



BRENT OZAR
UNLIMITED®

Mastering Query Tuning

1.1 p1

Introduce yourself in Slack:

	Developer	Development DBA	Production DBA
Write C#, Java code	Daily		
Build queries, tables	Daily	Sometimes	
Tune queries	Sometimes	Daily	
Design indexes		Daily	
Monitor performance		Daily	Sometimes
Troubleshoot outages			Daily
Manage backups, jobs			Daily
Install, config SQL			Sometimes
Install, config OS			Sometimes



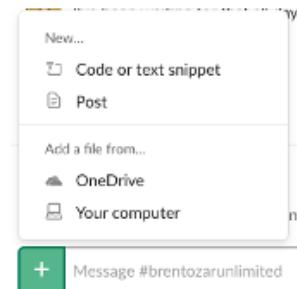
1.1 p2

Slack pro tips

Accidentally close your browser? Want to share screenshots? Lots of pro tips: BrentOzar.com/slack

To share code or T-SQL,
click the + sign next to where
you type text in, and choose
“code or text snippet.”

No direct messages please
– use the public room.



1.1 p3

Slack is a great cheat code.

Get stumped on a lab?

Wondering how other students solved it?

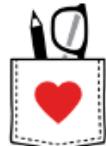
Wondering how *quickly* other students work?

Take a peek in the Slack room.

Otherwise, don't look (spoilers.)



1.1 p4



BRENT OZAR UNLIMITED

99-05: dev, architect, DBA
05-08: DBA, VM, SAN admin
08-10: MCM, Quest Software
Since: consulting DBA

www.BrentOzar.com
Help@BrentOzar.com



1.1 p5

My job: 2-day SQL Critical Care®

Day 1, morning: rapidly assess a single SQL Server, database indexes, queries running against it, team

Day 1 afternoon & day 2 morning: write findings

Day 2 afternoon: deliver findings & training to get the team out of the emergency, quickly

This week: sharing my techniques, experiences



1.1 p6

I'll show you with lectures + labs.

This isn't about theory: it's about practice.

We'll step through this cycle repeatedly:

1. **Lecture:** you watch me for 1-2 hours
2. **Hands-on lab:** you spend 1 hour working on a lab about the concepts you just saw
3. **I do the lab:** you watch me spend 30-45 minutes on that same lab so you can check your work



1.1 p7

Instant Replay & lab scripts

For a year from your date of purchase, you can:

- Watch the videos (I'm recording this class)
- Download the scripts and database
- Re-run the labs on your own home machines

Problems? Comment on the module.



1.1 p8

We'll be doing labs to cover:

Finding the worst queries to focus on with `sp_BlitzCache`

Once you've identified the worst queries, how to tune them and measure that your changes helped

How to decide if query or index changes are a better fit

How to discover when SQL Server's cardinality estimation is holding performance back

How to build and tune dynamic SQL



1.1 p9

General agenda

9AM-Noon: lectures, short lab exercises

Noon-2PM: big lab & lunch

2PM-3PM: watch me do the lab you just did

3PM-4PM: lecture

4PM-5PM: big lab homework



1.1 p1O

Today

Lectures:

- Optimization levels
- Cardinality estimation, and improving it

Lab 1: you tune a single ugly query at a time

Lectures: CTEs, temp tables, APPLY, sp_BlitzCache

Lab 2: you find the worst queries in a live workload



1.1 p11

Next 2 days

- Progressively larger & harder query rewrites
- Tuning for **SELECT ***, paging
- Building & tuning dynamic SQL
- How parallelism balances work across threads
- How to use batches properly to avoid lock escalation
- Avoiding deadlocks & blocking
- What's new for query tuners in SQL Server 2017 & 2019
- Labs where everything's up for grabs



1.1 p12

Things we won't cover, but I love:

