Index Strategies for SQL Server Performance
Part 2: Performance Tips for Devs

Kevin Kline, Head Geek, SolarWinds

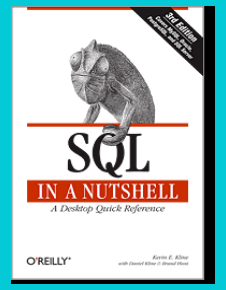
Thomas LaRock, Head Geek, SolarWinds

Kevin Kline

Head Geek™

Kevin has over 30 years experience in roles including programmer, data scientist, DBA, enterprise architect, and manager. He enjoys traveling, writing, teaching, and indie rock.

database TRENDS AND APPLICATIONS



MVP Microsoft Most Valuable Professional

facebook.com/kekline

twitter.com/kekline

linkedin.com/in/kekline/

instagram.com/kevin_e_kline

Thomas LaRock

Head Geek

Thomas has over 20 years experience in roles including programmer, developer, analyst, and DBA. He enjoys working with data, probably too much to be healthy, really.

Microsoft CERTIFIED Master

vmware vEXPERT

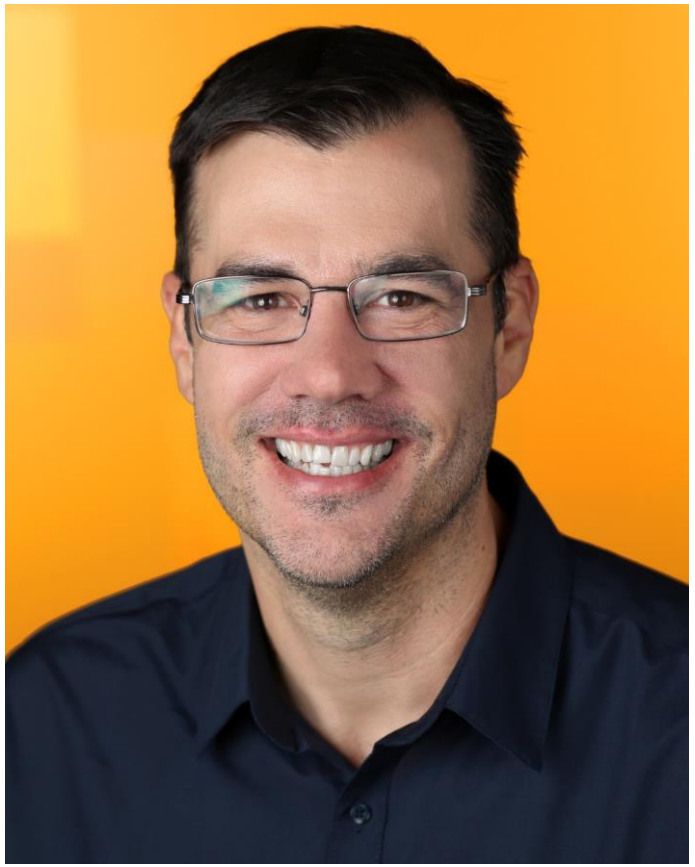
MVP Microsoft Most Valuable Professional

facebook.com/thomas.larock

twitter.com/SQLRockstar

linkedin.com/in/sqlrockstar/

instagram.com/sqlrockstar



Index Structures and Fundamentals

- Index structures and fundamentals
  - Indexes and statistics for the optimizer
  - Statistics, histograms, cardinality, and other metadata
- Writing Code that helps SQL Server choose indexes

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

POLL

Would you like to learn more?

