효율적인 Tableau 통합 문서 디자인 우수 사례

Tableau 10 Edition

Alan Eldridge
Tableau Software

문서 정보

다시 한 번, 본문에 들어가기에 앞서 본 문서는 다양한 저자가 작성한 글의 핵심을 집대성한 것이며 필자는 이러한 핵심을 하나의 문서로 통합하고 여기에 특정한 구조를 적용해보는 역할을 담당했을 뿐임을 미리 밝힙니다. 본 문서는 여러 부분에 다양한 저자의 손길이 닿아 있으며 일부는 텍스트의 상당 부분을 그대로 인용하고 있습니다. 이 문서를 작성할 수 있도록 도와주신 모든 분께 감사드리며 이분들의 지속적인 훌륭한 업적과 저작권에 대한 너그러운 관용이 없었다면 이 문서는 탄생할 수 없었을 것임을 다시 한 번 밝힙니다.

또한 본 문서의 정확성과 가독성을 검토해주신 많은 분들께도 감사드립니다. 이러한 분들이 세부적인 부분까지 살펴보고 세심한 설명을 덧붙여주신 덕분에 혼자서는 이루지 못했을 훨씬 가독성이 높은 문서를 만들 수 있었습니다.

본 문서는 Tableau 10 의 기능을 반영하도록 업데이트되었습니다. Tableau 제품의 향후 출시 버전에서는 본 문서에 언급되지 않은 새로운 기능이 제공될 예정이며 이로 인해 일부 권장 사항이 변경될 수 있습니다.

감사합니다.

Alan Eldridge
2016 년 6 월

백서가 긴 이유

이 백서를 읽거나 추천하신 분들 중 일부는 종종 내용이 너무 길다고 언급합니다. 광범위한 자료를 적절한 깊이로 다루기 위해서는 이만큼 길 수밖에 없음을 양해하여 주시기 바랍니다.

다음은 이 문서의 주요 내용에 대한 요약입니다.

- 비효율적인 통합 문서를 해결할 묘책은 없습니다. 먼저 성능 기록을 살펴보고 어디에서 시간이 소요되고 있는지 파악해 보십시오. 쿼리가 오래 실행됩니까? 쿼리가 너무 많습니까? 계산이 느립니까? 렌더링이 복잡합니까? 이러한 통찰력을 사용하여 올바른 방향으로 나아가십시오.
- 본 문서에 포함된 권장 사항은 말 그대로 권장 사항에 불과합니다. 일정 수준의 우수 사례를 보여주긴 하지만 여러분의 특수한 상황에서 성능이 향상될지는 직접 테스트해 보아야 합니다. 많은 권장 사항은 사용 중인 데이터의 구조와 데이터 원본(예: 플랫 파일, RDBMS, 데이터 추출)에 따라 다를 수 있습니다.
- 추출은 대부분의 통합 문서를 더 빨리 실행하는 빠르고 손쉬운 방법입니다.
- 데이터 정리가 잘 되어 있을수록 질문 구조에 일치할 가능성이 높아지고(예: 준비와 조작이 덜 필요함) 통합 문서가 더 빨리 실행됩니다.
- 대시보드가 느린 경우는 대부분 디자인 문제로 인해 발생합니다. 특히 대시보드 하나에 차트가 너무 많거나 한꺼번에 너무 많은 데이터를 보여주려는 경우가 여기에 해당합니다. 단순하게 만드십시오. 모든 것을 보여주고 필터링하는 대신 사용자가 세부적으로 드릴다운할 수 있도록 해야 합니다.
- 참조하는 필드뿐 아니라 반환하는 레코드의 세부 수준 측면에서 필요한 만큼의 데이터만 사용하십시오. 그렇게 해야 Tableau 에서 보다 적은 수의 적절한 쿼리가 빠르게 생성되며 데이터 원본에서 Tableau 엔진으로 이동해야 하는 데이터 양이 감소됩니다. 또한 통합 문서의 크기가 줄어들어 공유가 간편하고 더 빨리 열 수 있습니다.
- 데이터를 줄일 때 필터를 효율적으로 사용해야 합니다.
- 문자열과 날짜는 속도가 느리고 숫자와 부울 값은 빠릅니다.

마지막으로, 본 문서의 일부 권장 사항에는 크거나 복잡한 데이터 집합에만 효과가 있는 자료가 포함되어 있습니다. 크거나 복잡한 데이터 집합의 의미는 경우에 따라 다릅니다. 그래도 데이터가 언제 커질지 모르므로 모든 통합 문서에 이 권장 사항을 따라도 무방합니다. 연습하면 완벽해집니다.

목차

소개

Tableau 는 무엇에 가장 적합합니까?

Tableau 에서는 사람들이 새로운 방식으로 데이터를 바라보고, 파악하고, 데이터와 상호 작용할 수 있도록 하기 위해 노력하고 있습니다. Tableau Software 는 기존 엔터프라이즈 비즈니스 인텔리전스 플랫폼과는 완전히 다른, 새로운 경험을 선사합니다. Tableau 는 다음과 같은 기능을 지원하는 통합 문서를 작성할 때 가장 뛰어난 성능을 발휘합니다.

- **비주얼라이제이션** – 거대하고 복잡한 데이터를 이해하는 가장 효과적인 방법은 데이터를 시각적으로 표현하는 것이며 이는 다양한 자료에서 입증되고 있습니다. Tableau 는 기본적으로 모든 데이터를 차트, 도표 및 대시보드를 통해 표시하며 고유한 기능을 담당하는 테이블 및 크로스탭이 다양한 기능과 함께 제공됩니다. 테이블 및 크로스탭을 활용하는 방법은 추후에 별도의 섹션에서 설명하도록 하겠습니다.
- **뛰어난 상호 작용** – Tableau 문서는 데스크톱, 웹 또는 휴대기기에 관계없이 상호 작용이 가능한 형태로 제공되도록 디자인되어 있습니다. 기본적으로 종이나 PDF 같은 문서로 인쇄할 수 있는 결과물을 제공하는 데 초점이 맞춰져 있는 다른 비즈니스 인텔리전스 도구와 달리 Tableau 는 데이터를 직접 살펴보고 비즈니스 관련 질문에 대한 답을 찾아볼 수 있는 다채롭고 상호 작용이 가능한 경험을 선사하는 데 초점이 맞춰져 있습니다.
- **반복 작업 지원** – 새로운 사실을 발견하기 위해서는 본질적으로 순환적인 프로세스를 거쳐야 합니다. Tableau 는 질문에서 통찰, 다시 질문으로 이어지는 순환적인 프로세스를 신속하게 처리할 수 있도록 디자인된 제품을 통해 가정을 세우고, 사용 가능한 데이터로 이 가정을 검토하고, 원래의 가정을 수정하고 이를 테스트하는 작업을 신속하게 처리할 수 있도록 해줍니다.
- **빠른 속도** – 지금까지 비즈니스 인텔리전스 프로세스는 시간이 매우 오래 걸리는 작업으로 인식되어 왔습니다. 소프트웨어를 설치 및 구성하고 데이터를 분석할 수 있도록 처리하고 문서, 보고서, 대시보드 등을 디자인하여 구현하는 작업에 많은 시간이 필요했기 때문입니다. 하지만 Tableau 는 설치, 연결 및 작성 작업을 매우 신속하게 처리할 수 있어 이전에는 답을 구하는 데 몇 주에서 몇 달이 걸리던 작업을 몇 분에서 몇 시간 내에 처리할 수 있습니다.
- **단순함** – 기존 엔터프라이즈 비즈니스 인텔리전스 도구는 일반 사용자가 사용하기엔 너무 비싸거나 복잡한 경우가 많았으며 대부분의 사용자는 원하는 쿼리나 문서를 생성하기 위해 IT 부서나 경험이 많은 사용자의 도움을 받아야 했습니다. Tableau 는 데이터베이스나 스프레드시트에 대해 잘 모르는 비전문가도 복잡한 데이터를 분석하고 쿼리를 생성할 수 있는 직관적인 인터페이스를 제공합니다.
- **시야를 사로잡는 디자인** – 선호하는 디자인은 사람마다 다를 수 있지만 비주얼 커뮤니케이션의 측면에서 볼 때에는 따라야 하는 우수 사례가 존재합니다. Tableau 는 '자동 표시'와 같은 기능을 통해 비전문가도 사용 중인 데이터를 바탕으로 효과적이고 손쉽게 이해할 수 있는 차트를 만들 수 있도록 지원합니다.
- **보편성** – 문서를 만들 때 단일 플랫폼보다는 다양한 플랫폼에 제공할 수 있도록 문서를 만드는 추세가 확산되고 있습니다. 오늘날 사용자는 데스크톱, 웹, 휴대기기에 관계없이 다른 애플리케이션이나 문서 등에 삽입된 데이터를 확인하고 이와 상호 작용할 수 있어야 합니다. Tableau 는 한 형태의 문서를 게시하면 별도의 이식 또는 재디자인 작업 없이도 다양한 플랫폼에서 사용할 수 있도록 지원합니다.

Tableau 가 적합하지 않은 작업은 무엇입니까?

Tableau 는 매우 다채로우면서도 강력한 도구이지만 상황에 따라서는 Tableau 를 사용하는 것이 적합하지 않을 수도 있다는 점을 시작 단계에서부터 이해해야 합니다. 이는 Tableau 가 특정 기능을 수행할 수 없다는 뜻이 아닙니다. Tableau 는 원래 제품에 포함되지 않은 다양한 기능을 수행하도록 변칙적으로 사용할 수 있습니다. 단, Tableau 가 해결하도록 고안된 유형의 문제가 아닌 다른 문제에 Tableau 를 사용할 경우 노력 대비 보상의 비율이 원하는 수준에 미치지 못할 가능성이 높으며 문제에 대한 솔루션의 효용성 또는 유연성이 떨어질 수 있다는 점을 유념하시기 바랍니다.

다음과 같은 경우에는 요구사항을 다시 검토하거나 다른 접근 방식을 고려해 보십시오.

- 화면이 아니라 종이에 인쇄할 목적으로 디자인된 문서가 필요한 경우입니다. 복잡한 페이지 레이아웃을 조정해야 하거나 페이지, 섹션, 그룹 머리글/바닥글과 같은 기능이 필요하거나 WYSIWYG 서식을 정확하게 지정해야 하는 경우가 여기에 속합니다. Tableau 는 여러 페이지로 구성된 보고서를 작성할 수 있지만 다양한 스타일을 활용할 수 있는 보고 전용 도구 수준의 서식 제어 기능은 제공하지 않습니다.
- 개별적으로 사용자 지정된 문서('버스트'로도 지칭)를 다양한 전달 모드를 통해 푸시 형식으로 전달할 수 있는 복잡한 메커니즘이 필요한 경우입니다. Tableau Server 에는 사용자가 Tableau 10 등에서 이메일로 보고서를 수신하도록 직접 구독을 할 수 있는 기능은 제공하지만, 고객이 그보다 유연한 방법을 원하는 경우도 있습니다. Tableau 를 사용하여 다른 형식의 푸시 전달 시스템을 구축하는 것은 가능하지만 이는 Tableau 의 기본 기능이 아닙니다. 이러한 시스템을 구축하려면 TABCMD 유틸리티를 기반으로 사용자 지정 솔루션을 개발하거나 *VizAlerts* (http://tabsoft.co/1stldFh) 또는 Metric Insights 의 *Push Intelligence for Tableau* (http://bit.ly/1HACxul)와 같은 타사 솔루션을 도입해야 합니다.
- Reader 는 주로 데이터를 다른 형식(대개 CSV 나 Excel 파일)으로 내보내는 데 사용됩니다. 대개 상세한 데이터가 여러 행에 기재되어 있는 테이블 형식의 보고서가 여기에 속합니다. 참고로 Tableau 에서도 단순 요약 또는 세부정보 수준으로 특정 뷰 또는 대시보드의 데이터를 Excel 로 내보낼 수 있습니다. 단, 데이터를 내보내는 것이 주된 용도라는 것은 진행하는 작업이 단순한 추출, 변환 및 로드(ETL) 프로세스에 불과하다는 것을 의미하며 Tableau 와 같은 보고 도구보다 이러한 목적에 더 부합하는 솔루션은 많습니다.
- 복잡한 소계, 상호 참조 등이 포함된 기존 스프레드시트 보고서를 미러링하는 고도로 복잡하며 크로스탭 형식으로 구성된 문서가 필요한 경우입니다. 가장 일반적인 예로는 손익계산서, 대차대조표와 같은 재무 보고서가 있습니다. 이에 덧붙여 시나리오 모델링, 가정을 토대로 한 분석, 가정 데이터의 후기입 기능 등이 필요한 경우도 여기에 속합니다. 세분화된 기본 데이터를 사용할 수 없거나 보고서의 논리가 레코드를 롤업하여 종합하는 것이 아니라 '셀 참조'에 바탕을 두고 있는 경우에는 스프레드시트를 계속 사용하는 것이 더 좋습니다.

효율성 이해

'효율적인' 통합 문서란 무엇입니까?

'효율적인' 통합 문서를 판가름하는 요소에는 몇 가지가 있습니다. 이 중 일부 요소는 기술 중심적이거나 사용자 중심적이지만 일반적으로 효율적인 통합 문서의 특징은 다음과 같습니다.

- **단순함** – 손쉽게 통합 문서를 만들고 이후에도 간편하게 유지 관리됩니까? 시각적 분석 원칙을 활용하여 작성자와 데이터 간 메시지가 명확하게 전달됩니까?
- **유연함** – 사용자의 하나 또는 여러 질문에 통합 문서가 답변할 수 있습니까? 사용자에게 상호 작용이 가능한 환경을 제공합니까? 아니면 간단한 정적 보고서에 불과합니까?
- **신속성** – 통합 문서가 사용자에게 빠르게 답변을 제공합니까? 여기서 빠르다는 것은 통합 문서를 여는 시간, 새로 고치는 시간 또는 상호 작용에 대한 응답 시간일 수 있습니다. 신속성은 다분히 주관적이긴 하지만, 일반적으로 통합 문서가 몇 초 이내에 정보를 처음 표시하고 사용자 상호 작용에 응답할 수 있어야 합니다.

대시보드의 성능에 영향을 주는 요인은 다음과 같습니다.

- 대시보드 및 워크시트 수준의 시각적 디자인 – 예: 요소 수, 데이터 요소 수, 필터 및 동작의 사용 등
- 계산 – 예: 계산 종류, 계산이 수행된 위치 등
- 쿼리 – 예: 반환된 데이터의 양, 사용자 지정 SQL 여부
- 데이터 연결 및 초기 데이터 원본
- Tableau Desktop 과 Tableau Server 의 몇 가지 차이점
- 하드웨어 구성과 용량 등 기타 환경적 요소

효율성을 신경 써야 하는 이유는 무엇입니까?

다음과 같은 여러 이유가 있습니다.

- 분석가나 통합 문서 작성자가 효율적으로 작업하면 더 빠르게 답변이 제공됩니다.
- 효율적으로 작업하면 분석의 '흐름'이 유지됩니다. 즉, 결과를 얻기 위해서 도구 조작 방법이 아니라 질문과 데이터만 생각하면 됩니다.
- 디자인이 유연한 통합 문서를 만들면 유사한 요구사항을 해결하는 여러 통합 문서를 만들고 유지 관리할 필요성이 줄어듭니다.
- 디자인이 단순한 통합 문서를 만들면 다른 사용자가 손쉽게 이 통합 문서를 선택하고 이후에 이 작업을 반복 수행할 수 있습니다.
- 응답성은 최종 사용자가 보고서와 대시보드를 볼 때 중요한 성능 요인이므로 가능한 빠르게 실행되는 통합 문서를 만들면 사용자의 만족도가 높아집니다.

그동안의 많은 경험에 따르면 고객에게 발생하는 대부분의 성능 문제는 통합 문서 디자인의 실수로 인한 것이었습니다. 따라서 교육을 통해 처음부터 실수를 방지할 수 있다면 더 좋겠지만 이러한 실수를 해결할 수 있다면 문제도 해결될 것입니다.

사용하는 데이터 양이 적으면 대부분의 권장 사항은 그다지 중요하지 않습니다. 사용자가 원하는 방법을 동원하여 문제를 해결할 수 있습니다. 하지만 대량의 레코드나 많은 통합 문서, 여러 작성자가 관련된 경우 통합 문서의 디자인 품질이 낮으면 영향이 매우 커지므로 본 백서의 지침을 좀 더 숙고해야 합니다.

물론 연습은 아무리해도 지나치지 않으며 모든 통합 문서에 대해 이 지침을 따르는 것이 좋습니다. 그리고, 예상되는 운영 데이터 양에 따라 통합 문서를 테스트할 때까지는 디자인이 완전할 수 없음을 명심하시기 바랍니다.

한 가지 중요한 점은 본 문서 전체에서 Tableau Server 를 가리키지만 사내 배포가 아니라 호스팅 솔루션을 사용하려는 경우 이 지침은 Tableau Online 에도 대부분 적용될 수 있습니다. 서버 구성 매개 변수의 변경/조정 및 서버 계층의 소프트웨어 설치/업데이트에는 명백한 예외 사항이 있으며 특히 SaaS 환경에서는 이러한 사항을 고려해야 합니다.

물리 법칙

다양한 기능이 통합 문서의 성능에 영향을 미치는 방식을 기술적으로 세세하게 다루기에 앞서, 효율적인 대시보드와 보고서를 작성하는 데 도움이 되는 몇 가지 기본 원칙을 먼저 설명하도록 하겠습니다.

데이터 원본에서 느리면 Tableau 에서도 느립니다.

Tableau 통합 문서가 실행이 느린 쿼리를 기반으로 할 경우 통합 문서의 속도도 느려집니다. 다음 섹션에는 쿼리 실행에 소요되는 시간을 단축할 수 있도록 데이터베이스를 조정하는 방법이 설명되어 있습니다. 이에 덧붙여 Tableau 의 신속한 데이터 엔진을 사용하여 쿼리 성능을 개선하는 방법도 설명하도록 하겠습니다.

Tableau Desktop 에서 느리면 거의 언제나 Tableau Server 에서도 느립니다.

Tableau Desktop 에서 신속하게 작동하지 않는 통합 문서를 Tableau Server 에 게시한다고 해서 통합 문서의 속도가 빨라지지는 않습니다. 사용자는 로컬 PC 보다 서버에 CPU/RAM 등이 더 많기 때문에 Tableau Server 에서 통합 문서가 더 빠르게 실행될 것이라고 생각하는 경우가 많습니다. 일반적으로 통합 문서는 다음과 같은 이유로 Tableau Server 에서 약간 더 느리게 수행됩니다.

- 여러 사용자가 모든 서버 리소스를 공유하여 동시에 통합 문서를 생성합니다. 그렇지만 Tableau Server 내 캐싱 메커니즘으로 인해 공유를 시작할 때는 통합 문서가 더 빨리 응답한다고 생각할 수도 있습니다.
- 대시보드와 차트의 렌더링 작업이 클라이언트 워크스테이션이 아니라 서버에서 추가로 수행되어야 합니다.

먼저 Tableau Desktop 에서 통합 문서를 조정한 후에 Tableau Server 에서 성능 조정을 고려해야 합니다.

단, Tableau Desktop 의 리소스는 한도에 도달했지만 서버의 리소스는 여유가 있는 경우, 예를 들어, PC 의 RAM 이 분석하려는 데이터 볼륨을 지원하기에 부족하거나 서버와 데이터 원본과의 연결 속도가 빠르고 지연 시간이 낮은 경우에는 예외적으로 Tableau Server 의 속도가 더 빠를 수 있습니다. 일부 사용자의 경우 RAM 이 2GB 에 불과한 저사양 워크스테이션에서 데이터 집합을 가지고 작업을 수행할 때에는 속도가 느리거나 '메모리 부족' 오류가 발생하다가 Tableau Server 에 게시한 통합 문서에서 작업을 수행할 때에는 속도가 개선되기도 하는데 이는 서버의 메모리와 프로세스 성능이 워크스테이션보다 훨씬 뛰어나기 때문입니다.

성능이 더 뛰어난 최신 버전

Tableau 개발 팀은 소프트웨어의 성능과 유용성을 향상하기 위해 끊임없이 노력하고 있습니다. 때로는 Tableau Desktop 및 Tableau Server 를 최신 버전으로 업그레이드하면 통합 문서를 변경하지

않아도 뛰어난 성능 향상을 이룰 수 있습니다. 예를 들어 많은 고객들이 V8 에서 V9 로 업그레이드하는 것만으로도 통합 문서의 성능이 3 배 이상 향상되었다고 보고했으며, 성능 향상은 Tableau 10 이상에서도 계속 주력하고 있는 부분입니다. 물론 Tableau Online 은 출시될 때 항상 최신 버전으로 업그레이드되므로 이와 같은 문제가 없습니다.

이 문제는 유지 관리 버전과 주요/소규모 출시 버전에 해당됩니다. 다음 참고 페이지에서 Tableau 출시 주기와 각 출시별 세부정보에 대해 자세히 알아볼 수 있습니다.

http://www.tableau.com/ko-kr/support/releases

또한 데이터베이스 공급업체에서도 제품 향상을 위해 노력하고 있습니다. 다음 웹 페이지에서 적절한 데이터 원본 드라이버의 최신 버전을 사용하고 있는지 확인하십시오.

http://www.tableau.com/ko-kr/support/drivers

단순한 디자인

과유불급이란 말이 있듯이 아무리 좋은 것도 지나치면 되려 좋지 않은 결과를 초래하곤 합니다. 통합 문서를 만들 때에는 모든 항목을 하나의 단일화된 통합 문서에 억지로 포함시키려 하지 마십시오. Tableau 통합 문서는 50 개의 대시보드(각 대시보드에 20 개의 차트 개체), 50 개의 다른 데이터 원본 연결을 지원할 수 있지만 이 경우 통합 문서의 성능이 저하될 수 있습니다.

이와 같은 통합 문서가 있는 경우 이 통합 문서를 여러 개의 파일로 분할하는 것을 고려해 보십시오. 간편하게 통합 문서 간에 대시보드를 복사하기만 하면 Tableau 에서 연결된 모든 워크시트와 데이터 원본을 가져옵니다. 대시보드가 너무 복잡한 경우에는 대시보드를 단순화하고 상호 작용을 통해 최종 사용자를 각 보고서별로 지원하는 방법을 고려해 보십시오. 문서를 아무리 많이 만들어도 별도의 요금이 책정되지 않으므로 필요에 따라 데이터를 여러 개의 파일로 분할하시기 바랍니다.

성능과는 다른 확장성

확장성은 공유 통합 문서를 보는 사용자 수와 관련이 있는 반면, 성능은 개별 통합 문서의 실행 속도와 관련이 있습니다. 본 문서에 제공된 많은 권장 사항은 Tableau Server 에 게시된 통합 문서의 확장성에 긍정적인 영향을 주지만 본 문서의 기본 초점은 성능 향상에 맞춰져 있습니다.

도구별 차이점

통합 문서의 성능을 이해하려면 a) 발생하는 상황과 b) 소요되는 시간을 이해해야 합니다. 이 정보는 통합 문서를 실행하는 위치(예: Tableau Desktop 또는 Tableau Server)에 따라 여러 세부 수준으로 수집되었습니다. 이 섹션에서는 사용 가능한 다양한 옵션에 대해 간략하게 설명합니다.

성능 기록기

성능 정보를 가장 먼저 찾아야 하는 곳은 Tableau Desktop 과 Tableau Server 의 성능 기록기 기능입니다. Tableau Desktop 에서는 다음과 같이 도움말 메뉴에서 이 기능을 활성화합니다.

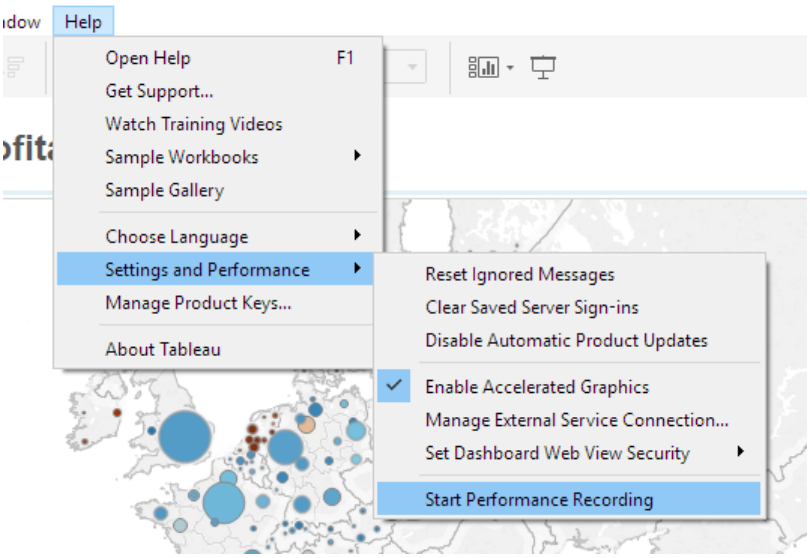
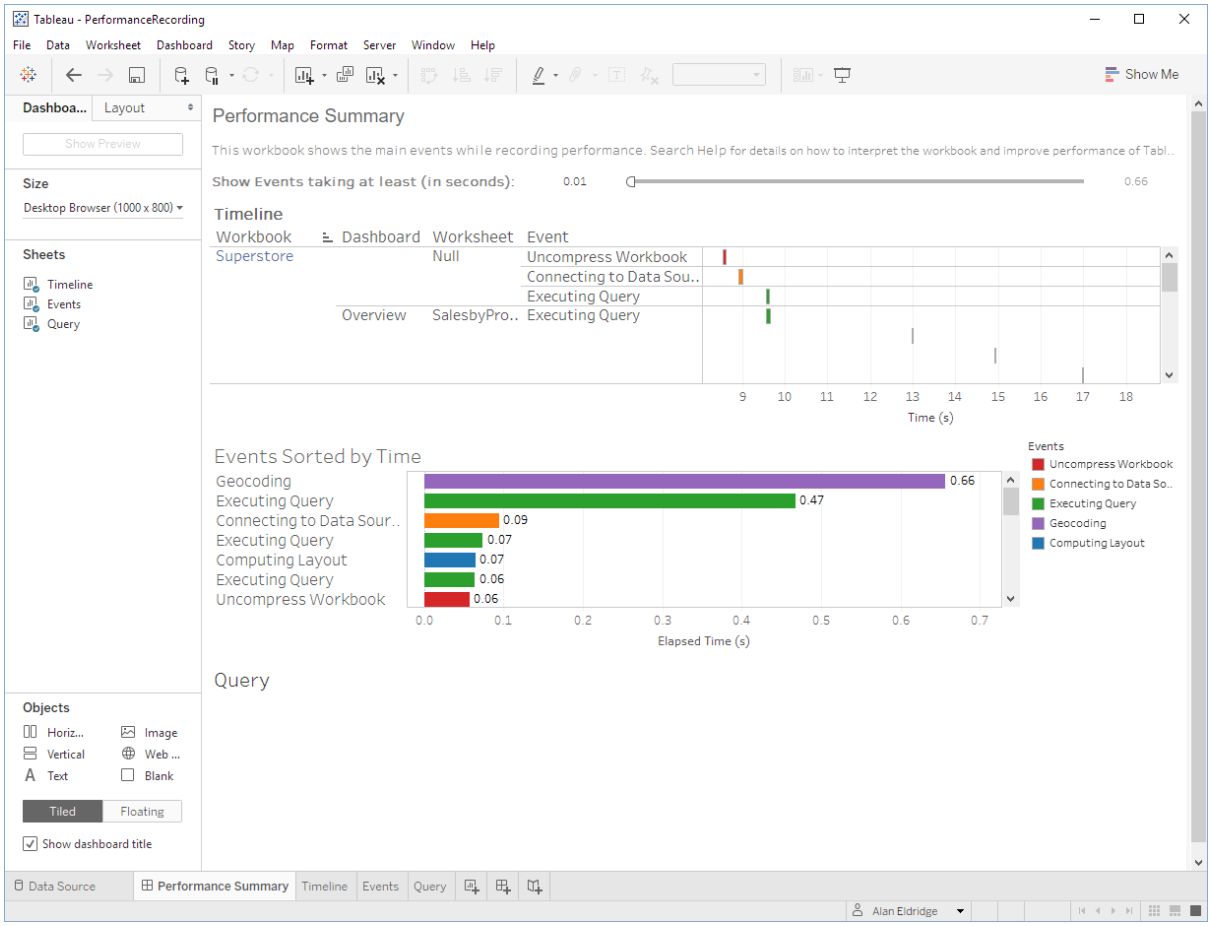


Tableau 를 실행하고 성능 기록을 시작한 다음 통합 문서를 엽니다. 이 작업을 수행하는 동안 의도치 않게 리소스를 놓고 경쟁하지 않도록 다른 통합 문서는 열지 않는 것이 좋습니다. 최종 사용자의 입장에서 통합 문서와 상호작용하고 데이터를 충분히 수집했다고 생각하면 도움말 메뉴로 돌아가 기록을 중지합니다. 이렇게 하면 데이터가 캡처된 또 다른 Tableau Desktop 창이 열립니다.



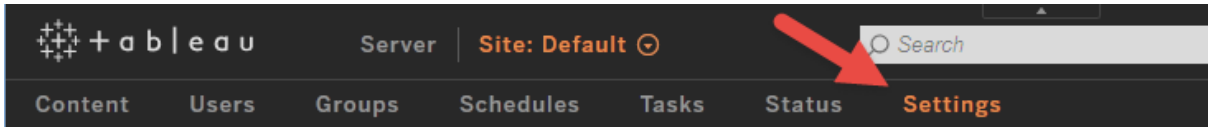
이제 통합 문서에서 대부분의 시간을 차지하는 동작을 확인할 수 있습니다. 예를 들어 위의 이미지에서는 Timeline 워크시트에서 선택한 쿼리를 완료하는 데 30.66 초가 소요됩니다. 막대를 클릭하면 실행 중인 쿼리의 텍스트가 표시됩니다. 성능 기록기는 Tableau 통합 문서로 출력되므로 추가 뷰를 만들어서 이 정보를 다른 방법으로 살펴볼 수도 있습니다.

> 참고: 기본적으로 0.1 초 미만이 소요되는 이벤트는 표시되지 않습니다. 이러한 이벤트는 대시보드 상단에 있는 필터를 조정하여 표시할 수 있지만 오래 실행하는 작업에 초점을 맞춰야 하기 때문에 1 초로 설정하는 것이 좋습니다.

Tableau Server 에서 성능 기록을 생성하여 통합 문서가 게시될 때 발생하는 문제를 식별할 수도 있습니다. Tableau Server 에서는 성능 기록이 기본적으로 활성화되지 않습니다. 이 기능은 사이트별로 조정할 수 있습니다.

서버 관리자는 사이트별로 성능 기록을 활성화할 수 있습니다.

1. 성능 기록을 활성화하려는 사이트로 이동합니다.
2. **설정**을 클릭합니다.



3. 통합 문서 성능 메트릭에서 **통합 문서 성능 메트릭 기록**을 선택합니다.
4. **저장**을 클릭합니다.

성능 기록 생성 방법

1. 성능을 기록하려는 뷰를 엽니다. 뷰를 열 때 Tableau Server 에서 URL 다음에 ':iid=<*n*>'을 추가합니다. 이것은 세션 ID 입니다. 예를 들면 다음과 같습니다.

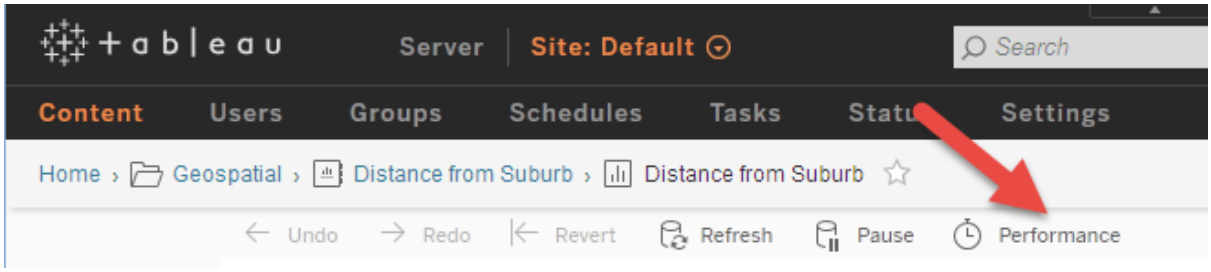
   ```
   http://<tableau_server>/#/views/Coffee_Sales2013/USSalesMarginsByAreaCode?:iid=1
   ```

2. 뷰 URL 끝에, 세션 ID 직전에 :record_performance=yes&를 입력합니다. 예를 들면 다음과 같습니다.

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

3. 뷰를 로드합니다.

성능 기록이 시작되었는지 보려면 뷰 툴바에서 **성능** 옵션을 확인합니다.



작업을 완료하면 성능 기록을 볼 준비가 됩니다.

1. **성능**을 클릭하여 성능 통합 문서를 엽니다. 현재까지의 성능 데이터 스냅샷이 표시됩니다. 뷰를 사용하면서 계속 추가 스냅샷을 찍을 수 있으며 성능 데이터는 누적됩니다.
2. 기록을 중지하려면 다른 페이지로 이동하거나 URL 에서 `:record_performance=yes` 를 삭제합니다

이 정보를 활용하면 통합 문서에서 어떤 섹션을 가장 먼저 검토해야 하는지 알 수 있습니다. 예를 들어 위 예에서는 소요 시간을 가장 많이 개선할 수 있는 섹션이 어디인지 파악할 수 있습니다. 기록을 해석하는 방법에 대한 자세한 내용을 확인하려면 다음 링크를 참조하십시오.

