効率的に作業できる Tableau ワークブックを設計するためのベストプラクティス

Tableau 10 版

アラン・エルドリッジ
Tableau Software

このホワイトペーパーについて

このドキュメントは、多くの作成者による資料の抜粋をまとめて作成したものであることを再度申し上げたいと思います。私の作業のほとんどは、多くの資料を 1 つにまとめ構造的に整理したことだけです。各セクションの文章に見覚えのある読者の方もいらっしゃることでしょう。実際、すべての部分を認識できる方もいるかもしれません。このドキュメントが存在するのは、作者の方々が継続的に素晴らしい資料を生み出し、著作権侵害の申し立てを起こさずにいてくださるおかげであり、そのことに感謝しています。

また、正確さと読みやすさを改善するためにこのドキュメントを校正してくださった多くの方々にも、お礼を申し上げます。細かい点に注意を払い辛抱強く説明していただいたことで、このドキュメントは私が以前自分だけで作成したものに比べずっと読みやすくなりました。

このドキュメントは、Tableau 10 の機能を反映して更新されたものです。Tableau のアップグレードで新機能が登場するのを受け、推奨事項が一部変更になることがあります。

どうぞよろしくお願いします。

アラン・エルドリッジ
2016 年 7 月

長い文章の要約

このホワイトペーパーを読んだ方や推奨した方から、ドキュメントとして長すぎると言われることがよくあります。それについては、幅広い内容をある程度深くカバーするためには、相当の長さが必要なのだと答えています。

ただし、このドキュメントの重要な点は以下のようにまとめることができます。

- 効率的に作業できるワークブックを作るための特効薬はありません。まずはパフォーマンスレコーダーを見ることから始め、どこに時間がかかっているのかを理解します。クエリーに時間がかかっているか。クエリーが多すぎるか。計算が遅いか。レンダリングが複雑であるか。こうしたインサイトを使って、正しい方向に力を向けるよう取り組んでください。
- このドキュメントにある推奨事項は、単なる推奨事項にすぎません。ベストプラクティスと言えるレベルではありますが、それが自分のケースでパフォーマンスを改善するかどうかはテストする必要があります。多くの場合は、データの構造や使用しているデータソース（たとえばフラットファイルか、RDBMS か、データ抽出か）に依存しているものです。
- 抽出を使うと、手早く簡単にほとんどのワークブックをより速く実行することができます。
- データがクリーンであるほど、質問の構造によくマッチし（すなわち準備や操作が少なくて済む）、ワークブックの実行が早くなります。
- 動作の遅いダッシュボードは、多くの場合、設計不良が原因となっています。特に、1 枚のダッシュボード上のチャートや一度に表示するデータが多すぎる場合です。シンプルさを心がけましょう。何もかも表示してからフィルターをかけるよりは、ユーザーが徐々に詳細をドリルダウンできるような作りにしましょう。
- 参照するフィールドにおいても返すレコードの粒度についても、必要なデータだけを扱いましょう。これによって、Tableau が生成するクエリーはより少なく、より適切に、より速くなり、データソースから Tableau のエンジンに移動させるデータ量を減らすことが可能になります。ワークブックのサイズも軽くなるので、共有しやすくなり早く開けます。
- データ量を減らすときは、フィルターを効果的に使いましょう。
- 文字列と日付の処理速度は遅く、数値とブールは速く処理されます。

最後に、扱うデータセットが大きい、あるいは複雑な場合、このドキュメントの推奨事項を実行してもあまり効果のない場合があります。サイズが大きい、あるいは複雑であるとはどういうことか。
それは条件によりますが、自分のデータがいつ大きくなるか分からないため、どのワークブックでもこの推奨事項に従っておいて間違いはないでしょう。習うより慣れろです。

目次

はじめに

Tableau の得意分野

Tableau Software は、ユーザーがデータを見て操作し、理解する際の、データの見せ方や操作方法を変えようとしています。そのため、従来のエンタープライズ版 BI プラットフォームのような機能や使用感をお届けしようとは思っていません。Tableau は、次のようなワークブックを作成する場合に最高の力を発揮します。

- **視覚的なワークブック** – 人間が大量で複雑なデータを理解するために最も効果的な方法は、視覚的表現を用いることです。これを裏付ける証拠は多数あります。Tableau は、チャート、図、ダッシュボードなどを使ってデータを提示するよう既定で設定されています。表やクロス集計もサポートされており、それぞれの役割があります。これらの最適な使用法は後で詳しくご紹介します。
- **インタラクティブに使えるワークブック** – Tableau ドキュメントは、デスクトップ、Web、モバイルのいずれでもインタラクティブに使えるように設計されています。印刷中心のアウトプット（紙ベース、PDF などのデジタル文書）の生成を主とする他の BI ツールとは異なり、Tableau が目指すのは、ユーザーが自分でデータを分析しビジネス上の問題を解決できる、豊富な機能やサービスを利用できるインタラクティブな体験を提供することです。
- **繰り返し分析できるワークブック** – 発見とは、本質的に反復のプロセスです。Tableau は、質問から発見へ、発見から質問へというサイクルをすばやく行えるよう設計されています。そのためユーザーは、仮定を立て、データを使って検証し、仮定を修正してもう一度検証するという作業をスピーディに行うことができます。
- **動作の速いワークブック** – これまでの BI プロセスは時間のかかるものでした。ソフトウェアをインストールして構成するにも、データを分析できるように準備するにも、また、ドキュメントやレポート、ダッシュボードなどを設計し実装するにも、すべてに時間がかかりました。しかし、Tableau ならこれまでよりずっと速くソフトをインストールし、サーバーに接続し、ドキュメントを作成できます。数週間から数か月かかっていた問題解決の作業を、数分から数時間程度に短縮できることも少なくありません。
- **シンプルなワークブック** – 従来のエンタープライズ版 BI ツールは、価格か、機能の複雑さのどちらかで、多くの場合一般のビジネスユーザーの能力を超えていました。多くの場合、ユーザーは、IT の専門家かパワーユーザーの手を借りないと思い通りのクエリやドキュメントを作成できませんでした。Tableau はハイテクに強くないユーザーでも直感的に操作できるインターフェイスを備え、データベースやスプレッドシートの専門家でなくても、クエリを実行したり複雑なデータを分析したりできるようになっています。
- **見た目のいいワークブック** –「美の基準は見る人次第」と言いますが、視覚的コミュニケーションには推奨されるベストプラクティスがあります。Tableau では表示形式などの機能により、一般ユーザーでも、効果的で理解しやすいチャートを、使用するデータに基づいて作成できるようになっています。
- **どこでも使用できるワークブック** – 1 つのプラットフォームを対象としたドキュメントはもう作成されなくなっています。ユーザーは、デスクトップ、Web、モバイルと、プラットフォームを問わずデータの表示や操作をする必要があり、データの状態も他のアプリケーションやドキュメントに埋め込まれているなど多岐にわたっています。Tableau では、1 種類のドキュメントをパブリッシュし、パブリッシュしたドキュメントをあらゆるプラットフォームで利用できます。その際、移植や再設計などの操作は必要ありません。

Tableau の不得意分野

Tableau は機能豊富でパワフルなツールですが、最適なソリューションを提供できない問題もあることを最初に理解していただくことが重要です。不可能なタスクがあるという意味ではありません。Tableau は、元の設計仕様に含まれていないタスクでも、工夫次第で実行することができます。しかし、Tableau の開発段階では想定されていなかったタスクもあり、そのようなタスクを何とかして Tableau で実行したとしても、努力が十分に報われない可能性もあります。結果として得られるソリューションも成果が上がらなかったり融通のきかないものになることがあります。

次のようなケースでは、要件を再検討するか、別のアプローチを検討することをお勧めします。

- 画面表示ではなく、紙への印刷を前提にデザインされたドキュメントが必要な場合。つまり、複雑なページレイアウトを管理する必要がある場合、ページ分けや章立て、ヘッダーやフッターのグループ化などの機能が必要な場合、正確な WYSIWYG フォーマットが必要な場合などです。Tableau でも複数ページのレポートを生成できますが、フォーマット管理については、印刷向けの専用レポート作成ツールにはかないません。
- カスタマイズされたドキュメントを複数の配信モードで送信する際に、プッシュ型配信（または「バースト配信」）する複雑なメカニズムが必要な場合。Tableau Server にはレポートのサブスクリプションというコンセプトがあり、ユーザーが自分で（Tableau 10 では他のユーザーも）サブスクライブすることができます。レポートをメールで受信できますが、より柔軟なソリューションを必要とされるお客様もいます。Tableau を使用してプッシュ型配信システムを構築することはできますが、Tableau の標準機能ではありません。そのためには、TABCMD ユーティリティに基づいて構築されるカスタムソリューションの開発、あるいは VizAlerts (http://tabsoft.co/1stldFh)、または Metric Insights の Push Intelligence for Tableau http://bit.ly/1HACxul のようなサードパーティー製ソリューションが必要です。
- 読み手の主な利用方法が、データを別のフォーマット（多くの場合 CSV ファイルや Excel ファイル）にエクスポートすることである場合。これは、詳細データを含む行が多数ある表タイプのレポートでよくあるケースです。誤解のないように言うと、Tableau でも、ビューやダッシュボードのデータを Excel にエクスポートできます。しかも、エクスポートのレベルをサマリーと詳細から選べます。しかし、主な利用方法がエクスポートである場合、これは Tableau の本来の用途でない ELT（抽出・加工・書き出し）プロセスに当たります。レポート作成ツールよりも、ずっと効率的にこの作業を行えるソリューションは多数あります。
- 多くの場合、複雑な小計機能や相互参照機能を含む既存のスプレッドシートレポートを反映している、非常に複雑な、クロス集計式のドキュメントが必要です。よくある例は損益計算書や貸借対照表などの財務諸表です。また、シナリオモデリング、what-if 分析、さらには仮定データの変更などが必要になることもあります。元になる粒度の高いデータがない場合や、レポートの論理が、レコードを合計値までロールアップするのでなく「セル参照」に基づく場合、この形式のレポートには引き続きスプレッドシートを使用するほうが適していることもあります。

効率的な作業を理解する

「効率的に作業できる」なワークブックとは

「効率的に作業できる」ワークブックにはいくつかの要素があります。それには、技術的な要素とユーザーに焦点を合わせた要素がありますが、一般的に見て、効率的に作業できるワークブックとは次のようなワークブックです。

- **シンプルである** – 容易に作成でき、その後も容易に維持することができるか。ビジュアル分析の本質を活かし、作成者とデータのメッセージが明確に伝わるようになっているか。
- **柔軟性がある** – ユーザーが聞きたい複数の質問に答えられるか、あるいは 1 つの質問にしか答えられないか。ユーザーがインタラクティブに操作することができるか、静的なレポートだけを提供するのか。
- **速い** – ユーザーが満足できる速度で応答するか。これは、ファイルを開く、または更新にかかる時間、操作に対する応答時間を意味することがあります。主観的な評価基準ではありますが、Tableau は、ファイルを最初に開くときやユーザーが操作したときに数秒で情報を表示できるワークブックを目指しています。

ダッシュボードのパフォーマンスは、次のようなものに影響を受けます。

- ダッシュボードとワークシートの両レベルでのビジュアルデザイン。たとえば表示する要素やデータポイントの数、フィルターやアクションの使用です。
- 計算。たとえば計算の種類、計算を行う場所などです。
- クエリー。たとえば返すデータの量、カスタム SQL かどうか、などです。
- データ接続と参照元データソース。
- Tableau Desktop と Tableau Server との違い。
- その他、ハードウェア構成や容量などの環境的要素。

効率性が重要な理由

それには、いくつかの理由があります。

- アナリストあるいはワークブック作成者が効率的に仕事をすると、答えを早く得られるようになります。
- 効率的に仕事ができれば、分析の「流れ」から逸脱することがありません。つまり、結果を得るするためにツールをどう扱うかよりも、質問とデータのことを考えていられるということです。
- デザインの柔軟なワークブックを作ると、似たような要件のために複数のワークブックを作って維持する必要性を減らせます。
- シンプルなデザインのワークブックを作ると、他のユーザーがそのワークブックを手にとり、それを元にして別バージョンを作ることが容易になります。
- 反応速度を認識しておくことは、エンドユーザーがレポートとダッシュボードを適切に表示できるようにするための重要な要素です。できる限り速く動作するワークブックを作るとユーザーの満足度が高まります。

これまでのさまざまな経験では、お客様が直面するパフォーマンスの問題は、そのほとんどがワークブックの設計ミスによるものです。このミスを教育によって直すか、教育によって最初の段階でミスを避けることができれば、問題は解決できます。

ボリュームの小さなデータを扱う場合、ここに記載している推奨事項の多くは重要ではありません。そのままのやり方で作業を進めてかまいません。しかし、億単位のレコードを扱うとなると、多数の

ワークブックや作成者に与えるワークブック設計不良の影響が増大するため、このホワイトペーパーにあるアドバイスを検討する必要があります。

もちろん、習うより慣れろですから、すべてのワークブックについてこのアドバイスに従うことを推奨します。予想される本番データボリュームでワークブックをテストするまでは、設計完了にはならないと覚えておいてください。

1 つ、大事なことを書いておきます。このドキュメント全体を通して Tableau Server に言及しますが、あなたがオンプレミスよりもホスティング型ソリューションの方を好む場合、このアドバイスはほとんど Tableau Online にも当てはまります。明らかな例外は、サーバーの構成パラメーターの微調整/調整や、サーバーティアでのインストール/アップデートに関する点です。SaaS では、これらの問題に対応しています。

物理の法則

ワークブックのパフォーマンスに影響する機能について技術的な詳細を論じる前に、効率的に作業できるダッシュボードやレポートを作成するうえで役立つ基本的な原理を説明します。

遅いデータソースは Tableau も遅くする

Tableau ワークブックが実行速度の遅いクエリを利用していれば、Tableau ワークブックも遅くなります。次のセクションで、データベースを調整する際のヒントをご紹介します。クエリの実行速度を改善するためにお役立てください。また、Tableau の高速データエンジンを使用してクエリのパフォーマンスを向上する方法についても説明します。

Tableau Desktop での動作が遅い場合は、(ほとんどの場合) Tableau Server でも遅くなる

Tableau Desktop で動作の遅いワークブックは、Tableau Server にパブリッシュしてもそれ以上速くはなりません。多くの場合ユーザーは、ワークブックの実行速度はローカル PC より CPU/RAM などの容量が大きなサーバー上で動く Tableau Server の方が速いと思っています。一般的に、ワークブックの実行速度は Tableau Server の方がやや遅めです。その理由は、

- 同時に複数のユーザーがワークブックを作成するときは、その全員がサーバーリソースを共有しているため (ただし、直感とは反対に、ワークブックを共有すると反応が速くなることに気付くかもしれません。これは Tableau Server 内でのキャッシュの仕組みによるものです)、そして
- クライアントワークステーションでなくサーバーが、ダッシュボードとチャートのレンダリングという追加処理を実行しなければならないためです。

まずは Tableau Desktop でワークブックの調整に取り組んでから、Tableau Server でのパフォーマンス調整を始めるようにしましょう。

このルールの例外は、Tableau Desktop がサーバーにはないリソースの制限を受けるときです。たとえば PC が分析したいデータボリュームに対応する十分な RAM を搭載していない場合や、サーバーがデータソースに接続する待ち時間が短い場合があります。低スペックで RAM が 2GB のワークステーションでデータセットを操作すると、動作が遅かったり、「メモリ不足」というエラーが発生することもあります。そのような場合は、ワークブックをパブリッシュすればサーバーのより大きなメモリや処理能力を利用できるため、許容できる速度までパフォーマンスが向上します。

新しいバージョンほど高機能

Tableau の開発チームは、ソフトウェアのパフォーマンスとユーザビリティの向上に絶えず取り組んでいます。Tableau Desktop と Tableau Server を最新バージョンにアップグレードすると、ワークブッ

クを変更しなくてもかなりのパフォーマンス改善を生み出すことがあります。たとえば、多くのお客様が、V8 から V9 にアップグレードしただけでワークブックのパフォーマンスが 3 倍（以上）改善したと報告しています。パフォーマンスの向上は、Tableau 10 およびそれ以降でも継続的に重視されています。もちろん、Tableau Online は最新バージョンのリリースに合わせて常時アップグレードされるため、問題ありません。

メンテナンスリリースやメジャー/マイナーリリースでも同様です。Tableau のリリースサイクルの詳細な情報と各リリースの個別内容については、リリースノートページでご紹介しています。

http://www.tableau.com/ja-jp/support/releases

さらに、データベースベンダーも製品の向上に取り組んでいます。次の Web ページのリストで、適切なデータソースドライバーの最新バージョンを使用していることを確認してください。

http://www.tableau.com/ja-jp/support/drivers

少ないほど効果的

何についても言えることですが、良いものでも度が過ぎればかえって害になります。たった 1 つのモノリシックなワークブックにすべてを詰め込もうとしてはいけません。1 つの Tableau ワークブックにはダッシュボードが 50 件入り、各ダッシュボードにはチャートオブジェクトが 20 個入り、各オブジェクトはそれぞれ 50 件の異なるデータソースから情報を取得することができます。しかし、処理速度は確実に遅くなります。

作成したワークブックがどうしてもそのようになってしまうという場合は、ファイルをいくつかに分けることを検討してください。難しいことではありません。ワークブック間でダッシュボードをコピーするだけで、関連するワークシートとデータソースは全部 Tableau が持ってきてくれます。ダッシュボードが必要以上に複雑なときは、単純化して、エンドユーザーをレポートからレポートへと誘導する操作を加えることも考えてみてください。ドキュメントによって Tableau の価格が決定されるわけではありません。データを自由に分散させてください

スケーラビリティはパフォーマンスと同じではない

スケーラビリティとは、複数のユーザーによる共有ワークブックの表示をサポートできることです。パフォーマンスとは、個々のワークブックが可能な限り早く実行できるようにすることです。このドキュメントにある推奨事項の多くは、Tableau Server にパブリッシュされたワークブックにスケーラビリティの向上をもたらしますが、このドキュメントで主に焦点を当てているのはパフォーマンスの向上です。

推奨ツール

自分のワークブックのパフォーマンスを理解するためには、a) 何が起こっているかと、b) それにかかる時間を理解する必要があります。この情報は、ワークブックの実行場所 (すなわち Tableau Desktop または Tableau Server) に応じて、複数の場所、および複数の詳細レベルで取得できます。このセクションでは、利用可能な様々なオプションについて説明します。

パフォーマンスレコーダー

パフォーマンス情報として最初に確認するものは、Tableau Desktop と Tableau Server のパフォーマンスレコーダー機能です。Tableau Desktop では、[ヘルプ] メニュー内にこの機能があります。

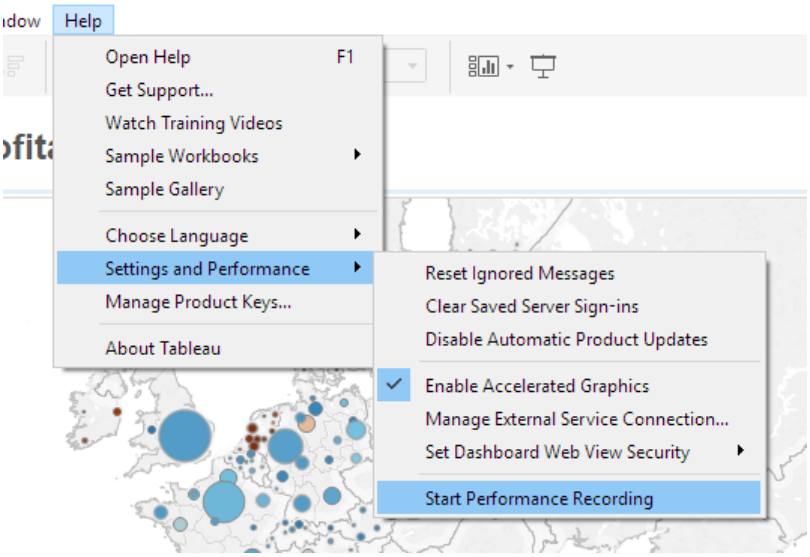
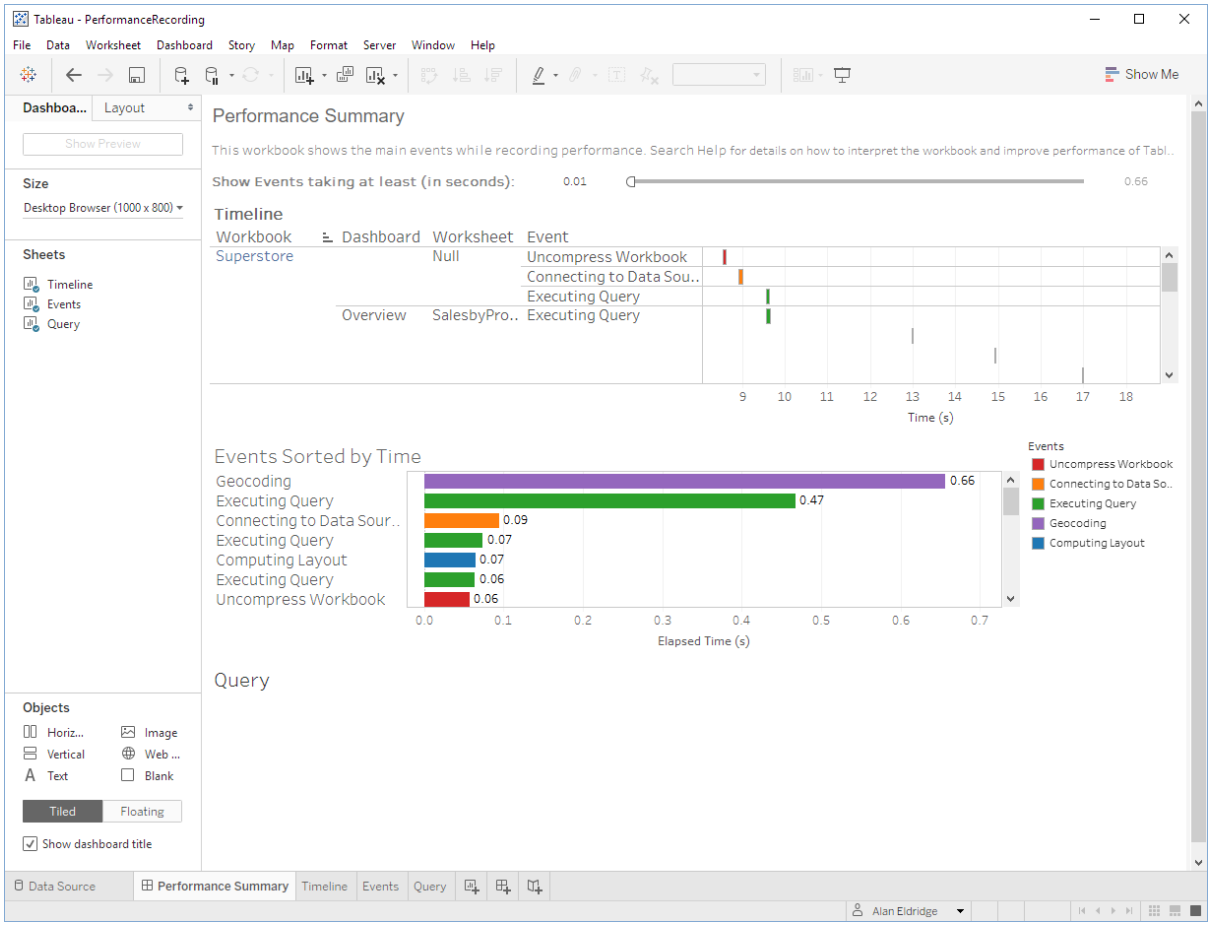


Tableau を起動してパフォーマンスの記録を開始してから、ワークブックを開きます (この間、別のワークブックは開かないことをお勧めします。気付かないうちにリソースを競合させないためです)。エンドユーザーになったつもりでワークブックを操作し、十分なデータを収集したと感じたところで [ヘルプ] メニューに戻り、記録を停止します。Tableau Desktop で、データが記録された別のウィンドウが開きます。



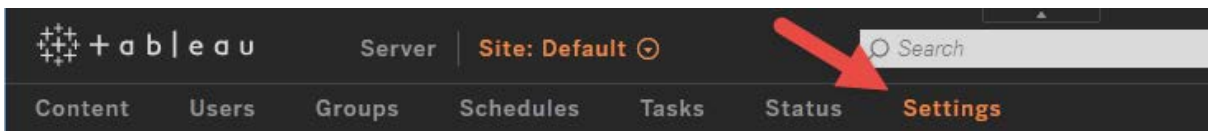
これで、ワークブックで最も時間のかかるアクションを特定できるようになりました。たとえば上図の例では、Timeline ワークシートから選択されたクエリは完了するまでに 30.66 秒もかかっています。棒グラフをクリックすると、実行されているクエリのテキストが表示されます。パフォーマンスレコーダーは Tableau ワークブックに結果を出力するので、追加のビューを作成してこの情報を別の方法で調べることができます。

> メモ: 既定では、0.1 秒未満のイベントは表示されません。ダッシュボードの上部にあるフィルターを調節すれば表示することもできますが、処理時間の長いタスクに注意を払う必要があります。1 秒に設定することをお勧めします。

パフォーマンスの記録を Tableau Server 上に作成し、ワークブックがパブリッシュされるときに起こる問題を特定させることも可能です。既定では、パフォーマンスの記録は Tableau Server 上では有効化されていません。これはサイト単位で管理される機能です。

サーバー管理者は、パフォーマンスの記録をサイトごとに有効化することができます。

1. パフォーマンスの記録を有効化したいサイトに移動します。
2. [**設定**] をクリックします。



3. [ワークブックパフォーマンスメトリックス] 内の [**ワークブックパフォーマンスメトリックスを記録**] を選択します。

4. [**保存**] をクリックします。

パフォーマンスの記録を作成する:

1. パフォーマンスを記録したいビューを開きます。ビューを開くとき、Tableau Server が URL の後に ":iid=<*n*>" を追加します。これがセッション ID です。たとえば次のようなものです。

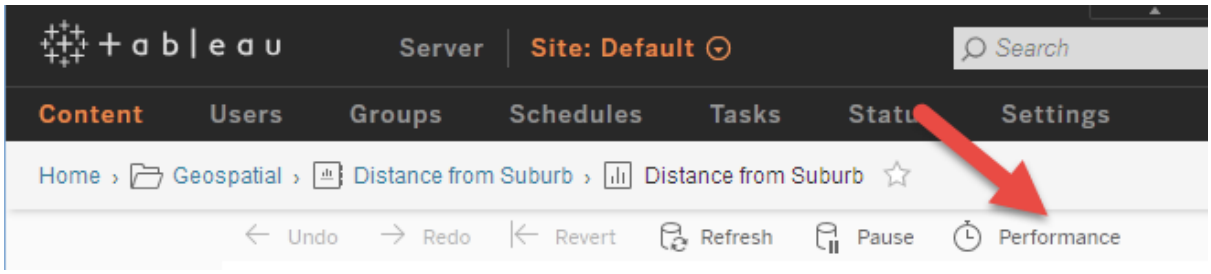
   ```
   http://<tableau_server>/#/views/Coffee_Sales2013/USSalesMarginsByAreaCode?:iid=1
   ```

2. ビューの URL の最後の部分の、セッション ID の直前に :record_performance=yes& と入力します。たとえば次のようになります。

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

3. ビューを読み込みます。

パフォーマンスの記録が開始されたことが、ビューのツールバー内に [**パフォーマンス**] オプションとして画面表示されます。



完了し、パフォーマンスの記録を表示する:

1. [**パフォーマンス**] をクリックしてパフォーマンスワークブックを開きます。これはパフォーマンスデータの最新のスナップショットです。ビューの分析を続行しながら追加でスナップショットを取ることも可能で、パフォーマンスデータは累積されます。

2. 別のページに移動するか、:record_performance=yes を URL から削除して記録を停止します。

この情報を使用して、ワークブック内で最優先で検証するべき箇所、つまり改善に費やす時間に対して最も成果が上がるセクションを特定します。これらの記録を分析するための詳細情報については、以下のリンクをご覧ください。

https://onlinehelp.tableau.com/current/pro/desktop/ja-jp/help.htm#perf_record_create_desktop.html

ログ

Tableau は、ログファイルで完全なクエリテキストを見ることができます。既定の場所は C:\Users\<username>\Documents\My Tableau Repository\Logs\log.txt です。ファイルはかなり冗長なもので、JSON エンコードのテキストで書かれています。そのため、分析には Notepad++ や Sublime などのテキストエディターを推奨します。「begin-query」または「end-query」で検索すると、データソースにパスされたクエリの文字列が見つかります。「end-query」ログでも、クエリの実行にかかった時間と Tableau に返されたレコードの数を見ることができます。

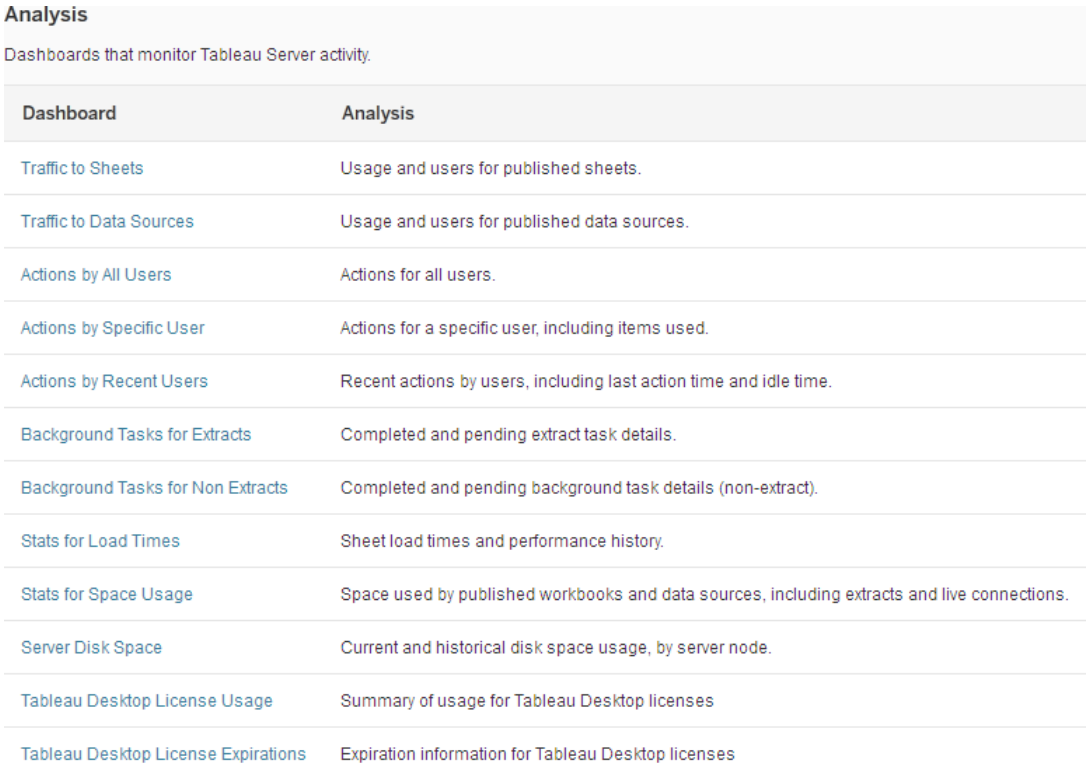
```
{"ts":"2015-05-24T12:25:41.226","pid":6460,"tid":"1674","sev":"info","req":"-","sess":"-
```

```
","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]\nFROM [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}
}
```


Tableau Server の場合は、ログの場所は `C:\ProgramData\Tableau\Tableau Server\data\tabsvc\vizqlserver\Logs` になります。

Tableau Server パフォーマンスビュー

Tableau Server には、管理者が Tableau Server でのアクティビティを監視するためのビューがいくつか用意されています。これらのビューは、サーバーの [メンテナンス] ページの [分析] 表にあります。



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

これらのビューについて詳しくは、以下のリンクをご覧ください。

http://onlinehelp.tableau.com/current/server/ja-jp/adminview.htm

また、カスタムの管理ビューは Tableau リポジトリの一部を構成する PostgreSQL データベースに接続することで作成できます。手順についてはこちらをご覧ください。

http://onlinehelp.tableau.com/current/server/ja-jp/adminview_postgres.htm

監視とテスト

TabMon

TabMon は、一定期間にわたりパフォーマンスの統計を収集できる Tableau Server 用オープンソースクラスタモニターです。TabMon はコミュニティのサポートを受け、完全なソースコードを MIT オープンソースライセンスに基づいて公開しています。

TabMon は、追加の設定なしにシステムの健全性とアプリケーションのメトリックスを記録することができます。ネットワーク全体の Tableau Server マシンで、Windows Perfmon、Java Health、Java Mbean (JMX) カウンターなど、ビルトインのメトリックスを収集します。TabMon では、物理 (CPU、RAM)、ネットワーク、ハードディスクの使用状況を監視できます。キャッシュ-ヒット率、リクエストの待ち時間、アクティブセッションなどがトラッキングできます。データは読みやすく統一された構造で表示され、Tableau Desktop でデータを簡単に可視化することができます。

