



Designing dashboards for performance

Reference deck

Basic principles

1. Everything in moderation
2. If it isn't fast in database, it won't be fast in Tableau
3. If it isn't fast in desktop, it won't be fast in server



Data sources

File-based data sources

Relational data sources

OLAP data sources

Web-based data sources

File-based data sources

Import into Tableau's fast data engine

Microsoft JET driver

- 255 columns, 255 characters
- No COUNT DISTINCT or MEDIAN
- Files greater than 4GB in size

Tableau text parser

- Used when no data modelled

Relational data sources

Indexes

- On all columns part of table JOINS
- On any column used in FILTER

Referential integrity

- Primary and foreign key explicitly defined
 - Helps bypass integrity checks
- Join culling

Partitioning

- Split larger table into smaller, individual tables
- Partition across dimension

Relational data sources

NULLs

- Define dimension as NOT NULL
- Increase effectiveness of indexes

Calculations

- Very complex calculations in a view or function within DB
- Create a custom SQL

Summary tables

- Summarize data to higher level of aggregation

Other data sources

OLAP data sources

- Underlying language differences
 - Metadata definition
 - Filtering
 - Totals and aggregations
 - Data blending

Web-based data sources

- “Connect live” not an option
- Extracts can be refreshed
 - Automated and scheduled



Queries

Understanding the query

Multiple tables vs. custom SQL

Blending vs. joining

Understanding the query

Slow-running visualisation

- Time it takes to query
- Time to stream records back

Number of records

- Large number vs. smaller number of aggregated records

Desktop log files

Multiple tables vs. custom SQL

Multiple tables

- Dynamically create SQL based on fields
- Join culling to drop unused dimension tables
 - Referential integrity

Custom SQL

- Never deconstructed, and executed atomically
- Use in conjunction with Tableau's data engine
- Context filters materialise results in a temp table
- Include parameters in SQL statements

Blending vs. joining

More than one data source

- Blend data or federated database system

Join is better on same data source

- Improve performance
- Improve filtering control

Blend is better across data sources

- Too many records for a join to be practical
- Display summary and details at the same time



Extracts

Creating extracts

Aggregated extracts

Optimizing extracts

Extracts vs. live connections

Creating extracts

Factors

- Database technology
- Network speed
- Data volumes
- Workstation specs
 - Fast CPU with multiple cores
 - Lots of RAM
 - Fast I/O

Creation requires temp disk space

- Up to square of the size of resulting extract file

Create extract on workstation, populate on server

Aggregated extracts

Helps improve performance

- Aggregate to summary level
- Filter unnecessary values
- Hide all unused fields

Multiple level of detail across extracts

- Querying from higher aggregation to detail

Optimizing extracts

Optimize deterministic calculations

- String manipulations and concatenations
- Groups and sets

Non-deterministic calculations not stored

Cheaper to store data than recalculate

Extracts vs. live connections

Speed of data engine is relative

Data engine is faster than

- Non-optimized data base
- File-based data source

Data engine probably slower than

- Big cluster of fast machines

Aggregate extracts to offload summary-style analysis

- Detailed source data in data warehouse

Filtering

Filtering categorical dimensions

Filtering dates

Context filters

Quick filters

User filters

Filtering categorical dimensions

Discrete filters can be slower

- Keep only and exclude perform poor
 - Complex WHERE clause
 - Join on a temp table

Ranged filters can be faster

- Faster than large itemised list of discrete values
- Faster when increasing dimensions cardinality

Indexes impact efficiency of filters

Filtering dates

Discrete dates or date levels

- Can result in poor query execution
 - Tables not partitioned on DATEPART
- Data extract can optimise performance
 - DATEPART materialised in extract

Range of contiguous dates

- Very efficient for query optimisers
 - Leverage indexes and partitions

Relative to a specific date

- Uses ranged date filter

Context filters

All filters are computed independently

Set one or more filters as context filters

- Any other filters are defined as dependent filters
- Writes filter result set to temp table
 - Subsequent filters and queries on smaller dataset

Creation of temp table expensive activity

- Context filters not frequently changed by user

Custom SQL statements can be optimised

Quick filters

Too many quick filters will slow you down

- 'Only Relevant Values'
- Lots of discrete lists

Try guided analytics over many quick filters

- Multiple dashboards with different levels
- Action filters within a dashboard

Quick filters

Enumerated quick filters can be slow

- Requires a query for all potential field values
 - Multiple value list
 - Single value list
 - Compact list
 - Slider
 - Measure filters
 - Ranged date filters