Monitoring tools, including Query Store

Red Gate SQL Prompt: better than IntelliSense

SentryOne Plan Explorer:
free-as-in-beer query plan viewer & analysis

sp_WhoIsActive: better than Activity Monitor

Azure Data Studio: the new SSMS for T-SQL





BRENT OZAR
UNLIMITED®

How SQL Server Builds Query Plans

1.1 p14

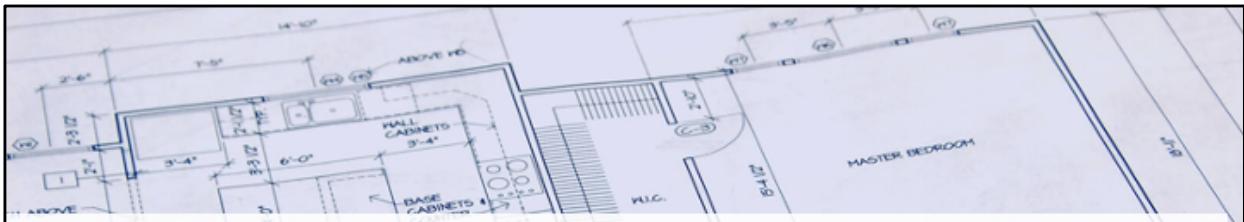
**Say you want
a custom house.**



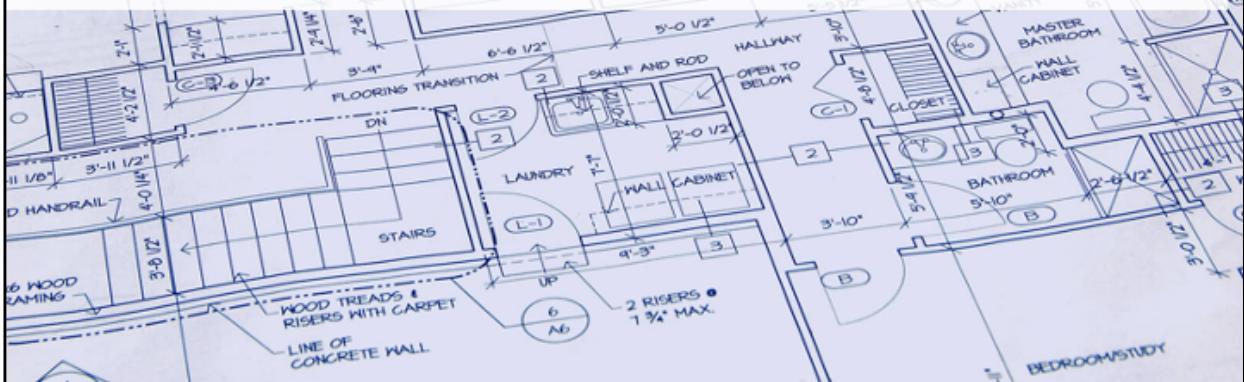
1.1 p15



You meet with an architect.



The architect designs a blueprint plan.





**The builder takes the plan,
and implements it. This takes a while.**





**Change your mind along the way?
The builder can make some small tweaks.**



“I want a 3-bedroom, 2-bath house”	<code>SELECT * FROM dbo.Sales WHERE Location = ‘NYC’</code>
Architect builds a blueprint	Query Optimizer builds a query plan
Builder executes the plan	Query Processor executes the plan
Supply store delivers the materials	Storage Engine fetches the pages



1.1 p20

“I want a 3-bedroom,
2-bath house”

```
SELECT * FROM dbo.Sales  
WHERE Location = ‘NYC’
```

The more specific you are about your needs and your budget,
the better home blueprint you’ll get.

This will save you time on construction later,
keeping your house on time and on budget.



1.1 p21

Architect
builds a blueprint

Query Optimizer
builds a query plan

The architect builds a blueprint based on what he knew at the time.

Your needs might change over time, like starting a family.

The architect doesn't even know much about
which builder you plan on using, and how big they are.

The builder doesn't have the right to tell the architect,
“Hey, this plan is makes no sense given my supplies.”

Builder
executes the plan

Query Processor
executes the plan

He just does what he's told,
and he has very, very limited leeway.

The supply store doesn't know anything about your blueprints.

You also can't really tweak it much:
either you're shopping at the right store, or you're not.

Either it's got the right hardware, or not.