TabMon を使えば、どのメトリックスをどのマシンから収集して監視するかを完全に管理できます。スクリプトやコーディングは不要で、マシン名とメトリックス名が分かっているだけで十分です。TabMon はリモートでもクラスタから独立させても実行でき、クラスタの健全性をネットワーク上のどのコンピューターからでも監視、集計、分析することができます。本番マシンに負荷が加わることはありません。

TabMon について、詳しくはこちらをご覧ください。

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

TabJolt

TabJolt は「ポイントアンドラン」で読み込みとパフォーマンスをテストするツールで、特に Tableau Server で容易に使えるように設計されています。従来の読み込みテストツールと異なり、TabJolt は Tableau Server に対し自動的に読み込みを実行させます。スクリプトの開発やメンテナンスは不要です。TabJolt は Tableau のプレゼンテーションモデルとして認識されているため、ビジュアライゼーションを自動で読み込み、テストの実行中に可能な操作を解釈することができます。

これによって、TabJolt をサーバー上の 1 つまたは複数のワークブックにポイントし、自動的に読み込み Tableau のビュー上で操作を実行することが可能です。TabJolt はまた、主なメトリックスとして平均応答時間、スループット、95 百分位の応答時間などを収集し、Windows のパフォーマンスメトリックスを相関関係確認用として採取します。

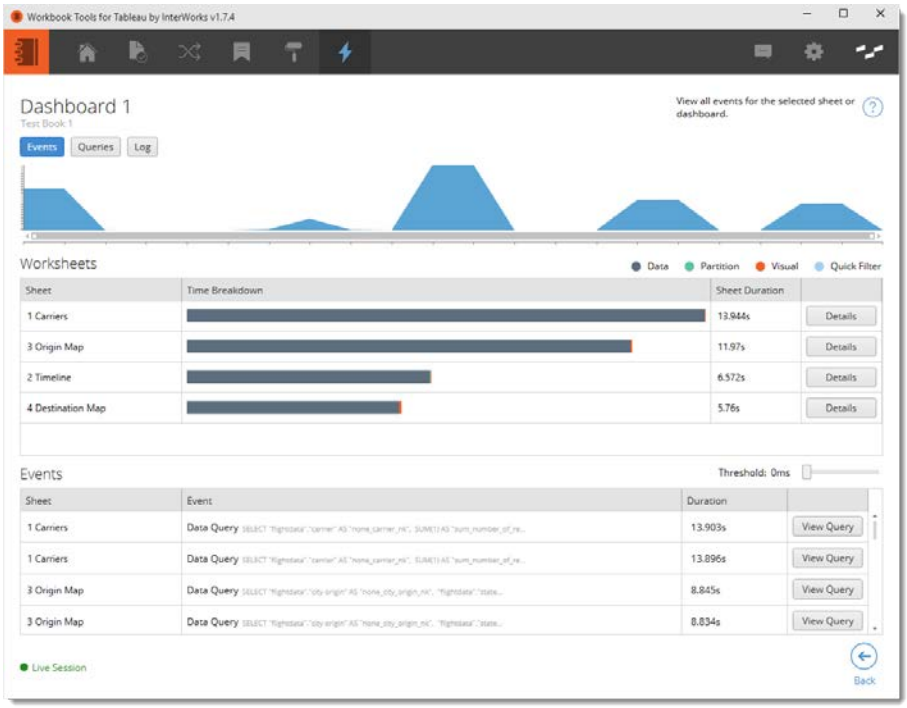
もちろん TabJolt があっても、ユーザーには Tableau Server のアーキテクチャに関する十分な知識が必要です。読み込みテストにおいて、Tableau Server をブラックボックス的に扱うことは推奨できません。期待に沿った成果を生み出す可能性が低くなります。TabJolt は自動化されたツールで、人間による様々な操作を簡単に再生できないため、TabJolt の出す結果が実環境での効果を反映しているか慎重に検討してください。

TabJolt について、詳しくはこちらをご覧ください。

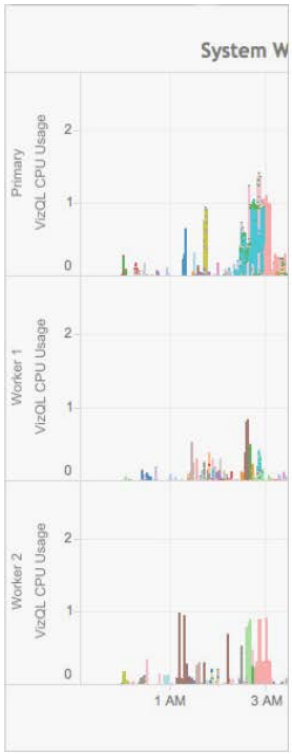
https://github.com/tableau/tabjolt/releases

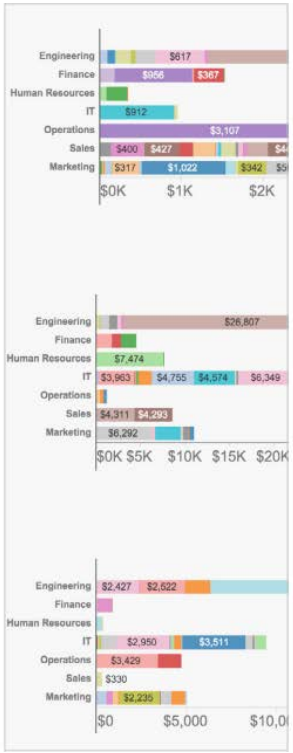
その他のツール

その他にも利用可能なサードパーティー製ツールがいくつかあり、ワークブックのパフォーマンスの特徴を見出すことができます。その 1 つが Interworks の「Power Tools for Tableau」です。これにはパフォーマンスアナライザー（ビルトインのパフォーマンスレコーダーに類似）が含まれており、どのシートとクエリに一番長く時間がかかっているかを掘り下げて理解することができます。



Palette Software には Palette Insight という製品も含まれています。これは Tableau Server からパフォーマンス情報を取得することで、キャパシティプランニング、リソースを多く使うユーザーやワークブックの特定、ユーザーアクセスの監査、チャージバックモデルの構築を可能にします。





また、現在の DBMS プラットフォームはほとんど、クエリ実行の追跡と分析ができる管理ツールを備えています。パフォーマンスの記録でクエリの実行時間が大きな問題点だと分かった場合は、お使いの DBA も大いに役立つ可能性があります。

クライアントのブラウザとサーバー間の通信に問題があると思う場合、Telerik Fiddler などのツールやブラウザのディベロッパーツールでクライアントとサーバー間のトラフィックをよく調べてみてもいいでしょう。

ワークブックの設計を再検討する

多くのユーザーにとって、Tableau を使っての作業は新しい経験です。効率的に作業できるワークブックを作成するために、学んでおくべき設計の手法やベストプラクティスがあります。それでも、慣れないユーザーの多くが従来の設計手法を Tableau でも使用してしまい、残念な結果を招いていることが分かっています。このセクションでは、ベストプラクティスを反映した設計の原則について説明することを目的とします。

優れたダッシュボード設計

Tableau を使って、エンドユーザーのためにインタラクティブな体験を生み出します。Tableau Server により提供される完成品は、データをただ見るだけでなく、ユーザー自身がデータを分析できるインタラクティブなアプリケーションです。そこで、効果的な Tableau ダッシュボードを作成するためには、静的なレポートを作成するときのような考え方を改める必要があります。

以下は、Tableau を使い始めたばかりのユーザーの多くが作成しがちなダッシュボードの例です。特に、Excel や Access などのツールを使っていた人や、「従来の」レポート作成ツールを使った経験がある人は、ここで紹介する間違いをしてしまいます。それは「何もかも」詰め込んだ表タイプのレポートで、何種類ものフィルターが用意されているのが特徴です。ユーザーはこのフィルターを使って、自分が見たい数件のレコードが表示されるよう表を絞り込むことができます。

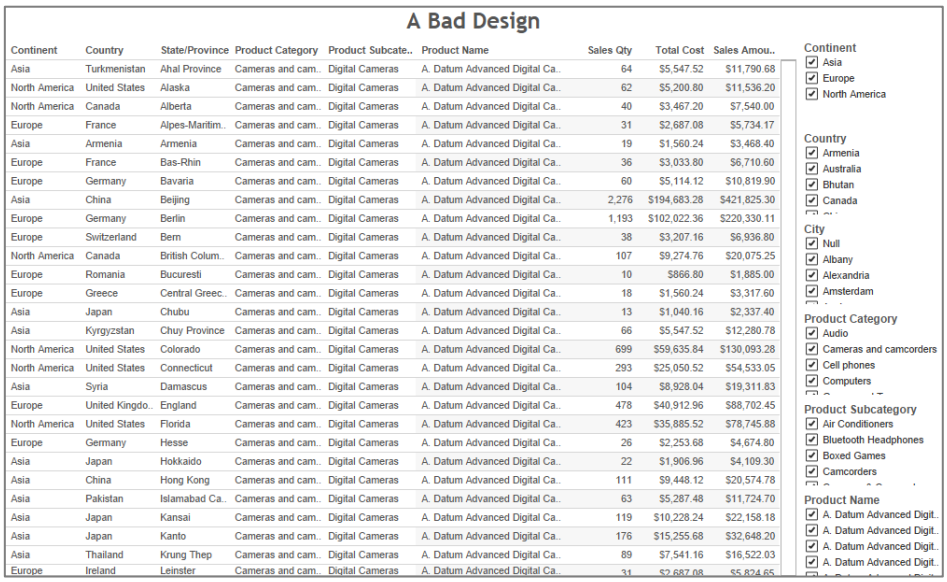
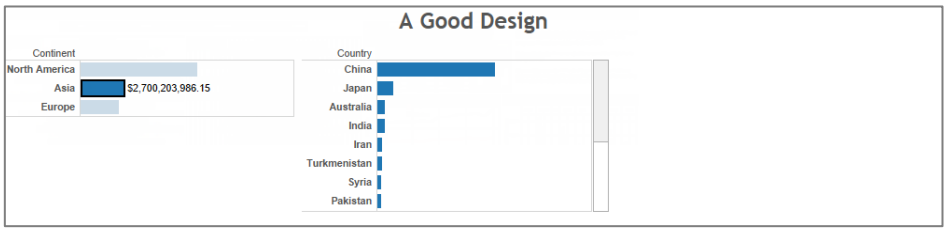


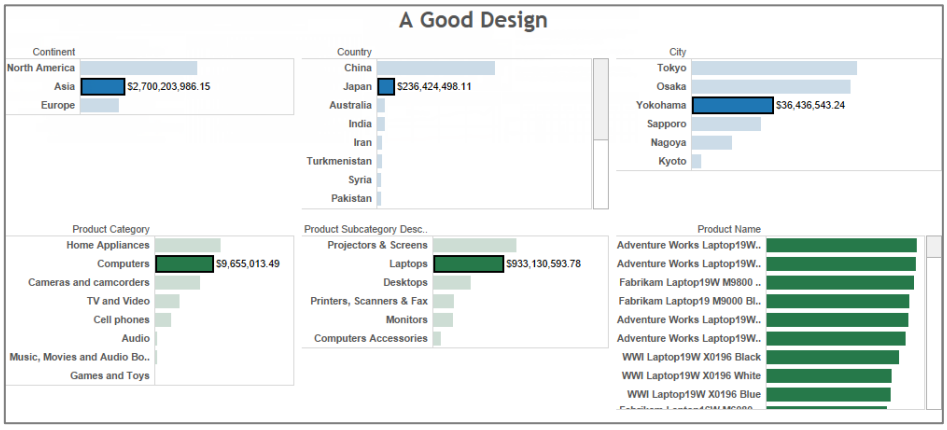
Tableau ダッシュボードでは、これは悪い例です（実際、まったく「良い」例ではありません）。厳しいことを言えば、これは見せかけのデータ抽出プロセスです。データを Excel など別のツールに移して詳細な分析やチャート作成をしたい、とユーザーに思わせるからです。好意的に解釈すれば、エンドユーザーがどのようにデータを分析したいか実際には分からないため、「最初の基準に基づいてすべての情報を用意しました。さらに必要なデータを見つけるために結果セットを絞り込むことができるようにフィルターオブジェクトもあります」というアプローチを取っています。

以下は設計し直したダッシュボードです。まったく同じデータを使っています。ここでは、集計のレベルを最も高くしてあります。

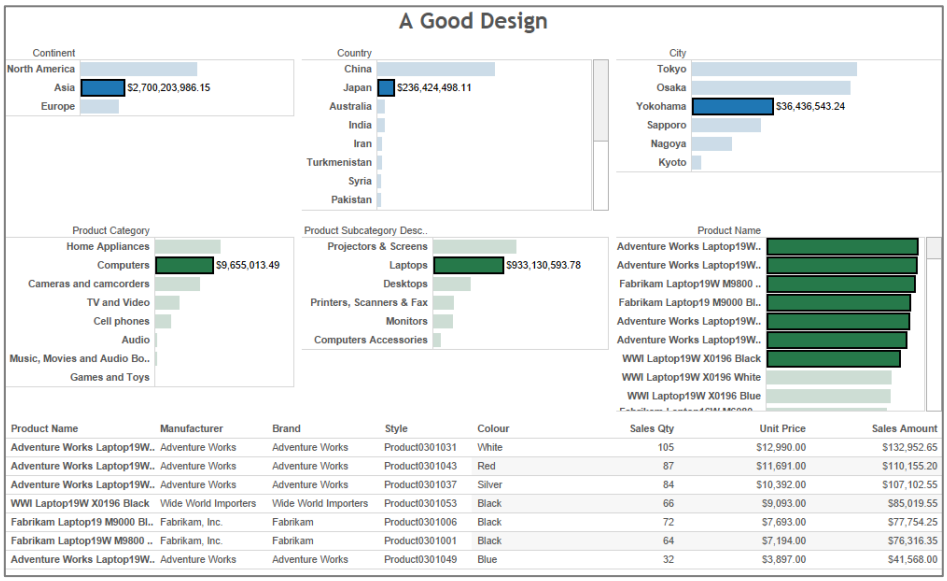
1 つ以上の要素を選択すると、次のレベルの詳細が表示されます。



これを繰り返すたびに、より詳しいデータが明らかになっていきます。



最終レベルまで表示すると、最初にご紹介したクロス集計ダッシュボードと同じデータになります。



Product Name	Manufacturer	Brand	Style	Colour	Sales Qty	Unit Price	Sales Amount
Adventure Works Laptop19W..	Adventure Works	Adventure Works	Product0301031	White	105	$12,990.00	$132,952.65
Adventure Works Laptop19W..	Adventure Works	Adventure Works	Product0301043	Red	87	$11,691.00	$110,155.20
Adventure Works Laptop19W..	Adventure Works	Adventure Works	Product0301037	Silver	84	$10,392.00	$107,102.55
WWI Laptop19W X0196 Black	Wide World Importers	Wide World Importers	Product0301053	Black	66	$9,093.00	$85,019.55
Fabrikam Laptop19 M9000 Bl..	Fabrikam, Inc.	Fabrikam	Product0301006	Black	72	$7,693.00	$77,754.25
Fabrikam Laptop19W M9800 ..	Fabrikam, Inc.	Fabrikam	Product0301001	Black	64	$7,194.00	$76,316.35
Adventure Works Laptop19W..	Adventure Works	Adventure Works	Product0301049	Blue	32	$3,897.00	$41,568.00

データをどう提示するかを考えないようにしてください（この点は重要ですが、後で取り上げます）。代わりに、このダッシュボードを使用した場合の操作性について考えます。レイアウトが左から右へ、上から下へと自然に移動していることに注目してください。この例では元になるデータ量が膨大ですが、ダッシュボードの役目は、エンドユーザーが徐々にデータを掘り下げ、自分が探している詳細なレコードにたどり着けるよう誘導することです。

ここで紹介した 2 つの例の最も重要な違いは、分析プロセスを通じてエンドユーザーをどのように誘導しているかです。最初の例では、ユーザーが求めている可能性のあるレコードをすべて広範囲

に表示することから始め、エンドユーザーが自分でフィルターを適用して、表示されるレコード数を減らすようにします。この手法はそれ自体に問題を含んでいます。

- 1 つ目の問題は、エンドユーザーにデータを表示する前に実行する必要のある最初のクエリが、本質的に最大のクエリ（「すべてのレコードを取得」）になることです。実際のデータセットでこれを行うと、クエリを実行して Tableau エンジンにデータが届くまでに、かなりの時間がかかってしまいます。ソリューションに対するエンドユーザーの印象を決めるうえで、第一印象は非常に重要です。起動して数秒以内に何も起こらないとなれば、印象が悪くなります。
- 2 つ目の問題は、数十万から数百万のマーク（クロス集計の各セルを「マーク」と言います）を含むビューを作成するために、かなり高性能の CPU とメモリが必要になることです。それにも時間がかかるため、システムの応答時間についての印象がさらに悪くなります。Tableau Server で複数のユーザーがサイズの大きいクロス集計を作成すると、パフォーマンスが低下し、最悪の場合にはメモリ不足が起こることも考えられます。その結果サーバーの不安定やエラーを引き起こし、あらゆる面でエンドユーザーの使い心地が悪くなる可能性があります。もちろん、サーバーにメモリを追加してこの可能性を低く抑えることはできますが、それは対処療法であって根本的な解決にはなりません。
- 3 つ目の問題は、最初に適用するフィルターの絞り込み範囲が広いか狭いかについて、コンテキストに基づく指針がユーザーに与えられていないことです。最初のクエリで数万件ものレコードが返されてサーバーの RAM を使い尽くしてしまわないよう、レポートのユーザーが利用できるカテゴリーをすべて確認できているかどうかを知るには、どうすればよいのでしょう。それを知るには、手間のかかる方法以外ありません。

対照的に、設計し直した例では、最初のクエリで最も高位の集計データのみをリクエストします。

- 実行する必要がある最初のクエリが非常に集計されているため、返されるデータもほんの数件のレコードに抑えられます。優れた設計のデータベースを使用する場合、これは非常に効率的です。第一印象を形成する応答時間も非常に短縮され、システムに対する印象も良くなります。データを掘り下げるたびに実行される他のクエリも、高レベルでの選択内容によって集計され制約されています。そのため、クエリを実行して Tableau エンジンにデータが返されるまでの時間は速いままです。
- ダッシュボードを完成したときにはより多くのビューが表示されますが、各ビューに含まれるマークの数は数十個に抑えられています。それぞれのビューを生成するのに必要なリソースは、システム上で多数のエンドユーザーがアクティブな場合でも非常に少なく、メモリ不足が発生する可能性は低くなります。
- 最後に、ナビゲーションの高位のレベルでは、販売量がカテゴリーごとに示されています。これにより、選択する範囲に含まれるレコードが多いか少ないかを判断するコンテキストが、ユーザーに提供されます。また、色別表示により、各カテゴリーの収益性も分かるようになっています。やみくもにナビゲートするのではなく、どのエリアに注目する必要があるかが分かるため、ユーザーにとってデータがとても身近になります。

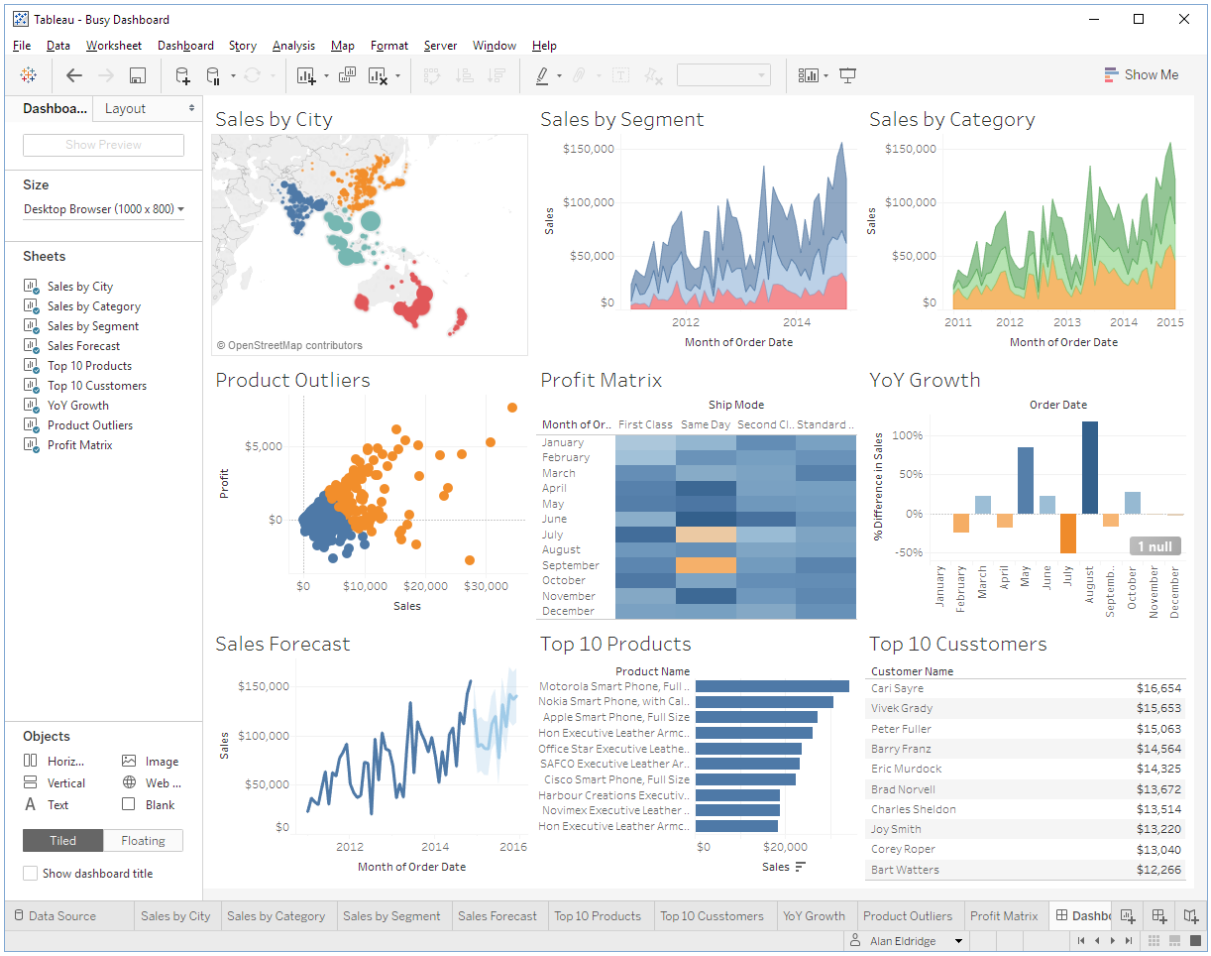
あくまでシンプルに

初心者のユーザーが犯しがちな間違いは、必要以上に「複雑な」ダッシュボードを作ることです。以前使っていたドキュメントを別のツールから再作成しようとしているのかもしれないし、レポートを印刷するために特別に設計したものを作ろうとしているのかもしれません。その結果、パフォーマンスが遅く効率の悪いワークブックができてしまいます。

複雑なダッシュボードの原因には、次のようなものがあります。

1 枚のダッシュボードにワークシートを多く作り過ぎる

初心者が陥りやすい間違いに、チャートやワークシートを 1 枚のダッシュボードに多く詰め込みすぎることがあります。



各ワークシートはデータソースに対し 1 つ (おそらく複数) のクエリーとして実行されることに注意してください。つまり、ワークシートが多いほどダッシュボードをレンダリングする時間がかかるということです。Tableau は、エンドユーザーにインタラクティブなダッシュボードを提供するよう設計されているため、複数のダッシュボードやページにわたってデータを分散してください。

フィルターカードが多過ぎる

フィルターカードは Tableau の非常にパワフルな機能で、エンドユーザー向けに情報が豊富でインタラクティブなダッシュボードを作成することができます。しかし、各フィルターはオプションを列挙するためにクエリを必要とすることがあるため、ダッシュボードに追加し過ぎるとダッシュボードのレンダリングに予想以上に時間がかかってしまいます。また、フィルターで [関連する値を表示] を使うと、他のフィルターが変更されるたびに、表示されている値を更新するためのクエリを必要とします。この機能は控えめに使いましょう。

また、複数のワークシートに適用しているフィルターがある場合は、表示されているすべての対象ワークシートが表示を更新するので、1 つ変更するたびに複数のクエリがトリガーされることに注意してください (画面にないワークシートは更新されません)。これが終わるまで何秒もかかると、操作性が低下します。ユーザーが複数選択タイプのフィルターを何度も変更することが予想される場合は、ユーザーが選択項目の変更が終わってから更新をトリガーできるように、[適用] ボタンを表示することを考慮してください。

パフォーマンス向上のためにダッシュボードを調整する

ダッシュボードをできるだけシンプルにしたら、キャッシュ機能を利用して設計を調整し、さらにパフォーマンスを向上できます。

固定サイズのダッシュボード

パフォーマンスを向上するための一番簡単な方法は、ダッシュボードが固定サイズであるかを確認することです。

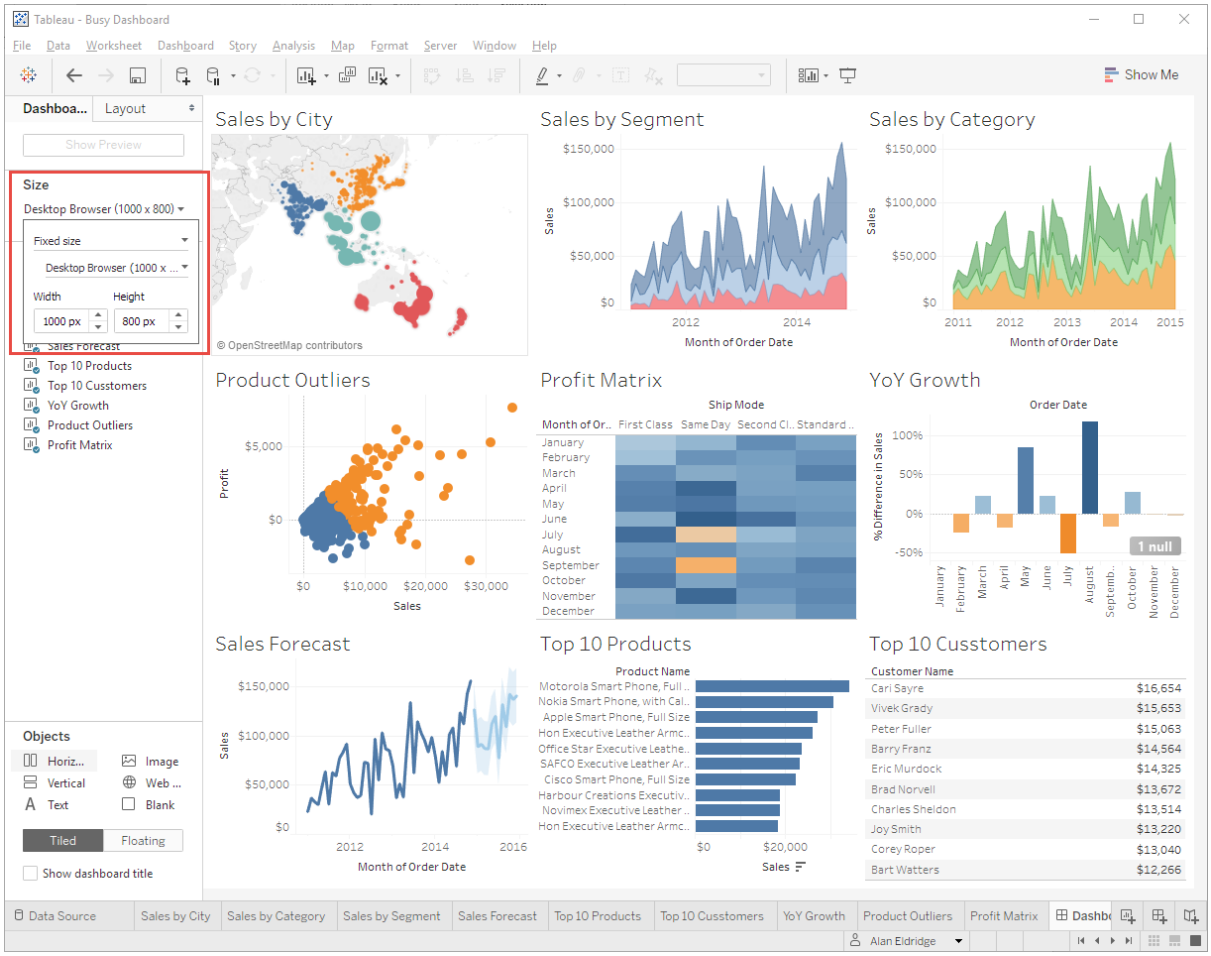


Tableau では、レンダリングプロセスの一部をレイアウト作成が占めています。たとえば、スモールマルチプルとクロス集計に表示する行と列の数、描画する軸目盛りやグリッド線の数と間隔、表示するマークラベルの数と位置などです。これはダッシュボードを表示するウィンドウのサイズによって決まります。

同じダッシュボードにサイズの異なるウィンドウから複数のリクエストを行う場合は、リクエストごとにレイアウトを生成する必要があります。・ダッシュボードレイアウトを固定サイズに設定することにより、どのリクエストにも再利用できるレイアウトを 1 つ作成するだけで済みます。サーバー側レンダリングを行う場合は、これがさらに重要になってきます。ダッシュボードを固定サイズにすると、サーバーでレンダリングされるビットマップをキャッシュして共有できるようになるため、パフォーマンスとスケーラビリティの両方が向上します。

デバイス固有のダッシュボード

Tableau 10 では、デバイス固有のダッシュボードという新しい機能を取り入れています。この機能により、使用しているデバイスに基づいて自動的に選択されるカスタムダッシュボードレイアウトを作成することができます。

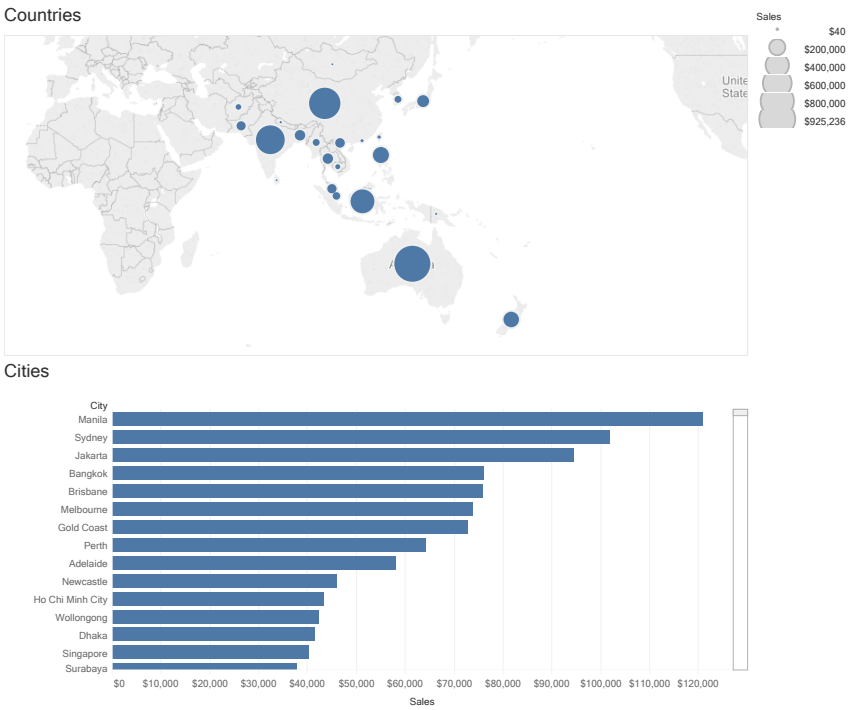
スクリーンサイズに基づいて、使用するレイアウトが選択されます。

- 最小の軸が 500px 以下 – スマートフォン
- 最小の軸が 800px 以下 – タブレット
- 800px 超 – デスクトップ

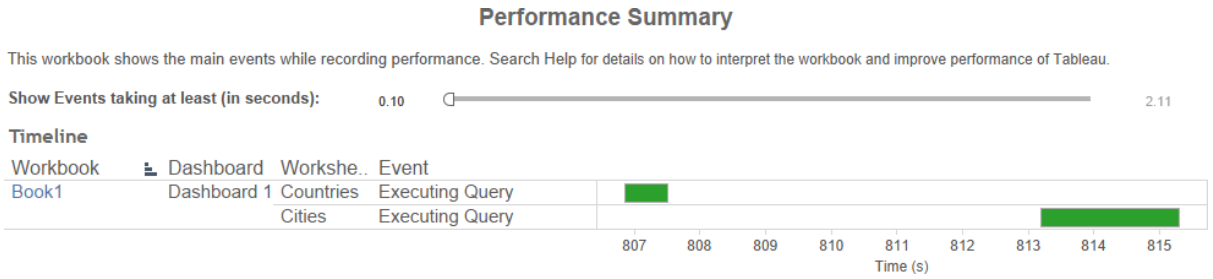
デバイスが変わるとスクリーンサイズもその範囲で変化します。デバイスは縦横に向きを変えることがあることから、スマートフォン/タブレットのレイアウトは基本的に自動サイズ調整に設定することをお勧めします。こうすることで、デバイスに関係なく最も見やすく表示できますが、これによってキャッシュの再利用 (サーバー側レンダリングの場合は、プレゼンテーションモデルのキャッシュとイメージタイルのキャッシュ両方)–に影響が及びます。一般的にはデバイスに合わせたサイズ調整のメリットの方が、キャッシュに及ぶ影響を上回りますが、これについては検討する必要があります。ユーザーがワークブックを使用し始めたら、最終的に最も一般的なスクリーンサイズ用にモデルとビットマップを設定するとパフォーマンスが向上することに注意してください。

