Understanding Tableau's Fast Data Engine

PRESENTED BY

Matthew Eldridge
Richard Wesley

Understanding Tableau’s Fast Data Engine

- What is the Data Engine?
  - Tableau’s purpose-built analytic database for extracts
- What are we going to talk about?
  - Functionality
    - Past, present, and future
    - Tips & tricks
  - Performance
    - Understanding performance on desktop and server
    - Maximizing performance of your extract
- Questions? Please ask!

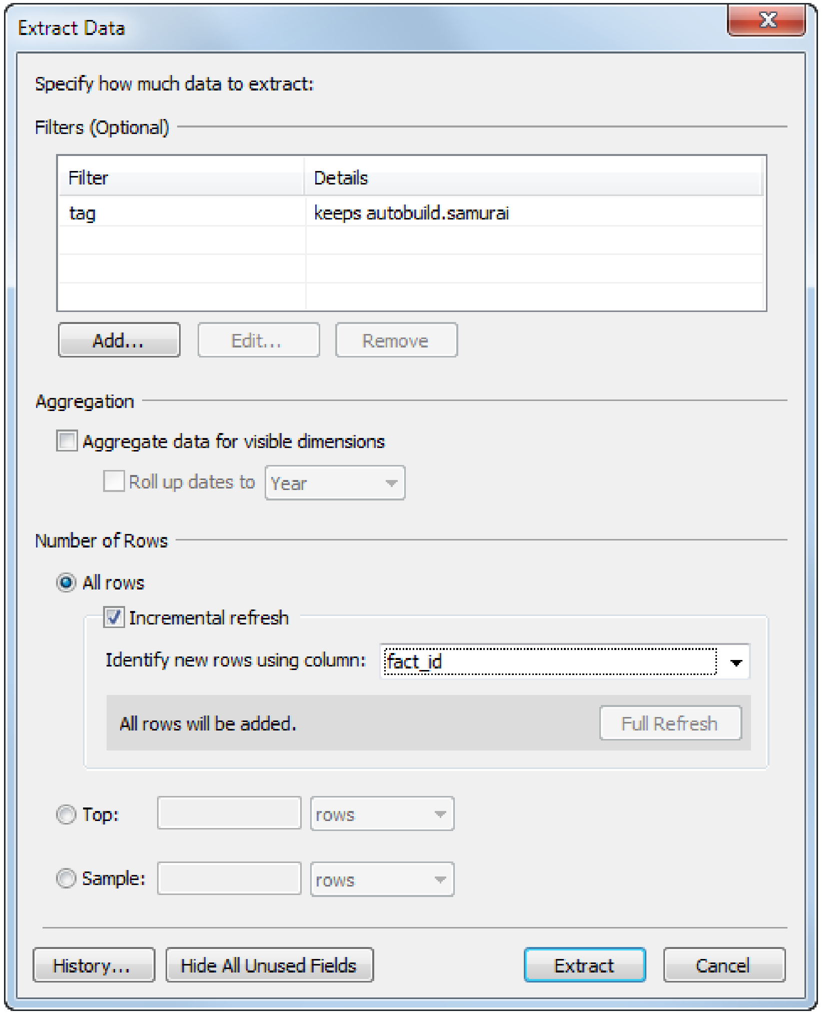
Past: new in 6.0 (October 2010)

- The Data Engine!
  - in-memory analytic database
    - column oriented
    - memory use determined by referenced columns
    - graceful degradation under memory pressure
  - laptop to server scalability
    - 32-bit and 64-bit executables
    - single interchangeable database format
  - ***much*** faster queries

Present: new in 6.1 (June 2011)

- Incremental refresh
  - Add new rows to an existing extract
- Incremental append
  - Add data from a file to an existing extract
- Faster extract creation
  - faster text file parsing
  - faster database query and load
  - faster column compression

Incremental refresh (6.1)



Extract Data

Specify how much data to extract:

Filters (Optional)

Filter	Details
tag	keeps autobuild.samurai

Aggregation

□ Aggregate data for visible dimensions

□ Roll up dates to Year

Number of Rows

● All rows

☑ Incremental refresh

Identify new rows using column: fact_id

All rows will be added.