Supply store
delivers the materials

Storage Engine
fetches the pages



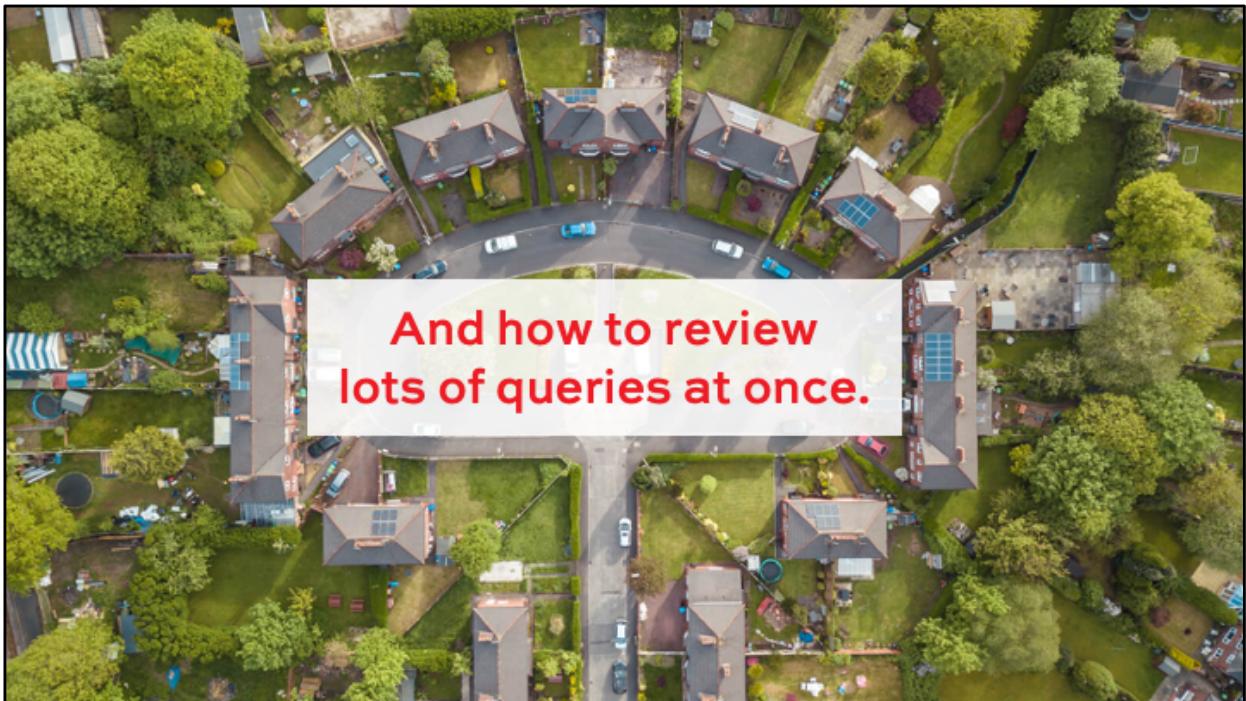
1.1 p24

“I want a 3-bedroom, 2-bath house”	<code>SELECT * FROM dbo.Sales WHERE Location = ‘NYC’</code>
Architect builds a blueprint	Query Optimizer builds a query plan
Builder executes the plan	Query Processor executes the plan
This class focuses on:	
what to tell the architect, how the architect builds plans, how the builder executes the architect’s plan, and how to get the plan you really want.	



We'll start with a
small, easy one.





**And how to review
lots of queries at once.**

Building our first blueprint

Trivial plans



1.1 p29

Let's go to SQL Server's defaults.

```
EXEC sys.sp_configure N'cost threshold for  
parallelism', N'5'  
GO  
EXEC sys.sp_configure N'max degree of  
parallelism', N'0'  
GO  
RECONFIGURE WITH OVERRIDE  
GO
```



1.1 p30

Let's start with a blank slate.

```
USE StackOverflow;
GO
EXEC DropIndexes;
GO
SET STATISTICS TIME ON;
GO
```



1.1 p31

Ask the architect to build a plan.

```
SELECT TOP 50 *
    FROM dbo.Users
    WHERE Reputation = 2
```

The only index on the Users table is the clustered index on Id.

Hit control-L to get an estimated plan.



1.1 p32

He built the plan FAST.

```
SELECT TOP 50 *
FROM dbo.Users
WHERE Reputation = 2;
GO
```

100 %

Messages Execution plan

```
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

(1 row affected)
SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.
```

Statistics time shows less than 1 millisecond of time spent parsing and compiling.