Viz LOD を使用してクエリを減らす

基本的には各ワークシートに必要なフィールドだけ使用することを推奨しますが、1 枚のワークシートに取り込む情報を増やして他のワークシートへのクエリを避けることにより、パフォーマンスを向上させることも可能です。下のダッシュボードを見てください。



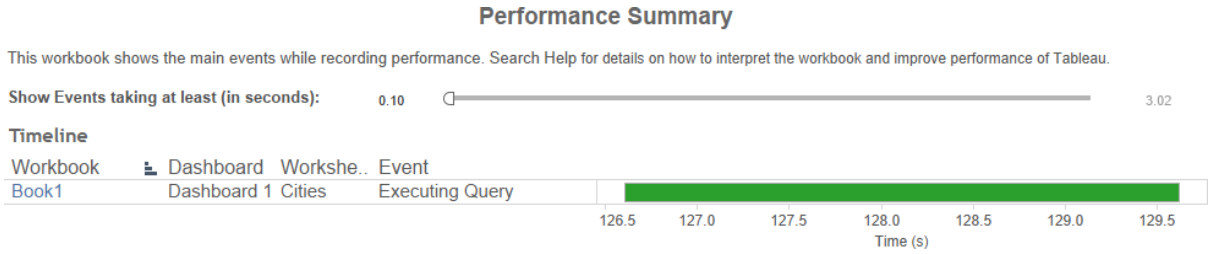
予想されるとおりにこれを作成すると、実行する計画によって各ワークシートに 1 つずつ、合わせて 2 つのクエリが実行されることになります。




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

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]
```

ダッシュボードの設計を変更して、([詳細] シェルフで) [Cities] ワークシートに [Country] を追加すると、Tableau はたった 1 つのクエリでダッシュボードを完成できます。Tableau は、[Cities] ワークシートのクエリをまず実行してから、そのクエリの結果をキャッシュして [Country] ワークシートに提供するスマートな機能を備えています。この機能を「クエリバッチ処理」といいます。

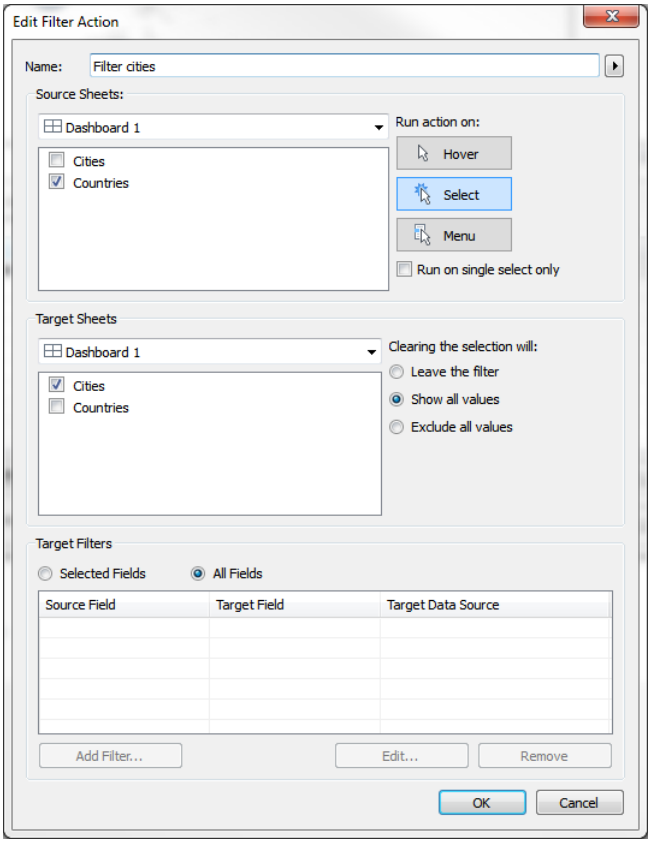



```
SELECT [Superstore APAC].[City] AS [City],
  [Superstore APAC].[Country] AS [Country],
  SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
GROUP BY [Superstore APAC].[Cities],
  [Superstore APAC].[Country]
```

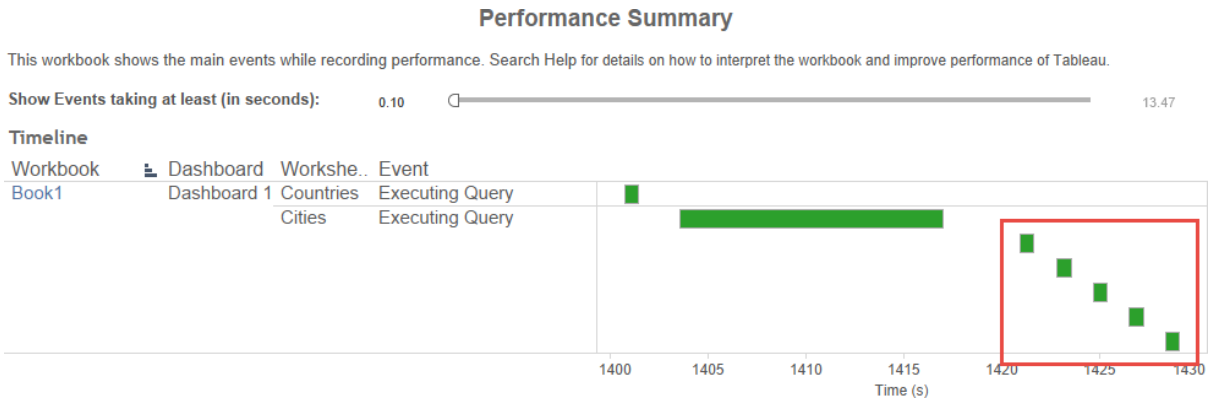
もちろん、すべてのケースでこれを行うことはできません。ディメンションを–viz に追加すると詳細レベルが変更されて、結果的に表示されるマークが増えるからです。ただしこれは、上の例のようにデータ内に階層リレーションシップがある場合は、表示される詳細レベルに影響がないため便利な手法です。

Viz LOD を使用してアクションを最適化する

アクションでも同様の手法を用いて、必要なクエリの数を減らすことができます。上の例の最適化する前のダッシュボードを使います。今度は [Country] ワークシートから [Cities] ワークシートにフィルターアクションを追加します。



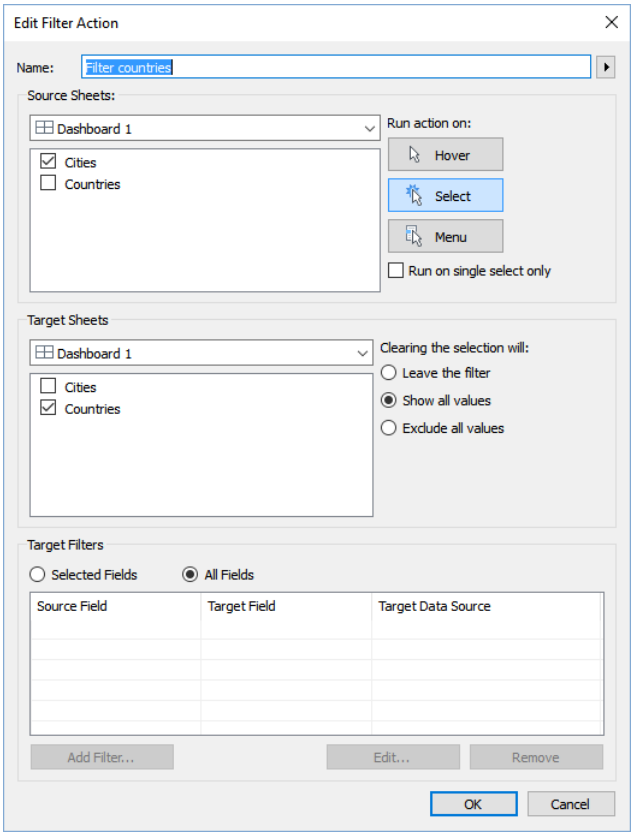
マップ上のマークをクリックすることでこのアクションをトリガーすると、Tableau はクエリを実行し [Cities] ワークシートの値を決定する必要があることがわかります。これは、[Cities] と [Country] のリレーションシップにあるクエリ結果のキャッシュにデータがないことが理由です。




```
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].[Cities]
```

[Country] を [Cities] ワークシートに追加すると、クエリ結果のキャッシュに十分な情報が入り、データソースに戻らなくてもこれらのフィルターを使うことができます。

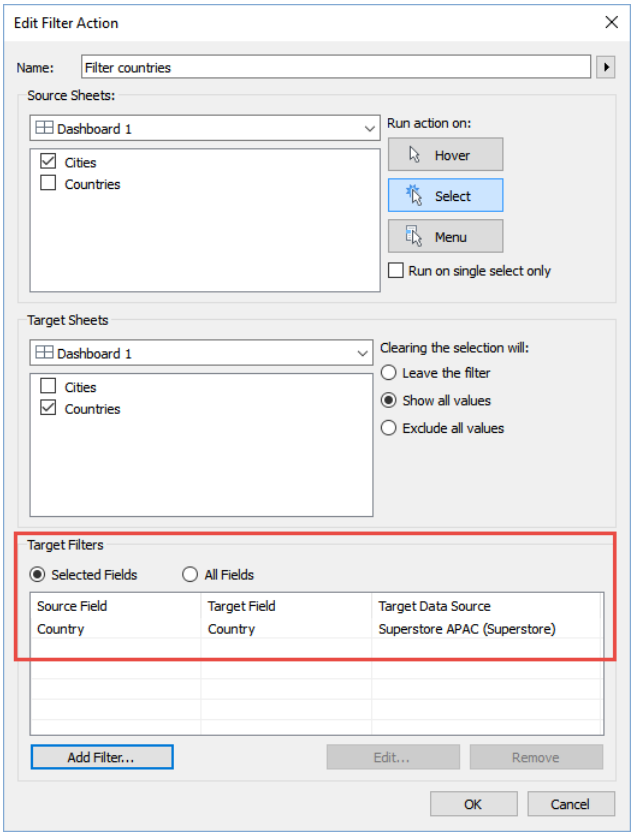
ソースワークシートがターゲットより詳細である状況でも、同様に最適化することができます。「すべてのフィールド」を使用してフィルターする、既定のアクション定義を使用する場合は、以下のようになります。



これにより、[Country] ワークシートのクエリ結果キャッシュから取得できない[Country] と [Cities] をフィルター句が参照するため、ワークブックはアクションごとにクエリを実行することになります。

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

アクションを [Country] のみでフィルターするように変更した場合、次のようになります。



今度はクエリ結果のキャッシュからフィルターをかけることができるため、データソースにクエリを戻す必要がありません。このように、詳細レベルの変更がワークシートの設計に影響する場合は考慮が必要です。影響しない場合でも、便利な手法となる可能性があります。

優れたワークシート設計

ダッシュボードから 1 レベル掘り下げて、ワークシートを見てみましょう。Tableau では本来、ワークシートの設計は実行されるクエリに関連しています。各ワークシートでは、1 つ または複数のクエリが生成されます。そこで、このレベルでは最適なクエリの生成を試みます。

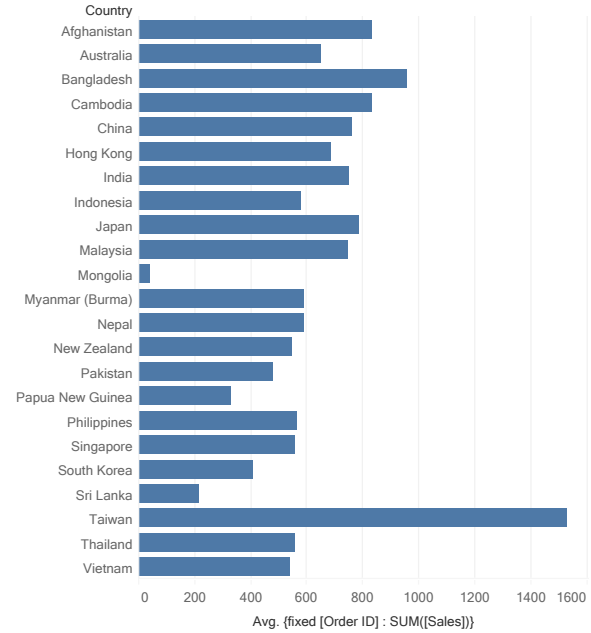
必要なフィールドのみを含める

[詳細] シェルフを確認して、viz で直接使用するフィールド、ツールヒントに必要なフィールド、要求されたマークの詳細レベルを出すためのフィールド以外はすべて削除します。これでデータソース内のクエリ実行が速くなり、クエリ結果に返すデータの量が少なくて済みます。すでに見てきたように、このルールには例外があります。クエリバッチ処理により、同様のクエリが他のワークシートで実行されないようにする場合などです。ただし、これらはあまり一般的ではなく、ワークシートのビジュアルの詳細レベルを変えない場合に限ります。

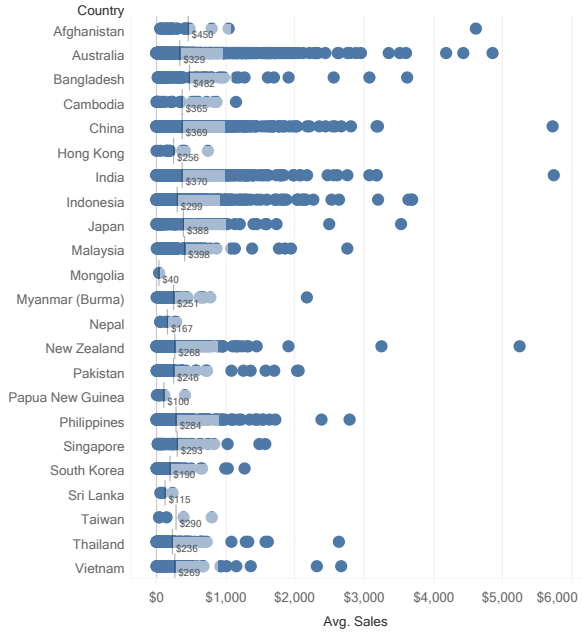
マークの数を最小限表示して質問に答える

Tableau では、同じ数値を計算する方法が複数あることがよくあります。下のダッシュボードを見てください。両方とも「国ごとの平均オーダーサイズは?」という質問に答えています。

Avg Order Size 1



Avg Order Size 2

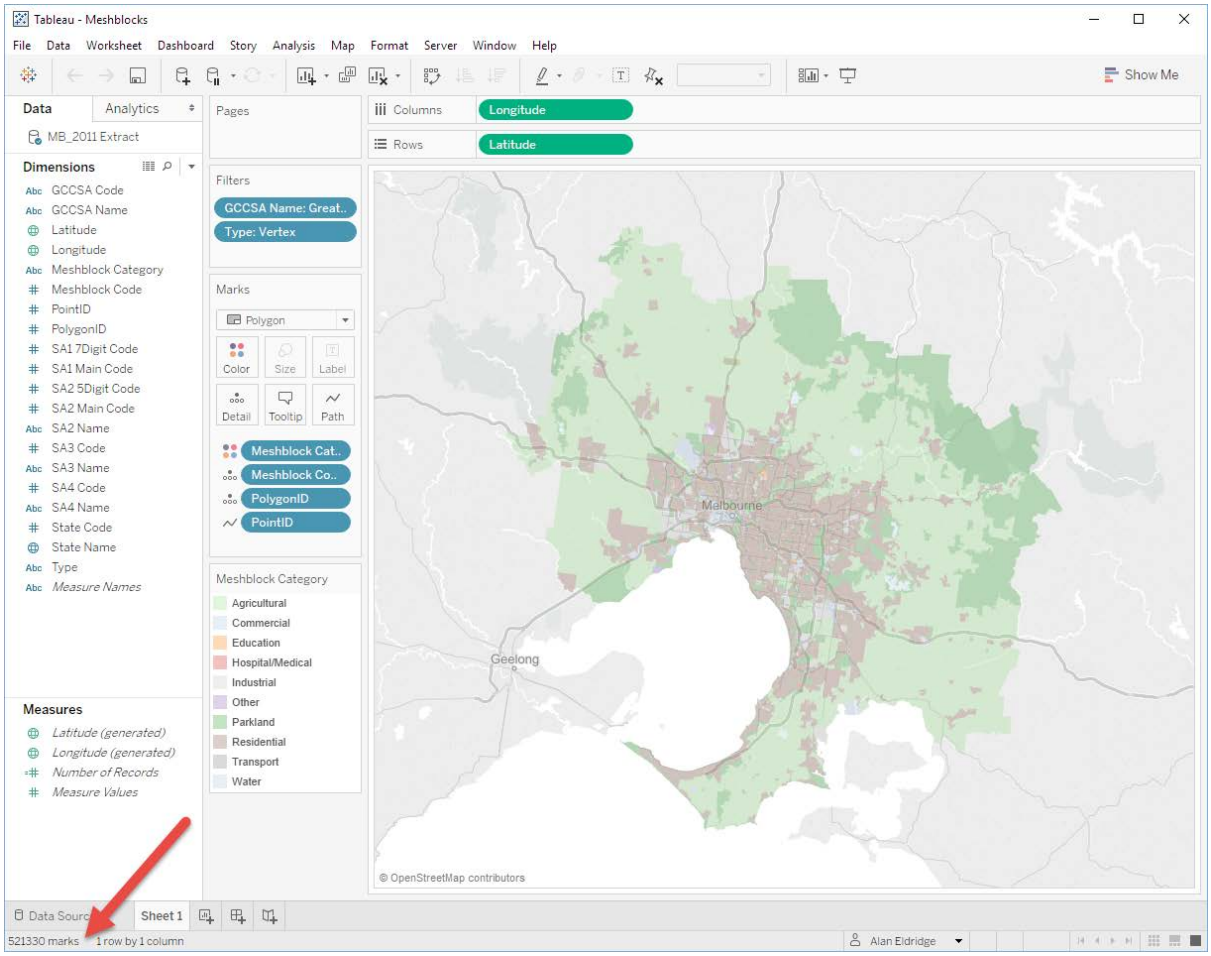


シート 1 では単一のマークのみ表示しており、各国の平均オーダーサイズを示しています。したがって、データソースから返すレコードは 23 件だけです。しかし、シート 2 では各国の各オーダーを表示しており、平均はリファレンスラインとして計算しています。ここでは、データソースから 5436 件のデータを取得する必要があります。

元の質問、つまり国ごとの平均オーダーサイズにだけ興味がある場合は、シート 1 の方が良いソリューションです。しかしシート 2 はその質問に答えながら、オーダーサイズの範囲についてより深いインサイトを提供し、外れ値も特定できます。

Viz が複雑になり過ぎないようにする

重要な指標として、各 viz でレンダリングされるデータポイントの数があります。これは、Tableau Desktop ウィンドウのステータスバーを見れば確認できます。

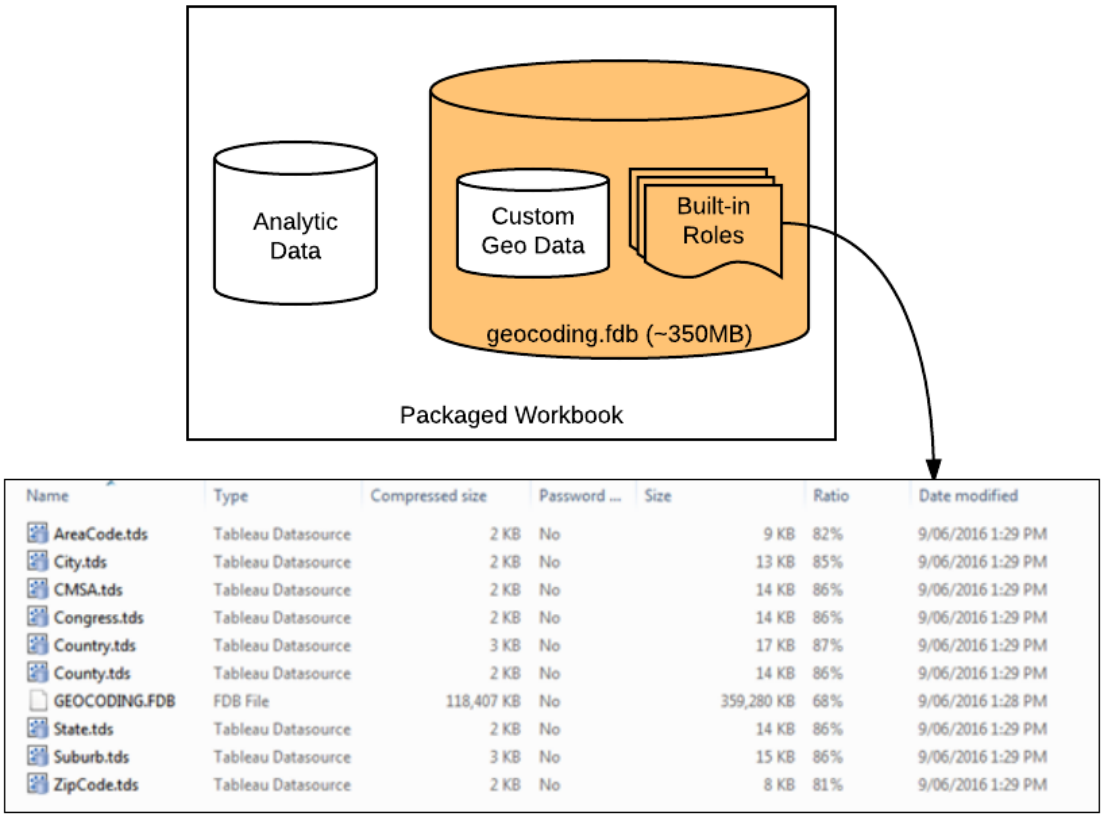


「マークが多過ぎる」状況を明確に定義するルールはありませんが、マークが多いほど強力な CPU と RAM がレンダリングに必要であることを認識してください。大きなクロス集計、散布図、および/または複雑なカスタムの多角形が表示されたマップには注意しましょう。

マップ

カスタムジオコーディング

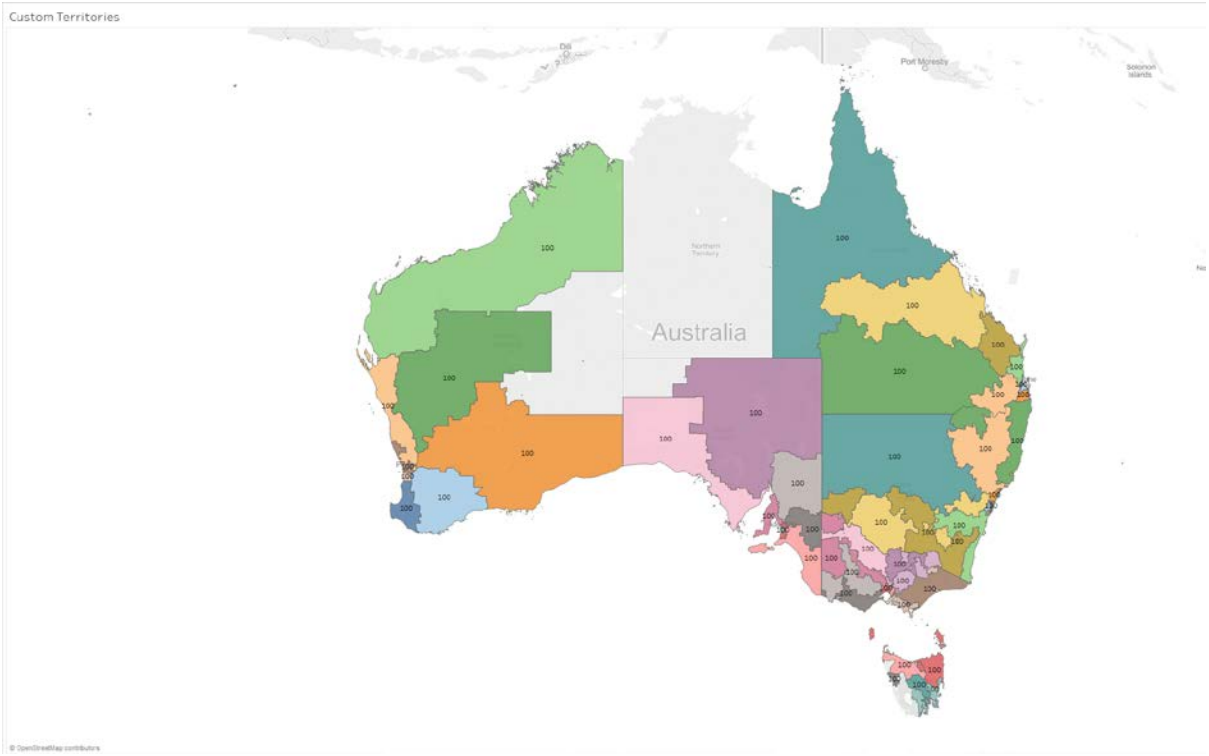
カスタムジオコーディングロールをインポートすると、ジオコーディングデータベースに書き込まれます。これは–Firebird DB ファイルで、既定では C: \Program Files\Tableau\Tableau 10.0\Local\data\geocoding.fdb に格納されています。ワークブックのロールを使用し、パッケージドワークブックとして保存する場合は、データベースファイル全体が TWBX ファイルに圧縮され、合計で約 350 MB にもなります。



これにより、作成された TWBX ファイルは a) 非常にサイズが大きく、ジオコーディングのサイズが圧縮されても約 110 MB になり、b) 最初の展開時にはずっと大きいデータセットを扱う必要があるため、開くのにかなり時間がかかります。さらに効率的なアプローチは、データをカスタムジオコーディングロールとしてインポートせずに、分析データと地理空間データを組み合わせるためにワークブック内でブレンディングを使用することです。このアプローチを用いると、geocoding.fdb ファイルは埋め込まれずに TWBX に分析とカスタムジオコーディングデータのみが含まれることになります。

地域区分のカスタマイズ

Tableau 10 の新機能に、地域区分のカスタマイズがあります。これを使ってユーザーは、内部のジオコーディングデータベースのエリアを結合して集計された地域を作成できます。



多数の低レベルの地域に基づいて実行される地域区分のカスタマイズの初期レンダリングには非常に時間がかかるため、この機能は慎重に使ってください。ただし、これを一度実行した後には、カスタマイズされた地域区分がキャッシュされるため高いパフォーマンスが期待できます。

色塗りマップとポイントマップ

色塗りマップのマークは（マップ上で使用される場合、または別のタイプのグラフでマークとして使用される場合に関わらず）、クライアント側でレンダリングすると、高くつくことがあります。これは、形状のための多角形データを送る必要があり、これが非常に複雑な作業になるとがあるためです。レンダリングのパフォーマンスが遅いことに気づいたら、代わりにシンボルマップの使用を検討してください。

多角形マーク

多角形のマークを使用するビジュアライゼーションでは、Tableau Server でサーバー側レンダリングが実行されるようにします。これは、エンドユーザーの操作性に影響する可能性があります。これは控えめに使用してください。

その他の要素

大きなクロス集計

このドキュメントの初期のバージョンでは、大きなクロス集計はレンダリングが非常に遅いこため、使用しないことを推奨しました。このタイプのビジュアライゼーションの根本的な仕組みは最近のリリースで改善されており、クロス集計は他の小さい複数のチャートタイプと同様に速くレンダリングできるようになりました。ただし引き続き、大きなクロス集計の使用には慎重になるようアドバイスします。大量のデータを参照元データソースから読み取る必要があり、分析に役立たないためです。

ツールヒント

既定では、ディメンションをツールヒントシェルフに配置すると、ATTR() 属性関数を使用して集計されます。このとき参照元データソース内で MIN() と MAX() という 2 つの集計を実行する必要があり、

両方の結果を結果セットに戻します。詳しくは、このドキュメントで後述する ATTR() の使用のセクションをお読みください。

複数のディメンション値があっても差し支えなければ、既定の ATTR() でなく 1 つの集計だけ使用する方がさらに効果的です。MIN() か MAX() のどちらかを選択します。どちらを使用しても構いませんが、1 つ選択したらそれだけを使用してキャッシュのヒット率が最大になるようにします。

別の方法は、viz の詳細レベルが影響を受けないことが分かっている場合は、ディメンションをツールヒントシェルフでなく詳細レベルシェルフに配置することです。こうすることで、ディメンションフィールドをクエリの SELECT と GROUP BY 句で直接使用できるようになります。データプラットフォームのパフォーマンスに依存することがあるので、集計が 1 つの場合よりもパフォーマンスが優れているかどうかテストすることを推奨します。

凡例

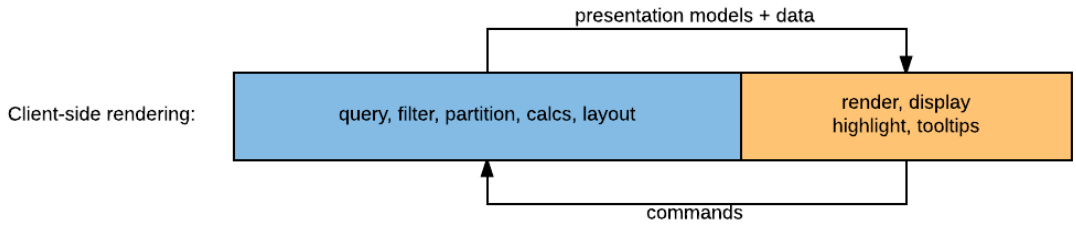
一般的に凡例は、クエリ結果キャッシュからその項目に書き込まれるためパフォーマンス問題の原因とはなりません。ただし、データをクライアントブラウザに転送する必要があるために、列挙された項目が大きくなっている場合は、レンダリングのオーバーヘッドが増大する可能性があります。
この場合、凡例は基本的に役に立たないので削除してください。

ページシェルフ

一部のユーザーは、ページシェルフがフィルターシェルフと同様の機能を果たし、データソースから返されるレコード数を減らすと考えています。それは正しくありません。ワークシートに対するクエリは、すべてのページにわたりすべてのマークのレコードを返します。高密度のページディメンションを含むページがある場合 (一意の値が多数ある場合)、ワークシートクエリのサイズが著しく増加し、パフォーマンスに影響を及ぼします。この機能の使用は控えめにしてください。

クライアント側レンダリングとサーバー側レンダリング

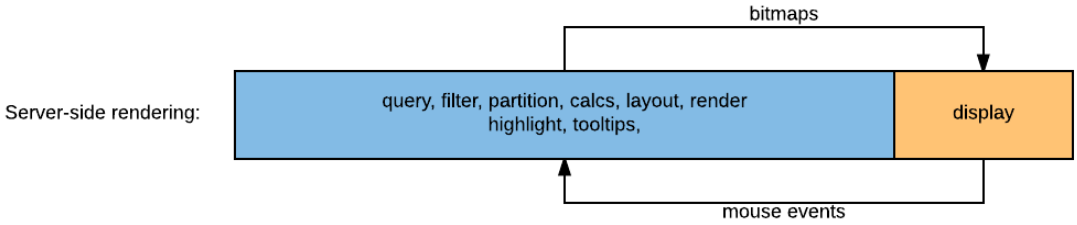
ビューのマークやデータは、クライアントの Web ブラウザに表示される前に取得、解釈、およびレンダリングされます。Tableau Server では、このプロセスをクライアントの Web ブラウザでもサーバーでも実行できます。クライアント側レンダリングは既定モードです。これは、サーバーでレンダリングとすべての操作を処理すると、ネットワークデータ転送とラウンドトリップ遅延が増大するためです。クライアント側レンダリングにすると、解釈やレンダリングがブラウザで実行されるため、ほとんどのビューの操作が速くなります。



(青: サーバーでの処理、オレンジ: クライアントのブラウザでの処理)

ただし、ビューによっては、計算能力の高いサーバーでレンダリングを行ったほうが効率が良くなる場合もあります。サーバー側レンダリングは、画像ファイルに必要な帯域幅が、その画像を作成するのに使用されるデータよりも大幅に少ない、複雑なビューに適しています。また、タブレットの場合は一般に PC よりもパフォーマンスがずっと遅いため、複雑なビューの取り扱いには適していません。ただしその

代わりに、ツールヒントやハイライトのような単純な操作は、サーバー側レンダリングはサーバーとのラウンドトリップが必要なことから遅くなることがあります。



（青: サーバーでの処理、オレンジ: クライアントのブラウザでの処理）

Tableau Server は、これらすべての状況に自動で対応するように設定されています。その際、Web ブラウザではなく、サーバー上でビューのレンダリングを行うトリガーとして、複雑性しきい値を使用します。このしきい値は PC と 携帯デバイスで異なるため、PC の Web ブラウザで開いたビューがクライアント側レンダリングでも、タブレットで同じビューを開くとサーバー側レンダリングになる場合があります。フィルターによっても、レンダリングの動作が変わることがあります。ワークブックが最初にサーバー側レンダリングを使用して開いても、フィルターが適用されるとクライアント側レンダリングに変わる可能性があります。また、viz で多角形マークやページシェルフを使用する場合は、別の方法でクライアント側レンダリングの条件を満たしている場合でも、サーバー側レンダリングのみを使用するため、これらの機能を使用する際は注意が必要です。

管理者は、この設定を-PC 用とタブレット用にテストしたり、微調整したりすることができます。詳細については、次のリンクを参照してください。

https://onlinehelp.tableau.com/current/server/ja-jp/browser_rendering.htm

効率的なフィルター

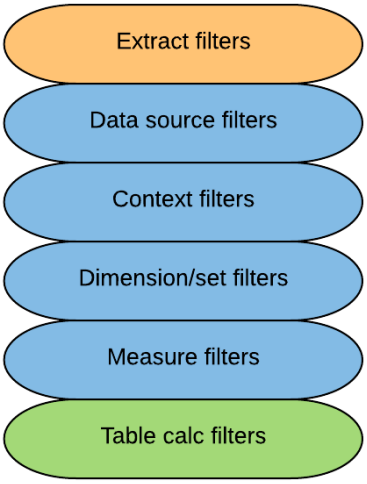
Tableau のフィルターは非常に強力で、表現力に優れています。しかし、非効率的なフィルターが原因で、ワークブックやダッシュボードのパフォーマンスが低下することもよくあります。以下のセクションに、フィルターを使用する際のベストプラクティスを多数提供しています。

なお、フィルターの効率性は、データソース内のインデックスの有無とインデックスの維持によって大きく左右されます。詳しくは、インデックスに関するセクションをご覧ください。

フィルターの種類

フィルターの優先度

Tableau では、フィルターは次の順番で適用されます。



抽出フィルター

このフィルターはデータ抽出を使用するときにのみ適用できますが、その場合は他のすべてのフィルターの前に論理的に適用されます。参照元データソースから取得するデータを制限し、ディメンションフィルターまたはメジャーフィルターのいずれかにできます。さらに、ソースデータのプラットフォームに応じて、TOP または SAMPLE を実行し、返されるレコード数を減らすことができます。

データソースフィルター

データソースフィルターはライブ接続で使用できる最優先レベルのフィルターです。データソースフィルターとコンテキストフィルターの主な相違は、データソースフィルターがデータソース全体を対象としているのに対し、コンテキストフィルターは各ワークシートに設定されるという点です。つまり、パブリッシュ済みのデータソースで使用される場合、データソースフィルターを適用できますが、コンテキストフィルターはワークシートレベルで適用されるということです。

データソースに制限を設けて、エンドユーザーが誤って大量のクエリを実行できないようにするための有効な方法として、データソースフィルターを利用できます。たとえば、データソースフィルターで制限したクエリをトランザクション表に配置し、過去 6 か月間のみを対象にすることが可能です。

コンテキストフィルター

既定では、Tableau で設定したすべてのフィルターは個別に計算されます。つまり、各フィルターは他のフィルターに関係なく、データソース内のすべての行にアクセスします。ただし、コンテキストフィルターを指定することにより、コンテキストフィルターを通過するデータのみが処理されるため、ユーザーが定義するその他すべてのフィルターを依存型にできます。

コンテキストフィルターは、正しい答えを取得する必要があるときに使用してください (例: 上位 N 件のフィルタリング)。たとえば、SUM(Sales) が上位の製品 10 点を地域別にフィルタリングして表示するビューを考えてください。コンテキストフィルターを使わない場合、以下のクエリが実行されます。