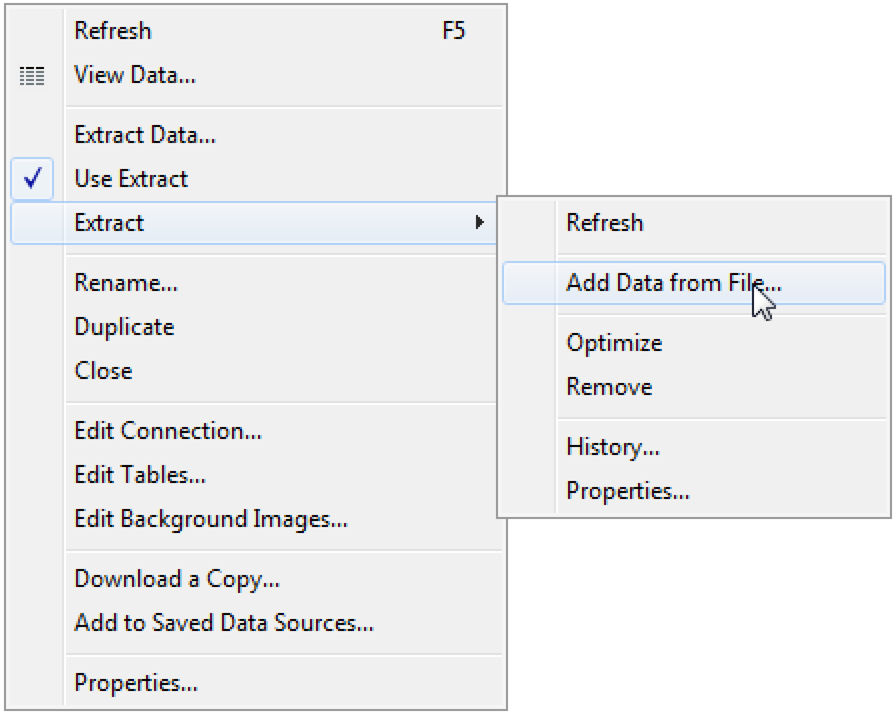
○ Top: rows

○ Sample: rows

- Append new rows from a data warehouse
  - New rows determined via primary key or a time stamp
  - Much faster than a full refresh
- Can be scheduled on server
- Updated or deleted rows

  → full refresh
- Example: performance data from automated nightly tests of Tableau

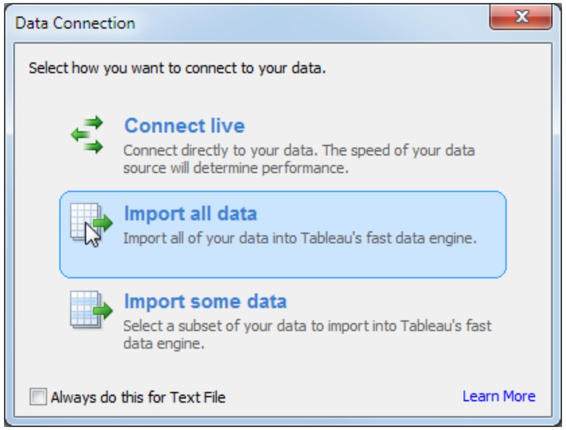
Incremental append (6.1)



- Append new rows from a local file (Access, CSV, Excel)
- Use case: weekly (daily, etc.) drops of new data

Faster extract creation (6.1)

- Fast text parser for CSV data
  - Requires:
    - Single table (no joins, no custom SQL)
    - Import all data (no filters, no aggregation)
  - Much much much faster than Jet
  - Handles files larger than 4GB



Select how you want to connect to your data.

Connect live
Connect directly to your data. The speed of your data source will determine performance.

Import all data
Import all of your data into Tableau's fast data engine.

Import some data
Select a subset of your data to import into Tableau's fast data engine.