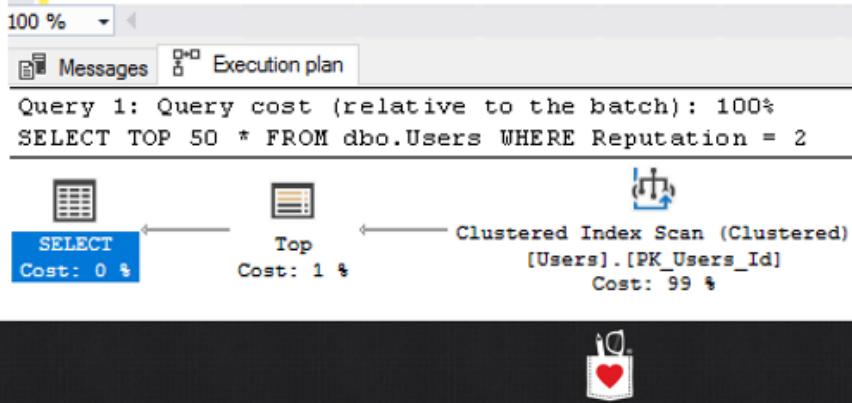


1.1 p33

It's just not a fast plan.

```
SELECT TOP 50 *
FROM dbo.Users
WHERE Reputation = 2;
GO
```

It's a clustered index scan: starting at the beginning, and reading until he finds 50 users with Reputation = 2.

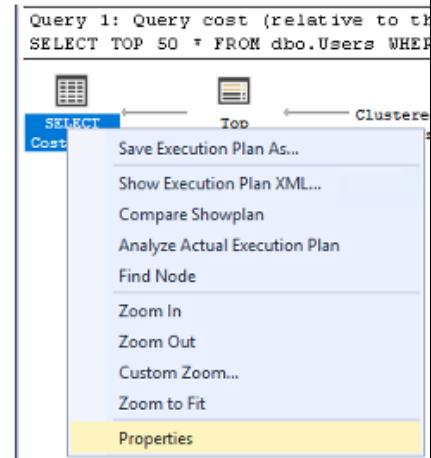


1.1 p34

Why? Right-click on the select.

Right-click on the select operator,
click Properties.

In the right hand side window, look
for Optimization Level.



Trivially simple

The architect (Query Optimizer) said, “This query is so trivially easy that I don’t have to put much work into building a blueprint.”

“I’ll just slap in any old blueprint and ship it. They’ll be fine.”

And in fairness, he's usually right.



1.1 p36

Optimization levels

- Trivial optimization
- Full, but exiting early for a good reason
- Full, but exiting early for a bad reason (timeout)
- Full, after extensive evaluations



1.1 p37

Trivial queries/plans

Really easy to build a plan fast

It just may not be a fast plan

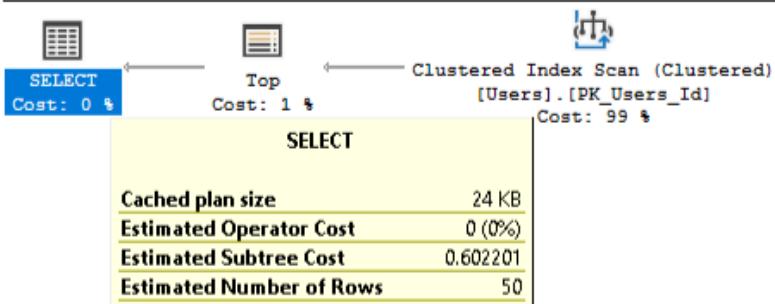
- No parallelism
- No missing index requests

But trivial queries are usually so simple it doesn't matter, and they're rarely your biggest bottleneck



The architect thinks it's cheap.

Query 1: Query cost (relative to the batch): 100%
SELECT TOP 50 * FROM dbo.Users WHERE Reputation = 2

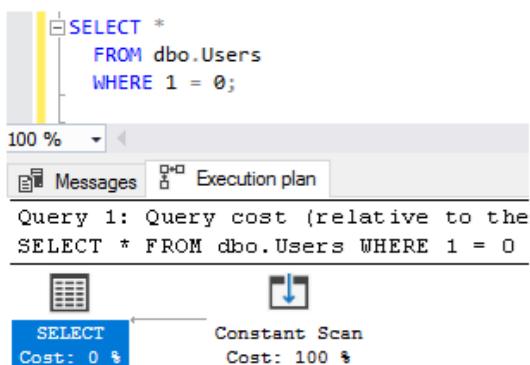


The architect estimates that this will only cost the builder 0.6 query bucks to build, so why bother putting more time into a blueprint?



