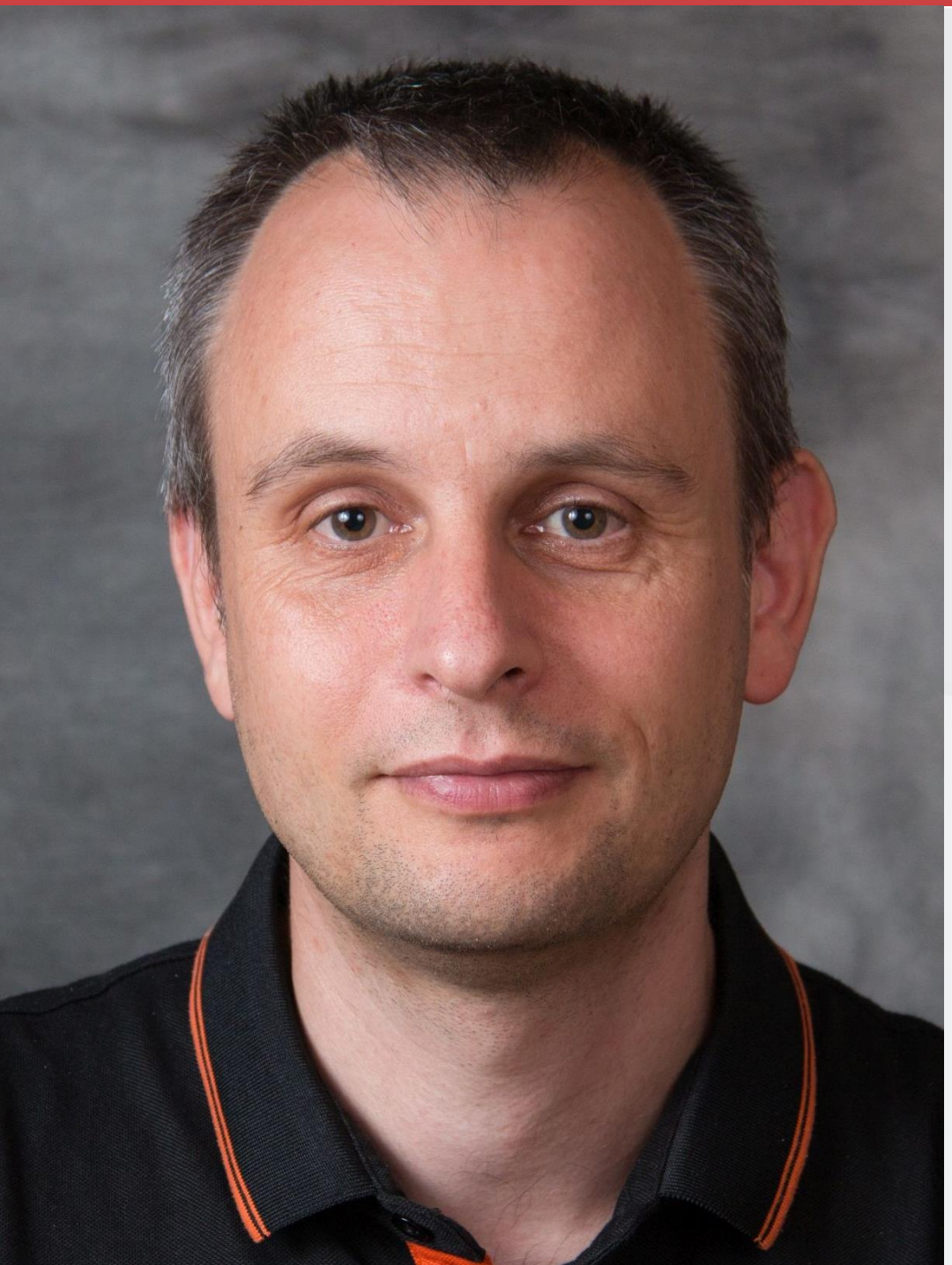
ENTERPRISE SOFTWARE DEVELOPMENT

Top Database Performance Techniques

We Are Developers World Congress 2024

Join the Conversation #DatabasePerformance @BrydenOliver



Bryden Oliver

SSW Solution Architect

linkedin.com/in/brydenoliver

github.com/brydeno

- 20 years experience in database performance
- 42 years of coding
- Worked with Azure since its launch
- Knows about Azure at scale

Find the presentation and demos at
github.com/brydeno/DatabasePerformance

What are we covering

Improving query performance

Lots of demos

Simple techniques

Where possible, we'll measure the improvement

Useful Tricks

It's always useful when running locally to get query statistics

Run SET STATISTICS IO ON;

Before you run your query

We'll see this in the demos ☺

There's loads more little things littered through the demos

Example database

StackOverflow 2013 database

1. Reduce Table Size

WHY?