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

로그

Tableau 에서는 로그 파일을 통해 전체 쿼리 텍스트를 확인할 수 있습니다. 기본 위치는 `C:\Users\<username>\Documents\My Tableau Repository\Logs\log.txt` 입니다. 파일이 꽤 길고 JSON 인코딩 텍스트로 작성되었으므로 제대로 파악하려면 Notepad++ 또는 Sublime 등 쓸만한 텍스트 편집기를 사용하는 것이 좋습니다. 'begin-query' 또는 'end-query'를 검색하면 해당 데이터 원본으로 전달되는 쿼리 문자열을 찾을 수 있습니다. 'end-query' 로그 레코드에는 쿼리 실행에 걸린 시간과 Tableau 에 반환된 레코드 수도 표시됩니다.

```
{"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 에서는 로그가 `C:\ProgramData\Tableau\Tableau Server\data\tabsvc\vizqlserver\Logs` 에 있습니다.

Tableau Server 성능 뷰

Tableau Server 는 Tableau Server 의 모니터링 작업을 지원하는 다양한 관리자용 뷰를 제공합니다. 이러한 뷰는 서버의 유지 관리 페이지에 있는 분석 테이블에서 확인할 수 있습니다.

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

이 뷰에 대한 자세한 내용을 확인하려면 다음 링크를 참조하십시오.

http://onlinehelp.tableau.com/current/server/ko-kr/adminview.htm

또한 Tableau 리포지토리를 구성하는 PostgreSQL 데이터베이스에 연결하여 사용자 지정 관리 뷰를 생성할 수도 있습니다. 이에 대한 안내는 다음 페이지를 참조하십시오.

http://onlinehelp.tableau.com/current/server/ko-kr/adminview_postgres.htm

모니터링 및 테스트

TabMon

TabMon 은 시간별 성능 통계를 수집할 수 있는 Tableau Server 의 오픈 소스 클러스터 모니터링 기능입니다. 또한 커뮤니티에서 사용할 수 있도록 Tableau 에서는 MIT 오픈 소스 라이선스에 따라 전체 소스 코드를 공개하고 있습니다.

TabMon 은 시스템 상태와 애플리케이션 메트릭을 즉시 기록합니다. 네트워크 전체에서 Tableau Server 시스템의 Windows Perfmon, Java Health 및 Java Mbean(JMX) 카운터와 같은 기본 제공되는 메트릭을 수집합니다. TabMon 을 사용하여 물리적 리소스(CPU, RAM), 네트워크 및 하드 디스크 사용을 모니터링할 수 있습니다. 이를 통해 캐시 적중률, 요청 지연 시간, 활성 세션 등을 추적할 수 있습니다. 또한 깔끔하고 통일된 구조로 표시되어 데이터를 Tableau Desktop 에서 손쉽게 시각화할 수 있습니다.

TabMon 을 사용하면 수집할 메트릭과 모니터링할 시스템을 완전히 제어할 수 있으며 스크립트 작성이나 코딩이 필요하지 않습니다. 시스템과 메트릭 이름만 알면 됩니다. 또한 클러스터와 관계없이 원격으로 실행할 수 있습니다. 운영 시스템에 로드를 거의 추가하지 않고 네트워크의 어느 컴퓨터에서든 클러스터의 상태를 모니터링, 집계 및 분석할 수 있습니다.

TabMon 에 대한 자세한 내용은 다음 페이지를 참조하십시오.

http://bit.ly/1ULFelf

TabJolt

TabJolt 는 Tableau Server 에서 간편하게 사용하도록 특별히 설계된 '포인트 앤 런(point-and-run)' 로드 및 성능 테스트 도구입니다. 기존 로드-테스트 도구와 달리 TabJolt 는 스크립트 개발이나 유지 관리가 없이 Tableau Server 에서 자동으로 로드를 생성할 수 있습니다. TabJolt 는 Tableau 의 프레젠테이션 모델을 인식하므로 비주얼라이제이션을 자동으로 로드하고 테스트 중의 상호 작용을 식별할 수 있습니다.

이 도구를 사용하여 서버에서 TabJolt 가 하나 이상의 통합 문서를 가리키도록 하면 Tableau 뷰에서 상호 작용을 자동으로 로드하고 실행할 수 있습니다. 또한 TabJolt 는 평균 응답 시간, 처리량, 95%의 응답 시간을 수집하고 Windows 성능 메트릭을 수집하여 상관 관계를 파악합니다.

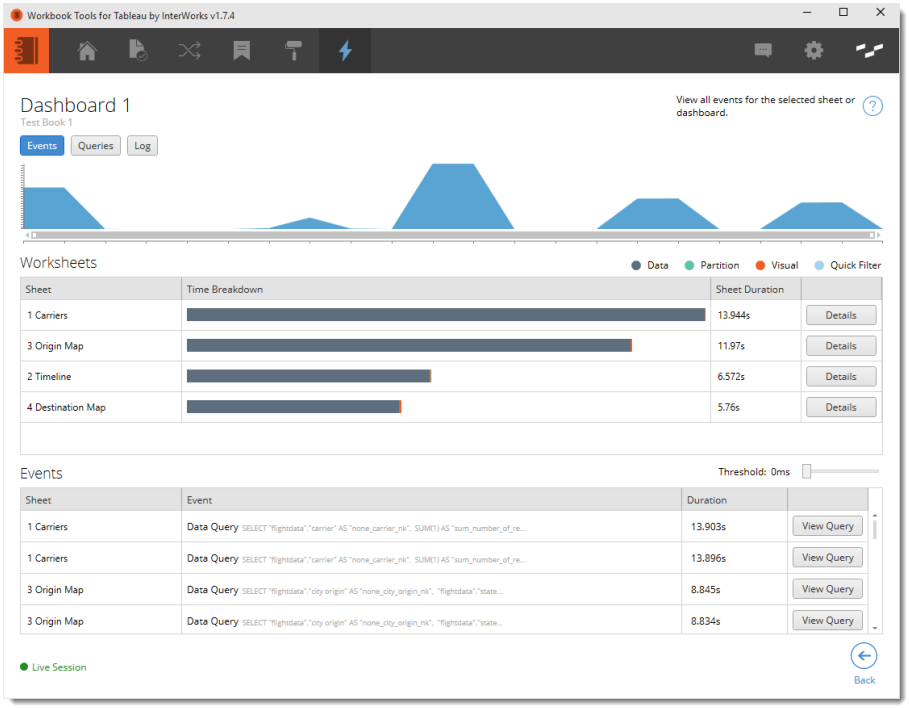
물론 TabJolt 사용자는 Tableau Server 아키텍처에 대한 충분한 지식을 보유하고 있어야 합니다. Tableau Server 를 로드 테스트용 블랙 박스로 취급하는 것은 권장되지 않으며 예상치 못한 결과가 나올 수 있습니다. 자동화 도구인 TabJolt 가 광범위한 사용자 상호 작용 기능을 모두 복제할 수는 없으므로 TabJolt 결과가 실제 결과를 반영하는지 신중하게 판단해야 합니다.

TabJolt 에 대한 자세한 내용은 다음 페이지를 참조하십시오.

http://bit.ly/1ULFtgi

기타 도구

통합 문서의 성능 특성 파악에 사용할 수 있는 여러 가지 타사 도구가 있습니다. Interworks 의 'Power Tools for Tableau'가 그중 하나입니다. 이 도구에는 시간이 오래 걸리는 시트와 쿼리를 자세히 알아보고 파악할 수 있도록 하는 성능 분석기(기본 제공되는 성능 기록기와 유사)가 포함되어 있습니다.



Palette Software 의 Palette Insight 라는 제품도 있습니다. 이 제품은 Tableau Server 에서 성능 정보를 수집하여 용량 계획을 수행하고 리소스를 많이 사용하는 사용자와 통합 문서를 식별하며 사용자 액세스 감사를 수행하고 차지백 모델을 구축합니다.

또한 최신 DBMS 플랫폼에는 실행 중인 쿼리를 추적하고 분석할 수 있는 관리 도구가 포함되어 있습니다. 성능 기록에 쿼리 실행 시간이 주요 요소로 나타나는 경우 DBA 가 큰 도움이 될 수 있습니다.

클라이언트 브라우저와 서버 간 상호 작용이 문제라고 생각되는 경우 Telerik Fiddler 같은 도구나 브라우저의 개발자 도구를 사용하여 클라이언트와 서버 간의 트래픽을 자세히 살펴볼 수도 있습니다.

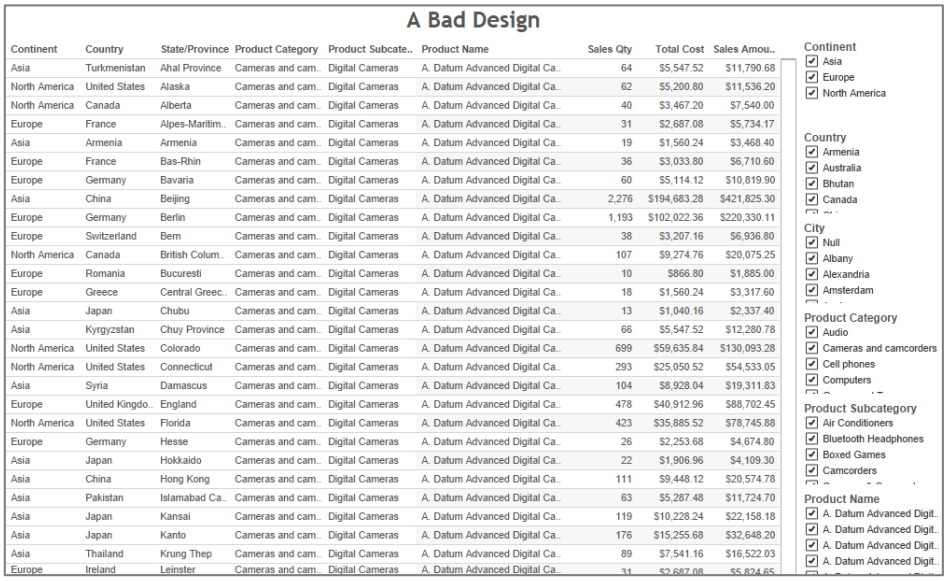
통합 문서 디자인

Tableau 에서 작업을 수행하는 것은 많은 사용자에게 생소한 일이며 효율적인 통합 문서를 만들기 위해서는 몇 가지 디자인 기술과 우수 사례를 학습해야 합니다. 하지만 Tableau 를 처음 사용하는 많은 사용자들이 기존의 디자인 접근 방식을 적용했다가 Tableau 의 장점이 전혀 발휘되지 않은 결과를 마주하곤 합니다. 이 섹션에서는 우수 사례를 반영하는 몇 가지 디자인 원칙에 대해 설명합니다.

좋은 대시보드 디자인

Tableau 는 최종 사용자에게 상호 작용이 가능한 환경을 제공하며 Tableau Server 는 궁극적으로 데이터를 단순히 보는 것이 아니라 상세하게 살펴볼 수 있는 상호 작용이 가능한 애플리케이션을 제공합니다. 따라서 효율적인 Tableau 대시보드를 만들기 위해서는 정적인 보고서를 작성할 때와는 전혀 다른 사고방식이 필요합니다.

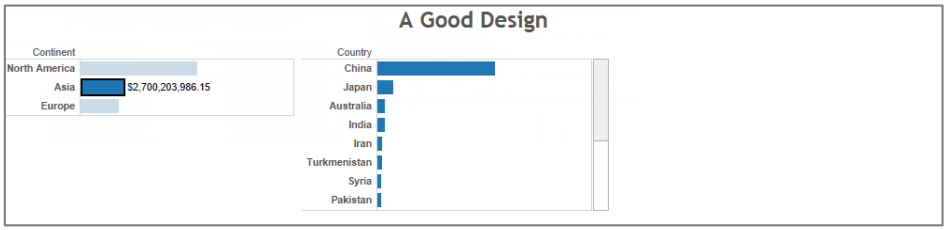
다음은 대다수의 신규 작성자, 특히 Excel 이나 Access 와 같은 도구를 사용하여 작업을 수행한 경험이 있거나 '기존' 보고 도구를 사용한 경험이 있는 작성자가 생성하는 대시보드의 예입니다. 기본적으로 보고서는 '모든' 항목이 표시된 테이블 형식으로 구성되며 사용자는 관심이 있는 몇 개의 레코드가 표시될 때까지 일련의 필터를 사용하여 테이블의 검색범위를 좁혀야 합니다.



이는 '좋은' Tableau 대시보드의 예가 아니며 단순히 대시보드만 놓고 보더라도 전혀 '좋은' 대시보드라 할 수 없습니다. 비판적인 관점에서 볼 때 이 대시보드는 데이터를 추가로 분석하거나 차트로 표시하려면 Excel 과 같은 별도의 도구를 사용해야 한다는 점에서 보기 좋게 꾸며놓은 데이터 추출 프로세스에 불과합니다. 또 최대한 긍정적인 관점에서 보더라도 이 대시보드는 최종 사용자가 어떤 방식으로 데이터를 살펴볼지 전혀 파악되지 않기 때문에 '기본적으로 제시된 기준으로 볼 때 모든 데이터가 제공되었고 몇 가지 필터를 사용할 수 있으므로 내가 원하는 결과 집합을 찾을 때까지 검색 범위를 좁히는' 접근방식을 선택할 수 밖에 없습니다.

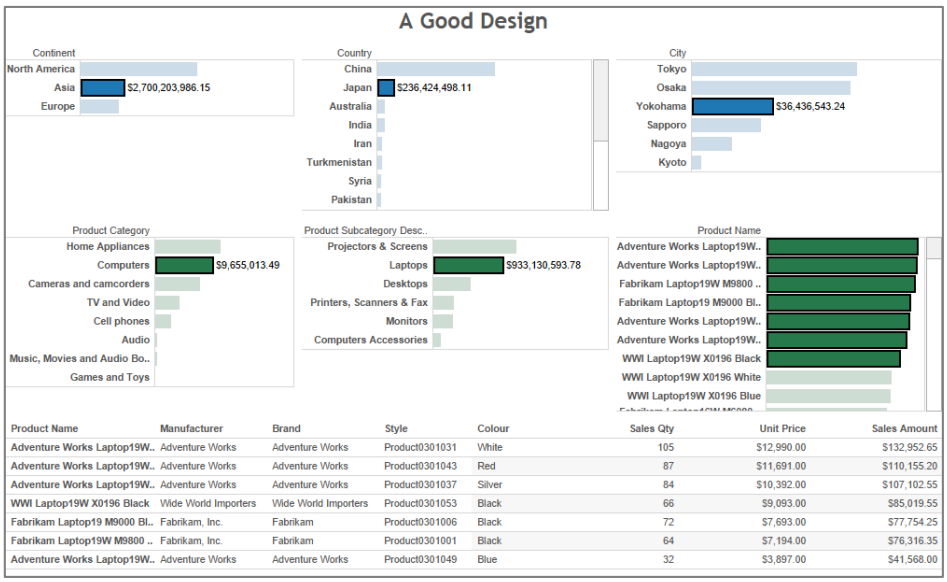
그렇다면 이제 동일한 데이터를 가지고 다른 방식으로 대시보드를 만들어 보도록 하겠습니다. 먼저 데이터를 최상위 수준으로 집계합니다.

요소를 선택하면 다음 세부 수준이 표시됩니다.



특정 요소를 선택할 때마다 그 다음 단계의 추가 세부정보가 표시됩니다.

최종적으로 위 크로스탭 대시보드에 표시된 데이터와 동일한 수준의 데이터에 도달합니다.



대시보드를 만들 때에는 데이터를 어떻게 표시할지가 아니라(데이터 표시 방식은 매우 중요하지만 이 주제는 추후에 다루도록 하겠습니다.) 대시보드 사용 경험이 어떠할지를 중점적으로 생각해야 합니다. 두 번째 예는 흐름이 자연스러우며 왼쪽에서 오른쪽으로, 위에서 아래로 진행됩니다. 이 예의 각 단계는 수많은 데이터로 구성되어 있지만 대시보드는

최종 사용자가 각 단계를 순서대로 이동하여 원하는 세부 레코드만 집중적으로 찾을 수 있도록 유도합니다.

두 예의 기본적인 차이는 바로 분석 프로세스에서 최종 사용자를 안내하는 방식에 있습니다. 첫 번째 예는 살펴볼 수 있는 모든 레코드를 한 번에 제시하여 시작 범위가 매우 넓으며 최종 사용자는 필터를 적용하여 레코드를 표시하는 범위를 좁혀야 합니다. 이 기법의 근본적인 문제는 다음과 같습니다.

- 최종 사용자에게 레코드를 표시하기 전에 먼저 '모든 레코드를 제공해 달라는' 쿼리를 실행해야 하는데 이는 너무 광범위합니다. 실제 상황에서 이 쿼리를 실행하고 쿼리에 대한 데이터를 Tableau 엔진에 다시 제공하려면 데이터 집합의 종류에 관계 없이 상당한 시간이 필요합니다. 대시보드를 '처음 접할 때의' 경험은 솔루션에 대한 최종 사용자의 인식을 형성하는 데 매우 중요하며 작업을 본격적으로 시작하기도 전에 몇 초 이상의 시간을 허비해야 한다면 솔루션에 대한 인식은 부정적으로 형성될 수밖에 없습니다.
- 수십만에서 수백만에 달하는 마크(크로스탭에서 각 셀을 지칭하는 단위)가 포함된 뷰를 만들기 위해서는 상당한 양의 CPU 및 메모리가 필요합니다. 이에 덧붙여 시간도 오래 걸려 시스템의 응답 역량에 대한 부정적인 인식을 심어주게 됩니다. Tableau Server 에서 여러 명의 사용자가 대용량 크로스탭을 동시에 생성할 경우 성능이 저하될 수 있으며 최악의 경우에는 시스템의 메모리가 부족한 상황이 발생할 수도 있습니다. 이로 인해 서버 안정성 문제, 오류 및 최종 사용자의 사용 경험을 저해하는 그 밖의 다양한 문제가 야기될 수 있습니다. 이러한 문제를 최소화하기 위해 서버에 메모리를 추가할 수는 있지만 이는 증상에 대한 대응일 뿐, 문제에 대한 근본적인 해결책은 아닙니다.
- 마지막으로 이 대시보드는 문맥에 따른 안내를 전혀 제공하고 있지 않아 초기 필터의 범위가 너무 넓은지 아니면 너무 좁은지를 사용자가 전혀 파악할 수 없습니다. 따라서 보고서 작성자는 처음 쿼리를 실행할 때 모든 범주를 선택할 경우 수만 개에 달하는 레코드가 반환되며 서버의 모든 RAM 이 이 작업에만 사용되어 다른 작업은 진행할 수 없다는 것을 직접 깨닫는 수밖에 없습니다.

이를 두 번째 접근 방식과 대비해 보십시오. 두 번째 접근 방식의 경우 첫 번째 쿼리를 실행하면 데이터를 최상위 수준으로 집계한 결과만 표시됩니다.

- 가장 처음 실행하는 쿼리가 데이터를 최상위 수준으로 집계하여 소수의 레코드만 반환합니다. 우수하게 설계된 데이터베이스의 경우 이 작업은 매우 효율적으로 진행되며 제품을 '처음 접할 때의' 응답 시간도 크게 단축되므로 시스템에 대한 긍정적인 인식을 심어줄 수 있습니다. 각 단계를 이동할 때 사용하는 쿼리의 경우에도 바로 위 단계에서 선택한 항목에 따라 데이터를 집계하고 이에 따라 제한되므로 신속하게 실행하고 결과를 Tableau 엔진에 반환할 수 있습니다.
- 대시보드 완료 시 제공되는 뷰는 더 많지만 각 뷰에는 몇 십개에 불과한 마크만 표시됩니다. 시스템을 사용하는 최종 사용자가 많은 경우에도 각 뷰를 생성하는 데 필요한 리소스는 극히 적으므로 시스템의 메모리가 부족할 가능성이 크게 줄어듭니다.
- 마지막으로 이 예에서는 높은 '탐색' 단계에서도 범주별 판매량을 표시할 수 있어 선택 항목에 포함된 레코드가 많을지 아니면 적을지 여부를 사용자가 문맥에 따라 파악할 수 있습니다. 뿐만 아니라 이 예는 각 범주의 수익성을 서로 다른 색상으로 표시하고 있는데, 이러한 방식은 대시보드를 단순히 탐색하는 데 그치지 않고 어떤 영역을 주의깊게 살펴봐야 하는지 손쉽게 파악할 수 있도록 도와줍니다.

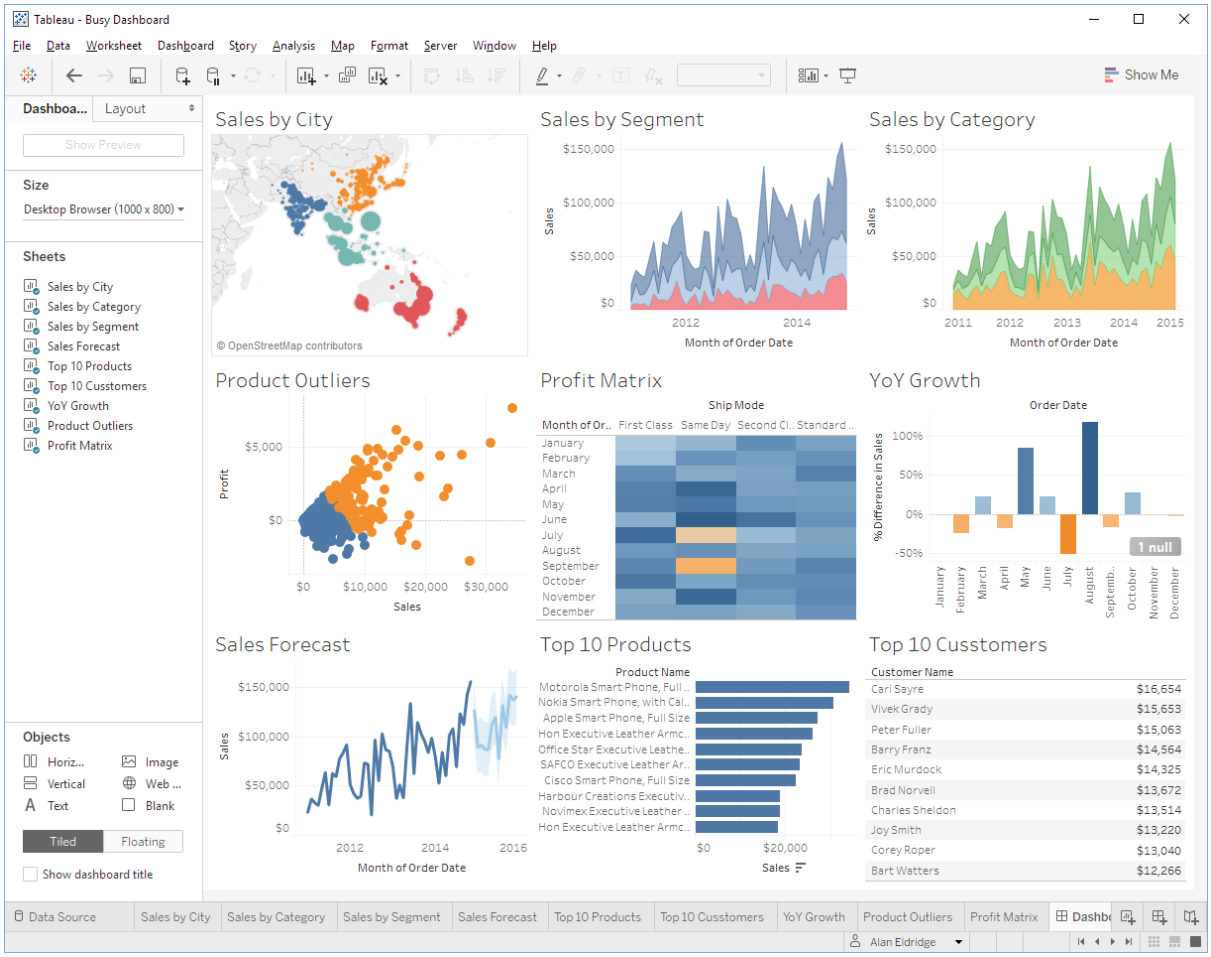
단순하게 만들기

신규 사용자가 흔히 하는 실수는 지나치게 '복잡한' 대시보드를 만든다는 것입니다. 이러한 실수는 다른 도구에서 이전에 사용했던 문서를 다시 만들려고 하거나 인쇄된 보고서 형식으로 특별히 디자인된 문서를 만들려다 발생했을 수 있습니다. 그 결과 통합 문서가 느리고 비효율적으로 수행됩니다.

복잡성을 유발하는 요소는 다음과 같습니다.

대시보드당 워크시트가 너무 많음

신규 사용자의 흔한 실수는 대시보드 하나에 너무 많은 차트/워크시트를 삽입하려는 것입니다.



각 워크시트는 데이터 원본에 대해 대개 하나 이상의 쿼리를 실행하므로 시트가 많을수록 대시보드를 렌더링하는 데 더 많은 시간이 소요됩니다. Tableau 는 최종 사용자에게 상호 작용 대시보드를 제공하도록 설계되었으므로 이를 활용하여 여러 대시보드/페이지로 데이터를 분산하십시오.

필터 카드가 너무 많음

필터 카드는 최종 사용자에게 다양한 기능이 포함된 상호 작용 대시보드를 만들 수 있도록 지원하는 Tableau 의 매우 강력한 기능입니다. 하지만 옵션을 열거하기 위해 각 필터에 쿼리가 필요할 수 있으므로 대시보드에 필터 카드를 너무 많이 추가하면 대시보드가 렌더링하는 데 예상외로 시간이 오래 걸릴 수 있습니다. 또한 필터에 '관련 값 표시'를 사용하면 다른 필터가 변경될 때마다 표시된 값을 업데이트하기 위한 쿼리가 필요합니다. 따라서 이 기능은 되도록 사용을 자제하는 것이 좋습니다.

또한 여러 워크시트에 필터를 적용한 경우 변경할 때마다 여러 쿼리가 실행되어 표시된 모든 워크시트가 업데이트되므로(표시되지 않은 워크시트는 실행되지 않음) 유의해야 합니다. 업데이트가 완료되는 데 몇 초가 소요된다면 사용자에게 부정적인 인식을 심어주게 됩니다. 사용자가 다중 선택 필터 유형을 여러 가지로 변경할 것으로 예상되는 경우 선택 내용 변경을 완료하면 업데이트를 실행할 수 있도록 '적용' 버튼을 표시하는 것이 좋습니다.

성능 향상을 위한 대시보드 조정

대시보드가 최대한 단순해졌다는 확신이 들면 캐싱을 이용하도록 디자인을 조정하여 성능을 더욱 향상할 수 있습니다.

고정된 크기의 대시보드

성능 향상을 위해 할 수 있는 가장 쉬운 것은 대시보드가 고정된 크기인지 확인하는 것입니다.

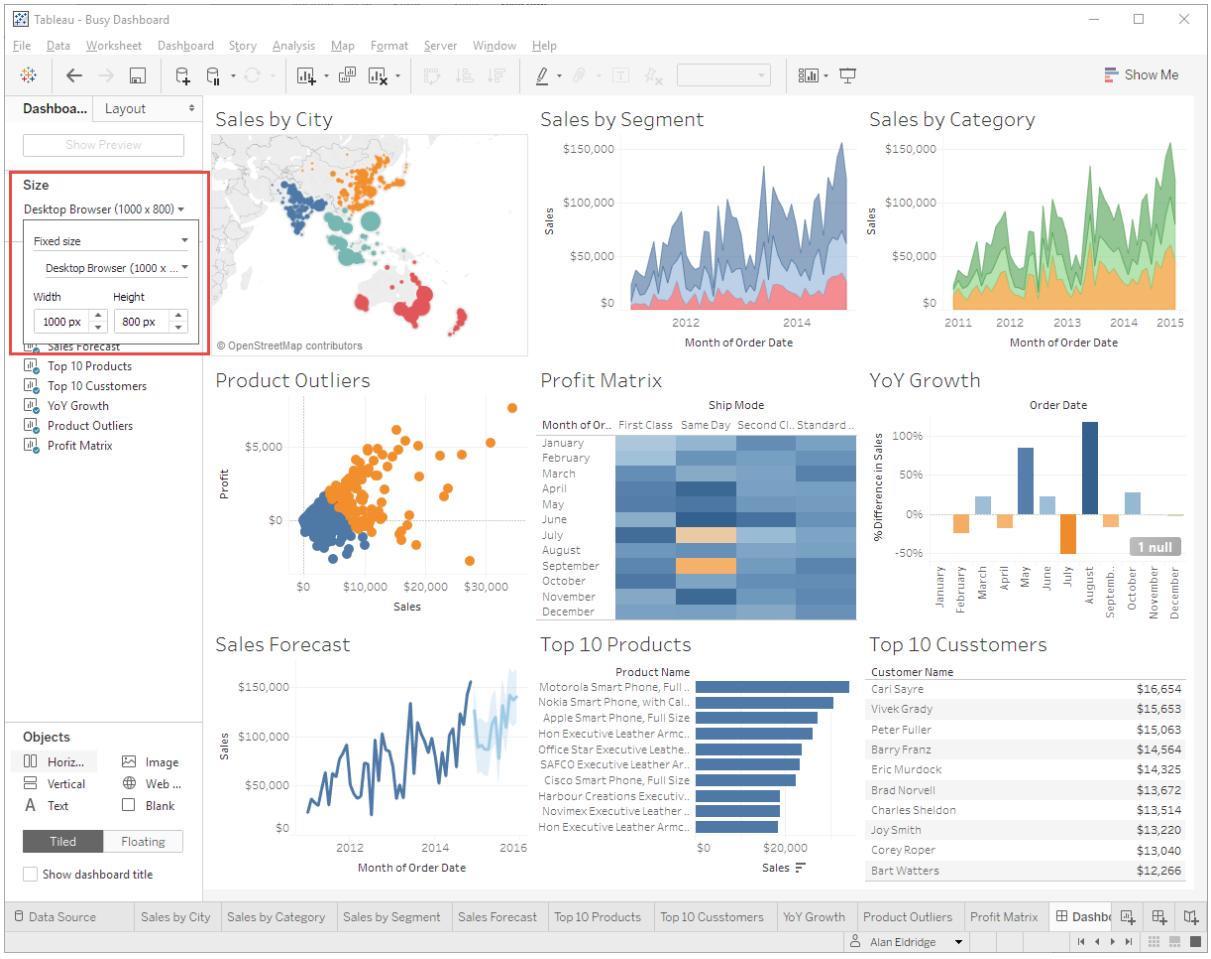


Tableau 에서는 렌더링 프로세스의 일부로 레이아웃, 즉 작은 배수와 크로스탭에 표시할 행/열 개수, 축 눈금선/격자선 개수 및 간격, 표시될 마크 레이블의 개수와 위치 등을 만듭니다. 이러한 레이아웃은 대시보드가 표시될 창의 크기로 결정됩니다.

동일한 대시보드에 대한 여러 요청이 다양한 크기의 창에서 오는 경우 요청에 대한 레이아웃을 각각 생성해야 합니다. 대시보드 레이아웃을 고정된 크기로 설정하면 단일 레이아웃만 만들어 모든 요청에서 재사용할 수 있습니다. 서버 쪽 렌더링의 경우 고정된 크기의 대시보드를 사용하면 서버에서 렌더링되는 비트맵을 캐싱하고 공유하여 성능과 확장성을 모두 향상할 수 있기 때문에 훨씬 더 중요합니다.

기기별 대시보드

Tableau 10 에는 기기별 대시보드라는 새로운 기능이 도입되었습니다. 이 기능을 사용하면 사용 중인 기기에 따라 자동 선택되는 사용자 지정 대시보드 레이아웃을 만들 수 있습니다.

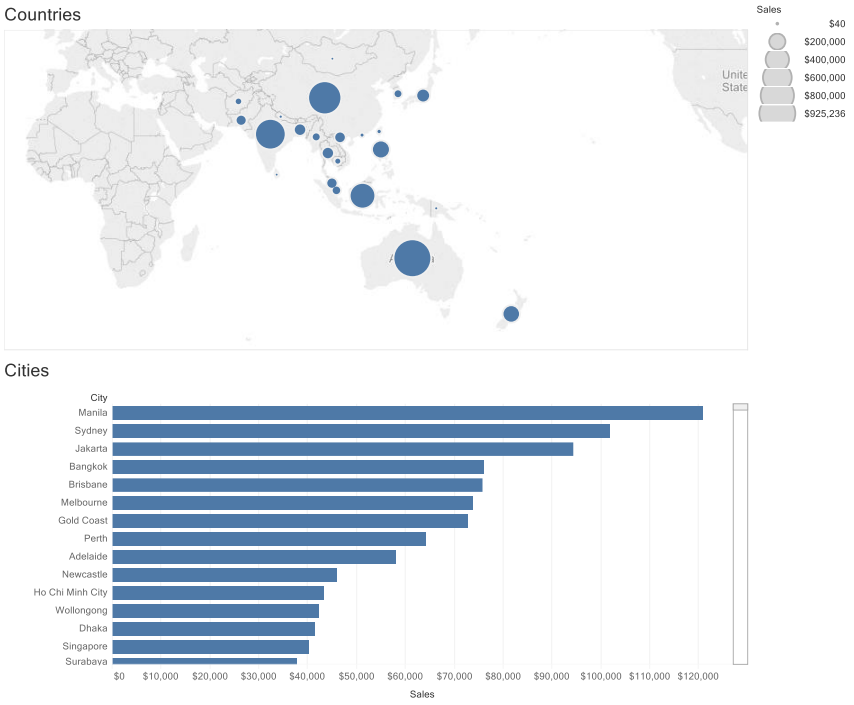
다음과 같이 화면 크기에 따라 사용할 레이아웃을 선택합니다.

- 가장 작은 축에서 500 픽셀 이하 – 휴대폰
- 가장 작은 축에서 800 픽셀 이하 – 태블릿
- 800 픽셀 이상 – 데스크톱

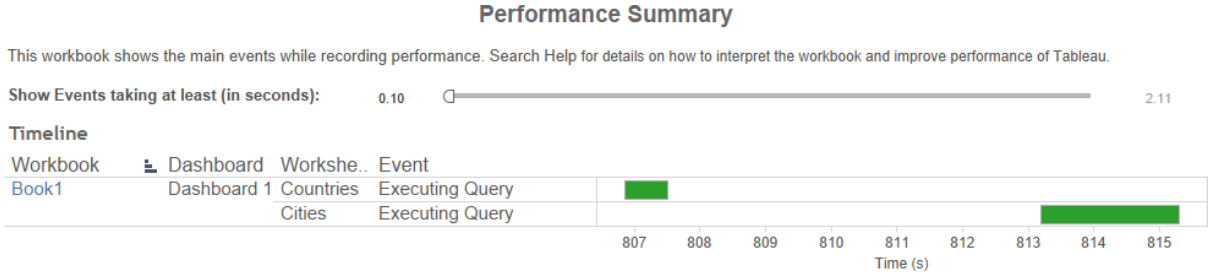
화면 크기가 다양하고 기기가 회전될 수 있기 때문에 일반적으로 사용자는 휴대폰/태블릿 레이아웃이 자동으로 크기 조정되길 원합니다. 이렇게 지정하면 모든 기기에서 최상의 화면 환경을 제공하겠지만 캐시(프레젠테이션 모델 캐시 및 서버 쪽 렌더링용 이미지 타일 캐시) 재사용에 영향을 줍니다. 일반적으로 기기 크기를 적절히 조정하여 얻는 이점이 캐싱으로 인한 이점보다 크긴 하지만 고려해 볼만한 사항입니다. 사용자가 통합 문서를 사용하기 시작하면 가장 일반적인 화면 크기에 맞는 모델과 비트맵들로 구성되어 성능이 향상됩니다.

비주얼라이제이션 세부 수준을 사용하여 쿼리 줄이기

대개 각 워크시트에 필요한 필드만 사용하는 것이 좋지만 때로는 한 워크시트에서 더 많은 정보를 가져옴으로써 다른 워크시트로의 쿼리를 방지하여 성능을 향상할 수도 있습니다. 다음과 같은 대시보드를 가정해 보겠습니다.

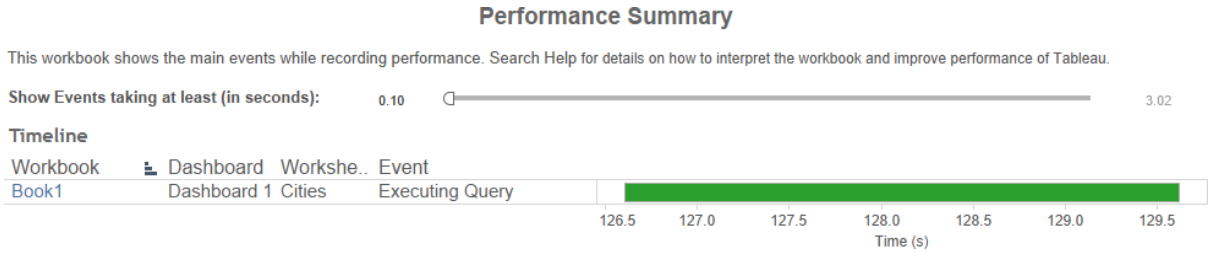


대시보드를 아래와 같이 구축하면 워크시트당 하나씩 두 개의 쿼리가 실행됩니다.



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

대시보드 디자인을 변경하고 Country 를 Detail 선반의 Cities 워크시트에 추가하면 Tableau 에서 단 하나의 쿼리로 대시보드를 완료할 수 있습니다. Tableau 에서는 Cities 워크시트에 대한 쿼리를 먼저 실행한 다음 쿼리 결과 캐시를 사용하여 Countries 워크시트에 대한 데이터를 제공합니다. 이 기능을 '쿼리 일괄 처리'라고 합니다.

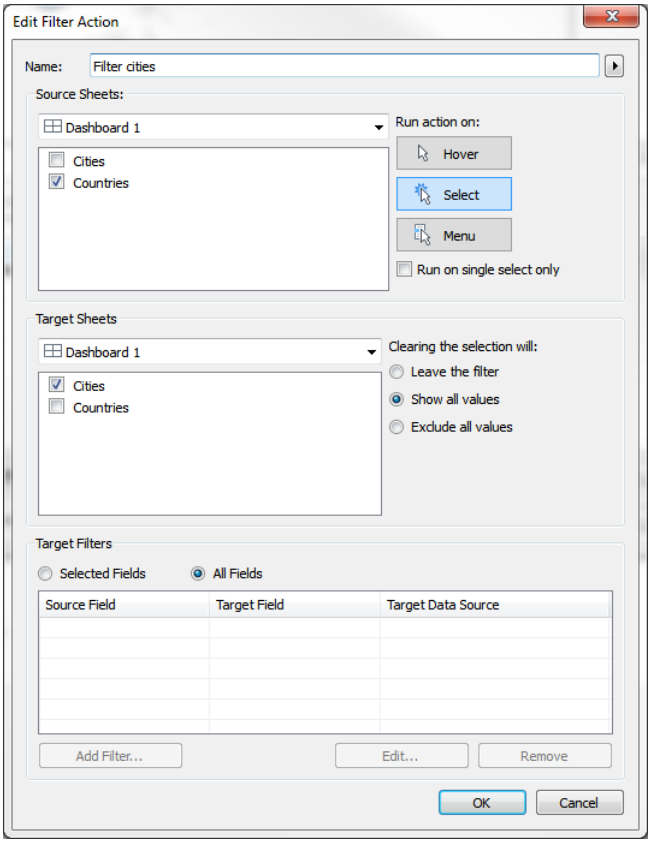



