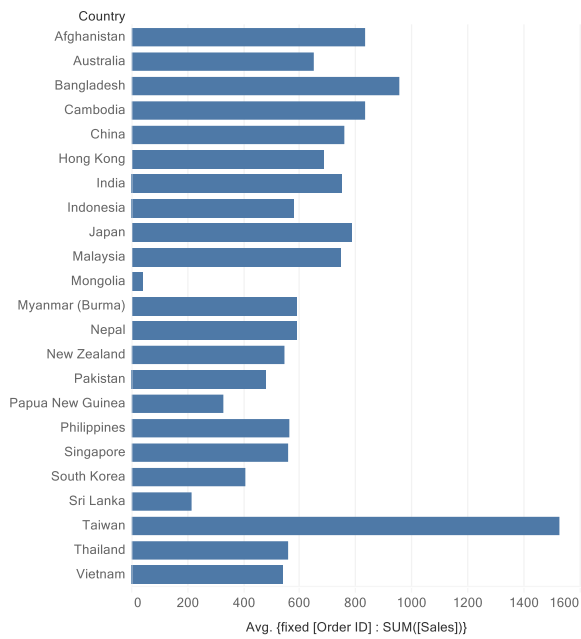
仅包括所需字段。

查看详细信息功能区，删除未直接用于可视化的、不为工具提示所需或提供要求级别标记详细信息的任何字段。这可加速数据源中的查询，并减少查询结果中所需返回数据。为帮助查询批处理在其他工作表中消除相同查询，我们发现此规则存在某些异常，但这些异常取决于是否更改工作表的可视化详细级别，不更改级别可减少异常。

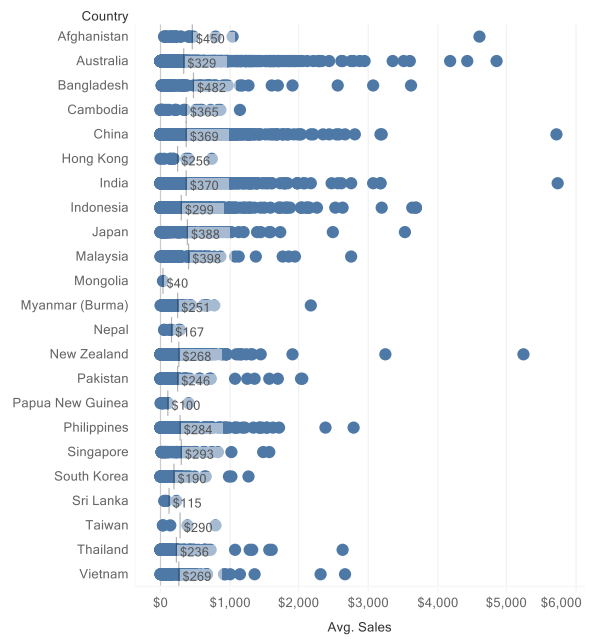
显示回答问题所需的最小标记数

在 Tableau 中，计算相同数字的方法有很多。请考虑以下仪表板 - 两个工作表都可解答 “每个国家/地区的平均订货量是多少？” 问题

Avg Order Size 1



Avg Order Size 2

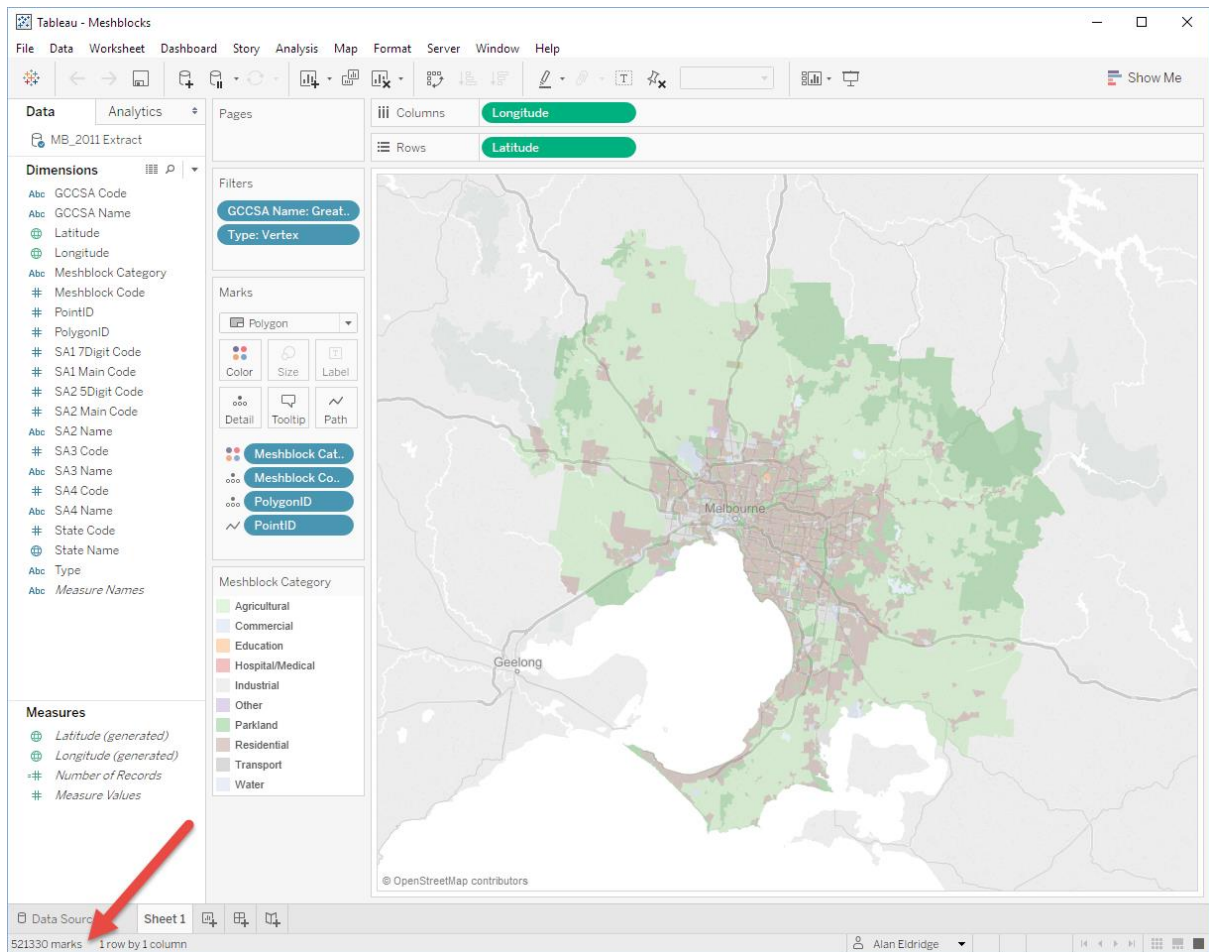


工作表 1 仅显示单个标记，表示每个国家/地区的平均订货量，因此仅从数据源返回 23 条记录。而工作表 2 为每个国家/地区的每笔订单显示一个标记，然后计算平均值作为参考线。这需从数据源提取 5436 条记录。

如果仅对原问题（每个国家/地区的平均订货量是多少）感兴趣，则工作表 1 是更好的解决方案，但工作表 2 不仅回答了这个问题，还提供了有关订货量范围的深入见解，让我们能够识别异常值。

避免高度复杂的可视化

应查看的一项重要指标是每个可视化中呈现的数据点数目。查看 Tableau Desktop 窗口中的状态栏可轻松找到此数据：

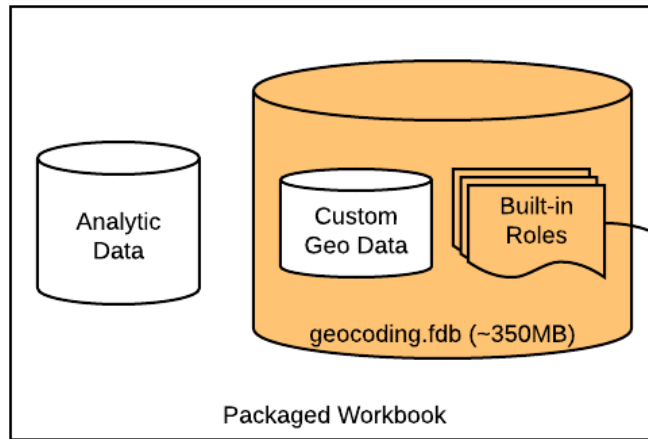


尽管不存在定义“过多标记”的硬性规定，但请注意，标记越多，呈现它们所需的 CPU 和 RAM 就越多。留意大型交叉表、散点图和/或包含复杂自定义多边形的地图。

地图

自定义地理编码

导入自定义地理编码角色时，会将其写入地理编码数据库，即 Firebird DB 文件，默认存储于 C:\Program Files\Tableau\Tableau 10.0\Local\data\geocoding.fdb。如果在工作簿中使用该角色并将其保存为打包工作簿，整个数据库文件都会被压缩成 TWBX 文件 - 总共约 350MB！

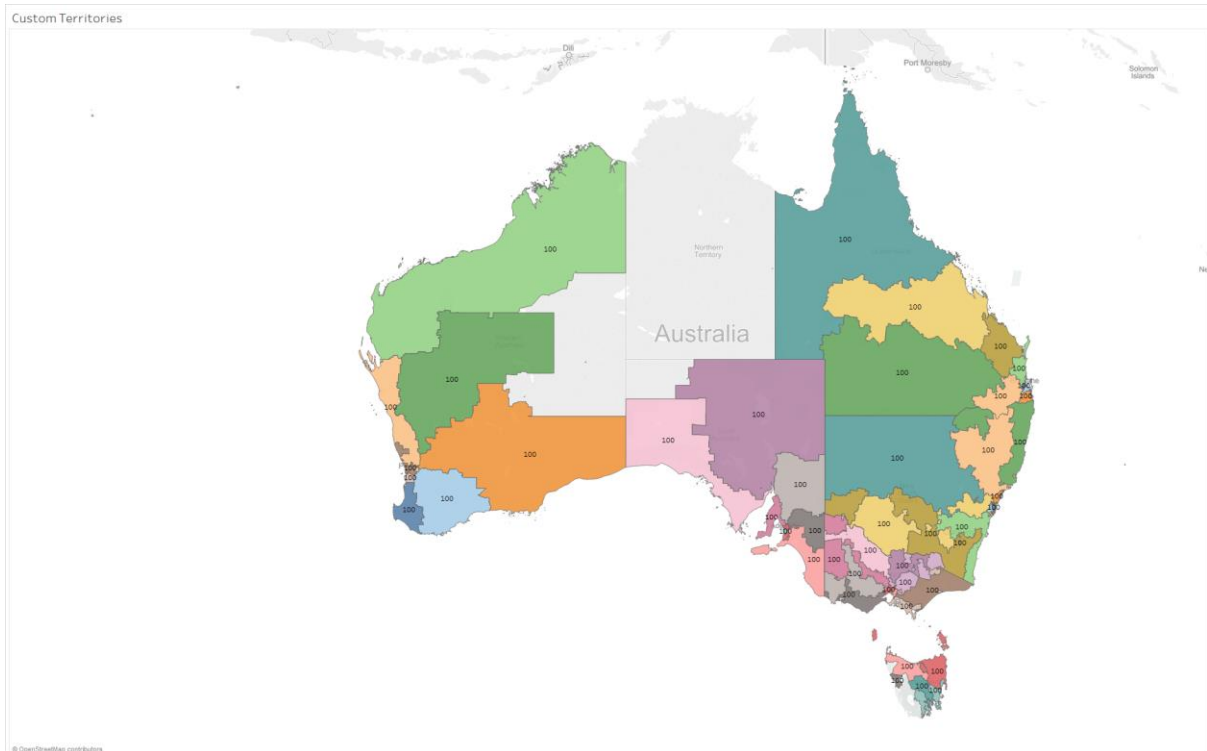


Name	Type	Compressed size	Password ...	Size	Ratio	Date modified
AreaCode.tds	Tableau Datasource	2 KB	No	9 KB	82%	9/06/2016 1:29 PM
City.tds	Tableau Datasource	2 KB	No	13 KB	85%	9/06/2016 1:29 PM
CMSA.tds	Tableau Datasource	2 KB	No	14 KB	86%	9/06/2016 1:29 PM
Congress.tds	Tableau Datasource	2 KB	No	14 KB	86%	9/06/2016 1:29 PM
Country.tds	Tableau Datasource	3 KB	No	17 KB	87%	9/06/2016 1:29 PM
County.tds	Tableau Datasource	2 KB	No	14 KB	86%	9/06/2016 1:29 PM
GEOCODING.FDB	FDB File	118,407 KB	No	359,280 KB	68%	9/06/2016 1:28 PM
State.tds	Tableau Datasource	2 KB	No	14 KB	86%	9/06/2016 1:29 PM
Suburb.tds	Tableau Datasource	3 KB	No	15 KB	86%	9/06/2016 1:29 PM
ZipCode.tds	Tableau Datasource	2 KB	No	8 KB	81%	9/06/2016 1:29 PM

这使得生成的 TWBX 文件 a) 非常大，因为地理编码负载压缩后约 110MB；b) 打开非常缓慢，因为初始未压缩文件必须对大得多的数据集执行操作。更有效的一种方法是，不将数据作为自定义地理编码角色导入，而在工作簿内使用混合功能，组合分析数据和地理空间数据。使用此方法，不会嵌入 geocoding.fdb 文件，TWBX 仅包含分析数据和自定义地理编码数据。

自定义区域

自定义区域是 Tableau 10 的一个新功能，使用此功能，用户可以合并内部地理编码数据库中的区域，以便创建聚合区域。



基于多个较低级别区域的自定义区域最初呈现可能非常缓慢，因此请谨慎使用此功能。不过，一旦完成，便会缓存自定义区域且预期性能良好。

已填充地图与点地图

使用客户端呈现时，已填充地图（无论在地图上使用还是用作某些其他类型图表上的标记）成本很高，因为需要发送多边形数据以呈现形状，而这相当复杂。如果发现呈现性能缓慢，建议改为使用符号地图。

多边形标记

使用多边形标记的任何可视化都将强制 Tableau Server 执行[服务器端呈现](#)，这可能影响最终用户体验。请谨慎使用多边形标记。

其他因素

大型交叉表

本文档的早期版本中建议避免使用大型交叉表，因为其呈现非常缓慢。最近的版本中已改善此可视化类型的基础机制，交叉表的呈现速度现与其他小型多图表类型一样快。但是，仍建议仔细考虑是否使用大型交叉表，因为它们需要从基础数据源读取大量数据，并且对于分析也并无用处。

工具提示

默认情况下，在工具提示功能区中设置维度会导致其使用 `ATTR()` 属性函数进行聚合。这需要在基础数据源中执行两种聚合（即 `MIN()` 和 `MAX()`），并将二者所得结果传递回结果集。有关详细信息，请参阅本文档稍后介绍的[使用 `ATTR\(\)`](#) 部分。

如果不考虑是否拥有多个维度值，更有效的解决方案是仅使用一种聚合，而不是默认 `ATTR()`。选取 `MIN()` 或 `MAX()` - 具体使用何种聚合无关紧要，重要的是选择并坚持使用一种聚合，以便最大限度提高缓存命中率。

其他选项（如确信不会影响可视化的可见详细级别）是指在详细级别功能区，而非工具提示功能区设置维度。这会导致查询的 SELECT 和 GROUP BY 子句直接使用维度字段。建议测试此操作性能是否优于单个聚合，因为它可能依赖于数据平台的性能。

图例

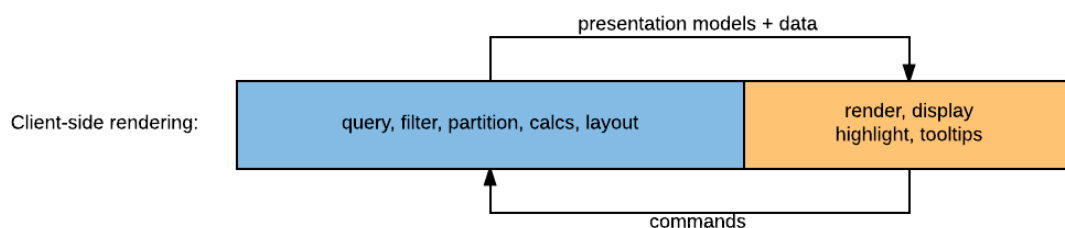
通常，图例不会引起性能问题，因为其域由查询结果缓存填充。但是，如果枚举大型域，图例就可能增加呈现开销，因为需要将数据传输到客户端浏览器。如果是这种情况，则图例通常是无益的，因此请将其删除。

页面功能区

一些用户认为页面功能区的运作方式与筛选器功能区相同，即它可以减少从数据源返回的记录数。这是并不正确 - 工作表查询将为所有页面上所有标记返回记录。如果页面维度势度较高（即包含大量唯一值），这可能大大增加工作表查询的大小，从而影响性能。请谨慎使用此功能。

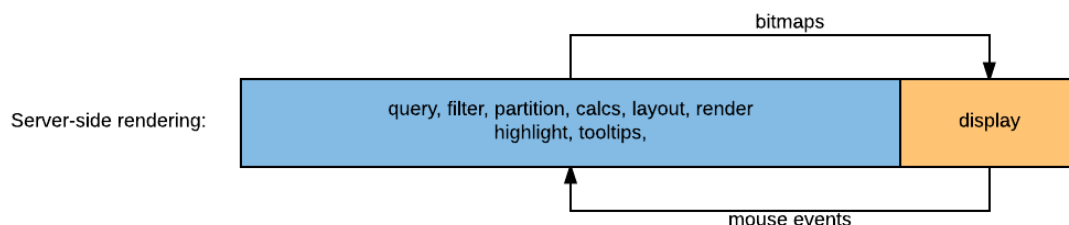
客户端呈现与服务器端呈现

先检索、解读并呈现视图的标记和数据，然后才在客户端 Web 浏览器中显示这些标记和数据。Tableau Server 可以在客户端 Web 浏览器或服务器上执行此过程。客户端呈现是默认模式，因为处理呈现以及与服务器的所有交互可能产生更多网络数据传输以及更长的往返延迟。通过客户端呈现，许多视图交互操作会更快，因为这些交互操作直接在浏览器中进行解读和呈现。



（蓝色 - 在服务器上执行；橙色 - 在客户端浏览器中执行。）

但是某些视图在计算能力更高的服务器上呈现起来更高效。如果视图非常复杂，以至于图像文件占用的带宽要远远小于用来创建图像的数据，那么对这样的视图，服务器端呈现将合理可行。此外，由于平板电脑的性能通常远远不如 PC，因此它们可处理的视图复杂度较低。但是，缺陷在于浏览器端呈现中的工具提示和突出显示等简单交互可能非常缓慢，因为它们需要服务器往返。



（蓝色 - 在服务器上执行；橙色 - 在客户端浏览器中执行。）

Tableau Server 配置为使用“复杂度阈值”作为触发器来自动处理所有这些情况，超过此阈值时，触发在服务器上呈现视图，反之则触发在 Web 浏览器中呈现。PC 和移动设备的阈值有

所不同，因此可能出现自 PC Web 浏览器打开的视图为客户端呈现，而自平板电脑 Web 浏览器打开的相同视图为服务器呈现的情况。筛选还可能更改呈现行为 - 起初可能使用服务器端呈现打开工作簿，应用筛选器之后则更改为通过客户端呈现打开。此外，如果可视化使用多边形标记或页面功能区，则该可视化将仅使用服务器端呈现，即使它在其他方面符合客户端呈现的条件也不例外，因此请谨慎使用这些功能。

作为管理员，您可以针对 PC 和平板电脑测试或调优此设置。有关更多详细信息，请参阅下面的链接：

https://onlinehelp.tableau.com/current/server/zh-cn/browser_rendering.htm

高效筛选器

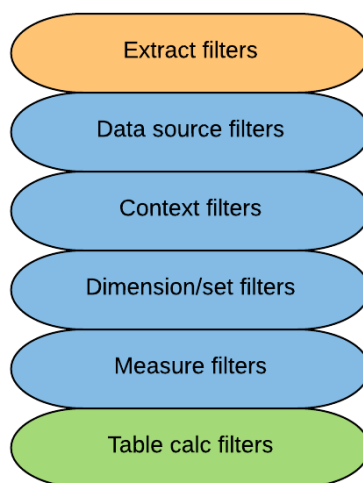
Tableau 中的筛选极其强大且容易表达。但是，低效的筛选器是工作簿和仪表板性能低下的最常见原因之一。下面的部分将展示多种使用筛选器的最佳做法。

注 - 数据源中索引存在与否，以及索引的维护情况对筛选器的效率有显著影响。有关更多详细信息，请参阅有关索引的部分。

筛选器类型

筛选器优先级

在 Tableau 中，按以下顺序应用筛选器：



数据提取筛选器

这些筛选器仅在使用数据提取时可用，但在这种情况下，逻辑上会在应用所有其他筛选器之前应用数据提取筛选器。它们会限制从基础数据源提取的数据，并且可能是维度筛选器或度量筛选器。此外，根据源数据平台，它们还可能执行 TOP 或 SAMPLE 以减少返回的记录数。

数据源筛选器

数据源筛选器是实时连接上可用的最高级别筛选器。数据源筛选器和上下文筛选器的主要区别在于，数据源筛选器的范围限于整个数据源，上下文筛选器则专为每个工作表而设置。这意味着，在已发布数据源中使用时，可以强制使用数据源筛选器，上下文筛选器则在工作表所在级别应用。

数据源筛选器可作为在数据源上设置约束，以阻止最终用户意外运行大量查询的一种有效方式 - 例如，您可以设置数据源筛选器，以便将交易表上的查询限制为仅最近 6 个月的查询。