1.1 p39

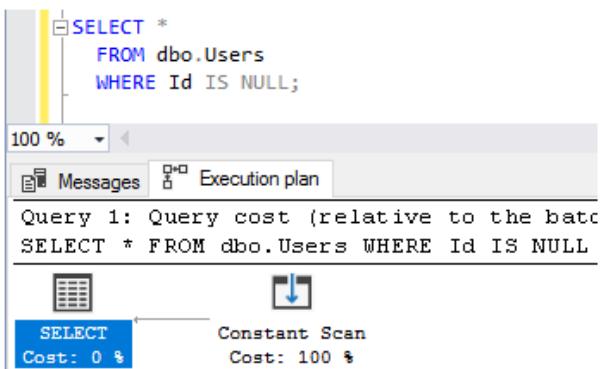
Trivial plan example



The architect figured out that the builder doesn't even need to go to the supply store – no results are needed.



Constraints help too



A screenshot of the SQL Server Management Studio interface. The top pane shows a T-SQL query: `SELECT * FROM dbo.Users WHERE Id IS NULL;`. The bottom pane displays the execution plan for this query. The plan consists of two nodes: a 'SELECT' node with a cost of 0 % and a 'Constant Scan' node with a cost of 100 %. A line connects these two nodes, indicating a flow from the constant scan to the select.

```
SELECT *
FROM dbo.Users
WHERE Id IS NULL;

100 %

Messages Execution plan

Query 1: Query cost (relative to the batch)
SELECT * FROM dbo.Users WHERE Id IS NULL

SELECT Cost: 0 %
Constant Scan Cost: 100 %
```

`Id` is the primary key, non-nullable. The more the architect (QO) knows about your data, the more work he can eliminate ahead of time!



Reputation is a non-nullable field

The screenshot shows a SQL query window and its execution plan. The query is:

```
SELECT *  
FROM dbo.Users  
WHERE Reputation IS NULL
```

The execution plan shows a Constant Scan operator with a cost of 100% and a SELECT operator with a cost of 0%. The plan is as follows:

```
Constant Scan  
Cost: 100 %  
SELECT  
Cost: 0 %
```

So we get table elimination here too



1.1 p42

I love trivially simple queries.

But they're not usually your biggest problem.

However, they can be when:

- You have a seemingly easy query
- But it's run thousands of times per second
- It really desperately needs an index
- But SQL Server isn't asking for one

Catch em: `sp_BlitzCache @SortOrder = 'executions'`



1.1 p43

In this class, I'll avoid trivial plans.

I love using trivial plans for public demos because they really surprise people.

Here, we're done with the tiny-plan surprises.

From here on out, we're going to focus on full optimization queries because that's what you're usually up against at work.

I just needed to warn you in case you ever hit a trivial query as your biggest bottleneck.



1.1 p44

Let's upgrade to a doghouse

Full optimization (non-trivial queries)



1.1 p45

Let's add just a little complexity.

```
SELECT TOP 50 *
    FROM dbo.Users
    WHERE Reputation = 2
    AND Location = 'San Diego'
```

This is more selective, so it's going to produce less rows as output. However, the builder will need to scan through more stuff at the store to find them.



1.1 p46

Suddenly, a lot changes.

Now we get parallelism and a missing index request.

The screenshot shows a SQL query window and its execution plan. The query is:

```
SELECT TOP 50 *
FROM dbo.Users
WHERE Reputation = 2
AND Location = 'San Diego'
GO
```

The execution plan shows the following steps:

- SELECT (Cost: 0 %)
- Top (Cost: 0 %)
- Parallelism (Gather Streams) (Cost: 2 %)
- Clustered Index Scan (Clustered) [Users].[PK_Users_Id] (Cost: 98 %)

Below the plan, a message indicates a missing index:

Query 1: Query cost (relative to the batch): 100%
SELECT TOP 50 * FROM dbo.Users WHERE Reputation = 2 AND Location = 'San Diego'
Missing Index (Impact 99.4039): CREATE NONCLUSTERED INDEX [<Name of Missing Ind]

A small number '47' is visible in the bottom right corner of the screenshot area.

Because the architect rolled up his sleeves.

He decided that this query was not trivial, so he did a full optimization.