```
SELECT [Superstore APAC].[City] AS [City],
  [Superstore APAC].[Country] AS [Country],
  SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok]
FROM [dbo].[Superstore APAC] [Superstore APAC]
GROUP BY [Superstore APAC].[City],
  [Superstore APAC].[Country]
```

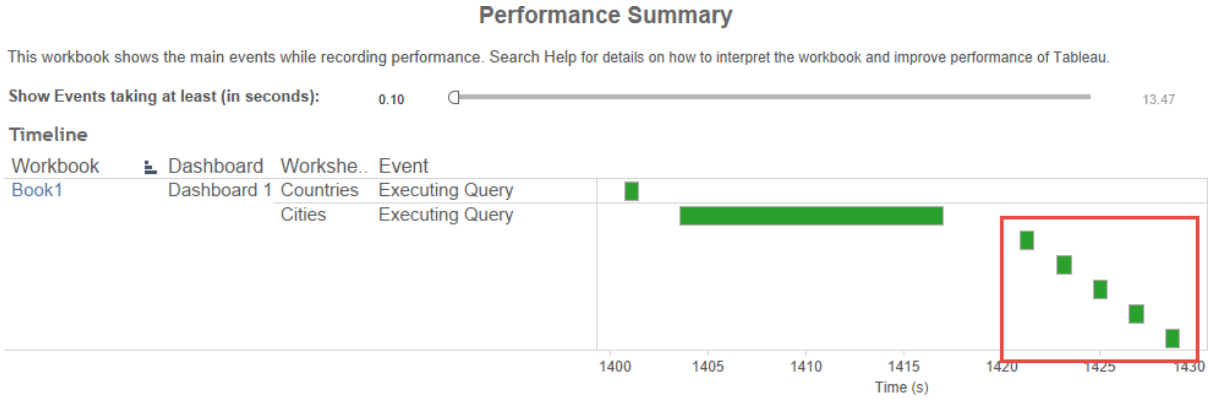
비주얼라이제이션에 차원을 추가하면 세부 수준이 변경되어 마크가 더 표시될 수 있으므로 모든 경우에 적용되는 것은 아닙니다. 하지만 위의 예에서와 같이 데이터에 계층 관계가 있다면 이 기법이 시각적 세부 수준에 영향을 주지 않으므로 유용합니다.

비주얼라이제이션 세부 수준을 사용하여 동작 최적화

비슷한 방법을 동작에 사용하여 실행해야 하는 쿼리 수를 줄일 수 있습니다. 최적화 전에 위와 동일한 대시보드를 가정하고 Countries 워크시트에서 Cities 워크시트로 필터 동작을 추가해 보겠습니다.



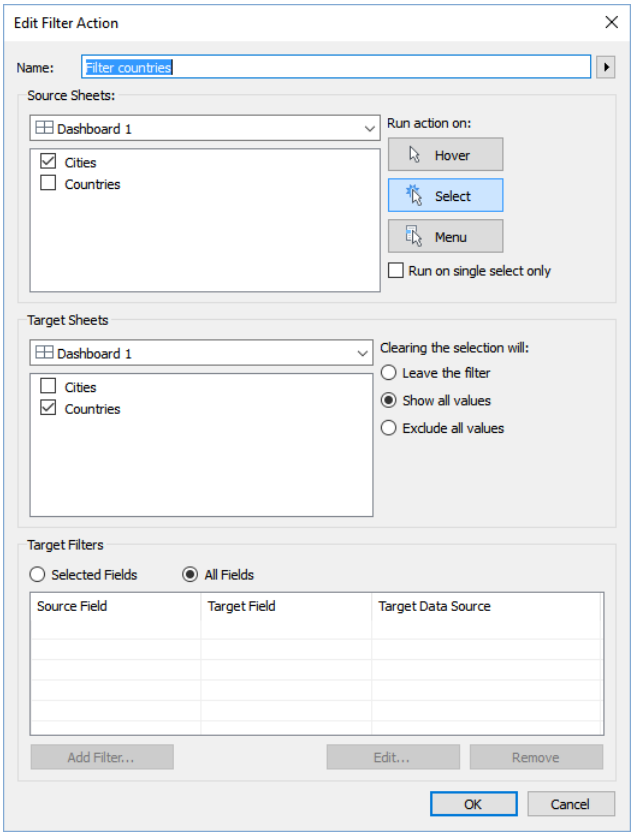
맵의 마크를 클릭하여 이 동작을 실행하면 Tableau 에서 Cities 워크시트의 값을 파악하기 위해 쿼리를 실행해야 합니다. 도시-국가 관계에 대한 쿼리 결과 캐시에 데이터가 없기 때문입니다.




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

Country 를 Cities 워크시트에 추가하면 이제 쿼리 결과 캐시에 정보가 충분하여 데이터 원본으로 돌아가지 않아도 이 필터를 수행할 수 있습니다.

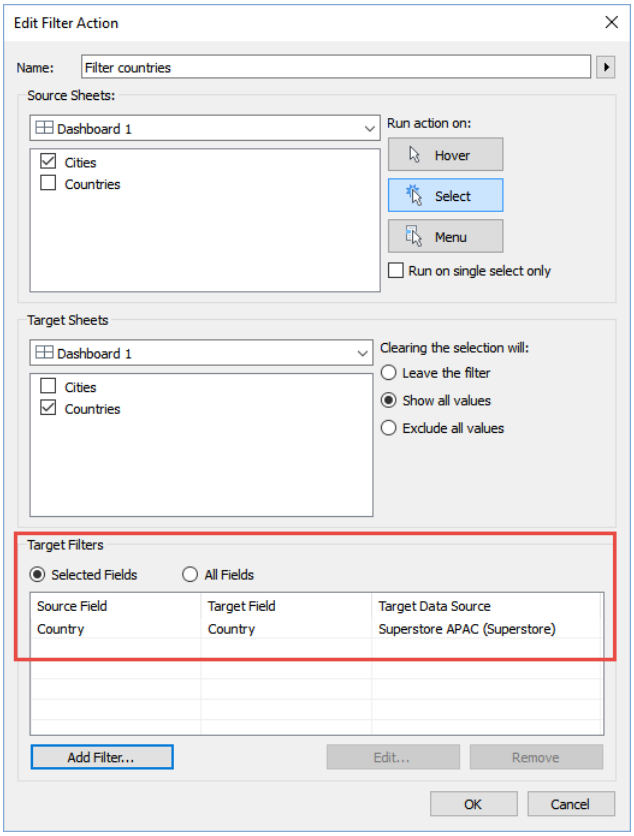
원본 워크시트에 대상 워크시트보다 더 자세한 정보가 있는 경우 동일한 방법으로 최적화할 수 있습니다. 다음과 같이 '모든 필드'를 사용하여 필터링하는 기본 동작 정의를 사용한 경우:



필터 구문이 Countries 워크시트에 대한 쿼리 결과 캐시에서 제공하지 못한 Country 와 City 를 참조하기 때문에 통합 문서에서 각 동작에 대한 쿼리를 실행합니다.

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

다음과 같이 동작을 변경하여 Country 를 기반으로 필터링하는 경우:



쿼리 결과 캐시에 이 필터가 적용되어 데이터 원본에 다시 쿼리할 필요가 없습니다. 위에서와 같이 세부 수준이 시트의 디자인에 영향을 줄 경우를 고려해야 하며 그렇지 않다면 유용한 기법일 수 있습니다.

좋은 워크시트 디자인

대시보드의 그 다음 아래 수준은 워크시트입니다. Tableau 에서 워크시트 디자인은 본질적으로 실행하는 쿼리와 관련되어 있습니다. 각 워크시트는 하나 이상의 쿼리를 생성하므로 이 수준에서 가능한 한 최적의 쿼리를 생성해야 합니다.

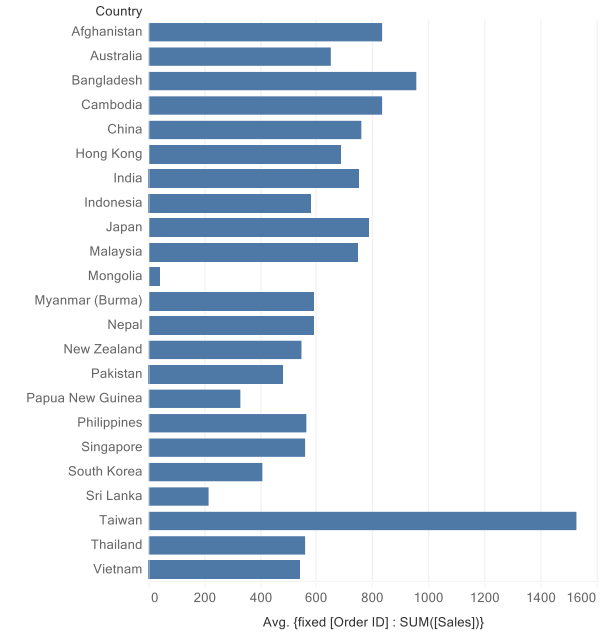
필요한 필드만 포함

세부 정보 선반을 살펴보고 도구 설명에서 필요하거나 필수 마크 세부 수준을 이끌어내는 데 필요한, 비주얼라이제이션에서 곧바로 사용되지 않는 필드를 모두 삭제하십시오. 이러한 필드를 삭제하면 데이터 원본에서 쿼리가 더 빨리 실행되고 쿼리 결과에 반환해야 할 데이터가 줄어듭니다. 이미 설명한 이 규칙에 몇 가지 예외가 있습니다. 쿼리 일괄 처리를 위해 다른 워크시트에서 유사한 쿼리를 제거한다는 것입니다. 하지만 이러한 예외는 흔하지 않으며 시트의 시각적 세부 수준을 변경하지 않는 경우에 적용됩니다.

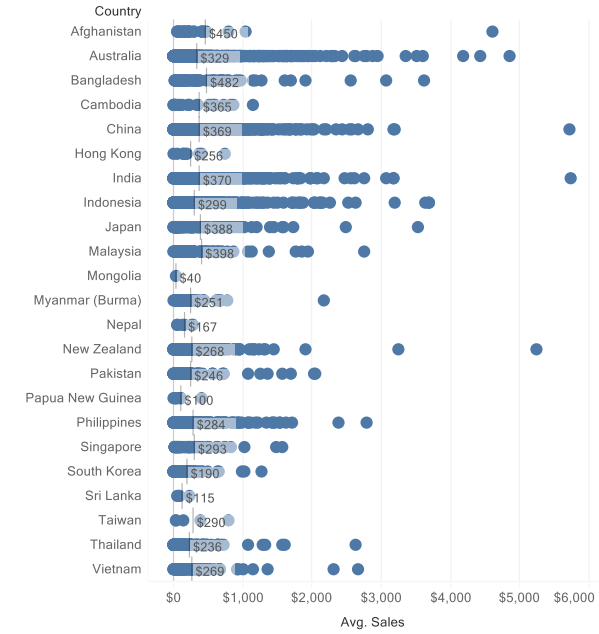
마크를 최소한으로 표시하여 질문에 답변

Tableau 에서는 대개 여러 가지 방법으로 동일한 숫자를 계산합니다. 다음 대시보드에서는 두 워크시트에서 '국가당 평균 주문 크기는?'이라는 질문에 답변한다고 가정해 보겠습니다.

Avg Order Size 1



Avg Order Size 2

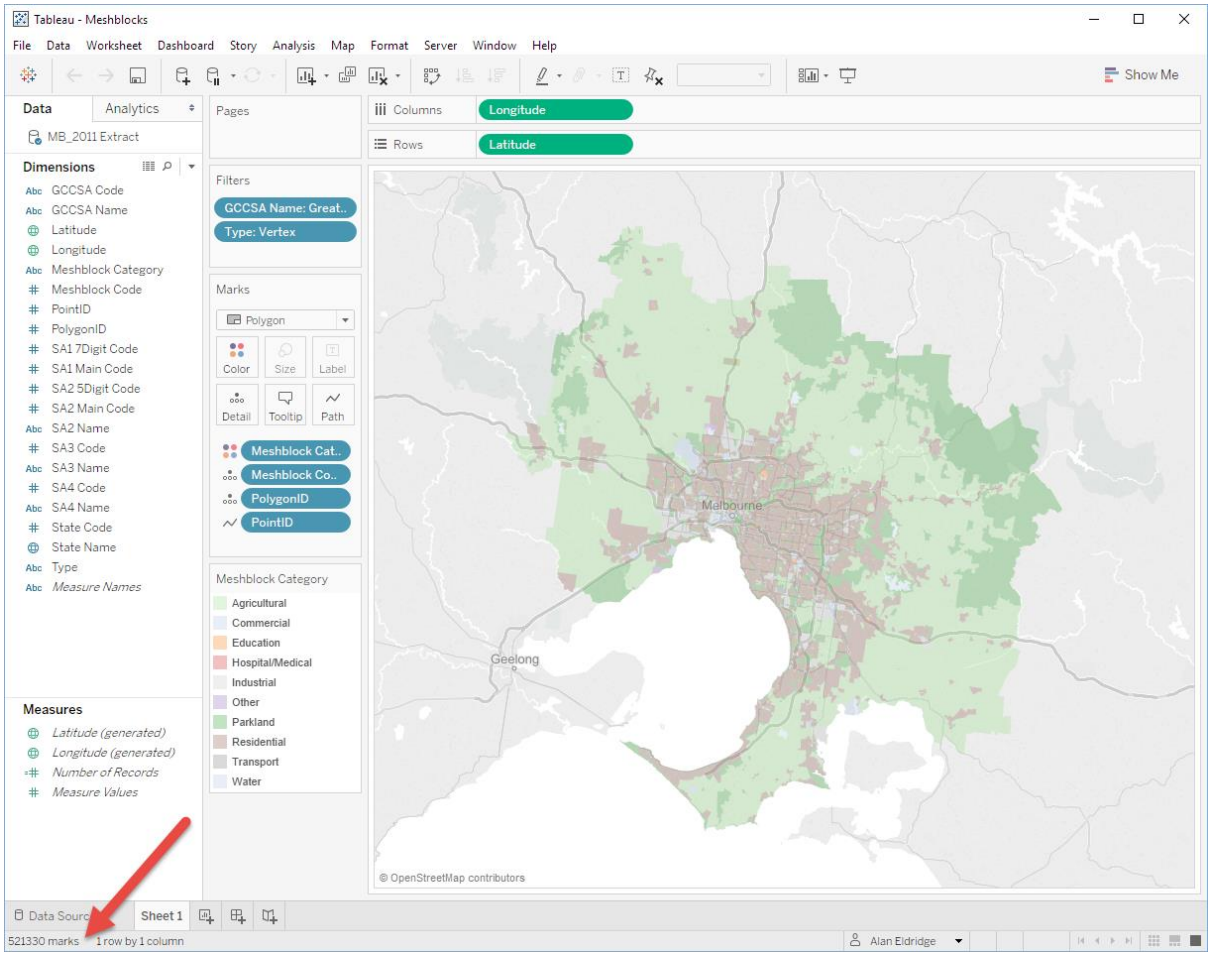


시트 1 은 단일 마크를 표시하여 국가당 평균 주문 크기를 나타내므로 데이터 원본에서 23 개의 레코드만 반환합니다. 하지만 시트 2 는 각 국가 내 주문당 마크를 표시한 다음 참조 라인으로 평균을 계산합니다. 이렇게 하려면 데이터 원본에서 5436 개의 레코드를 가져와야 합니다.

원래 질문인 국가당 평균 주문 크기에만 관심이 있다면 시트 1 이 더 나은 방법이지만 시트 2 에서는 이 질문의 답변과 주문 크기의 범위에 대한 자세한 통찰력을 제공하여 이를 통해 이상값을 확인할 수 있습니다.

너무 복잡한 비주얼라이제이션 방지

살펴보아야 할 중요한 메트릭은 각 비주얼라이제이션에서 렌더링되는 데이터 요소 개수입니다. 요소 개수는 Tableau Desktop 창의 상태 표시줄에서 쉽게 찾을 수 있습니다.

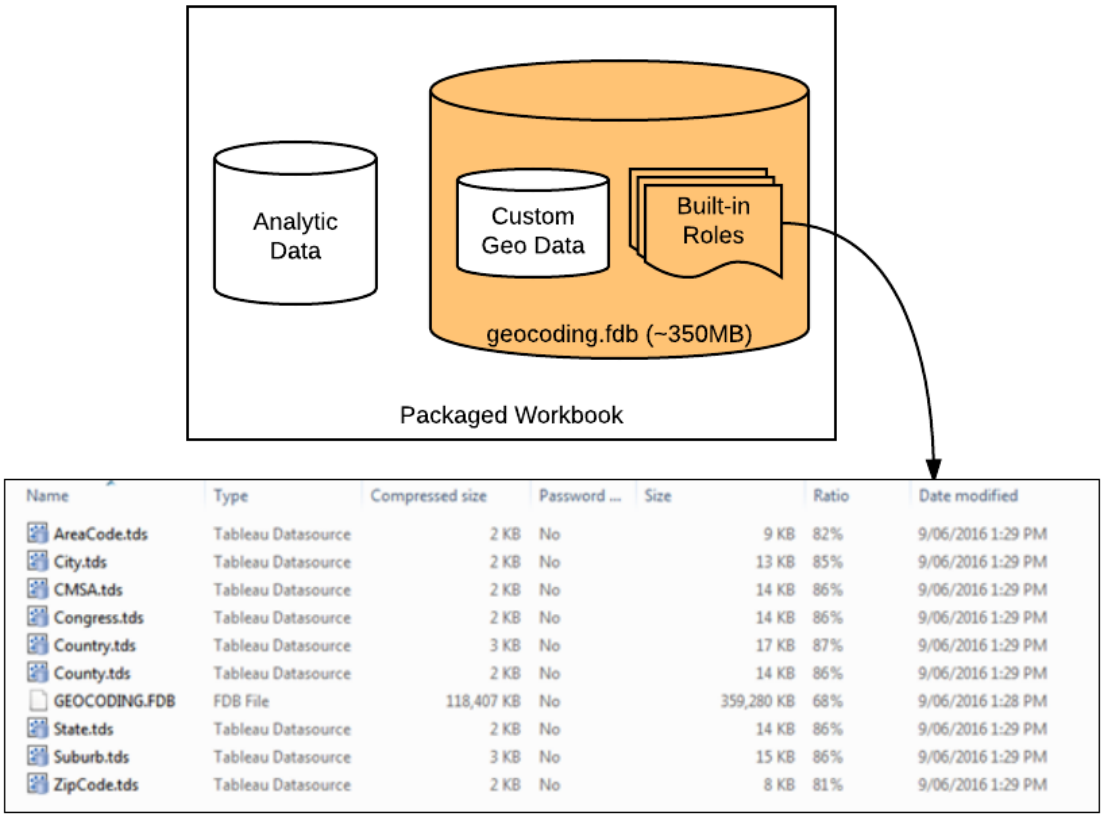


'너무 많은 마크'를 정의할 만한 엄밀한 규칙은 없지만 마크가 많을수록 렌더링하는 데 더 많은 CPU 와 RAM 이 필요하니 유의해야 합니다. 대형 크로스탭과 분산형 차트 또는 사용자 지정 다각형이 포함된 맵을 조심하십시오.

맵

사용자 지정 지오코딩

사용자 지정 지오코딩 역할을 가져오면 지오코딩 데이터베이스에 쓰여집니다. 이 Firebird DB 파일은 기본적으로 C:\Program Files\Tableau\Tableau 10.0\Local\data\geocoding.fdb 에 저장됩니다. 통합 문서에서 역할을 사용하고 패키지 통합 문서로 저장하면 전체 데이터베이스 파일이 TWBX 파일로 압축됩니다. 모두 350MB 가량이 압축됩니다.

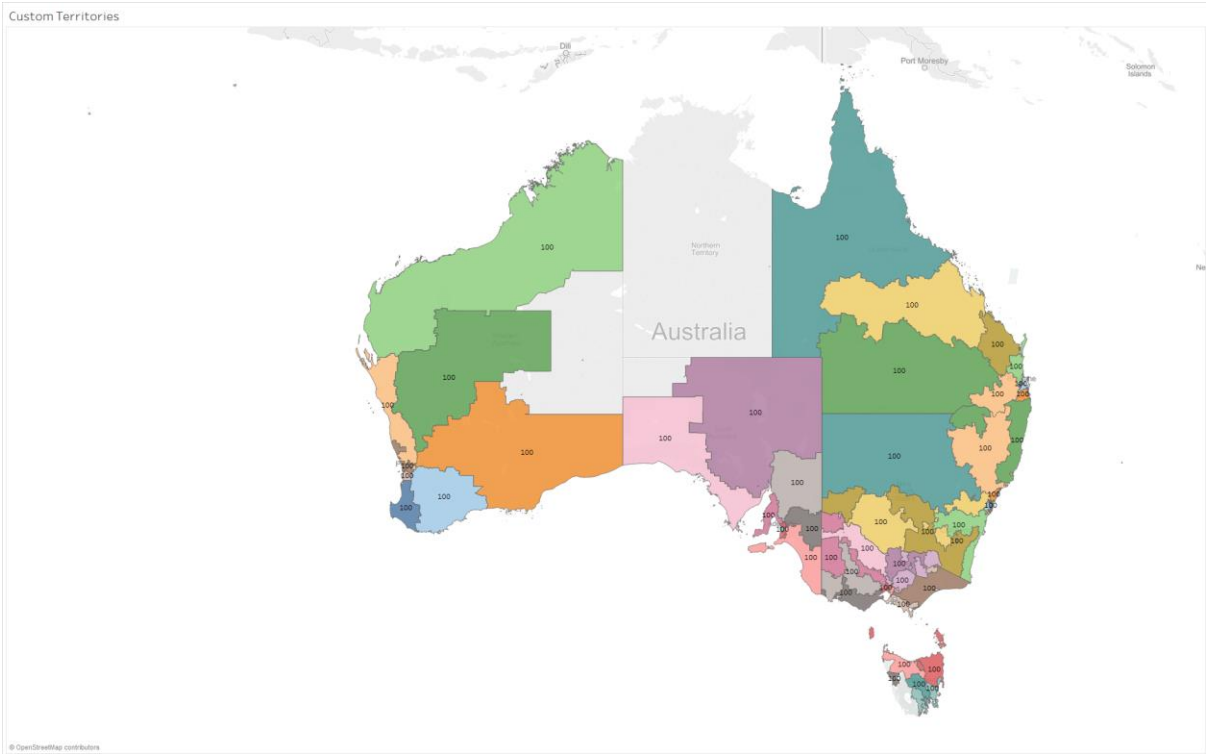


Name	Type	Compressed size	Password ...	Size	Ratio	Date modified
AreaCode.tds	Tableau Datasource	2 KB	No	9 KB	82%	9/06/2016 1:29 PM
City.tds	Tableau Datasource	2 KB	No	13 KB	85%	9/06/2016 1:29 PM
CMSA.tds	Tableau Datasource	2 KB	No	14 KB	86%	9/06/2016 1:29 PM
Congress.tds	Tableau Datasource	2 KB	No	14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

이에 따라 생성되는 TWBX 파일은 a) 지오코딩 페이로드가 약 110MB 로 압축되어 매우 크고 b) 훨씬 큰 데이터 집합에서 초기 압축 해제가 수행되어야 하므로 여는 속도가 매우 느립니다. 보다 효율적인 방법은 사용자 지정 지오코딩 역할로 데이터를 가져오지 않는 것이지만 통합 문서 내에서 혼합 기능을 사용하여 분석 데이터와 지리공간 데이터를 결합하는 것이 좋습니다. 이 방법을 사용하면 geocoding.fdb 파일이 삽입되지 않으며 TWBX 에는 분석 및 사용자 지정 지오코딩 데이터만 포함됩니다.

사용자 지정 지역

Tableau 10 의 새 기능인 사용자 지정 지역은 사용자가 내부 지오코딩 데이터베이스의 영역을 결합하여 집계된 지역을 만드는 것입니다.



많은 하위 수준 지역을 바탕으로 하는 사용자 지정 지역의 초기 렌더링은 속도가 매우 느릴 수 있으므로 이 기능은 주의하여 사용하십시오. 하지만 완료된 뒤에는 사용자 지정 지역이 캐시되어 성능이 좋아질 수 있습니다.

채워진 맵 및 지점 맵

채워진 맵 마크(맵에서 사용되거나 일부 다른 차트 유형의 마크)는 클라이언트 쪽 렌더링을 사용하는 경우 모양 다각형 데이터를 보내야 하므로 꽤 복잡한 작업이며 많은 리소스가 소요됩니다. 렌더링 성능이 느려지는 경우 대신 기호 맵 사용을 고려해 보십시오.

다각형 마크

다각형 마크를 사용하는 모든 비주얼라이제이션은 Tableau Server 가 서버 쪽 렌더링을 수행하도록 하여 최종 사용자의 환경에 영향을 줄 수 있습니다. 따라서 되도록 사용을 자제하는 것이 좋습니다.

기타 요소

대형 크로스탭

본 문서의 이전 버전에서는 대형 크로스탭이 매우 느리게 렌더링되므로 사용하지 않도록 권장했습니다. 이 비주얼라이제이션 유형의 기본 메커니즘이 최근 출시에서 향상되어 이제 크로스탭이 다른 작은 다중 차트 유형만큼 빠르게 렌더링됩니다. 하지만 초기 데이터 원본에서 읽어야 할 데이터가 많고 분석적으로 유용하지 않으므로 대형 크로스탭 사용은 신중하게 생각하는 것이 좋습니다.

도구 설명

기본적으로 도구 설명 선반에 차원을 배치하면 ATTR() 속성 함수를 사용하여 집계가 이루어집니다. 함수를 사용하려면 초기 데이터 원본에서 두 개의 집계, 즉 MIN() 및 MAX() 함수가 수행되고 두 결과가 결과 집합에 다시 전달되어야 합니다. 자세한 내용은 문서 후반의 *ATTR() 사용* 섹션을 참조하십시오.

다중 차원 값이 발생해도 괜찮다면 기본 ATTR() 대신 집계를 하나만 사용하는 것이 더 효율적입니다. MIN() 또는 MAX() 중에서 선택하십시오. 둘 중 어느 것을 사용해도 상관없지만 하나를 선택하면 계속 사용해야 캐시 적중률이 극대화됩니다.

또는 비주얼라이제이션의 시각적 세부 수준에 영향을 주지 않는다고 확신하는 경우 도구 설명 선반 대신 세부 수준 선반에 차원을 배치할 수 있습니다. 이렇게 하면 차원 필드가 곧바로 쿼리의 SELECT 및 GROUP BY 구문에서 사용됩니다. 결과는 데이터 플랫폼의 성능에 따라 다를 수 있으므로 단일 집계보다 성능이 나은지 테스트하는 것이 좋습니다.

범례

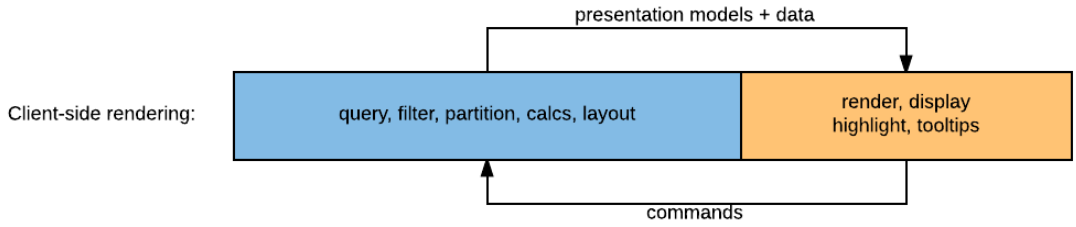
범례는 대개 도메인이 쿼리 결과 캐시에서 수행되기 때문에 성능 문제의 원인이 아닙니다. 하지만 데이터가 클라이언트 브라우저에 전송되어야 하기 때문에 열거된 도메인이 크면 렌더링 오버헤드가 가중될 수 있습니다. 따라서 일반적으로 범례가 유용하지 않는 경우라면 삭제하십시오.

페이지 선반

일부 사용자는 페이지 선반이 필터 선반과 동일한 방식으로 작동하여 데이터 원본에서 반환되는 레코드 수를 줄인다고 생각합니다. 하지만 그렇지 않습니다. 워크시트에 대한 쿼리는 전체 페이지의 모든 마크에 대한 레코드를 반환합니다. 높은 수준의 카디널리티(예: 많은 고유 값)가 포함된 페이지 차원이 있는 경우 워크시트 쿼리 크기를 현저하게 증가시켜서 성능에 영향을 줄 수 있습니다. 따라서 이 기능은 되도록 사용을 자제하는 것이 좋습니다.

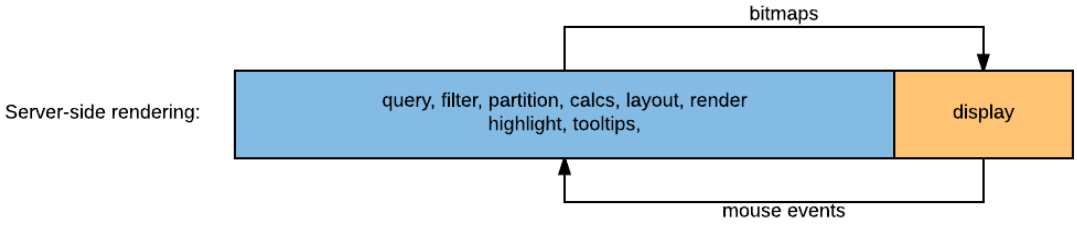
클라이언트 쪽 렌더링 및 서버 쪽 렌더링

뷰의 마크 및 데이터는 클라이언트 웹 브라우저에 표시되기 전에 검색되고 해석되고 렌더링됩니다. Tableau Server 는 이 프로세스를 클라이언트 웹 브라우저 또는 서버에서 수행할 수 있습니다. 서버에서 렌더링 및 상호작용을 처리할 경우 네트워크 데이터 전송 및 왕복으로 인한 지연이 발생할 수 있으므로 클라이언트 쪽 렌더링이 기본 모드로 설정됩니다. 클라이언트 쪽 렌더링을 활용하면 상호 작용에 대한 해석 및 렌더링 작업이 브라우저에서 바로 수행되므로 많은 뷰 상호작용이 더욱 신속하게 처리됩니다.



(파란색 - 서버에서 완료; 주황색 - 클라이언트 브라우저에서 완료)

단, 일부 뷰의 경우 컴퓨터 성능이 더 뛰어난 서버에서 렌더링하는 것이 더 효율적입니다. 서버 쪽 렌더링은 이미지 파일에 필요한 대역폭이 이미지를 만드는 데 사용되는 데이터의 대역폭보다 훨씬 적을 정도로 복잡한 뷰에 더 적합합니다. 또한 태블릿의 경우 대개 PC 보다 성능이 떨어지므로 복잡한 뷰를 PC 만큼 잘 처리하지 못합니다. 하지만 도구 설명과 하이라이트 등 간단한 상호 작용을 서버에서 렌더링하면 서버 왕복이 필요하여 속도가 느려질 수 있다는 단점이 있습니다.



(파란색 - 서버에서 완료; 주황색 - 클라이언트 브라우저에서 완료)

Tableau Server 는 웹 브라우저가 아닌 서버에서 뷰를 렌더링하기 위한 트리거로 복잡성 임계값을 사용하여 이러한 상황을 자동으로 처리할 수 있도록 구성됩니다. 임계값은 PC 와 휴대기기에 따라 다르므로 PC 웹 브라우저에서 열린 뷰가 클라이언트에서 렌더링되는데 태블릿 웹 브라우저에서 열린 동일한 뷰가 서버에서 렌더링되는 경우도 있습니다. 필터링도 렌더링 동작을 변경할 수 있습니다. 즉, 통합 문서가 처음에 서버 쪽 렌더링을 사용하여 열린 다음 필터가 적용될 때 클라이언트 쪽 렌더링으로 변경될 수 있습니다. 또한 비주얼라이제이션이 다각형 마크나 페이지 선반을 사용하는 경우 클라이언트 쪽 렌더링 기준을 충족한다고 해도 서버 쪽 렌더링만 사용하기 때문에 이 기능은 주의하여 사용해야 합니다.

관리자는 PC 및 태블릿에서 모두 이 설정을 테스트하고 미세 조정할 수 있습니다. 자세한 내용을 확인하려면 다음 링크를 참조하십시오.

https://onlinehelp.tableau.com/current/server/ko-kr/browser_rendering.htm

효율적인 필터

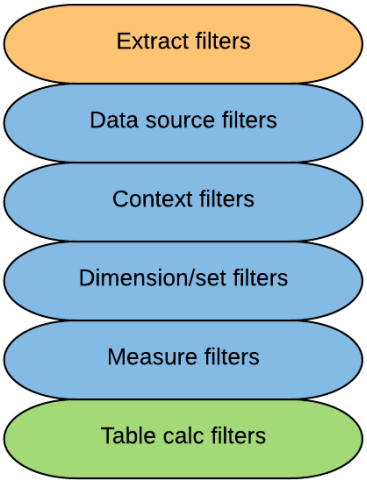
Tableau 의 필터링은 매우 강력하면서도 다양하게 표현할 수 있는 도구입니다. 하지만 필터를 효율적으로 사용하지 않을 경우 통합 문서와 대시보드의 성능이 저하될 수 있습니다. 다음 섹션은 필터를 사용할 때 참조할 수 있는 다양한 우수 사례를 설명하고 있습니다.

참고 – 데이터 원본에 색인이 있는지 및 유지 관리되는지 여부에 따라 필터의 효율성에 큰 영향을 미칩니다. 자세한 내용은 색인 섹션을 참조하십시오.

필터 유형

필터 우선 순위

Tableau 에서 필터는 다음 순서로 적용됩니다.



추출 필터

이 필터는 데이터 추출을 사용할 때에만 적용할 수 있지만 이 경우 다른 모든 필터에 앞서 먼저 논리적으로 적용됩니다. 초기 데이터 원본에서 가져오는 데이터를 제한하며 차원이나 측정값 필터 중 하나일 수 있습니다. 또한 원본 데이터 플랫폼에 따라 TOP 또는 SAMPLE 을 수행하여 반환되는 레코드 수를 줄일 수 있습니다.

데이터 원본 필터

데이터 원본 필터는 라이브 연결에서 사용할 수 있는 최고 수준의 필터입니다. 데이터 원본 필터와 컨텍스트 필터의 주요 차이점은 데이터 원본 필터의 경우 전체 데이터 원본으로 범위가 제한된 반면 컨텍스트 필터의 경우 각 워크시트에 설정된다는 점입니다. 즉, 게시된 데이터 원본에 사용되는 경우 데이터 원본 필터를 적용할 수 있는 반면 컨텍스트 필터는 워크시트 수준에 적용됩니다.

데이터 원본 필터는 데이터 원본에 제한을 두어 최종 사용자가 실수로 대규모 쿼리를 실행하지 못하도록 방지하는 효과적인 방법입니다. 예를 들어 트랜잭션 테이블의 쿼리를 지난 6 개월로만 제한하는 데이터 원본 필터를 지정할 수 있습니다.

컨텍스트 필터

기본적으로 Tableau 에서 설정한 모든 필터는 독립적으로 수행됩니다. 즉, 각 필터는 다른 필터에 관계없이 데이터 원본의 모든 행에 액세스합니다. 하지만 그 외 다른 필터는 컨텍스트 필터를 통해 전달되는 데이터만 처리하기 때문에 컨텍스트 필터를 지정하면 사용자가 정의하는 다른 모든 필터를 종속시킬 수 있습니다.

컨텍스트 필터는 정확한 답(예: 필터링된 TopN)을 가져와야 할 때 사용해야 합니다. 예를 들어 SUM(Sales)별 상위 10 개 제품을 지역으로 필터링하여 표시하는 뷰를 가정해 보겠습니다. 컨텍스트 필터가 없다면 다음과 같은 쿼리가 실행됩니다.

```
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]
```

그 결과 전세계 상위 10 개 제품에 대한 오세아니아의 기여도가 반환됩니다. 실제로 원하는 결과가 오세아니아 지역 내 상위 10 개 제품이었다면 컨텍스트에 지역 필터를 추가하여 다음 쿼리를 실행합니다.

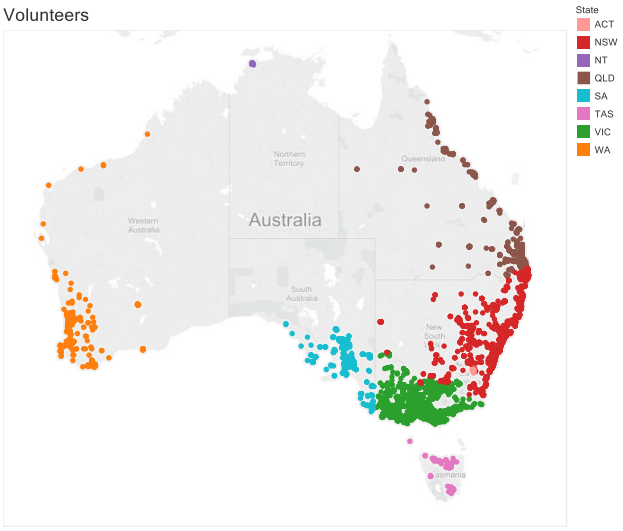
```
SELECT TOP 10 [Superstore APAC].[Product Name] AS [Product Name],
   SUM([Superstore APAC].[Sales]) AS [sum:Sales:ok],
   SUM([Superstore APAC].[Sales]) AS [$__alias__0]
FROM [dbo].[Superstore APAC] [Superstore APAC]
WHERE ([Superstore APAC].[Region] = 'Oceania')
GROUP BY [Superstore APAC].[Product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

지금까지 컨텍스트 필터는 데이터 원본에서 임시 테이블로 구현되었습니다. 하지만 이제는 거의 대부분의 경우 컨텍스트 필터가 위에 표시된 것처럼 데이터 원본 쿼리의 일부로 구현되거나 데이터 엔진에서 로컬로 처리됩니다.

더 이상 컨텍스트 필터를 쿼리 성능 향상을 위한 메커니즘으로 사용해서는 안됩니다.

범주별 차원 필터링

오스트레일리아 맵에 우편번호별로 마크를 표시한 아래 비주얼라이제이션을 살펴보십시오.



웨스턴 오스트레일리아의 우편번호(주황색 점)만 표시하도록 맵을 필터링하는 방법에는 몇 가지가 있습니다.

- 웨스턴 오스트레일리아 안에 있는 모든 마크를 선택한 다음 선택 항목만 유지
- 웨스턴 오스트레일리아 밖에 있는 모든 마크를 선택한 다음 선택 항목을 제외
- 주 차원과 같은 다른 속성만 유지
- 범위로 필터링(이 예에서는 우편번호 값 또는 위도/경도 값)

차원형

첫 번째 옵션과 두 번째 옵션을 사용하는 경우 일부만 유지하거나 제외하는 옵션은 성능이 떨어지는 것을 알 수 있습니다. 실제로 두 옵션은 대개 필터링을 거치지 않은 데이터 집합을 사용할 때보다 속도가 느립니다. 그 이유는 두 옵션 모두 DBMS 를 통해 복잡한 WHERE IN 구문을 사용하거나 값이 많은 경우에 선택 값으로 채워지는 임시 테이블을 만들고, 이 테이블과 기본 테이블에 INNER JOIN 을 사용하여 필터링되는 우편번호 값의 차원형 목록으로 표현되기

때문입니다. 마크 집합이 큰 경우 이 방법을 사용하면 쿼리를 수행하는 데 많은 오버헤드가 걸릴 수 있습니다.