```
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]
```

これは、世界の上位 10 製品についてオセアニアにおける貢献度を返します。実際に必要なデータがオセアニア地域の上位 10 製品である場合、地域フィルターをコンテキストに追加すると以下のクエリが実行されます。

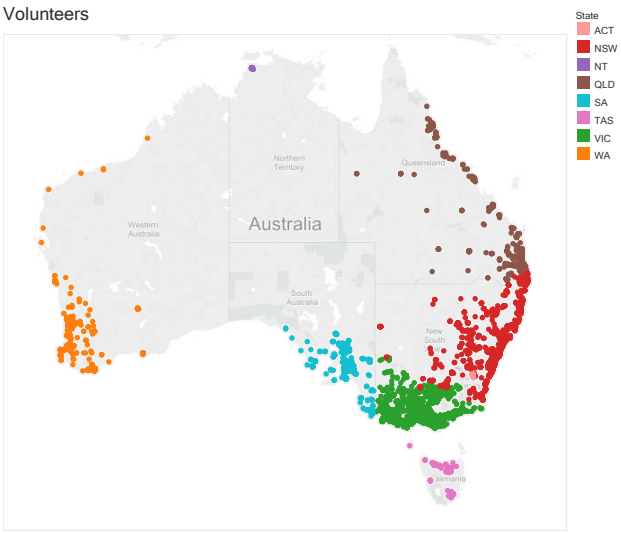
```
SELECT TOP 10 [Superstore APAC].[Product Name] AS [Product Name],
  SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok],
  SUM([Superstore APAC].[Sales]) AS [$__alias__0]
FROM [dbo].[Superstore APAC] [Superstore APAC]
WHERE ([Superstore APAC].[Region] = 'Oceania')
GROUP BY [Superstore APAC].[Product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

従来、コンテキストフィルターはデータソース内の一時表として導入されたものでした。今はそのようなことはありません。ほとんどの場合、コンテキストフィルターは（上述のように）データソースクエリの一部として導入されているか、データエンジン内でローカルに処理されています。

クエリのパフォーマンスを向上させる仕組みとしてコンテキストフィルターを使用する必要はなくなりました。

カテゴリー別ディメンションのフィルタリング

次のビジュアライゼーションを例に考えてみましょう。オーストラリアの地図に郵便番号を示すマークが記されています。



この地図にフィルターを適用して、西オーストラリア州の郵便番号（オレンジ色の点）のみを表示する方法はいくつかあります。

- 西オーストラリア州のマークをすべて選択し、選択したデータのみを保持する
- 西オーストラリア州以外のマークをすべて選択し、選択したデータを除外する
- 州ディメンションなどの別の属性で選択したデータのみを保持する
- 郵便番号の値または経度/緯度の値のいずれかで、範囲のフィルターを適用する

不連続

最初の 2 つのオプションを使用すると、選択したデータのみを保持するオプションと除外するオプションではパフォーマンスが低くなることが分かります。実際、多くの場合フィルタリングしていないデータセットよりも処理速度が遅くなります。これは、複雑な WHERE IN 句を使用することにより、

あるいは多くの値がある場合には、選択された値を設定した一時的な表を作成し、INNER JOIN–を使ってこの表と主要な表を内部結合することにより、DBMS によるフィルタリングで選別される郵便番号の値が不連続のリストとして表されるためです。マークの数がさらに多ければ、このようなクエリには膨大な時間とリソースが必要になります。

この例で 3 つ目のオプションは、結果として得られるフィルター（`WHERE STATE="Western Australia"`）が非常に単純で、データベースで効率良く処理できるため、速く実行できます。ただしこの方法では、フィルターを表すのに必要なディメンションの数が増えるにつれて効率が悪くなり、最終的には投げ縄選択したデータのみを保持するオプションと同レベルのパフォーマンスになります。

値の範囲

値の範囲指定フィルターを適用するやり方でも、データベースは単純なフィルター句（`WHERE POSTCODE >= 6000 AND POSTCODE <= 7000` または `WHERE LONGITUDE < 129`）を求めることになるため、実行速度が速くなります。ただしこの方法は、フィルターや関連ディメンションの場合と異なり、ディメンションの濃度を上げてもこれ以上複雑にはなりません。

結論として、値の範囲指定フィルターは多くの場合、不連続の値が項目別に分けられた大規模なリストよりも速く評価されます。マークの数が多い場合は、できるだけ選択したデータのみを保持するオプションまたは除外するオプションよりも優先して使用する必要があります。

スライシングフィルター

スライシングフィルターはディメンション上のフィルターで、viz では使用されません（viz の詳細レベルの一部にはなりません）。たとえば、国別の合計売上を表示する viz があり、これが地域ごとにフィルタリングされているとします。この場合、実行されるクエリは以下のようになります。

```
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]
```

これらのフィルターは、集計の結果をスライスするとさらに複雑になります。たとえば、上記の viz を、地域でフィルタリングするのではなく、利益率の高い製品上位 10 品目の売上を表示するようにフィルタリングする場合、Tableau では 2 つのクエリを実行する必要があります。1 つは利益率の高い製品上位 10 品目を特定するために製品レベルで実行し、もう 1 つは最初のクエリ結果によって制限される国レベルで実行します。

```
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]
```

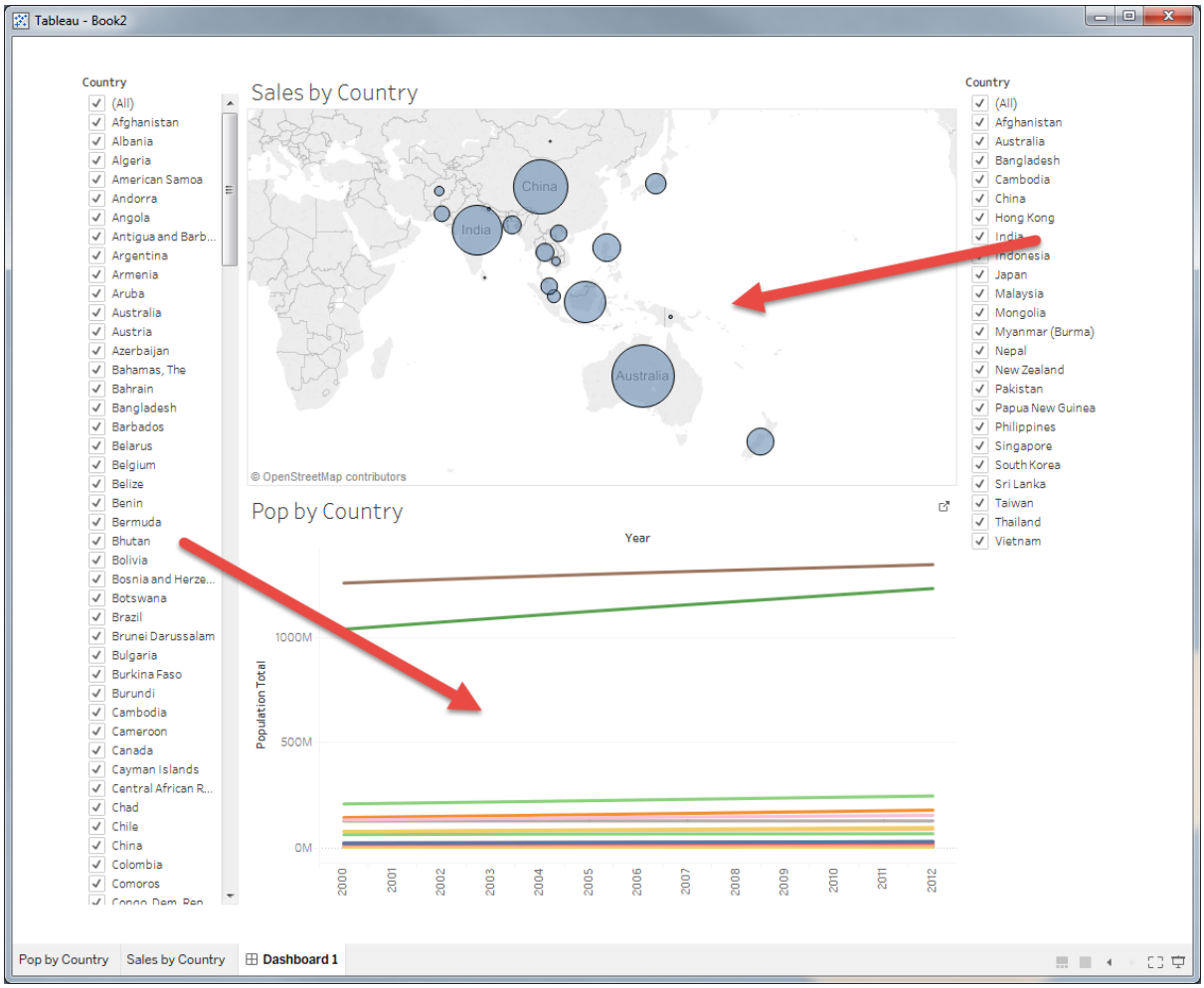
スライシングフィルターは、評価に高くつく場合があるので、使用するときは注意してください。また、ディメンションはクエリ結果のキャッシュの一部ではないため、スライシングフィルターでブラウザ内フィルタリングを高速に行うことはできません（クライアント側レンダリングとサーバー側レンダリングに関する前のセクションを参照）。

クロスデータソースフィルター

クロスデータソースフィルターは、Tableau 10 の新機能です。これにより、1 つまたは複数のフィールドを共有する複数のデータリソースに対してフィルターを適用できます。リレーションシップはブレンディングと同様の方法で定義します。つまり、名前/タイプの一致に基づいて自動で定義されるか、データメニュー内のカスタムリレーションシップ経由で手動で定義されます。

クロスデータベースフィルターは、ダッシュボード上のクイックフィルターと同様、パフォーマンスと密接な関係があります。このフィルターを変更すると、複数ゾーンで更新が発生して複数のクエリの実行が必要になる可能性があります。この機能は慎重に使用してください。hユーザーが複数の変更を行うことが予想される場合は、選択が完了した後にのみクエリを開始できるように、[適用]ボタンを表示することを検討してください。

また、フィルターの領域は「一次」データソースから取り込んでいることにも留意してください。「一次」データソースとは、フィルターが作成されたシートの最初に使用されるデータソースです。関連フィールドが異なるデータソース内に異なる領域を持っている場合、下図のように同一のフィルターが異なる値を表示する結果となる可能性があるため、どれを使用するか注意しなければなりません。

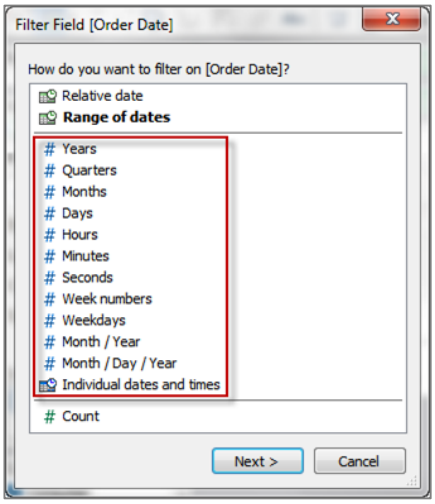


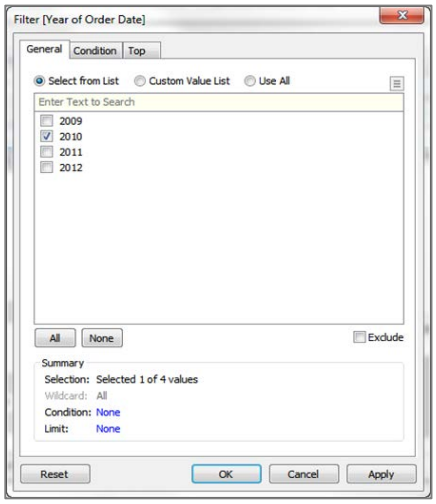
日付のフィルタリング: 不連続、範囲指定、相対

日付フィールドは特殊な種類のディメンションであるため、Tableau では標準のカテゴリーデータとは異なる方法で処理されることがよくあります。これは、日付フィルターを作成する場合に特に当てはまります。日付フィルターは非常に一般的で、次の 3 つのカテゴリーがあります。特定の日付を基準とした日付範囲を示す相対日付フィルター、定義済みの不連続な日付の範囲を示す範囲指定日付フィ

ルター、およびリストから選択した個別の日付を示す不連続日付フィルターです。上記のセクションで説明したとおり、使用する方法はクエリの効率に重大な影響を与える可能性があります。

不連続





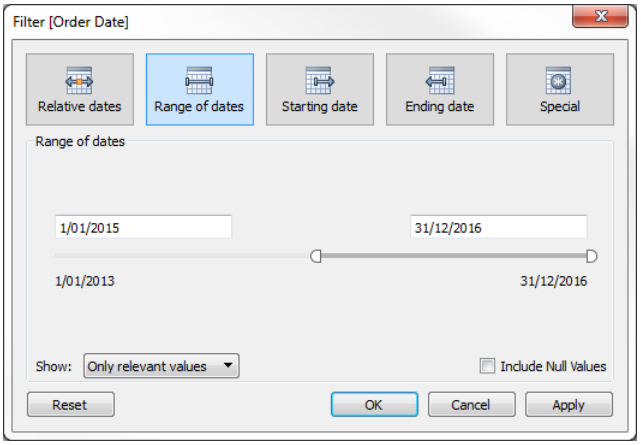
特定の日付または日付レベル全体を含めるフィルターが必要な場合があります。このタイプのフィルターは不連続日付フィルターと呼ばれます。これは、範囲ではなく不連続の値を定義するためです。このタイプのフィルターでは、日付式が動的計算としてデータベースに送信されます。

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

ほとんどの場合、クエリ最適化ツールは DATEPART 計算を合理的に求めますが、不連続日付フィルターを使用するとクエリの実行パフォーマンスが低下するケースもいくつかあります。たとえば、日付のパーティションキーで不連続日付フィルターを使用して分割された表に対してクエリを実行するとします。この表は DATEPART 値で分割されていないため、データベースによっては、必要がなくても、すべてのパーティションにわたって計算を求め基準に一致するレコードを探すことがあります。この場合、「日付の範囲」フィルターまたは特定の日付を基準とした相対日付フィルターを使用することで、パフォーマンスの大幅な向上が見られる場合があります。

この種のフィルターのパフォーマンスを最適化する方法の 1 つに、計算フィールドのデータを計算済みのデータとしてデータ抽出に持つ方法があります。まず、DATEPART 関数を明示的に実装する計算フィールドを作成します。その後 Tableau データ抽出を作成すると、この計算フィールドは保存された値として抽出に持ち続けることができます (式の出力が決定性であるため)。動的な式ではなく、計算フィールドに対してフィルタリングを行うほうが処理が速くなります。クエリの実行時に計算する代わりに値を検索できるためです

日付の範囲

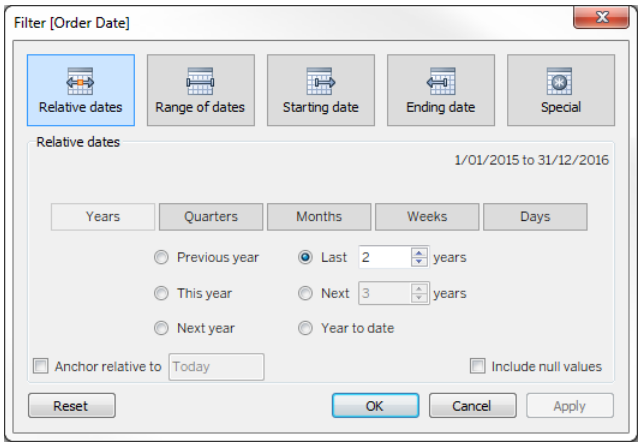


このタイプのフィルターは、連続する日付の範囲を指定するときに使用します。結果として、次のような構造のクエリがデータベースに渡されます。

```
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 ()
```

このようなタイプの WHERE 句はクエリ最適化ツールにとって非常に効率が良く、インデックスやパーティションを最大限に活用する実行計画が可能にあります。不連続日付フィルターを追加するとクエリに時間がかかる場合は、日付範囲フィルターに変えてパフォーマンスが向上するか確認してみてください。

相対



相対日付フィルターを使用すると、ビューを開いた日付と時刻に基づいて、更新する日付の範囲を定義できます。たとえば、今年度の累計売上高、過去 30 日間のすべてのレコード、先週処理したバグの情報などを表示するとします。相対日付フィルターは、当日ではなく特定のアンカー日付を基準とすることもできます。

```
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 ()
```

見て分かるとおり、結果として得られる WHERE 句では日付の範囲構文が使用されるため、これも効率的な日付フィルターとなります。

上記の相対日付オプションにより、あるいはフィルター式で NOW() または TODAY() を明示的に使用することにより、現在の日付/時刻に相対的な日付フィルターは変更する性質があるため、クエリのキャッシュはそれらを使用するクエリにとってそれほど効率的でないことに注意してください。Tableau はキャッシュサーバー内でこれらのクエリに「一時クエリ」のフラグを付け、そのクエリ結果は他のクエリ結果ほど長く保持されません。

フィルターカード

フィルターカードを多く表示し過ぎると処理時間が長くなります。特に「関連値のみ」を使用するよう設定すると、多数の不連続リストを処理することになります。代わりに誘導形式の分析アプローチを採用し、ダッシュボード内でアクションフィルターを使用してみてください。非常に多くのフィルターを含めて高度にインタラクティブなビューを構築している場合は、レベルやテーマが異なる複数のダッシュボードを作ると動作が改善するか自問してみましょう（答えはおそらく「イエス」です）。

列挙と非列挙

列挙型のフィルターカードの場合、Tableau は、フィルターカードをレンダリングするために、フィールド値についてデータソースに対してクエリを実行する必要があります。このようなフィルターカードには次のものが含まれます。

- 複数の値のリスト – すべてのディメンションメンバー
- 単一の値のリスト – すべてのディメンションメンバー
- 圧縮リスト – すべてのディメンションメンバー
- スライダー – すべてのディメンションメンバー
- メジャーフィルター – 最小値および最大値
- 日付の範囲フィルター – 最小値および最大値

反対に、非列挙型のフィルターカードの場合、潜在的なフィールド値を知る必要はありません。このようなフィルターカードには次のものが含まれます。

- カスタム値リスト
- ワイルドカード照合
- 相対日付フィルター
- 期間参照日付フィルター

その結果、非列挙型のフィルターカードでは、データソースによって実行される必要がある、フィルター関連のクエリの数が減ります。また、非列挙型のフィルターカードでは、表示するディメンションメンバーが多い場合に、レンダリング処理を速くすることができます。

非列挙型のフィルターカードを使うことでパフォーマンスを向上できますが、エンドユーザーからの可視性が犠牲になります。

関連値

列挙型のフィルターカードは、次の 3 種類の方法で潜在的なフィールド値を表示するよう設定できます。

- [データベース内のすべての値] – このオプションを選択すると、ビュー上のその他のフィルターにかかわらず、データベース内のすべての値が表示されます。他のフィルターが変更されても、フィルターはデータベースに対してクエリを再度実行する必要がありません。
- [コンテキスト内のすべての値] – このオプションは、アクティブなコンテキストフィルターがある場合にのみ利用できます。フィルターカードは、ビューにある他のフィルターに関係なくコンテキスト内のすべての値を表示します。他のディメンションやフィルターが変更されても、フィルターはデータベースに対してクエリを再度実行する必要がありません。ただしコンテキストが変更された場合にはクエリを再度実行する必要があります。
- [関連値のみ] – このオプションを選択すると、他のフィルターが考慮され、それらのフィルターを通過した値のみが表示されます。たとえば、[地域] のフィルターを設定している場合、[州] でフィルターをかけると東部の州のみが表示されます。その結果、このフィルターは、他のフィルターが変更されたときにデータソースに対してクエリを再度実行する必要があります。

見てわかるように、[関連値のみ] 設定はユーザーが関連する選択を行ううえで非常に役立ちますが、ダッシュボードとやり取りしている間に実行する必要のあるクエリの数が大幅に増加することもあります。適度に使用することをお勧めします。

フィルターカードの代替手段

フィルターカードを使用する代わりに、同様の分析結果を取得することができ、追加クエリのオーバーヘッドも発生しない方法があります。たとえば、ユーザーの選択に基づいてパラメーターとフィルターを作成することができます。

- メリット:
  - パラメーターの場合、レンダリングの前にデータソースに対するクエリを実行する必要がない
  - パラメーターと計算フィールドを併用することで、シンプルなフィールドフィルターでできるよりも複雑な論理を実装できる
  - パラメーターを使用して、複数のデータソース全体にわたってフィルターを適用できる (Tableau 10 より前のバージョンのフィルターは 1 つのデータソース内でしか動作しない) Tableau 10 以降のバージョンでは、関連するデータソース全体にわたってフィルターを設定できる

- デメリット:
  - パラメーターは単一の値のみ。ユーザーに複数の値を選択させたい場合には使用できない
  - パラメーターは動的でない。値のリストは作成時に定義され、DBMS にある値に基づいて更新されない

別の方法は、ビュー間でフィルターアクションを使用することです。

- メリット:
  - アクションでは、視覚的に範囲を囲むか、CTRL / SHIFT のクリックによって複数の値を選択できる
  - アクションでは、実行時に求められた値の動的リストが表示される
  - アクションを使用して、複数のデータソース全体にわたりフィルターを適用できる (Tableau 10 より前のバージョンのフィルターは 1 つのデータソース内でしか動作しない) Tableau 10 以降のバージョンでは、関連するデータソース全体にわたってフィルターを設定できる
- デメリット:
  - フィルターアクションは、フィルターカードよりも設定が複雑である
  - アクションでは、パラメーターやフィルターカードのようなユーザーインターフェイスが提供されず、通常、表示に必要な画面の面積が大きくなる
  - アクションソースシートでは、データソースに対して引き続きクエリを実行する必要があるが、Tableau 処理パイプライン内のキャッシュを活用できる

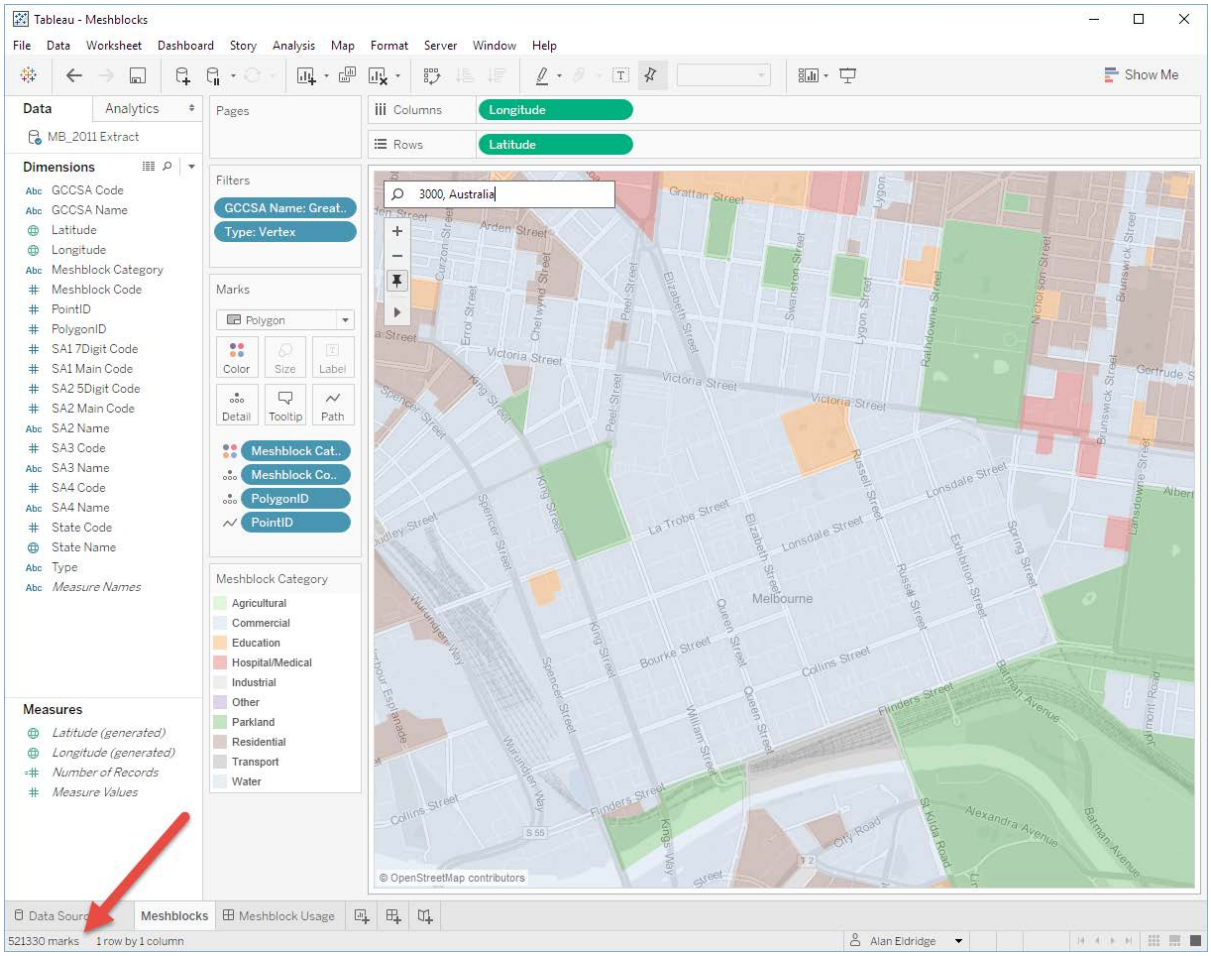
フィルターカードに依存し過ぎない代替の設計手法についての詳細は、優れたダッシュボード設計に関する上述のセクションをご覧ください。

ユーザーフィルター

ワークブックにユーザーフィルターが含まれているとき、[ユーザーフィルターの作成] ダイアログ、または ISMEMBEROF() などのビルトインのユーザー関数のいずれかを使用した計算フィールドのいずれを使用した場合でも、モデルキャッシュは複数のユーザーセッションにわたって共有されることはありません。これによりキャッシュの再利用率が大幅に下がり、Tableau Server にとっては負荷が増えることになります。これらのフィルタータイプは、慎重に使用することをお勧めします。

ズームとフィルタリング

マークが多数入った viz をズームするとき、表示されないマークはフィルタリングされません。変更中のものはすべて、データの見えるビューポイントです。コントロール対象のマークの合計数は、下図でご覧のとおり変わりません。



データのサブセットだけが必要な場合は、不要なデータをフィルターで除外して Tableau の自動ズーム機能でビューポイントを設定します。

計算を再検討する

多くの場合、ソースデータには、すべての質問に答えるのに必要なフィールドが揃っていません。計算フィールドは、分析に必要なすべてのディメンションやメジャーを作成するうえで役立ちます。

計算フィールド内では、ハードコードされた定数（たとえば税率など）を定義する、減算や乗算など非常に単純な数学演算（「収益－経費」など）を行う、より複雑な数式を使用する、論理テスト（IF/THEN、CASE など）を実行する、型変換を行うなどの操作が可能です。

一度定義すると、計算フィールドはワークブック全体で利用できます。ただし、これはワークシートが同じデータソースを使用している場合に限ります。計算フィールドは、ソースデータからのディメンションやメジャーを使う場合と同様に、ワークブックで使用できます。

Tableau には 4 つの異なる計算タイプがあります。

- 行レベルの計算
- 集計計算
- 表計算
- LOD 表現

以下のフローチャートを使って、ベストなアプローチを選択する際の参考にしてください。

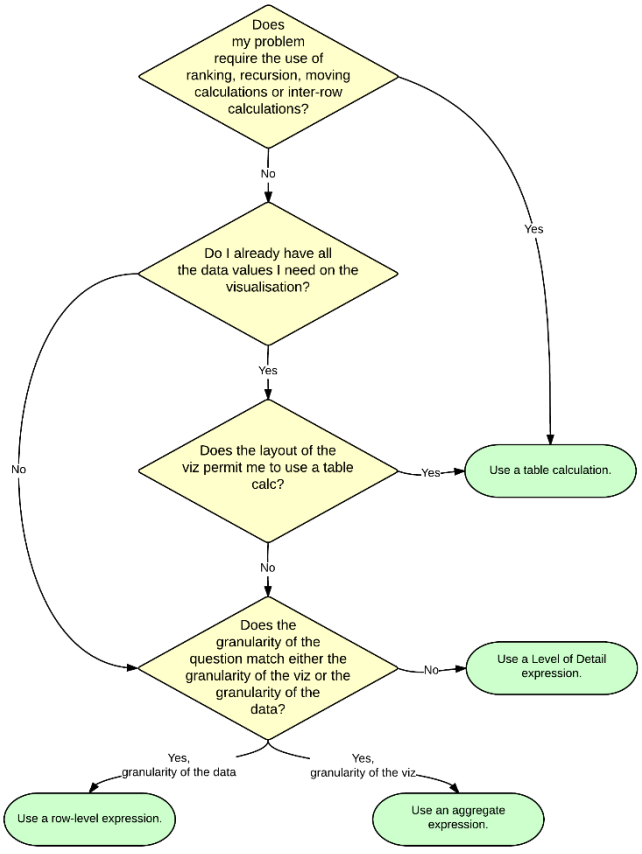


Tableau 計算レファランスライブラリに、複雑な計算を行う方法についての優れた参考資料や、よくある問題に対するソリューションを共有するフォーラムがあります。

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

計算タイプ

行レベルの計算と集計計算

行レベルの計算と集計計算は、データソースに送信されるクエリの一部として表現され、そのためデータベースが計算を実行します。たとえば、毎年の総売上高の合計を表示している viz は、以下のクエリーをデータソースに送ります。

```
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 計算が行レベルの計算の例で、SUM(Sales) が集計計算の例です。

一般に、行レベルの計算と集計計算はスケーラビリティに優れ、計算のパフォーマンスを向上するために採用できるデータベース調整の手法も多数あります。

Tableau では、実際の DB クエリは viz で使う基本的な計算を直接変換したものとは限らないことに注意してください。たとえば、以下のクエリは viz が SUM([Profit])/SUM([Sales]) として定義される計算されたフィールド [Profit Ratio] を含む場合に実行されます。