He took more time to consider more ways to build this query plan, like parallelizing the work across multiple cores, and recommending indexes.



Properties	
SELECT	
Misc	
Cached plan size	32 KB
CardinalityEstimationModelVers	140
CompileCPU	1
CompileMemory	296
CompileTime	1
Estimated Number of Rows	4,74379
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	91.4892
	MemoryGrantInfo
	MissingIndexes
	Optimization Level
	FULL
	OptimizerHardwareDependentP
	OptimizerStatsUsage
QueryHash	0xE3217CD7D1
QueryPlanHash	0x3F3BCD2C34
RetrievedFromCache	false
SecurityPolicyApplied	False
	Set Options
Statement	ANSI_NULLS: T
	SELECT TOP 50
	ThreadStat

The architect has lots of tricks.

Rewriting your join order, getting different data first

Considering different indexes, batch mode

Using objects you didn't ask for:
indexed views, computed columns

Eliminating joins using foreign keys & constraints

Rewriting UNION, UNION ALL

But doing all these computations has a drawback...



It took the architect longer. The plan itself cost us more.

```
SELECT TOP 50 *
FROM dbo.Users
WHERE Reputation = 2
AND Location = 'San Diego'
GO
```

100 %

Messages Execution plan

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server parse and compile time:
CPU time = 2 ms, elapsed time = 2 ms.

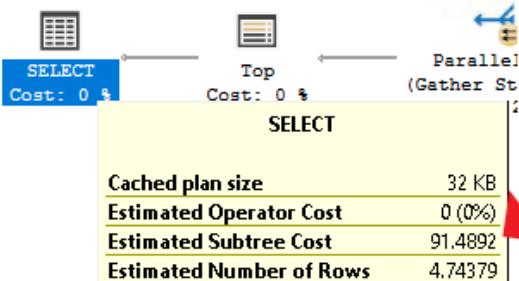
(1 row affected)

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

1.1 p50

The builder's cost is higher, but...

```
Query 1: Query cost (relative to the
SELECT TOP 50 * FROM dbo.Users WHERE
Missing Index (Impact 99.4039): CREA
```



Before: “This will cost the builder only \$0.60 query bucks to build.” Now: whoa!



Estimated costs are like budgets.

The architect is just guessing the builder's costs solely for the purpose of designing better blueprints.

THE COST ISN'T REPORTED BY THE BUILDER.

Even in actual plans, you're only looking at the architect's estimates of what the builder will spend, not how much he actually spent.



1.1 p52

This is where costs get weird.

	Architect's Costs (Query Optimizer)	Builder's Costs (Query Processor)
Estimated	Not calculated	Shown in plan, but completely inaccurate
Actual	Statistics Time, but varies constantly	Statistics Time & IO, but somewhat inaccurate



1.1 p53

Our query's estimates

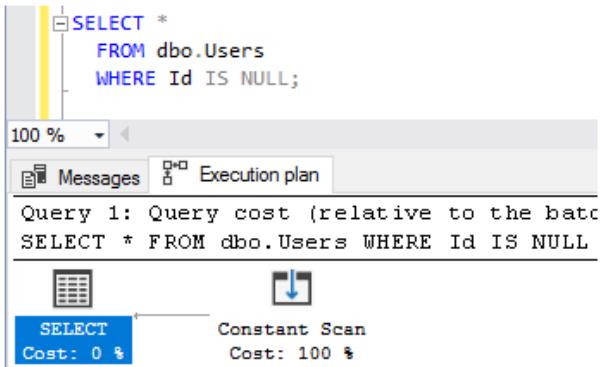
	WHERE Reputation = 2	AND Location = 'San Diego'
Optimization level	Trivial	Full
Estimated cost	0.6	91

It's not just that the left query results are easier to build. It's also that the architect spent more time estimating the builder's costs, and came up with a wildly different number.



1.1 p54

Remember when the architect eliminated null searches?