이 예의 세 번째 옵션은 결과 필터(`WHERE STATE="Western Australia"`)가 매우 간단하고 데이터베이스에서 효율적으로 처리될 수 있으므로 신속하게 작동할 수 있습니다. 하지만 이 접근 방식은 필터를 표현하는 데 필요한 차원 멤버의 수가 늘어날수록 효율성이 떨어져 결국에는 묶음 옵션 및 일부 항목만 유지하는 옵션 수준으로 성능이 떨어지게 됩니다.

값 범위

값 범위 필터를 사용하면 데이터베이스에서 간단한 필터 구문(`WHERE POSTCODE >= 6000 AND POSTCODE <= 7000` 또는 `WHERE LONGITUDE < 129`)을 수행하여 실행 속도가 빨라집니다. 또한 이 접근 방식은 기준 차원의 필터와 달리 차원의 카디널리티를 높여도 더 복잡해지지 않습니다.

여기서 핵심은 값 범위 필터는 크기가 크고 항목이 구분된 차원형 값의 목록보다 신속하게 수행될 수 있으며 크기가 큰 마크 집합을 다룰 때에는 가능한 일부 항목만 유지하는 옵션 또는 제외 옵션 대신 사용해야 한다는 점입니다.

분할 필터

분할 필터는 비주얼라이제이션에서 사용되지 않는 차원의 필터입니다(예: 비주얼라이제이션 세부 수준에 속하지 않음). 예를 들어 비주얼라이제이션에서 국가별 총 판매량을 표시하지만 지역별로 필터링되는 경우 다음과 같은 쿼리가 실행됩니다.

```
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]
```

이 필터는 집계 결과를 분할하는 경우 더 복잡해집니다. 예를 들어 위의 비주얼라이제이션을 지역별로 필터링하지 않고 가장 수익성이 좋은 제품 상위 10 개의 판매량을 표시하려면 Tableau 가 제품 수준에서 가장 수익성이 좋은 제품 상위 10 개를 분리하는 쿼리, 국가 수준에서 첫 번째 쿼리의 결과로 제한하는 다른 쿼리 등 2 개의 쿼리를 실행해야 합니다.

```
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]
```

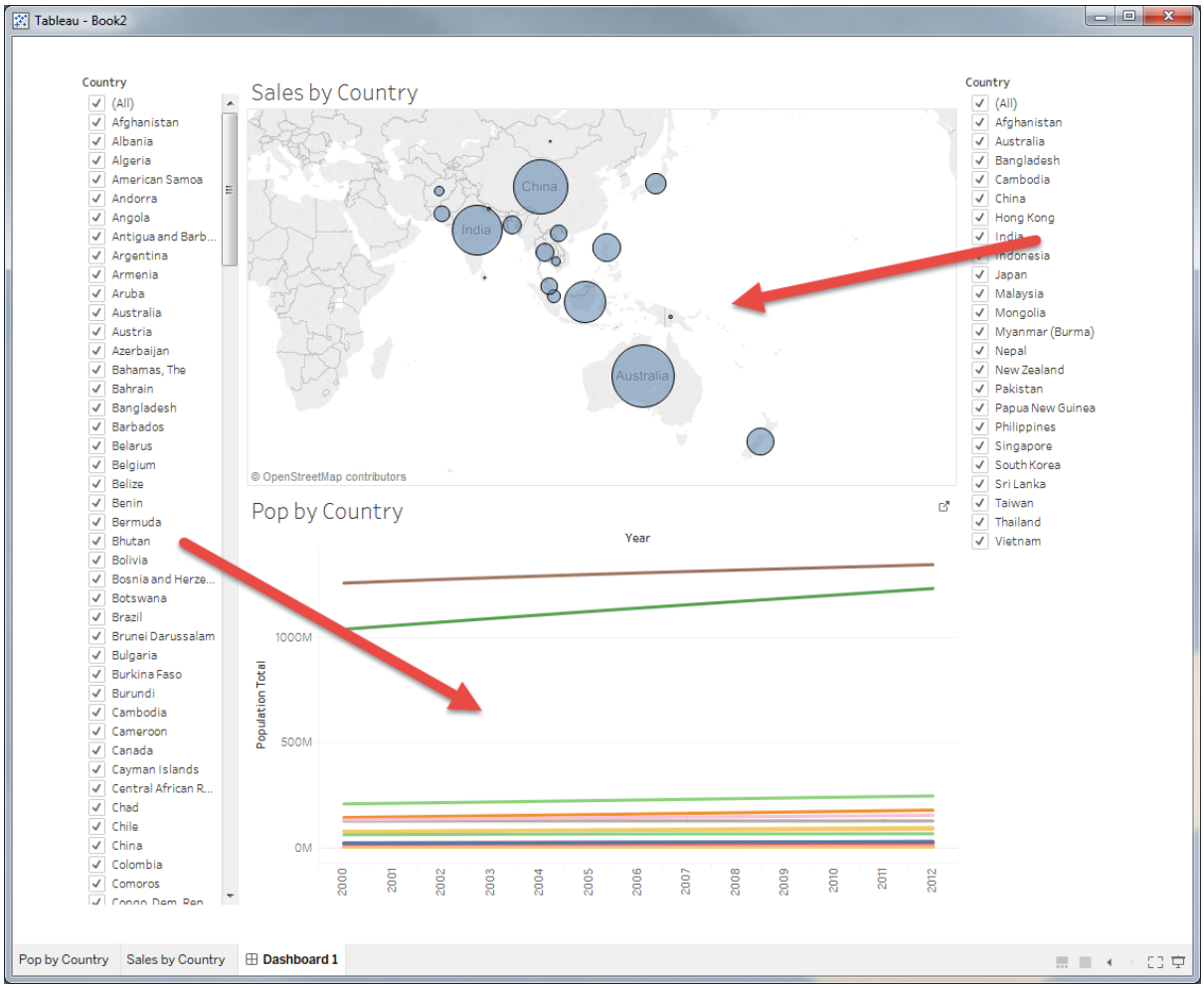
분할 필터는 수행하는 데 오버헤드가 많이 걸릴 수 있으므로 주의하여 사용해야 합니다. 또한 차원은 쿼리 결과 캐시에 포함되지 않기 때문에 분할 필터의 브라우저 내 필터링을 빠르게 수행할 수 없습니다(클라이언트 쪽 렌더링 및 서버 쪽 렌더링에 대해서는 이전 섹션 참조).

교차 데이터 원본 필터

교차 데이터 원본 필터링은 Tableau 10 의 새로운 기능입니다. 이 필터링을 사용하면 하나 이상의 필드가 공통으로 포함된 여러 데이터 원본에 필터를 적용할 수 있습니다. 관계는 혼합과 동일한 방법으로 이름/유형 일치를 토대로 하여 자동으로 또는 데이터 메뉴에서 사용자 지정 관계를 통해 수동으로 정의됩니다.

교차 데이터베이스 필터는 대시보드의 퀵 필터만큼 빠른 성능을 가집니다. 필터를 변경하면 여러 영역의 업데이트가 발생하여 여러 쿼리가 필요할 수 있습니다. 따라서 현명하게 사용해야 하며 사용자가 여러 번 변경할 것으로 예상되는 경우 '적용' 버튼을 표시하여 선택이 완료된 경우에만 쿼리를 실행하도록 하는 것이 좋습니다.

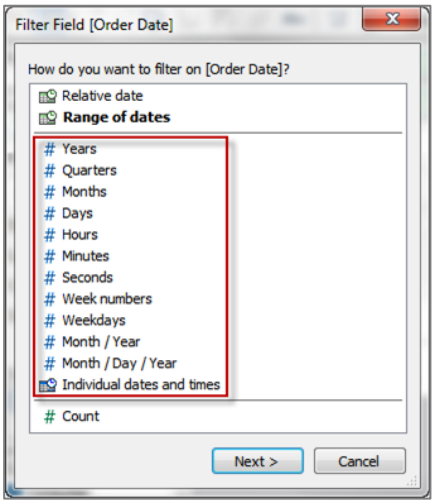
또한 필터의 도메인은 '기본' 데이터 원본(예: 필터가 생성된 시트에서 처음 사용된 데이터 원본)에서 가져옵니다. 관련 필드에 다른 데이터 원본의 다른 도메인이 있는 경우 아래 표시된 것처럼 같은 필터가 다른 값을 표시할 수 있으므로 어떤 도메인을 사용하는지 유의해야 합니다.

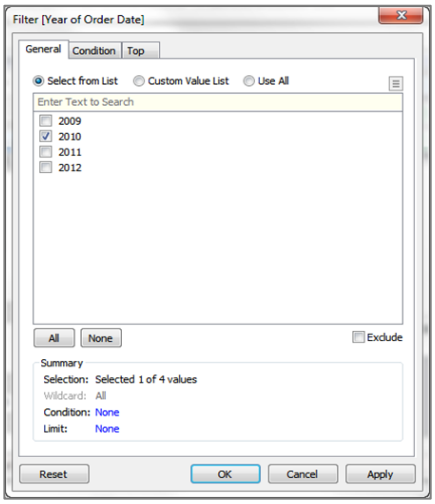


날짜 필터링: 차원형, 범위형, 기준형

날짜 필드는 대체로 Tableau 에서 표준 범주형 데이터와 다르게 처리하는 특별한 유형의 차원입니다. 특히 날짜 필터를 만드는 경우 이러한 차이가 두드러집니다. 날짜 필터는 자주 사용되며 세 가지 범주로 나뉩니다. 기준 날짜 필터는 특정 날짜를 기준으로 날짜 범위를 표시하고, 날짜 범위 필터는 정의된 차원형 날짜 범위를 표시하고, 차원형 날짜 필터는 목록에서 선택한 개별 날짜를 표시합니다. 위 섹션에서 설명한 바와 같이 어떤 방법을 사용하느냐에 따라 결과 쿼리의 효율성은 크게 달라질 수 있습니다.

차원형





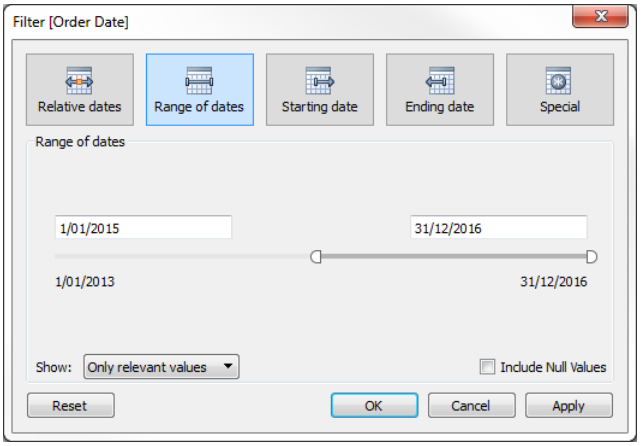
상황에 따라서는 특정 개별 날짜나 전체 날짜 수준을 포함하도록 데이터를 필터링해야 하는 경우가 있습니다. 이러한 유형의 필터는 범위가 아닌 차원 값을 정의한다는 점에서 차원형 날짜 필터로 불립니다. 이 필터 유형을 사용하면 날짜 식이 데이터베이스에 동적 계산으로 전달됩니다.

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

대개 쿼리 최적화 도구는 DATEPART 계산을 지능적으로 수행하지만 일부 시나리오의 경우 차원형 날짜 필터를 사용해도 쿼리가 효율적으로 실행되지 않을 수 있습니다. 예를 들어, 날짜 파티션 키에 차원형 날짜 필터를 적용하여 분할한 테이블에 쿼리를 보낸다고 가정해 보겠습니다. 이 테이블은 DATEPART 값을 기준으로 분할되지 않았으므로 쿼리를 실행하면 일부 데이터베이스가 범위에서 누락되며 불필요한 경우에도 조건을 충족하는 레코드를 찾기 위해 모든 파티션에 걸쳐 계산을 수행하게 됩니다. 이 경우 '날짜 범위' 필터 또는 고정 날짜가 지정된 기준 날짜 필터를 사용하면 성능이 훨씬 향상될 수 있습니다.

이러한 유형의 필터를 사용할 때 성능을 최적화하는 방법 중 하나는 바로 데이터 추출을 사용하여 계산을 구체화하는 것입니다. 먼저, DATEPART 함수를 명시적으로 구현하는 계산된 필드를 만드십시오. 그런 다음 Tableau 데이터 추출을 생성하면 식에서 결정 값을 산출하므로 계산된 필드가 추출의 저장된 값으로 구체화됩니다. 계산된 필드를 필터링하면 쿼리 시간 동안 값을 계산할 필요 없이 손쉽게 검색할 수 있으므로 동적 식을 필터링하는 것보다 더 신속하게 작업을 처리할 수 있습니다.

날짜 범위

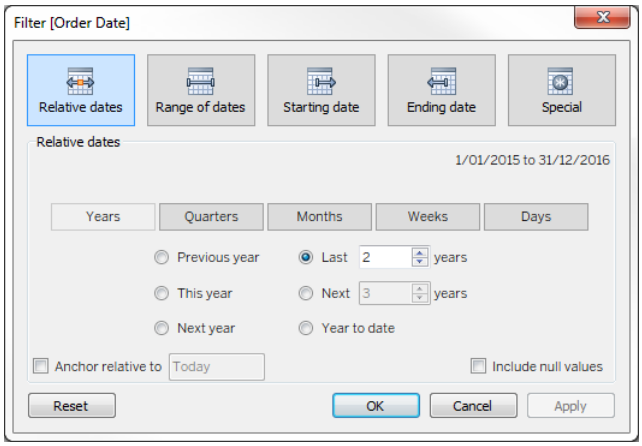


이 유형의 필터는 인접한 날짜의 범위를 지정할 때 사용됩니다. 이 필터를 사용하면 다음 쿼리 구조가 데이터베이스에 전달됩니다.

```
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 ()
```

이와 같은 유형의 WHERE 구문은 쿼리 최적화 도구에서 사용할 경우 매우 효율적이며 실행 계획에서 색인과 파티션을 최대한 활용할 수 있도록 해줍니다. 차원형 날짜 필터를 추가했을 때 쿼리 시간이 느리면 범위 날짜 필터로 대체한 다음 차이가 있는지 확인해 보십시오.

기준형



기준 날짜 필터를 사용하면 뷰를 여는 날짜와 시간에 따라 업데이트되는 날짜 범위를 정의할 수 있습니다. 예를 들어 연간 누계 판매량, 최근 30 일의 모든 레코드 또는 지난 주의 버그 수정을 확인할 수 있습니다. 기준 날짜 필터는 오늘이 아니라 특정 고정 날짜를 기준으로 할 수도 있습니다.

```
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 ()
```

보다시피 결과 WHERE 절에 날짜 범위 구문이 사용되므로 이 유형의 날짜 필터도 효율성이 뛰어나다 할 수 있습니다.

위의 기준 날짜 필터를 통해서든 필터 수식에서 NOW() 또는 TODAY() 함수를 명시적으로 사용하든 현재 날짜/시간이 기준인 날짜 필터의 변경되는 특성 때문에 쿼리 캐시는 이들을 사용하는 쿼리에는 효율적이지 않습니다. Tableau 에서는 이러한 쿼리를 캐시 서버의 '일시적인 쿼리'로 표시하며 그 결과는 다른 쿼리 결과만큼 유지되지 않습니다.

필터 카드

필터 카드를 너무 많이 표시하면, 특히 차원형 목록이 많은 상황에서 '관련된 값만' 사용하도록 설정할 경우 속도가 저하됩니다. 분석 방법을 선택할 때에는 안내를 충분히 참고하시기 바라며 가급적 대시보드에 있는 동작 필터를 사용하시기 바랍니다. 많은 필터를 사용하여 상호 작용이 매우 뛰어난 뷰를 구축하려는 경우 다양한 수준과 테마로 구성된 대시보드를 여러 개 구축하는 것이 더 나은지 자문해 보십시오. 대부분의 경우에는 대시보드를 여러 개 구축하는 편이 더 좋은 방법입니다.

열거형과 비열거형

열거형 필터 카드를 사용하려면 필터 카드를 렌더링하기 전에 Tableau 에서 필드 값의 데이터 원본을 쿼리해야 합니다. 여기에는 다음이 포함됩니다.

- 다중 값 목록 – 모든 차원 멤버
- 단일 값 목록 – 모든 차원 멤버
- 압축 목록 – 모든 차원 멤버
- 슬라이더 – 모든 차원 멤버
- 측정값 필터 – *MIN* 값 및 *MAX* 값
- 범위 날짜 필터 – *MIN* 값 및 *MAX* 값

이와 달리 비열거형 필터 카드는 잠재적인 필드 값을 파악할 필요가 없습니다. 여기에는 다음이 포함됩니다.

- 사용자 지정 값 목록
- 와일드카드 일치 항목
- 기준 날짜 필터
- 탐색 기간 날짜 필터

결과적으로 비열거형 필터 카드를 사용하면 데이터 원본에서 실행해야 하는 필터 관련 쿼리의 개수를 줄일 수 있습니다. 뿐만 아니라 비열거형 필터 카드는 표시해야 하는 차원 멤버가 많은 경우에도 신속하게 렌더링됩니다.

비열거형 필터 카드를 사용하면 성능을 개선할 수 있지만 최종 사용자에게 제공되는 시각적 컨텍스트의 품질은 다소 떨어지게 된다는 점을 유념하십시오.

관련 값

열거형 필터 카드는 잠재 필드 값을 다음 세 가지 방식으로 표시하도록 설정될 수 있습니다.

- 데이터베이스의 모든 값 - 이 옵션을 선택하면 뷰의 다른 필터에 관계 없이 데이터베이스의 모든 값이 표시되며 다른 필터가 변경되는 경우에도 필터에서 데이터베이스에 쿼리를 다시 보낼 필요가 없습니다.

- 컨텍스트의 모든 값 – 이 옵션은 활성화된 컨텍스트 필터가 있는 경우에만 사용할 수 있습니다. 필터 카드는 뷰에 있는 다른 필터와 상관없이 컨텍스트의 모든 값을 표시합니다. 차원이나 측정값 필터가 변경되어도 필터가 데이터베이스를 다시 쿼리할 필요가 없지만 컨텍스트가 변경되면 다시 쿼리해야 합니다.
- 관련 값만 - 이 옵션을 선택하면 다른 필터가 적용되며 이 필터를 통과한 값만 표시됩니다. 예를 들어 Region 에 필터가 설정된 경우 State 필터에서 Eastern 주만 표시합니다. 결과적으로 필터는 다른 필터가 변경되면 데이터 원본에 쿼리를 다시 보내야 합니다.

보시다시피 '관련 값만' 설정은 관련 항목을 선택할 때 매우 유용하지만 이 설정으로 인해 대시보드와 상호작용하는 동안 실행해야 하는 쿼리의 개수가 크게 늘어날 수 있습니다. 따라서 이 설정은 꼭 필요한 경우에만 사용하는 것이 좋습니다.

필터 카드 대안

필터 카드와 유사한 분석 결과를 제공하면서도 추가 쿼리를 과도하게 실행하지 않는 대안도 있습니다. 예를 들어 사용자의 선택을 기반으로 매개 변수와 필터를 만들 수 있습니다.

- 장점:
  - 매개 변수의 경우 렌더링 전에 데이터 원본에 쿼리를 보낼 필요가 없습니다.
  - 매개 변수와 계산된 필드를 통합하면 단순한 필드 필터에서는 구현할 수 없는 복잡한 논리를 구현할 수 있습니다.
  - 매개 변수를 사용하면 다양한 데이터 원본에 걸쳐 필터링을 수행할 수 있습니다. Tableau 10 이전 버전에서는 필터가 단일 데이터 원본에서만 작동합니다. Tableau 10 이상에서는 관련 데이터 원본 전체에서 작동하도록 필터를 설정할 수 있습니다.
- 단점:
  - 매개 변수는 단일 값만 지원합니다. 따라서 사용자가 여러 값을 선택할 수 있도록 하려는 경우에는 매개 변수를 사용할 수 없습니다.
  - 매개 변수는 동적으로 실행되지 않습니다. 매개 변수를 생성할 때 값 목록이 정의되며 DBMS 값에 따라 업데이트되지 않습니다.

또 다른 대안으로는 뷰 간에 필터 동작을 활용하는 방법이 있습니다.

- 장점:
  - 필터 동작으로 다양한 값을 선택할 수 있습니다(시각적으로 묶기 또는 CTRL/SHIFT 클릭).
  - 필터 동작으로 런타임에 수행되는 값의 동적 목록을 표시할 수 있습니다.
  - 필터 동작으로 다양한 데이터 원본에 걸쳐 필터링을 수행할 수 있습니다. Tableau 10 이전 버전에서는 필터가 단일 데이터 원본에서만 작동합니다. Tableau 10 이상에서는 관련 데이터 원본 전체에서 작동하도록 필터를 설정할 수 있습니다.
- 단점:
  - 필터 동작은 필터 카드보다 설정이 복잡합니다.

- 필터 동작에서 표시하는 사용자 인터페이스가 매개 변수 또는 필터 카드에서 표시하는 사용자 인터페이스와 다릅니다. 필터 동작의 인터페이스는 일반적으로 화면 면적을 더 많이 차지합니다.
- 동작 원본 시트에서 데이터 원본에 여전히 쿼리를 보내야 합니다. 단, 필터 동작은 Tableau 프로세스 내 캐싱의 장점을 활용합니다.

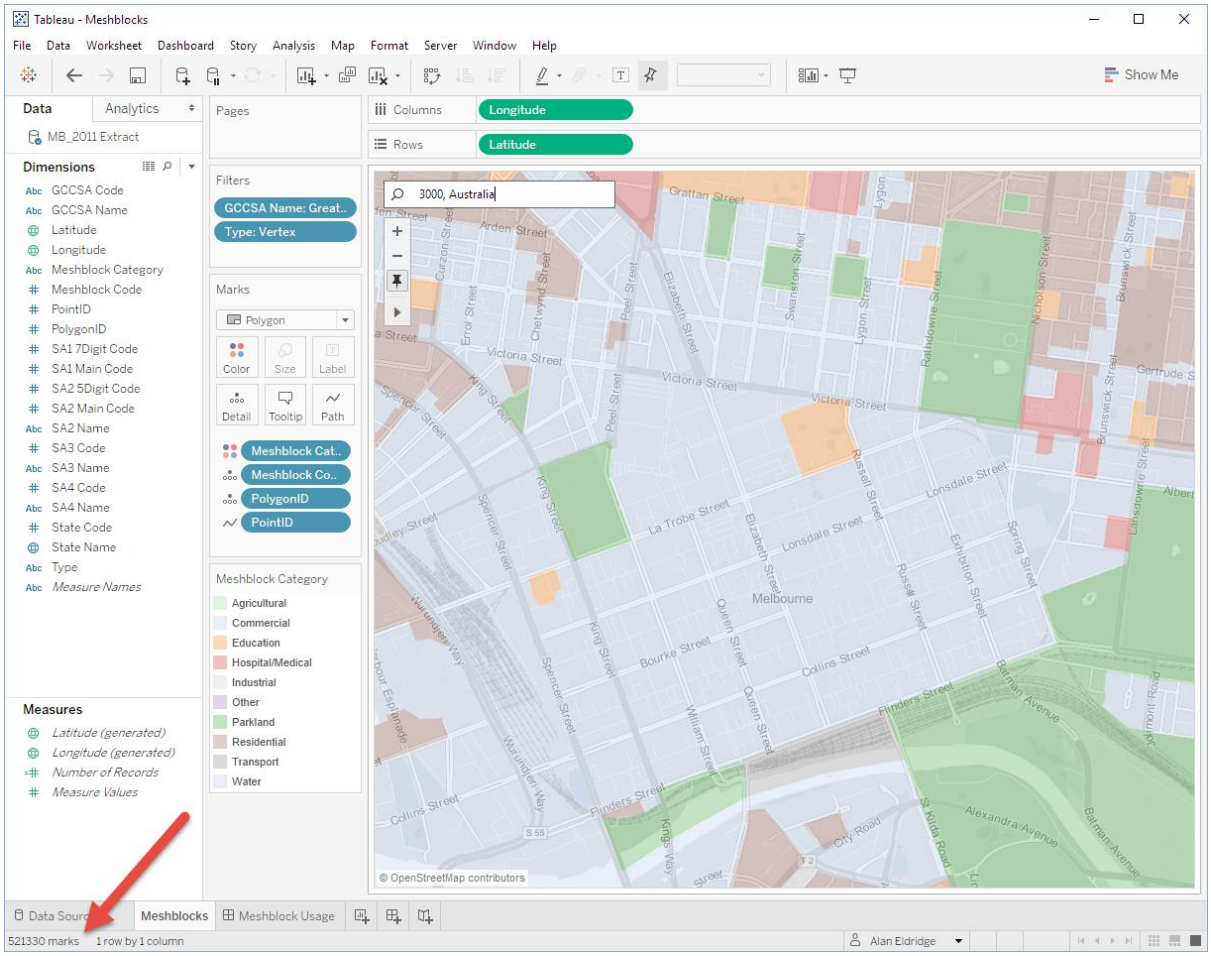
필터 카드에 많이 의존하지 않는 다른 디자인 기술에 대한 자세한 내용을 확인하려면 이전 섹션의 좋은 대시보드 디자인을 참조하십시오.

사용자 필터

'사용자 필터 만들기...' 대화 상자를 통하거나 ISMEMBEROF() 같은 내장된 사용자 함수를 사용하는 계산된 필드를 통해 통합 문서에 사용자 필터가 포함되면 사용자 세션에서 모델 캐시가 공유되지 않습니다. 이로 인해 캐시 재사용 비율이 현저하게 낮아져서 Tableau Server 의 작업이 늘어날 수 있습니다. 따라서 이 필터 유형은 주의하여 사용하시기 바랍니다.

줌 기능과 필터링

마크 개수가 많은 비주얼라이제이션을 줌인한다고 해서 보이지 않는 마크가 필터링되는 것은 아닙니다. 데이터를 나타내는 표시 영역만이 변경될 뿐입니다. 다음 이미지에서 볼 수 있는 것처럼 총 마크 수는 변경되지 않습니다.



데이터의 하위 집합만 필요한 경우 원하지 않는 데이터를 필터링하고 Tableau 의 자동 줌 기능에서 표시 영역을 설정하도록 합니다.

계산

대부분의 경우 원본 데이터는 질문에 대한 답을 이끌어내는 데 필요한 모든 필드를 제공하지 않습니다. 이 경우 계산된 필드를 사용하면 분석에 필요한 차원 및 측정값을 생성할 수 있습니다.

계산된 필드에서는 하드 코딩된 상수(예: 세율)를 정의하거나, 빼기 또는 곱하기와 같은 매우 단순한 수학 연산 작업(예: 수입 - 비용)을 수행하거나, 더 복잡한 수학 공식을 활용하거나, 논리 테스트(예: IF/THEN, CASE)를 수행하거나, 유형을 변환하는 등 다양한 작업을 수행할 수 있습니다.

계산된 필드는 한 번 정의하면 워크시트에서 동일한 데이터 원본을 사용하는 한, 모든 통합 문서에 걸쳐 사용할 수 있습니다. 또 계산된 필드는 원본 데이터의 차원과 측정값을 사용하는 방식과 동일한 방식으로 통합 문서에서 사용할 수 있습니다.

Tableau 에서 사용할 수 있는 계산 유형에는 네 가지가 있습니다.

- 행 수준 계산
- 집계 계산
- 테이블 계산
- 세부 수준 표현식

다음 플로우 차트를 잘 활용하여 최상의 방법을 선택하십시오.

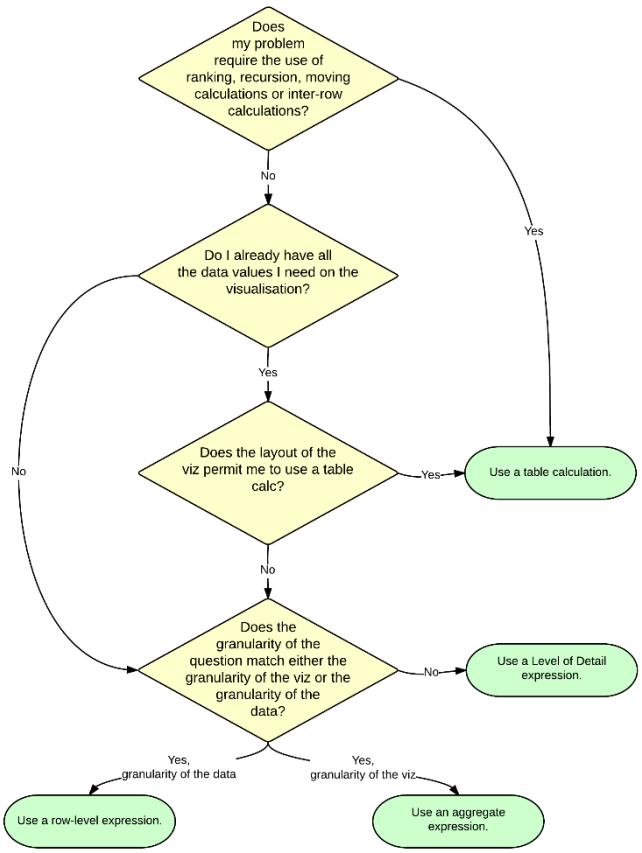


Tableau 계산 참조 라이브러리에서 복잡한 계산을 수행하는 방법에 대한 유용한 참고 자료를 확인하고 포럼에서 다양한 사용자와 일반적인 문제에 대한 해결책을 공유해 보십시오.

http://tabsoft.co/1I0SsWz

계산 유형

행 수준 및 집계 계산

행 수준 및 집계 계산은 데이터 원본에 전송되는 쿼리의 일부로 표현되므로 데이터베이스를 통해 계산됩니다. 예를 들어 각 주문 데이터 연도의 총 판매량 합계를 표시하는 비주얼라이제이션에서 데이터 원본에 다음 쿼리를 보낸다고 가정해 보겠습니다.

```
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])
```

여기서 YEAR 계산은 행 수준 계산이며 SUM(SALES) 계산은 집계 계산입니다.

일반적으로 행 수준 계산 및 집계 계산은 확장성이 매우 뛰어나며 다양한 데이터베이스 조정 기술을 활용하여 성능을 개선할 수 있습니다.

Tableau 에서 실제 DB 쿼리는 비주얼라이제이션에서 사용되는 기본 계산을 곧바로 해석한 것이 아닐 수 있습니다. 예를 들어 다음 쿼리는 비주얼라이제이션에 `SUM([Profit])/SUM([Sales])` 함수로 정의된 계산된 필드 Profit Ratio 가 포함되어 있을 때 실행됩니다

`SUM([Profit])/SUM([Sales])`.

```
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 는 실제로 계산 요소를 가져와서 클라이언트 계층에서 분할 함수를 수행합니다. 이렇게 하면 Year 수준의 `SUM([Profit])` 및 `SUM([Sales])` 함수가 캐싱되고 데이터 원본으로 돌아가지 않아도 통합 문서 내 다른 곳에서 사용될 수 있습니다.

마지막으로 쿼리 융합(여러 논리 쿼리를 하나의 실제 쿼리에 결합)으로 데이터 원본에 대한 쿼리 실행을 수정할 수 있습니다. 다른 워크시트에서 동일한 세부 수준을 공유하는 경우 이로 인해 여러 워크시트의 여러 측정값이 하나의 쿼리로 결합될 수 있습니다.

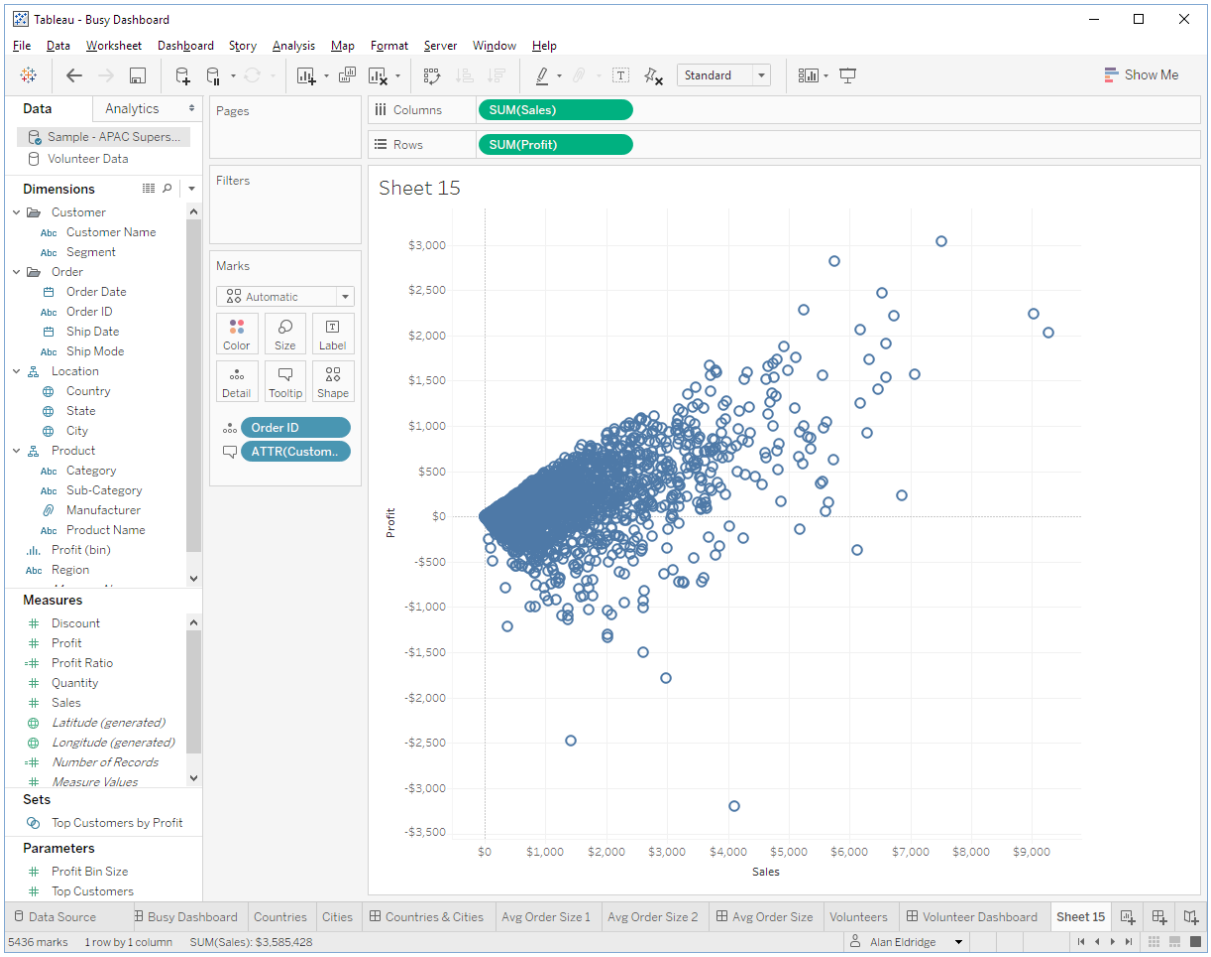
ATTR() 사용

ATTR() 함수는 주로 범주형 차원의 집계로 사용되는 강력한 함수입니다. 간단히 말해 고유한 경우 값을 반환하고 그렇지 않으면 *를 반환합니다. 기술적으로 함수는 다음 논리 테스트를 수행합니다.

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

함수를 사용하려면 초기 데이터 원본에서 두 개의 집계, 즉 MIN() 및 MAX() 함수가 수행되고 두 결과가 결과 집합에 다시 전달되어야 합니다. 다중 차원 값이 발생해도 괜찮다면 MIN() 또는 MAX() 중 하나의 집계만 사용하는 것이 더 효율적입니다. 둘 중 어느 것을 사용해도 상관없지만 하나를 선택하면 계속 사용해야 캐시 적중률이 극대화됩니다.

ATTR() 함수가 데이터 원본에 미치는 영향을 보여주기 위해 다음 비주얼라이제이션을 만들었습니다.



기본적으로 도구 설명 선반에 차원을 배치하면 기본 집계로 ATTR() 함수를 사용합니다. 함수를 사용하면 다음 쿼리가 실행됩니다.

```
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]
```

각 주문 ID 에 여러 고객이 있지 않은 경우 집계를 MIN() 함수로 변경하여 쿼리를 단순화할 수 있습니다.