上下文筛选器

默认情况下，Tableau 中设置的所有筛选器都是独立计算的。也就是说，每个筛选器都会访问数据源中的所有行，而与其他筛选器无关。但是，通过指定上下文筛选器，可以使您定义的任何其他筛选器产生从属关系，因为这些筛选器仅处理通过上下文筛选器传递的数据。

需要获取正确答案（例如，筛选出的 TopN）时，应使用上下文筛选器。例如，考虑采用某种视图来显示按区域筛选，SUM(Sales)（销售额总和）排名前 10 的产品。如不使用上下文筛选器，则运行以下查询：

```
SELECT [Superstore APAC].[Product Name] AS [Product Name],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
     INNER JOIN (
       SELECT TOP 10 [Superstore APAC].[Product Name] AS [Product Name],
                    SUM([Superstore APAC].[Sales]) AS [$__alias__0]
       FROM [dbo].[Superstore APAC] [Superstore APAC]
       GROUP BY [Superstore APAC].[Product Name]
       ORDER BY 2 DESC
     ) [t0] ON ([Superstore APAC].[Product Name] = [t0].[Product Name])
WHERE ([Superstore APAC].[Region] = 'Oceania')
GROUP BY [Superstore APAC].[Product Name]
```

这将返回全球排名前 10 的产品在大洋洲所占份额。如果真正希望了解的是大洋洲区域内排名前 10 的产品，可将“区域”筛选器添加到上下文，然后运行以下查询：

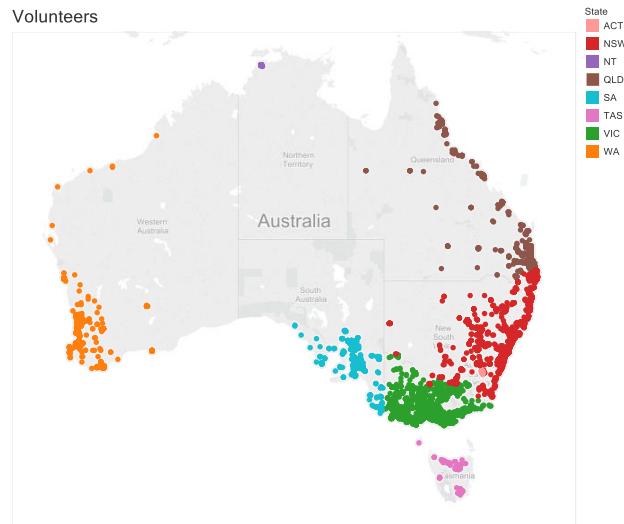
```
SELECT TOP 10 [Superstore APAC].[Product Name] AS [Product Name],
              SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok],
              SUM([Superstore APAC].[Sales]) AS [$__alias__0]
FROM [dbo].[Superstore APAC] [Superstore APAC]
WHERE ([Superstore APAC].[Region] = 'Oceania')
GROUP BY [Superstore APAC].[Product Name]
ORDER BY 3 DESC
```

过去，上下文筛选器被实现为数据源中的临时表。现已不再如此 - 在大多数情况下，会将上下文筛选器实现为数据源查询（如上所示）的一部分或在数据引擎本地处理上下文筛选器。

不宜再将上下文筛选器用作提升查询性能的机制。

筛选分类维度

请考虑下面的可视化图表 - 一张澳大利亚地图，上面标有邮政编码标记：



有多种方法可以筛选该地图，使其只显示西澳的邮政编码（橙色点）：

- 我们可以选择所有西澳的标记，然后只保留选定的标记；
- 我们可以选择西澳以外的所有标记，然后排除选定的标记；
- 我们可以只保留另一个属性，如“State（州）”维度；
- 我们可以按范围筛选 – 按邮政编码值筛选或按纬度/经度值筛选。

离散

如果使用前两个选项，我们会发现只保留和排除选项性能不佳 – 事实上，它们通常可能比未筛选的数据集还要慢。这是因为，它们表示为邮政编码值的离散列表。邮政编码值是由 DBMS 通过 WHERE IN 子句，或（如存在多个值）通过创建内嵌所选值的临时表并使用与之关联的内联接和主表筛选出来的。对于大量标记，这可能导致查询的计算开销很大。

第三个选项在这个例子中很快，因为得出的筛选条件 (WHERE STATE="Western Australia") 非常简单，可以由数据库高效处理。但是，随着表示筛选条件所需的维度成员数不断增加，此方法会变得越来越低效 – 最终其性能会与套索且只保留选项一样。

值范围

使用值范围筛选方法同样可以使数据库计算简单的筛选子句 (WHERE POSTCODE >= 6000 AND POSTCODE <= 7000 或 WHERE LONGITUDE < 129)，实现快速执行。但是，与基于相关维度的筛选不同，此方法不会随着维度基数的增加而变得越来越复杂。

上述内容的结论是，值范围筛选通常比计算很大的离散值逐项列表要快，与只保留或排除相比，对于大量标记，应尽可能优先使用范围筛选。

切片筛选器

切片筛选器是维度上的筛选器，该维度不会在可视化中使用（即，它们不是可视化详细级别的组成部分）。例如，您可能拥有一个按国家/地区显示总销售额的可视化，但该可视化却是按区域筛选的。在本例中，查询运行如下：

```
SELECT [Superstore APAC].[Country] AS [Country],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
WHERE ([Superstore APAC].[Region] = 'Oceania')
GROUP BY [Superstore APAC].[Country]
```

如果我们切割聚合结果，这些筛选器会变得更加复杂。例如，如果我们不按区域筛选上述可视化，而是将其用于显示盈利排名前 10 的产品销售额，则 Tableau 需要运行两个查询 - 一个在产品级别分隔盈利排名前 10 的产品，另一个处于国家/地区级别，且受限于第一个查询的结果：

```
SELECT [Superstore APAC].[Country] AS [Country],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
     INNER JOIN (
       SELECT TOP 10 [Superstore APAC].[Product Name] AS [Product Name],
                  SUM([Superstore APAC].[Profit]) AS [$_alias_0]
       FROM [dbo].[Superstore APAC] [Superstore APAC]
       GROUP BY [Superstore APAC].[Product Name]
       ORDER BY 2 DESC
     ) [t0] ON ([Superstore APAC].[Product Name] = [t0].[Product Name])
GROUP BY [Superstore APAC].[Country]
```

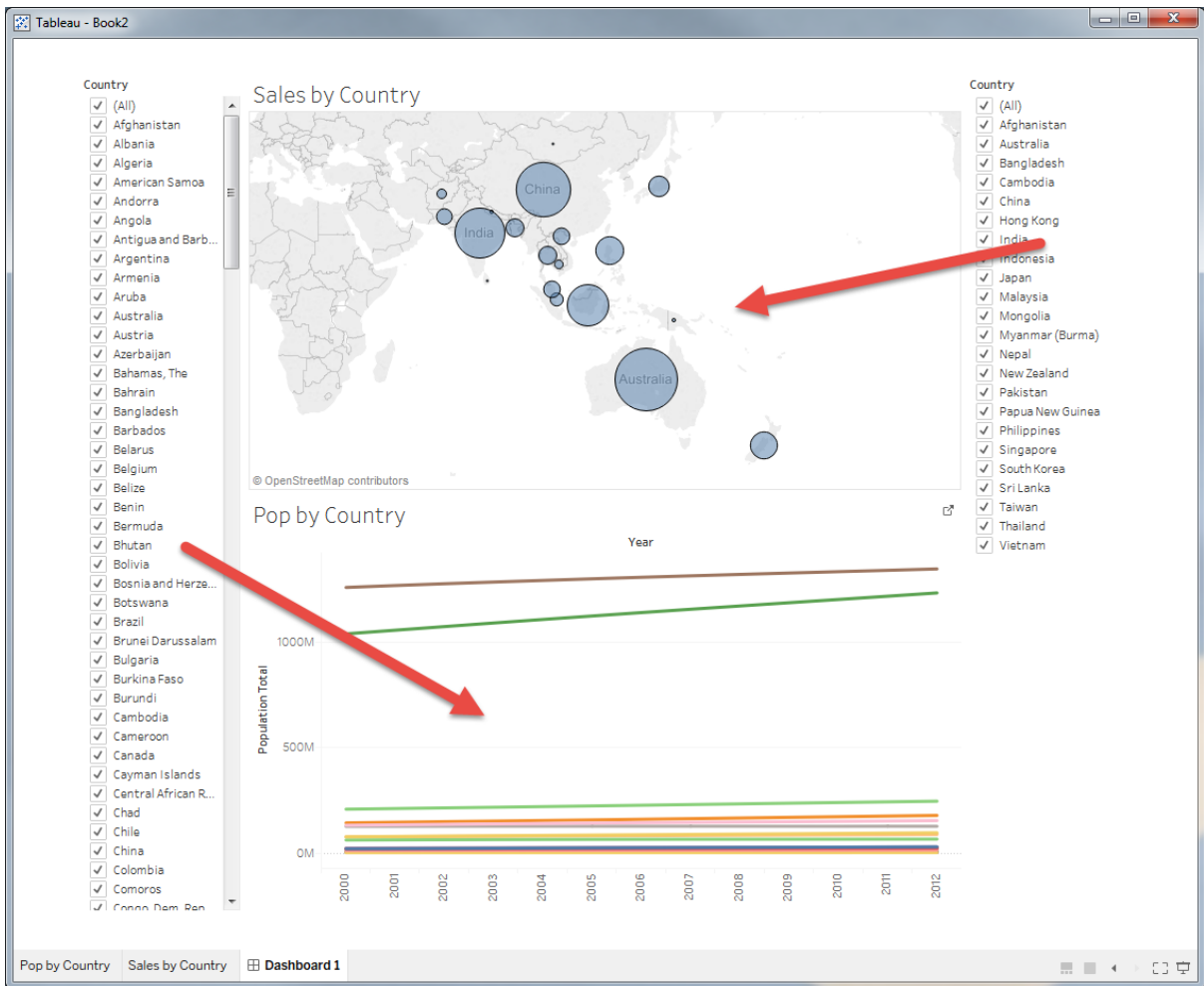
谨慎使用切片筛选器，因为其计算开销可能很大。另请注意，由于维度并非查询结果缓存的一部分，因此我们无法在切片筛选器上快速执行浏览器内筛选（有关客户端呈现与服务器端呈现的信息，请参阅[前面部分](#)）。

跨数据源筛选器

跨数据源筛选器是 Tableau 10 的一项新增功能。借助本功能，您可以将筛选器应用到多个数据源，它们通常包含一个或多个相同的字段。采用与混合相同的方式定义关系 - 基于名称/类型匹配自动定义或通过“数据”菜单中的自定义关系手动定义。

跨数据库筛选器存在与仪表板上的快速筛选器相同的性能隐忧。更改这些筛选器可能导致更新多个区域，进而可能需要多个查询。请审慎使用；如果希望用户进行多处更改，建议显示“应用”按钮，以便仅在所选内容完成时启动查询。

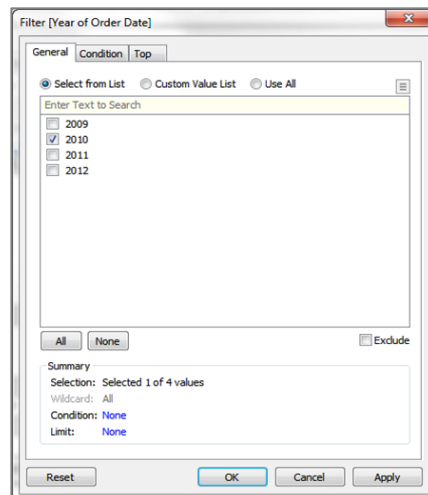
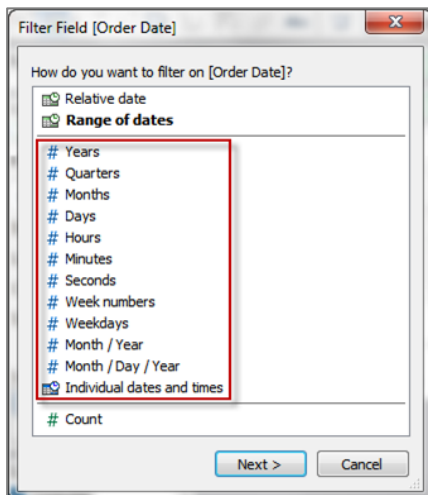
另请注意，筛选器域摘取自“主”数据源 - 即，创建筛选器的工作表首次使用的数据源。如果相关字段在不同数据源中的域有所不同，需要仔细考虑使用哪个域，因为同一筛选器可能以显示不同的值而结束，如下所示：



筛选日期：离散、范围、相对

日期字段是一种特殊的维度，Tableau 处理这种维度的方式通常不同于标准分类数据。在创建日期筛选器时尤其如此。日期筛选器是十分常见的筛选器，可分为三个类别：相对日期筛选器（显示相对于特定日的日期范围）、日期范围筛选器（显示定义的离散日期范围）和离散日期筛选器（显示从列表中选择个别日期）。如前面部分所示，所用方法对最终查询的效率有实质性的影响。

离散



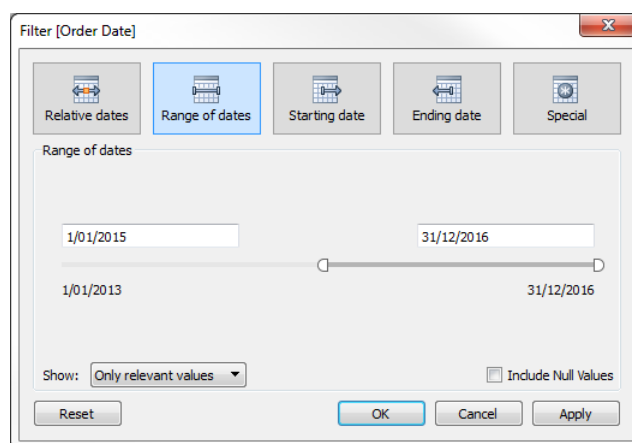
有时，您可能需要筛选来包含特定的个别日期或整个日期级别。这种筛选器称为离散日期筛选器，因为您定义的是离散值而不是范围。这种筛选器类型会产生将作为动态计算传给数据库的日期表达式：

```
SELECT [FactSales].[Order Date], SUM([FactSales].[SalesAmount])
FROM [dbo].[FactSales] [FactSales]
WHERE (DATEPART(year,[FactSales].[Order Date]) = 2010)
GROUP BY [FactSales].[Order Date]
```

大多数情况下，查询优化程序会智能地为 DATEPART 计算求值，但是某些情况下，使用离散日期过滤器可能导致较差的查询执行。例如，查询一个分区表，而表有一个离散日期筛选器建在日期分区键上。由于表不是基于 DATEPART 值分区的，因此某些数据库会不起作用并转而跨所有分区来求出该计算的值，以找出符合条件的记录，尽管这样做没有必要。在这种情况下，使用“日期范围”筛选器或具有指定锚点日期的相对日期筛选器可以大幅度提升性能。

优化这类筛选器性能的一种方法是，使用数据提取具体化计算。首先，创建一个显式实现 DATEPART 函数的计算字段。如果您随后创建 Tableau 数据提取，此计算字段将具体化为提取中的存储值（因为该表达式的输出是确定的）。筛选计算字段而不是动态表达式会更快，因为只需要查找值，而不是在查询时计算值。

日期范围

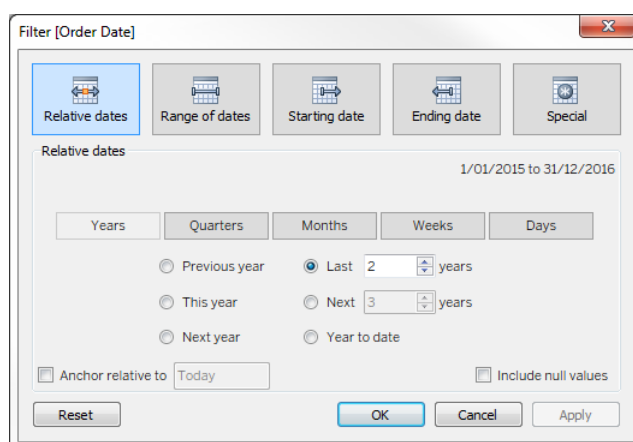


如果您要指定连续日期范围，那么可以使用范围筛选器。它可产生传给数据库的以下查询结构：

```
SELECT SUM([factOrders].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[factOrders] [factOrders]
WHERE (([factOrders].[Order Date] >= {d '2015-01-01'}) AND
([factOrders].[Order Date] <= {d '2016-12-31'}))
GROUP BY ()
```

这种 WHERE 子句对查询优化器来说非常高效，因为这能让执行计划充分利用索引和分区。如果在添加离散日期筛选器后发现查询时间很长，请考虑将其替换为范围日期筛选器，然后看看有没有变化。

相对



通过相对日期筛选器，可以定义基于打开视图时的日期和时间进行更新的日期范围。例如，您可能需要查看本年迄今的销售额、过去 30 天的所有记录或上周消除的错误。相对日期筛选器也可以相对于特定的锚点日期而不是相对于今天。

```
SELECT SUM([factOrders].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[factOrders] [factOrders]
WHERE (([factOrders].[Order Date] >= {ts '2015-01-01 00:00:00'}) AND
([factOrders].[Order Date] < {ts '2017-01-01 00:00:00'}))
GROUP BY ()
```

如您所见，得到的 WHERE 子句使用日期范围语法，因此这也是高效形式的日期筛选器。

请注意，由于相对于当前日期/时间的日期筛选器的可变性（要么通过上述相对日期筛选器选项更改，要么通过显式使用筛选公式中的 NOW() 或 TODAY() 更改），查询缓存不如使用日期筛选器的查询来得有效。Tableau 会在缓存服务器中将这此查询标记为“临时查询”，其结果保留时间短于其他查询结果保留时间。

筛选卡

显示过多筛选卡将减缓运行速度，将其设置为“仅相关值”并得到众多离散列表时尤甚。请尝试更有引导性的分析方法，并在仪表板中改用操作筛选器。如果为了让视图达到高交互性而建立包含大量筛选器的视图，那么应该自问，分不同级别和主题的多个仪表板是否效果更好（提示：是的，很可能效果更好）。

枚举和非枚举

枚举筛选卡要求 Tableau 查询数据源以获得字段值，然后才能呈现筛选卡。这些对象包括：

- 多值列表 - 全部维度成员
- 单值列表 - 全部维度成员
- 精简列表 - 全部维度成员
- 滑块 - 全部维度成员
- 度量筛选器 - 最小和最大值
- 范围日期筛选器 - 最小和最大值

另一方面，非枚举筛选卡不需要知道潜在的字段值。这些对象包括：

- 自定义值列表

- 通配符匹配
- 相对日期筛选器
- 浏览周期日期筛选器。

因此，非枚举筛选卡减少了需要由数据源执行的筛选相关查询的数量。另外，要显示的维度成员过多时，非枚举筛选卡呈现更快。

虽然使用非枚举筛选卡可以提高性能，但这是以牺牲最终用户的视觉上下文为代价的。

相关值

枚举筛选卡可设置为以三种方式显示潜在字段值：

- 数据库中的所有值 - 选择此选项时，数据库中的所有值都会显示，而不管视图中是否还有其他筛选器。其他筛选器更改时，该筛选器无需重新查询数据库。
- 上下文中的所有值 - 只有在启用了上下文筛选器时，该选项才可用。筛选卡将显示上下文中的所有值，无论视图中是否还有其他筛选器。维度或度量发生更改时，筛选器无需重新查询数据库，但如果上下文更改，则需要重新查询。
- 仅相关值 - 选择此选项时，将会考虑其他筛选器，并且仅显示通过这些筛选器的值。例如，如果在“Region（区域）”上设置了筛选器，则“State（州）”上的筛选器将仅显示“Eastern（东部）”各州。因此，其他筛选器更改时，该筛选器必须重新查询数据源。

可以看到，“仅相关值”设置对于帮助用户作出贴切的选择非常有用，但是当用户与仪表板交互时，这可能大大增加需要运行的查询的数量。应该适度使用该设置。

筛选卡替代方法

可不使用筛选卡而使用几种替代方法，这些方法可提供相似的分析结果，而不会导致额外的查询开销。例如，您可以基于用户所选内容创建参数和筛选器。

- 优点：
 - 参数在呈现之前无需查询数据源
 - 参数和计算字段组合可以实现逻辑，这种逻辑可以比简单字段筛选器实现的逻辑还要复杂
 - 参数可用于跨多个数据源筛选 - 对于 Tableau 10 之前的版本，筛选器仅在单个数据源中有效。在 Tableau 10 及更高版本中，筛选器可设置为跨相关数据源工作。
- 缺点：
 - 参数只是单值的 - 如果需要用户选择多值，就不能使用
 - 参数不是动态的 - 值列表在创建参数时就确定了，不会根据 DBMS 中的值而更新

另一种替代方法是在视图之间使用筛选动作 -

- 优点：
 - 操作支持选择多值，即用鼠标可视化地套选，或者按住 Ctrl/Shift 单击选择多项
 - 操作显示的动态值列表是在运行时求出的
 - 操作可用于跨多个数据源筛选 - 对于 Tableau 10 之前的版本，筛选器仅在单个数据源中有效。在 Tableau 10 及更高版本中，筛选器可设置为跨相关数据源工作。

- 缺点：
 - 筛选操作的设置比筛选卡更复杂
 - 操作不呈现与参数或筛选卡相同的用户界面 – 通常它们需要更多屏幕空间来显示
 - 操作源工作表仍需要查询数据源；但是它得益于 Tableau 处理管道中的缓存