```
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])
```

Tableau は実際に、計算の要素を取得し、クライアントレイヤーで division 関数を実行しています。これで年レベルの SUM([Profit]) と SUM([Sales]) がキャッシュされ、ワークブック内の任意の場所からデータソースにアクセスしなくても使用できます。

最後に、クエリ統合 (複数の論理クエリを 1 つの実際のクエリと組み合わせること) により、データソースに対するクエリの実行が変更される可能性があります。別のワークシートで同じ粒度が共有される場合、結果的にその複数のワークシートにある複数のメジャーが結合されて単一のクエリとなります。

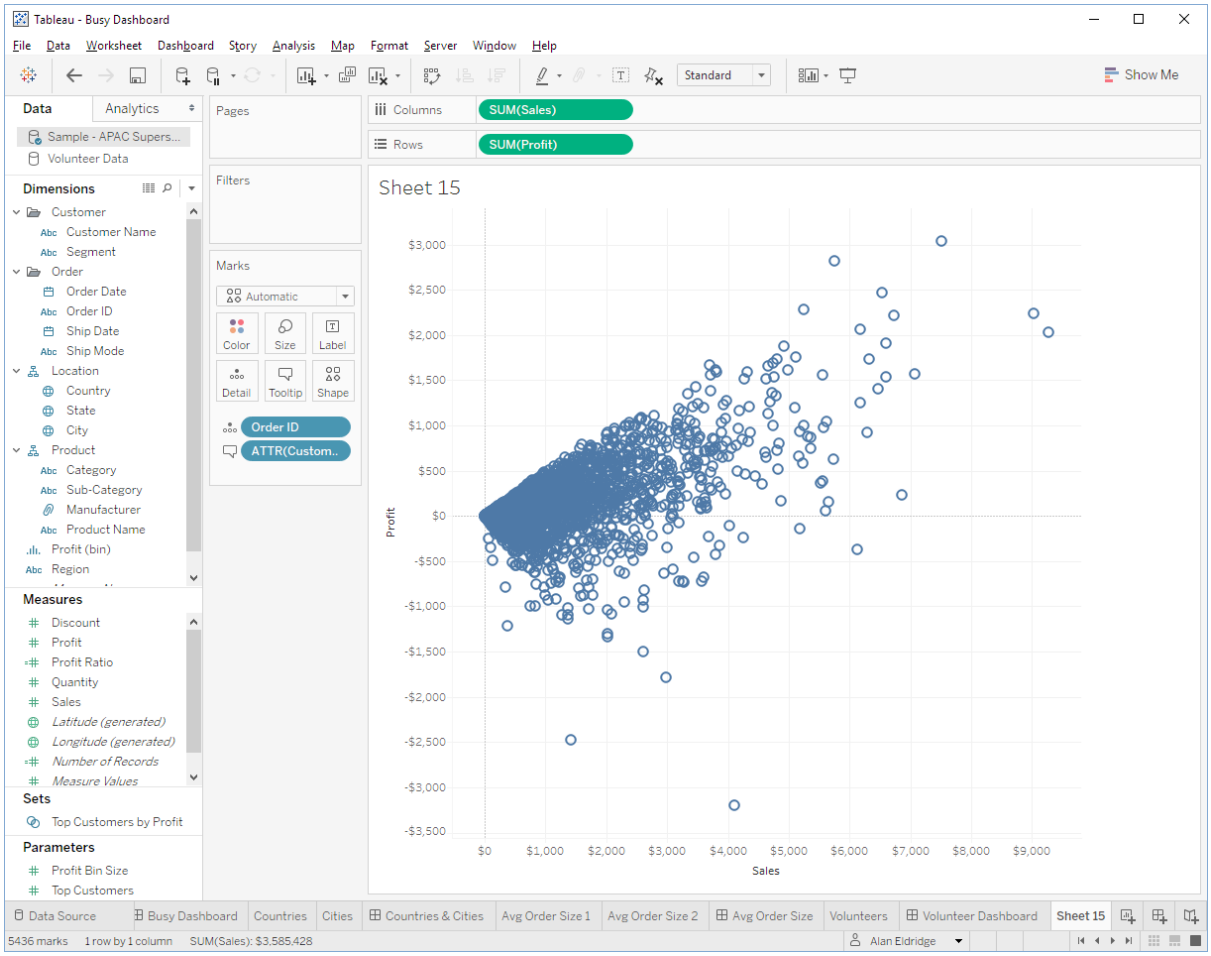
ATTR() の使用

ATTR() は協力な関数で、カテゴリーディメンションの集計によく使用されます。簡単に言うと、一意であれば値を返し、それ以外のときは * を返します。技術的には、この関数は以下のような論理テストを実行しています。

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

このとき参照元データソース内で MIN() と MAX() という 2 つの集計を実行する必要があり、両方の結果を結果セットに戻します。複数のディメンション値があっても構わないと思う場合、MIN() か MAX() のどちらか 1 つの集計だけ使用するとさらに効率が上がります。どちらを使用しても構いませんが、1 つ選択したらそれだけを使用してキャッシュのヒット率が最大になるようにします。

ATTR() のデータソースへの影響を示すために、以下のような viz を作成しました。



既定では、ディメンションをツールヒントシェルフに配置すると、ATTR() が既定の集計として使用されます。これにより、以下のクエリが実行されます。

```
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]
```

各 [オーダー ID] に複数のお客様がいないことがわかっていれば、集計を MIN() に変更してクエリをより簡単にすることができます。

```
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]
```

次の表は 2 つのオプションでパフォーマンスが異なることを示しています (結果はクエリが完了するまでの秒数で、ログファイルから取得)。

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%**

ATTR() について詳しくは、InterWorks からの次のブログ記事を参照してください。

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

詳細レベルの計算

詳細レベル (LOD) 表現は、Viz LOD とは無関係に、計算を実行する詳細レベルを定義できます。

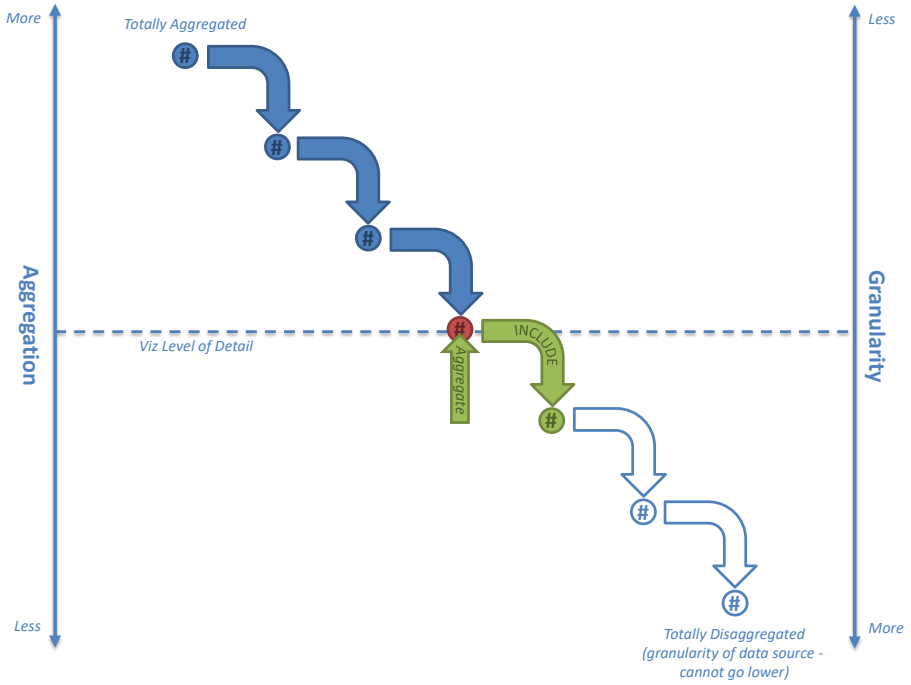


Tableau の以前のバージョンでは （場合によって) LOD 表現を使用して、作成した計算をより扱いにくい方法で置き換えることができます。

- 表計算を使用して、区分内の最初の期間と最後の期間を見つけることができました。たとえば、毎月の最初の日に組織に在籍する従業員数を計算するような場合です。
- 表計算、計算フィールド、リファレンスラインを使用して、ビン集計フィールドを設定しようと試みたことがあるかもしれません。たとえば、顧客の個別カウントの平均を求めるような場合です。
- データブレンディングを使用して、データ内の最大日数に対して相対的な日付フィルターを適用しようとしたかもしれません。たとえば、データが 1 週間ごとに更新される場合に、年度初めから最大の日付までの合計を計算するときです。

表計算とは異なり、LOD 表現は、参照元のデータソースに対するクエリの一部として生成されます。LOD 表現は、ネストされた Select 文として表現されるので、DBMS のパフォーマンスの影響を受けます。

```
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]
```

これは、1 つのアプローチまたは他のアプローチを使用してより効率的に問題を解決できるシナリオがある可能性があることを意味します。表計算やブレンディングは、LOD より優れたパフォーマンスを発揮する可能性があり、その逆も可能です。LOD 表現のためにパフォーマンスが遅くなっていると感じる場合は、表計算やデータブレンド (可能な場合) と置き換えて、パフォーマンスが向上するか見てみましょう。また、LOD 表現は結合選択から強い影響を受けることもあるため、LOD 表現の使用時にクエリの実行が遅いと感じたら、戻ってそのセクションをもう一度読むことをお勧めします。

LOD 表現の仕組みをより良く理解するには、ホワイトペーパー「詳細レベル (LOD) 表現について」をお読みください。

http://www.tableau.com/ja-jp/learn/whitepapers/understanding-lod-expressions

また、よくあるタイプの問題を数多く説明しているベタニー・リヨンのブログ記事「LOD 表現トップ 15」もお勧めします。

http://www.tableau.com/ja-jp/about/blog/LOD-expressions

最後に、このテーマについては、コミュニティからも非常に役立つ多くのブログ記事が出ています。data + science に役立つリストがあります。

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

表計算

表計算は、行レベルの計算や集計計算とは異なり、データベースにより実行されるのではなく、クエリ結果セット上で Tableau によって実行されます。このため Tableau での処理量が増えますが、一般的に元のデータソースに含まれるよりずっと小さいレコードセットが処理されます。

表計算のパフォーマンスに問題がある場合 (Tableau に返された結果セットが非常に大きいことが原因と考えられます) は、計算の一部をデータソースレイヤーに戻すことを検討してください。1 つの方法は、LOD 表現を介して行うことです。別の方法として、集計データ抽出の活用も考えられます。複数の店舗における 1 日の売上合計の週平均を求める必要がある場合を考えてみてください。表計算でこれを求めるには、次の式を使います。

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

ただし、日数や店舗数が非常に大きくなると、この計算には時間がかかります。SUM([Sales]) をデータレイヤーに戻すには、日付ディメンションを日レベルまでロールアップする集計抽出を作成します。抽出に 1 日の合計が既に含まれているため、計算は AVG([Sales]) で実行できます。

場合によっては、集計要素の値が区分を超えて変更されないことが分かっているかもしれません。その場合は、集計関数として AVG() または ATTR() の代わりに、ずっと速く評価できる MIN() または MAX() を使用します。1 つ選択したらずっとそれだけを使用して、キャッシュの一致率を上げるようにしてください。

外部計算

Tableau には、複雑な論理外部エンジンに渡して計算できる仕組みがあります。これには、R (Rserve 接続経由) や Python (Dato サーバー接続) への呼び出しの作成が含まれます。

Tableau 内では、これらの操作は表計算に類似しているので区分を超えて計算されます。これは、1 つの viz のためにこの操作が複数回呼び出されることを意味するため、パフォーマンスのオーバーヘッドが発生する可能性があります。最適化も融合もまったく行われていないため、複数の関数呼び出しを作成せずに、単一の関数呼び出し (連結文字列など) で複数の戻り値を取得できるかどうか検討してください。最後に、外部計算エンジンとの間でデータ転送にかかる時間もオーバーヘッドとなる可能性があります。

アナリティクス

アナリティクスペインでは、次のような数多くの詳細分析を簡単に利用できます。

- 合計
- リファレンスライン
- 傾向線
- 予想
- クラスタリング (Tableau 10 の新機能)

これらのアナリティクスには、通常、追加のクエリ実行が必要ではありません。どちらかといえば、クエリ結果キャッシュ内のデータ上での表計算のように機能します。表計算のように、クエリ結果内に非常に大きなデータのセットがある場合は計算プロセスに時間がかかりますが、通常はワークブックの全体的なパフォーマンスが低下する要因にはなりません。

合計とリファレンスラインのパフォーマンスに影響する別の要因は、メジャーの集計が加法集計であるか非加法集計であるかということです。加法集計 (例: SUM、MIN、MAX) の場合、合計の計算またはリファレンスラインは、Tableau 内でローカルで実行できます。非加法集計 (例: COUNTD、TOTAL、MEDIAN、PERCENTILE) の場合、データソースに戻って合計/リファレンスラインの値を計算する必要があります。

計算とネイティブ機能

ユーザーが計算フィールドを作成して実行する関数のいくつかは、Tableau のネイティブ機能で簡単に実現できる場合があります。その一例をご紹介します。

- ディメンションメンバーをグループ化する – グループまたはセットを使用します。
- メジャー値をグループ化して「ビン」を作成する – ビンを使用します。
- 月や週など、粒度を下げて日付を切り捨てる – カスタム日付フィールドを使用します。
- 2 つの異なるディメンションの連結である値のセットを作成する – 結合済みフィールドを使用します。
- 特定の形式で日付を表示する、または数値を KPI インジケーターに変換する – ビルトインの書式設定機能を使用することを検討してください。
- ディメンションメンバーに表示される値を変更する – 別名を使用します。

基本のビンでは不可能な可変幅のビンが必要な場合など、これがすべてのケースで可能であるとは限りませんが、ネイティブ機能がある場合はそれを利用することを検討してください。そうすることで多くの場合、手動で計算するよりも効率が上がります。また、当社の開発者が Tableau のパフォーマンス向上に継続的に取り組んでいるため、その成果が現れてくることが予想されます。

データ型の影響

計算フィールドを作成するときは、使用するデータ型が計算速度に大きく影響することを理解しておく必要があります。一般的な指針は次のとおりです。

- 整数とブールは、文字列と日付よりずっと速い

文字列と日付の計算は非常に時間がかかります。各計算に対して、10 ～ 100 件の基本的な指示の実行が必要となる場合がよくあります。一方、数値やブール値の計算は非常に効率的です。

このことは、Tableau の計算エンジンだけでなく、ほとんどのデータベースにも当てはまります。行レベルの計算と集計計算はデータベースに戻されるため、数値論理と文字列論理で表現されている場合は、ずっと速く実行されます。

パフォーマンス向上のための手法

計算をできるだけ効率良く実行するために、以下の手法を検討してみてください。

基本論理計算にブール値を使用

結果が 2 つの値のいずれかになる計算（はい/いいえ、合否、超過/未満など）の場合は、文字列よりもブール値が返されるようにします。例:

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

この場合は文字列が使われているため、計算が遅くなります。この計算をより速く行うには、ブール値で結果を得るようにします。

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

次に、別名を使って、TRUE 結果と FALSE 結果の名前を「Today」と「Not Today」に変更します。

文字列検索

製品名に参照文字列が含まれるすべてのレコードを表示したいとします。これを実行するために、ユーザーからの参照文字列を取得するパラメーターを使用し、次の計算フィールドを作成することができます。

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

この計算は、含まれていることをテストするには効率の悪い方法であるため時間がかかります。これを効率良く行うには、特定の CONTAINS 関数を使用することをお勧めします。この関数は、データベースに送信される際に最適化 SQL に変換されます。

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

ただし、この場合の最適なソリューションは、計算フィールドの代わりにワイルドカード一致のフィルターカードを使用することです。

条件付き計算のパラメーター

Tableau でよく使われる手法は、エンドユーザーにパラメーターを提供し、計算の実行方法を決定する値を自分で選べるようにすることです。たとえば、エンドユーザーが、ビューに表示する日付の集計レベルを可能な値のリストから選択して管理できるようにするとします。多くの場合、次のような文字列型パラメーターが作成されます。

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

次に、これらを次のような計算に使用します。

```
CASE [Parameters].[Date Part Picker]
  WHEN “Year” THEN DATEPART('year',[Order Date])
  WHEN “Quarter THEN DATEPART('quarter',[Order Date])
..
END
```

さらに良い方法は、ビルトインの DATEPART() を使用して以下の計算を作成することです。

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

Tableau の旧バージョンでは、後者のアプローチのほうがずっと速く処理できました。ただし、今では 2 つのソリューション間にパフォーマンスの相違はなくなりました。パラメーター値に基づく CASE ステートメントの条件付き論理を省略して、適切な DATEPART() をデータソースに渡すだけにしたためです。

後々のソリューションの保守性を考慮する必要があることも忘れてはなりません。その場合は後者の計算方法の方が適切な計算式となるかもしれません。

日付変換

日付データがネイティブの日付形式以外で保存されている（文字列や数字のタイムスタンプなど）こともよくあります。DateParse() 関数を使えばこの変換が容易にできます。書式設定文字列として送信するだけです。

```
DATEPARSE(“yyyyMMdd”, [YYYYMMDD])
```

ただし、DateParse() は、次のデータソースのサブセットでのみサポートされています。

- 非レガシーの Excel とテキストファイル接続
- MySQL
- Oracle
- PostgreSQL
- Tableau データ抽出

お使いのデータソースで DateParse() がサポートされていない場合、これを適切な Tableau の日付形式に変換する別の方法は、フィールドを日付文字列（「2012-01-01」など。国際標準に準じた ISO 文字列が推奨されます）に解析してから、DATE() 関数に渡す方法です。

元のデータが数値フィールドの場合は、文字列に変換してから日付に変換すると非常に効率が悪くなります。そのような場合はデータを数値のまま維持し、DATEADD() と日付リテラル値を使用して計算を実行します。

たとえば、速度の遅い計算は（ISO 形式の数値フィールドの変換）は次のようになります。

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

この計算をもっと効率良く行うには、次のようにします。

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

データソースでサポートされている場合、さらに良い方法は MAKEDATE() 関数を使用することです。

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

データセットが大きい場合、パフォーマンスの向上は際立ったものになります。レコード数が 10 億を超えるサンプルでは、最初の計算が完了するのに 4 時間以上かかりましたが、2 つ目の計算は約 1 分で完了しました。

論理ステートメント

最も頻度の高い結果を最初にテストする

Tableau が論理テストを評価するとき、可能な結果を求める処理は、一致が見つかった時点で停止します。これは最も可能性の高い結果に対してテストを最初に行うことを意味します。評価されている大多数のケースで、それらが最初のテストの後に停止するようにするためです。

以下の例を見てください。この論理です。

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

上記の論理は以下の論理よりも評価が遅くなります。

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

ELSEIF > ELSE IF

複雑な論理ステートメントを使用するとき、ELSEIF > ELSE IF を思い出してください。その理由は、ネストされた IF は、最初の IF ステートメントの一部として計算される代わりに、結果クエリ内のネストされた CASE ステートメントを計算するからです。そのため、次の計算フィールドは、

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

以下のように 2 つのネストされた CASE ステートメントと 4 つの条件テストを含む SQL コードを作ります。

```
(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)
```

これは、以下のようににネストされた IF でなく ELSEIF で記述し直し、条件テストをより効率的に使用すると、より速く実行できます。

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

この結果として、SQL コード内に CASE ステートメント が 1 つのみ記述されます。

```
(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)
```

ただし、このほうが依然として速くなります。ネストされた IF ステートメントを使用しながら、条件テストを最も効果的に使用できます。

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

その結果、SQL コードはこうなります。

```
(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)
```

論理テストを重複して行わない

同様に、冗長な論理チェックを避けるようにします。次の計算により、

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

以下の SQL が生成されます。

```
(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)
```

これは、さらに効率的に記述すると以下のようになります。

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

そして、以下の SQL を生成します。

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

その他の細かな手法

パフォーマンスを改善するには、その他多くの細かな手法があります。以下のヒントを検討してください。

- 日付論理を扱うと、複雑になるおそれがあります。MONTH() と YEAR() のように複数の日付関数を使用して複雑なテストを記述するよりは、DATETRUNC()、DATEADD()、DATEDIFF() のようなビルトイン関数を使用することを考慮してください。データソースに対して生成されるクエリの複雑さを大幅に軽減できます。

- 個別のカウント値は、ほぼすべてのデータソースで最も処理が遅い集計タイプの 1 つです。COUNTD 集計は控えめに使ってください。
- 広範囲 (カスタム SQL ステートメント内など) に影響するパラメーターを使用すると、キャッシュのパフォーマンスに影響を及ぼします。
- 複雑な計算に対してフィルタリングすると、参照元データソースでインデックスが見つからない可能性があります。
- タイムスタンプの詳細レベルが必要な場合にのみ NOW() を使用します。日付レベルの計算には TODAY() を使用します。
- すべての基本的な計算は、ラベルの文字列のような文字の計算でも、参照元データソースにパススルーされます。ラベル (列のヘッダーなど) を作成する必要がありデータソースが非常に大きい場合は、必要なデータが入った 1 レコードだけのシンプルなテキスト/Excel ファイルのデータソースを作成し、大きいデータソースのオーバーヘッドが増加しないようにします。これは、データソースがストアドプロシージャを使用する場合、特に重要です。詳しくは、このセクションを参照してください。

クエリを再検討する

Tableau の非常に強力な機能の 1 つは、抽出接続によるインメモリデータと、ライブ接続によるデータの両方を使える点です。ライブ接続は、データソース内にすでにある処理力を活用できるので、Tableau の非常に強力な機能です。ただし、現在はパフォーマンスをソースデータプラットフォームに依存しているため、データソースへのクエリをできるだけ効率的かつ最適に作成することが非常に重要になっています。

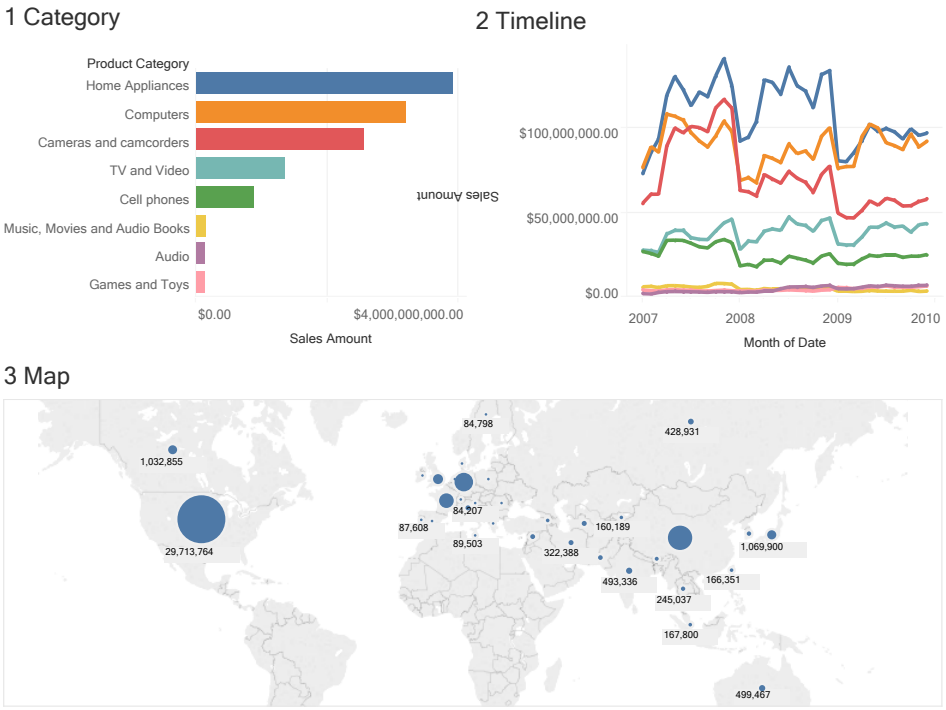
すでに述べてきたように、質問が複雑になるほど、より多くのデータが表示され、ダッシュボードに多くの要素を含めることになります。これらはすべて、データソースの負荷を増やすことを意味します。ワークブックでできる限り効率的に作業できるようにするには、クエリの数を最小限にし、クエリができるだけ効果的に評価されるようにし、クエリで返すデータ量を最小限にし、リクエスト間でデータをできるだけ再利用する必要があります。

自動最適化

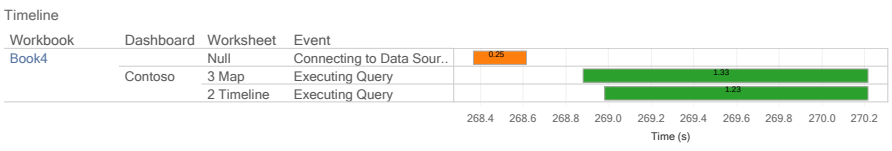
クエリのパフォーマンス向上のために意識して実行できることは多数ありますが、最適化によって自動的に Tableau がワークブックの効率的な実行を保証する方法も多数あります。そのほとんどはユーザーが直接コントロールすることはできず、多くの場合は考える必要すらありません。
ですがそれをよく理解しておけば、パフォーマンス向上に役立てることができます。

パラレル実行 – 複数のクエリを同時に実行する

Tableau は、ソースデータベースの持つ複数クエリ同時実行能力を活用しています。下のダッシュボードを見てください。



このワークブックを Tableau で開くと、次のクエリが表示されます。



ご覧のとおり、クエリを連続して実行すれば 2.66 秒かかることになります。ただしこれを並行して実行することで、最も時間がかかるクエリ (1.33 秒) の時間だけが必要になります。

クエリの並列性のレベルはソースシステム間で異なります。同時クエリを扱う能力はプラットフォームによって差があるためです。既定では、すべてのデータソースに対する並列クエリは、テキスト、Excel、統計ファイル (1 回のクエリを 1 つに制限) を除いて最大 16 まで実行できます。他のデータソースでは、既定より低く制限が設定されていることもあります。詳しくは以下のリンクで確認してください。

> http://kb.tableau.com/articles/HowTo/Configuring-Parallel-Queries-in-Tableau-Desktop?lang=ja-jp

ほとんどの場合、これらの設定を変更することは必要なく、既定の設定のままにしておくことをお勧めします。ただし何らかの必要によって並行性の程度をコントロールする場合は以下の設定が可能です。

- Tableau Server の並列クエリ数のグローバル制限
- SQL Server などの特定のデータソースタイプに対する制限
- 特定のサーバー上の特定のデータソースタイプに対する制限
- 特定のデータベースに接続するときの、特定のサーバー上の特定のデータソースタイプに対する制限

これらの設定は、Tableau Desktop ではユーザーが作成して Windows (`C:\Program Files\Tableau\Tableau 10.0`) または Mac (アプリを右クリックし、[パッケージコンテンツを表示] をクリックしてここにファイルを配置) のアプリフォルダーに保存する connection-configs.xml という xml ファイル内、あるいは Tableau Server では vizqlserver フォルダー (例: `C:\ProgramData\Tableau\TableauServer\data\tabsvc\config\vizqlserver`)–にある config ディレクトリ内で管理されています。この構成ファイルは、すべてのワーカーマシンにあるすべての vizqlserver 構成ディレクトリにコピーしなければなりません。

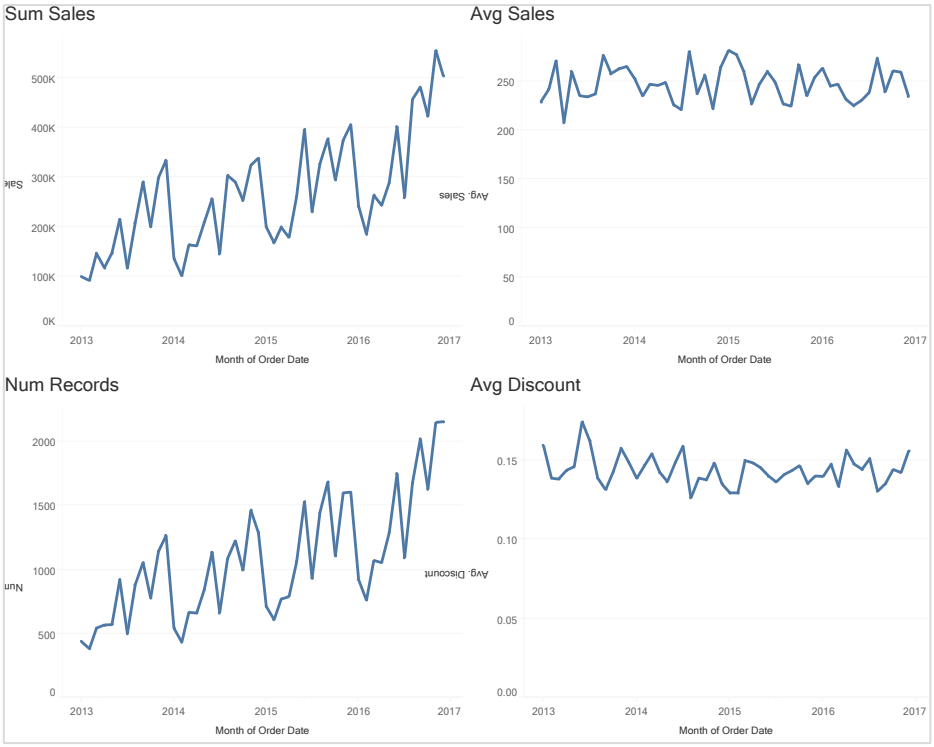
connection-configs.xml ファイルの構文を含む、並列クエリの構成に関する詳細は、こちらをご覧ください。

> http://kb.tableau.com/articles/knowledgebase/parallel-queries-tableau-server?lang=ja-jp

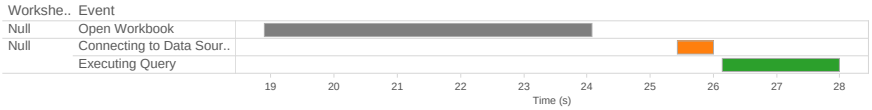
クエリの排除 – 実行するクエリの数を減らす

上記の例では、3 つでなく 2 つのクエリだけを実行しています。Tableau は、複数のクエリを合わせてバッチ化することで、冗長なクエリを排除することができます。Tableau のクエリオプティマイザーは、後に続くクエリがキャッシュの結果を利用できるように、一番複雑なクエリを最初にして実行するクエリを並べ替えます。例では、タイムラインが製品カテゴリーを含み、また売上高の SUM 集計が完全に加法集計であるため、カテゴリーチャートを描くデータはタイムラインワークシートのクエリキャッシュから解決でき、データソースと一致する必要がありません。

クエリオプティマイザーはまた、同じ詳細レベルにある (同じディメンションのセットにより指定される) クエリを探し、すべての要求されたメジャーを返す単一のクエリにまとめます。下のダッシュボードを見てください。



ご覧のように、このダッシュボードには 4 枚のシートが含まれていて、それぞれが一定期間にわたり異なるメジャーを表示しています。これらは、連続する月を使用したデータを表示するので、すべて同じ詳細レベルとなります。データベースのリクエストを 4 つ個別に実行する代わりに、Tableau がそれを 1 つのクエリに結合します。



そのクエリは以下のとおりです。