```
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]
```

다음 테이블은 두 옵션의 성능 차이를 보여줍니다. 결과는 쿼리가 완료되는 데 걸린 시간(초 단위)으로 로그 파일에서 제공합니다.

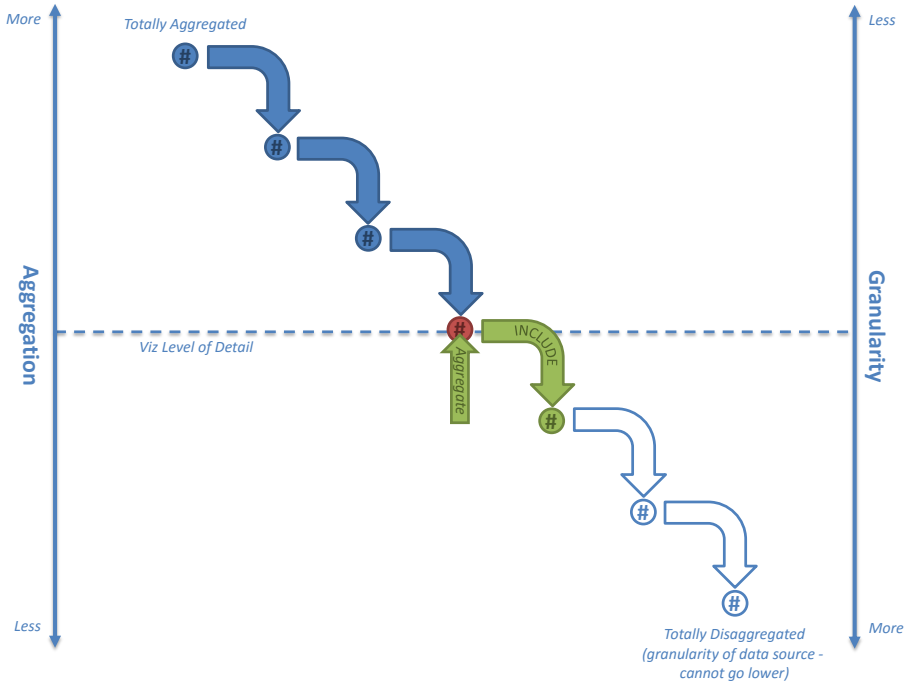
Test #	ATTR	MIN
1	2.824	1.857
2	2.62	2.09
3	2.737	2.878
4	2.804	1.977
5	2.994	1.882
Average	2.7958	2.1368
% improvement		**24%**

ATTR()에 대한 자세한 내용은 다음 InterWorks 블로그 게시물을 참조하십시오.

http://bit.ly/1YEuZhX

세부 수준 계산

세부 수준(LOD) 표현식을 사용하면 비주얼라이제이션 LOD 와 독립적으로 계산을 수행할 세부 수준을 정의할 수 있습니다.



일부 경우 LOD 표현식을 사용하여 Tableau 이전 버전에서 복잡한 방식으로 작성된 계산을 바꿀 수 있습니다.

- 테이블 계산을 사용하여 파티션 내 첫 번째 또는 마지막 기간을 찾으려고 시도했을 수 있습니다. 예를 들어 매월 첫날에 조직의 인원 수를 계산하는 경우 등이 있습니다.
- 테이블 계산, 계산된 필드 및 참조 라인을 사용하여 집계된 필드의 구간차원을 만들려고 시도했을 수도 있습니다. 예를 들어 고객 수의 평균을 구하는 작업 등의 경우가 있습니다.
- 데이터 혼합 기능을 사용하여 데이터의 최대값 날짜를 기준으로 기준 날짜 필터링을 시도했을 수도 있습니다. 예를 들어 데이터를 매주 새로 고치고 최대값 날짜에 따라 연간 누계 총계를 계산하는 경우 등이 있습니다.

테이블 계산과 달리 LOD 표현식은 초기 데이터 원본에 대한 쿼리의 일부로 생성됩니다. LOD 표현식은 중첩된 선택으로 표현되므로 DBMS 성능에 의존합니다.

```
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]
```

즉, 한 가지 방법이나 다른 방법으로 문제를 해결하는 보다 효율적인 시나리오가 될 수 있습니다. 예를 들어 테이블 계산이나 혼합이 LOD 표현식보다 성능이 좋을 수 있으며 그 반대의 경우도 마찬가지입니다. LOD 표현식 때문에 성능이 느려졌다고 의심되면 가능한 경우 테이블 계산이나 데이터 혼합으로 바꿔서 수행하여 성능이 개선되는지 확인할 수 있습니다. 또한 LOD 표현식은 조인 선별에 큰 영향을 받을 수 있으므로 LOD 표현식을 사용할 때 쿼리가 느리게 실행된다고 생각되면 해당 섹션으로 돌아가 다시 읽어보시기 바랍니다.

LOD 표현식의 메커니즘을 더 잘 파악하려면 '세부 수준 표현식의 이해' 백서를 참조하십시오.

http://www.tableau.com/ko-kr/learn/whitepapers/understanding-lod-expressions

다양한 일반적인 문제 유형의 예를 제공하는 Bethany Lyons 의 '상위 15 가지 LOD 표현식'도 참조할 수 있습니다.

http://www.tableau.com/ko-kr/about/blog/LOD-expressions

마지막으로 커뮤니티에 매우 유용한 주제를 다루는 많은 블로그 게시물이 있습니다. *data + science* 에서 잘 정리된 목록을 볼 수 있습니다.

http://bit.ly/1MpkFV5

테이블 계산

테이블 계산은 행 수준 계산 및 집계 계산과 달리 데이터베이스에서 실행하는 것이 아니라 쿼리 결과 집합에서 Tableau 가 계산합니다. 이는 Tableau 에서 더 많은 작업을 수행해야 한다는 것을 의미하지만 계산 시 사용되는 레코드 집합은 대개 원래 데이터 원본에 있는 레코드 집합보다 훨씬 작습니다.

테이블 계산의 성능에 문제가 있는 경우(대개 Tableau 로 반환되는 결과 집합의 크기가 매우 큰 경우 문제가 발생) 일부 계산을 데이터 원본 계층으로 푸시하는 것이 좋습니다. 한 가지 방법은 세부 수준 표현식을 통하는 것입니다. 또 다른 방법은 집계된 데이터 추출을 활용하는 것입니다. 여러 영업점에 걸쳐 일별 총 판매량의 주간 평균을 구하는 경우를 가정해 보겠습니다. 테이블 계산에서 이 값을 구하는 방법은 다음과 같습니다.

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

하지만 날짜/영업점의 수가 많은 경우에는 이 계산을 수행하는 데 많은 시간이 걸릴 수 있습니다. 이 경우 날짜 차원을 날짜 수준으로 롤업하는 집계 추출을 생성하여 SUM([Sales])을 데이터 계층으로 푸시하십시오. 이렇게 하면 추출에서 일별 총 판매량이 구체화되므로 AVG([Sales])만 사용해도 원하는 값을 계산할 수 있습니다.

일부 사용 사례에서는 집계 요소의 값이 파티션에 따라 변하지 않음을 알 수 있습니다. 이 경우 MIN() 또는 MAX() 함수가 더 빠르게 수행되므로 AVG() 또는 ATTR() 대신 집계 함수로 사용하십시오. 함수를 선택한 뒤에는 계속 사용해야 캐시 적중률이 높아집니다.

외부 계산

Tableau 에는 복잡한 논리를 외부 엔진에 전달하여 계산할 수 있도록 하는 메커니즘이 있습니다. 여기에는 Rserve 연결을 통한 R 의 호출 또는 Dato 서버 연결을 통한 Python 의 호출이 포함됩니다.

Tableau 내에서 이러한 작업은 테이블 계산과 비슷하여 전체 파티션에서 계산됩니다. 즉, 단일 비주얼라이제이션에서 여러 번 호출되어 성능에 영향을 줄 수 있습니다. 또한 어떤 식으로든 최적화되거나 통합되지 않으므로 여러 함수를 호출하는 대신 단일 함수 호출에(예: 연결된 문자열에) 여러 반환 값을 포함할 수 있는지 고려하십시오. 마지막으로 외부 계산 엔진과 데이터를 주고받는 데 걸리는 시간도 고려해야 할 사항입니다.

분석

분석 패널을 통해 다음과 같이 다양한 고급 분석 기능에 액세스할 수 있습니다.

- 합계
- 참조 라인
- 추세선
- 예측
- 클러스터 분석(Tableau 10 의 새로운 기능)

이러한 분석은 대개 추가 쿼리를 실행할 필요가 없으며 쿼리 결과 캐시의 데이터에 대해 테이블 계산과 유사하게 수행됩니다. 테이블 계산과 마찬가지로 쿼리 결과에 대용량의 데이터 집합이 있다면 계산 프로세스에 다소 시간이 걸릴 수 있지만 일반적으로 통합 문서의 전반적인 성능을 크게 저하시키지는 않습니다.

총계 및 참조 라인의 성능에 영향을 주는 다른 요소는 측정값 집계의 가산법 여부입니다. 가산법 집계(예: SUM, MIN, MAX)인 경우 총계나 참조 라인의 계산이 Tableau 에서 로컬로 수행될 수 있습니다. 가산법이 아닌 집계(예: COUNTD, TOTAL, MEDIAN, PERCENTILE)인 경우 데이터 원본으로 돌아가서 총계/참조 라인 값을 계산해야 합니다.

계산과 기본 기능 비교

Tableau 의 기본 기능을 사용하면 손쉽게 특정 기능을 수행할 수 있는데도 불구하고 이 기능을 수행하기 위해 계산된 필드를 생성하는 경우도 있습니다. 예를 들면 다음과 같습니다.

- 차원 멤버를 그룹화하려는 경우 – 그룹 또는 집합을 사용하십시오.
- 측정값을 범위 구간에 그룹화하려는 경우 – 구간차원을 사용하십시오.
- 날짜를 월이나 주 같이 더 넓은 세부 수준에 맞춰 잘라내려는 경우 – 사용자 지정 날짜 필드를 사용하십시오.
- 두 개의 다른 차원을 연결하는 값 집합을 만들려는 경우 – 필드 결합을 사용하십시오.
- 날짜를 특정 형식으로 표시하거나 숫자 값을 KPI 표시기로 변환하려는 경우 – 내장 서식 지정 기능을 사용하십시오.
- 차원 멤버의 표시 값을 변경하려는 경우 – 별칭을 사용하십시오.

기본 기능을 항상 사용할 수 있는 것은 아니지만(예를 들어 가변 너비 구간차원이 필요한 경우 기본 구간차원은 이 기능을 제공하지 않음) 가능한 경우에는 기본 기능을 사용하는 것을 고려해 보십시오. 기본 기능을 사용하면 직접 계산하는 것보다 훨씬 효율적으로 작업을 처리할 수 있습니다. 또 Tableau 개발 팀이 성능 개선을 위해 꾸준히 노력하고 있는 만큼 가까운 시일 내에 다양한 기본 기능이 추가로 제공될 예정입니다.

데이터 유형이 미치는 영향

계산된 필드를 생성할 때에는 어떤 데이터 유형을 사용하느냐에 따라 계산 속도가 크게 달라질 수도 있다는 점을 유념하십시오. 일반적인 지침은 다음과 같습니다.

- 정수와 부울 값은 문자열과 날짜보다 훨씬 빠릅니다.

문자열과 날짜 계산은 매우 느립니다. 문자열 계산의 경우 대개 계산별로 실행해야 하는 기본 명령만 하더라도 10~100 개에 달하기 때문입니다. 이와 비교할 때 숫자 및 부울 값 계산은 매우 효율적입니다.

숫자 및 부울 값 계산이 문자열 계산보다 빠르다는 것은 비단 Tableau 계산 엔진뿐만 아니라 대다수의 데이터베이스에 적용되는 사실입니다. 행 수준 계산 및 집계 계산은 데이터베이스로 푸시되므로 행 수준 계산 및 집계 계산을 숫자 대비 문자열 논리로 표현할 경우 전자가 훨씬 빠르게 실행됩니다.

성능 개선 기술

계산의 효율성을 극대화할 수 있도록 다음 기술을 사용하는 것을 고려해 보십시오.

기본 논리 계산에 부울 값 사용

결과가 이진법으로 생성되는 계산을 수행해야 하는 경우(예: 예/아니오, 통과/실패, 초과/미만) 문자열이 아닌 부울 값으로 결과를 반환하도록 설정하십시오. 예는 다음과 같습니다.

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

이 예의 경우 문자열을 사용하므로 속도가 느립니다. 하지만 부울 값을 반환하도록 설정할 경우 이 작업을 더 신속하게 처리할 수 있습니다.

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

부울 값이 반환되면 별칭을 사용하여 TRUE 및 FALSE 로 반환된 결과를 'Today'(오늘) 및 'Not Today'(다른 날)로 변경합니다.

문자열 검색

제품 이름에 조회 문자열이 포함된 모든 레코드를 표시하려는 경우를 가정해 보겠습니다. 이 경우 매개 변수를 사용하여 사용자로부터 조회 문자열을 확보한 다음 아래와 같은 계산된 필드를 생성할 수 있습니다.

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

하지만 이 방식은 제약을 테스트하는 방법이 비효율적이므로 계산을 수행하는 데 시간이 많이 걸립니다. 이보다 더 효율적인 방법으로는 데이터베이스에 전달될 때 최적 SQL 로 변환되는 특정 CONTAINS 함수를 사용하는 방법을 들 수 있습니다.

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

하지만 이 경우 가장 좋은 방법은 바로 계산된 필드 대신 와일드카드 일치 필터 카드를 사용하는 것입니다.

조건 계산의 매개 변수

Tableau 가 일반적으로 사용하는 방식은 바로 최종 사용자에게 매개 변수를 제공하여 계산 수행 방식을 결정하는 값을 최종 사용자가 직접 선택하도록 하는 것입니다. 예를 들어 최종 사용자가 사용 가능한 값의 목록에서 원하는 항목을 선택하여 뷰에 표시되는 날짜 집계의 수준을 제어할 수 있도록 한다고 가정해 보겠습니다. 이때 대다수의 사용자는 문자열 매개 변수를 생성할 것입니다.

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

그런 다음 이 매개 변수를 다음과 같이 사용합니다.

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

더 좋은 방법은 내장 DATEPART() 함수를 사용하고 다음과 같이 계산을 만드는 것입니다.

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

Tableau 의 이전 버전에서는 후자의 방법이 훨씬 빨랐습니다. 하지만 더 이상 두 방법 간에 성능 차이가 없습니다. 매개 변수 값을 기반으로 CASE 문의 논리 요소를 축소하여 적절한 DATEPART() 함수 요소만 데이터 원본에 전달하기 때문입니다.

솔루션의 향후 유지 관리성에 대해서도 고려해야 합니다. 어느 경우든 후자의 계산이 더 나은 옵션일 수 있습니다.

날짜 변환

사용자가 보유하고 있는 날짜 데이터는 기본 날짜 형식으로 저장되지 않았을 가능성이 높습니다(예: 문자열 또는 숫자 타임스탬프). DateParse() 함수는 이러한 변환을 손쉽게 수행합니다. 사용자는 다음과 같은 서식 지정 문자열을 전달하기만 하면 됩니다.

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

DateParse() 함수는 데이터 원본의 하위 집합에서만 지원됩니다.

- 기존이 아닌 Excel 및 텍스트 파일 연결
- MySQL
- Oracle

- PostgreSQL
- Tableau 데이터 추출

데이터 원본에서 DATEPARSE() 함수를 지원하지 않는 경우 필드를 날짜 문자열로 파싱(예: '2012-01-01' - 참고로 국제화에 더 적합한 ISO 문자열을 사용하는 것이 더 좋음)한 다음 DATE() 함수로 전달하면 날짜를 Tableau 에서 사용하는 날짜 형식으로 변환할 수 있습니다.

원래 데이터가 숫자 필드인 경우 원래 데이터를 문자열로 변환한 다음 다시 날짜로 변환하는 것은 매우 비효율적입니다. 이 경우에는 숫자 데이터를 유지하고 DATEADD()와 날짜의 리터럴 값을 사용하여 계산을 수행하는 것이 훨씬 좋습니다.

예를 들어 ISO 형식 숫자 필드를 변환하는 다음과 같은 계산은 속도가 매우 느릴 수 있습니다.

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

하지만 다음과 같은 방식을 사용하면 계산을 더 효율적으로 수행할 수 있습니다.

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

데이터 원본에서 지원되는 경우 MAKEDATE() 함수를 사용하는 것이 더 낫습니다.

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

데이터 집합의 크기가 클 경우 성능은 훨씬 더 많이 개선될 수 있다는 점을 유념하십시오. 샘플 레코드가 10 억 개를 초과하는 경우 첫 번째 방법으로 계산하면 4 시간이 넘게 걸리지만 두 번째 방법으로 계산하면 1 분만에 계산을 완료할 수 있습니다.

논리 문

가장 자주 나오는 결과를 먼저 테스트

Tableau 에서는 논리 테스트를 수행할 때 일치하는 항목을 발견하면 더 이상 가능한 결과를 찾지 않습니다. 즉, 대부분의 경우에서 첫 번째 테스트 후 중지하도록 가장 가능성이 높은 결과를 먼저 테스트해야 합니다.

다음 예를 고려해 보십시오. 이 논리는

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

다음 논리보다 느리게 수행됩니다.

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

ELSEIF > ELSE IF

복잡한 논리 문을 사용할 때는 ELSEIF > ELSE IF 를 기억하십시오. 그 이유는 중첩된 IF 가 첫 번째 논리 문의 일부로 계산되는 것이 아니라 결과 쿼리의 중첩된 CASE 문을 계산하기 때문입니다. 따라서 아래와 같은 계산된 필드의 경우:

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

2 개의 중첩된 CASE 문과 4 개의 조건 테스트가 포함된 다음 SQL 코드를 생성합니다.

```
(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)
```

이 코드를 아래에서 중첩된 IF 대신 ELSEIF 로 다시 작성하면 더 빨리 실행되며, 조건 테스트를 사용하면 더 효율적입니다.

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

그 결과 SQL 코드에 CASE 문이 하나만 생성됩니다.

```
(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)
```

하지만 이 방법이 여전히 더 빠릅니다. 이 코드에서는 중첩된 IF 문을 사용하지만 조건 테스트를 가장 효율적으로 사용합니다.

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

결과 SQL 코드는 다음과 같습니다.

```
(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)
```

논리 확인 중복 방지

앞의 예와 마찬가지로 논리 확인이 중복되지 않도록 주의하십시오. 아래 계산의 경우:

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

다음 SQL 문이 생성됩니다.

```
(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)
```

다음과 같이 작성하면 더 효율적입니다.

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

다음 SQL 문이 생성됩니다.

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

기타 도움이 되는 팁

성능에 영향을 줄 수 있는 팁들이 많이 있습니다. 다음 도움말을 고려해 보십시오.

- 날짜 논리를 처리하는 일은 복잡할 수 있습니다. MONTH() 및 YEAR() 등 여러 날짜 함수를 사용하여 복잡한 테스트를 작성하기보다는 DATETRUNC(), DATEADD(), DATEDIFF() 등 내장 함수를 사용해 보십시오. 이렇게 하면 데이터 원본에 생성되는 쿼리의 복잡성을 현저히 줄일 수 있습니다.
- 고유 계산 값은 거의 모든 데이터 원본에서 가장 느린 집계 유형 중 하나입니다. 따라서 COUNTD 집계 사용은 되도록 자제하는 것이 좋습니다.
- 광범위한 영향을 주는 매개 변수(예: 사용자 지정 SQL 문에서)를 사용하면 캐시 성능에 영향을 줄 수 있습니다.
- 복잡한 계산을 필터링하면 초기 데이터 원본에서 색인이 누락될 가능성이 있습니다.
- NOW()는 타임스탬프 수준의 세부정보가 필요한 경우에만 사용하십시오. TODAY()는 날짜 수준 계산이 필요한 경우 사용하십시오.

- 모든 기본 계산은 레이블 문자열 같은 리터럴 계산조차도 초기 데이터 원본을 거칩니다. 레이블을 만들어야 하고(예: 열 머리글에 사용) 데이터 원본이 매우 크다면 레이블이 저장될 레코드가 하나만 있는 간단한 텍스트/Excel 파일 데이터 원본을 만들어서 큰 데이터 원본에 오버헤드가 추가되지 않도록 하십시오. 데이터 원본이 저장 프로시저를 사용하는 경우 특히 중요합니다. 자세한 내용은 이 섹션을 참조하십시오.

쿼리

Tableau 의 가장 강력한 기능 중 하나는 인메모리 데이터(예: 추출된 데이터 연결)와 실상황 데이터(예: 라이브 데이터 연결)를 모두 사용할 수 있다는 점입니다. 라이브 연결을 사용하면 데이터 원본에 내장된 처리 능력을 활용할 수 있기 때문에 라이브 연결은 Tableau 의 매우 강력한 기능입니다. 하지만 이 경우에는 성능이 원본 데이터 플랫폼에 의존적이므로 데이터 원본에 대해 가장 효율적이고 최적화된 쿼리를 생성하는 것이 중요합니다.

이미 언급한 바와 같이 질문이 더 복잡하고 표시할 데이터가 더 많으며 대시보드에 포함되는 요소가 더 많으면 데이터 원본에서 더 많은 작업을 수행해야 합니다. 통합 문서를 최대한 효율적으로 만들려면 쿼리 수를 최소화하고 쿼리를 최대한 효율적으로 수행하며 쿼리에서 반환하는 데이터의 양을 최소화하고 요청 간에 가능한 한 많은 데이터를 재사용하도록 해야 합니다.

자동 최적화

쿼리 성능 향상을 위해 수동으로 할 수 있는 작업이 있는 반면, 통합 문서를 효율적으로 실행하기 위해 Tableau 에서 자동으로 수행하는 다양한 최적화 기능도 있습니다. 이러한 최적화 기능은 직접 제어할 수 없으며 대부분 생각할 필요가 없지만 잘 이해하면 성능 개선에 활용할 수 있습니다.

병렬 실행 – 여러 쿼리를 한꺼번에 실행

Tableau 에서는 원본 데이터베이스의 능력을 활용하여 여러 쿼리를 동시에 실행합니다. 다음과 같은 대시보드를 가정해 보겠습니다.

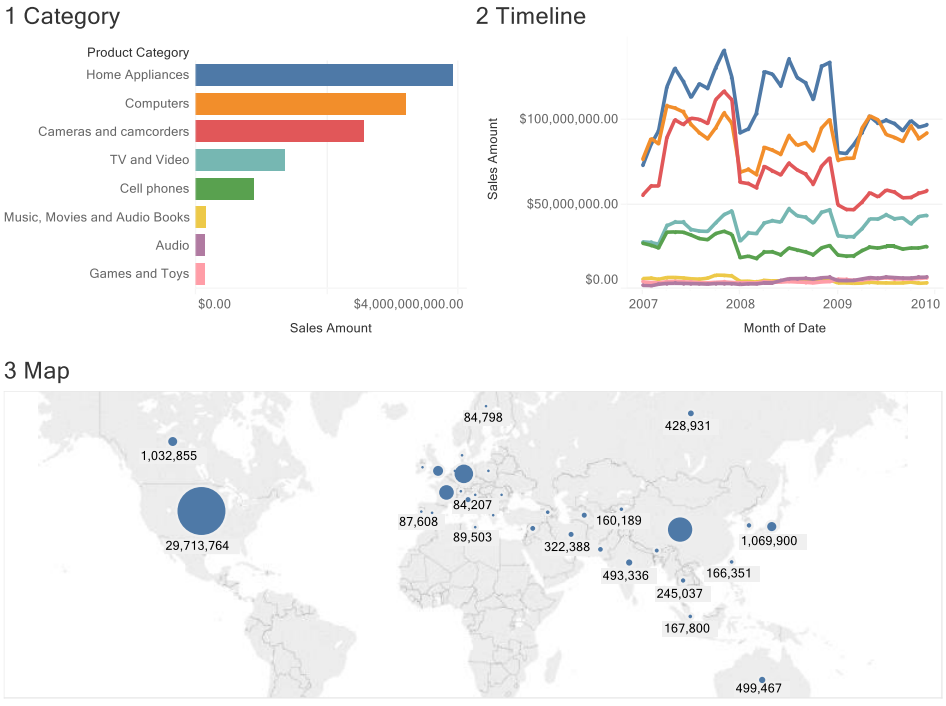
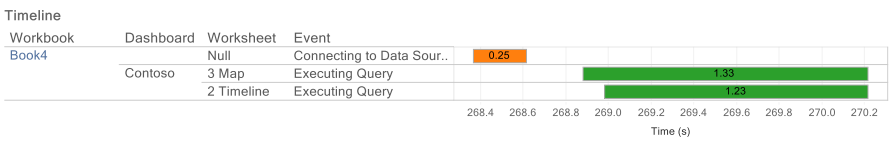


Tableau 에서 이 통합 문서를 열면 다음 쿼리가 표시됩니다.



보다시피 쿼리를 순차적으로 실행하면 2.66 초가 소요됩니다. 하지만 병렬로 실행하면 길이가 가장 긴 쿼리 시간(1.33 초)만큼만 소요됩니다.

일부 플랫폼은 다른 플랫폼보다 동시 쿼리를 더 잘 처리하므로 병렬 처리 수준은 원본 시스템에 따라 다릅니다. 기본값은 텍스트, Excel 및 통계 파일(한 번에 1 개의 쿼리로 제한됨)을 제외하고 모든 데이터 원본에 대해 최대 16 개의 병렬 쿼리를 실행하는 것입니다. 기타 데이터 원본에는 기본값 미만으로 한도가 설정되어 있을 수 있습니다. 자세한 내용은 다음 링크를 참조하십시오.

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

대부분의 경우 이 설정을 변경할 필요가 없으며 기본 설정으로 두어야 하지만 특별히 병렬 처리 수준을 제어해야 하는 경우 다음과 같이 설정할 수 있습니다.

- Tableau Server 의 병렬 쿼리 수에 대한 전역 제한
- SQL Server 와 같은 특정 데이터 원본 유형에 대한 제한
- 서버별 특정 데이터 원본 유형에 대한 제한 또는
- 특정 데이터베이스에 연결 중일 때 서버별 특정 데이터 원본 유형에 대한 제한

이러한 설정은 connection-configs.xml 이라는 xml 파일로 관리되며 이 파일을 만들어서 Tableau Desktop 의 경우 Windows(`C:\Program Files\Tableau\Tableau 10.0`) 및 Mac(App 을 마우스 오른쪽 버튼으로 클릭하고 패키지 내용 보기를 클릭한 다음 여기에 파일 배치)의 app 폴더에 저장하거나 Tableau Server 의 경우 vizqlserver 폴더의 config 디렉터리(예: `C:\ProgramData\Tableau\TableauServer\data\tabsvc\config\vizqlserver`)에 저장합니다. 이 구성 파일을 모든 작업자 컴퓨터의 모든 vizqlserver 구성 디렉터리에 복사해야 합니다.

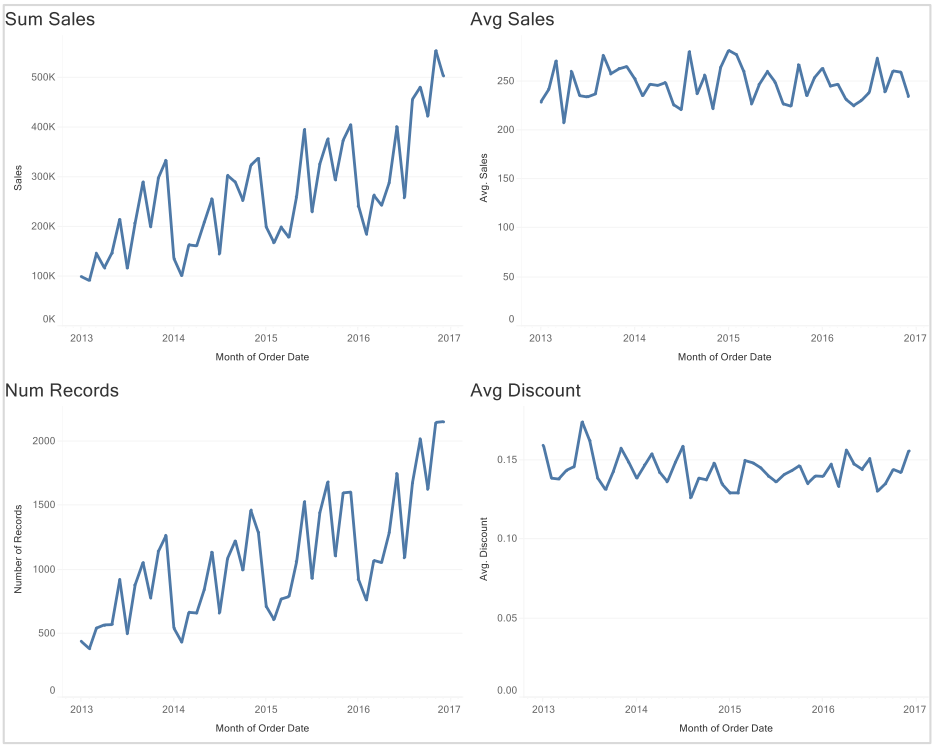
connection-configs.xml 파일의 구문을 비롯하여 병렬 쿼리 구성에 대한 자세한 내용은 다음 페이지를 참조하십시오.

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

쿼리 제거 – 되도록 적은 쿼리 실행

위의 예에서는 쿼리를 3 개가 아니라 2 개만 실행했습니다. 쿼리를 함께 일괄 처리하면 Tableau 에서 중복 쿼리를 제거할 수 있습니다. Tableau 의 쿼리 최적화 도구는 가장 복잡한 쿼리를 먼저 실행하도록 정렬하여 이어지는 쿼리에서 결과 캐시를 사용할 수 있도록 합니다. 예에서 보다시피 일정에 제품 범주가 포함되어 있고 판매량 금액의 합계 집계는 완전히 가산적이기 때문에 범주 차트의 데이터가 일정 워크시트의 쿼리 캐시를 사용하게 되며, 이에 따라 데이터 원본을 탐색할 필요가 없습니다.

또한 쿼리 최적화 도구는 동일한 세부 수준에 있는 쿼리(예: 같은 차원 집합으로 지정된 쿼리)를 찾고 이들은 요청된 모든 측정값을 반환하는 단일 쿼리로 축소됩니다. 다음과 같은 대시보드를 가정해 보겠습니다.



보다시피 대시보드에 각각 시간 경과에 따라 다른 측정값을 표시하는 4 개의 시트가 포함되어 있습니다. 연속된 월을 사용하여 데이터를 표시하므로 이들 시트는 모두 같은 세부 수준을 가집니다. Tableau 에서는 4 개의 별도 데이터베이스 요청을 실행하는 대신 이 요청을 모두 단일 쿼리로 결합합니다.

쿼리는 다음과 같습니다.