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

Make It Easy for the Optimizer

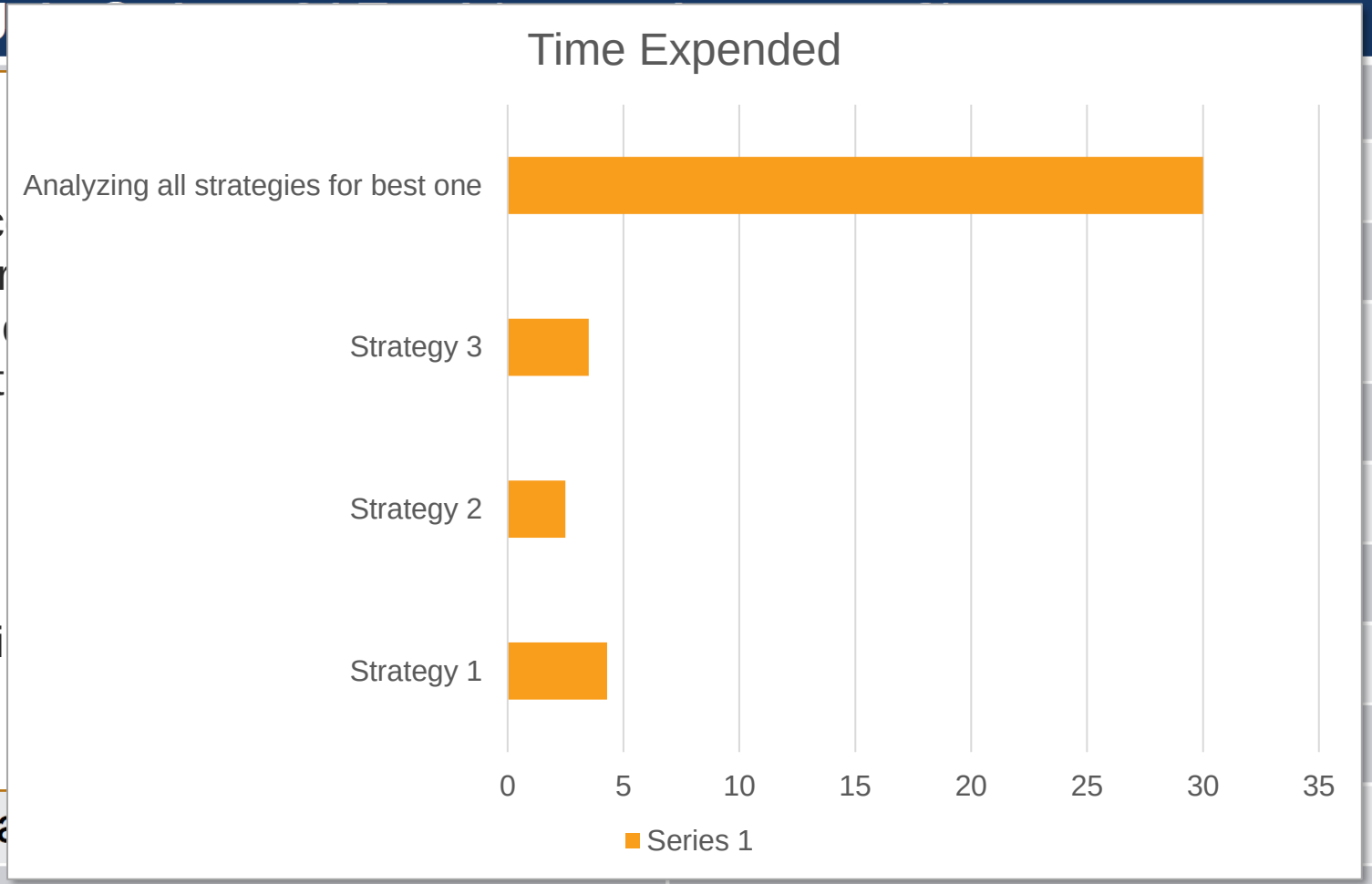
- 72 possible physical data acc
- 120 possible logical join order
- 3 physical joins possible per l
- May require intermediate sort
- = **25,920 possible plans**

- The more complex the query,

- Optimizer uses heuristics to li

methods

Logical J	
b⋈c⋈d⋈a	
b⋈d⋈a⋈c	d⋈c⋈a⋈b
b⋈d⋈c⋈a	d⋈c⋈b⋈a



What Is a Statistic?