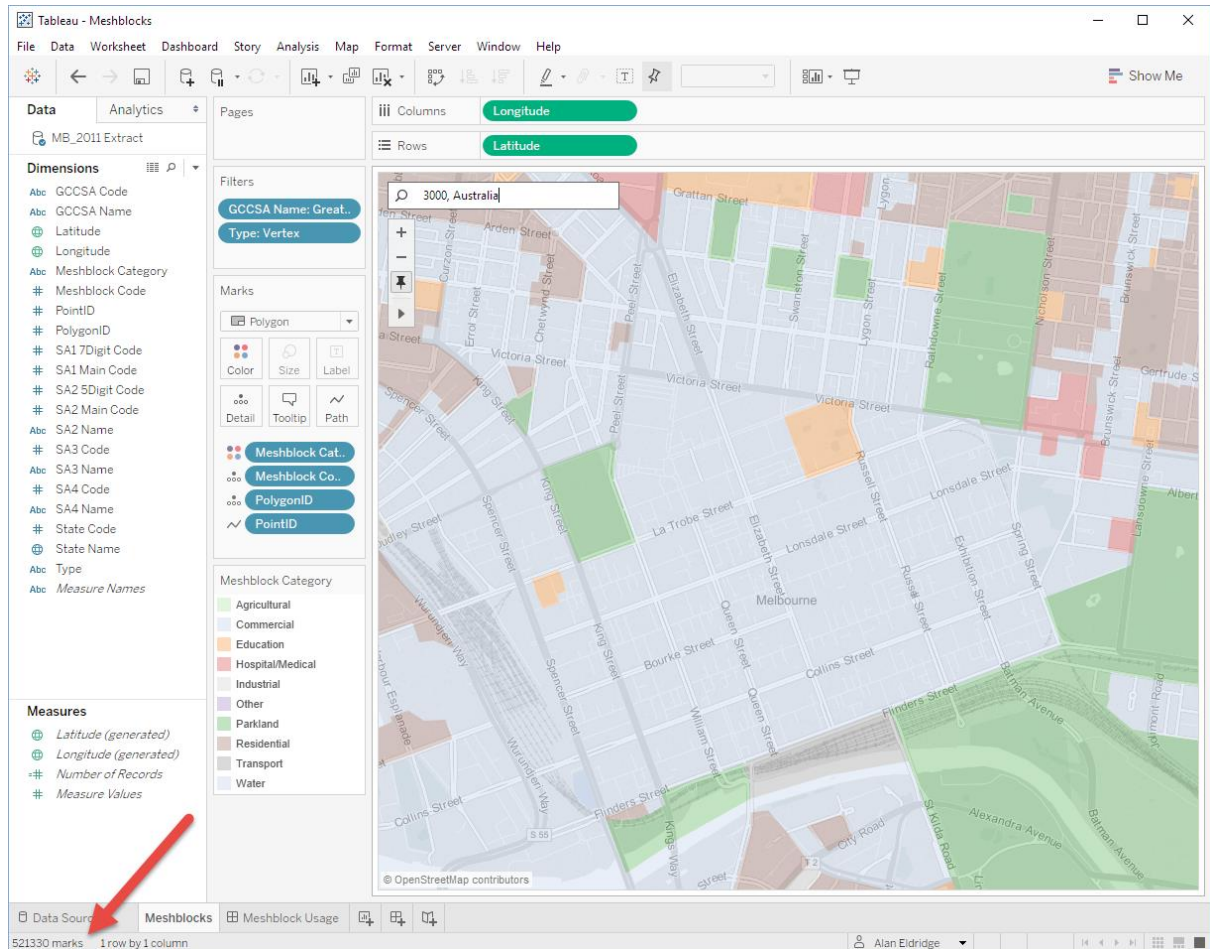
有关不严重依赖筛选卡的其他设计技术的进一步讨论，请参阅[本文前面的“优秀的仪表板设计”部分](#)。

用户筛选器

工作簿包含用户筛选器（要么通过“创建用户筛选器...”对话框，要么通过使用 ISMEMBEROF() 等任何内置用户函数的计算字段）时，不会跨用户会话共享模型缓存。这可能导致缓存重复使用率降低，转而增加 Tableau Server 的工作量。请谨慎使用这些筛选器类型。

缩放和筛选

放大包含大量标记的可视化时，不会筛选掉看不见的标记。我们只会更改数据视图。正在处理的标记总数不会发生更改，如下图中所示：



如果只需要小部分数据，可筛选掉多余数据，然后对该视图设置 Tableau 的自动缩放功能。

是不是计算问题？

很多情况下，源数据不会提供解答所有问题所需要的全部字段。计算字段将帮助您创建完成分

析所需的所有维度和度量。

在计算字段中，可以定义硬编码的常量（比如税率），执行减法或乘法等非常简单的数学运算（如收入减去成本），使用更为复杂的数学公式，执行逻辑检验（如 IF/THEN, CASE）、执行类型转换等更多任务。

一旦定义计算字段，只要工作表仍使用同一个数据源，那么该字段就可以在整个工作簿中使用。可以像使用源数据中的维度和度量一样，在工作簿中使用计算字段。

Tableau 中有四种不同的计算类型：

- 行级别计算、
- 聚合计算、
- 表计算和
- 详细级别表达式。

有关如何选择最佳方法的指南，请遵循下面的流程图：

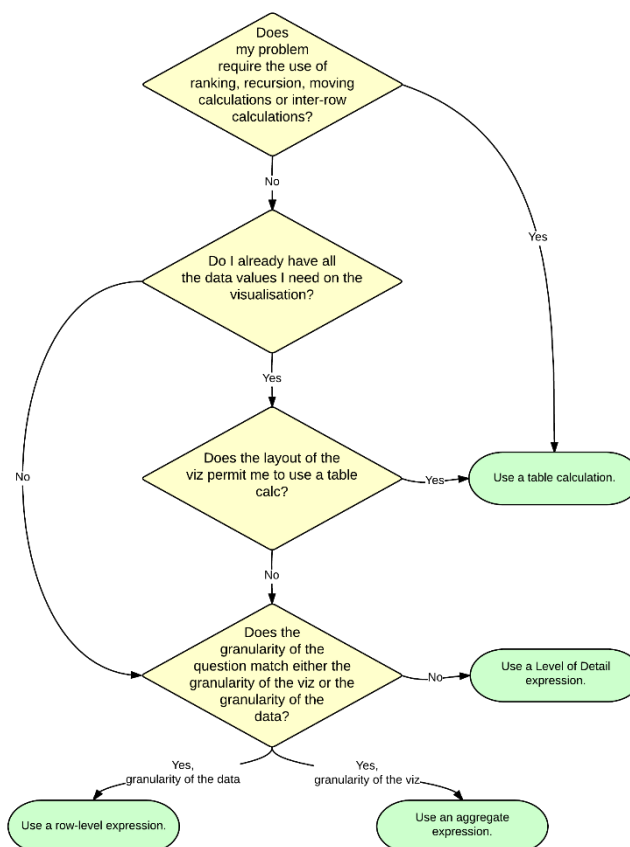


Tableau 计算参考库是很好的参考资源，可以从上面了解如何执行复杂计算，同时它也是一个论坛，用户在上面分享常见问题的解决方法。它的地址是：

<https://community.tableau.com/community/viz-talk/tableau-community-library/calculation-reference-library>

计算类型

行级别计算和聚合计算

行级别计算和聚合计算表示为查询的一部分发送给数据源，因此二者是由数据库计算的。例如，显示每年订货量总销售额的可视化会将以下查询发送给数据源：

```
SELECT DATEPART(year,[OrdersFact].[Order Date]) AS [yr:Order Date:ok],
       SUM([OrdersFact].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[OrdersFact] [OrdersFact]
GROUP BY DATEPART(year,[OrdersFact].[Order Date])
```

YEAR（年份）计算是行级别计算，**SUM(SALES)**（SUM（销售额））是聚合计算。

一般来说，行级别计算和聚合计算伸缩性很好，可采用很多数据库调优技术来提高其性能。

请注意，在 Tableau 中，实际的 DB 查询可能不是可视化中所用基本计算的直接转换。例如，可视化包含定义为 $SUM([Profit])/SUM([Sales])$ 的计算字段“Profit Ratio（利润率）”时，则会运行以下查询：

```
SELECT DATEPART(year,[OrdersFact].[Order Date]) AS [yr:Order Date:ok],
       SUM([OrdersFact].[Profit]) AS
[TEMP(Calculation_0260604221950559)(1796823176)(0)],
       SUM([OrdersFact].[Sales]) AS
[TEMP(Calculation_0260604221950559)(3018240649)(0)]
FROM [dbo].[OrdersFact] [OrdersFact]
GROUP BY DATEPART(year,[OrdersFact].[Order Date])
```

实际上，Tableau 会提取计算元素并在客户端层执行除法函数。这可确保缓存“Year（年份）”级别的 $SUM([Profit])$ 和 $SUM([Sales])$ ，并在工作簿内其他任意位置使用，而无需转到数据源。

最后，查询融合（将多个逻辑查询合并成单个实际查询）可以修改针对数据源所运行的查询。如果其他工作表共享同一精度，这可能导致多个工作表中的多个度量合并成单个查询。

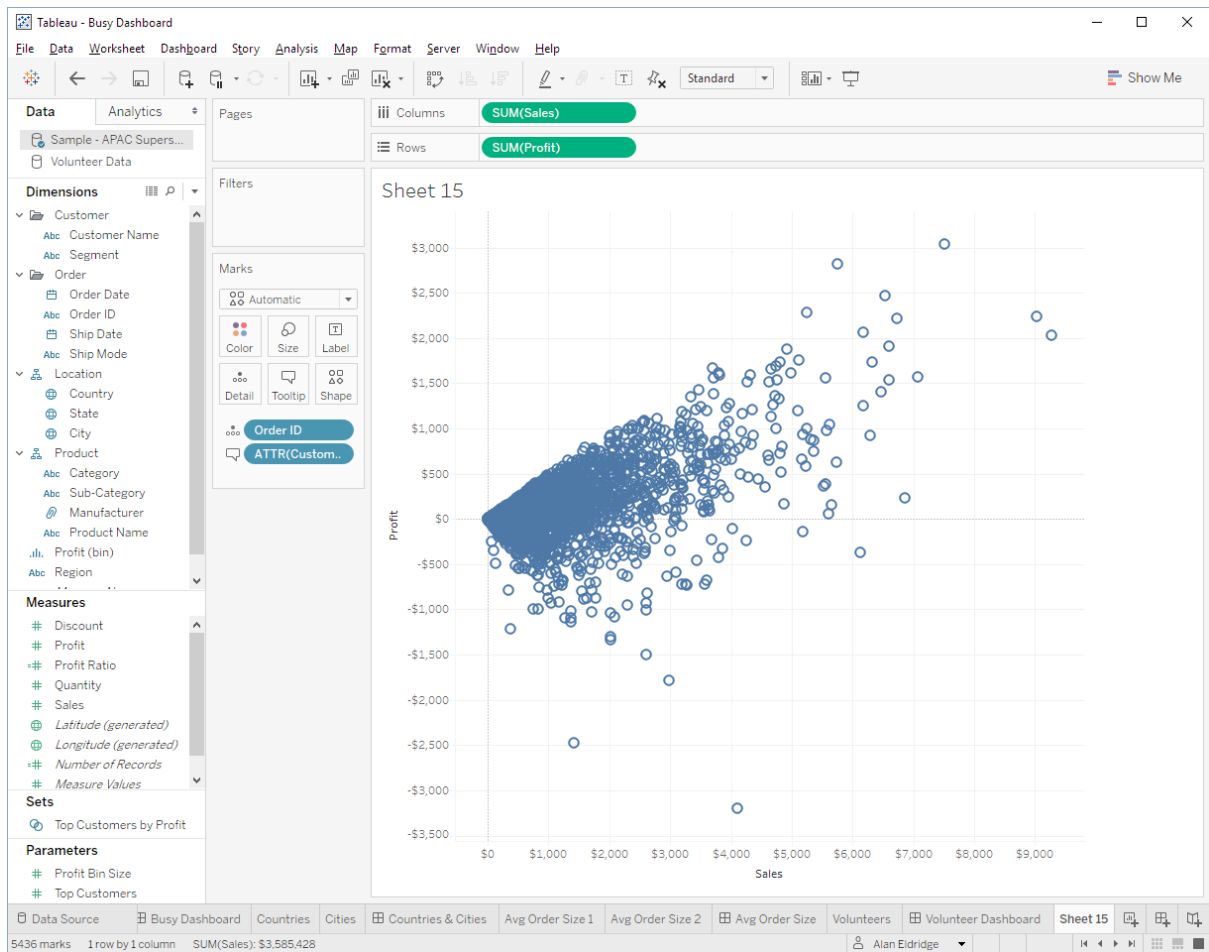
使用 ATTR()

ATTR() 是一个功能强大的函数，常用作分类维度的聚合。简言之，如果是唯一的，则返回值，否则返回 *。从技术上讲，该函数执行以下逻辑测试：

```
IF MIN([dimension]) = MAX([dimension])
  THEN [dimension]
  ELSE "*"
END
```

这需要在基础数据源中执行两种聚合（即 MIN() 和 MAX()），并将二者所得结果传递回结果集。如果不考虑是否拥有多个维度值，更有效的解决方案是仅使用一种聚合 - MIN() 或 MAX()，二选一。具体使用何种聚合无关紧要，重要的是选择并坚持使用一种聚合，以便最大限度提高缓存命中率。

为了表明 ATTR() 如何影响数据源，我们创建了以下可视化：



默认情况下，在工具提示功能区上设置维度会将 ATTR() 用作默认聚合。这导致以下查询：

```
SELECT [Superstore APAC].[Order ID] AS [Order ID],
       MAX([Superstore APAC].[Customer Name]) AS [TEMP(attr:Customer
Name:nk) (2542222306) (0)],
       MIN([Superstore APAC].[Customer Name]) AS [TEMP(attr:Customer
Name:nk) (3251312272) (0)],
       SUM([Superstore APAC].[Profit]) AS [sum:Profit:ok],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
GROUP BY [Superstore APAC].[Order ID]
```

如果每个订单 ID 下所含的客户数不多，可以将聚合更改为 MIN() 并简化查询：

```
SELECT [Superstore APAC].[Order ID] AS [Order ID],
       MIN([Superstore APAC].[Customer Name]) AS [min:Customer Name:nk],
       SUM([Superstore APAC].[Profit]) AS [sum:Profit:ok],
       SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
GROUP BY [Superstore APAC].[Order ID]
```

下表显示了这两个选项的性能差异（源自日志文件的查询结果需要数秒方可完成）：

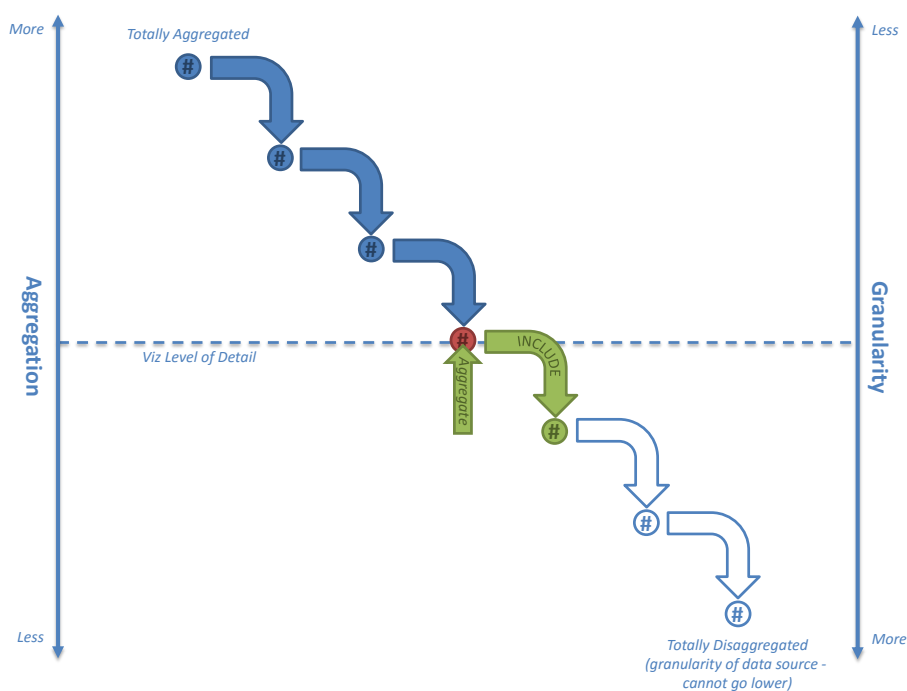
Test #	ATTR	MIN
1	2.824	1.857
2	2.62	2.09
3	2.737	2.878
4	2.804	1.977
5	2.994	1.882
Average	2.7958	2.1368
% improvement		24%

有关 ATTR() 的详细信息，请阅读下方 InterWorks 中的博客文章：

<https://www.interworks.com/blog/tcostello/2014/05/15/attr-tableaus-attribute-function-explained>

详细级别计算

通过详细级别 (LOD) 表达式，您可以独立于可视化详细级别，定义应以何种详细级别来执行计算。



在某些情况下，使用详细级别表达式可以替换在 Tableau 之前版本中以更繁杂方式创建的计算：

- 使用表计算，可能要设法找出分区中的第一个或最后一个时段。例如要在每月的第一天计算组织中的总人数。
- 使用表计算、计算字段和参考线，可能要设法将聚合字段归入数据桶。例如要找出客户确切数量的平均值。
- 使用数据混合，可能要设法相对于数据中的最晚日期进行相对日期筛选。例如要在数据每周刷新的情况下，根据最晚日期计算本年迄今总额。

与表计算不同，详细级别表达式作为针对基础数据源查询的一部分而生成。它们是以嵌套选择的形式表现，因此依赖于 DBMS 性能：

```
SELECT T1.[State],SUM(T2.[Sales per County, State])
FROM [DB Table] T1 INNER JOIN
    (SELECT [State], [County], AVG([Sales]) AS [Sales per County, State]
    FROM [DB Table] GROUP BY [State],[County]) T2
ON T1.[State] = T2.[State]
GROUP BY T1.[State]
```

这意味着，可能存在使用其他某种方法解决问题却更有效情况 - 即，表计算或混合的性能可能优于详细级别表达式，反之亦然。如果怀疑性能因详细级别表达式而变得缓慢，可以尝试将其替换为表计算或数据混合（如果可能），然后查看性能是否好转。此外，联合选择可能严重影响详细级别表达式。因此，如果发现使用详细级别表达式时查询运行缓慢，建议您返回重新阅读该部分。

若要深入了解详细级别表达式机制，请阅读“了解详细级别表达式”白皮书：

<http://www.tableau.com/zh-cn/learn/whitepapers/understanding-lod-expressions>

还可阅读 Bethany Lyons 编写的“15 大详细级别表达式”博客文章，其中提供了许多常见问题类型的示例：

<http://www.tableau.com/zh-cn/about/blog/LOD-expressions>

最后，社区中还有很多与本主题相关的博客文章，这些文章都非常有用。*data + science* 中提供了不错的策展列表：

<http://www.dataplusscience.com/TableauReferenceGuide/index.html>

表计算

与行级别计算和聚合计算不同，表计算不由数据库执行，而是由 Tableau 计算出查询结果集。虽然这对 Tableau 来说意味着工作量更大，但是计算的记录集通常要比原始数据源中的记录集小得多。

如果表计算性能成问题（可能因为返回给 Tableau 的记录集非常大），请考虑将某些方面的计算推回给数据源层。执行此操作的一种方法是，通过详细级别表达式。另一种方法是利用聚合的数据提取。假设您想查找多个店铺每日总销售额的周平均值。可以使用以下函数的表计算来求得：

```
WINDOW_AVG(SUM([Sales]))
```

但是，如果天数/店铺数很大，此计算可能变得很慢。为了将 SUM([Sales]) (SUM([销售额])) 推送回数据层，可创建一个将“Date (日期)”维度提升到日级别的聚合提取。然后只要执行 AVG([Sales]) (AVG([销售额])) 即可完成计算，因为提取已经具体化了每日总计。

在某些用例中，我们可能知道聚合因素的值不会随分区而改变。发生这种情况时，可将 MIN() 或 MAX() 用作聚合函数，而不是 AVG() 或 ATTR()，因为前两者的计算速度更快。选取一项后，请坚持使用该函数，以便提高缓存命中率！

外部计算

Tableau 提供可将复杂逻辑传递给外部引擎进行计算的机制。这些机制包括调用 R（通过 Rserve 连接）或 Python（通过 Dato 服务器连接）。

在 Tableau 中，这些操作类似于表计算，因此可以跨分区对其进行计算。这意味着，单个可视化可能调用多次上述对象，从而产生性能开销。此外，不会以任何方式优化或合并这些对象，请考虑您是否可以在单次函数调用中（例如，在串联字符串中）获取多个返回值，而不必调用多个函数。最后，数据与外部计算引擎之间的转移所用时间也可能成为一项开销。

分析

分析窗格提供对大量高级分析（如下）的快速访问权限：

- 合计
- 参考线
- 趋势线
- 预测
- 群集（Tableau 10 中的新增项）

这些分析通常不需要运行其他查询 - 其表现更像是对查询结果缓存中的数据执行表计算。与表计算相同，如果查询结果中包含超大数据集，计算进程可能耗费一些时间，但一般情况下，这不是造成工作簿性能整体不佳的主要原因。

影响合计和参考线性性能的另一因素在于是否可以累加度量聚合。如果是可累加聚合（例如 SUM、MIN、MAX），则可在 Tableau 本地计算合计或参考线。如果是不可累加聚合（例如，COUNTD、TOTAL、MEDIAN、PERCENTILE），则需返回到数据源才能计算合计/参考线值。

计算与本机功能

有时候，如果使用 Tableau 的本机功能就能轻松实现某些功能，那么用户可以创建计算字段来执行这些功能。例如：

- 将维度成员组合在一起 - 请考虑使用 [组](#) 或 [集](#)；
- 将度量值组合成为范围区间 - 请考虑使用 [数据桶](#)；
- 将日期截断为粗精度，例如月或周 - 请考虑使用 [自定义日期字段](#)；
- 创建一组合并了两种不同维度的值 - 请考虑使用 [合并字段](#)；
- 采用特定格式显示日期，或者将数值转换为 KPI 指示符 - 请考虑使用内置的格式设置功能；
- 更改维度成员的显示值 - 请考虑使用 [别名](#)。