Non-enumerated quick filters can be helpful

- Do not require field values
 - Custom value list
 - Wildcard match
 - Relative date filters
 - Browse period date filters

Performance at the expense of visual context for end user

Quick filters

Show potential values in 3 different ways

- All values in database
 - No need to re-query when other filters changed
- All values in context
 - Temp table regardless of other filters
- Only relevant values
 - Other filters are considered

Quick filter alternatives

Create a parameter and filter based on users' selection

- PROS
 - Do not require a query before rendering
 - Parameters + calculated fields = complex logic
 - Can be used to filter across data sources
- CONS
 - Single-value selection only
 - Selection list cannot be dynamic

Quick filter alternatives

Use filter actions between views

- PROS
 - Supports multi-value selection
 - Evaluated at run-time to show a dynamic list
 - Can be used to filter across data sources
- CONS
 - Filter actions are more complex to set up
 - Not the same UI as parameters or quick filters
 - Source sheet still needs to query the data source

User filters

More data source I/O

- Need to ask exact same query again

More cache space required

- Each user session creates own query results and model cache

Caches being cleared can result in more I/O



Calculations

Basic and aggregate vs. table calculations

Calculations vs. native features

Performance techniques

Basic and aggregate calculations

Basic and aggregate calculations

- Expressed as part of the query sent to data source
- Calculated by the database
- Basic calculations scale very well
 - Tuning techniques can improve performance

Table calculations

- Calculated locally on query results returned
 - Generally done over a smaller set of records
- If performance is slow...
 - Then push calculations to data source layer
 - Consider aggregated data extracts

Calculations vs. native features

Native features often more efficient than a manual calculation

- Grouping dimension members together
 - Consider using groups
- Grouping measure values together into 'bins'
 - Consider using bins
- Changing displayed values for dimension members
 - Consider using aliases

Performance techniques

Data type used has a significant impact on performance

- Integers are faster than Booleans
- Both are faster than Strings

Use Booleans for basic logic calculations

- Bad
 - IF [DATE]= TODAY() THEN “TODAY” ELSE “NOT TODAY” END
- Good
 - [DATE]=TODAY()

String searches

- FIND() slower than CONTAINS()
- CONTAINS() slower than wildcard match quick filter

Performance techniques

Parameters for conditional calculations

- Take advantage of 'display as'
 - Integer values for calculation logic

VALUE	DISPLAY AS
YEAR	YEAR
QUARTER	QUARTER
MONTH	MONTH
WEEK	WEEK
DAY	DAY

VALUE	DISPLAY AS
1	YEAR
2	QUARTER
3	MONTH
4	WEEK
5	DAY

Performance techniques

Date conversion

- Numeric field to a string to a date is inefficient
 - Bad
 - `DATE(LEFT(STR([YYYYMMDD]),4) + "-" + MID(STR([YYYYMMDD]),4,2) + "-" + RIGHT(STR([YYYYMMDD]),2))`
- Keep numeric field and use DATEADD()
 - Good
 - `DATEADD('DAY', [YYYYMMDD]%100-1, DATEADD('MONTH', INT(([YYYYMMDD]%10000)/100)-1, DATEADD('YEAR', INT([YYYYMMDD]/10000)-1900, #1900-01-01#)))`

Date functions

- NOW() for time stamp
- TODAY() for date level

Performance techniques

Logic statements

- ELSEIF != ELSE IF

```
IF [REGION] = "EAST" AND [CUSTOMER SEGMENT] = "CONSUMER"  
THEN "EAST-CONSUMER"  
ELSE IF [REGION] = "EAST" AND [CUSTOMER SEGMENT] <>"CONSUMER"  
THEN "EAST-ALL OTHERS"  
END  
END
```

- would run much faster as:

```
IF [REGION] = "EAST" AND [CUSTOMER SEGMENT] = "CONSUMER"  
THEN "EAST-CONSUMER"  
ELSEIF [REGION] = "EAST" AND [CUSTOMER SEGMENT] <>"CONSUMER"  
THEN "EAST-ALL OTHERS"  
END
```

- but this is faster still:

```
IF [REGION] = "EAST" THEN  
IF [CUSTOMER SEGMENT] = "CONSUMER" THEN  
"EAST-CONSUMER"  
ELSE "EAST-ALL OTHERS"  
END  
END
```

Performance techniques

Separate basic and aggregate calculations

- When using extracts and custom aggregations
 - Divide calculations into multiple parts
 - Row level calculations on one calculated field
 - Aggregated calculations on second calculated field