☐ Always do this for Text File

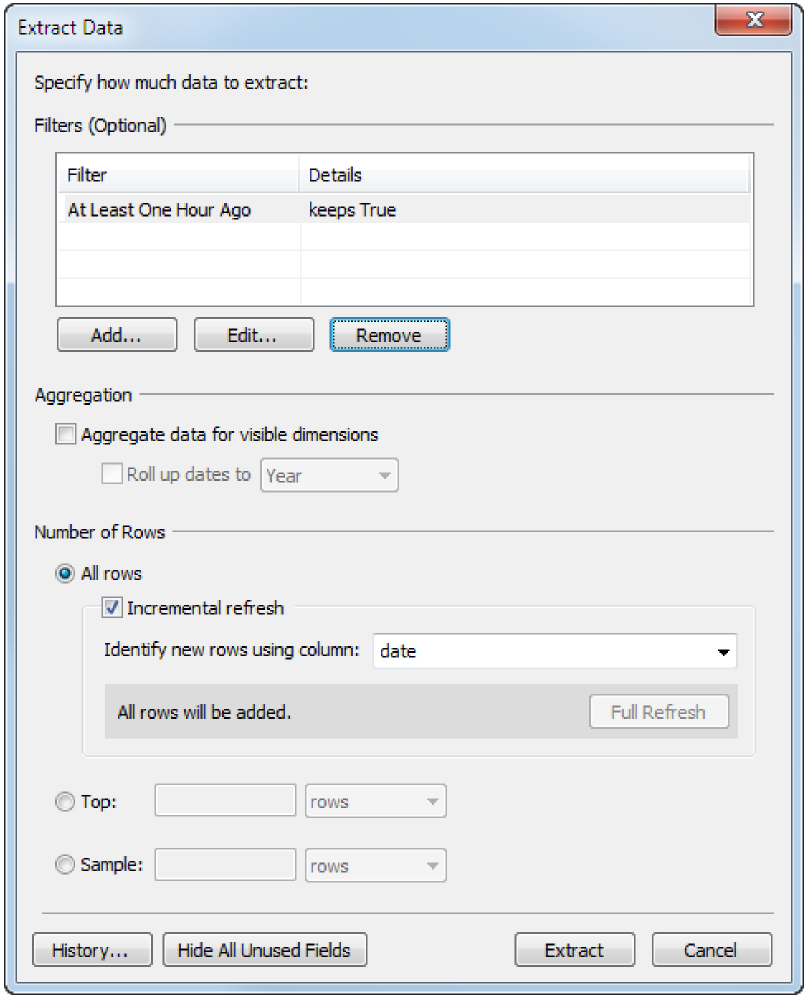
Future: coming in 7.0 (Samurai!)

- Server support for shared extracts
  - Server managed data sources shared across workbooks
    - Extracts remain on Tableau Server
    - Queries executed by Data Engine server
  - Support for scheduled full & incremental refresh
  - Publish a data source to the server from desktop
  - Connect to a published data source from desktop
  - Talk: “Managing Extracts with Tableau Server”

Tips & tricks: Getting the most out of extracts

- Extract filters
- Aggregate data
- Hiding columns
- Raw SQL

Extract filters



Extract Data

Specify how much data to extract:

Filters (Optional)

Filter	Details
At Least One Hour Ago	keeps True

Aggregation

☐ Aggregate data for visible dimensions

☐ Roll up dates to Year

Number of Rows

● All rows

☑ Incremental refresh

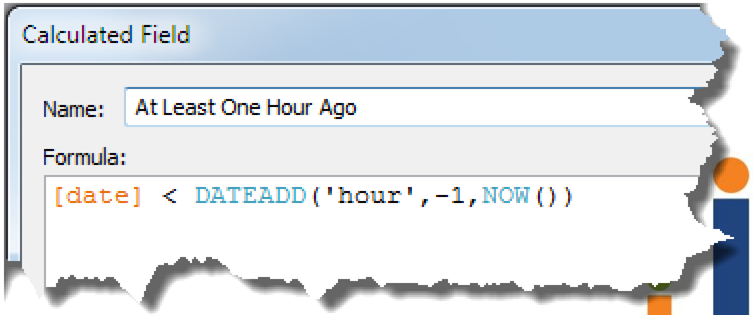
Identify new rows using column: date

All rows will be added.