```
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))
```

ご覧のとおり、この最適化（「クエリ統合」）は全体のパフォーマンスを大幅に改善します。1 つのダッシュボードにある複数のシートには、可能であれば同じ詳細レベルを設定することを検討してください。

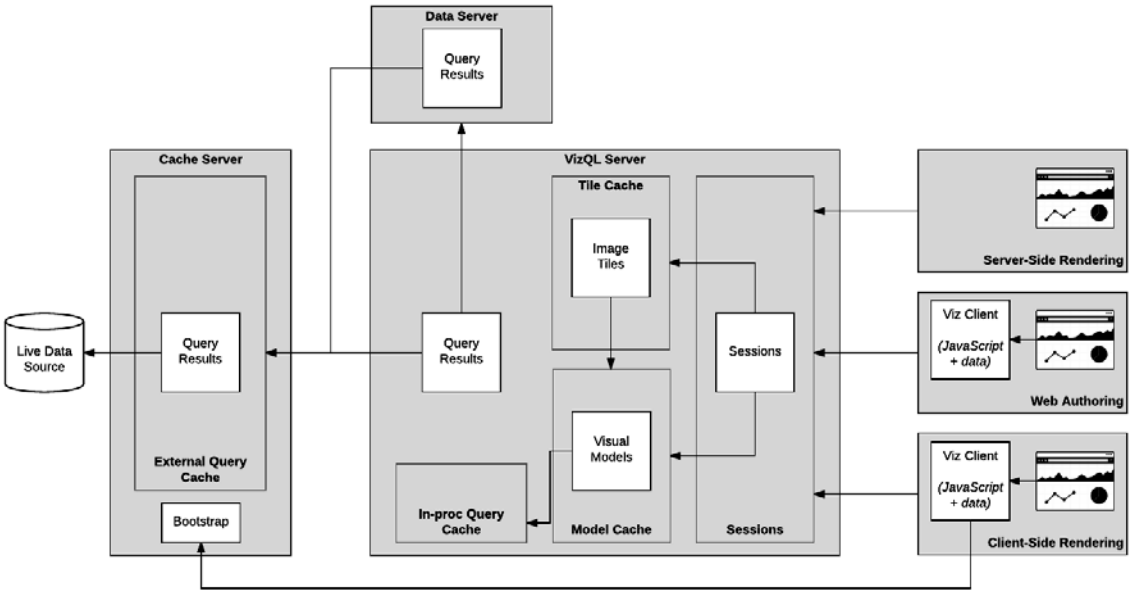
クエリ統合は、TDE データソースに対するクエリについては発生しません。

キャッシュ機能 – クエリをまったく実行しない

クエリの数を減らして実行するよりも良い方法とは何でしょうか。それは、クエリをまったく実行しないことです。Tableau は、Tableau Desktop と Tableau Server の両方で広範なキャッシュ機能を用意しているため、参照元データソースで実行する必要のあるクエリの数を大幅に減らすことができます。

Tableau Server

Tableau Server のキャッシュ機能には複数のレイヤーがあります。



キャッシュの最初のレイヤーは、セッションがクライアント側レンダリングかサーバー側レンダリングのどちらを使用するかによって異なります。この前のセクションで、2 つのレンダリングモデルの詳細、および Tableau がどちらを使用するか決定する方法について確認することができます。

セッションがクライアント側レンダリングを使用している場合、ブラウザでまず行う必要があることは、ビューアーのクライアントを読み込むことです。これには、最初のビューをレンダリングするために JavaScript ライブラリのセットとデータの読み込みが含まれます。これは「ブートストラッピング」と呼ばれるプロセスで、数秒かかります。複数のユーザーがダッシュボードを表示することが予想されるため、後続の表示リクエストの待ち時間を減らすために、ブートストラップの応答はキャッシュされています。各セッションはまずブートストラップキャッシュをチェックして、使える応答がすでに作成されているかを確認します。応答が見つかった場合、ブラウザはキャッシュから応答を読み込んで、最初のビューを非常に速くレンダリングすることができます。ビューアーのクライアントがクライアントブラウザに読み込まれると、一部の操作が完全にローカルデータ内で実行され (例: ハイライティング、ツールヒント)、結果としえt迅速でスムーズなユーザーエクスペリエンスが実現します。

ダッシュボードがサーバー側レンダリング経由で表示される場合は、サーバーはダッシュボードの要素を一連のイメージファイル (「タイル」という) としてレンダリングします。これらのタイルはブラウザに送られ、組み立てられて viz を表示します。このように、複数のユーザーがダッシュボードを表示すると想定しているので、サーバーはそれらのイメージをディスク上にキャッシュしています。リクエストごとにタイルのキャッシュをチェックして、すでにレンダリングされたイメージがあるか確認し、あった場合は単純にキャッシュから読み込みます。タイルのキャッシュがヒットすると応答時間は速

くなり、サーバーの負荷が軽減されます。タイルのキャッシュの効率を高める簡単な方法は、ダッシュボードで固定サイズの設定をしておくことです。

全体的には、クライアント側レンダリングの方がスムーズでユーザーの操作に良く反応し、Tableau Server 上の負荷も少なくて済みます。可能であれば、ワークブックはクライアント側でレンダリングされるよう設計する必要があります。

キャッシュの次のレイヤーは、ビジュアルモデルと呼ばれるものです。ビジュアルモデルは個々のワークシートのレンダリング方法を記述するので、ダッシュボードを表示するときに、使用されているワークシートに 1 つずつ、複数のビジュアルモデルを参照できます。また、ローカルの計算結果（例: 表計算、リファレンスライン、予想、クラスタリングなど）、およびビジュアルなレイアウト（スモールマルチプルとクロス集計を表示する行/列の数、描画する軸目盛り/グリッドラインの数と間隔、表示するマークラベルの位置など）を含んでいます。

ビジュアルモデルは VizQL Server によりインメモリに作成されてキャッシュされ、可能な場合はユーザーセッション間での結果の共有に効果を発揮します。ビジュアルモデルの共有が可能かどうかは、主に以下の要素で決まります。

- Viz の表示エリアサイズ – モデルは viz のサイズが同じセッション間だけで共有できます。ダッシュボードを固定サイズに設定すると、タイルのキャッシュにメリットがあるのと同様に、モデルのキャッシュにもメリットがあります。これにより、さらに再利用の回数が増え、サーバーの負荷が軽減します。
- セクション/フィルターが一致しているか – ユーザーがフィルターやパラメーターを変更したり、ドリルダウン/アップなどをする場合は、モデルはビューの状態が一致するセッションとだけ共有されます。[選択項目を表示] にチェックを入れた状態でワークブックをパブリッシュしないようにしてください。異なるセッションが一致する可能性が低くなります。
- データソースへの接続に使用する認証資格情報 – データソースに接続する際に認証資格情報の入力を求めるメッセージが表示されたら、そのモデルは同じ認証資格情報を持つユーザーセッション間だけで共有されます。この機能はモデルのキャッシュの効率を下げるため、慎重に使用してください。
- ユーザーフィルターが使用されているかどうか – ワークブックにユーザーフィルター、または USERNAME() や ISMEMBEROF() などの関数を含む計算が含まれている場合、そのモデルは他のユーザーセッションとは共有されません。この機能はモデルのキャッシュの効率を大幅に下げるため、慎重に使用してください。

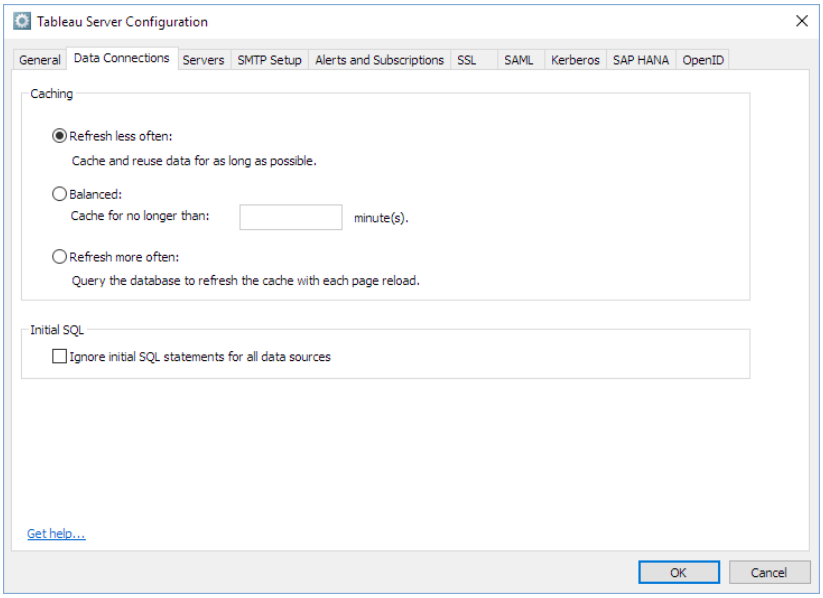
キャッシュの最後のレイヤーは、クエリーキャッシュと呼ばれるものです。これは、後々のクエリで役立つと予測してクエリを実行し、その結果を保存するものです。このキャッシュにヒットすると、データソース内で繰り返しクエリを実行せずにキャッシュのデータを読み込むだけで済むため、非常に効果的です。また状況によっては、別のクエリの結果を使用してクエリに役立てることも可能です。このメリットについては、クエリの排除と統合について述べた前のセクションで記述しています。

このキャッシュは 2 つに分かれていて、1 つは VizQL のプロセス内にあり（「プロセス内キャッシュ」という）、もう 1 つはキャッシュサーバー経由で複数プロセスに共有されています（「外部キャッシュ」という）。あるクエリが、前に同じクエリを実行したことのある VizQL インスタンスに送られると、そのリクエストにはプロセス内キャッシュが利用されます。なお、このキャッシュは各 VizQL プロセスにローカルで、インメモリに維持されています。

これらのプロセス内キャッシュに加え、外部キャッシュもあります。これは VizQL インスタンス間だけでなく、参照元データソースにアクセスする「すべての」プロセス間 (例: バックグラウンダー、データサーバーなど) で共有されます。キャッシュサーバーマネージャーというサービスが、外部キャッシュをクラスタ全体にわたり管理しています。なお、すべてのクエリが外部キャッシュに書き込まれるわけではありません。非常に速いクエリの場合は再実行する方がキャッシュをチェックするよりも速いため、最小クエリ時間のしきい値があります。また、クエリの結果が非常に大きい場合は、キャッシュサーバーへの書き込みが効率的でないことがあるため、最大サイズのしきい値もあります。

外部キャッシュは、複数の VizQL インスタンスがある場合に展開すると、キャッシュの効果を大幅に向上します。揮発性のプロセス内キャッシュとは対象的に、外部キャッシュには永続性があり (キャッシュサーバーがディスクにデータを書き込む)、サービスのインスタンス化とインスタンスの間に残ります。

キャッシュの効果を最大限にするには、キャッシュをできるだけ長く維持するように Tableau Server のインストール設定でを調整します。



サーバーの容量が利用可能な分だけ、キャッシュのサイズを増やすことができます。この設定を増やすことに対するペナルティはないため、十分に RAM がある場合はこの数値を上げることで、コンテンツがキャッシュから押し出されることがかなり回避できます。キャッシュの効果は、ログファイルまたは TabMon 経由で監視することができます。

- **モデルキャッシュ** – 既定の設定では VizQL インスタンスあたり 60 モデルをキャッシュします。この設定は、tabadmin コマンドで調整できます。十分使用できる RAM がある場合は、サーバー上にパブリッシュされたビューの数に合わせて増やすことができます。
    - tabadmin set vizqlserver.modelcachesize 200
- **プロセス内キャッシュ** – 既定の設定では VizQL インスタンスあたり 512 MB のクエリ結果をキャッシュします。これは大量に聞こえませんが、集計クエリからのクエリ結果をキャッシュしていることを思い出してください。この設定は、tabadmin コマンドで調整できます。
    - tabadmin set vizqlserver.querycachesize 1024

キャッシュサーバーのインスタンスをさらに追加することで、外部クエリキャッシュの容量とスループットを増やすことも可能です。各 VizQL Server インスタンスごとに、キャッシュサーバーのインスタンスを 1 つ実行することをお勧めします。

- **外部キャッシュ** – 既定の設定ではキャッシュサーバーの各インスタンスに対し 512 MB のクエリ結果をキャッシュします。

最後に、共有したキャッシュには嬉しい副作用もあります。最初の表示が遅いことが分かっているワークブックに対してサブスクリプションを実行しておくことで、そのワークブックのためのキャッシュが準備できるのです。よく利用するユーザーが出勤する前に、朝一番にこれをしておくと、ワークブックが実行されクエリ結果がキャッシュに読み込まれます。クエリ結果がキャッシュから押し出されたり中断時に無効になったりしないと想定すれば、ユーザーがワークブックを表示するときにキャッシュがヒットして、最初の表示にかかる時間を短縮できます。

Tableau Desktop

Tableau Desktop はシングルユーザーアプリケーションで、複数のセッションを管理する必要がないため、Tableau Server よりもずっと簡単にキャッシュできます。Tableau Desktop にはクエリのキャッシュだけが存在します。

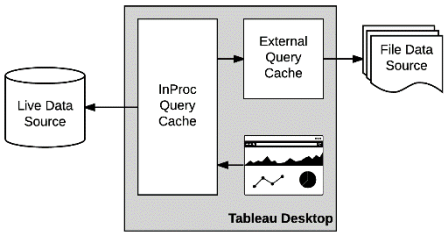


Tableau Server のように、プロセス内キャッシュと外部キャッシュがあります。プロセス内キャッシュは揮発性でメモリーベースで、すべてのデータソース接続に使用しますが、外部キャッシュはファイルベースのデータソースのみに使用します (例: データ抽出、Excel、Access、テキストファイル、統計ファイルなど)。Tableau Server 同様、外部キャッシュは永続的です。つまり、Tableau Desktop 内にキャッシュの結果がセッション間で保持されるため、ファイルデータソースを使用している場合は、Tableau Desktop を再起動して再度使用するときもキャッシュを活用できます。

外部キャッシュは Windows の場合は `%LOCALAPPDATA%\Tableau\Caching`、Mac の場合は `~/Library/Caches/com.tableau/` に保存されています。既定では合計 750 MB に制限されており、ユーザーがデータソースを強制リフレッシュする (F5 または ⌘R を押す) と無効になります。

最後に、Tableau Desktop では、ファイルをパッケージドワークブックとして保存するとき、外部のクエリキャッシュデータが含まれます。これにより Tableau Desktop と Tableau Reader は、大きなデータソースファイルをバックグラウンドで継続して開きながら、ワークブックの初期のビューを高速でレンダリングすることができます。エンドユーザーにとっては、これによってワークブックを開くときの反応が劇的に向上します。なお、キャッシュデータは、データがソースデータファイルのサイズの 10% 以下である場合のみ含まれ、相対日付フィルターを使用するクエリは含まれません。

接続の遅延

Tableau 10 より前は、複数のデータソースがあるワークブックを開くとき、まず認証資格情報のないデータソース (すなわちユーザー名/パスワードの入力を必要としないデータソース) にすべて接続

していました。これは、ユーザーが表示しているシート/ダッシュボード/ストーリーで使わないデータソースへの接続に時間を費やしていた可能性があったことを意味します。

Tableau 10 では、ユーザーが選択したシート/ダッシュボード/ストーリーで表示するのに必要なデータソースだけに接続するようになっています。この変更によって、タブ付きのビューを備えたワークブックの読み込み時間（Tableau Desktop 上）が短くなるので、ユーザーはデータ分析をより早く始めることができます。

結合

基本的に、Tableau で複数の表を操作する場合は、データ接続ウィンドウで結合を定義するアプローチを推奨します。このとき、定義するのは特定のクエリではなく、それぞれの表が互いにどう関連するかだけです。フィールドを viz にドラッグ&ドロップすると、Tableau はこの情報を使って、要求されたデータだけを取得するのに必要な特定のクエリを作成します。

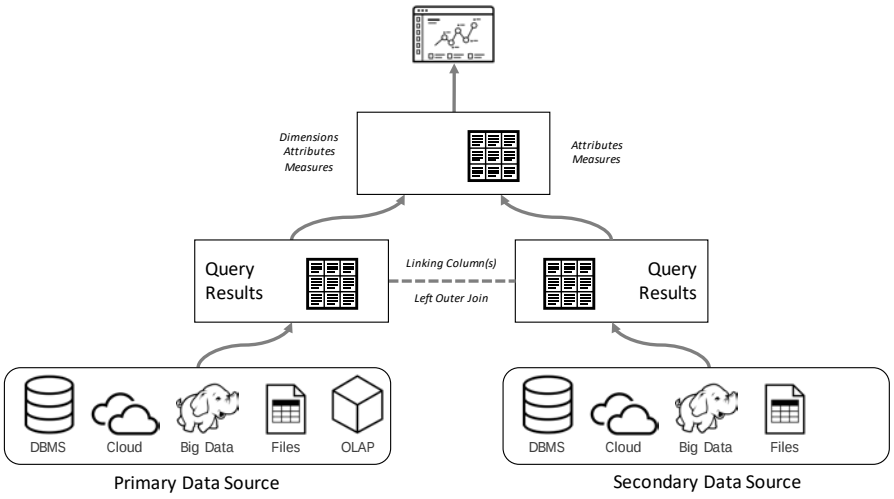
パフォーマンスを良くするための一般的なルールとして、結合した表は特定のワークシート/ビジュアライゼーションに必要な表のみを含めて最小化する必要があります。データ接続とワークシートの複雑さによっては、データ接続を各ワークシートで別にして、特定の結合パターンをそれらのシートに作成するのも価値があるでしょう。

結合のパフォーマンスは、表の間で適切な制約のある結合の方が良くなります。この制約で Tableau Desktop にはクエリを簡素化する自由が与えられ、特定の質問のサブセット（凡例、フィルターカード）の答えを出すに必要な表だけを対象にできます。

ブレンディング

Tableau でデータ表を結合するかブレンドするかのどちらかに決める際には、データがどこから来ているか、データ接続の数、データ内のレコード数などを考慮します。

ブレンディングを使用する場合、ブレンディングのパフォーマンスに影響する主な要素の 1 つは、各データソースのレコード数でなく、むしろ 2 つのデータセットをリンクしているブレンド対象のフィールドの密度です。ブレンディングでは、リンクしているフィールドのレベルで両方のデータソースからデータをクエリし、両方のクエリの検索結果を Tableau の ローカルメモリで統合します。



そのため、一意の値が多いと大容量のメモリが必要になります。ブレンディング時のメモリ不足を避けるには、64 ビットバージョンの Tableau Desktop の使用を推奨します。それでも、この性質のブレンディングには計算に時間がかかる傾向があります。

ブレンドの際に推奨されるベストプラクティスは、一意の値（オーダー ID、顧客 ID、正確な日時など）が多いディメンションでのブレンディングを避けることです。

プライマリグループおよび別名

2 つのデータソース（一方は「ファクト」のレコードを含み、他方はディメンション属性を含む）をブレンドする必要が生じた場合は、プライマリグループまたは別名を作成すると、パフォーマンスが向上する可能性があります。

これは、プライマリデータソース内でグループおよび/または別名を作成し、セカンダリデータソース内にリストされている属性を反映すると可能になります。プライマリグループは 1 対多のリレーションシップに、プライマリ別名は 1 対 1 のリレーションシップに使用されます。そうすると、プライマリデータソースから関連するオブジェクトを使用することができ、表示時にブレンディングが必要なくなります。

以下の例では、次の 3 つの表について考えます。

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

プライマリグループが役立つのは、プライマリデータソースのディメンションメンバーに対して 1 対多でマッピングされる属性が、セカンダリデータソースに含まれている場合です。上記のデータから導き出したい情報が次の情報だとします。

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

ブレンディングによってもこの表を作成できますが、これまで説明してきたとおり、ID の数が多ければパフォーマンスは非常に低下します。

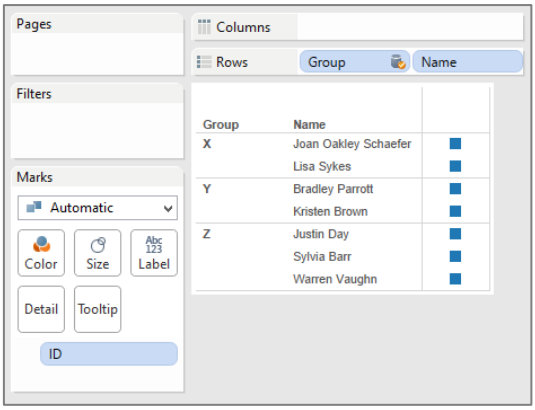
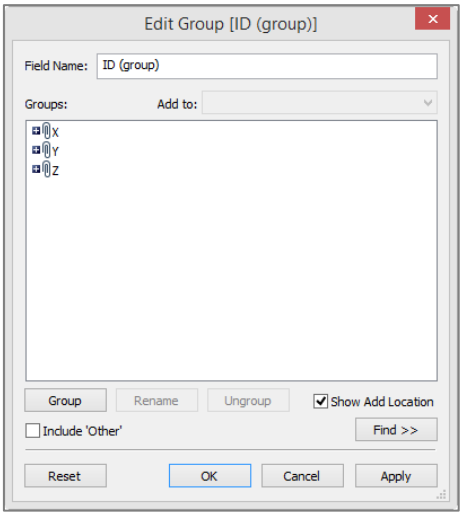
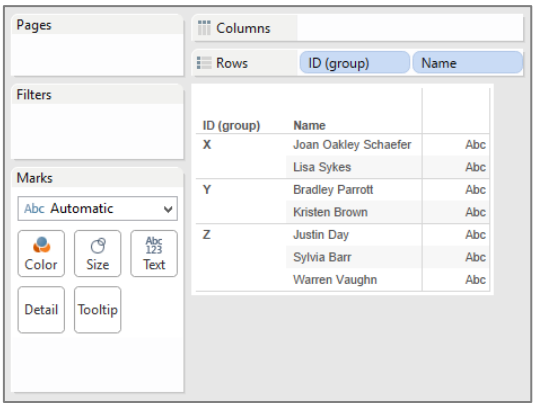


Tableau では、[グループ] フィールドを右クリックして [プライマリグループの作成] を選択すると、リンクしているフィールド (この場合は ID) を選択したセカンダリデータソースのディメンション–(この場合はグループ) にマッピングするグループオブジェクトが、プライマリデータソースに作成されます。



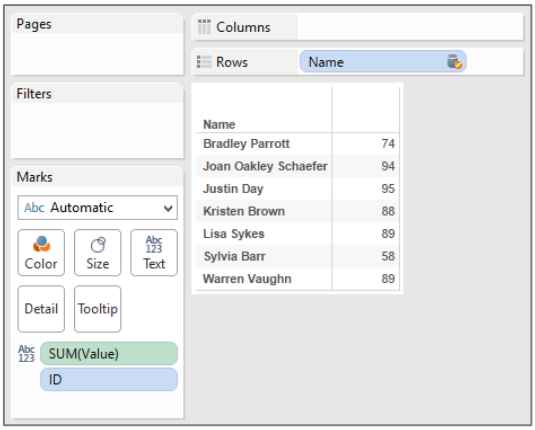
これで、ブレンディングの必要なくこの表を再作成できるようになりました。



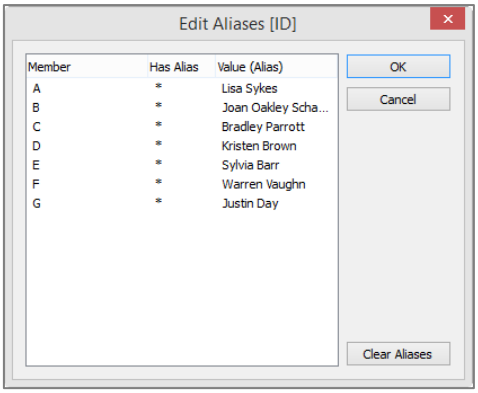
プライマリ別名が役立つのは、プライマリデータソースのディメンションメンバーに対して 1 対 1 でマッピングされる属性が、セカンダリデータソースに含まれている場合です。上記のデータから導き出したい情報が次の情報だとします。

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

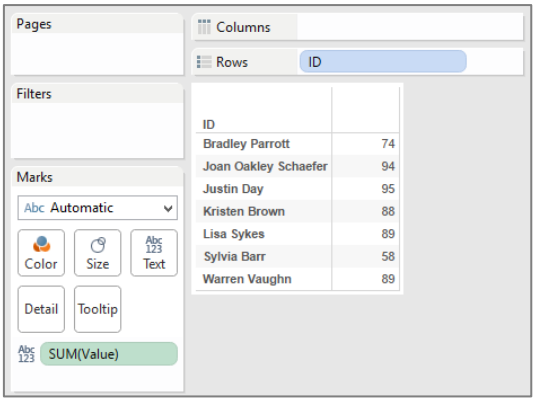
2 つのデータソースをブレンドしてもこの表を作成できますが、これまで説明してきたとおり、ID の数が多ければパフォーマンスは非常に低下します。



[Name] フィールドを右クリックし Edit Primary Aliases] を選択すると、[Name] フィールドを別名値として [ID] フィールドに一回限りのマッピングができます。



これで、ブレンディングの必要がないためずっと速く、必要なビジュアライゼーションを作成できるようになりました。



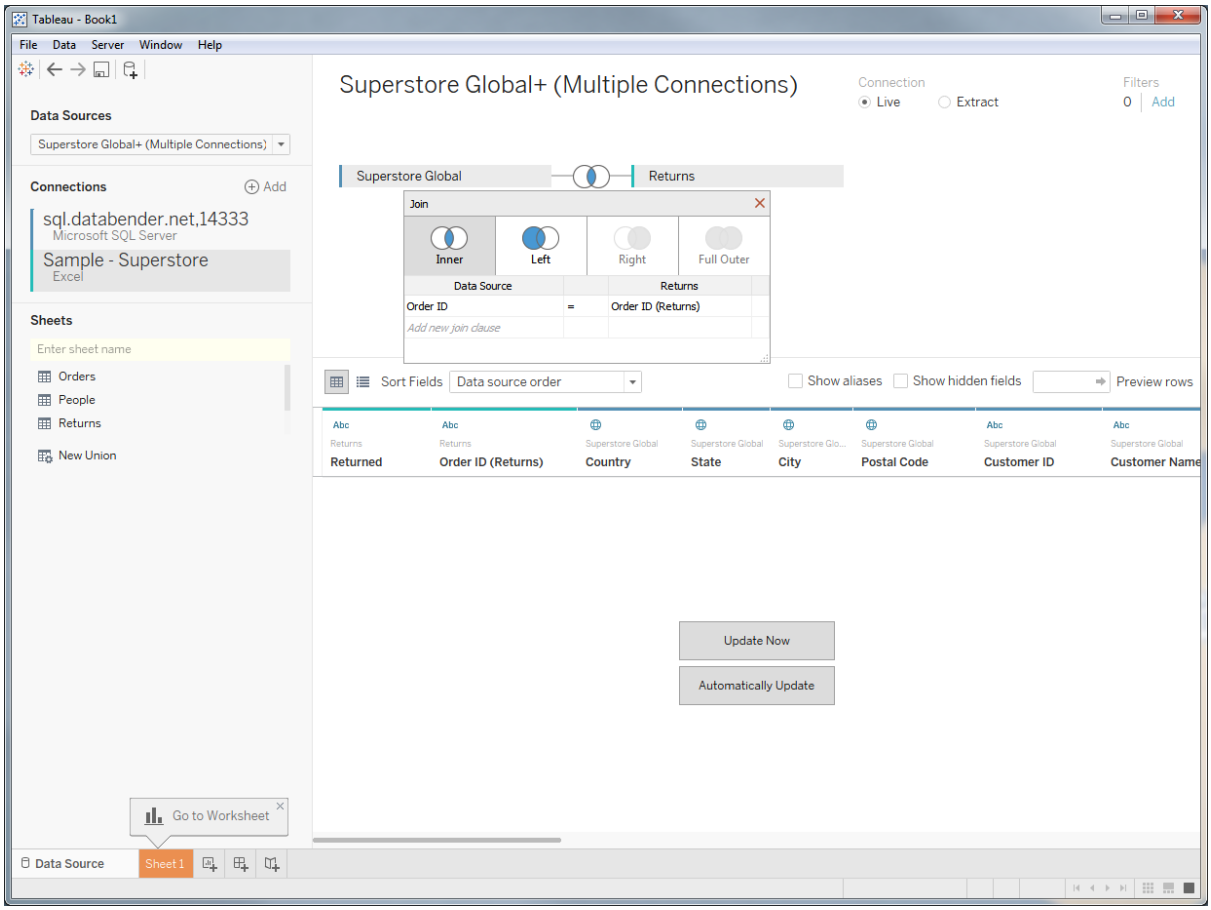
プライマリグループもプライマリ別名も動的ではなく、データが変更されたら更新する必要があります。そのため、頻繁に更新されるデータにとっては優れたソリューションとは言えませんが、すばやくマッピングをする必要がある場合は、時間やリソースがかかるブレンディングが必要なくなる可能性もあります。

詳細情報については、以下のナレッジベース記事をご覧ください。

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

データの統合

データの統合は Tableau 10 で導入された新機能で、これによりデータソースは、複数の、異機種環境にある可能性があるデータ接続からデータを結合することができます。



データ統合とブレンディングの主な違いは、データ統合が行レベルの結合であるのに対し、データブレンディングは各データソースからの集計結果のセット上で機能する点です。つまり、データ統合は参照元データソースのサイズに影響を受けやすいということで、主に次の 4 点について考慮が必要です。

- 移動するデータが多いほど時間がかかる – データは行レベルで各データ接続から抽出されるため、データの量は大きな要素です。
- データの移動先が遠いほど時間がかかる – 接続の待ち時間が長いデータソースに結合する場合は、パフォーマンスに影響します。
- データフローが遅いほど時間がかかる – 帯域幅に制約がある接続先にあるデータソースと結合する場合は、パフォーマンスが悪くなります。
- マッチさせるものが多いほど時間がかかる – 1 点目同様、パフォーマンスは結合するレコードの数に影響を受けます。

Tableau 10 では、データ統合は抽出されたデータソースとのみ使用可能です。つまり、データ接続の結果は TDE ファイルに抽出しなければなりません。そうすると、抽出されたデータソースは Tableau Server にパブリッシュされて他のユーザーが使用できるようになります。

Tableau 10 ではライブデータソース上でデータ統合をすることも、パブリッシュされたデータソースに対してデータ統合することもできません。この機能は、今後の製品強化として追加していきたいと考えています。

カスタム SQL