```
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))
```

보다시피 '쿼리 융합'이라는 이 최적화 기능은 전체 성능을 현저히 향상할 수 있습니다. 가능하다면 대시보드에 여러 시트가 있는 경우 동일한 세부 수준을 설정하는 것이 좋습니다.

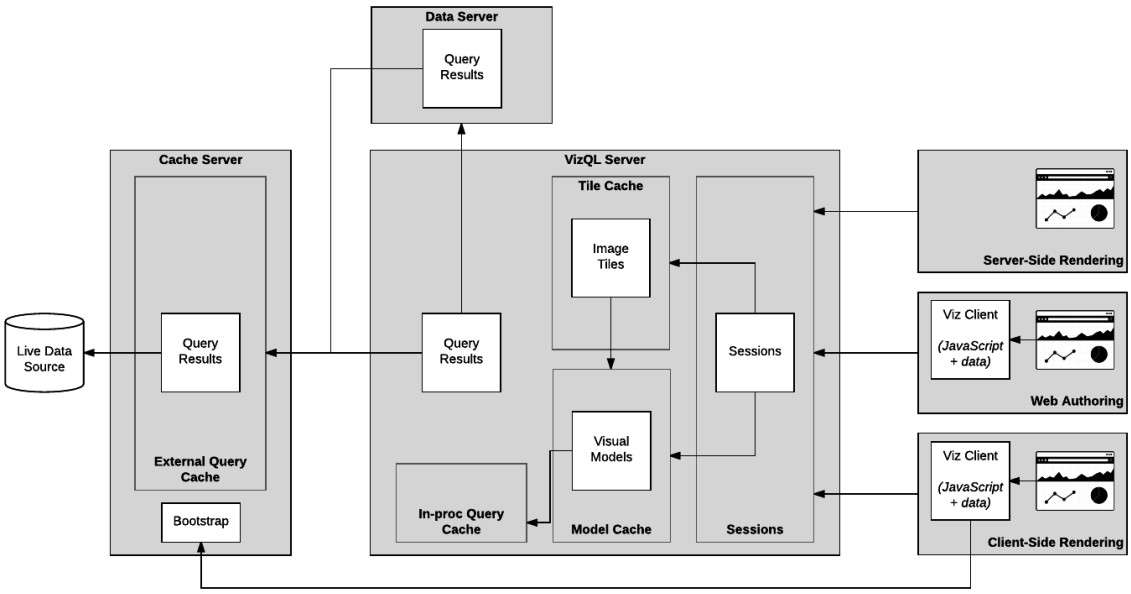
쿼리 융합은 TDE 데이터 원본에 대한 쿼리에서는 일어나지 않습니다.

캐싱 – 쿼리를 전혀 실행하지 않음

쿼리를 적게 실행하는 것보다 더 나은 것은 쿼리를 전혀 실행하지 않는 것입니다. Tableau Desktop 및 Tableau Server 에는 광범위한 캐싱이 있어서 초기 데이터 원본에서 실행되어야 하는 쿼리 수를 상당히 줄일 수 있습니다.

Tableau Server

Tableau Server 의 캐싱에는 다음과 같이 여러 계층이 있습니다.



첫 번째 캐싱 계층은 세션이 클라이언트 쪽 렌더링을 사용하는지 또는 서버 쪽 렌더링을 사용하는지 여부에 따라 다릅니다. 두 렌더링 모델과 Tableau 에서 어떤 렌더링을 사용할지 결정하는 방식에 대한 자세한 내용은 이전 섹션을 참조하십시오.

세션에서 클라이언트 쪽 렌더링을 사용하는 경우 브라우저에서 수행해야 하는 첫 번째 작업은 뷰어 클라이언트를 로드하는 것입니다. 여기서는 초기 뷰를 렌더링할 수 있도록 일련의 자바스크립트 라이브러리와 데이터를 로드합니다. 이 프로세스를 '부트스트래핑'이라고 하며 수행하는 데 몇 초가 소요될 수 있습니다. 대시보드는 여러 사용자가 볼 수 있으므로 이후 뷰 요청에 대한 대기 시간을 줄이기 위해 부트스트랩 응답이 캐시됩니다. 각 세션에서는 먼저 부트스트랩 캐시를 확인하여 이미 만들어져 있어 바로 사용할 수 있는 응답이 있는지 확인합니다. 응답을 찾으면 브라우저가 간단히 캐시에서 응답을 로드하여 초기 뷰를 매우 빠르게 렌더링할 수 있습니다. 뷰어 클라이언트가 클라이언트 브라우저에 로드된 뒤에는 로컬 데이터와의 일부 상호 작용 기능(예: 하이라이트, 도구 설명)이 완전히 수행될 수 있어 빠르고 원활한 사용자 환경이 구현됩니다.

대시보드가 서버 쪽 렌더링을 통해 표시되는 경우 서버는 '타일'이라는 일련의 이미지 파일로 대시보드 요소를 렌더링합니다. 이들 타일은 브라우저로 전달되고 조합되어 비주얼라이제이션을 표시합니다. 위에서와 같이 대시보드는 여러 사용자가 볼 것으로 예상되므로 서버는 이러한 이미지를 디스크에 캐시합니다. 각 요청은 타일 캐시를 확인하여 이미지가 이미 렌더링되었는지 확인하고 이미지를 찾으면 간단히 캐시에서 파일을 로드합니다. 타일 캐시에 있으면 응답 시간이 빨라지고 서버의 워크로드가 감소됩니다. 타일 캐시의 효율성을 높이는 한 가지 간단한 방법은 대시보드가 고정된 크기를 갖도록 설정하는 것입니다.

전체적으로 클라이언트 쪽 렌더링은 더 원활하고 응답성이 높은 사용자 상호 작용을 구현하며 Tableau Server 의 워크로드를 줄입니다. 따라서 가능하다면 클라이언트 쪽에서 렌더링될 수 있도록 통합 문서를 디자인해야 합니다.

다음 캐싱 계층은 시각적 모델이라고 합니다. 시각적 모델은 개별 워크시트를 렌더링하는 방법을 설명하므로 대시보드를 보면 사용된 워크시트당 하나씩 여러 시각적 모델을 참조할 수 있습니다. 여기에는 로컬 계산에서 얻은 결과(예: 테이블 계산, 참조 라인, 예측, 클러스터 분석 등)와 시각적 레이아웃(작은 여러 워크시트 및 크로스탭에 표시할 행/열 개수, 그릴 축 눈금선/격자선의 수 및 간격, 표시될 마크 레이블의 수 및 위치 등)이 포함됩니다.

시각적 모델은 VizQL Server 에서 만들고 인메모리로 캐시되며 가능한 경우 전체 사용자 세션에서 최대한 결과를 공유합니다. 시각적 모델을 공유할 수 있는지 여부의 중요한 요소는 다음과 같습니다.

- 비주얼라이제이션 표시 영역의 크기 – 모델은 비주얼라이제이션의 크기가 같은 세션에서만 공유될 수 있습니다. 대시보드가 고정된 크기를 갖도록 설정하면 재사용을 대폭 늘리고 서버의 워크로드를 낮출 수 있어서 타일 캐시에 유용한 만큼 모델 캐시에도 유용합니다.
- 선택 내용/필터의 일치 여부 – 사용자가 필터, 매개 변수를 변경하고 드릴다운/드릴업 등을 수행하는 경우 모델은 일치하는 뷰 상태가 있는 세션하고만 공유됩니다. '선택 내용 표시'가 선택된 통합 문서는 다른 세션이 일치할 가능성이 줄어들 수 있으므로 게시하지 않는 것이 좋습니다.
- 데이터 원본과 연결하기 위해 사용되는 자격 증명 – 데이터 원본과 연결하기 위해 자격 증명을 묻는 메시지를 사용자에게 표시하는 경우 모델은 동일한 자격 증명이 있는 사용자 세션하고만 공유될 수 있습니다. 이 기능은 모델 캐시의 효율성을 감소시킬 수 있으므로 주의하여 사용하십시오.
- 사용자 필터링이 사용되었는지 여부 – 통합 문서에 사용자 필터가 포함되어 있거나 USERNAME() 또는 ISMEMBEROF() 등의 함수를 포함하는 계산이 있는 경우 다른 사용자 세션과 공유되지 않습니다. 이들 기능은 모델 캐시의 효율성을 현저히 감소시킬 수 있으므로 주의하여 사용하십시오.

캐싱의 마지막 계층은 쿼리 캐시입니다. 이 계층은 이후 쿼리에서 사용할 수 있다고 예상하여 실행한 쿼리에서 얻은 결과를 저장합니다. 이렇게 캐시에 있으면 매우 효과적입니다. 데이터 원본에서 반복하여 쿼리를 실행할 필요가 없으며 캐시에서 데이터를 로드하기만 하면 되기 때문입니다. 또한 일부 경우 다른 쿼리의 결과를 사용하여 쿼리에 제공할 수도 있습니다. 이러한 이점은 이전에 쿼리 제거와 쿼리 융합에 대해 이야기할 때 논의되었습니다.

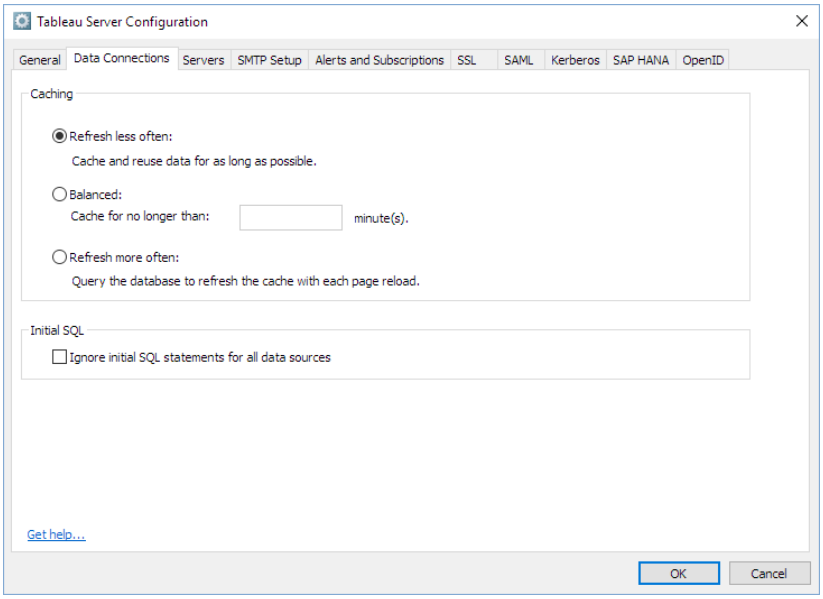
이 캐시에는 두 부분이 있습니다. 하나는 VizQL 프로세스 내에 있고('인프로세스 캐시'라고 함) 다른 하나는 캐시 서버를 통해 여러 프로세스에서 공유합니다('외부 캐시'라고 함). 쿼리가 이전에 같은 쿼리를 실행한 VizQL 인스턴스에 보내지면 요청이 인프로세스 캐시로부터 데이터를 받습니다. 이 캐시는 각 VizQL 프로세스에 대해 로컬이며 인메모리로 유지 관리됩니다.

이러한 인프로세스 캐시 외에도 VizQL 인스턴스에서뿐 아니라 초기 데이터 원본에 액세스하는 모든 프로세스에서 공유되는 외부 캐시가 있습니다(예: 백그라운더, 데이터 서버 등). 캐시 서버라는 서비스는 전체 클러스터의 외부 캐시를 관리합니다. 외부 캐시에는 일부 쿼리만 쓰여집니다. 쿼리가 매우 빠르게 실행되는 경우 캐시를 확인하는 것보다 쿼리를 다시 실행하는

것이 더 빠를 수 있으므로 최소 쿼리 시간 임계값이 있습니다. 또한 쿼리 결과가 매우 큰 경우 캐시 서버에 쓰는 것이 효율적이지 않을 수 있으므로 최대 크기 임계값도 있습니다.

외부 캐시는 여러 VizQL 인스턴스가 있는 배포에서 캐싱 효율성을 대폭 향상시킵니다. 일시적인 인프로세스 캐시와 달리 외부 캐시는 캐시 서버가 디스크에 데이터를 쓰기 때문에 지속적이며 서비스 인스턴스가 실행되는 동안 유지됩니다.

캐시를 가능한 한 오래 보존하도록 Tableau Server 설치의 설정을 조정하여 캐시의 효율성을 극대화할 수 있습니다.



사용 가능한 서버 용량이 있는 경우 캐시 크기를 늘릴 수 있습니다. 이러한 설정을 늘려도 문제가 없으므로 RAM 이 충분하다면 수치를 대폭 늘려서 콘텐츠가 캐시에서 푸시되지 않도록 할 수 있습니다. 캐시의 효율성은 로그 파일 또는 TabMon 을 통해 모니터링할 수 있습니다.

- **모델 캐시** – 기본 설정은 VizQL 인스턴스당 60 개의 모델을 캐시하는 것입니다. 다음과 같이 *tabadmin* 명령으로 이 설정을 조정할 수 있습니다. 사용 가능한 RAM 이 있다면 이 값을 늘려서 서버에 게시되는 뷰의 수와 일치시킬 수 있습니다.
    - tabadmin set vizqlserver.modelcachesize 200
- **인프로세스 캐시** – 기본 설정은 VizQL 인스턴스당 512MB 의 쿼리 결과를 캐시하는 것입니다. 적은 것 같지만 집계 쿼리의 쿼리 결과를 캐시하고 있다는 점을 기억해야 합니다. 이 설정은 다음과 같이 *tabadmin* 명령으로 설정할 수 있습니다.
    - tabadmin set vizqlserver.querycachesize 1024

또한 캐시 서버의 인스턴스를 더 추가하여 외부 쿼리 캐시의 용량과 처리량을 늘릴 수도 있습니다. VizQL Server 인스턴스당 캐시 서버의 인스턴스 한 개를 실행하는 것이 좋습니다.

- **외부 캐시** – 기본 설정은 캐시 서버 인스턴스당 512MB 의 쿼리 결과를 캐시하는 것입니다.

마지막으로 캐시 공유 시의 부작용이라면 통합 문서의 구독을 실행하여 첫 번째 뷰에서 느리게 실행되는 통합 문서의 캐시를 촉진할 수 있다는 것입니다. 상호 작용 사용자가 직장에 도착하기 전 이른 아침에 이를 수행하면 통합 문서가 실행되고 쿼리 결과가 캐시에 로드된 상태가

됩니다. 결과가 캐시에서 푸시되지 않았거나 중간 시기에 만료되었다고 가정하면 사용자가 통합 문서를 볼 때 캐시되어 초기 뷰가 표시되는 속도가 빨라집니다.

Tableau Desktop

Tableau Desktop 의 캐싱은 Tableau Server 보다 훨씬 간단합니다. Tableau Desktop 은 단일 사용자 애플리케이션이어서 여러 세션을 관리할 필요가 없기 때문입니다. Tableau Desktop 에는 다음과 같이 쿼리 캐시만 있습니다.

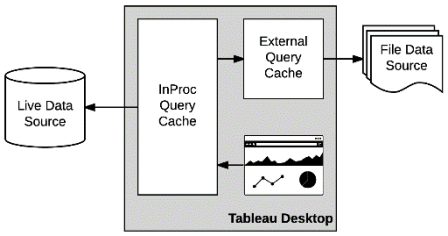


Tableau Server 처럼 인프로세스와 외부 캐시가 있습니다. 일시적인 인프로세스 메모리 기반 캐시는 모든 데이터 원본에 대한 연결에 사용되고 외부 캐시는 파일 기반 데이터 원본(예: 데이터 추출, Excel, Access, 텍스트 파일, 통계 파일 등)에만 사용됩니다. Tableau Server 에서와 같이 외부 캐시는 지속적입니다. 즉, Tableau Desktop 의 세션 사이에 캐시 결과가 보존되므로 파일 데이터 원본을 사용하는 경우 Tableau Desktop 을 다시 시작하고 다시 해당 데이터 원본을 사용하면 계속 캐시에서 사용할 수 있습니다.

외부 캐시는 Windows 의 경우 `%LOCALAPPDATA%\Tableau\Caching` 에 저장되고 Mac 에서는 `~/Library/Caches/com.tableau/`에 저장됩니다. 기본적으로 총 750MB 까지로 제한되며 사용자가 강제로 데이터 원본을 새로 고치면(예: F5, ⌘R 누름) 무효화됩니다.

마지막으로 Tableau Desktop 에서 파일을 패키지 통합 문서로 저장하면 외부 쿼리 캐시 데이터가 포함됩니다. 이를 통해 백그라운드에서 더 큰 데이터 원본 파일의 압축을 계속 해제하면서 Tableau Desktop 및 Tableau Reader 에서 통합 문서의 초기 뷰를 신속하게 렌더링할 수 있습니다. 또한 최종 사용자가 통합 문서를 열 때 응답성이 대폭 향상될 수 있습니다. 캐시 데이터는 데이터가 원본 데이터 파일 크기의 10% 이하인 경우에만 포함되며 기준 날짜 필터를 사용하는 쿼리는 포함되지 않습니다.

느린 연결

Tableau 10 이전 버전의 경우 사용자가 여러 데이터 원본이 포함된 통합 문서를 열면 처음에 자격 증명이 없는 모든 데이터 원본(예: 사용자가 사용자 이름/비밀번호를 입력할 필요가 없는 데이터 원본)에 연결했습니다. 즉, 사용자가 보고 있던 시트/대시보드/스토리에서 사용되지 않은 데이터 원본에 연결하느라 시간을 소비했을 가능성이 있습니다.

Tableau 10 에서는 이제 사용자가 선택한 시트/대시보드/스토리를 보는 데 필요한 데이터 원본에만 연결합니다. 이러한 변화로 Tableau Desktop 에서 탭 보기가 포함된 통합 문서를 로드하는 시간이 줄어들어 사용자가 데이터 탐색을 더 빠르게 시작할 수 있습니다.

조인

일반적으로 Tableau 에서 다중 테이블을 사용하는 경우 데이터 연결 창에 조인을 정의하는 것이 좋습니다. 조인을 정의할 때는 특정 쿼리를 정의하지 않고 간단히 테이블이 서로 관련되는

방식을 정의합니다. 필드를 비주얼라이제이션으로 드래그 앤 드롭하면 Tableau 에서 이 정보를 사용하여 필수 데이터만 가져오는 데 필요한 특정 쿼리를 생성합니다.

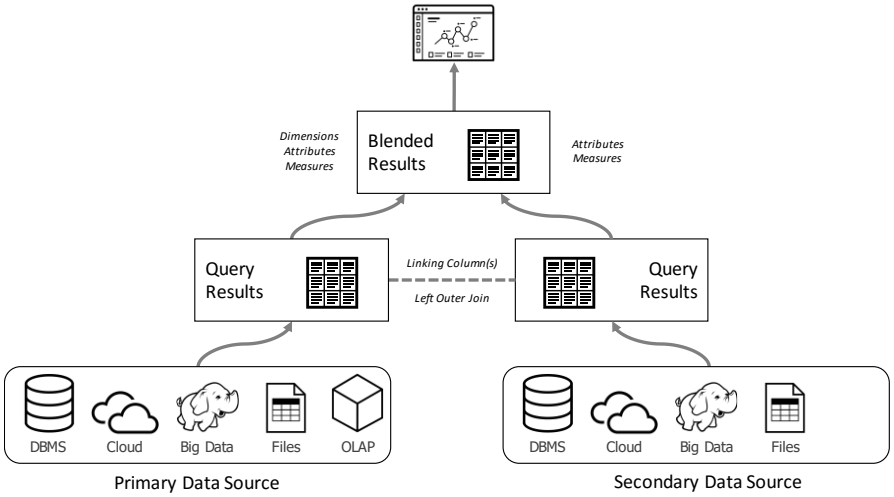
일반적으로 조인된 테이블 수는 특정 워크시트/비주얼라이제이션에 필요한 테이블만 포함하도록 최소화되어야 합니다. 데이터 연결과 워크시트의 복잡성에 따라 각 워크시트에 대한 데이터 연결을 분리하고 해당 시트에 특정 조인 패턴을 만드는 것이 나을 수도 있습니다.

테이블 간의 조인에 올바로 제한을 두면 조인 성능이 향상됩니다. 이러한 제한을 통해 Tableau Desktop 에서 특정 질문 하위 집합(범례, 필터 카드)에 답변하는 데 필요한 테이블로만 쿼리를 단순화할 수 있습니다.

혼합

Tableau 에서 데이터 테이블 조인 또는 데이터 테이블 혼합 중 하나를 선택할 때에는 데이터가 제공되는 출처, 데이터 연결 수, 그리고 데이터에 있는 레코드 수를 고려하십시오.

혼합을 사용하는 경우 혼합 성능 면에서 볼 때 각 데이터 원본에서 처리할 수 있는 레코드의 수가 아니라 두 데이터 집합을 연결하는 혼합 필드의 카디널리티가 가장 많은 영향을 끼칩니다. 혼합은 연결 필드 수준에서 두 데이터 원본의 데이터를 쿼리한 다음 두 쿼리의 결과를 Tableau 의 로컬 메모리에서 결합합니다.



따라서 고유한 값이 많으면 그만큼 많은 메모리를 사용하게 됩니다. 데이터를 혼합할 때 메모리 부족을 방지하려면 Tableau Desktop 64 비트 버전을 사용하는 것이 좋습니다. 그래도 이러한 성격의 혼합은 여전히 계산하는 데 시간이 오래 걸릴 수 있습니다.

따라서 혼합을 사용할 때에는 고유한 값이 많은 차원(예: 주문 ID, 고객 ID, 정확한 시간/날짜 등)을 혼합하지 않는 것이 가장 좋습니다.

기본 그룹 및 별칭

한 데이터 원본에는 '팩트' 레코드가 있고 다른 데이터 원본에는 차원 속성이 포함되어 있어 두 개의 데이터 원본을 혼합해야 하는 경우 기본 그룹 또는 별칭을 만들면 성능이 개선될 수도 있습니다.

기본 데이터 원본에 그룹이나 별칭을 만들어서 보조 데이터 원본에 나열된 속성을 반영하도록 하면 됩니다. 기본 그룹은 일대다 관계에 사용되고 기본 별칭은 일대일 관계에 사용됩니다.

그런 다음 기본 데이터 원본에서 관련 개체를 사용할 수 있으며 이에 따라 뷰 시간에 혼합할 필요가 없어집니다.

다음 예를 참조할 때에는 다음 세 가지 테이블을 고려하십시오.

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

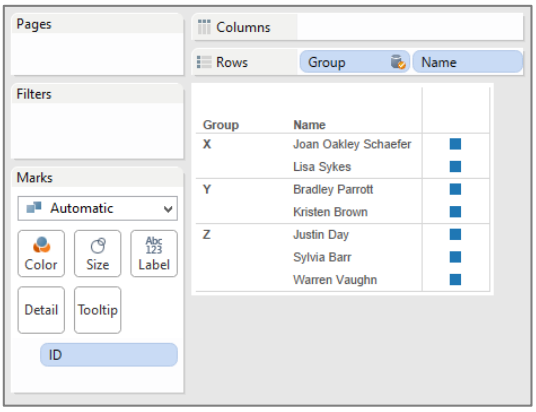
ID	Name
A	Lisa Sykes
B	Joan Oakley Schaefer
C	Bradley Parrott
D	Kristen Brown
E	Sylvia Barr
F	Warren Vaughn
G	Justin Day

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

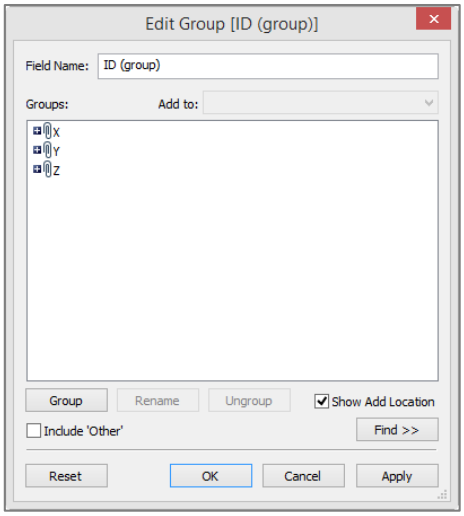
기본 그룹은 기본 데이터 원본의 차원 멤버에 일대다로 다시 매핑되는 속성이 보조 데이터 원본에 있는 경우 유용합니다. 위 데이터로부터 다음을 생성하려고 한다고 가정해 보겠습니다.

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

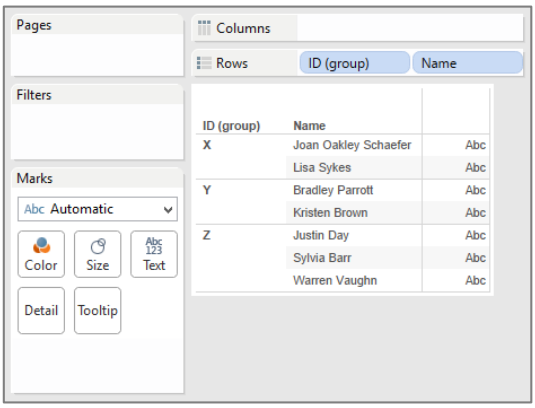
이 테이블은 혼합을 통해서도 만들 수 있지만 앞서 설명한 바 있듯이 ID 가 매우 많은 경우 혼합을 사용하면 성능이 크게 저하됩니다.



이 경우 마우스 오른쪽 버튼으로 그룹 필드를 클릭한 다음 '기본 그룹 만들기'를 선택하면 Tableau 는 연결 필드(이 경우 ID)를 보조 데이터 원본에서 선택한 차원(이 경우 그룹)으로 매핑하는 기본 데이터 원본에 그룹 개체를 생성합니다.



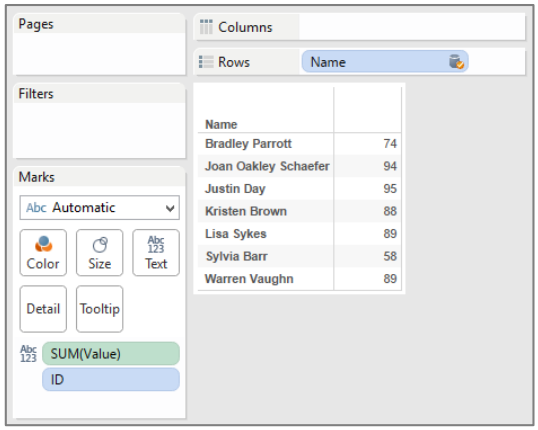
이렇게 하면 혼합을 사용하지 않아도 이 테이블을 다시 생성할 수 있습니다.



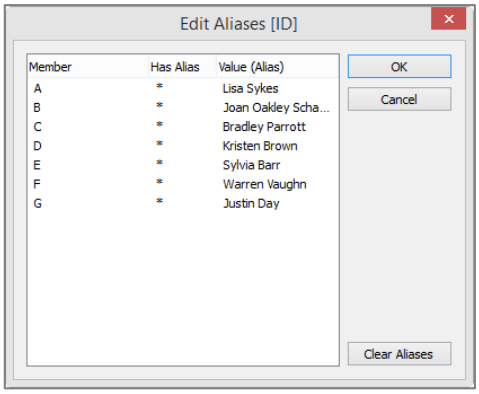
기본 별칭은 기본 데이터 원본의 차원 멤버에 일대일로 다시 매핑되는 속성이 보조 데이터 원본에 포함된 경우 유용합니다. 위 데이터로부터 다음을 생성한다고 가정해 보겠습니다.

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

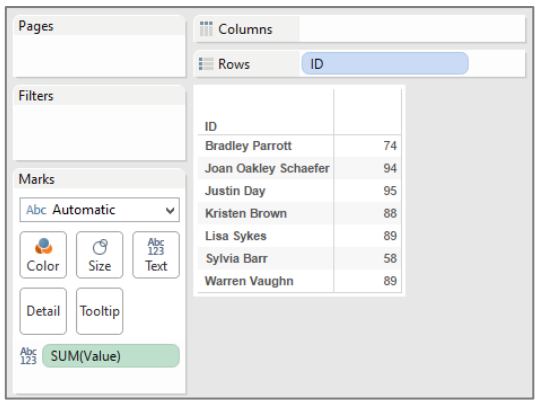
이 테이블은 두 데이터 원본을 혼합하는 방법으로도 만들 수 있지만 앞서 설명한 바 있듯이 ID 가 매우 많은 경우 혼합을 사용하면 성능이 크게 저하됩니다.



이 경우 이름 필드를 마우스 오른쪽 버튼으로 클릭한 다음 '기본 별칭 수정'을 선택하면 이름을 별칭 값으로 ID 필드에 한 번만 매핑할 수 있습니다.



이렇게 하면 혼합을 사용하지 않아도 필요한 비주얼라이제이션을 더 신속하게 생성할 수 있습니다.



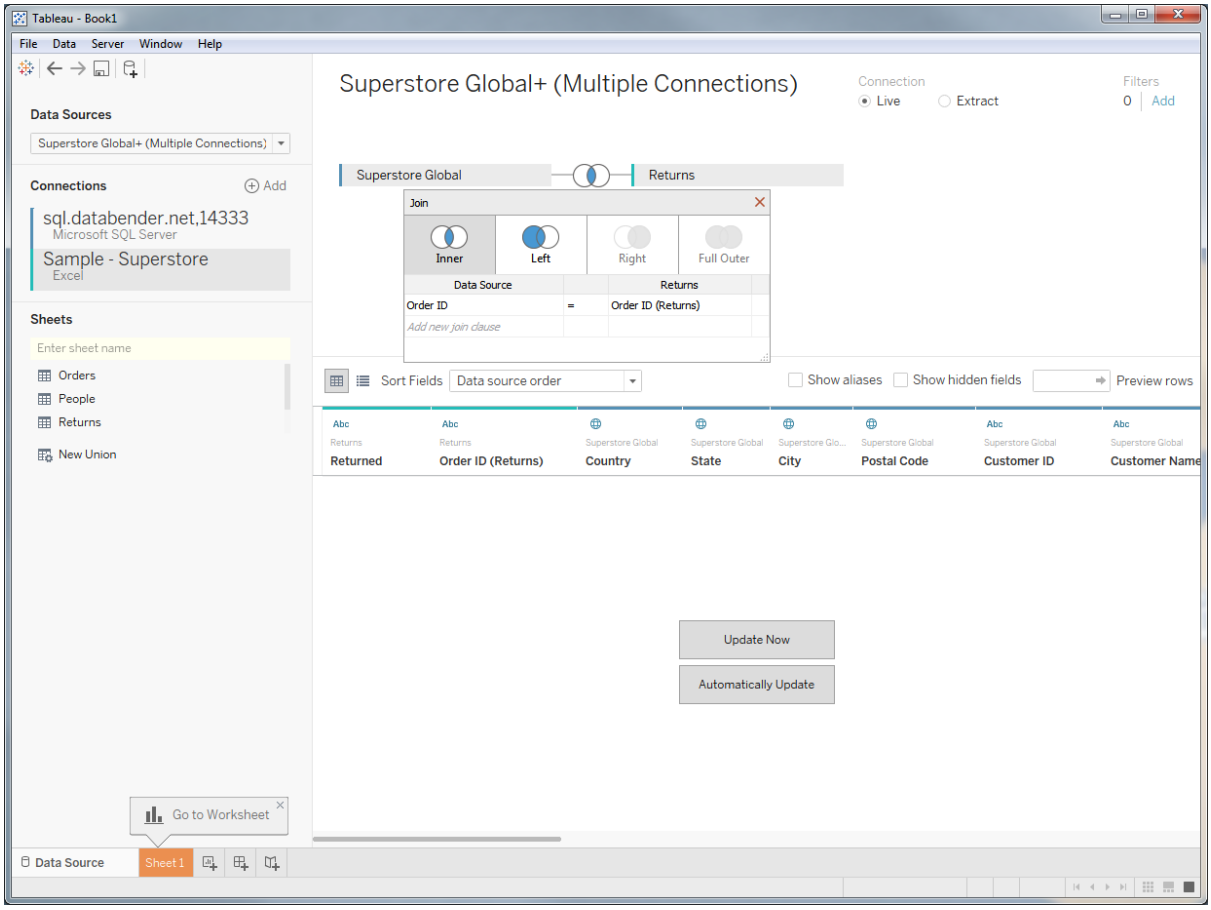
기본 그룹과 별칭은 모두 동적 기본 그룹과 별칭이 아니며 데이터가 변경될 경우 업데이트되어야 한다는 점에 유의하십시오. 위 예에서 기본 그룹과 별칭은 데이터가 자주 업데이트되는 경우 좋은 방법이 아니지만 데이터를 신속하게 매핑해야 할 때에는 부담이 많은 혼합을 사용하지 않아도 된다는 장점이 있습니다.

자세한 내용은 다음 기술 문서를 참조하시기 바랍니다.

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

데이터 통합

데이터 통합은 Tableau 10 에 도입된 새 기능입니다. 데이터 통합을 사용하여 데이터 원본이 서로 이질적일 수 있는 여러 데이터 연결을 통해 데이터를 결합할 수 있습니다.



데이터 통합과 데이터 혼합의 가장 큰 차이점은 데이터 통합은 행 수준 조인인 반면 데이터 혼합은 각 데이터 원본에서 집계된 결과 집합에서 수행된다는 점입니다. 즉, 데이터 통합은 초기 데이터 원본의 크기에 민감합니다. 네 가지 주요 고려사항은 다음과 같습니다.

- 이동해야 할 데이터가 많을수록 시간이 오래 걸립니다. 각 데이터 연결에서 행 수준으로 데이터가 추출되므로 데이터 양이 주요 요인입니다.
- 데이터가 멀리 이동해야 할수록 시간이 오래 걸립니다. 연결 지연 시간이 긴 데이터 원본에 조인하는 경우 성능이 영향을 받게 됩니다.
- 데이터 흐름 속도가 느릴수록 시간이 오래 걸립니다. 연결에 대역폭 제한이 있는 데이터 원본에 조인하는 경우 성능이 영향을 받게 됩니다.
- 일치시켜야 할 요소가 많을수록 시간이 오래 걸립니다. 첫 번째 항목과 마찬가지로 성능은 조인해야 하는 레코드 수의 영향을 받습니다.

Tableau 10 에서 데이터 통합은 추출된 데이터 원본에서만 사용할 수 있습니다. 예를 들어 각 데이터 연결에서 얻은 결과는 TDE 파일로 추출되어야 합니다. 그런 다음 이렇게 추출된 데이터가 Tableau Server 에 게시되어 다른 사용자가 사용할 수 있습니다.

Tableau 10 에서는 라이브 데이터 원본뿐 아니라 게시된 데이터 원본에서도 데이터 통합을 수행할 수 없습니다. 이러한 기능은 이후 개선 사항에 추가할 예정입니다.

사용자 지정 SQL

Tableau 를 처음 사용하는 사용자가 손으로 쓴 SQL 문을 사용하여 데이터 원본을 만드는 등 통합 문서에 기존 기술을 적용하려는 경우도 있습니다. 단순히 테이블 간 조인 관계를 정의할 때와 생성 중인 뷰와 관련된 SQL 을 쿼리 엔진이 작성하도록 할 때 Tableau 에서 훨씬 효율적인 쿼리를 생성할 수 있기 때문에 대개 이러한 방법은 역효과를 낳습니다. 하지만 데이터 연결 창에 조인을 지정해도 데이터에 관계를 정의하는 데 필요한 모든 유연성이 제공되지 않을 때가 있습니다.

손으로 쓴 SQL 문을 사용하여 데이터 연결을 만들면 매우 강력할 수 있지만 스파이더맨에 언급된 유명한 말처럼 '큰 힘에는 큰 책임이 따릅니다'. 사용자 지정 SQL 이 실제로 성능을 저하시키는 경우도 있습니다. 조인을 정의하는 것과 달리 사용자 지정 SQL 은 해체되지 않으며 항상 개별적으로 실행되기 때문입니다. 즉, 조인 선별이 발생하지 않으며 데이터베이스에 다음과 같이 단일 열에 대한 전체 쿼리를 처리하라는 명령이 전달되는 경우가 발생할 수 있습니다.

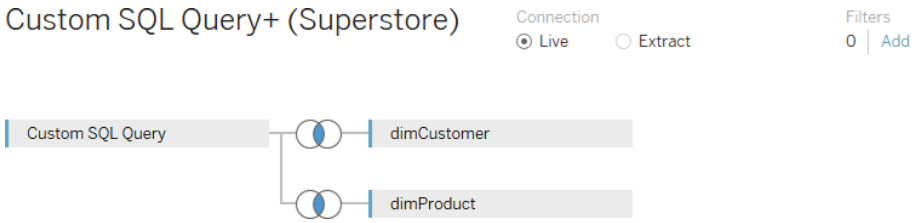
```
SELECT SUM([TableauSQL].[Sales])
FROM (
   SELECT [OrdersFact].[Order ID] AS [Order ID],
       [OrdersFact].[Date ID] AS [Date ID],
       [OrdersFact].[Customer ID] AS [Customer ID],
       [OrdersFact].[Place ID] AS [Place ID],
       [OrdersFact].[Product ID] AS [Product ID],
       [OrdersFact].[Delivery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

사용자 지정 SQL 문에 불필요한 구문이 포함되지 않도록 하는 것이 중요합니다. 예를 들어 쿼리에 GROUP BY 또는 ORDER BY 구문이 포함된 경우 Tableau 에서 비주얼라이제이션의 구조를 기반으로 자체 구문을 만들기 때문에 이러한 구문은 대개 부담을 가중시킬 뿐입니다. 가능한 경우 쿼리에서 이러한 구문을 삭제하십시오.

이때 가장 좋은 방법은 바로 사용자 지정 SQL 을 데이터 추출과 함께 사용하는 것입니다. 이렇게 하면 개별 쿼리가 데이터를 데이터 추출에 로드할 때에 한하여 한 번만 실행되며 Tableau 에서 진행되는 모든 후속 분석에는 데이터 추출에 대해 최적화된 동적 쿼리가 사용됩니다. 물론 모든 규칙에는 예외가 있으므로 여기에도 예외가 있습니다. 바로 사용자 지정 SQL 에 매개 변수가 포함되는 경우를 제외하고 사용자 지정 SQL 사용으로 데이터 추출이 만들어지는 경우입니다.

사용자 지정 SQL 문에서 매개 변수를 사용하면 라이브 연결이 더 성능 기준에 맞는 경우가 있습니다. 기본 쿼리가 더 동적일 수 있기 때문입니다(예: 매개 변수를 사용하는 필터 구문이 적절하게 수행됨). 또는 매개 변수를 통해 TOP 또는 SAMPLE 과 같은 성능 리미터의 값을 전달하여 데이터베이스에서 반환되는 데이터의 양을 제한할 수도 있습니다. 하지만 데이터 추출을 사용하는 경우 매개 변수를 변경할 때마다 삭제된 다음 다시 생성되므로 이로 인해 성능이 저하될 수 있습니다. 또한 매개 변수는 리터럴 값을 전달하는 데에만 사용할 수 있으며 SELECT 또는 FROM 구문을 동적으로 변경하는 데에는 사용할 수 없다는 점을 유념해 주시기 바랍니다.

마지막으로 다음과 같이 테이블을 사용자 지정 SQL 과 조인할 수도 있습니다.



조인하면 총계 스키마의 하위 집합을 참조하는 보다 구체적인 사용자 지정 SQL 을 작성할 수 있습니다. 그런 다음 여기에서 다른 테이블과의 조인이 일반 테이블 조인처럼 선별될 수 있으며 이를 통해 보다 효율적인 쿼리를 만들 수 있습니다.

사용자 지정 SQL 의 대안

Tableau 에서 사용자 지정 SQL 을 직접 사용하는 대신 초기 데이터 원본으로 쿼리를 이동하는 것이 더 나은 경우도 있습니다. 많은 경우 이를 통해 성능을 개선할 수 있습니다. 이 방법으로 데이터 원본이 쿼리를 더 효율적으로 파싱할 수 있습니다. 또는 복잡한 쿼리가 한 번만 실행해야 함을 의미할 수도 있습니다. 또한 정의 하나를 여러 통합 문서와 데이터 원본에서 공유할 수 있기 때문에 관리에 큰 도움이 됩니다.

이렇게 하는 방법에는 여러 가지가 있습니다.

뷰

거의 모든 DBMS 는 데이터베이스 쿼리의 결과를 나타내는 가상 테이블인 뷰의 개념을 지원합니다. 일부 데이터베이스 시스템의 경우 사용자 지정 SQL 문에서 쿼리를 가져와서 데이터베이스에 뷰 결과로 인스턴스화하는 것만으로 성능이 대폭 향상됩니다. 쿼리가 외부 SELECT 문에 포함되어 있을 때보다 쿼리 최적화 도구가 더 나은 실행 계획을 생성할 수 있기 때문입니다. 자세한 내용은 다음 커뮤니티 토론을 참조하십시오.

http://tabsoft.co/1su5YMe

사용자 지정 쿼리 논리를 Tableau 통합 문서 대신 뷰에 정의하면 여러 통합 문서와 데이터 원본에서 다시 사용할 수 있습니다. 또한 많은 DBMS 에서 뷰 쿼리의 결과가 미리 계산되고 캐시되어 쿼리 시 훨씬 빠르게 응답할 수 있도록 하는 스냅샷 또는 구체화된 뷰 개념을 지원합니다. 요약 테이블(아래 참조)과 유사할 수 있지만 DBMS 에서 자동으로 관리합니다.

축적 절차

축적 절차는 뷰와 비슷하지만 훨씬 더 복잡한 논리를 포함할 수 있으며 데이터 준비의 여러 단계를 수행할 수 있습니다. 또한 매개 변수화되어 사용자 입력을 토대로 대상 데이터 집합을 반환할 수 있습니다.

축적 절차는 Sybase ASE, SQL Server 및 Teradata 에서 지원됩니다. Tableau 에서는 단일 비주얼라이제이션에 대해 축적 절차를 여러 번 실행하지 않도록 축적 절차를 실행한 후 결과 집합을 데이터베이스의 임시 테이블에 씁니다. 그런 다음 비주얼라이제이션의 실제 쿼리가 임시 테이블에서 실행됩니다. 축적 절차 실행과 임시 테이블 작성은 통합 문서를 처음 열 때와 축적 프로세스 매개 변수가 변경될 때마다 수행됩니다. 이 프로세스를 수행하는 데 다소 시간이 걸릴 수 있으며 이러한 상호 작용은 속도가 느릴 수 있습니다.

매개 변수화된 축적 절차의 결과를 데이터 추출로 추출하면 매개 변수 값을 변경할 때마다 추출이 제거되고 새로 고침 상태가 됩니다.

축적 절차 사용에 대한 자세한 내용은 다음 Tableau Online 도움말을 참조하십시오.

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

요약 테이블

쿼리를 전송할 때마다 요약해야 할 정도로 매우 상세한 대용량 데이터 집합이 있는 경우(예: 개별 트랜잭션을 저장하지만 대개 날짜, 지역, 고객, 제품별로 요약된 데이터를 사용하는 경우) 요약 테이블을 만들고 Tableau 를 이 테이블에 사용하여 쿼리 시간을 단축하는 것이 좋습니다.

참고 – Tableau 데이터 추출을 활용하여 집계된 데이터 추출을 생성해도 쿼리 시간을 비슷하게 단축할 수 있습니다. 자세한 내용은 추출 섹션을 참조하십시오.

초기 SQL

사용자 지정 SQL 대신 사용할 수 있는 또 다른 대안으로는(데이터 원본에서 지원하는 경우) 초기 SQL 블록에 사용자 지정 SQL 문을 사용하는 방법이 있습니다. 이 방법의 경우 임시 테이블을 생성하면 쿼리에서 이 임시 테이블을 선택합니다. 초기 SQL 은 비주얼라이제이션이 변경될 때마다 실행되는 사용자 지정 SQL 과 달리 통합 문서를 열 때 한 번만 실행되므로 상황에 따라서는 성능을 크게 개선할 수 있습니다.

Tableau Server 에서 관리자는 초기 SQL 에 제한을 설정하여 실행되지 않도록 할 수 있습니다. 통합 문서를 게시하여 다른 사람과 공유하려는 경우 해당 환경에서 사용해도 괜찮은지 확인해야 할 수 있습니다.

초기 SQL 에 대한 자세한 내용은 다음 온라인 설명서를 참조하십시오.

http://onlinehelp.tableau.com/current/pro/desktop/ko-kr/help.htm#connect_basic_initialsql.html

데이터

Tableau 의 가장 강력한 기능 중 하나로는 다양한 플랫폼에 걸쳐 데이터를 연결할 수 있는 기능을 들 수 있습니다. 광범위한 관점에서 볼 때 위에서 말하는 플랫폼은 다음 중 하나에 해당한다고 볼 수 있습니다.

- 파일 기반 데이터 원본 – Excel, CSV 등
- 관계형 데이터베이스 데이터 원본 – Oracle, Teradata, SQL Server 등과 HP Vertica, IBM Netezza 같은 전문화된 분석 어플라이언스 등
- OLAP 데이터 원본 – Microsoft Analysis Services, Oracle Essbase 등
- '빅 데이터' 데이터 원본 – Hadoop 등
- 클라우드 기반 데이터 원본 – Salesforce, Google 등

각 유형의 데이터 원본은 저마다의 장단점을 가지고 있으며 고유한 방식으로 처리됩니다.

Tableau Desktop 는 Windows 와 Mac OS X 에서 모두 지원되며 Mac 에서 지원되는 데이터 원본 집합은 Windows 에서 지원되는 데이터 원본 집합과 다릅니다. Tableau 에서는 플랫폼 간 차이를 최소화하기 위해 노력하고 있지만 현재 일부 데이터 원본은 한 플랫폼에서만 지원됩니다.