○ Top: rows

○ Sample: rows

- Filters applied to query of source database
- Limit data to values of interest
- Example: use a date filter together with incremental refresh



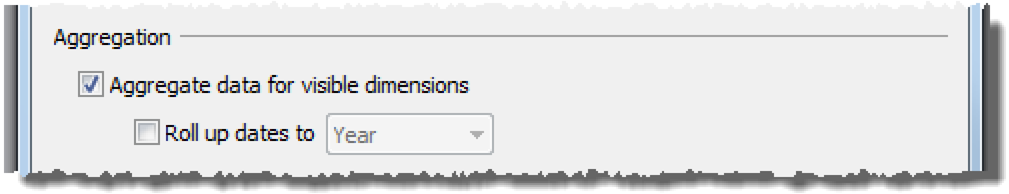
Calculated Field

Name: At Least One Hour Ago

Formula:

```
[date] < DATEADD('hour',-1,NOW())
```

Aggregate data



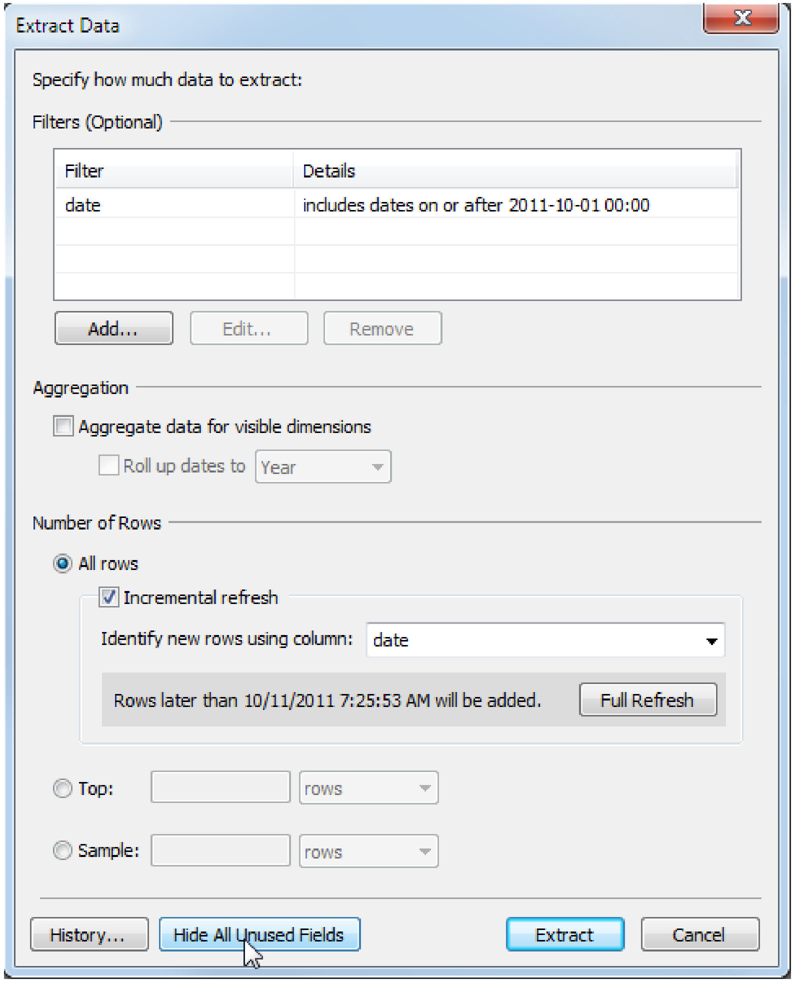
Aggregation

☑ Aggregate data for visible dimensions

□ Roll up dates to Year

- Reduce extracted data to less detailed aggregates
  - Grouped by visible (non-hidden) dimensions
  - Measures aggregated according to default aggregation (SUM)
- Less data →smaller extracts → faster queries!
- Number of Records is the number of source rows in each row of the extract
- Think about your aggregations
  - Average of averages may not be what you expect…
  - … but SUM([column]) / SUM([Number of Records]) is the same as the underlying average!