Tableau を使い始めたばかりのお客様は、手書きの SQL ステートメントを使用したデータソースを作成するなど、旧世界の技術をワークブックに持ち込もうとすることがあります。多くの場合、これは逆効果となります。Tableau は、表同士の結合リレーションシップを簡単に定義するときにずっと効率的なクエリを生成でき、作成されたビューに合わせた SQL をクエリエンジンに記述させることができるからです。ただし、データ接続ウィンドウ内で結合を指定しても、自分のデータ内のリレーションシップを定義できる柔軟性を提供できないことがあります。

手書きの SQL ステートメントを使用したデータ接続の作成は非常に強力ですが、コミック「スパイダーマン」で知られる戒めにあるように、「大いなる力には大いなる責任が伴う」ものです。場合によっては、カスタムの SQL は実際パフォーマンスの低下をもたらすことがあります。その理由は、結合の定義とは対象的に、カスタム SQL は決して分解されず常に自動的に実行されるからです。これは、結合選択が起こることがなく、データベースは 1 つの列のためだけにクエリ全体を処理させられるという結果になる可能性があることを意味します。たとえば、以下のようになります。

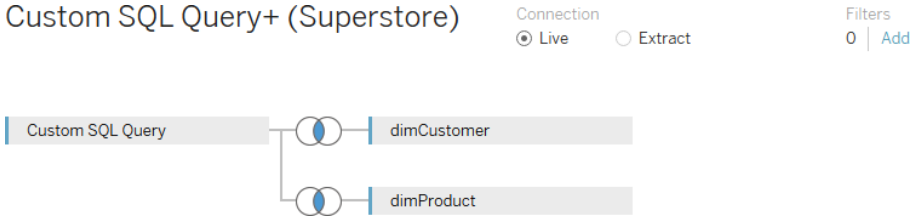
```
SELECT SUM([TableauSQL].[Sales])
FROM (
  SELECT [OrdersFact].[Order ID] AS [Order ID],
       [OrdersFact].[Date ID] AS [Date ID],
       [OrdersFact].[Customer ID] AS [Customer ID],
       [OrdersFact].[Place ID] AS [Place ID],
       [OrdersFact].[Product ID] AS [Product ID],
       [OrdersFact].[Delivery 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)
```

重要なのは、カスタム SQL ステートメントが不必要な句を含まないようにすることです。たとえば、クエリに GROUP BY または ORDER BY 句がある場合、Tableau がビジュアライゼーションの構造に基づいて自分の句を作るため、通常これはオーバーヘッドとなります。できれば、これはクエリから削除してください。

カスタム SQL を使用する際は、データ抽出と併用することをお勧めします。こうすると、アトミッククエリは一度のみ実行され（データをデータ抽出に読み込む）、それ以降 Tableau で行うすべての分析は、そのデータ抽出に対して動的で最適化されたクエリを使用して行われます。もちろん、すべてのルールには例外があり、これもその 1 つです。カスタム SQL を使用する際は、自分の SQL にパラメーターが含まれていない場合に限り、データ抽出を作成します。

カスタム SQL でパラメーターを使用すると、基本クエリがさらに動的になる可能性があるため（例: パラメーターを使用するフィルター句が適切に評価される）、ライブ接続のパフォーマンスが向上する場合もあります。また、「TOP」や「SAMPLE」など、データベースから返されるデータ量を制限するパフォーマンスリミッターの値を渡すのにも使用できます。ただしデータ抽出を使用する場合は、パラメーターを変更するたびにパージされ再生成されるので、速度が低下することがあります。また、パラメーターはリテラル値を渡す場合にのみ使用でき、「SELECT」句や「FROM」句を動的に変更するためには使用できないことに注意してください。

最後に、表をカスタム SQL に結合することも可能です。



これにより、トータルスキーマのサブセットを参照する、より具体的なカスタム SQL を記述できます。そうすると、ここから他の表への結合が（潜在的に）通常の表結合のように選択され、より効率的なクエリを作成できます。

カスタム SQL に代わる選択肢

カスタム SQL を Tableau で直接使用する代わりに、クエリを参照元データベースに移動したほうが良い場合があります。多くの場合、これによりデータソースがクエリを解析する効率が上がったり、複雑なクエリは 1 回だけ実行すれば済む場合もあるため、パフォーマンスが向上します。複数のワークブックとデータソースの間で共有する定義が 1 つだけになるので、優れた管理方法ともなります。

これを実施する方法はいくつかあります。

ビュー

ほぼすべての DMBS は、ビューの概念をサポートしています。ビューとは、データベースクエリの結果を表すバーチャルな表のことです。一部のデータベースシステムでは、単にカスタム SQL ステートメントからクエリを取り出し、ビューとしてデータベース内でインスタンス化を行うだけで、パフォーマンスが大きく向上します。その理由は、クエリが外部の SELECT ステートメントに包含されている場合よりも、クエリオプティマイザーがより優れた実行計画を作成できるからです。詳しくは、以下のコミュニティのディスカッションをご覧ください。

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

カスタムクエリの論理は Tableau ワークブック内ではなくビュー内に定義することでも、複数のワークブックとデータソース間での再使用が可能になります。さらに、ビューのクエリ結果を事前に計算してキャッシュとしてビューやスナップショットに保存し、クエリ時間のレスポンスを速くするというコンセプトをサポートする DBMS は多数あります。これはサマリー表（以下参照）に似ていますが、DBMS によって自動的に維持されます。

ストアドプロシージャ

ストアドプロシージャはビューと似ていますが、ずっと複雑な論理を含み、何段階ものデータ準備を実行できる可能性があります。パラメーター化することも可能で、ユーザーの入力に基づいて対象のデータのセットを返すことができます。

ストアドプロシージャは Sybase ASE、SQL Server、Teradata でサポートされています。1 つの viz に対して何度もストアドプロシージャを実行しないように、Tableau はストアドプロシージャを実行すると結果セットをデータベースの一時表に書き込みます。すると、viz の実際のクエリは一時表に対して実行されます。ストアドプロシージャの実行と一時表の値の設定は、ワークブックを初めて開くときと、保存されたプロセスのパラメーターが変更されるたびに行われます。このプロセスは時間がかかり、操作が遅くなることがあります。

パラメーター化されたストアドプロシージャの結果をデータ抽出内に抽出する場合、その抽出はパラメーターの値を変更する度にパージされ変更されます。

ストアドプロシージャの使用に関する詳細は、Tableau Online ヘルプをご参照ください。

> http://onlinehelp.tableau.com/current/pro/desktop/ja-jp/help.htm#connect_basic_stored_procedures.html

サマリー表

非常に規模が大きく詳細なデータセットがあり、クエリを実行する際に通常サマリーを作成している場合（たとえば、トランザクションを個別に保存してあるが、通常は日付、地域、顧客、製品別のサマリーを作成して使用している場合） は、サマリー表を作成し、それに対して Tableau を使用してクエリ速度を上げることを検討してみてください。

メモ – Tableau データ抽出を使用して集計データ抽出を作成することで、同様の結果が得られます。詳しくは、抽出に関するセクションをご覧ください。

初期 SQL

カスタム SQL の代替方法として、初期 SQL ブロックにカスタム SQL ステートメントを使用することもできます（データソースでサポートされている場合）。この方法を使用すると、後でクエリで選択される一時表を作成できます。カスタム SQL はビジュアライゼーションが変更されるたびに実行されますが、初期 SQL はワークブックを開くときに一度しか実行されないため、これによりパフォーマンスが大幅に向上することもあります。

Tableau Server で、管理者は初期 SQL が実行しないように制限を設定できることに注意してください。ワークブックをパブリッシュして他のユーザーと共有することを計画している場合は、ご利用の環境でこれが問題ないことを確認する必要があるでしょう。

初期 SQL については、詳しくはオンラインドキュメントをご覧ください。

> http://onlinehelp.tableau.com/current/pro/desktop/ja-jp/help.htm#connect_basic_initialsql.html

データを再検討する

Tableau のパワフルな機能の 1 つは、プラットフォームを問わずデータに接続できることです。Tableau が接続できるプラットフォームは、大まかに分けて以下のいずれかに特徴付けられます。

- ファイルベースのデータソース – Excel や CSV など
- リレーショナルデータベースのデータソース – Oracle、Teradata、SQL Server など、ならびに、HP Vertica や IBM Netezza などの特別な分析ツール
- OLAP データソース – Microsoft Analysis Services、Oracle Essbase など
- 「ビッグデータ」のデータソース – Hadoop など
- クラウドベースのデータソース – Salesforce や Google など

データソースの種類ごとに長所と短所があり、それぞれが独自に取り扱われます。

Tableau Desktop は Windows と Mac OS X の両方でサポートされており、Mac でサポートされるデータソースは Windows と同じではないことに注意してください。プラットフォーム間での相違をできるだけなくすことは、Tableau が今後目指す取り組みですが、現時点では 1 つのプラットフォームでのみサポートされているデータソースがあります。

基本的なアドバイス

ネイティブのドライバーを使用する

Tableau 10 はネイティブで 40 を超えるデータソースへの接続をサポートしています。すなわち、Tableau には、技術と機能と、これらのデータソースに合わせた最適化が備わっているということです。これらの接続性については、エンジニアリングおよびテストアクティビティによって、Tableau が提供する最も安定した接続であることが保証されています。

Tableau はまた、ネイティブ接続のリストにないデータソースにアクセスするために、汎用の ODBC をサポートしています。公式に定義された標準として、多くのデータベースベンダーが 自社のデータベースで ODBC ドライバーを利用可能にしており、Tableau もこの ODBC ドライバーをデータ接続に使用することができます。ODBC 標準の機能は、各データベースベンダーによって解釈や導入の方法が異なることがあります。場合によっては、特定のドライバーを使用して作業を続けるために、Tableau がデータ抽出の作成を推奨または要求します。ODBC ドライバーとデータベースの中には、Tableau が接続できないものもあります。

クエリを実行するデータソースに対するネイティブドライバーがあれば、基本的にその方がパフォーマンスが良いため、ODBC 接続で優先的に使用してください。ODBC 接続は Windows のみで利用可能であることにも注意してください。

できるだけデータの近くでテストする

前述のとおり、基本的原理では、データソースでクエリの実行が遅くなっている場合、Tableau の反応も遅くなります。データソースの生のパフォーマンスをテストする良い方法は、(可能なら) データソースが常駐してクエリを実行するマシンに Tableau Desktop をインストールすることです。この方法では、ネットワークの帯域幅や待ち時間といった要素がパフォーマンスから排除され、ユーザーはデータソースにあるクエリの生のパフォーマンスをよく理解できます。さらに、データソースに対して DNS 名でなくローカルホスト名を使用することで、名前解決やプロキシサーバーの遅さといった環境要因がパフォーマンスをさらに低下させているかどうかが分かります。

データソース

ファイル

このカテゴリでは、最も一般的なものとして CSV、Excel スプレッドシート、MS Access といったファイルベースのデータ形式をすべてカバーしますが、統計プラットフォームの SPSS、SAS、R からのデータファイルも含まれます。ビジネスユーザーはこの形式をよく使用しますが、その理由は、これがレポートの実行によっても抽出クエリの実行によっても「管理された」データセットからデータを取り出す一般的な方法だからです。

一般的にはこれが、ファイルベースのデータソースを Tableau の高速データエンジンにインポートする際のベストプラクティスです。これによって、クエリの実行がかなり速くなり、またデータ値を保存するファイルサイズをずっと小さくできます。ただし、ファイルが小さい場合や、変化を続けるデータを反映させるためにリアルタイムにファイルに接続する必要がある場合は、ライブ接続も可能です。

シャドー抽出

非レガシーの Excel/テキストまたは統計ファイルに接続するとき、Tableau は接続プロセスの一部として透過的に抽出ファイルを作成します。この機能はシャドー抽出と呼ばれ、ファイルで直接クエリを実行するよりもずっと速くデータを扱うことができます。

大きなファイルを初めて使うとき、データプレビューペインの読み込みに数秒かかることに気付くことがあります。これは、Tableau がファイルからデータを抽出し、シャドー抽出ファイルに書き込んでいるためです。既定ではこれらのファイルは、データファイルのパス名と最終更新日に基づいてハッシュ化された名前で `C:\Users\<username>\AppData\Local\Tableau\Caching\TemporaryExtracts` に作成されます。Tableau は、シャドー抽出のためにファイルデータソースのうち最も最近使用された 5 件をこのディレクトリに保持し、新しいファイルが作成されると最も前に使用されたものを削除します。シャドー抽出のあるファイルを次に再使用する場合、Tableau は抽出ファイルを開くだけで、データのプレビューがほぼ瞬時に表示されます。

シャドー抽出ファイルには、参照元データと標準の Tableau 抽出に類似するその他の情報が含まれていますが、シャドー抽出ファイルは異なる形式で保存され (拡張子は .ttde)、Tableau 抽出ファイルと同じ方法で使用することはできません。

Excel とテキストファイルへのレガシーコネクター

Tableau の旧バージョンでは、Excel とテキストファイルへの接続には Microsoft の JET データエンジンドライバーを使用していました。Tableau 8.2 以降は、より優れたパフォーマンスを実現し、より大きく複雑なファイルを扱うことのできるネイティブドライバーを既定にしています。ただし、レガシードライバーの使用が好まれる状況もあります。たとえばカスタム SQL を使用したいときなどです。
このような場合は、レガシーの JET ドライバーに戻るというオプションもあります。なお、
MS Access ファイルの読み込みには今でも JET ドライバーを使用しています。

2 つのドライバーの違いについては、詳しいリストをここでご覧いただけます。

> http://onlinehelp.tableau.com/current/pro/desktop/ja-jp/help.htm#upgrading_connection.html

JET ドライバーは Mac OS で使用できないため、Mac 版 Tableau Desktop では MS Access の読み取りをサポートせず、Excel と テキストファイルのレガシー接続オプションも提供していません。

リレーショナルデータベース

Tableau ユーザーにはリレーショナルデータソースが最も一般的なタイプのデータソースであり、Tableau では幅広いプラットフォーム用にネイティブドライバーをご用意しています。行ベースや列ベースのもの、パーソナル版やエンタープライズ版などがあり、アクセスにはネイティブドライバーや汎用 ODBC を使用します。厳密にいえば、Map-Reduce データソースも Hive や Impala などの SQL アクセスレイヤーを介してアクセスするため、このカテゴリーに含まれますが、これについては後の「ビッグデータ」のセクションで詳しく説明します。

リレーショナルデータベースシステム (RDBMS) でのクエリ速度に影響する内的要因は多数あります。これらの変更や調整には通常データベース管理者のサポートが必要ですが、大幅なパフォーマンス向上につながる可能性があります。

行ベースと列ベース

RDBMS システムは、主なカテゴリとして行ベースと列ベースの 2 種類で提供されています。行ベースのストレージレイアウトは、インタラクティブなトランザクションで大量に読み込まれる OLTP 型のワークロードに適しています。列ベースのストレージレイアウトは、高度に複雑なクエリを大きなデータセット上に含むことの多い分析型のワークロード (例: データウェアハウス) に適しています。

今日では、高機能分析ソリューションの多くは列ベースの RDBMS を使用しており、そのようなソリューションを使用するとクエリがより速く実行できる場合があります。Tableau がサポートしている列ベースのデータベースの例には、Actian Vector、Amazon Redshift、HP Vertica、IBM Netezza、Pivotal Greenplum、SAP HANA、SAP Sybase IQ があります。

インターフェイスとしての SQL

従来の RDBMS 技術をベースとしていなくても、SQL ベースのインターフェイスを提供することからリレーショナルソースのように見えるシステムが多数あります。Tableau では、複数の NoSQL プラットフォーム (例: MarkLogic、DataStax、MongoDB など)、また Kognito や AtScale、JethroData などのクエリ高速化プラットフォームに対応しています。

インデックス

データベースに正しいインデックスを付けることは、クエリのパフォーマンスを向上するうえで不可欠です。

- 表の結合の一部を構成する列に必ずインデックスを付けてください。
- フィルターで使用されている列に必ずインデックスを付けてください。
- データベースによっては、不連続の日付フィルターを使用すると、クエリが日付列や日付時刻列のインデックスを使用しないことがあります。この点についてはフィルターに関するセクションで詳しく述べますが、日付範囲フィルターを使うと日付インデックスが確実に使用されます。たとえば、`YEAR([DateDim])=2010` とする代わりに、フィルターを `[DateDim] >= #2010-01-01# and [DateDim] <= #2010-12-31#)` と表します。
- クエリ最適化ツールで高品質のクエリプランを作成できるようにするには、データで統計を有効にしておきます。

DBMS 環境では多くの場合、クエリを確認して有用なインデックスを推奨する管理ツールがあります。

NULL

ディメンション列に NULL 値があると、クエリの効率を下げることがあります。国別にトップ 10 製品の総売上高を表示する viz を考えてみましょう。Tableau は、まずトップ 10 製品を見つけるサブクエリを作成してから国別のクエリに戻って結合し、最終結果を出します。

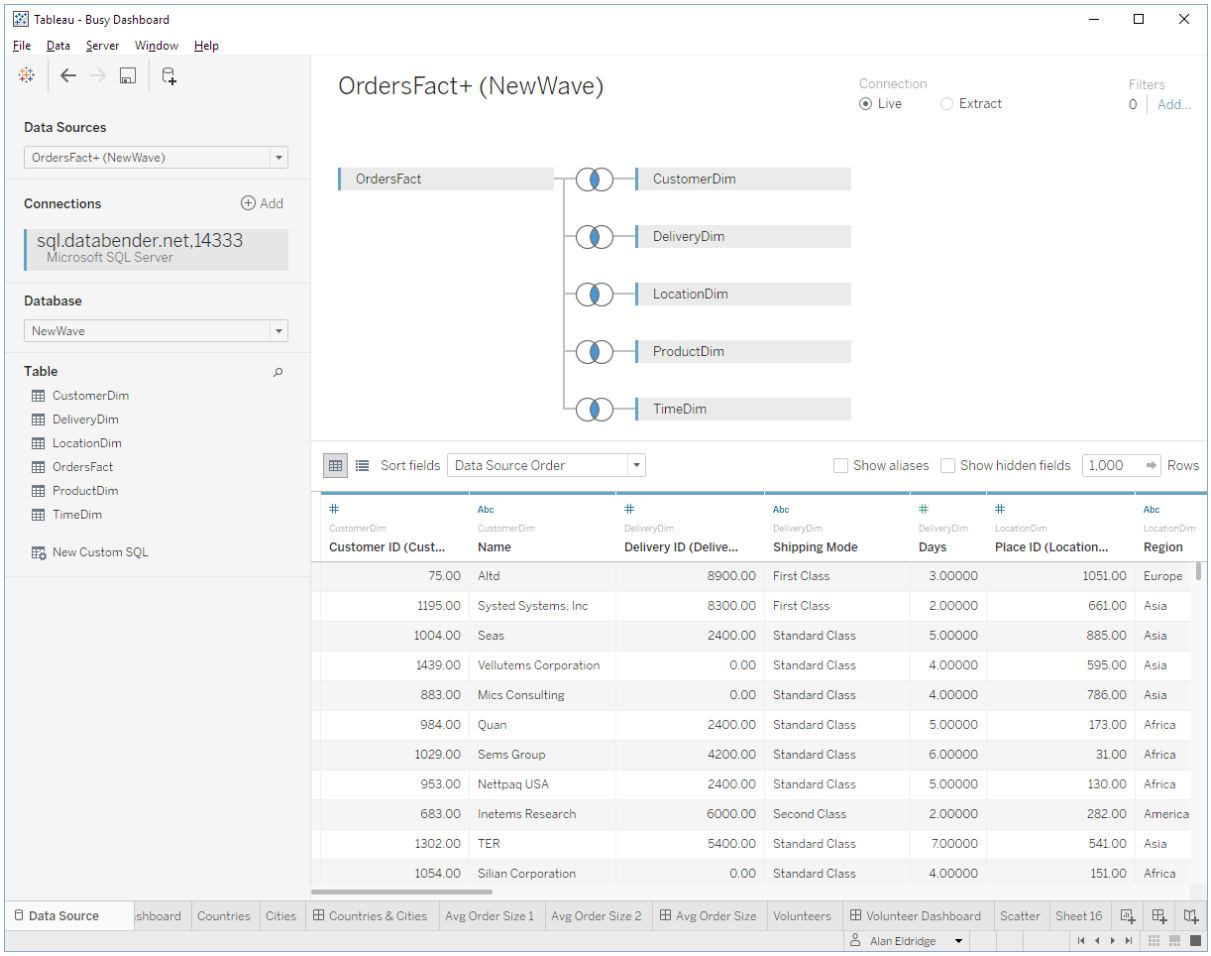
製品の列が ALLOW NULL に設定されている場合、NULL がサブクエリで返されたトップ 10 製品に入っているかチェックするクエリを実行する必要があります。NULL が入っている場合は、NULL を除外する結合を実行する必要がありますが、これは通常の結合よりコンピューターの負荷が高くなります。製品の列が NOT NULL に設定されている場合、サブクエリの結果に NULL はありえないことが分かっているので、「NULL のチェック」クエリをスキップして通常の結合を実行します。

上記の理由から、可能な場合はディメンション列を NOT NULL と定義する必要があります。

参照整合性

1 つのデータソースに含まれる複数の表を結合する場合、Tableau には「結合選択」と呼ばれる素晴らしい (そして通常ユーザーには見えない) 機能があります。データベースサーバーで結合を処理するには時間もリソースも必要なため、常にデータソースで宣言した結合をすべて列挙するようなことは避けたほうが無難です。結合選択によって、結合で定義されるすべての表ではなく、関連性のある表のみにクエリを実行することができます。

以下のシナリオで、小さいスタースキーマに含まれる複数の表を結合した場合を考えてみましょう。



結合選択を使用すると、[Sales] メジャーをダブルクリックすることで以下のクエリが作成されます。

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

結合選択を使用しないと、ずっと効率の悪いクエリが作成されます。