这并非总是可行（例如，您可能需要宽度可变的数据桶，而使用基本数据桶无法实现），但是应该尽可能考虑使用本机功能。这样做通常比手动计算更高效，随着我们的开发人员不断提高 Tableau 的性能，他们的努力一定会让您获益匪浅。

数据类型的影响

在创建计算字段时，应了解所用的数据类型对计算速度有明显的影响，这一点很重要。作为一般准则

- 相比字符串和日期，整数和布尔值要快得多

字符串和日期计算非常慢 – 通常每次计算需要执行 10-100 条基本指令。相比之下，数值和布尔值计算非常高效。

上面这些话不仅适用于 Tableau 计算引擎，也适用于大多数数据库。由于行级别计算和聚合计算都下推给数据库，因此，如果这些计算表示为数值逻辑，而不是字符串逻辑，那么它们执行起来会快得多。

性能技巧

为了确保计算尽可能高效，请考虑以下技巧：

对基本逻辑计算使用布尔

如果计算生成二进制结果（如是/否、合格/失败、高于/低于），请确保返回布尔结果，而不是字符串。例如：

```
IF [Date] = TODAY() THEN "Today"
  ELSE "Not Today"
END
```

因为使用了字符串，所以这会很慢。更快的方法是返回布尔值：

```
[Date] = TODAY()
```

然后使用别名将 TRUE 和 FALSE 结果重命名为 “Today (今天)” 和 “Not Today (不是今天)”。

字符串搜索

假设您想显示产品名包含某个查找字符串的所有记录。您可以使用参数来获得来自用户的查找字符串，然后创建下面的计算字段：

```
IF FIND([Product Name],[Product Lookup]) > 0
  THEN [Product Name]
  ELSE NULL
END
```

此计算比较慢，因为这是测试一个值是否包含另一个值的低效方法。更好的方法是使用具体的 CONTAINS 函数，因为在传给数据库时，这将转换为最优的 SQL：

```
CONTAINS([Product Name],[Product Lookup])
```

但是在这个例子中，最佳的解决方案完全不是使用计算字段，而是使用通配符匹配筛选卡。

条件计算的参数

Tableau 中的一个常用技巧是为最终用户提供参数，以使它们可以选择值来决定如何执行计算。例如，假设您想让最终用户在一系列可能的值所组成的列表中，选择一个值来控制视图中所显示的日期聚合级别。很多人会创建字符串参数：

```
Value
Year
Quarter
Month
Week
Day
```

然后在计算中像这样使用该参数：

```
CASE [Parameters].[Date Part Picker]
```

```

WHEN "Year" THEN DATEPART('year',[Order Date])
WHEN "Quarter" THEN DATEPART('quarter',[Order Date])
..
END

```

更好的方式是，使用内置的 DATEPART() 函数并创建如下计算：

```
DATEPART(LOWER([Parameters].[Date Part Picker]), [Order Date])
```

在 Tableau 的较早版本中，后一种方法更快。但是，这两种解决方案现不再有性能差异，因为我们根据参数值折叠了 CASE 语句的条件逻辑，仅向数据源传递适当 DATEPART()。

切记，还需要考虑解决方案的未来可维护性，在这方面，后一种计算可能更佳。

日期转换

用户的日期数据通常不是以本机日期格式存储的 – 例如，这些数据可能是字符串或数字时间戳。DateParse() 函数可简化这些转换操作。只需传入格式字符串即可：

```
DATEPARSE("yyyymmdd", [YYYYMMDD])
```

请注意，只有部分数据源支持 DATEPARSE()：

- 非旧版 Excel 和文本文件连接
- MySQL
- Oracle
- PostgreSQL
- Tableau 数据提取

如果数据源不支持 DATEPARSE()，则将此字符串转换为正确 Tableau 日期的另一种技巧是，将字段解析为日期字符串（如 "2012-01-01" – 请注意首选 ISO 字符串，因为它们支持国际化），然后将其传入 DATE() 函数。

如果原始数据是数字字段，则转换为字符串，再转换为日期就非常低效。更好的方法是将数据保留为数字，然后使用 DATEADD() 和日期文字值进行计算。

例如，较慢的计算可能为（转换 ISO 格式的数字字段）：

```

DATE(LEFT(STR([YYYYMMDD]),4)
+ "-" + MID(STR([YYYYMMDD]),4,2)
+ "-" + RIGHT(STR([YYYYMMDD]),2))

```

更高效的计算方法是：

```

DATEADD('day', INT([yyyymmdd])% 100 - 1,
DATEADD('month', INT([yyyymmdd]) % 10000 / 100 - 1,
DATEADD('year', INT([yyyymmdd]) / 10000 - 1900,
#1900-01-01#)))

```

比上者更好的是使用 MAKEDATE() 函数（若数据源支持）：

```
MAKEDATE(2004, 4, 15)
```

请注意，这可显著提升大型数据集的性能。如果样本有 10 亿条记录，首次计算会耗时 4 个多小时才能完成，第二次却只需约 1 分钟。

逻辑语句

首先测试频率最高的结果

Tableau 评估逻辑测试时，一旦找到匹配项就会停止查找可能的结果。也就是说，要首先测试最可能的结果，以便在大多数评估中首次测试后即停止。

请看以下示例。此逻辑：

```
IF <unlikely_test>
  THEN <unlikely_outcome>
  ELSEIF <likely_test>
    THEN <likely_outcome>
  END
```

评估速度比以下逻辑慢：

```
IF <likely_test>
  THEN <likely_outcome>
  ELSEIF <unlikely_test>
    THEN <unlikely_outcome>
  END
```

ELSEIF > ELSE IF

处理复杂逻辑语句时请记住 ELSEIF > ELSE IF。这是因为嵌套的 IF 计算结果查询中嵌套的 CASE 语句，而不是作为第一个 IF 的一部分进行计算。因此该计算字段：

```
IF [Region] = "East" AND [Segment] = "Consumer"
  THEN "East-Consumer"
  ELSE IF [Region] = "East" and [Segment] <> "Consumer"
    THEN "East-All Others"
  END
END
```

生成以下具有两个嵌套 CASE 语句和 4 个条件测试的 SQL 代码：

```
(CASE
  WHEN (([Global Superstore].[Region] = 'East')
  AND ([Global Superstore].[Segment] = 'Consumer'))
    THEN 'East-Consumer'
  ELSE (CASE
    WHEN (([Global Superstore].[Region] = 'East')
    AND ([Global Superstore].[Segment] <> 'Consumer'))
      THEN 'East-All Others'
    ELSE CAST(NULL AS NVARCHAR)
    END)
  END)
```

由于下例使用 ELSEIF 而不是嵌套的 IF 重写，因此这会加快运行速度，还会提高条件测试的使用效率：

```
IF [Region] = "East" AND [Segment] = "Consumer"
  THEN "East-Consumer"
  ELSEIF [Region] = "East"
    THEN "East-All Others"
  END
```


由此导致 SQL 代码中仅有 1 个 CASE 语句：

```
(CASE
  WHEN (([Global Superstore].[Region] = 'East')
  AND ([Global Superstore].[Segment] = 'Consumer'))
    THEN 'East-Consumer'
  WHEN ([Global Superstore].[Region] = 'East')
    THEN 'East-All Others'
  ELSE CAST(NULL AS NVARCHAR)
END)
```

但这样仍会更快。尽管使用嵌套的 IF 语句，这却是最高效的条件测试使用方式：

```
IF [Region] = "East" THEN
  IF [Segment] = "Consumer"
    THEN "East-Consumer"
    ELSE "East-All Others"
  END
END
```

生成的 SQL 代码：

```
(CASE
  WHEN ([Global Superstore].[Region] = 'East')
  THEN (CASE
    WHEN ([Global Superstore].[Segment] = 'Consumer')
      THEN 'East-Consumer'
      ELSE 'East-All Others'
    END)
  ELSE CAST(NULL AS NVARCHAR)
END)
```

避免冗余的逻辑检查

同样，应避免冗余的逻辑检查。下面的计算：

```
IF [Sales] < 10 THEN "Bad"
  ELSEIF [Sales] >= 10 AND [Sales] < 30 THEN "OK"
  ELSEIF [Sales] >= 30 THEN "Great"
END
```

生成以下 SQL：

```
(CASE
  WHEN ([Global Superstore].[Sales] < 10)
    THEN 'Bad'
  WHEN (([Global Superstore].[Sales] >= 10)
  AND ([Global Superstore].[Sales] < 30))
    THEN 'OK'
  WHEN ([Global Superstore].[Sales] >= 30)
    THEN 'Great'
  ELSE CAST(NULL AS NVARCHAR)
END)
```

以下编写方式更高效：

```
IF [Sales] < 10 THEN "Bad"
  ELSEIF [Sales] >= 30 THEN "Great"
  ELSE "OK"
END
```

产生以下 SQL :

```
(CASE
  WHEN ([Global Superstore].[Sales] < 10)
    THEN 'Bad'
  WHEN ([Global Superstore].[Sales] >= 30)
    THEN 'Great'
  ELSE 'OK'
END)
```

多个小操作

可执行很多小操作来影响性能。请考虑以下技巧：

- 处理日期逻辑可能很复杂。请勿使用多个日期函数编写复杂的测试 - 如 MONTH() 和 YEAR(), 考虑使用一些其他内置函数, 如 DATETRUNC()、DATEADD() 和 DATEDIFF()。这可大幅简化数据源生成的查询。
- 清晰的统计值几乎是所有数据源中最慢的聚合类型之一。请谨慎使用 COUNTD 聚合。
- 使用影响范围为较大的参数 (如在自定义 SQL 语句中) 会影响缓存性能。
- 筛选复杂计算可能导致基础数据源中的索引丢失。
- 仅在详细级别达到时间戳级别时才使用 NOW()。应使用 TODAY() 进行日期级别计算。
- 请记住, 所有基本计算都会传递到基础数据源 - 包括标签字符串等文本计算。如需创建标签 (例如为列标题) 且数据源很大, 请创建仅由 1 条记录托管的简单文本/Excel 文件数据源, 避免其增加较大数据源的开销。如果数据源使用存储的进程, 这一点尤为重要, 更多详细信息请参阅[此部分](#)。

这是我的查询吗？

Tableau 有一种很强大的功能，即能使用内存中数据（即提取的数据连接），还能使用本地数据（即实时数据连接）。实时连接是一种很强大的 Tableau 功能，因为用户可借此利用数据源中已存在的处理功能。但是，由于目前数据源平台决定了所获得的性能，因此用户有必要按最佳方式且尽可能高效地生成自己的数据源查询。

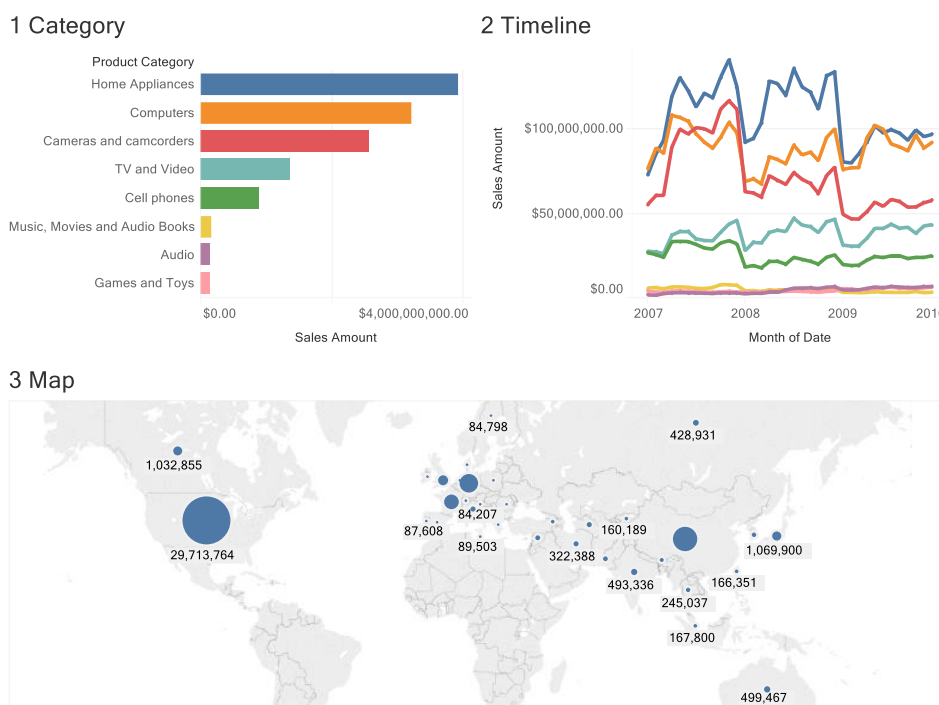
如讨论的一样，问题越复杂，显示的数据越多，仪表板中包含的元素越多 - 这些都意味着增加数据源的工作量。为了尽可能提高工作簿的效率，需要将查询数降至最低，确保尽可能高效地评估查询，将查询返回数据量降至最低并尽量重复使用请求间的数据。

自动优化

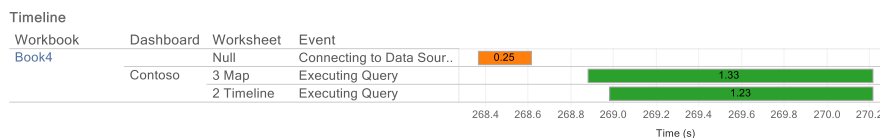
尽管可专门采取一些操作来提高查询性能，但 Tableau 将自动执行很多优化，确保工作簿高效执行。其中大多数操作不可由用户直接控制，且多数情况下无需考虑，但如果较为熟悉，可利用它们提高性能。

并行执行 - 一次运行多个查询

Tableau 利用源数据库的功能同时执行多个查询。请考虑以下仪表板：



在 Tableau 中打开此工作簿会显示以下查询：



可看到，串行运行查询需要 2.66 秒。但是并行运行只需考虑最长查询的长度（1.33 秒）。

由于某些平台处理同步查询的能力更好，因此源系统间的查询并行的级别有所不同。默认情况下，最多运行所有数据源的 16 个并行查询，文本、Excel 和统计文件除外（这些一次仅限 1 个查询）。其他数据源可能默认设有限制 - 请参阅以下链接了解详细信息：

<http://kb.tableau.com/articles/HowTo/Configuring-Parallel-Queries-in-Tableau-Desktop>

大多数情况下，无需更改这些设置（保留默认设置），但如果存在特定需求要控制并行的程度，可将其设置为：

- 针对 Tableau Server 并行查询数量的全局限制；
- 针对 SQL Server 等特殊数据源类型的限制；
- 针对特定服务器上的特殊数据源类型的限制；或者
- 针对连接到特定数据库时，特定服务器上的特殊数据源类型。

这些设置由名为 connection-configs.xml 的 xml 文件管理，该文件创建并保存在以下位置：

对于 Tableau Desktop，在 Windows (C:\Program Files\Tableau\Tableau 10.0) 和 Mac (右键单击“应用”，单击“显示包内容”，然后将文件放于此处) 中的应用文件夹；

对于 Tableau Server，则在 vizqlserver 文件夹等的

C:\ProgramData\Tableau\TableauServer\data\tabsvc\config\vizqlserver

配置目录中。必须将此配置文件复制到所有工作计算机的全部 vizqlserver 配置目录中。

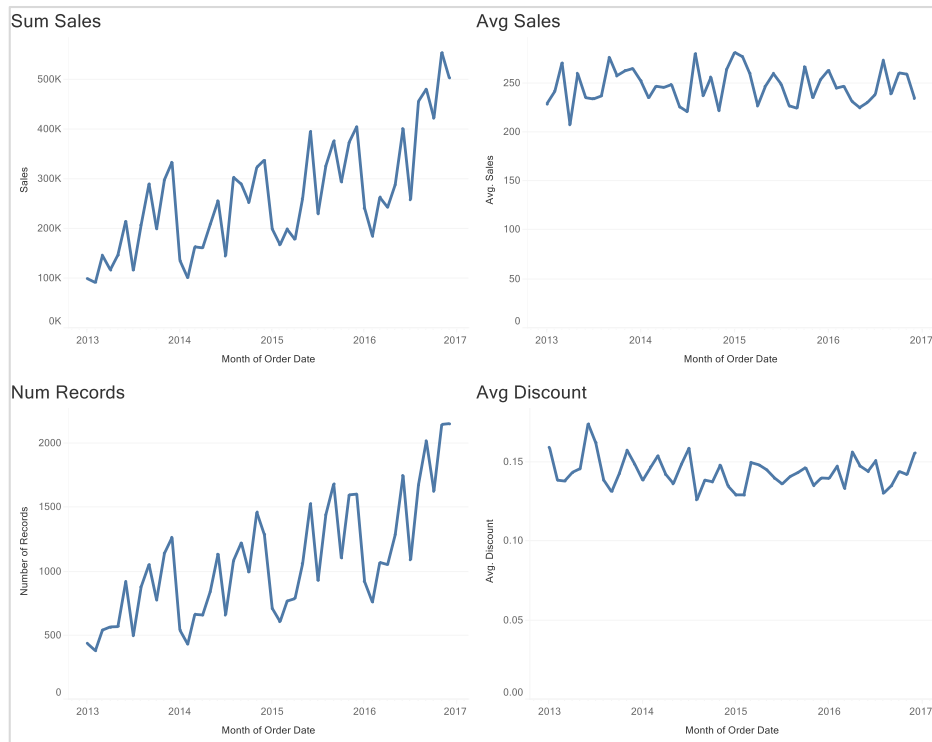
可通过此链接深入了解如何配置并行查询，包括 connection-configs.xml 文件的语法：

<http://kb.tableau.com/articles/knowledgebase/parallel-queries-tableau-server?lang=zh-cn>

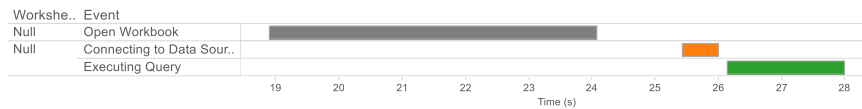
消除查询 - 运行较少查询

还可由上例所示，我们仅执行了 2 个查询，而不是 3 个。通过批处理查询，Tableau 可消除冗余的查询。Tableau 的查询优化程序会对查询进行排序，首先运行最复杂的查询，以便在结果缓存中处理后续查询。在示例中，由于时间范围包括“Product Category（产品类别）”且销售额的 SUM 聚合为全相加，因此“Category（类别）”图表的数据可解析自“Timeline（时间范围）”工作表的查询缓存，无需使用数据源。

查询优化程序还会查找相同详细级别的查询（即由相同维度集指定），并将其折叠为返回所有请求度量的单个查询。请考虑以下仪表板：



可看到，此仪表板包含 4 个工作表 - 每个工作表在不同时间显示不同的度量。由于它们使用连续月份显示数据，因此其详细级别相同。Tableau 会将其合并为单个查询，而不是运行 4 个单独的数据库请求：



查询如下：

```
SELECT AVG(cast([Global Superstore].[Discount] as float)) AS
[avg:Discount:ok],
  AVG(cast([Global Superstore].[Sales] as float)) AS [avg:Sales:ok],
  SUM(CAST(1 as BIGINT)) AS [sum:Number of Records:ok],
  SUM([Global Superstore].[Sales]) AS [sum:Sales:ok],
  DATEADD(month, DATEDIFF(month, CAST('0001-01-01 00:00:00' AS datetime2),
[Global Superstore].[Order Date]), CAST('0001-01-01 00:00:00' AS datetime2))
AS [tmn:Order Date:ok]
FROM [dbo].[Global Superstore] [Global Superstore]
GROUP BY DATEADD(month, DATEDIFF(month, CAST('0001-01-01 00:00:00' AS
datetime2), [Global Superstore].[Order Date]), CAST('0001-01-01 00:00:00' AS
datetime2))
```

可看出，该“查询融合”优化能显著提高整体性能。尽可能考虑将仪表板上的多个工作表设置为相同的详细级别。

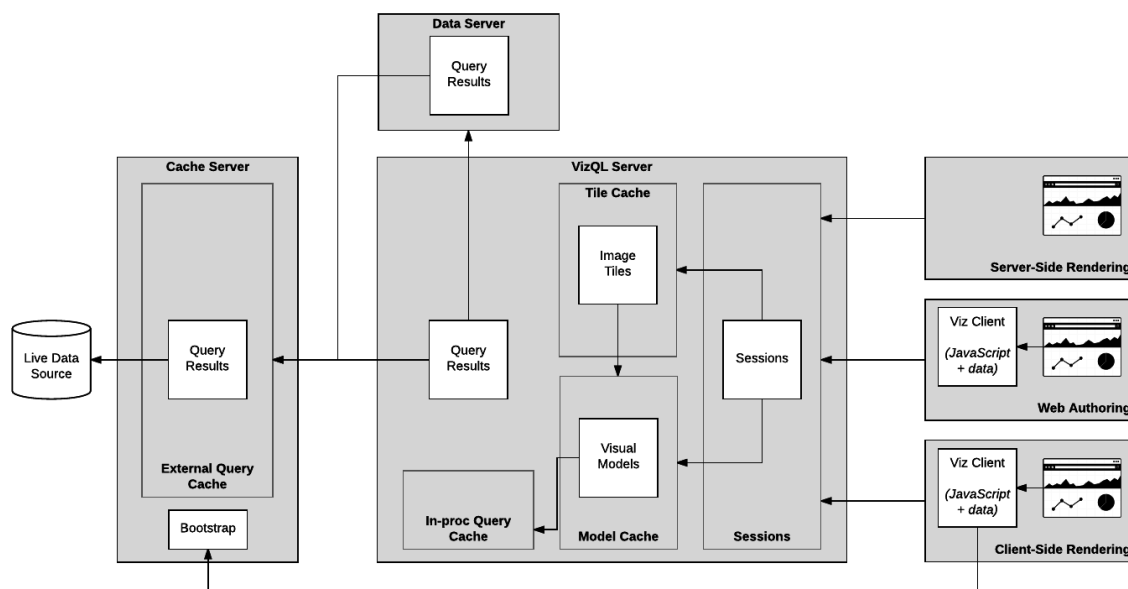
请注意，TDE 数据源的查询不会进行查询融合。

缓存 - 完全不运行查询

比运行较少查询更好的做法是什么？完全不运行查询！Tableau 的 Tableau Desktop 和 Tableau Server 中均具有大量缓存，可大幅减少需在基础数据源上运行的查询数量。