Hiding columns



Extract Data

Specify how much data to extract:

Filters (Optional)

Filter	Details
date	includes dates on or after 2011-10-01 00:00

Aggregation

□ Aggregate data for visible dimensions

□ Roll up dates to Year

Number of Rows

● All rows

☑ Incremental refresh

Identify new rows using column: date

Rows later than 10/11/2011 7:25:53 AM will be added.

○ Top: rows

○ Sample: rows

- Hidden dimensions and measures are not included in the extract
  - Remove sensitive data
  - Reduce extract size
  - Reduce extract creation time
- One click option
- Unhiding a column will require refreshing extract
- Note: columns in *use* will always be included in the extract, whether or not they are hidden

Raw SQL (7.0)

- Raw SQL calculations materialized when extract created
  - Can be used just like any other column
  - Editing calculation will invalidate it until extract is refreshed
  - Note, aggregate raw SQL calculations are not supported

- Example: convert usernames into opaque identifiers
  - RAWSQL_STR(“md5(%1)”,[username])
  - “eldridge” → "08cfc6800e414c144a850ac10aee8f0d"

Performance

- Is it fast enough? Hooray!
- Data
    - Optimize input types
    - Optimize input data
    - Optimize calculations
- Hardware
    - Memory
    - CPU
    - Disk
- Troubleshooting

Performance: Optimize input types

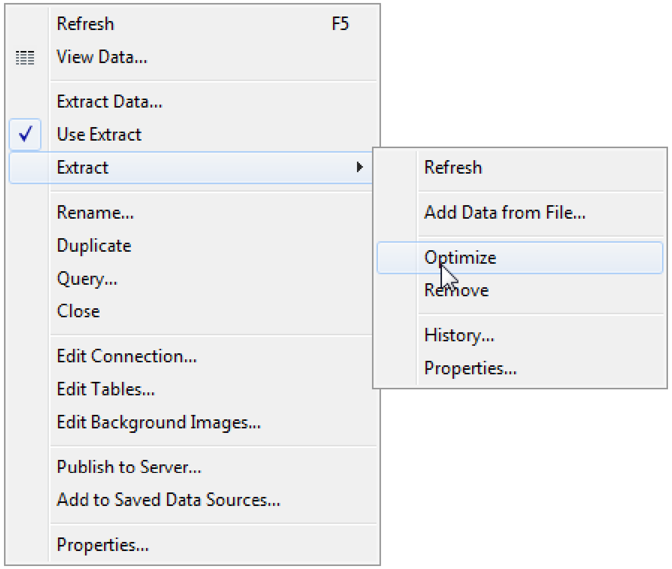
- Clean up data types
  - Dates are smaller than time stamps, etc.
  - Prefer integer to decimal(n,0)
  - However, Data Engine doesn’t care about declared string widths
  - Storage size automatically minimized for integers and reals
    - Don’t check datetime => date
    - Don’t check real => integer
    - Users should still fix these
  - Can improve source database performance as well

Performance: Optimize input data

- Nothing's faster than removing the data ahead of time
  - Reduces extract size and creation time
  - Hide unused columns
  - Extract filters
  - Aggregate extracts
    - Harder than pre-filtering, but can yield huge performance increases

Performance: Optimize calculations

- Pre-computed calculations are faster than ad hoc calcs
  - ➔ Calcs are already evaluated when you use them!
- Optimize command
  - Creates additional columns that materialize your calculations
  - Applies to both measures and dimensions
  - Removes materialized columns for deleted calculations
  - Automatically performed when extract refreshed



- Restrictions on materialized calcs
  - Never applied to aggregate calculations
  - Can only reference fields from datasource
    - No parameters, no secondary datasources
  - Only applied to dimensions in 6.1