```
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 ()
```

最初から正しいメジャーの合計が計算されるようにするには、すべてのディメンション表を結合する必要があります。たとえば、ファクト表に 2008～2012 年のデータが含まれているのに、ディメンション表には 2010～2012 年のデータしか含まれていない場合、得られる SUM([Sales]) はタイムテーブルが含まれる場合と含まれない場合とで異なる可能性があります。

これまで、DBMS がルールを強制していた場合は「ハード」な参照整合性が要求されました。しかし、多くのお客様はアプリケーションレイヤーで、または ETL 処理により参照整合性を適用しています。このようなケースは「ソフト」な参照整合性といいます。ユーザーは Tableau に対し、ソフトな参照整合性が適用されていること、そして「参照整合性の想定」を有効にすることで結合選択が安全に使用できるということを、設定できます。

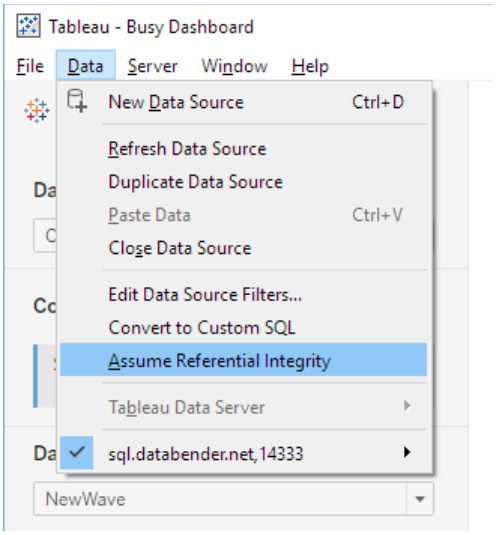


Tableau は参照整合性がハードでもソフトでも使用できますが、データベースが結合選択も可能になるためハードな参照整合性を使用した方がいいことに注意してください。詳しくは、ラッセル・クリストファー氏のブログ「Tableau Love」に連載されている以下の記事をご覧ください。

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

区分の指定

データベースを区分するとは、大きな表を小さな別個の表（「パーティション」「シャード」などと呼ばれます）に分割することであり、これによりパフォーマンスが向上します。つまり、スキャン対象のデータ量が少なくなるため、または I/O に利用できるドライブが増えるため、クエリの実行速度が上がるのです。区分はデータ量が多いときに推奨される方法で、Tableau に対して透過的です。

Tableau で区分の威力が発揮されるのは、フィルターが適用されることの多いディメンション（時間、地域、カテゴリーなど）にわたって区分を行い、クエリが 1 つのパーティション内のレコードのみを読み取ればよいように構成されたときです。

データベースによっては、区分インデックスを正しく使用するために日付範囲フィルター（不連続フィルターではありません）が必要になる場合があります。このような場合に日付範囲フィルターを使用しないと、表を完全にスキャンしても結果は非常に悪くなります。

一時表

Tableau には、一時表の使用を生じる操作が数多くあります。たとえばアドホックグループとセットの作成、データブレンディングの実行などです。一時表の作成とドロップができるようにユーザーの許可を得て、環境内にクエリが十分実行できるスプールスペースを保証することをお勧めします。

OLAP

Tableau では次の OLAP データソースをサポートしています。

- Microsoft Analysis Services
- Microsoft PowerPivot（Excel 用 PowerPivot と SharePoint 用 PowerPivot）
- Oracle Essbase
- SAP Business Warehouse
- Teradata OLAP

OLAP への接続とリレーショナルへの接続とでは、MDX/DAX と SQL 間に基本的な言語的違いがあるため、機能的に異なります。主な留意点は、両方とも Tableau では同じユーザーインターフェイス、同じビジュアライゼーション、計算されたメジャーに対し同じ記述言語を持っていることです。主な差はメタデータ（定義の方法と場所）、フィルタリング、合計や集計の方法、データブレンディングの際にデータソースがどのように使用されるかなどに関するものです。

リレーショナルデータソースと OLAP データソース上ので Tableau を使用する際の違いについて、詳しくは以下のナレッジベース記事をご覧ください。

http://kb.tableau.com/articles/knowledgebase/functional-differences-olap-relational?lang=ja-jp

SAP BW データ抽出

SAP BW は、OLAP データソースの中でも独特です。その理由は SAP BW キューブから Tableau のデータエンジンにデータを抽出することができるためです（注意 – これを実行するには、Tableau から取得できる特別なキーコードが必要です）。Tableau はリーフノード（それより下に階層のないデータ）を検索し、それらでリレーショナルデータソースを作ります。多次元からリレーショナルに変換するときは、すべてのキューブ構造が維持されるわけではないため、キューブ抽出では、ビジュアライゼーションの状態に影響することなく抽出とライブ接続を自由に切り替えることはできません。viz の構築を始める前に決めておかなければいけないことがありますが、すべてを事前に

決定する必要はありません。抽出後、複数の別名オプション（キー、長い名前など）を切り替えることができます。

SAP BW について、詳しくは以下をご参照ください。

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

ビッグデータ

ビッグデータは、データ分析の世界では意味が多数ありすぎる用語ですが、このドキュメントでは特に Hadoop をベースにしたプラットフォームを指す言葉として使います。Tableau 10 では、Hive および/または Impala 接続をサポートする Hadoop のディストリビューションが 4 つ あります。

- 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

  SharkServer と SharkServer2 接続は使用する場合に提供されますが、Tableau ではサポートしていません。

Hive は SQL-Hadoop 変換レイヤーとして動作し、クエリを MapReduce に変換して、HDFS データ上で実行できるようにします。Impala は HDFS データに対して SQL ステートメントを直接実行します（MapReduce は必要なくなります）。Tableau はまた、ビッグデータ向けにオープンソース処理エンジンである Spark SQL もサポートしています。これは、オンディスクでなくインメモリで実行することにより MapReduce よりも最高 100 倍速く実行できます。

Impala は基本的に Hive よりもかなり高速ですが、Spark はそれよりも速くなります。

ただし、このような追加コンポーネントがあっても、Hadoop の応答性は多くの場合、Tableau で作成されるような分析クエリには十分ではありません。クエリの応答時間を改善するには、多くの場合、Tableau データ抽出を使います。抽出の詳細、抽出を「ビッグデータ」でどのように活用するかについては、後で取り上げます。

Hadoop データソースでのパフォーマンス向上について詳しくは、次のページをご覧ください。

http://kb.tableau.com/articles/knowledgebase/hadoop-hive-performance?lang=ja-jp

クラウド

Tableau は現在、以下のクラウドデータソースをサポートしています。

- Salesforce.com
- Google アナリティクス
- oData (Windows Azure Marketplace DataMarket を含む)

この最初のソース群は、Web サービスからデータレコードのセットを読み取り、Tableau データ抽出ファイルに読み込みます。「ライブ接続」はこのようなデータソース向けのオプションではありませんが、抽出ファイルを更新して含まれるデータをアップデートすることができます。Tableau Server を使用すると、この更新プロセスを自動化し、スケジュール設定できます。

Tableau は、以下のクラウドベースのデータプラットフォームもサポートしています。

- Amazon Aurora
- Amazon Redshift
- Amazon RDS
- Google BigQuery
- Google Cloud SQL
- Google スプレッドシート
- Microsoft Azure SQL Data Warehouse
- Snowflake

前のデータソースのグループとは対象的に、これらはリレーショナルデータソースのように操作でき、ライブ接続と抽出の両方とも可能です。これについて、ここではこれ以上詳しく述べません (前述のリレーショナルデータソースのセクションを参照)。クラウドからの大量データ移動を避けるため、一般的にこれをライブ接続として維持しておくことをお勧めします。

最後に、Tableau は、Web ベースのサービスからデータをインポートするための汎用データコネクタも提供します。これは JSON、XML、HTML 形式にデータをパブリッシュするもので、Web データコネクタといいます。

Salesforce

Salesforce への接続時は、以下に挙げたコネクタの制限を考慮する必要があります。

「ライブで接続する」オプションがない

ライブ接続でなく抽出のみを選択する理由は、以下を含めいくつかあります。

- パフォーマンス – Salesforce に対するライブ分析クエリは (一般的に) 時間がかかります。
- API の割り当て – Salesforce ライブにあまりに頻繁にヒットすると、1 日あたりの割り当て上限に届いた場合にアカウントの一時停止を起こすおそれがあります。抽出によって API を効率的に利用し、要求される API コール数を最小限にして上限の到達を回避します。最適なパフォーマンスを維持し、Force.com API がすべてのお客様に利用可能であるよう保証するために、Salesforce.com は 2 種類の制限を課すことでトランザクションの負荷のバランスをとっています。それは、同時 API 要求制限 (http://sforce.co/1f19cQa) と総 API 要求制限 http://sforce.co/1f19kiH) です。

最初の抽出は非常に遅い場合がある

Salesforce からデータを初めて抽出するとき、表のサイズや force.com の読み込みなどによって時間がかかることがあります。これは、オブジェクト全体がダウンロードされるためです。

結合オプションに制限がある

複数の表オプションを選択するときは、PK/FK キー上のみでオブジェクトを結合できる（左と内部結合のみ）ことに留意してください。

データの事前のフィルターはできない

Salesforce コネクタを通してデータを事前にフィルターすることはできません。この要件が非常に重要な場合は、ライブ接続をサポートするサードパーティー製の Salesforce ODBC ドライバー（Simba や DataDirect など）を使用できます。そうすることでこの接続に対し抽出を実行できます。

DBAmp もまた、Salesforce のデータを SQL Server データベースにアンロードするソリューションを提供します。これを使用すると、SQL Server コネクタ経由で Tableau に接続できます。

式の列は移行ができない

計算の入ったフィールドがある場合は、データの抽出後に Tableau 内で再度作成する必要があります。

クエリに 10,000 文字の制限がある

Force.com API ではクエリの合計文字数を 10,000 文字に制限しています。非常に幅の広い表（列数が多く、列名も長いなど）を 1 つ以上接続すると、抽出時、制限に達してしまうおそれがあります。そのような場合は、選択する列数を少なくし、クエリのサイズを小さくする必要があります。場合によっては、Salesforce.com が会社に対してクエリの上限を引き上げることもあります Salesforce の管理者に詳細をお問い合わせください。

Google アナリティクス

Google アナリティクス（GA）は、レポートに多数のディメンションが含まれる場合やデータ量が多い場合に、データのサンプリングを行います。（通常の GA アカウントで）所定の日付範囲内の特定の Web ページに関するデータが表示回数 50,000 回を超える場合、GA は結果を集計して、そのデータのサンプルセットを返します。GA が Tableau にデータのサンプルセットを返すと、Tableau はビューの右下隅に次のメッセージを表示します。

「Google アナリティクスによってサンプルデータが返されました。接続に多数のディメンションまたは大量のデータが含まれている場合にサンプリングが発生します。サンプリングがレポート結果に与える影響の詳細については、Google アナリティクスドキュメントを参照してください」

データがいつサンプリングされているかを知るのは重要です。特定のデータのサンプルセットを集計すると、非常に不均衡で不正確な推測値が得られる可能性があるためです。たとえば、データの中でもめずらしいカテゴリーがサンプリングされたサンプルセットを集計したとします。すると、集計されたサンプルセットの推測値は、そのカテゴリーに含まれるサンプル数が不十分なために偏っていることがあります。データを正確に推測できる GA ビューを構築するには、推測するカテゴリー内に十分な数のサンプルがあるようにします。推奨される最小のサンプルサイズは 30 件です。

GA のサンプルサイズを調整する方法や、GA のサンプリングについて詳しくは、GA ドキュメントをご覧ください。

https://support.google.com/analytics/answer/1042498?hl=ja

サンプリングを避ける方法は 2 つあります。

- セッションまたはヒットレベルで複数の GA レポートを実行し、データをサンプリングされない塊に分割します。その後データを Excel ファイルにダウンロードし、Tableau の抽出エンジンを使用してデータソースからデータを追加し、データを 1 つのデータセットに再構成します。
- GA アカウントをプレミアムアカウントにアップグレードします。これで、レポートに含められるレコード数が増加します。これで、分析目的でデータを塊にするのもかなり簡単になります。今後について言うと、Google は、GA プレミアムユーザーがセッションやヒットレベルデータを Google BigQuery にエクスポートし、さらに分析できるようにすると発表しました。Tableau は BigQuery に直接接続できるため、データへのアクセスがさらに簡単になります。

最後に、Tableau が GA へのクエリに使用する API では、クエリのディメンションは最大で 7 個、メジャーは 10 個に制限されます。

Web データコネクタ

Tableau の Web データコネクタを使用すると、まだコネクタが用意されていないデータに接続できます。Web データコネクタにより、HTTP を介してアクセス可能なほぼすべてのデータに対する接続を作成し、使用できます。これには、社内 Web サービス、JSON データ、XML データ、REST API、その他の多数のソースが含まれます。データの取得方法をコントロールできるので、複数のソースからのデータを組み合わせることもできます。

Web データコネクタは、JavaScript と HTML を含む Web ページを記述することで作成されます。Web データコネクタを記述した後に、Tableau Server にパブリッシュすることで他の Tableau ユーザーと共有することができます。

Web データコネクタの作成を支援するために、Tableau ではソフトウェア開発キット (SDK) を用意しています。SDK には、テンプレート、サンプルコード、Web データコネクタをテストできるシミュレーターが含まれています。このドキュメントにはまた、Web データコネクタを一から作成する方法を順を追って説明するチュートリアルも含まれています。Web データコネクタ SDK はオープンソースプロジェクトです。詳しくは、GitHub の Tableau Web データコネクタのページをご覧ください。

Web データ コネクタ について詳しくは、こちらをご覧ください。

http://tableau.github.io/webdataconnector/#WDC/wdc_resources.htm?TocPath=___3

データ準備

ワークブックで使用する必要があるデータには、時折不備があることがあります。クリーンでなかったり、体裁が整っていなかったり、複数のファイルやデータベースに分散していることもあります。従来、Tableau はクリーンで標準化されたデータが提供されると最大限の効果を発揮していました。すなわち、データを Tableau に読み込む前に他のツールを使用してデータの準備をしなければならない場合があったのです。

今はもう必ずしもそうではなく、Tableau には、乱雑なデータの読み込みを支援する機能がいくつもあります。この機能には、次のようなものがあります。

- ピボット
- ユニオン
- データインタプリター
- 列のマージ

臨機応変にデータ準備を実行することで、a) ユーザーが分析の流れから逸脱しないようにし、b) 他のツールを使用しなくても Tableau 内でデータ分析を続けることを可能にします。ただし、これらの操作は基本的に負荷が高く（特に大容量のデータの場合）、パフォーマンスの面でレポートユーザーに影響を及ぼします。これらの操作を使用した後は、可能な限りデータ抽出を使用して、抽出内にデータを持つことをお勧めします。

これらの機能は既存の ETL プロセスを補完するものであると考える必要があります。データが複数のレポートで使用されて他の BI ツールと共有されている場合は、Tableau のデータアップストリームの事前処理がより効率的なアプローチかどうかを考慮してください。

データ抽出

ここまでは、データが元の形式のままのデータ接続に関して、そのパフォーマンスを向上するための手法をご紹介してきました。これらはライブデータ接続といい、ライブデータ接続では、パフォーマンスの面でも機能の面でもソースデータのプラットフォームに依存していました。ライブ接続でパフォーマンスを向上するためには、データソースに変更を加える必要があることも少なくありませんが、これが可能ではないお客様も多数いらっしゃいます。

誰でも実行可能な代替方法は、Tableau の高速データエンジンを活用し、ソースデータシステムから Tableau データ抽出にデータを抽出することです。こうすればほとんどのユーザーが、最も手早く簡単に、どんな種類のデータソースでもワークブックのパフォーマンスを大幅に向上させることができます。

抽出には、次の特徴があります。

- ディスクに書き込まれており、再現可能なデータの永続的なキャッシュ
- カラム型データストア（分析クエリに対してデータが最適化されたフォーマット）
- クエリ中はデータベースから完全に接続解除事実上、抽出はライブデータ接続の代わりとなる
- 抽出を完全に再生成するか、既存の抽出にデータの行を徐々に追加することで、更新可能
- アーキテクチャを意識。ほとんどのインメモリ技術とは異なり、利用可能な物理的 RAM の量による制約を受けない
- 移植可能。抽出はファイルとして保存されるため、ローカルハードドライブにコピーして、ユーザーが企業ネットワークに接続していないときにも使用可能また、Tableau Reader 用に配信されるパッケージドワークブックにデータを埋め込む際にも使用できる
- 多くの場合、参照元のライブデータ接続よりずっと動作が速い

The Information Lab の トム・ブラウン氏が、抽出によりメリットが得られる事例について説明する素晴らしい記事を書いています。他のユーザーが別の例を紹介しているコメントも、あわせてお読みください。

http://bit.ly/1F2iDnT

データ抽出については、重要な点がもう 1 つあります。データ抽出はデータウェアハウスの代用ではなく、むしろ補完的なものです。どちらも時間をかけたデータの収集および集計（たとえば周期的なデータの増分更新など）に使用できますが、抽出は、長期的ソリューションではなく、戦術的なソリューションとして使用することをお勧めします。増分更新では、処理済みのレコードの更新または削除アクションをサポートしていません。これらを変更するには、抽出を完全に読み込み直す必要があります。

最後に、SQL Server Analysis Services や Oracle Essbase などの OLAP データソースでは抽出を作成できません。このルールの例外として、SAP BW からは抽出を作成することができます（前述の関連セクションを参照）。

抽出を使うケース、ライブ接続を使うケース

データ抽出にも、使用に適した時と場所があります。以下に、抽出が役立つ場合があるシナリオをいくつかご紹介します。

- クエリの実行が遅い場合 – Tableau Desktop で生成されるクエリをソースデータシステムで処理するのに時間がかかる場合、抽出を作成することで簡単にパフォーマンスを向上できることがあります。抽出データフォーマットは、本質的に、分析クエリに短い時間で応答するよう設計されているため、このような場合には抽出をクエリ高速化キャッシュと考えることができます。接続の種類によっては（サイズの大きいテキストファイル、カスタム SQL 接続などの場合）、これが推奨されるベストプラクティスであり、一部のソースはこのモデルでのみ動作します（クラウドデータソースに関するセクションを参照）。
- オフラインで分析する場合 – 元ののデータソースが利用できない状態でデータを操作する必要がある場合（出張や在宅勤務でネットワークに接続していない場合など）がこれに該当します。データ抽出はファイルとして永続し、ノートパソコンなどのポータブル機器に簡単にコピーできます。ネットワークへの接続状況が頻繁に変わる場合でも、簡単に抽出とライブ接続を切り替えることができます。
- Tableau Reader 用、Online 用、Public 用などのパッケージドワークブック – ワークブックを共有して他のユーザーが Tableau Reader で開けるようにする場合、または Tableau Online や Tableau Public にパブリッシュする場合は、パッケージドワークブックファイルにデータを埋め込む必要があります。埋め込み可能なデータソース（ファイルベースのデータソース）をワークブックで使用している場合でも、データ抽出は本質的にデータ圧縮のレベルが高いため、得られるパッケージドワークブックはずっと小さいサイズになります。
- 追加機能 – 一部のデータソース（MS JET ドライバー経由のファイルベースのデータソースなど）では、Tableau Desktop でサポートされていない機能があります（中央値、個別カウント、ランク、百分位集計、セットの IN/OUT 操作など）。データを抽出すると、これらの機能を簡単に有効にできます。
- データセキュリティ – ソースデータシステムからデータのサブセットを共有する必要がある場合、抽出を作成して他のユーザーが利用できるようにすることが可能です。含めるフィールドや列の数を制限できるほか、個別のレコードレベルのデータの代わりにサマリー値を他のユーザーに見せる場合は集計データを共有することもできます。

抽出は非常に強力な機能ですが、すべての問題に対する特効薬ではありません。抽出の使用が不適切なこともあるシナリオを以下にご紹介します。

- リアルタイムデータ – 抽出はある時点におけるデータのスナップショットであるため、分析にリアルタイムデータが必要な場合は使用に適しません。Tableau Server を利用して抽出を自動更新することは可能であり、多くのお客様が日々この方法を実践していますが、真にリアルタイムのデータにアクセスするにはライブ接続が必要です。
- 膨大なデータ – 対象となるデータ量が膨大な場合（「膨大」の定義はユーザーによって異なりますが、一般にレコード数が数百万件から数十億件ある場合を指します）、抽出は実用的ではありません。得られる抽出ファイルも、サイズが大きすぎるか、作成プロセスに何時間もかかることがあります。ただし、このケースにはいくつか例外があります。ソースデータセットが膨大であっても、実際に作業するのはこのデータにフィルタリング、サンプリング、

集計処理などを行って得られたサブセットである場合、抽出が適していることも大いに考えられます。一般論を言えば、Tableau 抽出エンジンは数億件のレコードまでなら十分扱えるよう設計されています。ただし、これはデータの形状や密度に左右されます。

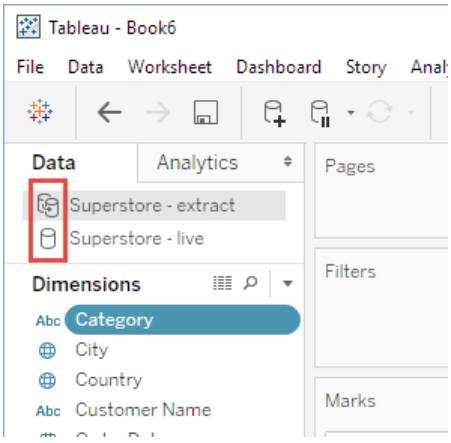
- パススルー RAWSQL 関数 – ワークブックでパススルー関数を使っている場合、データ抽出には適しません。
- ユーザーレベルのセキュリティが強固な場合 – ユーザーレベルのセキュリティを強固にする必要がある場合は、データソースに実装する必要があります。ワークブックレベルで適用されるユーザーレベルのフィルターがある場合、ユーザーはこれらをいつでも削除でき、抽出に含まれるすべてのデータにアクセスできるようになります。このケースでの例外は、データソースフィルターがある状態で抽出が Tableau Server にパブリッシュされており、他のユーザーは Data Server を経由してこの抽出にアクセスしている場合です。ただし、ユーザーのダウンロード許可を取り消し、実装済みのフィルターを迂回できないようにする必要があります。

Tableau Desktop での抽出の作成

ほとんどの場合、初回の抽出作成は Tableau Desktop 内で行われ、非常にシンプルです。データに接続したら、[データ] メニューに移動し [データの抽出] をクリックします。次にダイアログボックスのデフォルトに同意します (詳しくは後述します)。Tableau が抽出の保存先を選択するように求めるため、ファイルを保存する任意の場所を選択します。Tableau は「My Tableau Repository | Datasources」を提案しますが、別の場所を指定してもかまいません。

後は抽出が作成されるのを待つだけです。所要時間は、使用されているデータベース技術、ネットワーク速度、データ量などによって異なります。また、抽出の作成はメモリやプロセッサを大量に使用するため、お使いのワークステーションの処理速度や容量にも左右されます。

データソースのアイコンが変化して、作成の終了を知らせます。アイコンの後ろに別のデータベースのアイコンが表示されます。これコピーであることを表すもので、これがまさに抽出の定義です。



この方法 (Tableau Desktop 経由) で抽出を作成するとき、処理はワークステーション上で発生するため、タスクを実行する容量が十分あるか確認する必要があります。抽出の作成には、CPU、RAM、ディスク容量、ネットワーク I/O などあらゆる種類のリソースを使用します。また、スペックの低い PC で大量のデータを処理する場合、いずれかのリソースが不足するとエラーが発生する可能性があります。大規模な抽出には、それに適したワークステーション (コアプロセッサが複数ある高速 CPU、大容量の RAM、速い I/O など) を使用することをお勧めします。

抽出の作成プロセスには、作業ファイルを書き込む一時ディスクスペースが、最終抽出ファイルサイズの最高 2 倍必要です。この作業スペースは TEMP 環境変数が既定するディレクトリ内に割り当てられます (通常は `C:\WINDOWS\TEMP` または `C:\Users\USERNAME\AppData\Local\Temp`)。このドライブのスペースが不十分な場合は、環境変数をより容量に余裕がある場所にポイントします。

ワークステーションで初期抽出プロセスを実行できない場合 (または非実用的な場合)、空の抽出を作成して後で Tableau Server にパブリッシュされるようにするために、次の回避策を実行できます。まず `DateTrunc("minute", now())` を含む計算フィールドを作成します。次に抽出フィルターを追加し、表示される単一の値を除外します。1 分後にはフィルターが有効でなくなるため、この操作は手早く行ってください。もっと時間が必要な場合は、パブリッシュ間隔を広めにとります (5 分、10 分、必要に応じて 1 時間など)。これで、空の抽出がデスクトップに作成されます。サーバーにパブリッシュして更新スケジュールをトリガーすると、除外したタイムスタンプがすでに変わっているため、完全な抽出が設定されます。

データ抽出 API を使用した抽出の作成

Tableau はまた、開発者が直接 Tableau データ抽出 (TDE) ファイルを作成できるようにするアプリケーションプログラミングインターフェース (API) を提供しています。開発者はこの API を使用して、オンプレミスのソフトウェアとサービスとしてのソフトウェア (SaaS) から抽出を作成します。この API により、ご使用のデータソースに対応するネイティブコネクタがないときもデータを体系的に Tableau に取り込むことができます。

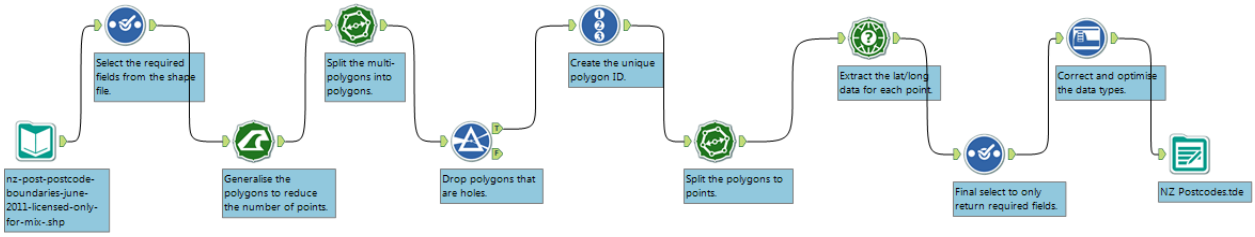
この API は、Windows と Linux 両 OS 上での Python、C/C++/Java 開発者が利用可能です。API について、詳しくはこちらをご覧ください。

http://onlinehelp.tableau.com/current/pro/desktop/ja-jp/extracting_TDE_API.html

サードパーティー製のツールによる抽出の作成

数多くのサードパーティーのツール開発者が、データ抽出 API を使用してネイティブ TDE の抽出結果を自分のアプリケーションに追加してきました。これらのアプリケーションには、Adobe Markething Cloud などの分析プラットフォームや Alteryx、Informatica のような ETL ツールがあります。

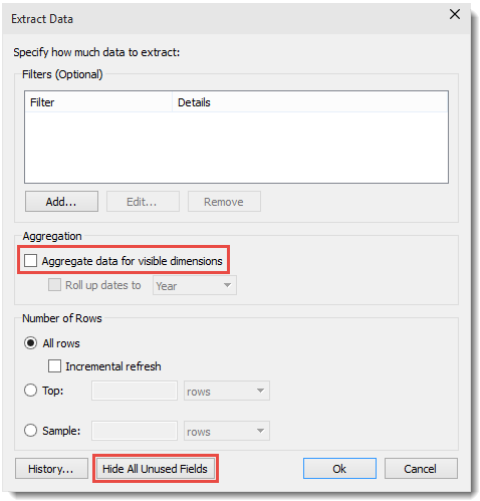
データ準備の要件が複雑な場合は、Alteryx と Informatica のようなツールが ETL 段階の実行に効果的です。そして準備のできたデータを TDE ファイルに直接出力すると、Tableau Desktop で使用することができます。



集計された抽出

集計された抽出を使用すると、パフォーマンスは必ず向上します。Teradata や Vertica で膨大な量のデータを扱っているとしても、データの集計やフィルタリングを適切に行えば、データ抽出による効果を得られます。たとえば、最新のデータのみに接続されている場合は、そのデータをフィルタリングできます。

必要なフィールドを選択し、Tableau Desktop の [データの抽出] ダイアログボックスにある [表示可能なディメンションのデータの集計] チェックボックスを選択すると、事前に抽出を定義できます。また、分析を行いダッシュボードを作成した後、パブリッシュする用意が整ったときにも、[データの抽出] ダイアログボックスに戻り、[使用していないフィールドをすべて非表示] ボタンをクリックできます。その後データを抽出すると、データはビューを作成するのに必要な絶対最小限にとどまります。



集計された抽出を作成することは、大量のデータベースがありながら、データセット全体にわたりクエリを実行するサマリービューを作成しなければならないときに非常に役立つ手法です。たとえば、10 年間の売上を表す詳細なトランザクションデータのレコードが 10 億件あり、10 年間全体の売上推移の表示から始める場合を考えましょう。この初期ビューのクエリは 10 億行全部にクエリを実行する必要があるので、時間がかかる可能性があります。年レベルで集計された抽出を作成しておけば、抽出内には数字が 10 個しかないため、表示時間に必要なクエリの実行を減らすことができます。もちろんこれは、単純化しすぎた例です。実際は、ビュー時間にクエリを実行する必要があるレコードの数を大幅に減らすことを除き、時間以外にも多くの次元があると思われます。

ユーザーは、複数の集計された抽出 (それぞれが特定の詳細レベルをサポートするよう調整) を作成する、あるいは、抽出時の複製をライブ接続で組み合わせることによって、複数の詳細レベルを持つかなり洗練されたワークブックを作成することができます。たとえば、非常に集計された抽出を使用する初期のサマリービューがいくつかあるとします。しかし詳細へドリルダウンするときは、サマリービューからライブ接続を経由して接続する別のシートへアクションフィルターを使用します。つまり、サマリービューは、完全なベースデータセットをくまなく調べる必要がないため速くなりますが、ドリルダウンのアクションをサポートするためにすべてのベースデータを抽出する必要もありません。また、ドリルダウンレベルではレコードの小さいセットにだけアクセスするため、ライブ接続も速くなります。

このように、異なるレベルで組み合わせ、一致、および集計を行うことにより、パフォーマンスに関するほぼすべての問題を解決でき、必要な速度で結果を得ることができます。Tableau はメモリ効率がよいため、通常は比較的簡単にこの方法でパフォーマンスを向上でき、同時に複数の抽出を実行することもできます。

抽出の最適化

Tableau Server は、データベースにある物理的な列を最適化するだけでなく、Tableau で作成された追加の列も最適化します。これらの列には文字列操作や文字列連結などの決定性の計算の結果が含まれており、その結果はグループやセットとともに決して変わりません。実行時に計算され

るパラメーターや集計（合計や平均など）に関わる計算を含む、非決定性の計算の結果は、保存できません。

データをほんの 2 行追加して抽出を更新すると、抽出のサイズが 100 MB から 120 MB へと大幅に増加することがあります。この急な増加は、最適化により計算フィールド値を含む列が追加で作成されることに起因します。列を追加する理由は、ディスクにデータを保存するほうが、データが必要になるたびに再計算するよりも時間やリソースを削減できるからです。

データ抽出への接続を複製する際に気を付けないといけないことは、すべての計算フィールドが最適化/更新したい接続に含まれているようにすることです。計算フィールドが含まれていないと、Tableau は、その計算フィールドが使用されていないと認識し、計算済みのデータを抽出データに作成しません。必ず、プライマリデータソースにある計算フィールドをすべて定義し、必要に応じて別の接続にコピーして、プライマリデータソースからの抽出のみを更新または最適化するようにしてください。

> 注意：Tableau 開発チームの信頼できるリソースからのコメントによると、1 つの TDE に複数の接続を作成することは可能ですが、これをサポートするように開発されていないということです。接続が TDE の構造と一致しなくなると、多くの問題を引き起こす可能性があります。複数の接続を作成しないでください、というアドバイスを受けました。

抽出の更新

Tableau Desktop で抽出を更新するにはメニューからの選択で行います（[データ] メニュー > [{目的のデータソース}] > [抽出] > [更新]）。更新するとデータが最新のものになり、新しい行があれば追加されます。ただし、Tableau Server では、パブリッシュプロセスの最中またはその後で、抽出を自動更新するよう管理者が定義したスケジュールを付加することができます。スケジュールは 15 分間隔で設定でき、毎日、毎週などの設定で同時刻に更新するようにもできます。「移動窓」を設定して、データが最新のものに絶えず更新されるようにすることもできます。

注: 15 分ごとの更新間隔では不十分という方は、ライブデータに接続するか、同期型レポートデータベースを設定することをお勧めします。

つの抽出に対して 2 種類の更新スケジュールを選択できます。

- 増分更新では行が追加されるのみで、既存の行に対する変更は含まれません。
- 完全更新では現在の抽出が破棄され、データソースから新しい更新を最初から再生成します。

更新間隔よりも更新に時間がかかった場合

抽出の更新が更新間隔よりも長くかかった場合、更新間隔はスキップされます。たとえば、1 時間ごとにデータを更新するようスケジュール設定されているのに、データ量が多すぎて更新に 1 時間半かかるとします。その場合、

- 1 回目の更新が 1:00 に始まり 2:30 に終わります。
- 次の更新は 2:00 開始予定ですが、第 1 の更新がまだ実行中であるため、スキップされます。
- 2 回目の更新は 3:00 に始まり 4:30 に終わります。

抽出のメンテナンス

[メンテナンス] 画面に、現在実行中のバックグラウンドタスクが、過去 12 時間に実行されたタスクとともに表示されています。色分けはタスクのステータスを表しています。[メンテナンス] 画面を利用できるのは、管理者と、適切な許可を持つ一部のユーザーです。該当する人には、抽出のアド

ホック更新を開始する許可が与えられます。また、たとえばデータベースが読み込む場合に、データベースの読み込み完了後に抽出を開始するトリガーを設定することができます。

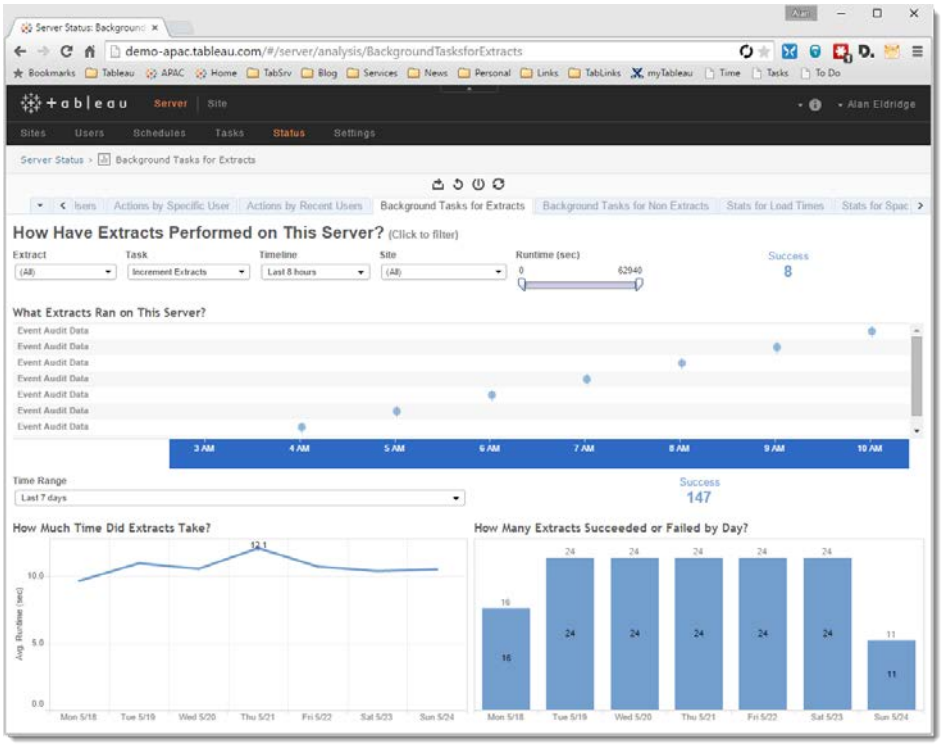


Tableau Server を使用している場合は `tabcmd` コマンドラインツールを、Tableau Desktop を使用している場合は Tableau.exe コマンドラインを使用して、ワークブックを段階的に、または完全に更新することもできます。複雑なスケジュール設定が必要な場合は、Windows タスクスケジューラなどの外部スケジュール管理ツールからこれを呼び出すことができます。Tableau Server インターフェイスで最小の 15 分間隔より短い更新サイクルを希望する場合は、この方法が必要になります。

なお、Tableau Online ユーザーの場合、Tableau Online 同期クライアントを使用し、オンラインサービスに定義されたスケジュールでオンプレミスのデータソースを最新に維持することができます。このユーティリティについて、詳しくはこちらをご覧ください。

http://onlinehelp.tableau.com/current/online/ja-jp/qs_refresh_local_data.htm

データガバナンス

これ自体はデータソースではありませんが、データソースに接続するもう 1 つの方法に、Tableau Server の Data Server を経由する方法があります。Data Server はライブ接続とデータ抽出の両方をサポートし、スタンドアロン型のデータ接続と比較した場合に利点がいくつかあります。

- メタデータが Tableau Server に一元的に保存されるため、それを複数のワークブックで、また、複数の作成者やアナリストで共有できます。ワークブックには一元化されたメタデータの定義へのポインターが含まれ、ワークブックが開かれるたびに、変更が行われたかがチェックされます。変更があった場合には、ワークブックに埋め込まれているコピーを更新するよう、ユーザーにメッセージが表示されます。つまり、ビジネス論理に対する変更を 1 か所で行うだけで、変更内容はその論理を使用するすべてのワークブックに反映されます。
- データソースがデータ抽出の場合、複数のワークブックにわたって使用できます。Data Server を使わなければ、ワークブックごとにその抽出のローカルコピーを含めることになります。

Data Server を使うと不必要なコピーの数が減り、サーバー上に必要な保存領域や更新プロセスの重複を削減することにつながります。

- データソースがライブ接続の場合、データソース用のドライバーを各アナリストの PC にインストールする必要はありません。Tableau Server にのみインストールしてください。Data Server が、Tableau Desktop からのクエリに対するプロキシとして動作します。

環境の再確認

単一ユーザーのテストでは問題なく機能したワークブックでも、Tableau Server (または Tableau Online)に展開して多数のユーザーで使用したらパフォーマンスが低下したという状況は時折発生します。次のセクションでは、単一ユーザーと複数ユーザーのシナリオで違いが出る分野を説明します。

アップグレードする

Tableau の開発チームは、ソフトウェアのパフォーマンスとユーザビリティの向上に絶えず取り組んでいます。Tableau Server を最新バージョンにアップグレードすると、ワークブックを変更しなくてもかなりのパフォーマンスの向上を生み出すことがあります。

Tableau のリリースノートページを確認して、最新版のビルドにアップグレードしてください。

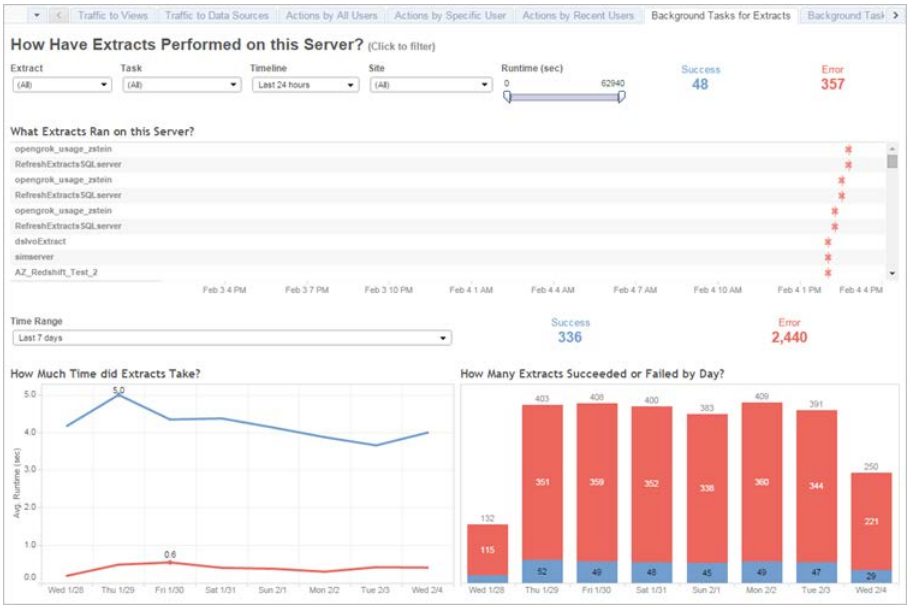
http://www.tableau.com/ja-jp/support/releases

サーバー上で Tableau Desktop をテストする

ワークステーション上の Tableau Desktop でパフォーマンスの良かったワークブックが、Tableau Server で表示すると遅くなったという経験をお持ちの方がいるのではないでしょうか。Tableau Server マシン上にインストールした Tableau Desktop のコピーを使ってワークブックを開くと、問題がワークブックにあるのか、サーバーの構成にあるのかを見定めるのに役立ちます。このアプローチでは、ドライバーの非互換性の問題なのか、不適切なルーティング、DNS、プロキシなどネットワークに問題があるのかを特定できます。

更新処理とインタラクティブなワークロードを切り離す

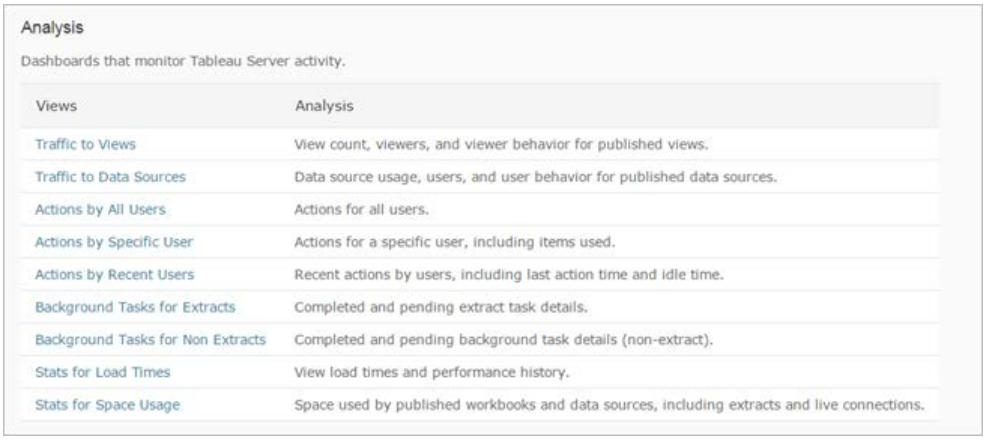
サーバーのパフォーマンスが低い場合は、バックグラウンドタスク管理ビューを使用して現在の更新タスクスケジュールを表示します。



オフピーク時間の更新をスケジュールできる場合は、実施してみてください。ハードウェア構成により可能な場合は、バックグラウンダープロセスを専用のワーカーノードに移動することもできます。

サーバーを監視し調整する

Tableau Server には、管理者が Tableau Server でのアクティビティを監視するためのビューがいくつか用意されています。これらのビューは、サーバーの [メンテナンス] ページの [分析] 表にあります。



Analysis

Dashboards that monitor Tableau Server activity.

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.

これらのビューについて詳しくは、以下のリンクをご覧ください。

http://onlinehelp.tableau.com/current/server/ja-jp/adminview.htm

また、カスタムの管理ビューは Tableau リポジトリの一部を構成する PostgreSQL データベースに接続することで作成できます。手順についてはこちらをご覧ください。

http://onlinehelp.tableau.com/current/server/ja-jp/adminview_postgres.htm

VizQL セッションのタイムアウト制限を確認する

VizQL セッションのタイムアウト制限は既定で 30 分です。VizQL セッションがアイドル状態でも、メモリや CPU サイクルは消費されています。下限値で済む場合は、tabadmin を使用して vizqlserver.session.expiry.timeout 設定を変更します。

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

プロセス構成の評価

Tableau Server はサーバープロセスと呼ばれる多くのコンポーネントに分割されます。既定の構成は幅広いシナリオに対応できるように設計されていますが、異なるパフォーマンス目標を達成するように再構成することもできます。特に、プロセスが実行されるマシンと実行台数を制御できます。1、2、および 3 マシンの配置については、Improving Server Performance (サーバーのパフォーマンスの向上) ガイドラインを参照してください。

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

インフラ

64 ビット

Tableau 10 より古いバージョンの Tableau Server は、32 ビットの Microsoft オペレーティングシステムで動作しますが、お客様には稼働環境に 64 ビットバージョンをインストールするよう強く推奨しています。32 ビットのインストールは、DEV/TEST のインストールに対してのみ推奨しています。

Tableau 10 以降は、Tableau Server は 64 ビットアプリケーションのみになります。

CPU/RAM の増設

Tableau Server を実行するマシンが 1 台か複数台かにかかわらず、原則としては CPU コアが多く RAM が大きい方がパフォーマンスが向上します。Tableau Server が推奨するハードウェアとソフト

ウェア要件 (http://onlinehelp.tableau.com/current/server/ja-jp/requ.htm#ida4d2bd02-00a8-49fc-9570-bd41140d7b74) を満たしていることを確認し、『ワーカーの追加と再構成の時期』(http://onlinehelp.tableau.com/current/server/ja-jp/distrib_when.htm#idfdd60003-298e-47e6-8133-3d2490e21e07) で、追加のマシンが必要かどうかを評価してください。

このドキュメントで前述した TabMon は、容量計画のプロセスに役立つ利用データを収集する優れたツールです。

IO を無視しない

Tableau のアクションの一部 (データ抽出の読み込み、作成、更新など) は頻繁な I/O を必要とするため、回転式ディスクよりも SSD を使用するほうが有利です。ラッセル・クリストファー氏が、ブログ「Tableau Love」で、コア数、CPU スピード、IOPS が全体のパフォーマンスに及ぼす影響を考察する素晴らしい記事をいくつか書いています。同氏の実験は AWS 上で実施されていますが、すべての環境に適用できます。

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

物理または仮想

今や多くのお客様が、Tableau Server を仮想インフラストラクチャーに展開するようになりました。仮想化は常にパフォーマンスのオーバーヘッドが生じるため、物理環境へのインストールほどは速くありませんが、新しいハイパーバイザテクノロジーはこのオーバーヘッドを大幅に削減しました。

仮想マシンにインストールするときは、Tableau Server が専用の RAM と CPU リソースを与えられていることの確認が重要です。物理ホスト上のリソースに対し他の仮想マシンとの競合が起きた場合、パフォーマンスが重大な影響を受けます。

仮想マシン配置の調整に役立つリソースとして、VMWare からホワイトペーパー "Deploying Extremely Latency-Sensitive Applications in vSphere 5.5" が出ています。15 ページに、レイテンシーの影響を受けやすいアプリケーションに対するベストプラクティスの充実したリストが記載されています。

> http://vmw.re/1L4Fyr1

ブラウザ

Tableau は JavaScript を多用するため、ブラウザの JavaScript 解釈プログラムの速度がレンダリング速度に影響します。最新のブラウザは、どれもこの分野での進化を競って早めていますが、お使いのブラウザのバージョンとパフォーマンスが使い心地に影響するか検討する価値はあります。

まとめ

かつて賢明な人がこのように言いました（そしてこのドキュメントのためにアドバイスを提供してくれました）。－「自分が伝えたいこと伝えて、伝えて、一度伝えた内容をまた伝えなさい」。実に聡明なアドバイスです。

というわけでここに再び、このドキュメントから得ていただきたい重要ポイントを挙げます。

- 特効薬はありません。パフォーマンスが低下する理由は数多くあります。データを集めて、どの部分に時間がかかっているかの特定に役立てましょう。そして最も負荷の高い部分に焦点を当てて改善を行い、その後で次の部分に進む、というように対応します。十分処理が速くなったか、対応策に対する効果がもう望めなくなった時点でやめましょう。
- 遅いダッシュボードは、設計不良が原因であることが多いものです。シンプルさを心がけましょう。あまり多くのものを一度に表示しないようにして、できるだけガイドに従った分析設計を使用してください。
- 不必要なデータを扱わないようにしましょう。フィルターを使用し、使用しないフィールドや集計は隠します。
- データエンジンを操作しようとするのはやめてください。データエンジンはスマートで、役に立とうと作られていますから、効率的なクエリを作成すると信頼してください。
- データがクリーンであるほど、質問の構造によくマッチし、ワークブックの動作が早くなって満足度が高くなります。
- 抽出を使うと、手早く簡単にほとんどのワークブックをより速く実行することができます。リアルタイムデータが不必要で何十億行ものデータを使うのでなければ、やってみるべきです。
- 文字列と日付の処理速度は遅く、数値とブールは速く処理されます。
- このドキュメントは数多くの賢い人々からのアドバイスに基づいて作成されていますが、それらは推奨事項にすぎません。個々のケースでパフォーマンスを改善するかどうかはテストする必要があります。
- 小さいボリュームのデータを扱う場合、この推奨事項はほとんど関係しないと思われます。その場合は不適切な手法で処理しても問題ありません。ただし、自分のデータがいつ大きくなるか分からないため、すべてのワークブックでこの推奨事項に従っておいて間違いはないでしょう。
- 習うより慣れろです。

それではこれから、効率的に作業できるワークブックを多数作成してください。