- Large tables take longer to read and update
- They take up more buffer space

1. Reduce Table Size

WHAT TO DO?

- Archive unneeded data
- Reduce column sizes
- Remove unnecessary columns

01 - KeepTablesAsSmallAsPossible.sql - DESKTOP.StackOverflow2013 (DESKTOP\bryde (54)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query | StackOverflow2013 | Execute

01 - KeepTablesAsS...ESKTOP\bryde (54))

```
-- So there are 2 tables in this example
-- Votes contains all the votes
-- Votes1000 contains only 1000 votes
-- This illustrates the improvement you can get by
-- archiving out old data. Maybe to separate archive tables,
-- or out to data warehouses / data lakes

SET STATISTICS IO ON;

SELECT * FROM dbo.votes1000
SELECT * FROM dbo.votes

-- So that first query came back in under 1 second. We're
-- still waiting on the second query. Ok, so that kind of proves
-- the point. We won't wait for this, so take my word for it,
-- the stats are 7 reads for the small table and
-- a quarter of a million reads for the large and
-- almost 4 minutes for the large one.
-- Other advantages are that we haven't trashed the buffer cache
-- when we selected 1000 rows, but the other query, our buffer
-- cache pretty much has just votes data now :(
```

200 %

Connected. (1/1) | DESKTOP (16.0 RTM) | DESKTOP\bryde (54) | StackOverflow2013 | 00:00:00 | 0 rows

Ready | Ln 1 | Col 1 | Ch 1 | INS

2. Use SELECT <fields> FROM rather than SELECT * FROM

- Retrieving all columns can cause both index and table to be read
- Takes more network to transmit
- More CPU work at both ends

02 - SelectOnlyRequiredColumns.sql - DESKTOP.StackOverflow2013 (DESKTOP\bryde (56))* - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query | StackOverflow2013 | Execute

02 - SelectOnlyReq...ESKTOP\bryde (56))*

```sql
-- So this one is a bit of a contrived example but it gives you the idea
-- First we'll create an index, this multiplies the effect of only selecting the columns we want

CREATE NONCLUSTERED INDEX [IX_DisplayName_Location] ON [dbo].[Users]
(
    [DisplayName] ASC,
    [Location] ASC
)
GO

SET STATISTICS IO ON;

SELECT * FROM dbo.Users

SELECT DisplayName,Location FROM dbo.Users

-- So when we compare the performance, we see there are just over 3 times as many reads
-- required because it was able to read from the index only.
```

Messages

```
Commands completed successfully.

Completion time: 2024-06-02T10:34:22.8576485+10:00
```

Query executed successfully. | DESKTOP (16.0 RTM) | DESKTOP\bryde (56) | StackOverflow2013 | 00:00:00 | 0 rows

Ready | Ln 1 | Col 1 | Ch 1 | INS

3. Create indexes

- Generally gets the biggest wins
- Takes some effort
- Can make more than 1,000,000 times improvements
- github.com/brydeno/DatabasePerformance

4. Verify indexes are being used

(Check for unused indexes)

- You'd be surprised how often they aren't
- Run your query with "Show Actual Execution Plan" on
- Check the plan is what you expect (or better)

DEMO

Execution plan

Unused index query

04 - IdentifyingUnusedIndexes.sql - DESKTOP.StackOverflow2013 (DESKTOP\bryde (53)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query | StackOverflow2013 | Execute