An object that stores information about values; usually, but not always, associated with indexes

Cardinality tells us how many rows meet a given criteria

- Many operators have predicates – filter or join criteria
- New cardinality estimator in SQL Server 2014

Selectivity refers to % of rows that meet criteria

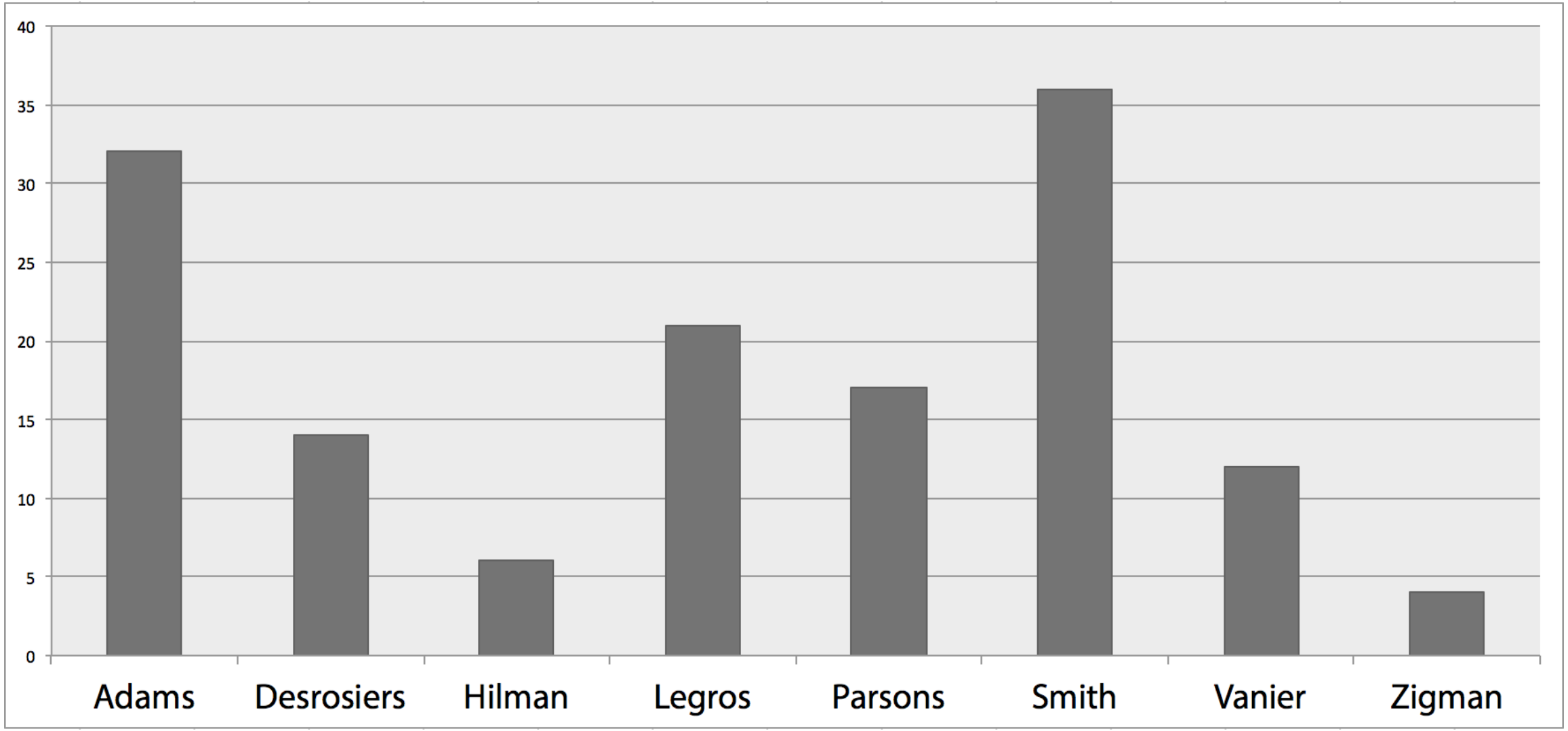
- Natural key is very selective; gender isn't very selective at all

Density refers to uniqueness of values

Histogram keeps distribution information in <= 200 steps

- Number of values in each range
- Like a bell curve, but usually less smooth

Statistics Histogram – Simple Illustration



How Statistics Help

They can help the optimizer:

- Make accurate estimates and memory grant requests
- Decide on the right indexes
- Decide between a seek or a scan
- Decide on join ordering and join algorithm
- Decide between serial and parallel execution

Bad (out-of-date) statistics tend to lead to bad execution plans with wrong answers on any or all the above decisions

Keep statistics up to date, especially on larger/volatile tables

DEMO

Index statistics in action

© 2020 SolarWinds Worldwide, LLC. All rights reserved.