The screenshot shows a SQL query window with the following text:

```
SELECT *
FROM dbo.Users
WHERE Id IS NULL;
```

Below the query window is the SQL Server Management Studio interface, specifically the 'Execution plan' tab. It displays the following information:

Query 1: Query cost (relative to the batch)
SELECT * FROM dbo.Users WHERE Id IS NULL

The execution plan shows two operations:

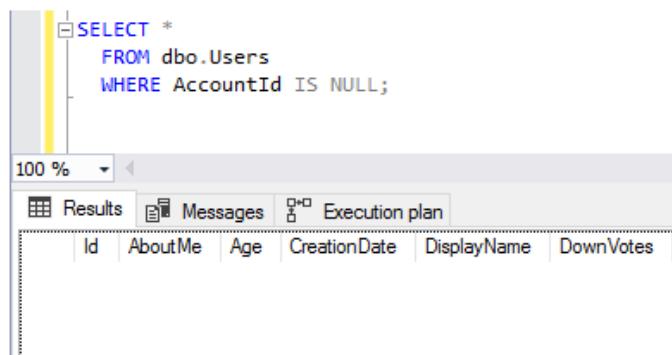
- A **SELECT** node with **Cost: 0 %**.
- A **Constant Scan** node with **Cost: 100 %**.

The Constant Scan node is highlighted with a yellow background.

He knew Id could never be null, so he eliminated this table altogether, cutting our builder's costs.



What if the architect doesn't know?



A screenshot of SQL Server Management Studio (SSMS) showing a query results grid. The query is:

```
SELECT *  
FROM dbo.Users  
WHERE AccountId IS NULL;
```

The results grid has columns: Id, AboutMe, Age, CreationDate, DisplayName, and DownVotes. There are no rows displayed.

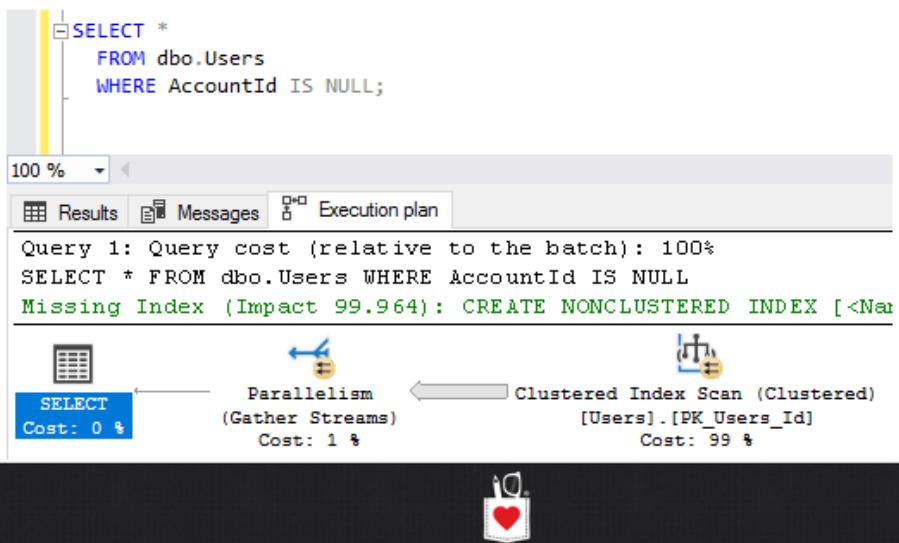
Here, we're asking for null AccountIds.

There aren't any.

But the architect doesn't know that, because the field is defined as nullable.



**So he builds a full blueprint,
and makes the builder scan it.**



1.1 p57

Query tuning is about...

Giving the architect as much help as possible to:

- Tell him as much as we can about the house, the end result that we want, and the supplies we have
- Lower his costs of building a plan
- Later, lower the builder's costs of building results with indexes, better hardware, settings

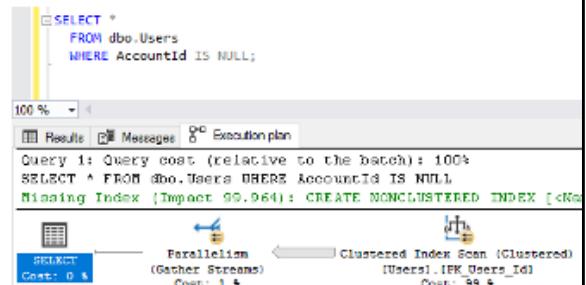


In this case...

The answer isn't to build an index.

The answer is to make AccountId non-nullable, and eliminate the scan altogether!

But it's up to you to know that: the architect can't know that you'll never put a null in AccountId.