04 - IdentifyingUn...DESKTOP\bryde (53))

```sql
-- Let's run this query and just verify using an execution plan that it uses an index.

SELECT * from dbo.Posts
WHERE OwnerUserId = 22

-- That's great from SSMS or similar, but how could we tell if our aplication uses the index. We need to work a little bit ha

-- Store this query up to use when you need it.
-- Best to run just before your workload, then straight after
-- The numbers should change :)

SELECT
    object_name(s.[object_id]) AS TableName,
    i.name AS IndexName,
    i.index_id AS IndexID,
    us.user_seeks AS UserSeeks,
    us.user_scans AS UserScans,
    us.user_lookups AS UserLookups,
    us.user_updates AS UserUpdates
FROM
    sys.dm_db_index_usage_stats us
INNER JOIN
    sys.indexes i ON (us.object_id = i.object_id AND us.index_id = i.index_id)
INNER JOIN
    sys.objects s ON us.object_id = s.object_id
```

200 %

Connected. (1/1) | DESKTOP (16.0 RTM) | DESKTOP\bryde (53) | StackOverflow2013 | 00:00:00 | 0 rows

Ready | Ln 4 | Col 23 | Ch 23 | INS

5. Avoid Implicit conversions

Make sure the type in **WHERE** clause matches column

Can make SQL Server scan a whole index 😱

05 - AvoidImplicitConversions.sql - DESKTOP.StackOverflow2013 (DESKTOP\bryde (53)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query | StackOverflow2013 | Execute

05 - AvoidImplicit...DESKTOP\bryde (53))

```
-- So here we are showing what happens with a string column,
-- where we use an integer as the where clause in the second instance.
-- The second instance needs to scan the whole table and convert postid
-- to an integer. Then it compares it to the value.
-- So in this example we do 10,000 times less work.


-- The plan tells us why. The first query just does an index seek
-- straight to the data, the second one scans the whole index
-- to find the relevant rows.

SET STATISTICS IO ON;
SELECT * FROM dbo.VotesString WHERE PostId = '9999997'
SELECT * FROM dbo.VotesString WHERE PostId = 9999997
```

200 %

Connected. (1/1) | DESKTOP (16.0 RTM) | DESKTOP\bryde (53) | StackOverflow2013 | 00:00:00 | 0 rows

Ready | Ln 1 | Col 1 | Ch 1 | INS

6. Avoid Looping

Looping causes the server to execute one after another

If you can get it to do in parallel much better

Often **GROUP BY** or aggregation a better choice

There are times looping is necessary

Looping database statistics is one (very few others)

06 - AvoidLooping.sql - DESKTOP.StackOverflow2013 (DESKTOP\bryde (76))* - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query | StackOverflow2013 | Execute

06 - AvoidLooping....ESKTOP\bryde (76))*

```sql
-- Note optimising to avoid looping generally simplifies the SQL too
-- That's especially true for this example
-- The query is meant to calculate the average number of answers per post.
-- The first example takes a long time, so I've capped it at 100 rows. Read the code then investigate :)
-- Even the 100 rows takes the same time as the non looping. But it's 3 logical reads per row.
-- The second solution does 1 logical read for every 4 rows. So it's doing 12 times less reads, and also doing it supereffici
SET STATISTICS IO ON;

DECLARE @TotalPosts INT
DECLARE @TotalAnswerCount DECIMAL(10, 2)
DECLARE @CurrentPostID INT
DECLARE @CurrentAnswerCount DECIMAL(10, 2)
```

200 %

Results | Messages

	(No column name)	(No column name)	AverageAnswerCount
1	17142169	0.705694361081144	518

Query executed successfully. DESKTOP (16.0 RTM) | DESKTOP\bryde (76) | StackOverflow2013 | 00:00:02 | 1 rows

Ready Ln 25 Col 30 Ch 30 INS

7. Use AND instead of OR where possible

(Or split into 2 queries and UNION the results)

- Think through how an OR works, the server needs to traverse both branches of the OR, but typically just scans the whole table
- UNION is the most efficient solution, but is much larger to write

07 - UseUnionOrAndInsteadOfOr.sql - DESKTOP.StackOverflow2013 (DESKTOP\bryde (76)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query | StackOverflow2013 | Execute

07 - UseUnionOrAn...ESKTOP\bryde (76))

```sql
SET STATISTICS IO ON;

SELECT Id, Location, DownVotes FROM dbo.Users WHERE location = 'USA' OR DownVotes > 5
-- This first query does 50 scans and 2000 logical reads

SELECT Id, Location, DownVotes FROM dbo.Users WHERE location = 'USA'
UNION
SELECT Id, Location, DownVotes FROM dbo.Users WHERE DownVotes > 5

-- The second query does 2 scans and only 500 logical reads. So we get the same data but with 4 times less work
-- Often you will get significantly better results than this. There is also a memory benefit to this too.
```

200 %

Connected. (1/1) | DESKTOP (16.0 RTM) | DESKTOP\bryde (76) | StackOverflow2013 | 00:00:00 | 0 rows

Ready | Ln 1 | Col 1 | Ch 1 | INS

8. Minimize large writes

This one is often caused by the amount of locking that goes on.