Statistics Quality

Stats are only used on leading column of index

- Therefore, leading key column should be selective and unique-ish

Stats updated less often as table gets bigger

- Ascending key problem – new values outside of histogram
- Filtered statistics problem
- Data growth problem – new buckets are bigger
- **Stats must be maintained**

2014 has a whole new cardinality estimator

- Can lead to many improvements but a few bad regressions
- Many options for turning this behavior on/off
- Incremental statistics: reduce maintenance, but partition-level stats are ignored by optimizer

Tips From Optimizing Index I/O

- Minimize the number of pages you need to read
  - Narrower indexes, narrower columns, compression, columnstore
- Minimize output columns to hopefully hit non-clustered and avoid lookups
- Keep statistics up to date (so memory grants are right)
  - Otherwise tempdb may be required to help sorts and joins
- Avoid sorting by columns not supported by indexes
  - Adding memory can keep bigger indexes in cache, but can’t help sort
- Minimize the number of indexes?
  - One wide index may be better than two skinny indexes

Important Index-related Query Behaviors

1. Scans versus Seeks and the “tipping point”
2. SQL code habits that eliminate a perfectly useful index
3. Covering indexes for the win
4. Concatenated indexes – handle with care

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

Seek vs. Scan

Seeks are used for single-row or range scans

- Seek really means “where does the scan start”

Scans are generally used to look at the whole index/table

- Scan count in stats I/O doesn’t necessarily mean full scan count
- e.g., a parallel query or partitioned query has multiple partial scans

Generally, a seek is better, but not universally true

- Depending on index structure, a scan can be better
- And an NC index scan can be better than a CIX/table scan
- Don’t blindly optimize for seeks, or assume that a scan is bad news

“Tipping Point”

Generally, if seek estimated to hit > 25-30% of pages, scan is better

Lookups needed with seek on non-covering indexes

- A lookup is executed for every row

Again, generally, tipping point looks like this:

- Decision can be guided by index coverage, row size, cost of lookups, hardware
- Often the tipping point will be much lower

http://www.sqlpassion.at/archive/2013/06/12/sql-server-tipping-games-why-non-clustered-indexes-are-just-ignored/

Sargability

Surprising Reasons Why Indexes Might Be Unused

The ability to actually *use* the index

Beware of applying functions against JOIN/WHERE columns

- WHERE DATEDIFF(DAY, Column, GETDATE()) = 0
  - Not sargable – will scan – converting to a string is even worse
- WHERE Column >= @today AND Column < DATEADD(DAY, 1, @today)
  - Sargable

WHERE CONVERT(DATE, Column) = @today

- Can be sargable, but there's a catch – bad estimates – see link in notes

ORDER BY key column(s) won't use index if it doesn't cover

Functions and Calculations Limit Indexes

Avoid using functions or calculations on the column in a WHERE clause because it causes SQL Server to limit the usability of any index on the column:

- WHERE qty * 12 > 10000
- WHERE ISNULL(ord_date, ‘Jan 01,2011’) > ‘Jan 01, 2012 12:00:00 AM’

Instead, move the function or calculation to the right side of the search argument:

- WHERE qty > 10000/12
- WHERE ord_date IS NOT NULL
- AND ord_date > ‘Jan 01, 2002 12:00:00 AM’

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

Covering Index

- Queries may filter on LastName, but return more columns
- Can be useful to add those other columns to the index
  - Don’t do this blindly; Bigger indexes = costlier DML maintenance
- Used to avoid lookups into heap or clustered index
  - Lookups are expensive – 1 read per row, more with LOBs
- In the key, put most selective columns first
  - Think about an index as eliminating rows, not finding rows