Performance: Optimize calculations

- Aggregate calculations cannot be materialized
  - Sometimes can be decomposed
- Example:
  - Calculation for budgeted average selling price, BudgetASP:

    ```
    SUM( [Price] * IF [Market]=“West” THEN 1.2 ELSE 1.0 END )
      / SUM( [ItemCount] )
    ```

  - Becomes
    - BudgetSellingPrice :

      ```
      [Price] * IF [Market]=“West” THEN 1.2 ELSE 1.0 END
      ```

    - BudgetASP :

      ```
      SUM( [BudgetSellingPrice] ) / SUM( [ItemCount] )
      ```

  - BudgetSellingPrice can be materialized, BudgetASP cannot

Performance: Optimize calculations

- Data Engine goes to pains to optimize their evaluation, but nothing is faster than no calculation at all!
- Materialize string calculations
  - Eliminates slower functions:
    - left/mid/right
    - find/contains
    - concatenate (+)
    - casts (converting to other types)
  - Don’t use the database to format data!
- Materialize slow functions
  - Materialize if/case whenever possible

Performance: Optimize calculations

- Write fast calculations
  - When binning, division is faster than if/then/else or case

    -
      ```
      IF [day] < 7 then “Week 1”
      ELSEIF [day] < 14 then “Week 2”
      …
      ```
    - `INT( [day] / 7)`
    - Use aliases to name the bins
  - Date arithmetic is faster than string parsing:
    - `MID(STR([ymd]),4,2)+”/”+RIGHT(STR([ymd]),2)+”/”+…`
    - `DATEADD(‘year’, INT([ymd]/10000), #1900-01-01#)…`
  - These changes work for most databases

Performance: Memory

- How much is enough?
  - Data Engine only reads the columns used in the query

    → extracts larger than memory can remain practical
- If the queried data doesn’t fit in memory, performance will be limited
  - Hard disk is at least 100x slower than memory
- Consider actual usage
  - Desktop: multiple extracts per workbook
  - Server: potentially as many different extracts open as there are active sessions
- 64-bit OS ideal, increased memory on a 32-bit OS can still yield performance benefits

Performance: CPU

- Processor speed
  - Given data that fits in memory, a faster processor will typically result in faster query execution
  - Defining “faster” is tricky: clock frequency, cache size, memory bandwidth, …
- Multiple cores/processors
  - Data engine is single threaded for most operations
  - Extract creation parallelizes sorting
  - Shared data engine in server runs multiple queries in parallel, limited by number of cores

Performance: Disk

- Avoid network volumes
  - Network disks are almost always slower than local disks
  - Significant performance issues for data engine in particular
- Larger disk
  - Intermediate storage during extract creation can be significantly larger than final extract size
  - Put temp directory on a distinct disk from extract storage
- Faster disk
  - Will improve performance in some cases
    - Initial query time
    - Creation of large extracts

Performance: Troubleshooting

- Look out for…
  - Anything that interferes with the disk
    - Windows Search
    - Windows Defender
    - Antivirus software
    - Disk defragmentation (Diskeeper)
  - Anything that interferes with the network software
    - Uncommon, but can corrupt desktop’s and server’s communication with the Data Engine

Performance: Summary

- Fix data types
- Remove unnecessary data
- Leverage materialized calculations
- 64-bit OS & enough memory

Please evaluate this session (TCC11 413)

Understanding Tableau's Fast Data Engine

- Text to **32075**
- In the body of the message, type: **TCC11<space>413**
  then letters from the table below to indicate each response.
- Provide additional comments after an asterisk "*"
- Sample text: **TCC11 413aho*That was great!**

Please give your response to the following:	Excellent	Great	Good	Average	Poor	Bad	Very Bad
What was the value of this session to you?	a	b	c	d	e	f	g
What are the chances you will apply what you learned in this session in your work?	h	i	j	k	l	m	n
What are the chances you would recommend this session to a colleague?	o	p	q	r	s	t	u

Each text evaluation you send enters you into a drawing for an iPad!