Typically using Bulk Insert libraries you can avoid pain here. Be aware that the more foreign keys attached from or to your table, the worse this will get.

Indexes also have significant effect here.

9. Avoid wildcards at the start of string filters

i.e. don't use **LIKE '%fred'**

For EndWith, reverse the string and index on that…

09 - AvoidWildcardsAtTheStartOfStrings.sql - DESKTOP.StackOverflow2013 (DESKTOP\bryde (76))* - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query | StackOverflow2013 | Execute

09 - AvoidWildcard...ESKTOP\bryde (76))*

```sql
SET STATISTICS IO ON;

SELECT Title FROM dbo.Posts WHERE Title LIKE 'Why%'

SELECT Title FROM dbo.Posts WHERE Title LIKE '%statement'

-- If you must do this regularly and you know the % will be at the start
-- Create a column with the reverse of the string and index that.
-- I've created a column called ReverseTitle to demonstrate

SELECT reverse(ReverseTitle) FROM dbo.Posts WHERE ReverseTitle LIKE reverse('%statement')

-- As we can see looking for the same data we almost 1000 times less work.
-- Wow, now that's worth the effort
```

200 %

Connected. (1/1) | DESKTOP (16.0 RTM) | DESKTOP\bryde (76) | StackOverflow2013 | 00:00:00 | 0 rows

Ready | Ln 9 | Col 60 | Ch 60 | INS

10. Create joins with inner joins not where

- Optimizer optimizes **JOIN**s better than **WHERE**
- Can convert most **WHERE** to **JOIN** (but not ALL)
- Easier to read
- Makes it clear what type of **JOIN**

10 - JoinWithInnerJoinRatherThanWhere.sql - DESKTOP.StackOverflow2013 (DESKTOP\bryde (74)) - Microsoft SQL Server Management Studio

```sql
SET STATISTICS IO ON;

SELECT p.CommentCount, p.CreationDate, u.DisplayName
FROM dbo.Users AS u, dbo.Posts AS p
WHERE p.OwnerUserId = u.Id
AND u.Location = 'USA';

SELECT p.CommentCount, p.CreationDate, u.DisplayName
FROM dbo.Users AS u
JOIN dbo.Posts AS p ON p.OwnerUserId = u.Id
WHERE u.Location = 'USA';

-- Notice the statistics are almost identical. However we do get to choose what type of join to do.
-- Also the optimizer won't always convert the WHERE to a JOIN. This means the execution won't
-- always be the same (particularly for multiple JOINS).
```

Connected. (1/1) | DESKTOP (16.0 RTM) | DESKTOP\bryde (74) | StackOverflow2013 | 00:00:00 | 0 rows

Ready | Ln 1 | Col 1 | Ch 1 | INS

11. Use **Exists()** instead of **count()** where you can

- **Exists()** stops as soon as it finds a match
- **Count()** scans to count all the occurrences
- Often people write
  - WHERE count() > 0
  - WHERE Exists()

11 - UseExistsInsteadOfCountToCheckExistence.sql - DESKTOP.StackOverflow2013 (DESKTOP\bryde (54)) - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query | StackOverflow2013 | Execute

11 - UseExistsInste...DESKTOP\bryde (54))

```sql
SET STATISTICS IO ON;

-- This is an optimization where you want to know if there are any records.
-- Often people will just do a simple count and then write an if statement in their code

SELECT COUNT(*) FROM dbo.Posts WHERE Title LIKE 'Why%'


-- But you can use the Exists clause like this. The server will short circuit and stop looking
-- as soon as 1 row matches. That ends up 500 times faster here, also super fast even with no match :)

SELECT 1 WHERE EXISTS ( SELECT title FROM dbo.Posts WHERE Title LIKE 'Why%')


-- But wait you say, what about where nothing matches, do we get the same effect
SELECT COUNT(*) FROM dbo.Posts where Title like 'rewjtkl%'
SELECT 1 WHERE EXISTS ( SELECT title FROM dbo.Posts WHERE Title LIKE 'rewjtkl%')

-- nowehere near the improvement, but the positive case makes this well worth while.
```