Tableau Server

Tableau Server 中的缓存有多个层：



缓存的第一层取决于会话是使用客户端呈现还是服务器端呈现。请参阅[上一部分](#)，深入了解这两种呈现模式以及 Tableau 如何确定应使用哪一种。

如果会话使用的是客户端呈现，浏览器首先会加载查看器客户端。这需要加载一套 JavaScript 库和数据，以便能呈现初始视图 - 该过程称作“引导”，可能需要几秒钟。由于可能有多名用户查看仪表盘，为减少后续视图请求的等待时间，会缓存引导响应。每个会话将首先查看引导缓存，查看是否创建有可用响应。如果找到响应，浏览器只需加载缓存中的响应，快速呈现初始视图。查看器客户端加载到客户端浏览器后，可完全使用本地数据（如突出显示和工具提示）实现部分交互，进而提供快速流畅的用户体验。

如果仪表板的当前显示方式是服务器端呈现，则服务器会将仪表盘元素显示为一系列图像文件（称作“磁贴”）。这些磁贴将传递到浏览器，并在此处聚合以显示可视化对象。如上所述，预计将有多名用户查看仪表盘，因此服务器会在磁盘上缓存这些图像。每个请求都会检查磁贴缓存，查看图像是否已呈现；若已呈现，其只需从缓存中加载文件即可。使用磁贴缓存会缩短响应时间并减少服务器上的工作负荷。一种增加磁贴缓存效率的简单方式是将仪表盘设置为[固定大小](#)。

整体而言，客户端呈现可实现更流畅、响应更敏捷的用户交互，并减少 Tableau Server 上的工作负荷。应尽量将工作簿设计为可进行客户端呈现。

缓存的下一层称作视觉模型。视觉模型说明了如何呈现单个工作表，以便查看仪表盘时可引用多个视觉模型 - 分别对应每个使用的工作表。该模型包括本地计算的结果和视觉布局。前者含有表格计算、参考线、预测和聚类分析等；后者包括为较小倍数和交叉表显示的行/列数，待绘制的轴刻度/网格线的数量和间隔，待显示的标记标签的数量和位置等。

视觉模型由 VizQL Server 创建并缓存在内存中，尽量按最佳方式跨用户会话共享结果。能否共享视觉模型的关键要素如下：

- 可视化对象显示区域的大小 - 只可在可视化对象大小相同的会话中共享模型。通过将仪表板设置为固定大小，可按相同方式帮助模型缓存和磁贴缓存 - 即增加重复使用率并降低服务器上的工作负荷。
- 所选内容/筛选器是否匹配 - 如果用户正在更改筛选器和参数、向上/向下钻取等，则模型仅可与具有匹配视图状态的会话共享。发布工作簿时请勿选中“显示选定内容”，因此这会降低不同会话的匹配度。
- 用于连接到数据源的凭据 - 如果系统提示用户提供凭据才能连接到数据源，则只能在凭据相同的用户会话间共享模型。请慎用此功能，因为会降低模型缓存的效率。
- 是否使用用户筛选器 - 如果工作簿包含用户筛选器或者包含带 USERNAME() 或 ISMEMBEROF() 函数的计算，则模型不可与任何其他用户会话共享。请慎用这些函数，因为会大幅降低模型缓存的效率。

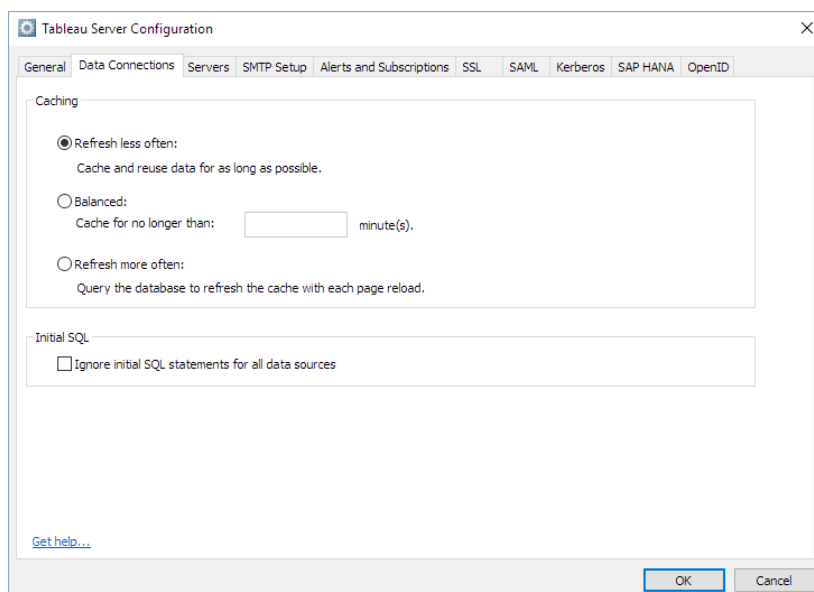
缓存的最后一层是查询缓存。它存储了已预先运行的缓存中的结果，可在以后的查询中使用。使用此缓存可避免在数据源中重复运行查询（只需加载缓存中的数据），进而显著提高效率。有时，它还可使用其他查询的结果辅助查询。我们在[前文](#)介绍查询消除和查询融合时探讨了该缓存的优势。

该缓存由两部分组成。一个称作“进程内缓存”，托管在 VizQL 进程内；另一个称作“外部缓存”，通过缓存服务器跨多个进程共享。如果将查询发送到先前运行同一查询的 VizQL 实例中，则通过进程内缓存处理请求。请注意，此缓存是每个 VizQL 进程的本地缓存，保留在内存中。

除了这些进程内缓存外，还存在外部缓存，其跨 VizQL 实例共享，还可跨访问基础数据源（如后台程序和数据服务器等）等所有进程共享。“缓存服务器”服务管理整个群集上的外部缓存。请注意，并非所有查询都会写入外部缓存。如果某查询运行很快，再次运行的速度比查看缓存的方式更快，因此存在最小的查询时间阈值。此外，如果查询的结果很大，则将其写入缓存服务器时可能较慢，因此会出现最大阈值。

在具有多个 VizQL 实例的部署中，外部查询可显著提升缓存效率。与不稳定的进程内缓存相比，外部缓存更持久（缓存服务器将数据写入磁盘），并在服务的实例化对象之间保留。

为最大程度提升缓存效率，可调整 Tableau Server 安装的设置，尽可能长久保留这些安装：



若有空闲的服务器容量，则增加缓存的大小。增加这些设置没有坏处，因此如果 RAM 充足，可大幅增加其数量，避免内容从缓存中溢出。可通过[日志文件](#)或 [TabMon](#) 监控缓存效率。

- **模型缓存** - 默认设置是每个 VizQL 实例缓存 60 个模型。可通过 `tabadmin` 命令调整此设置 - 若有空闲的 RAM，可增加此设置的数量，使其等于服务器上发布的视图数：
 - `tabadmin set vizqlserver.modelcachesize 200`
- **进程内缓存** - 默认设置是每个 VizQL 实例缓存 512MB 的查询结果。这听起来可能不多，但要记住，缓存的是来自聚合查询的查询结果。可通过 `tabadmin` 命令调整此设置：
 - `tabadmin set vizqlserver.querycachesize 1024`

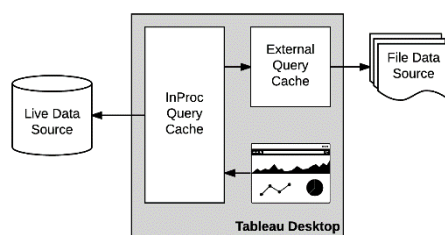
还可添加更多缓存服务器实例，增加外部查询的容量和吞吐量。最佳做法是为每个 VizQL Server 实例运行 1 个缓存服务器实例。

- **外部缓存** - 默认设置是为每个缓存服务器实例缓存 512MB 的查询结果。

最后一点，共享的缓存有一个有用的附带作用，即可通过运行工作簿的订阅，为已知首次查看较慢的工作簿预热缓存。请在交互式用户开始工作前的清晨执行此操作，这会确保工作簿已运行且查询结果已加载到缓存中。假设结果不会溢出缓存或未在干预时间内过期，随后在用户查看工作簿时将启动缓存，缩短初次查询的时间。

Tableau Desktop

相比于 Tableau Server，在 Tableau Desktop 中缓存更为简单，因此后者是单一用户应用程序，无需管理多个会话。Tableau Desktop 仅显示查询缓存：



与 Tableau Server 相似，其中有进程内缓存和外部缓存。进程内缓存不稳定且基于内存，用于到所有数据源的连接，而外部缓存仅用于基于文件的数据源（如数据提取、Excel、Access、文本文件和统计文件等）。与 Tableau Server 类似，外部缓存具有持久性。也就是说，缓存结果会在 Tableau Desktop 的会话之间保留，因此若使用文件数据源、重启 Tableau Desktop 和重新使用该源，仍可从缓存中受益。

外部缓存存储在 Windows 的 `%LOCALAPPDATA%\Tableau\Caching` 中和 Mac 的 `~/Library/Caches/com.tableau/` 中。默认情况下，该缓存总量限制为约 750 MB，且在用户强制刷新数据源（例如按 F5 和 \mathbb{R} ）时失效。

最后在 Tableau Desktop 中，文件另存为打包工作簿时将随附外部查询缓存数据。由此，Tableau Desktop 和 Tableau Reader 可快速呈现工作簿的初始视图，同时继续在后台解压较大的数据源文件。对最终用户而言，这可显著提高打开工作簿的响应速度。请注意，仅当数据小于等于源数据文件大小的 10%，且不包含使用相对日期筛选器的查询时，才能包含缓存数据。

不活跃连接

在 Tableau 10 之前，当用户打开带多个数据源的工作簿时，会首先连接到所有不带凭据的数据源（即无需用户输入用户名/密码的数据源）。意味着需要连接到用户当前查看工作表/仪表板/故事上未使用的数据源，而这可能要花时间。

在 Tableau 10 中，现仅在用户需要数据源来查看所选工作表/仪表板/故事时才进行连接。此更改将使用户更快加载带标签视图的工作簿（在 Tableau Desktop 上），进而更快开始探索数据。

联接

通常，如果在 Tableau 中处理多个表，首选方式是在数据连接窗口中定义联接。执行此操作时不会定义特定查询 - 仅是定义表格间的关系。将字段拖放到可视化内容中时，Tableau 会利用此信息生成必要的特定查询以仅提取所需数据。

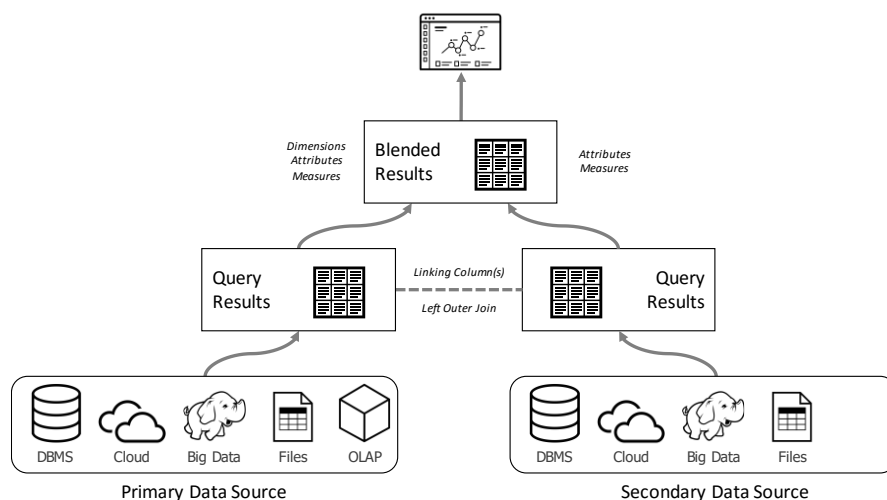
根据一般的性能经验法则，须尽量减少联接表的数量，使其仅包含特定工作表/可视化效果所需的表格。根据数据连接和工作表的复杂性，可能还需分离各工作表的数据连接，并为这些表创建特定的联接模式。

适当约束表格间的联接时，联接的性能更佳。通过此约束，Tableau Desktop 可简化查询，使其仅包含回答某些问题子集所需的表格（图例、筛选卡）。

混合

决定在 Tableau 中联接数据表还是进行混合时，需考虑数据的来源、数据连接的数量和数据中的记录数。

如果使用混合，则对混合性能的一项主要影响是联接两个数据集的混合字段的基数，而不是每个数据源中的记录数。混合操作将在链接字段级别上查询两个数据源中的数据，然后将两个查询的结果合并到 Tableau 的本地内存中。



如果有多个唯一值，则可能需要大量内存。建议混合数据时采用 64 位版本的 Tableau Desktop，避免内存用尽。但是，此类混合仍可能需要很长时间进行计算。

推荐的混合最佳做法是避免混合带大量唯一值的维度，例如订单 ID、客户 ID、精确日期/时间等。

主组和别名

如果您发现由于两个数据源分别包含“事实”记录和维度属性，因此有必要混合这两个源，则创建主组或别名可能会提升性能。

工作原理是在主数据源中创建组和/或别名，反映辅助数据源中列出的属性。主组针对一对多关系，主要别名针对一对一关系。随后可使用从主数据源中的相关对象，无需在查看期间进行混合。

对于下方示例，请考虑以下 3 个表：

ID	Value
A	89
B	94
C	74
D	88
E	58
F	89
G	95

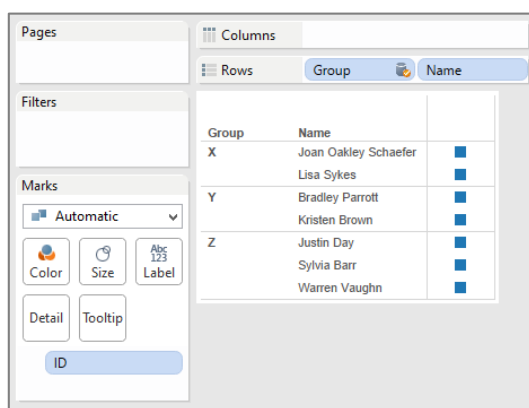
ID	Name
A	Lisa Sykes
B	Joan Oakley Schaefer
C	Bradley Parrott
D	Kristen Brown
E	Sylvia Barr
F	Warren Vaughn
G	Justin Day

ID	Group
A	X
B	X
C	Y
D	Y
E	Z
F	Z
G	Z

如果辅助数据源中的属性将一对多的关系映射回主数据源中的维度成员，则主组非常有用。假设要根据以上数据生成下面的结果：

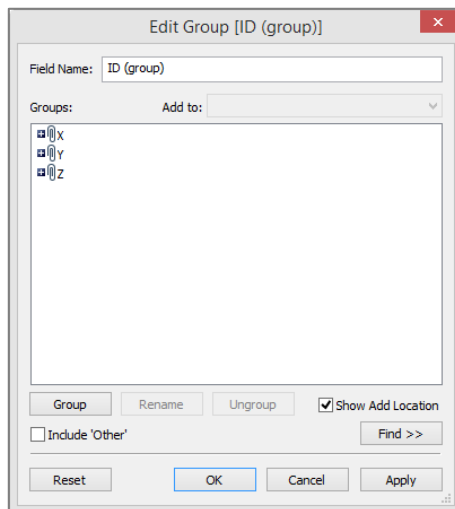
Group	Name
X	Lisa Sykes
	Joan Oakley Schaefer
Y	Bradley Parrott
	Kristen Brown
Z	Sylvia Barr
	Warren Vaughn
	Justin Day

可通过混合操作进行创建，但如前所述，若有大量 ID，这会导致性能很慢：

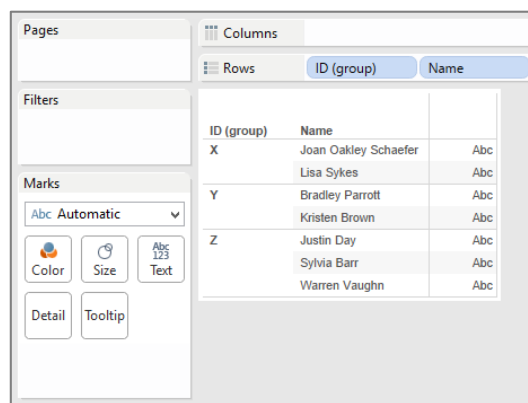


通过右键单击“Group（组）”字段再选择“Create Primary Group（创建主组）”，

Tableau 将在主数据源中创建一个组对象，该对象将链接字段（本例中的 "ID"）映射到所选辅助数据源维度（本例中的 "Group（组）"）。



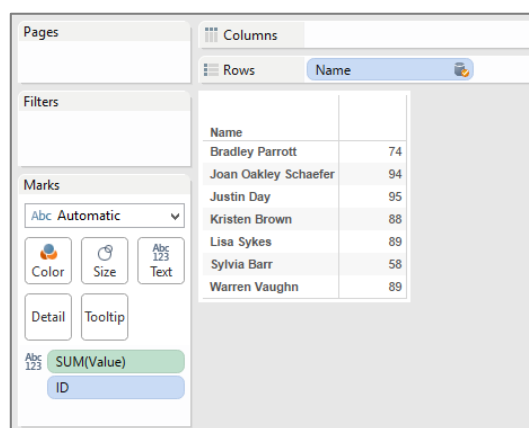
现无需混合即可重新创建此表：



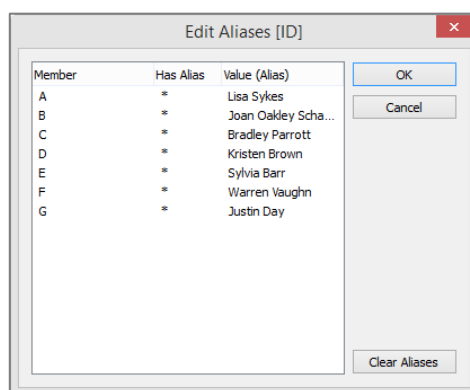
如果辅助数据源中的数据将一对一关系映射回主数据源中的维度成员，则主要别名非常有用。假设要根据以上数据生成下面的结果：

Name	Value
Lisa Sykes	89
Joan Oakley Schaefer	94
Bradley Parrott	74
Kristen Brown	88
Sylvia Barr	58
Warren Vaughn	89
Justin Day	95

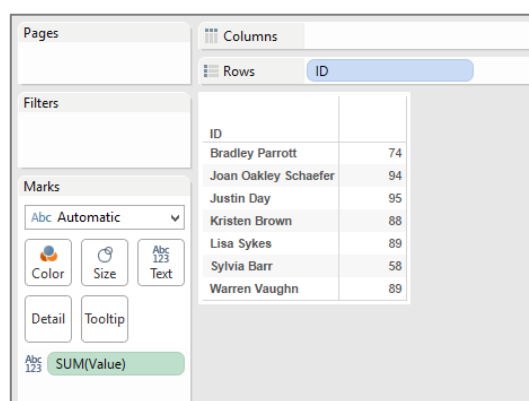
可通过混合两个数据源进行创建，但如前所述，若有大量 ID，这会导致性能很慢：



通过右键单击“Name（名称）”字段再选择“Edit Primary Aliases（编辑主要别名）”，可执行一次性映射，将 Name（名称）作为别名值映射到 ID 字段：



现无需混合即可创建所需的可视化内容，该操作速度更快：



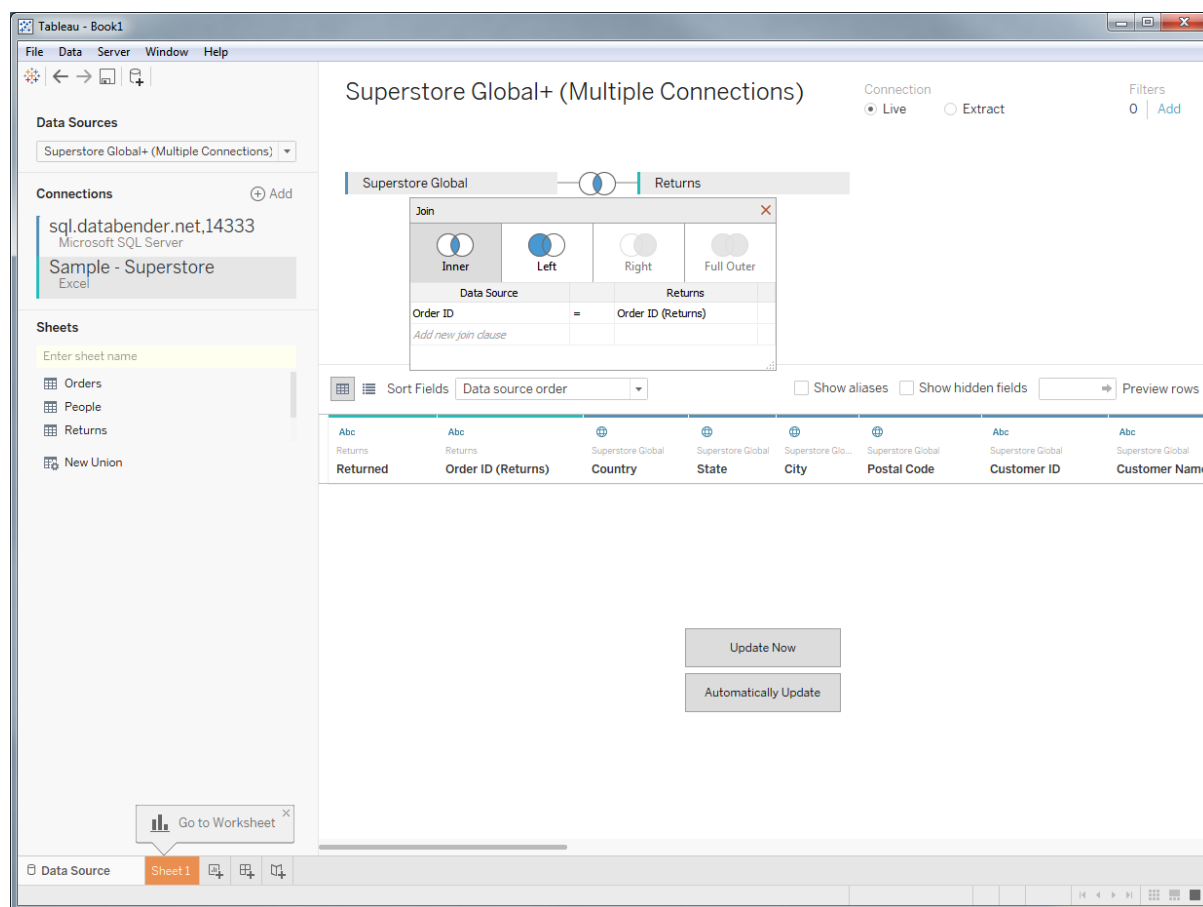
请注意两个主组和别名均是静态时，因此更改数据时需更新它们。因此，它们不适合频繁更新的数据，但如果需要快速映射，则该解决方案可能会消除耗时混合操作的需求。