일반 도움말

기본 드라이버 사용

Tableau 10 에서는 40 개 이상의 다양한 데이터 원본에 대한 기본 연결을 지원합니다. 이는 Tableau 에서 이러한 데이터 원본별 기술과 기능, 최적화를 구현했음을 의미합니다. 이러한 연결에 대한 엔지니어링과 테스트 활동을 통해 Tableau 에서 제공하는 가장 강력한 기술임을 보장합니다.

또한 Tableau 에서는 기본 커넥터 목록 외에도 데이터 원본에 액세스하기 위한 일반용 ODBC 도 지원합니다. 공개적으로 정의된 표준으로서 많은 데이터베이스 공급업체에서 자체 데이터베이스에 ODBC 드라이버를 사용할 수 있도록 하고 있으며 Tableau 에서도 이러한 ODBC 드라이버를 사용하여 데이터에 연결할 수 있습니다. 각 데이터베이스 공급업체에서 ODBC 표준의 기능을 해석하거나 구현하는 방법에는 차이가 있을 수 있습니다. Tableau 에서 특정 드라이버를 계속 사용하려면 데이터 추출을 만들어야 할 수도 있습니다. 또한 Tableau 에서 연결할 수 없는 ODBC 드라이버와 데이터베이스도 일부 있을 것입니다.

쿼리를 보내는 데이터 원본용 기본 드라이버가 있다면 ODBC 연결에 이 기본 드라이버를 사용해야 합니다. 대개 더 나은 성능을 제공하기 때문입니다. ODBC 연결은 Windows 에서만 사용할 수 있습니다.

가능하다면 데이터와 가까운 곳에서 테스트

앞서 말씀드린 대로 일반적으로 데이터 원본에서 쿼리를 느리게 수행하면 Tableau 환경도 느려집니다. 데이터 원본의 기본 성능을 테스트하는 좋은 방법은 가능하다면 데이터 원본이 있고 일부 쿼리를 실행할 시스템에 Tableau Desktop 을 설치하는 것입니다. 이렇게 하면 성능에서 네트워크 대역폭과 지연 시간 등의 요인이 제거되고 데이터 원본에서 쿼리의 기본 성능을 더 잘 파악할 수 있습니다. 또한 데이터 원본에 DNS 이름 대신 로컬 호스트 이름을 사용하면 느린 이름 해석이나 프록시 서버 등의 환경적 요인이 성능을 더욱 떨어뜨리는지를 판단할 수 있습니다.

데이터 원본

파일

이 범주에는 모든 파일 기반 데이터 형식, 즉 가장 일반적인 CSV, Excel 스프레드시트 및 MS Access 와 같은 텍스트 파일이 포함되며, 여기에는 통계 플랫폼 SPSS, SAS 및 R 에서 제공되는 데이터 파일도 포함됩니다. 비즈니스 사용자는 자주 이러한 형식의 데이터를 사용합니다. 보고서를 실행하거나 쿼리 추출을 수행하여 '관리되는' 데이터 집합에서 데이터를 가져오는 일반적인 방법이기 때문입니다.

일반적으로 파일 기반 데이터 원본을 Tableau 의 빠른 데이터 엔진으로 가져오는 것이 가장 좋습니다. 이렇게 하면 쿼리를 더욱 신속하게 실행할 수 있으며 데이터 값을 저장하는 파일의 크기를 크게 줄일 수 있습니다. 단, 파일의 크기가 작거나 데이터 변경사항을 반영하기 위해 파일에 대한 라이브 연결이 필요한 경우 라이브 연결을 사용할 수도 있습니다.

섀도 추출

기존이 아닌 Excel/텍스트 또는 통계 파일에 연결하면 Tableau 에서 연결 프로세스의 일부로 투명하게 추출 파일을 만듭니다. 이를 섀도 추출이라고 하며 이를 통해 파일에 직접 쿼리를 보냈을 때보다 훨씬 빨리 데이터를 사용할 수 있습니다.

대형 파일을 처음 사용할 때는 데이터 미리 보기 패널에 로드하는 데 몇 초가 소요될 수 있습니다. Tableau 가 파일에서 데이터를 추출한 뒤 섀도 추출 파일에 쓰기 때문입니다. 기본적으로 이러한 파일은 데이터 파일의 경로 이름과 마지막으로 수정한 날짜를 토대로 해시 이름이 지정되어 `C:\Users\<username>\AppData\Local\Tableau\Caching\TemporaryExtracts` 에 생성됩니다. Tableau 는 이 디렉토리에 최근에 사용된 파일 데이터 원본 5 개에 대한 섀도 추출을 보관하고 새 추출이 생성되면 사용된 파일 중 가장 오래된 파일을 삭제합니다. 이후에 섀도 추출이 있는 파일을 다시 사용하면 Tableau 에서 간단히 추출 파일을 열고 데이터 미리 보기가 거의 즉시 표시됩니다.

섀도 추출 파일에는 표준 Tableau 추출과 유사하게 초기 데이터와 기타 정보가 포함되어 있지만 섀도 추출 파일은 .ttde 확장자를 가진 다른 형식으로 저장됩니다. 즉, Tableau 추출과 같은 방식으로 사용될 수 없습니다.

Excel 및 텍스트 파일용 기존 커넥터

Tableau 의 이전 버전에서는 Excel 과 텍스트 파일에 연결하는 데 Microsoft 의 JET 데이터 엔진 드라이버를 사용했습니다. 하지만 Tableau 8.2 부터 내장된 기본 드라이버를 사용하여 더 나은 성능을 제공하고 더 크고 복잡한 파일에서 작동되도록 하고 있습니다. 그렇지만 사용자 지정 SQL 을 사용하려는 경우처럼 기존 드라이버를 사용하고 싶은 경우도 있습니다. 이 경우 사용자는 기존 JET 드라이버로 되돌릴 수 있습니다. MS Access 파일 읽기에는 여전히 JET 드라이버를 사용합니다.

두 드라이버 간 차이점에 대한 자세한 목록은 다음 페이지를 참조하십시오.

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

JET 드라이버는 Mac OS 에서 사용할 수 없으므로 Tableau Desktop for Mac 은 MS Access 파일 읽기를 지원하지 않으며 Excel 및 텍스트 파일에 대한 연결 옵션도 제공하지 않습니다.

관계형 데이터베이스

관계형 데이터 원본은 Tableau 사용자가 가장 일반적으로 사용하는 데이터 원본 유형으로 Tableau 는 광범위한 플랫폼에 대한 기본 드라이버를 제공합니다. 관계형 데이터 원본에는 행 또는 열 기반 데이터 원본, 개인 또는 엔터프라이즈용 데이터 원본, 기본 드라이버 또는 일반 ODBC 드라이버를 통해 액세스하는 데이터 원본 등이 있습니다. 이 범주에는 Hive 또는 Impala 와 같은 SQL 액세스 레이어를 통해 액세스하는 Map-Reduce 데이터 원본도 포함되나 이와 관련된 자세한 내용은 아래의 '빅 데이터' 섹션에서 설명하도록 하겠습니다.

관계형 데이터베이스 시스템(RDBMS)에서 쿼리의 속도에 영향을 미치는 내부 요소는 매우 다양합니다. 이러한 내부 요소를 변경 또는 조정하려면 대개 DBA 의 지원이 필요하나 내부 요소를 변경 또는 조정할 경우 성능을 크게 개선할 수 있습니다.

행 기반 및 열 기반

RDBMS 시스템은 두 개의 기본 유형인 행 기반 또는 열 기반으로 제공됩니다. 행 기반 저장소 레이아웃은 상호 작용 트랜잭션에서 더 많이 로드되는 OLTP 와 같은 워크로드에 더 적합합니다. 열 기반 저장소 레이아웃은 대개 대량의 데이터 집합에 대한 매우 복잡한 쿼리를 수반하는 분석 워크로드(예: 데이터 웨어하우스)에 더 적합합니다.

현재 많은 고성능 분석 솔루션은 열 기반 RDBMS 기반이며 이러한 솔루션을 사용하면 쿼리를 더 신속하게 수행할 수 있습니다. Tableau 에서 지원하는 열 기반 데이터베이스의 예로는 Actian Vector, Amazon Redshift, HP Vertica, IBM Netezza, MonetDB, Pivotal Greenplum, SAP HANA 및 SAP Sybase IQ 가 있습니다.

인터페이스로서의 *SQL*

많은 시스템에서 기존 RDBMS 기술을 기반으로 하지 않지만 여전히 SQL 기반 인터페이스를 제공하는 관계형 원본으로 나타납니다. Tableau 의 경우 여러 NoSQL 플랫폼(예: MarkLogic, DataStax, MongoDB 등)과 쿼리 가속화 플랫폼(예: Kognitio, AtScale, JethroData 등)이 여기에 포함됩니다.

색인

데이터베이스의 색인을 올바르게 구성하는 것은 쿼리 성능을 향상하는 데 매우 중요합니다.

- 테이블 조인에 포함된 모든 열에 색인이 있는지 확인합니다.
- 필터에 사용된 열에 색인이 있는지 확인합니다.
- 일부 데이터베이스의 경우 차원형 날짜 필터를 사용하면 쿼리에서 날짜 및 날짜/시간 열에 색인을 사용하지 않을 수 있습니다. 이와 관련된 내용은 필터 섹션에서 더 자세히 다루겠지만 범위 날짜 필터를 사용하면 날짜 색인을 항상 사용할 수 있다는 점을 참고하시기 바랍니다. 예를 들어 `YEAR([DateDim])=2010` 을 사용하는 대신 `[DateDim] >= #2010-01-01# and [DateDim] <= #2010-12-31#)`로 필터를 표현합니다.
- 쿼리 최적화 도구에서 양질의 쿼리 계획을 수립할 수 있도록 데이터에서 통계 기능을 사용 설정했는지 확인합니다.

대다수의 DBMS 환경에는 쿼리를 검토한 후 도움이 되는 색인을 권장하는 관리 도구가 포함되어 있습니다.

NULL

차원 열에 NULL 값을 사용하면 쿼리의 효율성이 줄어들 수 있습니다. 상위 10 개 제품에 대한 국가별 총 판매량을 표시하려는 비주얼라이제이션을 가정해 보겠습니다. Tableau 에서는 먼저 상위 10 개 제품을 찾는 하위 쿼리를 생성한 다음 이 쿼리를 다시 국가별 쿼리에 조인하여 최종 결과를 생성합니다.

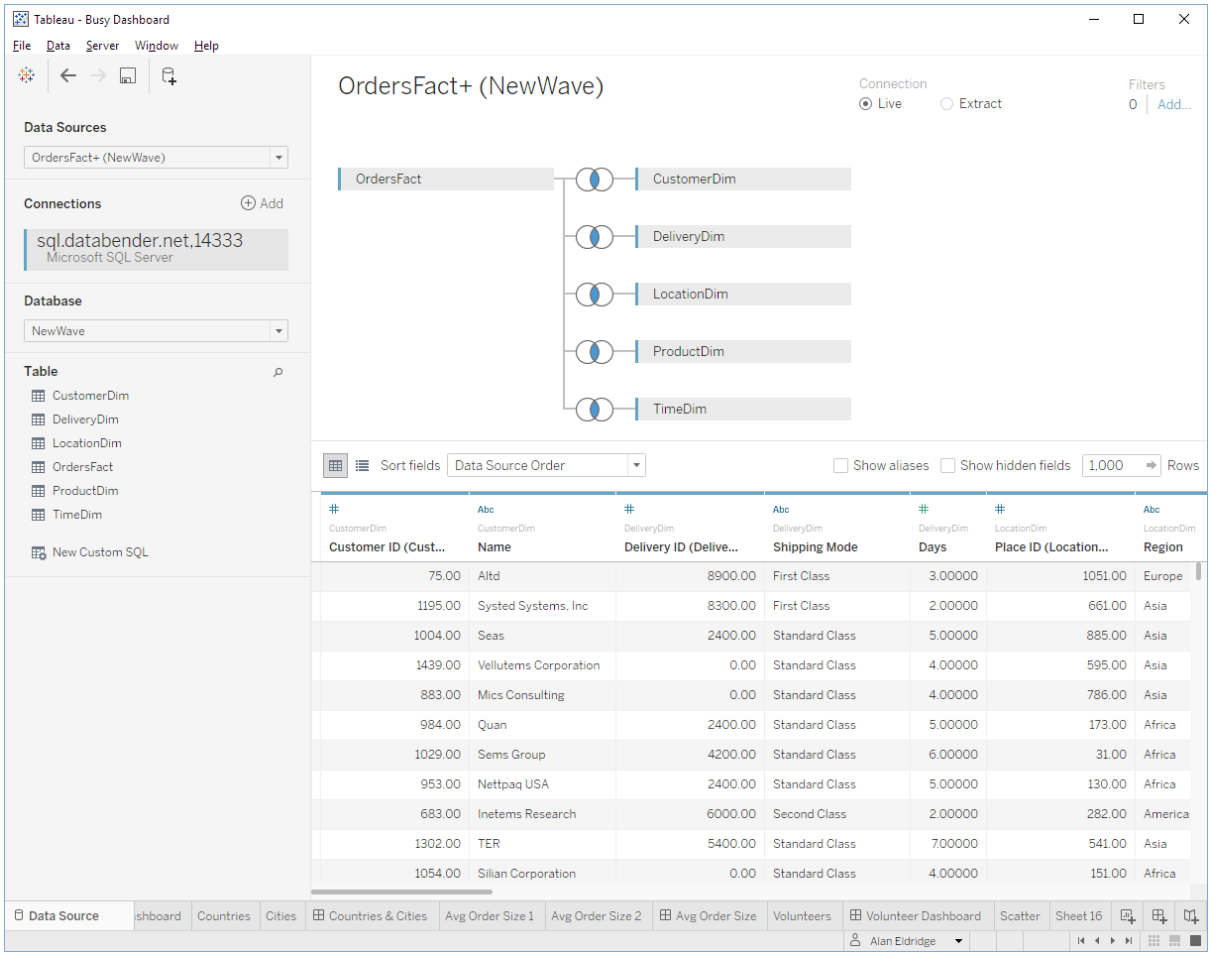
제품 열을 ALLOW NULL 로 설정하면 쿼리를 실행하여 NULL 이 하위 쿼리에서 반환한 상위 10 개 제품 중 하나인지 확인해야 합니다. 상위 10 개 제품에 해당하면 NULL 유지 조인을 수행해야 하는데, 이 조인은 일반 조인보다 아주 많은 계산이 필요합니다. 제품 열을 NOT NULL 로 설정하면 하위 쿼리 결과에 NULL 이 포함될 수 없음을 알고 있으므로 'check for NULL' 쿼리를 건너뛰고 일반 조인을 수행할 수 있습니다.

이런 이유로 가능하다면 차원 열을 NOT NULL 로 정의해야 합니다.

참조 무결성

Tableau 에는 데이터 원본에서 여러 개의 테이블을 조인할 때 사용되는 '조인 선별'(일반적으로 사용자에게는 표시되지 않음)이라는 유용한 기능이 있습니다. 데이터베이스 서버에서 조인을 처리하는 것은 많은 시간과 리소스를 필요로 하며 데이터 원본에서 조인을 선언할 때마다 열거하는 것은 매우 번거로운 일입니다. 조인 선별을 사용하면 조인에 정의된 모든 테이블에 쿼리를 보낼 필요 없이 관련된 테이블에만 쿼리를 보낼 수 있습니다.

크기가 작은 별모양 스키마에서 여러 개의 테이블을 조인으로 연결한 다음 시나리오를 고려해 보십시오.



조인 선별을 사용하여 판매량 측정값을 더블 클릭하면 다음 쿼리가 생성됩니다.

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

조인 선별을 사용하지 않으면 훨씬 효율적이지 않은 쿼리가 생성됩니다.

```
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 ()
```

측정값의 합계를 시작 단계에서부터 올바르게 계산하려면 모든 차원 테이블을 조인해야 합니다. 예를 들어 팩트 테이블에 2008~2012 년도 데이터가 포함되어 있지만 시간 차원 테이블에는 2010~2012 년도의 값만 있는 경우 시간 테이블이 포함되었는지 여부에 따라 SUM([판매량])의 결과가 바뀔 수 있습니다.

이전에는 DBMS 에서 규칙을 시행하는 '하드' 참조 무결성이 필요했습니다. 하지만 대부분의 고객은 애플리케이션 계층 또는 ETL 프로세스에 참조 무결성 규칙이 적용되는 데이터 원본을 사용합니다('소프트' 참조 무결성으로 지칭). 사용자는 '참조 무결성 가정'을 활성화하여 소프트 참조 무결성이 설정되어 있고 조인 선별을 안전하게 사용할 수 있음을 Tableau 에 알릴 수 있습니다.

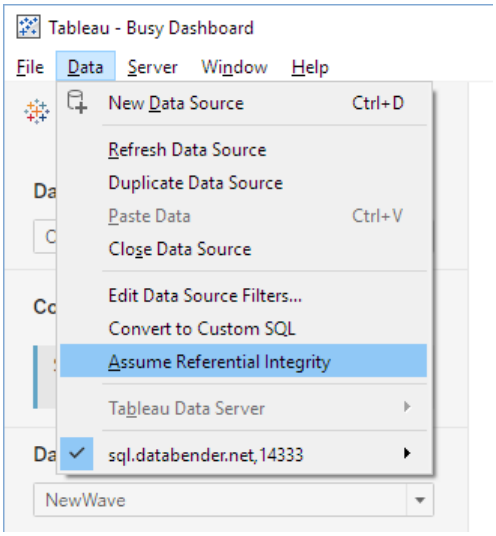


Tableau 는 하드 또는 소프트 참조 무결성을 사용할 수 있지만 주로 하드 참조 무결성을 사용하는 것이 훨씬 낫습니다. 데이터베이스에서 조인 선별도 수행할 수 있기 때문입니다. 자세한 내용은 Russell Christopher 가 Tableau Love 블로그에 게시한 다음 문서를 참조하십시오.

http://bit.ly/1HACmPV
http://bit.ly/1HACqPn

파티셔닝

데이터베이스를 분할하면 크기가 큰 데이블을 여러 개의 크기가 작은 개별 테이블(파티션 또는 샤드로 지칭)로 나누어 성능을 개선할 수 있습니다. 이는 다시 말해 스캔할 데이터가 줄어들고 I/O 를 지원할 드라이브가 늘어남에 따라 쿼리를 더 빨리 실행할 수 있다는 것을 의미합니다. 파티셔닝은 데이터의 양이 많을 때 권장되는 전략이며 Tableau 에서 손쉽게 파악할 수 있습니다.

Tableau 에서 시간, 지역, 범주 등 일반적으로 필터링되는 다양한 차원에 걸쳐 분할을 진행할 경우 각 쿼리가 한 파티션 내에 있는 레코드만 읽으면 되므로 좋은 효과를 거둘 수 있습니다.

단, 일부 데이터베이스의 경우 파티션 색인을 올바르게 사용하도록 하려면 '날짜 범위' 필터(차원형 필터가 아님)가 필요하다는 점을 유념하시기 바랍니다. 범위가 지정된 날짜 필터를 사용하지 않을 경우 테이블 전체가 스캔되므로 성능이 크게 저하될 수 있습니다.

임시 테이블

Tableau 에는 임시 테이블 사용으로 이어질 수 있는 다양한 작업(예: 애드혹 그룹 및 집합 만들기, 데이터 혼합 수행)이 있습니다. 사용자에게 임시 테이블을 만들고 드롭할 수 있는 권한을 부여하는 것이 좋으며 해당 환경에 사용자가 실행하는 쿼리를 위한 충분한 스풀 공간을 확보하십시오.

OLAP

Tableau 는 다양한 OLAP 데이터 원본을 지원합니다.

- Microsoft Analysis Services
- Microsoft PowerPivot(PowerPivot for Excel 및 PowerPivot for SharePoint 모두 지원)
- Oracle Essbase
- SAP Business Warehouse
- Teradata OLAP

MDX/DAX 와 SQL 간에 근본적인 언어 차이로 인해 OLAP 에 연결할 때와 관계형에 연결할 때 기능적인 차이가 있습니다. 유의해야 할 사항은 둘 다 Tableau 의 동일한 사용자 인터페이스, 동일한 비주얼라이제이션, 계산된 측정값에 대한 동일한 표현 언어가 있다는 점입니다. OLAP 와 관계형 데이터의 차이점은 대개 메타데이터(메타데이터가 정의되는 방식 및 위치), 필터링, 총계 및 집계의 작동 방식, 그리고 데이터 혼합 시 데이터 원본을 활용하는 방식과 관련이 있습니다.

관계형 데이터 원본과 OLAP 데이터 원본에서 Tableau 사용 시 차이점에 대한 자세한 내용은 다음 기술 문서를 참조하십시오.

> http://kb.tableau.com/articles/knowledgebase/functional-differences-olap-relational?lang=ko-kr

SAP BW 데이터 추출

SAP BW 는 OLAP 데이터 원본에서 고유합니다. SAP BW 큐브에서 데이터를 Tableau 의 데이터 엔진으로 추출할 수 있기 때문입니다(참고: 추출하려면 Tableau 에서 가져올 수 있는 특수 키코드 필요). Tableau 는 드릴스루 수준의 데이터가 아닌 리프 수준의 노드를 가져와 관계형 데이터 원본에 제공합니다. 다차원 데이터를 관계형 데이터로 전환할 때 일부 큐브 구조는 보존되지 않으므로 큐브 추출 시 추출과 라이브 연결을 필요할 때마다 전환하면 비주얼라이제이션의 상태에 영향을 미치게 됩니다. 따라서 비주얼라이제이션을 작성할 때에는 먼저 추출을 사용할지 아니면 라이브 연결을 사용할지 여부를 선택해야 합니다. 단, 시작 단계에서 모든 사항을 결정해야 하는 것은 아닙니다. 추출이 완료되고 나면 다양한 별칭 옵션(키, 긴 이름 등) 간에 전환하는 것이 가능합니다.

SAP BW 추출을 만드는 방법에 대한 자세한 내용은 다음 페이지를 참조하십시오.

> http://tabsoft.co/1SuYf9d

빅 데이터

빅 데이터는 데이터 분석 세계에서 여러 의미로 다양하게 사용되는 용어지만 본 문서에서는 특별히 Hadoop 기반의 플랫폼을 지칭하는 데 사용합니다. Tableau 10 에서는 다음과 같이 Hive 또는 Impala 연결을 지원하는 네 가지 지원되는 Hadoop 배포가 있습니다.

- 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* 와 *SharkServer2* 연결을 사용할 수 있지만 *Tableau* 에서는 지원하지 않습니다.

Hive 는 SQL-Hadoop 변환 계층의 역할을 담당하며 HDFS 데이터에서 실행할 수 있도록 쿼리를 MapReduce 로 변환합니다. Impala 는 SQL 문을 MapReduce 로 변환할 필요 없이 HDFS 데이터에서 바로 실행합니다. Tableau 에서는 Spark SQL 도 지원합니다. Spark SQL 은 디스크가 아니라 인메모리로 실행하여 MapReduce 보다 최대 100 배 더 빠르게 수행할 수 있는 빅 데이터용 오픈 소스 처리 엔진입니다.

Impala 는 일반적으로 Hive 보다 훨씬 빠르며 Spark 는 여전히 더 빠르다는 것이 입증되고 있습니다.

하지만 추가 구성 요소를 활용한다고 해도 Hadoop 은 Tableau 에서 생성하는 분석 쿼리를 처리하기에 충분한 응답성을 제공하지 못합니다. 이 경우 대개 쿼리 응답 시간을 개선하기 위해 Tableau 데이터 추출이 사용됩니다. 추출에 대한 자세한 내용 및 추출을 '빅데이터'와 함께 사용하는 방법은 본 문서의 다른 섹션에서 설명하도록 하겠습니다.

Hadoop 데이터 원본의 성능을 개선하는 방법에 대한 자세한 내용은 다음 페이지를 참조하시기 바랍니다.

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

클라우드

Tableau 에서는 현재 다음 클라우드 데이터 원본을 지원합니다.

- Salesforce.com
- Google Analytics
- oData(Windows Azure Marketplace DataMarket 포함)

첫 번째 그룹의 데이터 원본은 웹 서비스에서 데이터 레코드 집합을 읽은 다음 Tableau 데이터 추출 파일에 로드합니다. 이러한 데이터 원본의 경우 '라이브 연결' 옵션을 사용할 수 없지만 추출 파일을 새로 고쳐 포함된 데이터를 업데이트할 수 있습니다. 또 Tableau Server 를 사용하면 이 업데이트 프로세스를 자동화하거나 예약한 일정에 따라 진행할 수 있습니다.

Tableau 에서는 현재 다음 클라우드 기반 데이터 플랫폼도 지원합니다.

- Amazon Aurora
- Amazon Redshift
- Amazon RDS
- Google BigQuery
- Google Cloud SQL
- Google Sheets
- Microsoft Azure SQL 데이터 웨어하우스
- Snowflake

이러한 데이터 원본은 이전 데이터 원본 그룹과 달리 관계형 데이터 원본처럼 작동하며 라이브 연결과 추출을 허용합니다. 여기서는 더 자세히 다루지 않습니다(위의 관계형 데이터 원본 섹션 참조). 단, 일반적으로 클라우드로부터 대용량의 데이터 전송을 방지하기 위한 라이브 연결로 유지할 수 있다는 점을 알려드립니다.

마지막으로 Tableau 에서는 JSON, XML 또는 HTML 형식의 데이터를 게시하는 웹 기반 서비스에서 데이터를 가져오기 위한 일반 데이터 커넥터인 웹 데이터 커넥터도 제공합니다.

Salesforce

Salesforce 에 연결할 때 다음 커넥터 제한 사항을 고려해야 합니다.

'라이브 연결' 옵션 없음

다음을 비롯하여 라이브 연결 대신 추출 전용 옵션을 선택한 이유에는 여러 가지가 있습니다.

- 성능 – Salesforce 에 대한 라이브 분석 쿼리는 일반적으로 느립니다.
- API 할당량 - Salesforce 를 라이브로 너무 많이 사용하면 일일 할당량에 도달하여 계정이 일시 중단될 수 있습니다. 추출을 사용하면 API 를 효율적으로 사용하여 필요한 API 호출 수를 최소화하고 한도에 도달하는 것을 방지할 수 있습니다. 최적의 성능을 유지하고 모든 고객이 Force.com API 를 사용할 수 있도록 하기 위해 Salesforce.com 에서는 다음 두 가지 유형의 제한을 두어 트랜잭션 로드의 균형을 맞춥니다. 동시 API 요청 제한(http://sforce.co/1f19cQa) 및 총 API 요청 제한(http://sforce.co/1f19kiH)

초기 추출은 매우 느릴 수 있음

Salesforce 에서 처음 데이터를 추출하는 경우 테이블 크기, force.com 로드 등에 따라 시간이 걸릴 수 있습니다. 개체를 모두 다운로드해야 하기 때문입니다.

조인 옵션이 제한됨

다중 테이블 옵션을 선택하는 경우 PK/FK 키(왼쪽 및 내부 조인에만 해당)가 있는 개체만 조인할 수 있으니 유의해야 합니다.

데이터를 미리 필터링할 수 없음

Salesforce 커넥터를 통해 데이터를 미리 필터링할 수 없습니다. 미리 필터링해야 하는 경우 라이브 연결을 지원하는 타사 Salesforce ODBC 드라이버(예: Simba, DataDirect 등의 드라이버)를 사용할 수 있습니다. 이러한 드라이버를 사용하면 라이브 연결에서 추출을 가져올 수 있습니다.

DBAmp 도 Salesforce 데이터를 SQL Server 데이터베이스로 덤프할 수 있는 솔루션을 제공합니다. 그런 다음 SQL Server 커넥터를 통해 Tableau 에 연결할 수 있습니다.

수식 열을 마이그레이션할 수 없음

계산된 필드가 있는 경우 데이터를 추출한 뒤 Tableau 에서 다시 만들어야 합니다.

10,000 자의 쿼리 문자 수 제한이 있음

Force.com API 는 쿼리의 전체 문자 수를 10,000 자로 제한합니다. 폭이 넓은 테이블(열이 많고 열 이름이 긴)을 한 개 이상 연결하는 경우 추출을 생성할 때 문자 수 제한에 도달할 수 있습니다. 문자 수 제한에 도달한 경우에는 열을 선택 취소하여 쿼리의 크기를 줄여야 합니다. 경우에 따라서는 Salesforce.com 에서 사용자의 조직을 위해 쿼리의 문자 수 한도를 높일 수도 있습니다. 자세한 내용은 Salesforce 관리자에게 문의하십시오.

Google Analytics

보고서에 다수의 차원 또는 대량의 데이터가 포함된 경우 Google Analytics(GA)는 데이터를 샘플링합니다. 일반 GA 계정의 경우 주어진 날짜 범위 내에 특정 웹 사이트 데이터의 방문 횟수가 50,000 회를 초과하면 GA 에서 결과를 집계하여 이 데이터의 샘플 집합을 반환합니다. GA 에서 데이터의 샘플 집합이 반환되면 Tableau 는 뷰의 오른쪽 하단에 다음과 같은 메시지를 표시합니다.

'Google Analytics 에서 샘플링된 데이터가 반환되었습니다. 샘플링은 연결에 다수의 차원 또는 다량의 데이터가 포함된 경우 발생합니다. 샘플링이 보고서 결과에 어떠한 방식으로 영향을 미치는지 자세히 알아보려면 Google Analytics 문서를 참조하십시오.'

특정 샘플 데이터 집합에 따른 집계의 경우 유추 결과가 극심하게 왜곡되거나 부정확할 수 있으므로 데이터가 샘플링되는 경우를 파악하는 것은 매우 중요합니다. 예를 들어 데이터 내에서 특이한 범주를 설명하는 샘플 데이터 집합을 집계한다고 가정해 보겠습니다. 이 경우 이 범주에 포함된 샘플의 수가 충분하지 않으므로 집계한 샘플 집합을 토대로 한 유추 결과가 왜곡될 수 있습니다. 데이터에 대해 올바르게 유추할 수 있도록 도와주는 GA 뷰를 만들려면 유추하는 범주에 충분한 양의 샘플이 있어야 합니다. 참고로 권장되는 최소 샘플 크기는 30 개입니다.

GA 샘플 크기를 조정하는 방법과 GA 샘플링에 대한 자세한 내용을 확인하려면 다음 GA 문서를 참조하세요.

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

샘플링을 차단하려는 경우 선택할 수 있는 접근 방식에는 두 가지가 있습니다.

- 여러 개의 GA 보고서를 세션 또는 히트 수준으로 실행하여 데이터를 샘플링하지 않은 청크로 나눕니다. 그런 다음 데이터를 Excel 파일에 다운로드하고 Tableau 추출 엔진의 '데이터 원본에서 데이터 추가...'를 사용하여 단일 데이터 집합으로 다시 결합합니다.
- GA 를 프리미엄 계정으로 업그레이드합니다. 이렇게 하면 더 많은 레코드를 보고서에 포함할 수 있으므로 매우 손쉽게 데이터를 분석에 맞도록 청크 단위로 나눌 수 있습니다. 참고로 Google 은 GA 프리미엄 고객이 더 세부적인 분석을 진행할 수 있도록 세션 및 히트 수준의 데이터를 Google BigQuery 로 내보내는 기능을 지원할 계획이라고 밝힌 바 있습니다. 이 기능이 지원되면 Tableau 에서 BigQuery 를 직접 연결할 수 있어 원하는 작업을 더욱 손쉽게 진행할 수 있게 될 것입니다.

마지막으로 Tableau 가 GA 에 쿼리를 전송할 때 사용하는 API 는 쿼리의 차원을 최대 7 개, 측정값을 최대 10 개로 제한한다는 점을 유념해 주시기 바랍니다.

웹 데이터 커넥터

Tableau 웹 데이터 커넥터는 커넥터가 없이 데이터에 연결할 수 있는 방법을 제공합니다. 웹 데이터 커넥터를 사용하면 HTTP 를 통해 액세스할 수 있는 거의 모든 데이터에 연결할 수 있습니다. 여기에는 내부 웹 서비스, JSON 데이터, XML 데이터, REST API 및 기타 다양한 원본이 포함됩니다. 데이터를 수집하는 방법을 제어할 수 있어 여러 원본의 데이터를 조합할 수도 있습니다.

자바스크립트와 HTML 이 포함된 웹 페이지를 작성하여 웹 데이터 커넥터를 만들 수 있습니다. 웹 데이터 커넥터를 작성한 후에 Tableau Server 에 게시하여 다른 Tableau 사용자와 공유할 수 있습니다.

사용자가 웹 데이터 커넥터를 만들 수 있도록 Tableau 에서는 템플릿과 예제 코드, 웹 데이터 커넥터를 테스트할 수 있는 시뮬레이터가 포함된 소프트웨어 개발 키트(SDK)를 만들었습니다. 본 문서에도 웹 데이터 커넥터를 처음부터 만드는 방법을 안내하는 자습서가 포함되어 있습니다. 웹 데이터 커넥터 SDK 는 오픈 소스 프로젝트입니다. 자세한 내용은 GitHub 에서 Tableau webdataconnector 페이지를 참조하십시오.

웹 데이터 커넥터에 대한 자세한 내용은 다음 페이지에 제공되어 있습니다.

http://tabsoft.co/1NnGsBU

데이터 준비

통합 문서에서 사용해야 하는 데이터가 항상 완벽한 것은 아닙니다. 정리되지 않았거나 올바른 모양이 아니거나 여러 파일 또는 데이터베이스에 흩어져 있을 수 있습니다. 지금까지 Tableau 는 잘 정리되어 정규화된 데이터가 제공될 때 가장 잘 작동했습니다. 이는 Tableau 에 로드하기 전에 다른 도구를 사용하여 데이터를 준비해야 하는 경우가 있었음을 의미합니다.

하지만 더 이상 그럴 필요가 없습니다. Tableau 에는 이제 복잡한 데이터 로드를 지원하는 여러 기능이 있습니다. 다음과 같은 기능들이 여기에 속합니다.

- 피벗
- Union
- 데이터 해석기
- 열 병합

신속하게 데이터 준비를 수행하면 a) 분석 흐름을 따라갈 수 있으며 b) 사용자가 다른 도구에 액세스하지 않아도 Tableau 에서 계속 데이터를 분석할 수 있습니다. 하지만 이러한 작업에는 대개 많은 계산(특히 대용량의 데이터 경우)이 필요하고 보고서 사용자의 작업에 영향을 줄 수 있습니다. 가능하다면 이러한 작업을 진행한 후에는 데이터 추출을 사용하여 데이터를 구체화하는 것이 좋습니다.

또한 기존 추출, 변환 및 로드 프로세스를 보완하는 데 이러한 기능을 고려하고, 데이터가 여러 보고서에서 사용되거나 다른 BI 도구와 공유되는 경우 Tableau 의 데이터 업스트림 사전 처리가 더 낫고 더 효율적인지 확인해야 합니다.

데이터 추출

지금까지 우리는 데이터의 원래 형식을 보존하면서 데이터 연결의 성능을 개선하는 방법에 대해 살펴봤습니다. 이러한 방식은 라이브 데이터 연결로 지칭되는데 라이브 데이터 연결의

성능과 기능은 원본 데이터 플랫폼에 따라 달라집니다. 라이브 연결의 성능을 개선하려면 대개 데이터 원본을 변경해야 하는데 대다수 고객에게 이는 불가능한 작업에 가깝습니다.

이 경우 일반적인 사용자가 사용할 수 있는 대안은 바로 Tableau 의 신속한 데이터 엔진을 활용하여 원본 데이터 시스템에서 Tableau 데이터 추출로 데이터를 추출하는 것입니다. 이는 대부분의 사용자가 모든 데이터 원본에서 통합 문서의 성능을 대폭 향상할 수 있는 가장 빠르고 쉬운 방법입니다.

추출은 다음과 같습니다.

- 디스크에 기록되며 재생산이 가능한 영구적인 데이터의 캐시
- 열 형식의 데이터 저장소 - 데이터가 분석 쿼리에 최적화된 형식
- 쿼리를 전송하는 동안 데이터베이스와의 연결을 완전히 차단 (이로 인해 추출이 라이브 데이터 연결을 대체하게 됨)
- 추출을 완전히 재생성하거나 기존 추출에 여러 행의 데이터를 증분적으로 추가하는 방식으로 새로 고칠 수 있음
- 아키텍처 인식 – 다른 인메모리 기술과 달리 대부분의 추출에서는 사용 가능한 물리적 RAM 의 양으로 인한 제약이 없음
- 휴대성 – 추출은 파일 형식으로 보관되므로 로컬 하드 드라이브에 복사할 수 있으며 사용자가 회사 네트워크에 연결되지 않은 경우에도 사용할 수 있음. 또한 Tableau Reader 에서 사용할 수 있도록 배포되는 패키지 통합 문서에 데이터를 삽입하는 데에도 사용할 수 있음
- 대개 기본 라이브 데이터 연결보다 추출의 속도가 훨씬 빠름

The Information Lab 의 Tom Brown 이 작성한 다음 문서에는 추출을 사용하는 것이 더 좋은 사용 사례가 설명되어 있습니다. 다른 사용자의 예에 있는 의견도 읽어 보시기 바랍니다.

http://bit.ly/1F2iDnT

데이터 추출의 한 가지 중요한 점은 데이터 웨어하우스를 대체하는 것이 아니라 보완한다는 것입니다. 장기간에 걸쳐 데이터를 수집하거나 집계하는 데 추출을 사용할 수는 있지만(예: 정기적인 주기에 따라 데이터를 증분적으로 추가) 이는 장기적인 목적이 아닌 보완적인 해결책으로 활용되어야 합니다. 증분 업데이트의 경우 이미 처리된 레코드에 대한 동작 업데이트 또는 삭제를 지원하지 않습니다. 이 사항을 변경하려면 전체 추출을 다시 로드해야 합니다.