200 %

Connected. (1/1) | DESKTOP (16.0 RTM) | DESKTOP\bryde (54) | StackOverflow2013 | 00:00:00 | 0 rows

Ready | Ln 1 | Col 1 | Ch 1 | INS

12. Avoid SELECT DISTINCT

Needs to sort to eliminate the duplicates

May require excessive reading

GROUP BY or more appropriate filtering are better solutions

12 - UseGroupByInsteadOfSelectDistinct.sql - DESKTOP.StackOverflow2013 (DESKTOP\bryde (54))* - Microsoft SQL Server Management Studio

File Edit View Query Project Tools Window Help

New Query | StackOverflow2013 | Execute

12 - UseGroupByIns...ESKTOP\bryde (54))*

```
SET STATISTICS IO ON;

SELECT DISTINCT OwnerUserId
FROM dbo.Posts
WHERE AnswerCount > 5

SELECT OwnerUserId, Avg(AnswerCount)
FROM dbo.Posts
WHERE AnswerCount > 5
GROUP BY OwnerUserId

-- Not a huge improvement here, just 0.1%. But the improvement should rise with more complicated queries.
-- The other advantage is any aggregates can get calculated at the same time, adding significant
-- performance gains.
```

200 %

Connected. (1/1) | DESKTOP (16.0 RTM) | DESKTOP\bryde (54) | StackOverflow2013 | 00:00:00 | 0 rows

Ready | Ln 3 | Col 29 | Ch 29 | INS

13. Avoid too many joins

While this one is again “it depends”

In general 4-7 **JOIN**s is getting high

Anything more than this will result in the query optimizer struggling significantly.

14. Don't create indexes on tiny tables

If the data fits in a single page, there is no advantage

This is just a small storage optimization, it’s very hard to give a convincing example.

A single page is 8Kb

15. Use **TOP** for sampling

If you just need to show a few rows as a sample or to verify data. Remember to use **TOP** to cap the number of rows.

Ie: **SELECT TOP 100 * FROM customers**

16. Don't index columns that have few unique values

Sometimes… Don’t make it the primary column in the index if there is a less selective….

You want the least selective columns as early as possible in an index…. Remember **WHERE** and **ORDER BY** should be separated.

17. Use WHERE instead of HAVING

If you can using WHERE before aggregating is far more efficient than using HAVING after aggregation.

This is because WHERE can take advantage of an index while HAVING cannot.

But they do different things, so often you can't.

Recap:

1. Reduce Table Size ✨
2. **SELECT** only the columns you want ✨
3. Create Indexes ✨
4. Verify your indexes are used
5. Avoid implicit type conversions
6. Avoid looping
7. Use **AND** instead of **OR**
8. Minimise large writes

9. Avoid wildcards at the start of filters
10. Use **JOIN** over **WHERE**
11. Use **EXISTS** instead of **COUNT > 0**
12. Avoid **SELECT DISTINCT**
13. Avoid too many **JOIN**s
14. Don't index tiny tables
15. Use **TOP** for sampling
16. Don't index columns with repeating values
17. Use **WHERE** instead of **HAVING**

Resources

Brent Ozar

brentozar.com

Pinal Dave:

https://blog.sqlauthority.com/

PASS:

passdatacommunitysummit.com/sessions/video-library/

And loads more, **Stack Overflow** is also great!

The demo database used here (and that lots of SQL demos and courses use) is sourced from:
archive.org/details/stackexchange

Brent Ozar keeps a ready to deploy SQL Server database of that, easily downloadable in various sizes for people to learn with. Thanks Brent!!!

Specific further reading

Think like the SQL Server Engine

brentozar.com/training/think-like-sql-server-engine/

Sargable Expressions

sqlshack.com/how-to-use-sargable-expressions-in-t-sql-queries-performance-advantages-and-examples/

Thank you!

ssw.com.au

Sydney | Melbourne | Brisbane | Newcastle | Strasbourg | Hangzhou

github.com/brydeno/DatabasePerformance

Questions?

ssw.com.au

Sydney | Melbourne | Brisbane | Newcastle | Strasbourg | Hangzhou

github.com/brydeno/DatabasePerformance