详细信息可查看以下知识库文章：

http://onlinehelp.tableau.com/current/pro/desktop/zh-cn/help.htm#multipleconnections_create_primary_group.html

数据集成

数据集成是 Tableau 10 中引入的一项新功能，使数据源能合并多个数据连接（可能异类）中的数据。



数据集成和数据混合的主要区别在于，数据集成是行级联接，而数据混合针对的是每个数据源的聚合结果集。也就是说，数据集成对基础数据源的大小较为敏感，请注意以下 4 点：

- 要移动的数据越多，所需时间越长 - 数据按各数据连接的行级进行提取，因此数据大小很关键；
- 数据传输距离越长，所需时间越长 - 如果联接到高延迟连接上的数据源，则性能将受到影响；
- 数据流的速率越慢，所需时间越长 - 如果联接到带宽受限连接上的数据源，则性能将受到影响；
- 需匹配的对象越多，所需时间越长 - 与第一点类似，性能受到须联接的记录数量影响。

在 Tableau 10 中，仅可将数据集成用于提取的数据源，即每个数据连接的结果必须提取到 TDE 文件中。然后，可将提取的该数据源发布到 Tableau Server，供他人使用。

在 Tableau 10 中，不能对实时数据源和已发布的数据源执行数据集成。在未来的改进中，我们将努力增添这些功能。

自定义 SQL

有时，Tableau 的新用户会尝试在工作簿中使用旧版中的技巧，例如利用手写 SQL 语句创建数据源。很多情况下这会适得其反，因为 Tableau 可在用户定义表格间的联接关系时生成多个有

效的查询，并使查询引擎编写特定于所创建视图的 SQL。但某些时候，在数据连接窗口中指定联结无法让您随意灵活地定义数据中的关系。

使用手写 SQL 语句创建数据连接非常有用，但正如《蜘蛛侠》漫画的中著名警示所言，“能力越大，责任越大”。有时，自定义 SQL 实际上会导致性能降低。这是因为与定义联接相比，自定义 SQL 永远无法解构且始终自动执行。这就是说不执行联合选择，且最后的情况可能是数据库必须为单列处理整个查询，如下所示：

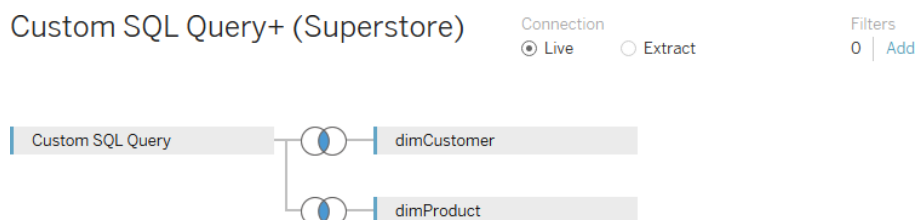
```
SELECT SUM([TableauSQL].[Sales])
FROM (
  SELECT [OrdersFact].[Order ID] AS [Order ID],
    [OrdersFact].[Date ID] AS [Date ID],
    [OrdersFact].[Customer ID] AS [Customer ID],
    [OrdersFact].[Place ID] AS [Place ID],
    [OrdersFact].[Product ID] AS [Product ID],
    [OrdersFact].[Delivery ID] AS [Delivery ID],
    [OrdersFact].[Discount] AS [Discount],
    [OrdersFact].[Cost] AS [Cost],
    [OrdersFact].[Sales] AS [Sales],
    [OrdersFact].[Qty] AS [Qty],
    [OrdersFact].[Profit] AS [Profit]
  FROM [dbo].[OrdersFact] [OrdersFact]
) [TableauSQL]
HAVING (COUNT_BIG(1) > 0)
```

请务必确保自定义 SQL 仅包含必需语句。例如，如果查询包含 GROUP BY 或 ORDER BY 子句，它们通常会挂起，因为 Tableau 将根据可视化对象的结构自行创建子句。尽可能从查询中删除这些语句。

好的推荐做法是结合数据提取使用自定义 SQL。这样，原子查询仅执行一次（将数据载入数据提取），然后即可对数据提取使用动态的优化查询，在 Tableau 中执行所有后续分析。当然，规则必有例外 - 使用自定义 SQL 创建数据提取时不遵从此规则，除非自定义 SQL 包含参数。

某些情况下，使用自定义 SQL 语句中的参数会提升实时连接的性能，因为基本查询更加动态（例如，将正确评估使用参数的筛选子句）。此类参数还可用于传入 TOP 或 SAMPLE 等性能限制符的值，进而限制数据库返回的数据量。但若使用数据提取，每次更改参数时都将清除并重新生成该提取，该过程速度缓慢。此外请注意，参数仅可用于传入文本值，因此不可用于动态更改 SELECT 或 FROM 子句。

最后，它还可能将表联接到自定义 SQL 中：



由此，用户可编写更具体的自定义 SQL 来引用总架构的子集。然后，（可能）可选择此项到其他表的联接（如普通表联接），创建更高效的查询。

自定义 SQL 的替代方法

有时更倾向将查询移入基础数据源，而不是在 Tableau 中直接使用自定义 SQL。很多情况下，这会提升性能，因为它可使数据源更高效地分析查询，或者意味着仅需运行一次复杂查询。它还使单个定义可跨多个工作簿和数据源进行共享，使其成为优秀的管理方式。

此操作有多种执行方式：

视图

几乎所有 DBMS 都支持视图的概念。视图指的是表示数据库查询结果的虚拟表。对于某些数据库系统，只需从自定义 SQL 语句中提取出查询并在数据库中实例化为视图结果，即可大幅优化性能。这是因为相比于查询包装在外部 SELECT 语句中，查询优化程序可生成更好的执行计划。更多详细信息请参阅以下社区讨论：

<https://community.tableau.com/thread/208019>

将自定义查询逻辑定义到视图中（而非 Tableau 工作簿中），也可使其跨工作簿和数据源重复使用。此外，很多 DBMS 支持具体化视图/快照概念，其中视图查询的结果经过预计算和缓存，可在查询时间更快响应。它们与摘要表类似（见下文），但由 DBMS 自动保留。

存储过程

存储过程与视图相似，但可包含更复杂的逻辑并可执行多步骤的数据准备。该过程可进行参数化，使其根据用户输入返回定向的数据集。

Sybase ASE、SQL Server 和 Teradata 均支持存储过程。为避免对单个可视化对象运行多次存储过程，Tableau 将执行存储过程并将结果集写入数据库的临时表中。随后，会针对临时表运行该可视化对象的实际查询。首次打开工作簿和每次更改已存储的进程参数时，都会执行存储过程和填充临时表。此过程可能会耗费一些时间，且交互操作可能很慢。

如果要将参数化存储过程的结果提取到数据提取中，则每次更改参数值时，都会删除并刷新该提取。

若要深入了解如何使用存储过程，可参阅 Tableau Online 帮助：

http://onlinehelp.tableau.com/current/pro/desktop/zh-cn/help.htm#connect_basic_stored_procedures.html

摘要表

如果您具有非常大的详细数据集，且该数据集通常在查询时进行汇总（例如，存储单个事务，但通常使用按日/区域/客户/产品等汇总的数据），则请考虑创建摘要表并对其使用 Tableau，进而缩短查询时间。

注意 – 借助 Tableau 数据提取，可通过创建聚合的数据提取实现类似的结果。更多详细信息，请参阅介绍提取的部分。

初始 SQL

自定义 SQL 的另一种替代方法（如果数据源支持）是在初始 SQL 块中使用自定义 SQL 语句。可利用该方法创建临时表，临时表随后将成为查询中选定的表。由于初始 SQL 仅在工作簿打开时执行一次，而不是每次更改自定义 SQL 的可视化对象时都要执行，这会在某些情况下明显提升性能。

请注意，在 Tableau Server 上，管理员可设置初始 SQL 限制来阻止其运行。如果计划发布工作簿与他人共享，可能需要检查环境是否支持此操作。

有关初始 SQL 的详细信息，可参阅在线文档：

[http://onlinehelp.tableau.com/current/pro/desktop/zh-cn/
help.htm#connect_basic_initialsql.html](http://onlinehelp.tableau.com/current/pro/desktop/zh-cn/help.htm#connect_basic_initialsql.html)

这是我的数据吗？

Tableau 的一项强大功能是可跨多个不同平台连接数据。广义上讲，这些平台可归结为以下某一数据源：

- 基于文件的数据源 - 例如 Excel 和 CSV；
- 关系数据库数据源 - 例如 Oracle、Teradata 和 SQL Server，以及 HP Vertica 和 IBM Netezza 等专门的分析应用程序。
- OLAP 数据源 - 例如 Microsoft Analysis Services 和 Oracle Essbase；
- “大数据”数据源 - 例如 Hadoop；
- 基于云的数据源 - 例如 Salesforce、Google 等。

每种类型的数据源都有各自的优缺点，并需要具体情况具体解决。

请注意，Windows 和 Mac OS X 均支持 Tableau Desktop，但 Mac 上支持的数据源集与 Windows 上的不同。Tableau 努力的方向是将平台间的差异降低到最低，但目前部分数据源仅在一个平台上受支持。

一般性建议

使用本地驱动程序

Tableau 10 支持从本地连接到 40 多个不同的数据源。也就是说，Tableau 实现了特定于这些数据源的技术、功能和优化。通过设计和测试这些连接，保证了它们是 Tableau 推出的最强大连接。

Tableau 还支持通用 ODBC，可用于访问本地连接器列表范围外的数据源。作为公开定义的标准，很多数据库供应商将 ODBC 驱动程序设为可用于其数据库，Tableau 也可使用这些程序连接到数据。在如何解释或实现 ODBC 标准的功能方面，每个数据库供应商可能有所不同。某些情况下，Tableau 会建议或要求您创建数据提取，以便继续使用特定驱动程序进行操作。还有一些 Tableau 无法连接到的 ODBC 驱动程序和数据库。

如果存在您正在查询的数据源的本地驱动程序，则应通过 ODBC 连接使用此程序，因为它通常提供更佳性能。还需注意，ODBC 连接仅在 Windows 上可用。

尽可能从较近位置测试数据

如前所述，一般原则是如果数据源执行查询的速度较慢，则 Tableau 的体验也会变慢。测试数据源原始性能的一种好方法是，（若可能）在数据源驻留的计算机上安装 Tableau Desktop 并运行一些查询。这样会消除网络带宽和性能延迟等因素，让您更好地了解数据源中查询的原始性能。此外，通过使用数据源的 *localhost* 名称（而非 DNS 名称），可帮助确定名称解析或代理服务器缓慢等环境因子是否加剧了性能不佳的情况。

数据源

文件

该类别中包含所有基于文件的数据格式。CSV、Excel 电子表格和 MS Access 等文本文件是最常见的格式，但来自统计平台 SPSS、SAS 和 R 的数据文件也属于此类别。业务用户经常使用此格式的数据，因为这是从“托管”数据集中提取数据的常见方式（无论是通过运行报表还是执行查询提取）。

一般来说，这是将基于文件的数据源导入 Tableau 快速数据引擎的最佳做法。这将大幅加快查询，还可大幅缩小存储数据值的文件。但如果文件较小或您需要实时连接到文件以反映不断变化的数据，则可进行实时连接。

卷影提取

连接到非旧版的 Excel/文本或数据文件时，Tableau 将在连接过程中以透明的方式创建提取文件。这称为卷影提取；与直接查询文件相比，这种数据处理方式速度快得多。

您可能会注意到，首次使用大文件时，需要几秒钟才能加载数据预览窗格。这是因为 Tableau 正在提取文件中的数据并将其写入卷影提取文件中。默认情况下，这些文件在 `C:\Users\\AppData\Local\Tableau\Caching\TemporaryExtracts` 中创建，且具有基于数据文件的路径名称和上次修改日期的哈希名称。Tableau 在此目录中保存 5 个最近使用的文件数据源的卷影提取，在创建新卷影时删除时间最早的文件。如果随后重新使用具有卷影提取的文件，Tableau 只需打开提取文件，数据预览几乎就可立即呈现。

尽管卷影提取含有与标准 Tableau 提取类似的基础数据和其他信息，但卷影提取文件按不同的格式（扩展名为 .ttde）进行保存，这意味着其使用方式与 Tableau 提取的不同。

面向 Excel 和文本文件的旧版连接器

在 Tableau 的早期版本中，利用 Microsoft 的 JET 数据引擎驱动程序连接到 Excel 和文本文件。但自 Tableau 8.2 起，默认使用本地驱动程序，它的性能更佳且可处理更大、更复杂的文件。但某些情况下您可能偏向使用旧版驱动程序，例如在要使用自定义 SQL 时。这些情况下，用户可选择降级为旧版 JET 驱动程序。请注意，阅读 MS Access 文件时仍使用 JET 驱动程序。

有关这两种驱动程序的详细差异列表，可参阅此处：

http://onlinehelp.tableau.com/current/pro/desktop/zh-cn/help.htm#upgrading_connection.html

请注意，不可在 Mac OS 上使用 JET 驱动程序，因此 Tableau Desktop for Mac 不可阅读 MS Access 文件，也不可按旧版方式连接 Excel 和文本文件。

关系数据库

关系数据源是 Tableau 用户最常见的数据源形式，且 Tableau 提供了多种平台的本机驱动程序。这些数据源可能基于行或列，可能是个人或企业数据源，并可通过本机驱动程序或通用 ODBC 进行访问。从技术上讲，此类别还包括 Map-Reduce 数据源，因为可通过 Hive 或 Impala 等 SQL 访问层进行访问，但此内容将在稍后的“大数据”部分中更详细探讨。

很多内部因素都会影响关系数据库系统 (RDBMS) 的查询速度。更改或调优这些系统时通常需要 DBA 的协助，但此操作会明显提升性能。

基于行和基于列

RDBMS 系统有两种主要类型 - 基于行和基于列。基于行的存储布局非常适合类似 OLTP 的工作负荷，该负荷需加载大量的交互事务。基于行的存储布局非常适合分析工作负荷（如数据仓库），该负荷通常需对较大数据集执行高度复杂的查询。

目前，许多高性能的分析解决方案均以基于列的 RDBMS 为基础。使用此类解决方案时，您会发现查询速度更快。Tableau 支持的基于列的数据库示例包括：Actian Vector、Amazon Redshift、HP Vertica、IBM Netezza、MonetDB、Pivotal Greenplum、SAP HANA 和 SAP Sybase IQ。

用作接口的 SQL

很多系统并非基于传统的 RDBMS 技术，但由于提供基于 SQL 的接口，因此仍可呈现为关系源。对 Tableau 而言，这包括几种 NoSQL 平台（如 MarkLogic、DataStax 和 MongoDB 等），以及 Kognitio、AtScale 和 JethroData 等查询加速平台。

索引

数据库的正确索引对于良好的查询性能至关重要：

- 确定参与表联接的列都有索引。
- 确定筛选器中使用的列都有索引。
- 注意，在某些数据库中使用离散日期筛选器时，可能会造成查询不使用日期和日期时间列的索引。我们将在筛选器部分进一步讨论这个问题，但是使用范围日期筛选器可以确保使用日期索引。例如，将筛选器表达为 `[DateDim] >= #2010-01-01# and [DateDim] <= #2010-12-31#`，而不是使用 `YEAR([DateDim])=2010`。
- 确保对数据启用统计，以允许查询优化器创建高质量的查询计划。

很多 DBMS 环境都有管理工具，这些工具可以调查查询，并推荐有帮助的索引。

NULL

维度列中具有 NULL 值会降低查询的效率。请设想一种可视化效果，我们想按国家/地区显示销量前 10 的产品的总销售额。Tableau 将首先生成子查询来查找销量前 10 的产品，然后按国家/地区将此数据联接回到查询，生成最终结果。

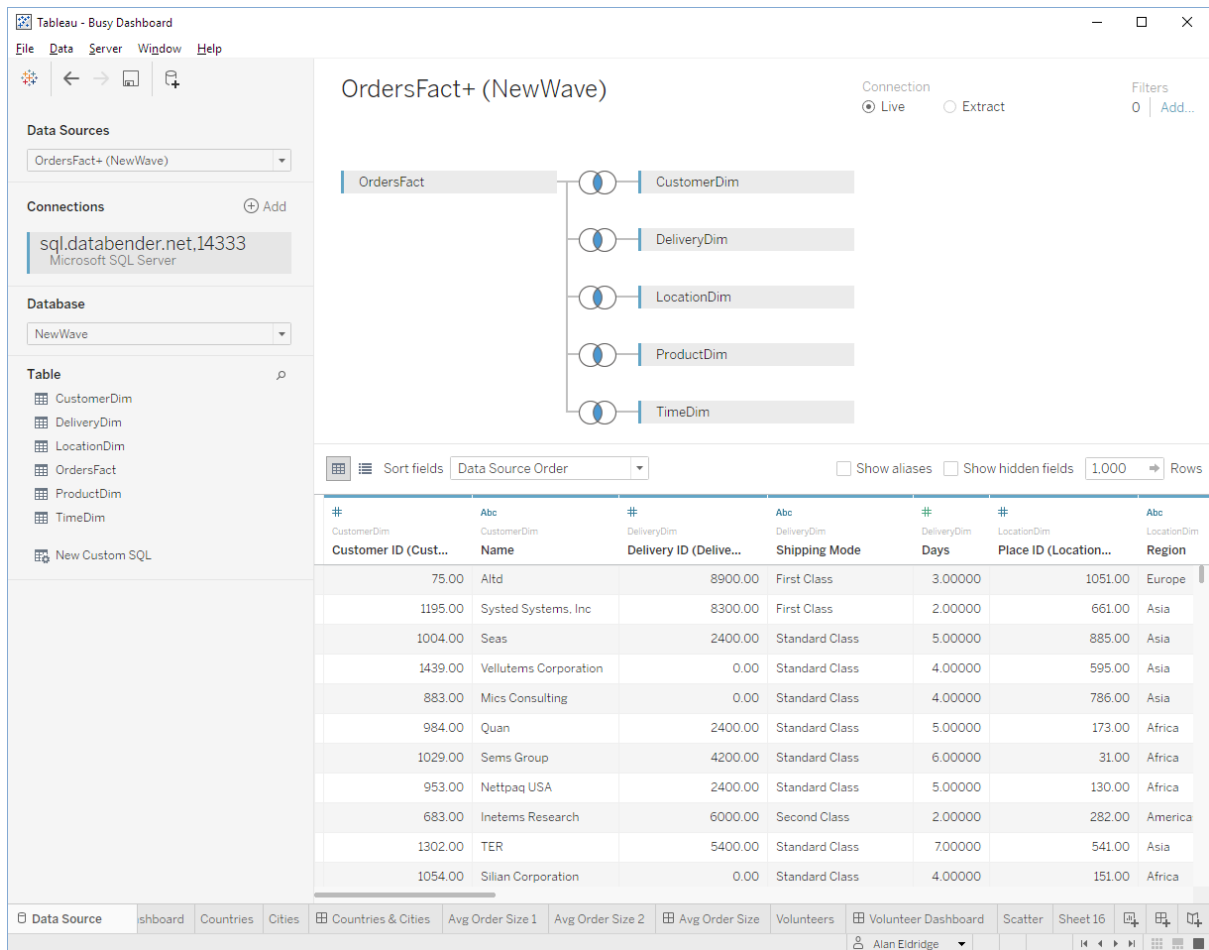
如果产品列设置为“ALLOW NULL（允许 NULL）”，则需运行查询，查看 NULL 是否属于子查询返回的销量前 10 产品。如果是，则需要执行保留 NULL 的联接，这比普通联接需要更大量的计算。如果产品列设置为“NOT NULL（非 NULL）”，则子查询的结果不可包含 NULL，由此可跳过“check for NULL（检查 NULL）”查询来执行普通联接。

因此，须尽可能将维度列定义为“NOT NULL（非 NULL）”。

引用完整性

联接数据源中的多个表时，Tableau 有一个很巧妙（且通常用户看不到）的功能，称为“联合选择”。由于联接在数据库服务器上处理起来耗时又耗资源，所以我们真的不想总是逐一处理在数据源中声明的每一个联接。联合选择用于可只查询相关表，而不联接中定义的所有表。