마지막으로 추출은 SQL Server Analysis Services 나 Oracle Essbase 같은 OLAP 데이터 원본을 통해 생성할 수 없습니다. 이 규칙의 예외는 SAP BW 에서 추출을 만들 수 있다는 점입니다(위의 관련 섹션 참조).

추출과 라이브 연결은 각각 언제 사용해야 합니까?

다른 모든 것과 마찬가지로 데이터 추출도 적절한 시간과 장소에 사용해야 합니다. 추출을 사용하는 것이 더 나은 시나리오는 다음과 같습니다.

- 쿼리 실행이 느린 경우 - Tableau Desktop 에서 생성한 쿼리를 원본 데이터 시스템에서 처리하는 속도가 느린 경우 추출을 생성하면 성능을 손쉽게 개선할 수 있습니다. 추출 데이터 형식은 본질적으로 분석 쿼리에 신속하게 응답할 수 있도록 설계되어 있으므로 추출은 쿼리의 속도를 높여주는 캐시라 할 수 있습니다. 일부 연결 유형(예: 크기가 큰 텍스트 파일, 사용자 지정 SQL 연결)의 경우 추출을 사용하는 것이 좋으며, 일부 원본의 경우 이 방식으로만 사용할 수 있습니다(클라우드 데이터 원본 섹션 참조).

- 오프라인 분석 - 원래 데이터 원본을 사용할 수 없는 상태에서 데이터를 가지고 작업을 수행해야 하는 경우(예: 출장 중이거나 자택에서 근무를 하여 네트워크에 연결할 수 없는 경우)에는 오프라인 분석을 사용하는 것이 좋습니다. 데이터 추출이 파일 형태로 유지되므로 노트북과 같은 휴대용 기기에 간편하게 복사할 수 있습니다. 또 네트워크에 지속적으로 연결할 수 없는 경우에는 추출과 라이브 연결 간에 손쉽게 전환할 수 있습니다.
- Tableau Reader/Online/Public 용 패키지 통합 문서 - 다른 사용자가 Tableau Reader 에서 열 수 있도록 통합 문서를 공유할 계획이거나 Tableau Online 또는 Public 에 게시할 계획인 경우 데이터를 패키지 통합 문서 파일에 삽입해야 합니다. 데이터 추출은 파일 데이터 원본과 같이 삽입이 가능한 데이터 원본을 사용하는 통합 문서의 경우에도 본질적으로 데이터를 고도로 압축합니다. 따라서 데이터 추출을 거친 패키지 통합 문서는 항상 크기가 작습니다.
- 추가 기능 - 일부 데이터 원본(예: 기존 JET 드라이버를 통한 파일 데이터 원본)의 경우 일부 Tableau Desktop 기능이 지원되지 않습니다(예: 중앙값/고유 개수/순위/백분위수 집계, IN/OUT 작업 설정 등). 이러한 기능을 간편하게 사용하는 방법 중 하나는 데이터를 추출하는 것입니다.
- 데이터 보안 - 원본 데이터 시스템에 있는 데이터의 하위 집합을 공유하려는 경우 추출을 생성한 다음 이 추출을 다른 사용자에게 제공할 수 있습니다. 이 방법을 사용하면 추출에 포함되는 필드/열을 제한할 수 있을 뿐만 아니라 다른 사용자가 요약된 값만 확인하고 개별 레코드 수준의 데이터는 확인할 수 없도록 집계된 데이터를 공유할 수도 있습니다.

추출은 매우 강력한 기능이지만 모든 문제에 대한 해결책은 아닙니다. 다음과 같은 시나리오의 경우에는 추출을 사용하는 것이 적합하지 않습니다.

- 실시간 데이터 - 추출은 특정 시점에 만든 데이터 스냅샷이므로 실시간 데이터를 분석에 사용해야 하는 경우에는 적합하지 않습니다. 물론 Tableau Server 를 사용하여 추출을 자동으로 새로 고칠 수 있으며 많은 고객이 업무 시간 중에 몇 번씩 이 작업을 수행하지만 실시간 데이터 액세스는 라이브 연결을 통해서만 제공할 수 있습니다.
- 대량 데이터 - 대량의 데이터를 가지고 작업을 진행해야 하는 경우('대량'의 정의는 사용자에 따라 다르지만 일반적으로는 수백만에서 수십억 개에 달하는 레코드가 있는 경우를 의미함) 데이터를 추출하는 것은 실용적이지 않을 수 있습니다. 추출 파일의 크기가 매우 크거나 추출 프로세스에 많은 시간이 소요되기 때문입니다. 단, 몇 가지 예외적인 경우는 있습니다. 대량의 원본 데이터 집합 중 필터링, 샘플링 또는 집계 등을 거친 데이터의 하위 집합을 가지고 작업을 수행하는 경우에는 추출을 사용하는 것이 매우 좋은 방법입니다. Tableau 추출 엔진은 최대 수억 개의 레코드를 원활하게 처리할 수 있도록 설계되었지만 데이터의 모양 및 카디널리티에 따라 성능은 달라질 수 있습니다.
- RAWSQL 통과 함수 - 통합 문서에서 통과 함수를 사용하는 경우 데이터 추출에서는 이 통과 함수가 작동하지 않습니다.
- 강력한 사용자 수준 보안 - 강력하게 적용되는 사용자 수준 보안 요구사항이 있는 경우 데이터 원본에도 이 요구사항을 구현해야 합니다. 통합 문서 수준에서 적용되는 사용자 수준 필터가 있는 경우 사용자가 언제든지 이 필터를 삭제하여 추출의 모든 데이터에 액세스할 수 있습니다. 단, 데이터 원본 필터를 정의한 상태로 추출을 Tableau Server 에 게시하였으며 다른 사용자가 데이터 서버를 통해 이 추출에 액세스하는 경우는 예외로

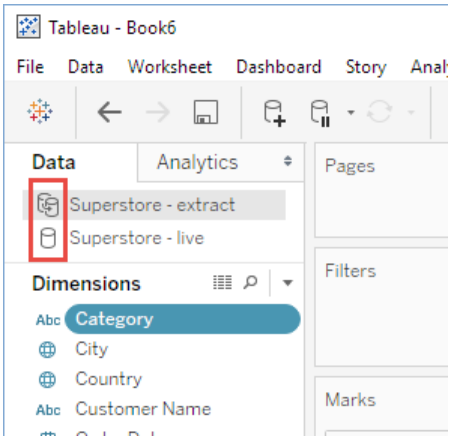
간주됩니다. 참고 - 적용되는 필터를 사용자가 우회할 수 없도록 사용자의 다운로드 권한이 취소되었는지 확인하시기 바랍니다.

Tableau Desktop 에서 추출 만들기

대부분의 경우 추출은 처음 Tableau Desktop 에서 만들어지며 매우 간단합니다. 먼저, 데이터를 연결하고 데이터 메뉴로 이동하여 '데이터 추출'을 클릭한 다음 대화 상자에서 기본 설정을 수락합니다(자세한 내용은 본 문서의 별도의 섹션에서 설명). 추출을 어디에 저장할 것인지 묻는 메시지가 표시되면 원하는 위치를 선택하여 파일을 저장합니다. 참고로 Tableau 가 기본적으로 제시하는 저장 위치는 '내 Tableau 리포지토리 | 데이터 원본'으로 여기에 저장하셔도 무방합니다.

추출이 생성될 때까지 대기합니다. 추출을 생성하는 데 소요되는 시간은 사용된 데이터베이스 기술, 네트워크 속도, 데이터 양 등에 따라 다릅니다. 또한 추출을 생성하는 데에는 메모리와 프로세서가 많이 사용되므로 워크스테이션의 속도와 성능도 속도에 영향을 미칩니다.

데이터 원본 아이콘이 변경되면 추출이 완료된 것입니다. 추출이 완료되면 원래 아이콘 뒤에 '복사본'(실제로 추출은 복사본을 만드는 작업이라 할 수 있음)을 나타내는 또 다른 아이콘이 표시됩니다.



이 방법으로 Tableau Desktop 을 통해서 추출을 만들 때 워크스테이션에서 처리되므로 워크스테이션에 작업을 완료할 충분한 용량이 있는지 확인해야 합니다. 추출을 생성할 때에는 CPU, RAM, 디스크 저장소, 네트워크 I/O 등 모든 유형의 리소스가 사용되며 성능이 떨어지는 PC 에서 대량의 데이터를 처리할 경우 리소스가 고갈되면 오류가 발생할 수 있습니다. 따라서 대량의 추출 작업을 수행할 때에는 빠른 멀티코어 CPU, 충분한 RAM, 빠른 I/O 등을 갖춘 적합한 워크스테이션에서 작업을 수행하시기 바랍니다.

추출을 생성하는 데 작업 파일을 쓸 임시 디스크 공간이 필요합니다. 이 크기는 최종 추출 파일 크기의 최대 2 배입니다. 이 작업 공간은 TEMP 환경 변수로 지정된 디렉터리에 할당됩니다(대개 `C:\WINDOWS\TEMP` 또는 `C:\Users\USERNAME\AppData\Local\Temp`). 이 드라이브의 공간이 부족한 경우 디스크 공간이 더 큰 위치로 환경 변수를 지정하십시오.

워크스테이션에서 초기 추출 프로세스를 수행하는 것이 불가능한 경우 다음과 같은 방법에 따라 내용이 없는 추출을 생성한 다음 이 추출을 Tableau Server 에 게시할 수 있습니다. `DateTrunc("minute", now())`가 포함된 계산된 필드를 만듭니다. 그런 다음 이 필드를 추출 필터에 추가하고 이 필드에서 표시하는
단일 값을 제외합니다. 1 분이 지나면 이 필터를 더 이상 사용할 수 없으므로 이 작업은 최대한

신속하게 수행하시기 바랍니다. 시간이 더 필요한 경우에는 게시 간격을 필요에 따라 5 분 또는 10 분이나 1 시간으로 늘리십시오. 이렇게 하면 데스크톱에 내용이 없는 추출이 생성됩니다. 이 추출을 서버에 게시한 다음 새로 고침 일정을 실행하면 앞서 제외한 타임스탬프가 더 이상 동일하지 않으므로 전체 추출 내용이 채워집니다.

데이터 추출 API 를 사용하여 추출 만들기

Tableau 에서는 개발자가 직접 Tableau 데이터 추출(TDE) 파일을 만들 수 있도록 애플리케이션 프로그래밍 인터페이스(API)도 제공합니다. 개발자는 이 API 를 사용하여 사내 소프트웨어와 소프트웨어 서비스(SaaS)에서 추출을 만들 수 있습니다. 이 API 를 통해 사용 중인 데이터 원본에 대한 기본 커넥터가 없을 때 체계적으로 Tableau 에 데이터를 가져올 수 있습니다.

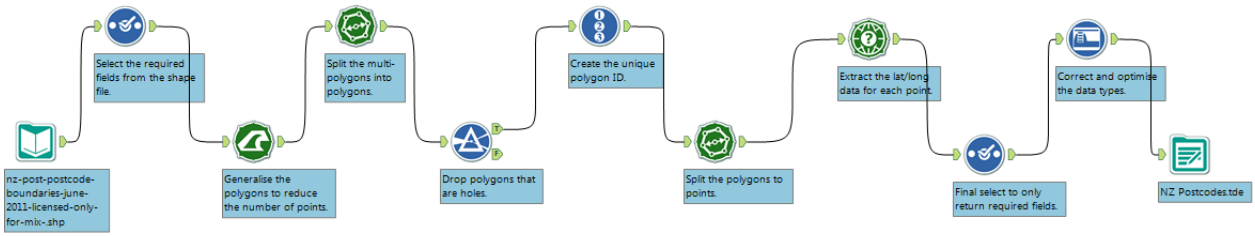
Windows 와 Linux 에서 Python 및 C/C++/Java 를 사용하는 개발자는 이 API 를 사용할 수 있습니다. AP 에 대한 자세한 내용은 다음 페이지를 참조하십시오.

http://onlinehelp.tableau.com/current/pro/desktop/ko-kr/extracting_TDE_API.html

타사 도구를 사용하여 추출 만들기

다양한 타사 도구 개발자들은 데이터 추출 API 를 사용하여 애플리케이션에 기본 TDE 결과를 추가해 왔습니다. 이러한 애플리케이션에는 Adobe Marketing Cloud 같은 분석 플랫폼과 Alteryx, Informatica 같은 ETL 도구가 포함되어 있습니다.

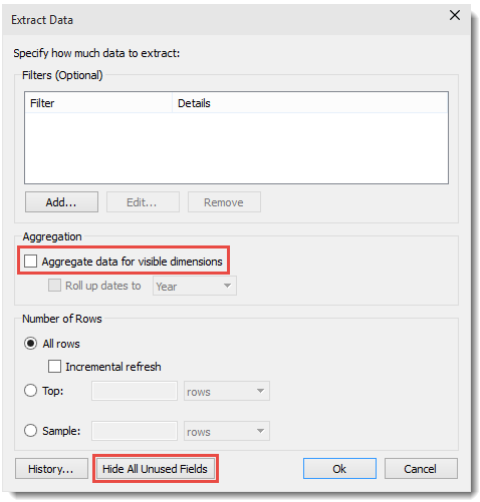
복잡한 데이터 준비가 필요한 경우 Alteryx 와 Informatica 같은 도구를 사용하면 추출, 변환 및 로드(ETL) 단계를 효율적으로 수행한 다음 준비된 데이터를 곧바로 TDE 파일로 출력하여 Tableau Desktop 에서 사용할 수 있습니다.



집계 추출

집계 추출 기능은 항상 성능을 개선하는 데 도움이 됩니다. Teradata 또는 Vertica 에서 대량의 데이터를 가지고 작업을 수행하는 경우에도 데이터를 추출하여 적절한 방식으로 집계하고 필터링하면 성능을 개선할 수 있습니다. 예를 들어, 가장 최근 데이터만 확인해야 하는 경우에는 데이터를 필터링하여 가장 최근 데이터만 표시할 수 있습니다.

원하는 필드를 선택한 다음 Tableau Desktop 의 데이터 추출 대화 상자에서 '표시된 차원에 대한 데이터 집계' 확인란을 선택하면 미리 추출을 정의할 수 있습니다. 또는 분석을 수행하고 대시보드를 만든 다음 게시하기 전에 데이터 추출 대화 상자로 다시 돌아가 '사용하지 않는 모든 필드 숨기기' 버튼을 클릭하여 데이터를 추출할 때 뷰를 생성하는 데 필요한 최소 데이터만 추출할 수도 있습니다.



집계 추출 생성은 대용량의 기본 데이터가 있지만 전체 데이터 집합에 쿼리를 보내어 요약 뷰를 만들어야 할 때 매우 강력한 기술입니다. 예를 들어 10 년 간의 판매량을 나타내는 세부 트랜잭션 데이터에 대한 레코드가 십억 개에 달하는데 전체 10 년의 영업 추세를 표시하려는 경우 십억 행 모두에 대해 쿼리를 수행해야 하기 때문에 초기 뷰의 쿼리가 느려질 수 있습니다. 연간 수준으로 집계된 추출을 만들면 추출에 10 개의 숫자만 있으므로 뷰를 표시할 때 필요한 쿼리를 줄일 수 있습니다. 물론 이는 지나치게 단순화된 예입니다. 실제로는 시간뿐 아니라 더 많은 차원이 있지만 뷰를 표시할 때 쿼리되어야 하는 레코드 수를 현저히 줄이는 효과가 있습니다.

각각 특정 세부 수준을 지원하도록 조정된 집계된 추출을 여러 개 만들거나 집계된 추출을 라이브 연결과 결합하여 여러 세부 수준이 있는 매우 세분화된 통합 문서를 만들 수 있습니다. 예를 들어 고도로 집계된 추출을 사용하는 초기 요약 뷰 집합이 있어도 세부로 들어갈 때는 요약 뷰에서 라이브 연결을 통해 연결되는 다른 시트에 동작 필터를 사용해야 합니다. 이는 요약 뷰가 전체 기본 데이터 집합을 샅샅이 조사할 필요가 없을 뿐 아니라 드릴다운 동작을 지원할 기본 데이터를 모두 추출할 필요가 없기 때문에 빠르게 수행될 것임을 나타냅니다. 또한 라이브 연결은 드릴다운 수준에서 소규모 레코드 집합에만 액세스하기 때문에 신속하게 수행됩니다.

이 방법을 사용하면 다양한 수준에서 조합 및 집계가 가능하므로 거의 모든 성능 문제가 해결되며, 궁극적으로는 원하는 결과를 최대한 빨리 확인할 수 있습니다. Tableau 는 메모리를 효율적으로 사용하므로 이 방법을 사용하면 대개 성능을 비교적 쉽게 개선할 수 있으며 동시에 여러 개의 추출을 실행할 수 있습니다.

추출 최적화

Tableau Server 는 데이터베이스에 있는 실제 열뿐만 아니라 Tableau 에서 생성되어 추가된 열 또한 최적화합니다. 이 열에는 결과가 변경되지 않는 문자열 조작 및 연결과 같은 결정 계산의 결과뿐만 아니라 그룹 및 집합 또한 포함되며 실행 중에 계산되는 매개 변수나 집계(합계 또는 평균 등)가 포함된 비결정 계산 결과의 경우에는 저장되지 않습니다.

두 행의 데이터만 추가한 다음 추출을 새로 고쳐도 추출의 크기가 100MB 에서 120MB 로 늘어나는 것을 확인하셨을 것입니다. 추출의 크기가 이와 같이 크게 늘어난 까닭은 데이터를 필요할 때마다 다시 계산하는 것보다 디스크에 저장하는 것이 더욱 경제적이므로, 추출을 최적화하는 과정에서 계산된 필드 값이 포함되어 있는 추가 열이 생성되었기 때문입니다.

참고로 데이터 추출 연결에 대한 중복 복사본을 만들 경우 '최적화' 또는 '새로 고침' 옵션에서 선택한 연결에 모든 계산된 필드가 포함되어 있는지 확인하십시오. 그렇지 않을 경우 Tableau 는 사용되지 않는다고 간주하는 필드를 구체화하지 않습니다. 이를 방지하는 가장 좋은 방법은 바로 기본 데이터 원본에서 모든 계산된 필드를 정의한 다음 필요에 따라 이 필드를 다른 연결에 복사하고 기본 데이터 원본에서 추출한 항목만 새로 고치거나 최적화하는 것입니다.

> 경고: 개발 팀에 따르면 단일 *TDE* 에 대한 다중 연결을 만들 수는 있지만 실제로 이를 지원하도록 개발되지 않았습니다. 연결이 *TDE* 구조와 동기화되지 않으면 많은 문제를 일으킬 수 있습니다. 따라서 수행하지 않는 것이 좋습니다.

추출 새로 고침

Tableau Desktop 에서 데이터 메뉴 > [내 데이터 원본] > 추출 > 새로 고침을 선택하면 내가 생성한 추출을 새로 고쳐 데이터를 업데이트하고 새 행을 추가할 수 있습니다. 이와 달리 Tableau Server 에서는 게시 프로세스를 진행하는 동안 또는 게시 프로세스가 완료된 후에 관리자가 정의한 일정을 연결하여 추출을 자동으로 새로 고칠 수 있습니다. 추출 증분 시 허용되는 최소 새로 고침 일정 간격은 15 분이며 매일 또는 매주 같은 시간 등 원하는 시간에 새로 고치도록 일정을 설정할 수 있습니다. 또는 '이동 창'을 설정하여 가장 최신 데이터만 표시되도록 데이터를 지속적으로 새로 고칠 수도 있습니다.

참고: 데이터 새로 고침 간격을 15 분 미만으로 설정하려면 라이브 데이터에 연결하거나 동기화된 보고서 데이터베이스를 설정해야 합니다.

각 추출에 대해 선택할 수 있는 새로 고침 일정은 총 두 가지입니다.

- 증분 새로 고침은 행만 추가하며 기존 행의 변경사항은 반영되지 않습니다.
- 전체 새로 고침은 현재 추출을 삭제하고 데이터 원본에서 새 추출을 처음부터 다시 생성합니다.

추출을 새로 고치는 데 소요되는 시간이 설정된 새로 고침 간격보다 긴 경우 어떠한 상황이 발생합니까?

추출의 새로 고침 시간이 새로 고침 간격보다 큰 경우 그 사이에 있는 새로 고침은 생략됩니다. 예를 들어 1 시간마다 데이터를 새로 고치도록 일정을 설정했지만 데이터가 너무 커서 새로 고치는 데 1 시간 30 분이 소요되는 경우

- 첫 번째 새로 고침은 1 시에 시작되어 2 시 30 분에 종료됩니다.
- 그 다음 새로 고침은 2 시에 시작될 예정이지만 첫 번째 새로 고침이 실행 중이어서 생략됩니다.
- 그 다음 새로 고침이 3 시에 시작되어 4 시 30 분에 종료됩니다.

추출 유지 관리

유지 관리 화면에는 현재 실행 중인 백그라운드 작업과 지난 12 시간 동안 실행되었던 작업이 표시되며 각 작업의 상태는 색상 코드로 표시됩니다. 유지 관리 화면은 관리자가 확인할 수 있으며 적절한 권한이 부여된 일부 사용자도 확인할 수 있습니다. 참고로 이 일부 사용자는 추출에 대한 즉각적인 업데이트를 시작할 수 있는 권한도 보유할 수 있습니다. 또한 로드할 데이터베이스가 있는 경우 데이터베이스 로드가 완료된 다음에 추출을 시작하도록 트리거를 설정할 수도 있습니다.

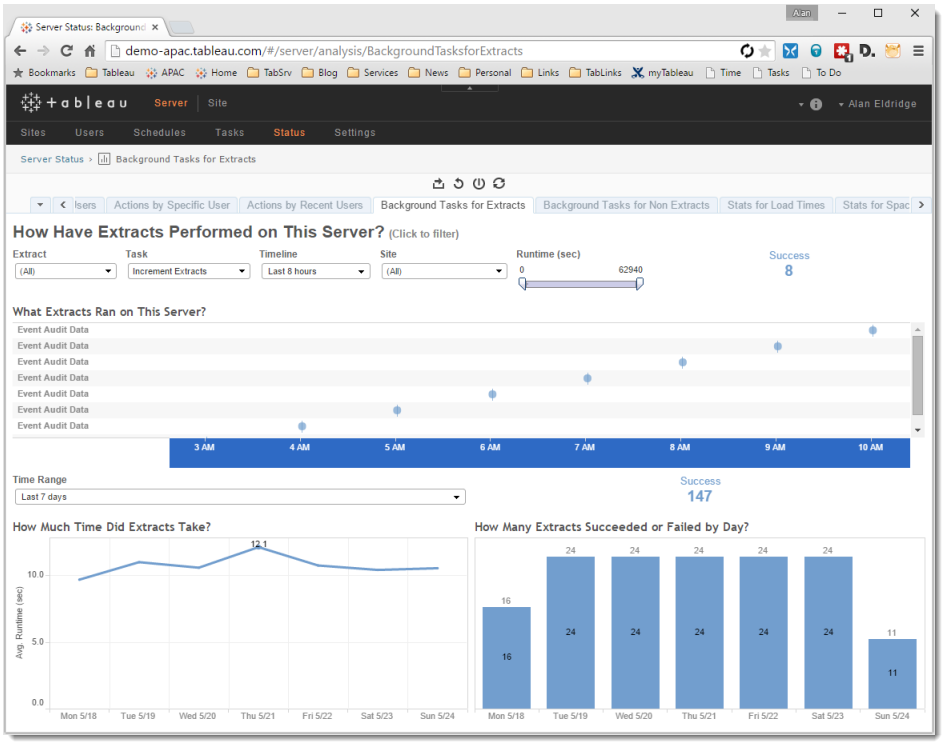


Tableau Server 를 사용하는 경우에는 tabcmd 명령줄 도구를 통해, Tableau Desktop 을 사용하는 경우에는 Tableau.exe 명령줄을 통해 통합 문서를 증분적으로 새로 고치거나 통합 문서 전체를 새로 고칠 수도 있습니다. 일정 요구사항이 복잡한 경우 Windows 작업 스케줄러와 같은 외부 일정 도구에서 이 명령을 호출하십시오. 새로 고침 간격을 Tableau Server 인터페이스에서 허용하는 최소 새로 고침 간격인 15 분보다 더 짧게 설정하려는 경우 이 방식을 사용하시기 바랍니다.

마지막으로 Tableau Online 사용자는 Tableau Online 동기화 클라이언트를 사용하여 Online 서비스에 정의된 일정을 통해 사내 데이터 원본을 최신 상태로 유지할 수 있습니다. 이 유틸리티에 대한 자세한 내용은 다음 페이지를 참조하십시오.

http://onlinehelp.tableau.com/current/online/ko-kr/qs_refresh_local_data.htm

데이터 거버넌스

Tableau Server 에 있는 데이터 원본은 아니지만 Tableau Server 의 Data Server 를 통해 데이터 원본에 연결하는 방법도 있습니다. Data Server 는 라이브 연결뿐만 아니라 데이터 추출 또한 지원하며 독립 실행형 데이터 연결에서는 제공하지 않는 몇 가지 장점을 제공합니다.

- 메타데이터가 Tableau Server 에 중앙 집중식으로 보관되므로 다양한 통합 문서 및 작성자/분석가가 메타데이터를 공유할 수 있습니다. 통합 문서에는 중앙 집중화된 메타데이터 정의로 연결되는 포인터가 보존되며 통합 문서는 실행될 때마다 변경사항이 있는지 여부를 확인하여 변경사항이 있는 경우 사용자에게 통합 문서에 삽입된 사본을 업데이트하라는 메시지를 표시합니다. 이는 다시 말해 비즈니스 논리를 한 번만 변경하면 모든 관련 통합 문서에 변경된 비즈니스 논리를 전파할 수 있다는 것을 의미합니다.
- 데이터 원본이 데이터 추출인 경우 다양한 통합 문서에 걸쳐 이 데이터 원본을 사용할 수 있습니다. 데이터 서버가 없어도 각 통합 문서에는 추출의 로컬 복사본이 포함됩니다. 이렇게 하면 중복된 복사본의 수를 줄일 수 있으며 궁극적으로는 서버에서 필요로 하는 저장 공간뿐만 아니라 중복되는 새로 고침 프로세스도 줄일 수 있습니다.

- 데이터 원본이 라이브 연결인 경우 데이터 원본의 드라이버는 Tableau Server 에만 설치하면 되며 다른 분석가의 PC 에는 설치할 필요가 없습니다. 이 경우 데이터 서버는 Tableau Desktop 에서 전송한 쿼리에 대한 프록시의 역할을 담당합니다.

환경

통합 문서의 성능이 단일 사용자 테스트에서는 뛰어나지만 많은 사용자가 사용할 수 있도록 Tableau Server(또는 Tableau Online)에 배포하면 급격하게 떨어지는 경우가 많습니다. 다음 섹션은 단일 사용자 시나리오와 다중 사용자 시나리오 간의 차이가 성능 간의 차이를 야기할 수 있는 영역에 대해 설명합니다.

업그레이드

Tableau 개발 팀은 소프트웨어의 성능과 유용성을 향상하기 위해 끊임없이 노력하고 있습니다. Tableau Server 를 최신 버전으로 업그레이드하면 통합 문서를 변경하지 않아도 성능과 안정성이 대폭 향상되는 경우가 있습니다.

다음 Tableau 참고 페이지를 확인하고 가능한 최신 빌드로 업그레이드하십시오.

http://www.tableau.com/ko-kr/support/releases

서버에서 Tableau Desktop 테스트

때로는 통합 문서를 워크스테이션의 Tableau Desktop 에서 실행할 때는 잘 수행되는데 Tableau Server 를 통해 볼 때는 느리게 실행되는 경우가 있습니다. Tableau Server 시스템에 설치된 Tableau Desktop 의 복사본으로 통합 문서를 열면 통합 문서에 문제가 있는지 아니면 서버 구성에 문제가 있는지 판단할 수 있습니다. 이 방법으로 드라이버 호환성 문제인지 또는 잘못된 라우팅, DNS, 프록시 구성 등 네트워크 문제인지 확인할 수 있습니다.

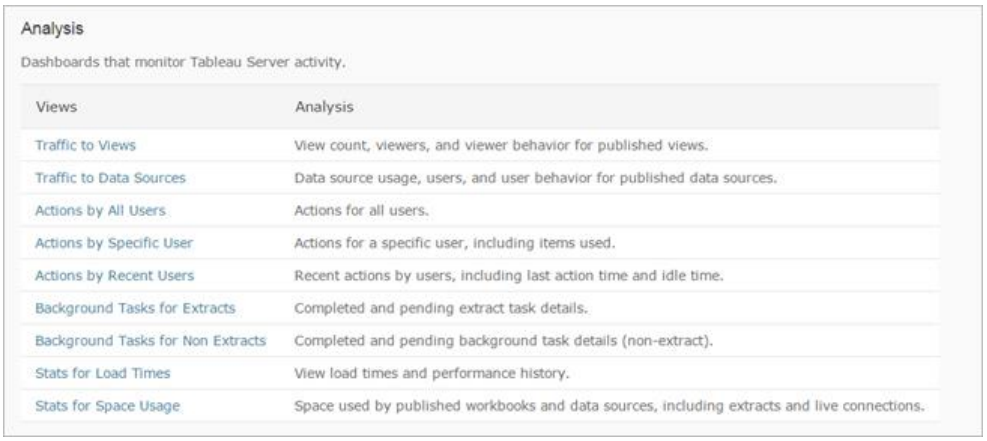
별도 새로 고침 및 상호 작용 워크로드

서버가 느린 경우 백그라운드 작업 관리 뷰를 사용하여 현재 새로 고침 작업 일정을 확인하십시오.

피크 이외 시간에 새로 고침을 예약할 수 있으면 그렇게 하십시오. 하드웨어 구성 상 가능한 경우 백그라운더 프로세스를 전용 작업자 노드로 이동할 수도 있습니다.

서버 모니터링 및 조정

Tableau Server 는 Tableau Server 의 모니터링 작업을 지원하는 다양한 관리자용 뷰를 제공합니다. 이러한 뷰는 서버의 유지 관리 페이지에 있는 분석 테이블에서 확인할 수 있습니다.



이 뷰에 대한 자세한 내용을 확인하려면 다음 링크를 참조하십시오.

http://onlinehelp.tableau.com/current/server/ko-kr/adminview.htm

또한 Tableau 리포지토리를 구성하는 PostgreSQL 데이터베이스에 연결하여 사용자 지정 관리 뷰를 생성할 수도 있습니다. 이에 대한 안내는 다음 페이지를 참조하십시오.

http://onlinehelp.tableau.com/current/server/ko-kr/adminview_postgres.htm

VizQL 세션 시간 제한 확인

기본 VizQL 세션 시간 제한은 30 분입니다. VizQL 세션이 유휴 상태인 경우에도 메모리와 CPU 사이클은 사용됩니다. 시간 제한을 낮출 수 있는 경우 tabadmin 을 사용하여 vizqlserver.session.expiry.timeout 설정을 변경하십시오.

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

프로세스 구성

Tableau Server 는 서비스라고 하는 다양한 구성 요소로 구분되어 있습니다. 기본 구성은 광범위한 시나리오용으로 설계되었지만 기본 구성을 재구성하여 다른 성능 목표를 달성할 수 있습니다. 특히 프로세스를 실행할 컴퓨터와 실행할 프로세스 수를 제어할 수 있습니다. 컴퓨터를 1 대, 2 대 또는 3 대 배포하는 경우에 대한 지침을 확인하려면 서버 성능 개선을 참조하십시오.

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

인프라

64 비트

Tableau 10 이전의 Tableau Server 버전에서는 32 비트 Microsoft 운영체제에서 실행할 수 있지만 고객은 64 비트 버전을 사용하는 프로덕션 환경을 설치하는 것이 좋습니다. 32 비트 버전은 개발 및 테스트에만 설치하여 사용하는 것이 좋습니다.

Tableau 10 이후부터 Tableau Server 는 64 비트 애플리케이션으로만 사용할 수 있습니다.

CPU/RAM 추가

Tableau Server 를 한 컴퓨터에서 실행하든, 여러 컴퓨터에서 실행하든, 가장 중요한 것은 CPU 코어 수와 RAM 이 많을수록 성능이 향상된다는 점입니다. Tableau Server 에서 권장하는 하드웨어 및 소프트웨어 요구사항(http://onlinehelp.tableau.com/current/server/ko-kr/requ.htm#ida4d2bd02-00a8-49fc-9570-bd41140d7b74)을 충족하는지 확인하고 작업자 추가 및 재구성 시기 (http://onlinehelp.tableau.com/current/server/ko-kr/distrib_when.htm#idfdd60003-298e-47e6-8133-3d2490e21e07)를 참조하여 컴퓨터를 더 추가해야 하는지 여부를 평가하십시오.

본 문서의 앞 부분에 언급된 TabMon 은 용량 계획 프로세스를 도와주는 이용 데이터 수집에 뛰어난 도구입니다.

IO 사용

Tableau 의 일부 동작은 고도로 IO 집약적이며(예: 데이터 추출 로드/생성/새로 고침) 회전식 디스크보다 SSD 가 더 적합합니다. Russell Christopher 는 자신의 Tableau Love 블로그에 코어 개수, CPU 속도, IOPS 가 전반적인 성능에 미치는 영향을 확인할 수 있는 몇 가지 유용한 게시물을 올렸습니다. 다음과 같이 실험은 AWS 에서 수행했지만 모든 환경에 적용할 수 있습니다.

> http://bit.ly/1f17oa3
> http://bit.ly/1f17n5O

실제와 가상

이제 많은 고객이 가상화된 인프라에 Tableau Server 를 배포하고 있습니다. 가상화는 항상 성능 오버헤드가 있으므로 가상화되지 않은 시스템에 설치하는 것만큼 빠르지 않지만 최신 하이퍼바이저 기술은 이 오버헤드를 대폭 줄였습니다.

가상 시스템에 설치할 때는 Tableau Server 에 전용 RAM 및 CPU 리소스가 제공되는지 확인하는 것이 중요합니다. 물리적인 호스트에서 다른 가상 시스템과 리소스를 놓고 경쟁하는 경우 성능에 심각한 영향을 줄 수 있습니다.

가상 배포 조정에 대한 추가 정보는 VMWare 의 'vSphere 5.5 에서 극도로 지연 시간에 민감한 애플리케이션 배포' 백서를 참조하십시오. 15 페이지에 지연 시간에 민감한 애플리케이션과 관련된 우수 사례 목록이 제공되어 있습니다.

> http://www.vmware.com/kr.html

브라우저

Tableau 는 자바스크립트를 많이 활용하므로 브라우저 내 자바스크립트 해석기의 속도는 렌더링 속도에 큰 영향을 미칩니다. 최신 브라우저는 빠르게 변화하는 분야이므로 브라우저의 버전과 성능이 환경에 영향을 줄 수 있는지 고려하는 것이 좋습니다.

결론

한 현자께서 본 문서와 관련하여 다음과 같이 조언해 주셨습니다. 즉, 어떤 말을 할 지 이야기하고, 그 말을 한 다음, 이야기한 내용을 다시 말하라는 것입니다. 정말 사려 깊은 조언이라고 생각합니다.

그래서 여기에서 본 문서에서 전달하는 내용을 간추려서 다시 말씀드리겠습니다.

- 묘책은 없습니다. 성능은 여러 가지 이유로 저하될 수 있습니다. 어디에서 시간이 많이 소요되는지 확인하려면 데이터를 수집한 다음 시간이 가장 많이 소요되는 영역을 개선하고 다음 영역으로 이동합니다. 충분히 빠르거나 노력에 따른 보상이 미미한 경우 중단하십시오.
- 속도가 느린 대시보드의 경우 대부분 형편 없는 디자인 때문입니다. 단순하게 만드십시오. 한꺼번에 너무 많은 정보를 표시하려 하지 말고 가능한 경우 안내가 제공된 분석 디자인을 사용하십시오.
- 필요하지 않은 데이터는 사용하지 마십시오. 필터를 사용하고, 사용하지 않는 필드와 집계는 숨깁니다.
- 데이터 엔진을 조정하려고 시도하지 마십시오. 스마트하며 유용한 데이터 엔진이 효율적인 쿼리를 생성할 것임을 신뢰하십시오.
- 데이터가 잘 정리될수록 질문 구조에 일치할 가능성이 높아지고 통합 문서 실행이 빨라지며 작업 만족도가 높아집니다.
- 추출은 대부분의 통합 문서를 더 빨리 실행하는 빠르고 손쉬운 방법입니다. 실시간 데이터가 필요하지 않고 사용하는 데이터 행이 수십억 개 이상이 아니라면 시도해 보십시오.
- 문자열과 날짜는 속도가 느리고 숫자와 부울 값은 빠릅니다.
- 본 문서는 많은 전문가의 조언을 바탕으로 하고 있지만 권장 사항은 권장에 지나지 않습니다. 특정 상황에서 어떤 권장 사항이 성능을 개선할지 직접 테스트해야 합니다.
- 사용하는 데이터 양이 적다면 대부분 중요하지 않습니다. 여러 방법을 동원하여 잘못된 기법을 해결할 수 있습니다. 그렇지만 데이터가 언제 늘어날지 알 수 없으므로 모든 통합 문서에 이 권장 사항을 적용해도 무방합니다.
- 연습하면 완벽해집니다.

보다 많은 통합 문서를 좀 더 효율적으로 직접 작성해 보십시오.