Dashboards and views

Views

Dashboards



Views

Only fetch and draw what you need

- Remove unnecessary fields from level of detail

Charts vs. crosstabs

- Marks display faster than a tabular report
 - Rendering a text table consumes more memory
- Tableau is not a back door data extract process
 - Leverage aggregate to detail action filters

Removing unnecessary geographic roles

- Save time to lookup generated latitudes & longitudes

Views

Blending vs. custom geographic roles

- Custom geocoding embeds entire GEOCODING.FDB
 - Significant increase in TWBX file size
- Joining or blending geographic data is smaller size
 - Save only relevant custom geographic data

Dashboards

Less views, less quick filters

- Each view requires at least one query
- Each quick filter requires at least one query
- Can result into a lot of I/O before rendering
- Dashboards process views from same data source in a serial fashion

Turn off tabs

- Must process every view in every tab
 - Need to understand structure of tabs for actions or filters
- Reducing tabs improves performance
 - Try `':tabs=no'`

Dashboards

“Exclude all values”

- Avoids expensive query of asking for all data

Fixed size dashboards

- Different window sizes mean views are drawn differently
 - VizQL server must render view separately for each user
- “Automatic (Fit To Window)” has low cache hit rate

Container considerations

- Consider reducing excessive usage of containers



Tools to analyze performance

Tableau desktop messages

Log files

Database performance monitors

Tableau 8 performance metrics

Tableau desktop messages

Executing query

- Process
 - Execute a query to return records for the view
- Investigate
 - Review log file to see queries taking long time
- Possible solution
 - Consider calculation, query and filter techniques

Tableau desktop messages

Computing View Layout

- Process
 - Tableau rendering display on all data received
- Investigate
 - Slow-running table calculations
 - Very large crosstab
 - Lots of marks rendered
- Possible solution
 - Review techniques for calculation optimisation and view design

Tableau desktop messages

Computing quick filters

- Process
 - Tableau is processing quick filters for view(s)
- Investigate
 - Long time rendering and refreshing quick filters
- Possible solution
 - Consider not using 'show relevant values'
 - Consider not using enumerated quick filters
 - Review techniques for filter performance and view design

Log files

Understand where bottlenecks are occurring

- Information on what Tableau is doing
- Tableau communication with the data source
- Time taken by each Tableau step

For Tableau Desktop

- C:\Users\username\Documents\My Tableau Repository\Logs

For Tableau Server, the VizQL log file:

- C:\ProgramData\Tableau\Tableau Server\data\tabsvc\vizqlserver\Logs

Database performance monitors

Database performance monitors

- Insight on queries hitting your DB
- Understand how database processes them
- Advice on additional tuning

Tableau 8 performance metrics

- Turn on to record metrics

Performance metrics V8

Interpret performance recording

- Timeline
 - Workbook, dashboard, worksheet
- Events
 - Event nature and duration
- Query
 - Executing query in either timeline or events

Performance metrics events

Computing layouts

- If layouts are taking too long, consider simplifying your workbook.

Connecting to data source

- Slow connections could be due to network issues or issues with the database server.

Executing query

- If queries are taking too long, consult your database server's documentation.

Generating extract

- To speed up extract generation, consider only importing some data from the original data source.

Geocoding

- To speed up geocoding performance, try using less data or filtering out data.

Blending data

- To speed up data blending, try using less data or filtering out data.

Server rendering

- You can speed up server rendering by running additional VizQL Server processes on additional machines.

Tableau server

General performance guidelines

Caching



General guidelines

General guidelines

- Use a 64-bit operating system
 - Ensures 64-bit version of Tableau data engine
 - 32-bit processes have access to more memory
- Add more cores and memory

Configuration

- Schedule extract refreshes for off-peak hours
- Check the VizQL session timeout limit
 - Default is 30 minutes
 - Idle session still consumes memory
 - Change using tabadmin
 - `vizqlserver.session.expiry.timeout.setting`
- Assess your process configuration

Caching

Maximise cache use

- Reuse image tile and model caches
 - Set dashboard size rule to 'exact size'

Tuning caches

- Tableau server configuration utility
 - Minimise queries
 - Balanced
 - Most up-to-date
 - » “:refresh=yes” to view URL
- Tabadmin
 - Model cache
 - » vizqlserver.modelcachesize:30
 - Query cache
 - » vizqlserver.querycachesize:64

FIN