请考虑下面的场景。在该场景中，我们以一个小的星型架构联接多个表：



如果具有联合选择，则双击“Sales（销售额）”度量会生成以下查询：

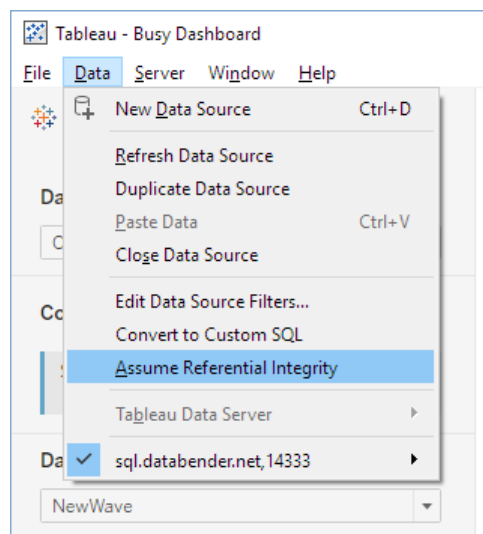
```
SELECT SUM([OrdersFact].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[OrdersFact] [OrdersFact]
GROUP BY ()
```

若没有，则会生成效率较低的查询：

```
SELECT SUM([OrdersFact].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[OrdersFact] [OrdersFact]
INNER JOIN [dbo].[CustomerDim] [CustomerDim]
ON ([OrdersFact].[Customer ID] = [CustomerDim].[Customer ID])
INNER JOIN [dbo].[DeliveryDim] [DeliveryDim]
ON ([OrdersFact].[Delivery ID] = [DeliveryDim].[Delivery ID])
INNER JOIN [dbo].[LocationDim] [LocationDim]
ON ([OrdersFact].[Place ID] = [LocationDim].[Place ID])
INNER JOIN [dbo].[ProductDim] [ProductDim]
ON ([OrdersFact].[Product ID] = [ProductDim].[Product ID])
INNER JOIN [dbo].[TimeDim] [TimeDim]
ON ([OrdersFact].[Date ID] = [TimeDim].[Date ID])
GROUP BY ()
```

2008-2012 年的数据，但时间维度表只有 2010-2012 年的值，则结果 SUM([Sales]) (SUM([销售额])) 可能根据是否包含时间表而有所不同。

以前，DBMS 强制执行规则时需要“硬”引用完整性。但是，在很多客户的数据源中，引用完整性在应用程序层中或通过 ETL 进程强制使用 - 这就是所谓的“软”引用完整性。用户可通知 Tableau 其已部署好软引用完整性且可通过启用“假设存在引用完整性”安全使用联合选择：



请注意，尽管 Tableau 可使用任一引用完整性，但通常最好使用硬引用完整性，因为此时数据库还可执行联合选择。相关详细信息，请参阅 Russell Christopher 在 Tableau Love 博客上发表的以下系列文章：

- <http://tableaulove.tumblr.com/post/11692301750/what-i-learned-about-tableau-join-culling-over>
- <http://tableaulove.tumblr.com/post/62447366098/what-i-learned-about-tableau-join-culling-over>

分区

数据库分区将把大表拆分成多个单独的更小的表（称为分区或碎片），从而改善性能。这意味着查询可以更快运行，因为要扫描的数据更少和/或有更多驱动器处理 I/O。分区是大数据卷的推荐策略，并且对 Tableau 透明。

如果是跨一个维度执行分区，那么分区对 Tableau 效果很好，例如时间、区域、类别等维度。通常筛选这类维度可以使各个查询只须读取单个分区内的记录。

请注意，对于某些数据库，必须使用“日期范围”筛选器（不是离散筛选器）才能确保分区索引得到正确使用，否则全表扫描可能导致极其糟糕的性能。

临时表

Tableau 中的许多操作都会导致使用临时表 - 例如创建临时组和集，以及执行数据混合。建议授予用户创建和删除临时表的权限，确保环境中足够多的假脱机空间可容纳正在运行的查询。

OLAP

Tableau 支持多种 OLAP 数据源：

- Microsoft Analysis Services
- Microsoft PowerPivot (PowerPivot for Excel 和 PowerPivot for SharePoint)
- Oracle Essbase

- SAP Business Warehouse
- Teradata OLAP

由于 MDX/DAX 和 SQL 间的基本语言差异，连接到 OLAP 和关系数据源时将存在功能性上的差异。关键是要记住，它们在 Tableau 中具有相同的用户界面和可视化效果，且用于计算度量的表达式语言相同。两者间的差异主要与元数据（如何定义元数据，以及元数据定义在何处）、筛选、合计和聚合的计算方式，以及数据源在数据混合中的使用方式有关。

若要更详细地了解针对关系数据源和 OLAP 数据源使用 Tableau 时的差异，可参阅以下知识库文章：

<http://kb.tableau.com/articles/knowledgebase/functional-differences-olap-relational?lang=zh-cn>

SAP BW 数据提取

SAP BW 是 OLAP 数据源中的一种独特数据源，因为可将 SAP BW 多维数据集中的数据提取到 Tableau 的数据引擎中（请注意，这需要可从 Tableau 中获取的特殊键码）。Tableau 检索叶级节点（不是钻透级数据），并使这些节点变成关系数据源。由于多维到关系的转换不保留所有多维数据集结构，因此多维数据集提取不支持在不影响可视化状态的前提下，在提取和实时连接之间随意来回切换。需在开始构建可视化对象之前做出选择。但无需事先决定一切。可在提取后，切换使用不同的别名选项（键、长名称等）。

若要更详细地了解如何创建 SAP BW 提取，可参阅此处：

<https://community.tableau.com/docs/DOC-9914>

大数据

在数据分析时代，大数据是一个频繁使用的术语，但本文中专用于指基于 Hadoop 的平台。Tableau 10 中存在 4 种受支持的分布，它们支持 Hive 和/或 Impala 连接：

- Amazon EMR
 - HiveServer
 - HiveServer2
 - Impala
- Cloudera Hadoop
 - HiveServer
 - HiveServer2
 - Impala
- Hortonworks Hadoop Hive
 - HiveServer
 - HiveServer2
 - Hortonworks Hadoop Hive
- MapR Hadoop Hive
 - HiveServer
 - HiveServer2
- Spark SQL
 - SharkServer *

- SharkServer2 *
- SparkThriftServer

*请注意，*SharkServer* 和 *SharkServer2* 连接可供使用，但不受 *Tableau* 支持。

Hive 充当 SQL-Hadoop 转换层，将查询转换为 MapReduce，后者随后对 HDFS 数据运行。Impala 直接对 HDFS 数据执行 SQL 语句（无需转换为 MapReduce）。Tableau 还支持 Spark SQL，这是一种面向大数据的开源处理引擎，通过在内存中（而非磁盘上）运行，该引擎的执行速度可比 MapReduce 的快达 100 倍。

Impala 通常比 Hive 快得多，但经证明，Spark 速度最快。

即使有了这些额外的组件，Hadoop 通常仍不足以响应分析查询，比如 Tableau 创建的那些查询。Tableau 数据提取通常用来缩短查询响应时间 - 有关提取的更多信息，以及如何配合“大数据”使用提取将在后面讨论。

若要更详细地了解如何提高 Hadoop 数据源性能，可参阅此处：

<http://kb.tableau.com/articles/knowledgebase/hadoop-hive-performance?lang=zh-cn>

云

Tableau 当前支持以下云数据源：

- Salesforce.com
- Google Analytics
- oData（包括 Microsoft Azure Marketplace DataMarket）

这第一组数据源从 Web 服务读取一组数据记录，然后将它们载入 Tableau 数据提取文件。“实时连接”选项不适合这些数据源，但是可以刷新提取文件来更新其包含的数据。使用 Tableau Server 时，这一更新过程可以自动化以及定时执行。

Tableau 还支持以下基于云的数据平台：

- Amazon Aurora
- Amazon Redshift
- Amazon RDS
- Google BigQuery
- Google Cloud SQL
- Google 表格
- Microsoft Azure SQL 数据仓库
- Snowflake

与之前的数据源组相比，这些平台处理诸如关系数据源的来源，且允许实时连接和提取。此处不会进一步说明（详见上文的关系数据源部分），但会指出通常需将其保留为实时连接，以避免从云端传输大量数据。

最后，Tableau 还会提供一个泛型数据连接器（即 Web 数据连接器），用于从按 JSON、XML 或 HTML 格式发布数据的基于 Web 的服务中导入数据。

Salesforce

连接到 Salesforce 时，需要考虑连接器的以下限制：

没有“实时连接”选项

选择仅使用提取而不使用实时连接有几个原因，包括：

- 性能 - 针对 Salesforce 的实时分析查询（通常）较慢。
- API 配额 - 过于频繁地使用 Salesforce 可能会导致帐户挂起（如果达到每日配额）。进行提取时，可高效利用 API，尽量减少所需的 API 调用次数并避免触及限制。若要维持最优性能并确保所有客户均能使用 Force.com API，Salesforce.com 将强制实行两种限制来平衡事务负载：并行 API 请求限制 (<http://sforce.co/1f19cQa>) 和全部 API 请求限制 (<http://sforce.co/1f19kiH>)。

首次提取可能非常慢

首次提取 Salesforce 中的数据时，可能需要一段时间，具体取决于表格大小和 Force.com 加载等。这是因为需下载完整对象。

联接选项受限

选择多个表选项时，请注意仅可在对象的 PK/FK 键上进行联接（仅允许左侧和内部联接）。

不能预筛选数据

不能通过 Salesforce 连接器预筛选数据。如果非常需要此功能，可使用支持实时连接的第三方 Salesforce ODBC 驱动程序（例如来自 Simba 或 DataDirect 等）。然后即可对此连接进行提取。

DBAmp 还提供可将 Salesforce 数据转储到 SQL Server 数据库的解决方案。可通过 SQL Server 连接器连接到 Tableau。

不能跨公式列迁移

如果具有计算字段，则在提取数据后，需要在 Tableau 中重新创建这些计算。

查询限制为 10 万个字符

Force.com API 将总查询数限制为 10 万个字符。如果要连接一个或多个很宽的表（很多列，可能有很长的列名），那么在尝试创建提取时，很可能达到这一限制。这种情况下，应该少选择列以减小查询的大小。有些情况下，Salesforce.com 可能会为您的公司放宽此查询限制。请联系 Salesforce 管理员了解详细信息。

Google Analytics

当报告包含大量维度或大量数据时，Google Analytics (GA) 会对数据取样。如果某个特定 Web 属性在给定日期范围内的数据超过了（对于某个常规 GA 帐户）5 万次访问，GA 将聚合结果，并返回该数据的样本集。当 GA 将该数据的样本集返回到 Tableau 时，Tableau 将在视图右下角显示以下消息：

“Google Analytics 返回了抽样数据。当连接包含大量维度或大量数据时会进行抽样。若要深入了解抽样如何影响报告结果，请参见 Google Analytics 文档。”

有必要知道的一点是，在抽样数据时，由于聚合，某些数据样本集可能产生非常歪曲和不准确的推论。例如，假设您聚合了一个数据样本集，这些数据描述了一个不常见的数据类别。因为该类别包含的样本数不足，因此对聚合的样本集的推论可能出现歪曲。为了建立能让您做出准

确数据推论的 GA 视图，应该确保您要做出推论的类别中包含足够大的样本。推荐的最小样本大小为 30。

若要了解如何调整 GA 样本大小并深入了解 GA 采样，请参阅 GA 文档：

<http://bit.ly/1BkFoTG>

为了避免采样，可以采取两种方法：

- 运行多个会话级或命中级的 GA 报告，将数据分成多个非抽样的块。然后将数据下载到 Excel 文件，再通过 Tableau 提取引擎启用“add data from data source... (从数据源添加数据...)”选项，将数据重新组合成单个数据集。
- 将 GA 升级到专业版帐户 – 这可增加可包含在报告中的记录数。这样就可以更容易地将数据分块以便分析。对于将来，Google 已经宣布，他们将使 GA 专业版帐户能够将其会话级和命中级数据导出到 Google BigQuery 做进一步分析。这会是一种更加简单的方法，因为 Tableau 可以直接连接到 BigQuery。

最后请注意，Tableau 用来查询 GA 的 API 将查询限制为最大 7 个维度和 10 个度量。

Web 数据连接器

Tableau Web 数据连接器让您能够连接此前没有连接器的数据。使用 Web 数据连接器，您可以连接并使用可通过 HTTP 访问的几乎所有数据。包括内部 Web 服务、JSON 数据、XML 数据、REST API 和许多其他数据源。您可以控制数据的收集方式，因此您甚至能够将来自多个源的数据合并到一起。

可通过编写包含 JavaScript 和 HTML 的网页，创建 Web 数据连接器。编写 Web 数据连接器后，可通过将其发布到 Tableau Server 与其他 Tableau 用户共享。

为帮助您创建 Web 数据连接器，我们创建了包含模板、示例代码和模拟器（可用于测试 Web 数据连接器）的软件开发工具包 (SDK)。本文还包括指导从头创建 Web 数据连接器的指南。Web 数据连接器 SDK 是一个开源项目。有关详细信息，请参阅 GitHub 上的“Tableau Web Data Connector (Tableau Web 数据连接器)”页面。

若要深入了解 Web 数据连接器，请参阅此处：

<http://tabsoft.co/1NnGsBU>

数据准备

工作簿中所需使用的数据有时并不完美。数据可能不清晰，格式可能不正确，且可能分布在多个文件或数据库中。过去，Tableau 在使用规范化的清晰数据时性能最佳，这意味着在将数据加载到 Tableau 之前，有时需要使用其他工具准备数据。

该情况现已不复存在 - Tableau 现有多项功能，可帮助加载杂乱数据。这些功能包括：

- 数据透视表
- 并集
- 数据解释器
- 合并列

在计算期间准备数据：a) 有助于继续执行分析；b) 允许用户无需借助其他工具即可继续在 Tableau 中分析数据。但是，这些操作通常需要大量计算（尤其是针对大量数据），还可能会影响报告用户的性能。执行这些操作后，建议尽可能使用数据提取来具体化数据。

还需考虑在现有 ETL 进程添加这些功能作为补充，并考虑在跨多个报告使用数据或与其他 BI 工具共享数据时，预处理 Tableau 的数据上游是否更好、更高效。

数据提取

目前为止，我们讨论了提高数据连接性能的技巧，在这些数据连接中，数据仍保留其原有格式。我们称其为实时数据连接，在这类连接中，我们依靠源数据平台来获得性能和功能。为了提高实时连接的性能，通常有必要改动数据源，这对很多客户来说简直没有可能。

适合所有用户的替代方法是，利用 Tableau 的快速数据引擎，并将数据从源数据系统提取到 Tableau 数据提取。这是适用于多数用户的最快、最简便方式，可针对任何数据源大幅提高工作簿的性能。

提取是指：

- 已写入磁盘且可复制的持久数据缓存；
- 列式数据存储 – 一种优化数据以供分析查询的格式；
- 查询期间，完全脱离数据库。实际上，提取是实时数据连接的替代品；
- 可刷新，刷新方式可以是完全重新生成提取，也可以是将数据行增量添加到现有提取；
- 体系结构感知 – 与大多数内存中技术不同，提取不受可用物理 RAM 量的限制；
- 可移植 – 提取存储为文件，因此可以复制到本地硬盘驱动器，并在用户未连接到公司网络时使用。它们还可用来将数据嵌入打包工作簿，可以分发打包工作簿以供在 Tableau Reader 中使用；
- 这通常比基础实时数据连接快得多。

The Information Lab 的 Tom Brown 曾写过一篇优秀的文章，解释了提取可带来好处的多种用例（请确保同时阅读其他用户对附加示例所作的评论）：

<http://www.theinformationlab.co.uk/2011/01/20/tableau-extracts-what-why-how-etc/>

关于数据提取需重点注意：数据提取不是数据仓库的替代，而是补充。虽然它们可用于收集和聚合一段时间的数据（即根据周期增量添加数据），但这须用作权宜之计，而非长期解决方案。增量更新不支持对已处理过的记录执行更新或删除操作 – 这样的更改需要完全重新加载提取。

最后，不能基于 OLAP 数据源创建提取，例如 SQL Server Analysis Services 或 Oracle Essbase 等数据源。此规则存在例外，即可从 SAP BW 创建提取（详见上文相关部分）。

何时使用提取？何时使用实时连接？

毫不例外，数据提取也有其使用时机和场合。以下是使用提取可能有利的某些场合：

- 查询执行缓慢 - 如果源数据系统在处理 Tableau Desktop 生成的查询时速度缓慢，那么创建提取可能是改善性能的简单方法。提取数据格式本来就是为了快速响应分析查询而设计的，因此在这种情况下，可以将提取视作查询加速缓存。对于某些连接类型，这是推荐的最佳做法（例如，大文本文件、自定义 SQL 连接），而某些数据源只有在这种模型下才有效（请参阅有关云端数据源的部分）。

- 离线分析 - 如果需要在原始数据源不可用时处理数据（例如，在旅行或在家工作时与网络断开）。数据提取可持久保存为文件，文件可轻松复制到便携式设备，如笔记本电脑。如果要在网上和网下工作，那么在提取和实时连接之间来回切换将是轻而易举的事情。
- 用于 Tableau Reader/Online/Public 的打包工作簿 - 如果您打算共享工作簿，以便其他用户在 Tableau Reader 中打开，或者如果您打算将工作簿发布到 Tableau Online 或 Public，那么您需要将数据嵌入打包工作簿文件。即使工作簿使用也可嵌入的数据源（即，文件数据源）。由于数据提取本身提供了很高的数据压缩度，因此得到的打包工作簿体积也明显更小。
- 额外功能 - 有些数据源（如通过旧版 JET 驱动程序的文件数据源）不支持 Tableau Desktop 中的某些功能（例如中值/无重复计数/级别/百分位聚合、集内/集外操作等）。提取数据是启用这些功能的简单方法。
- 数据安全性 - 如果希望共享源数据系统中的部分数据，可以创建提取并将其提供给其他用户。可以限制要包含的字段/列，如果想要用户只看到摘要值，而看不到单条记录级的数据，那么还可以共享聚合数据。

提取非常强大，但它们不是解决一切问题的万灵药。有些场合可能不适合使用提取：

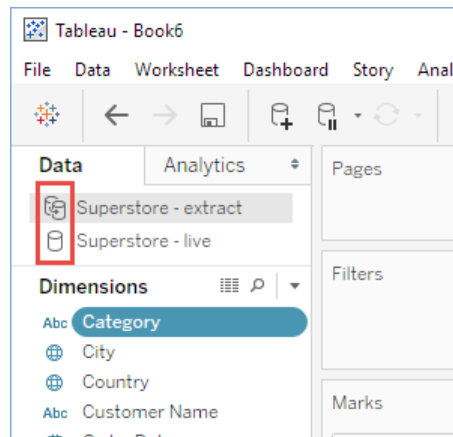
- 实时数据 - 由于提取是数据的时间点快照，因此如果您的分析需要实时数据，那么提取就不太适合。虽然可以使用 Tableau Server 自动刷新提取，而且很多客户也确实以一天不到的频率这样做，但是真正的实时数据访问需要实时连接。
- 海量数据 - 如果需要处理的数据量达到海量（“海量”的定义因用户而异，但是通常是指几百万到几十亿条记录），那么提取这么多数据可能不切实际。得到的提取文件可能超大，或者提取过程可能需要很多、很多小时才能完成。请注意，这条规则有几个例外。如果有一个庞大的数据集，但是您只想处理这些数据中经过筛选的、抽样的和/或聚合的部分数据，那么使用提取实际上可能不失为好主意。一般来说，Tableau 提取引擎的设计可以很好地处理多达几亿条记录，但是这会受到数据的形状和基数的影响。
- 传递 RAWSQL 函数 - 如果工作簿使用传递函数，这些函数对数据提取不起作用。
- 稳健用户级安全性 - 如果需要稳健实施的用户级安全性，那么这需要在数据源中实施。如果在工作簿级别应用了用户级别筛选器，则其始终可由用户删除，使用户可访问提取中的所有数据。此规则的例外是，提取发布到定义了数据源筛选器的 Tableau Server，且其他用户通过 Data Server 访问此提取。注 - 您需要确保撤消用户的下载权限，以保证他们无法绕过施加的筛选器。

在 Tableau Desktop 中创建数据提取

大多数情况下，数据提取的初始创建在 Tableau Desktop 中完成，而且很简单。在连接到数据后，转到“Data（数据）”菜单，单击“Extract Data（提取数据）” – 然后接受对话框中的默认设置（以后还会介绍更多这方面的内容）。Tableau 会询问您要将提取保存在何处 – 请选择任意位置保存文件；虽然 Tableau 可能会转到“My Tableau Repository | Datasources（我的 Tableau 存储库 | 数据中心）”，但此处当前也行！

现在等待创建提取，等待时间取决于所用的数据库技术、网络速度、数据量等。同时还取决于工作站的速度和容量，因为创建提取是很占内存和处理器的活动。

数据源图标改变时，就表示提取创建完成 – 图标后面还有一个数据库图标，表示与提取完全相同的“副本”。



按此方式（通过 Tableau Desktop）创建数据提取时，处理将在工作站上进行，因此需确保工作站的容量充足，可完成此任务。提取创建操作使用所有资源类型：CPU、RAM、磁盘存储、网络 I/O；要是在处理能力较弱的 PC 上处理大数据量，那么只要有任何资源耗尽，就会导致错误。所以建议在适当的工作站上执行较大的提取，也就是配备快速的多核 CPU、大量 RAM、快速 I/O 等的工作站。

创建数据提取时，需要临时磁盘空间来编写工作文件 - 所需空间为最终提取文件大小的两倍。此工作空间将分配在 TEMP 环境变量指定的目录中（通常是 C:\WINDOWS\TEMP 或 C:\Users\USERNAME\AppData\Local\Temp）。如果驱动器空间不足，请将该环境变量指向空间更大的位置。