Covering Indexes – The Inverse of Concatenated Indexes

An index that covers the entire data requested by the query

- A covering index answers a query entirely from its own intermediate pages, rather than going all the way down the tree to leaf pages.
- All columns referenced in the query need to be in the index. For example:

```
SELECT  lname, fname
FROM    employee
WHERE   lname LIKE '%ttlieb%'
```

- All columns in the index must be referenced in the query:
  - CREATE INDEX employee_name_lvl ON employee (lname, fname)
- Don’t confuse a covering index with an included columns index

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

Performance Penalty for Concatenated Index

- Don’t forget the clustered index is stored in EVERY non-clustered index! The shorter the clustered index the less the storage and fewer I/Os.
- Remember, composite keys are most useful from the leftmost column to the rightmost column, in the order they appeared in the CREATE INDEX statement.
- Consider this index:
  - ALTER TABLE yellow_pages ADD CONSTRAINT [UPKCL_sales] PRIMARY KEY CLUSTERED ([lname], [fname], [street])

LName	FName	Street
• Austin • Hanso • Linus	• Kate • Magnus • Benjamin	• 1010 Madison • 315 Ajira • 815 Oceanic

What Happens When Querying a Concatenated Index

Works well with:

- WHERE lname = @a AND fname = @b AND street = @c

Also works well with:

- WHERE lname = @a AND fname = @b

How does the index work with this?

- WHERE fname = @b OR title_id = @c

Best practice? Use a surrogate key for the primary key and then a natural key as an additional non-clustered index.

DEMO

SQL code that can sabotage index usage

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

Helping SQL Server to Choose the Best Indexes

Index Selection

The query optimizer will attempt to select indexes based on what it knows at the time the transaction is issued:

- Evaluate filter conditions on each table
- Satisfy joins
- Find columns without going to the leaf pages (covering indexes)

Even though the query optimizer will (usually) find a good enough plan very quickly, there are lots of things you can do to help it find the best plan.

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

Join Transformations Depend on Indexes

- SQL Server applies associative and commutative heuristics testing for the optimal join order
- Six joins (also, subqueries) or more often means performance hurdles
- More joins magnifies the complexity for the query optimizer. Note: a 12-table join has more than 28 trillion possible join orders
- Maxes allowed: 32 tables in inner joins, 2 tables in outer joins
- Not just joins are counted – also included are APPLY, subqueries, and set operators like INTERSECT and MINUS

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

Issues With “OR” Clauses

- Queries containing an OR clause are effectively the same as two separate queries using a UNION statement.
- Each OR condition requires an index to avoid a table scan.
- Evaluate using a UNION statement, so at least some portion of your query can perform an index seek.

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

Issues With “LIKE” Clauses

Queries in prod systems should NOT use SELECT * FROM…

- Any time the underlying table is changed, all query plans stored in the cache must be rebuilt.
- The SQL tools have Intellisense – so no excuses!

Queries that use the LIKE clause have two simple rules:

- LIKE can use indexes if the pattern starts with a character string, such as WHERE lname LIKE “w%”
- LIKE cannot use an index if the pattern starts with a leading wildcard, such as WHERE lname LIKE “%alton”

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

DEMO

Issues with OR and LIKE in SQL code

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

Query Strategy Review

- Limit the columns requested by restricting the asterisk wildcard.
- Limit the number of rows returned through search arguments.
- Use covering indexes to answer queries with few columns in the result set.
- Know the impact of join order, composite keys, and OR/LIKE in WHERE clauses.
- Keep up with functions and calculations in your WHERE clause.

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

POLL

Would you like to learn more?

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

Q&A

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

solarwinds.com/DPA

© 2021 SolarWinds Worldwide, LLC. All rights reserved.

The SolarWinds, SolarWinds & Design, Orion, and THWACK trademarks are the exclusive property of SolarWinds Worldwide, LLC or its affiliates, are registered with the U.S. Patent and Trademark Office, and may be registered or pending registration in other countries. All other SolarWinds trademarks, service marks, and logos may be common law marks or are registered or pending registration. All other trademarks mentioned herein are used for identification purposes only and are trademarks of (and may be registered trademarks) of their respective companies.

© 2021 SolarWinds Worldwide, LLC. All rights reserved.