如果初始提取过程不可能在工作站上执行（或者这样做不切实际），那么可以执行以下变通方法来创建一个空提取，空提取随后发布到 Tableau Server 上。创建内附 `DateTrunc("minute", now())` 的计算字段。然后将其添加到提取筛选器，并排除其显示的单个值 - 做这一步的时候要快，因为您只有一分钟时间，超过一分钟该筛选器就不再有效。如果需要更长时间，可以放宽发布时间间隔（例如，放宽到大约 5 分钟或 10 分钟，如果需要，甚至可以放宽到 1 小时）。这将在桌面上生成一个空提取。在发布到服务器并触发刷新计划后，系统将填充整个提取，因为我们排除的时间戳已经不再一样。

使用数据提取 API 创建数据提取

Tableau 还提供应用程序编程接口 (API)，允许开发人员直接创建 Tableau 数据提取 (TDE) 文件。开发人员可使用此 API 从本地软件和软件即服务中生成数据提取。如果本机连接器均不可用于当前使用的数据源，可借助此 API 将数据系统性地放入 Tableau。

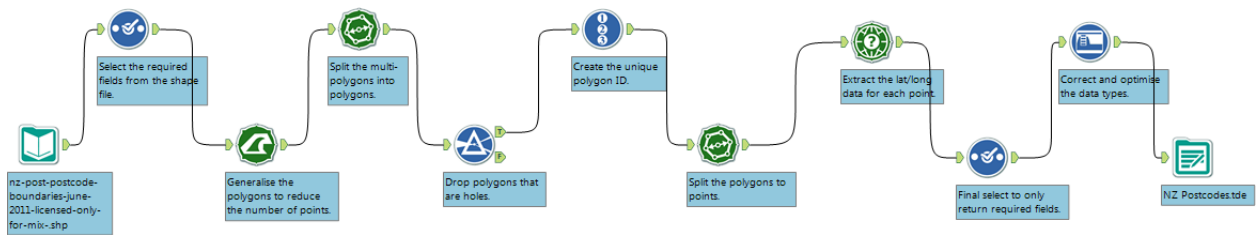
开发人员可在 Windows 和 Linux 的 Python 和 C/C++/Java 中使用此 API。有关 API 的详细信息，可参阅此处：

http://onlinehelp.tableau.com/current/pro/desktop/zh-cn/extracting_TDE_API.html

使用第三方工具创建数据提取

许多第三方工具开发人员使用数据提取 API 向应用程序添加本地 TDE 输出。这些应用程序包括 Adobe Marketing Cloud 等分析平台，还包括 Alteryx 和 Informatica 等 ETL 工具。

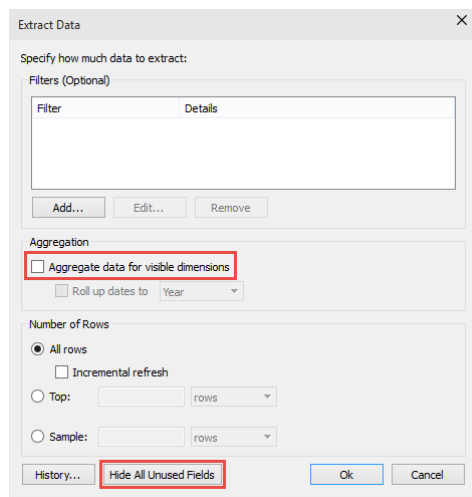
如果需要进行复杂的数据准备，可使用 Alteryx 和 Informatica 等工具高效执行 ETL 的各阶段，然后将准备好的数据直接输出到 TDE 文件中，以便在 Tableau Desktop 中使用。



聚合提取

使用聚合提取总是可以提高性能。即使是使用包含巨量数据的 Teradata 或 Vertica，只要聚合和筛选数据得当，提取数据仍能改善性能。例如，如果只关心最新数据，则可筛选数据。

可提前定义提取，方式是选择所需字段，然后在 Tableau Desktop 的“Extract Data（提取数据）”对话框中，勾选“Aggregate data for all visible dimensions（聚合可视维度的数据）”复选框。或者，在执行分析并生成仪表板后，在您准备发布时，可以返回到“Extract Data（提取数据）”对话框，随后单击“Hide All Unused Fields（隐藏所有未使用的字段）”按钮。提取数据时，这将是创建视图时所需的绝对极小量。



如果基本数据量较大，但需要创建跨整个数据集进行查询的摘要视图，则创建聚合数据提取非常强大。例如，您可能拥有 10 亿条详细交易数据记录表示 10 年的销售额，您希望首先显示 10 年的整体销售额趋势。此初始视图的查询可能较慢，因为它需要在 10 亿行记录中查询。通过创建按年份级别进行聚合的数据提取，由于此提取中只有 10 个数字，因此可减少查看时所需的查询。当然，这是一个过于简化的示例 - 在实际情况中，还会遇到更多维度，除了时间问题，还涉及到能显著减少查看时需查询的记录数。

可创建具有多个详细级别的复杂工作簿，方式是创建多个聚合提取，每个提取调整为支持特定的详细级别，或者通过将聚合提取与实时连接进行组合。例如，您可能拥有一组使用高度聚合提取的初始摘要视图，但在深入细节时，您会使用摘要视图中的操作筛选器，筛选通过实时连接进行连接的其他工作表。也就是说，摘要视图速度较快，因其无需查阅整个基本数据集，而且我们无需提取所有基本数据即可支持下钻查询操作。此外，实时连接可快速执行，因为下钻查询级别仅能访问小部分记录集。

这样，就可以混合搭配，并在不同级别进行聚合以解决几乎任何性能问题，从而尽量快地得出结果。由于 Tableau 使用内存非常高效，因此以这种方式提高性能通常相对容易，而且可以多个提取可以同时运行。

优化提取

Tableau Server 不仅优化数据库中的物理列，而且还优化在 Tableau 中创建的附加列。这些列包含确定性计算的结果，如字符串操作和连接。在这类计算中，结果不会变化，组和集也不会。非确定性计算的结果无法存储，比如在运行时计算的，涉及参数或聚合（如总计或平均）的那些计算。

用户可能在仅仅添加两行数据后刷新提取，然后发现提取的大小已从 100MB 跃升到 120MB。大小的这种跃升是由于创建包含计算字段值的附加列所需的优化，因为将数据存储到磁盘的代价要比每次需要数据时都重新计算来得低。

要注意的一点是，如果复制了数据提取连接的副本，则需确保选用于“Optimize（优化）”或“Refresh（刷新）”选项的所有计算字段均存在于此连接中，否则 Tableau 不会具体化其认为未使用的字段。好的习惯是，在主数据源中定义所有计算字段，并根据需要将其复制到另一个连接，然后只从主数据源刷新或优化提取。

附加说明：据开发团队中的可信来源称，虽然可将多个连接创建为单个 TDE，但之前从未真正支持过该功能。如果连接未与 TDE 结构同步，会导致很多问题。其建议是，不要执行此操作。

刷新提取

在 Tableau Desktop 中，为刷新提取，需选择菜单（“数据”菜单 > [您的数据源] > “提取” > “刷新”），这将更新数据并添加所有新行。但在 Tableau Server 中，在发布过程期间或之后，可以附加由管理员定义的计划来自动刷新提取。允许的最小计划增量是每 15 分钟；计划可以在每天、每周等同一时间刷新。可设置“移动窗口”，连续刷新数据以保持最新。

注意：如果希望数据刷新间隔小于 15 分钟，应考虑连接到实时数据，或者设置同步报告数据库。

可以为一个提取选择两个刷新计划：

- 增量刷新只添加行，不包括对现有行的更改。
- 完全刷新丢弃当前提取，并从新从数据源重新生成新的提取。

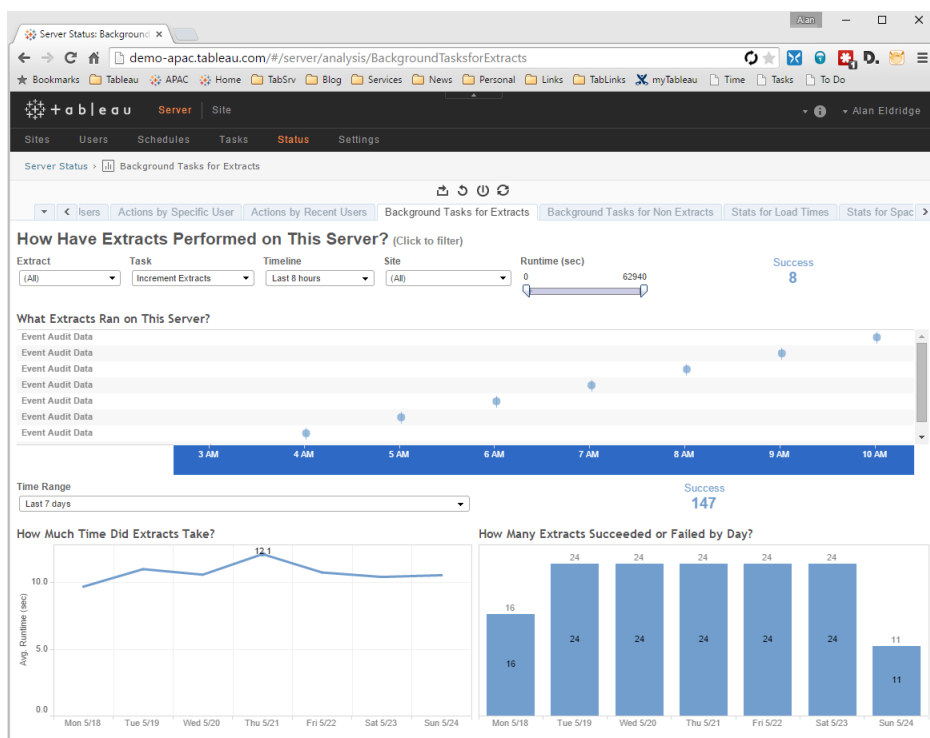
如果刷新时间比增量时间长会发生什么？

如果提取刷新时间大于增量时间，则不会干预刷新。例如，计划设置为每小时刷新一次数据，而数据量太大，以至于刷新需要一个半小时。那么：

- 首次刷新于 1:00 开始，2:30 结束。
- 下一次刷新计划于 2:00 开始，但由于仍在运行第一次刷新，因此跳过本次刷新。
- 下一次刷新在 3:00 开始，4:30 结束。

提取维护

“维护”屏幕显示了当前有哪些后台任务正在运行，还有在过去 12 小时里运行的那些任务。颜色编码用来显示那些任务的状态。“维护”屏幕可供管理员以及有相应权限的某些其他用户使用，他们有权发起对提取的临时更新。另外，例如要加载数据库，可以设置触发器在数据库完成加载后启动提取。



如果使用的是 Tableau Server，则可通过 `tabcmd` 命令行工具完全或按增量方式刷新工作簿；如果使用的是 Tableau Desktop，还可通过 `Tableau.exe` 命令行执行此操作。如果有复杂的时间安排要求，可以从外部计划工具（如 Windows 任务计划程序）调用这些命令。如果希望刷新周期短于 15 分钟（Tableau Server 界面上允许的最短时间），则需采用此方法。

最后，Tableau Online 的用户可使用 Tableau Online 同步客户端，按照 Online 服务上定义的计划将本地数据源保持为最新状态。关此实用程序的详细信息，可参阅此处：

http://onlinehelp.tableau.com/current/online/zh-cn/help.htm#qs_refresh_local_data.htm

数据管理

虽然 Data Server 本身不是数据源，但是连接数据源的另一种方法是通过 Tableau Server 的 Data Server。Data Server 支持实时连接，也支持数据提取，而且带来了胜过独立数据连接的多项优势：

- 由于元数据集中存储在 Tableau Server 上，因此元数据可在多个工作簿之间以及多名作者/分析师之间共享。工作簿保留指向集中元数据定义的指针，每当工作簿打开时，它们会检查是否发生过任何更改。如果有更改，系统将提示用户更新嵌在工作簿中的副本。也就是说，只需在一个地方做出对业务逻辑的更改，然后更改就会传播到所有相关的工作簿。
- 如果数据源是数据提取，它就可以在多个工作簿中使用。如果没有 Data Server，每个工作簿将包含各自的本地提取副本。这减少了冗余副本的数量，进而减少了服务器上所需的存储空间，以及重复的刷新过程。
- 如果数据源是实时连接，那就无需在每位分析师的 PC 上安装该数据源的驱动程序，只要在 Tableau Server 上安装即可。Data Server 充当 Tableau Desktop 发出查询的代理。

是环境吗？

有时，工作簿在单用户测试时性能优良，但在部署到 Tableau Server（或 Tableau Online）供多名用户使用时，性能大幅降低。以下部分介绍了单用户和多用户场景之间的差异可能会对哪些方面造成影响。

升级

Tableau 开发团队不懈努力，尽力提升软件性能和可用性。升级到 Tableau Server 最新版，有时完全无需更改工作簿即可大大提升性能和稳定性。

若要查看“Tableau 发行说明”页面并升级到最新版，可参阅：

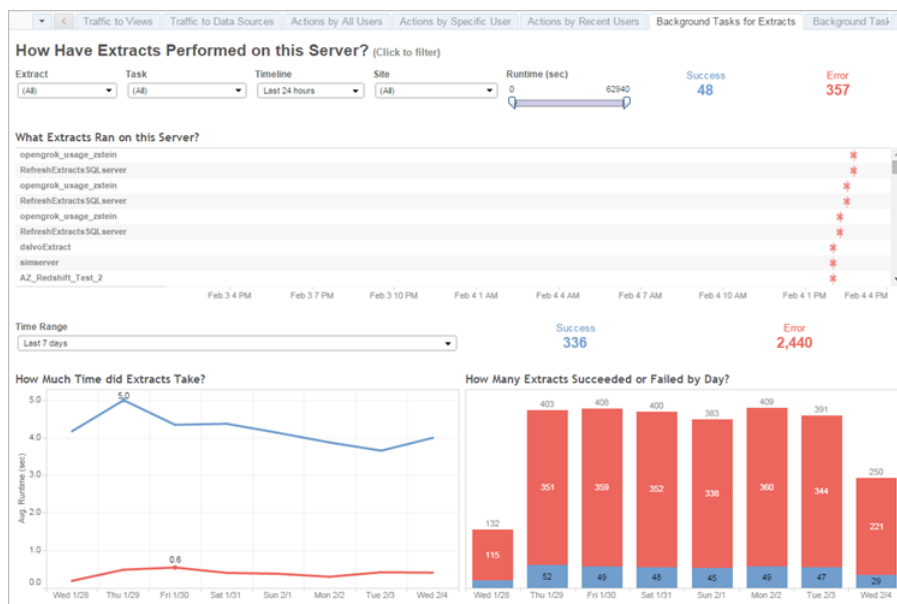
<http://www.tableau.com/zh-cn/support/releases>

在服务器上测试 Tableau Desktop

有时，您可能会发现工作簿在工作站上的 Tableau Desktop 中运行时性能良好，但通过 Tableau Server 查看时运行不佳。使用 Tableau Server 计算机上安装的 Tableau Desktop 副本打开工作簿，这有助于确定这是工作簿的问题还是服务器的配置问题。此方法能识别这是驱动程序的不兼容问题，还是路由、DNS 或代理配置不良等网络问题。

独立式刷新和交互式工作负荷

如果服务器性能较低，请使用“Background Tasks（后台任务）”管理视图查看当前的刷新任务计划。



如果可安排的非高峰时段进行刷新，请如此操作。如果硬件配置允许，还可将后台程序进程移到专用工作节点上。

监视和调整服务器

Tableau Server 为管理员提供多个视图，帮助监视 Tableau Server 中的活动。这些视图位于服务器“Maintenance（维护）”页面的“Analysis（分析）”表中：

Views	Analysis
Traffic to Views	View count, viewers, and viewer behavior for published views.
Traffic to Data Sources	Data source usage, users, and user behavior for published data sources.
Actions by All Users	Actions for all users.
Actions by Specific User	Actions for a specific user, including items used.
Actions by Recent Users	Recent actions by users, including last action time and idle time.
Background Tasks for Extracts	Completed and pending extract task details.
Background Tasks for Non Extracts	Completed and pending background task details (non-extract).
Stats for Load Times	View load times and performance history.
Stats for Space Usage	Space used by published workbooks and data sources, including extracts and live connections.

有关这些视图的更多信息，可访问以下链接：

<http://onlinehelp.tableau.com/current/server/zh-cn/adminview.htm>

此外，还可连接到 Tableau 存储库中的 PostgreSQL 数据库，创建自定义管理视图。相关说明请参阅下方：

http://onlinehelp.tableau.com/current/server/zh-cn/adminview_postgres.htm

检查 VizQL 会话超时限制

VizQL 会话超时限制默认为 30 分钟。VizQL 会话空闲时仍会占用内存和 CPU 周期。如果较低的限制即可足够，请使用 tabadmin 更改 vizqlserver.session.expiry.timeout 设置：

http://onlinehelp.tableau.com/current/server/zh-cn/reconfig_tabadmin.htm#ida6864815-db67-4a51-b8b6-c93617582091

评估进程配置

Tableau Server 分为多个称为服务的不同组件。虽然其默认配置旨在适用于各种情况，但也可重新配置，实现不同的性能目标。具体而言，可控制运行进程的计算机以及运行数量。有关单机、双机和三机部署的准则，请参阅“提高服务器性能”：

http://onlinehelp.tableau.com/current/server/zh-cn/perf_extracts_view.htm#idd21e8541-07c4-420f-913e-92dcaf5f0c34

基础设施

64 位

Tableau 10 之前的 Tableau Server 版本可在 32 位 Microsoft 操作系统上运行，但强烈建议用户使用 64 位版本安装产品环境。仅建议在安装“开发/测试”时使用 32 位版本。

自 Tableau 10 及更高版本起，Tableau Server 应用程序仅提供 64 位版本。

添加更多 CPU/RAM

无论在一台还是多台计算机上运行 Tableau Server，一般规则是 CPU 内核和 RAM 越多，获得的性能越佳。请确保符合 Tableau Server 推荐的硬件和软件要求

(<http://onlinehelp.tableau.com/current/server/zh-cn/requ.htm#ida4d2bd02-00a8-49fc-9570-bd41140d7b74>)，另请参阅“何时添加工作软件和重新配置”

(http://onlinehelp.tableau.com/current/server/zh-cn/distrib_when.htm#idfdd60003-298e-47e6-8133-3d2490e21e07)，评估是否应添加更多计算机。

TabMon (参见本文档前文) 是一款极其出色的工具，可收集有助于规划容量进程的使用率数据。

请勿忽略 IO

Tableau 的某些操作需要大量 I/O (如加载/创建/刷新数据提取)，使用 SSD 比使用旋转式磁盘更有利。Russell Christopher 在其 Tableau Love 博客上发表了几篇精彩博文，其中探讨了内核数、CPU 速度和 IOPS 对整体性能的影响。尽管他的实验是在 AWS 上完成的，但结论适用于所有环境：

<http://tableaulove.tumblr.com/post/80571148718/studying-tableau-performance-characteristics-on>
<http://tableaulove.tumblr.com/post/80783455782/tableau-server-performance-on-aws-ec2>

物理和虚拟

现在，很多客户的虚拟基础结构上都会部署 Tableau Server。虚拟总会出现性能瓶颈，因此速度不如在裸机上安装快，但是现代的管理程序技术已大大缓解了这种情况。

在虚拟机上安装时，请务必确保为 Tableau Server 提供专用的 RAM 和 CPU 资源。若与物理主机上的其他虚拟机竞争资源，将会严重影响性能。

若要调整虚拟部署，请参阅 VMWare 中的“Deploying Extremely Latency-Sensitive Applications in vSphere 5.5 (在 vSphere 5.5 中部署对延时极其敏感的应用程序)”白皮书。15 页的文档提供了针对延时敏感应用程序最佳做法的高级列表。

<http://www.vmware.com/cn.html>

浏览器

Tableau 大量使用 JavaScript，因此浏览器中 JavaScript 解释器的速度对渲染速度有影响。现代浏览器在快速发展的领域势均力敌，需要考虑的是浏览器的版本和性能是否会影响体验。

结语

某位智者曾说过（本文也受此启发） - “告诉人们您将要告诉他们的事，告诉他们，然后告诉他们您已经告诉他们的事”。真是睿智的建议。

再次强调，以下是我希望大家阅读本文后能够提取的要点：

- 没有什么是一劳永逸的。许多原因都会导致性能变慢。收集数据找到拖慢时间的原因。重点关注占速最大的区域，然后改进其他区域，以此类推。当速度足够快或者努力没有实际效果时，停止操作。
- 很多仪表板速度慢都是由设计不当导致的。简化设计。不要一次显示太多内容，尽可能使用指示的分析设计。
- 不使用不必要的数据库。使用筛选器、隐藏未使用的字段和进行聚合。
- 不要进行违背数据库引擎的操作。该引擎很智能且尽力提供帮助，因此请相信它会生成高效的查询。
- 数据越清晰，与问题结构的匹配度越高，工作簿运行就越快，您的体验也就会越愉快。
- 若要加快大多数工作簿的运行速度，数据提取是一种简单快捷的方法。如果不需要实时数据，且无需处理超过数十亿行的数据，则应尝试数据提取。
- 处理字符串和日期较慢，处理数字和布尔值较快。
- 尽管本文以多名有识之士的最佳建议为依据，但建议仅是建议而已。您需要测试哪些建议会针对您的具体情况改进性能。
- 如果处理的数据较少，上述多数建议则无需考虑 - 尽可以简单粗暴地解决。但在所有工作簿中实施这些建议没有坏处，毕竟您不知道数据何时会增加。
- 实践出真知。

立即行动，打造超级高效的工作簿吧！