1.1 p59

Like how I fully did my chores

Full optimization that isn't really full



1.1 p60

Let's add just one join.

```
SELECT u1.*  
FROM dbo.Users u1  
INNER JOIN dbo.Users u2 ON u2.Id = u1.Id  
WHERE u1.Reputation = -1
```

No users match Reputation = -1,
so this query will run fast.



1.1 p61

The query plan is simple.

And SQL Server generates it quickly:



1.1 p62

Good enough.

**Reason for Early Termination:
Good Enough Plan Found.**

**SQL Server spent hardly any CPU time
or memory in order to build it.**

**The architect recognized early on that
this plan will be easy enough for the
builder, and spending more time
building a plan won't pay off.**

SELECT	
E	Misc
Cached plan size	56 KB
CardinalityEstimationModelVers	140
CompileCPU	2
CompileMemory	352
CompileTime	2
Estimated Number of Rows	1
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0.0093577
E	MemoryGrantInfo
Optimization Level	FULL
E	OptimizerHardwareDependentP
E	OptimizerStatsUsage
QueryHash	0x14A10B3CC37C902
QueryPlanHash	0x7883C689F1FF6E9
Reason For Early Termination Of	Good Enough Plan Found
RetrievedFromCache	true



**THIS QUERY TAKES
FOUR HOURS TO RUN**

**GOOD ENOUGH
PLAN FOUND**



1.1 p64

```
INNER JOIN dbo.Users u62 ON u62.Id = u61.Id
INNER JOIN dbo.Users u63 ON u63.Id = u62.Id
INNER JOIN dbo.Users u64 ON u64.Id = u63.Id
INNER JOIN dbo.Users u65 ON u65.Id = u64.Id
INNER JOIN dbo.Users u66 ON u66.Id = u65.Id
INNER JOIN dbo.Users u67 ON u67.Id = u66.Id
INNER JOIN dbo.Users u68 ON u68.Id = u67.Id
INNER JOIN dbo.Users u69 ON u69.Id = u68.Id
INNER JOIN dbo.Users u70 ON u70.Id = u69.Id
INNER JOIN dbo.Users u71 ON u71.Id = u70.Id
INNER JOIN dbo.Users u72 ON u72.Id = u71.Id
INNER JOIN dbo.Users u73 ON u73.Id = u72.Id
INNER JOIN dbo.Users u74 ON u74.Id = u73.Id
INNER JOIN dbo.Users u75 ON u75.Id = u74.Id
INNER JOIN dbo.Users u76 ON u76.Id = u75.Id
INNER JOIN dbo.Users u77 ON u77.Id = u76.Id
INNER JOIN dbo.Users u78 ON u78.Id = u77.Id
INNER JOIN dbo.Users u79 ON u79.Id = u78.Id
INNER JOIN dbo.Users u80 ON u80.Id = u79.Id
INNER JOIN dbo.Users u81 ON u81.Id = u80.Id
INNER JOIN dbo.Users u82 ON u82.Id = u81.Id
INNER JOIN dbo.Users u83 ON u83.Id = u82.Id
INNER JOIN dbo.Users u84 ON u84.Id = u83.Id
INNER JOIN dbo.Users u85 ON u85.Id = u84.Id
INNER JOIN dbo.Users u86 ON u86.Id = u85.Id
INNER JOIN dbo.Users u87 ON u87.Id = u86.Id
INNER JOIN dbo.Users u88 ON u88.Id = u87.Id
INNER JOIN dbo.Users u89 ON u89.Id = u88.Id
INNER JOIN dbo.Users u90 ON u90.Id = u89.Id
INNER JOIN dbo.Users u91 ON u91.Id = u90.Id
INNER JOIN dbo.Users u92 ON u92.Id = u91.Id
INNER JOIN dbo.Users u93 ON u93.Id = u92.Id
INNER JOIN dbo.Users u94 ON u94.Id = u93.Id
INNER JOIN dbo.Users u95 ON u95.Id = u94.Id
INNER JOIN dbo.Users u96 ON u96.Id = u95.Id
INNER JOIN dbo.Users u97 ON u97.Id = u96.Id
INNER JOIN dbo.Users u98 ON u98.Id = u97.Id
INNER JOIN dbo.Users u99 ON u99.Id = u98.Id
INNER JOIN dbo.Users u100 ON u100.Id = u99.Id
WHERE u1.Reputation = -1
```

Let's add a few more.

Oh, I dunno, 100 sounds good.

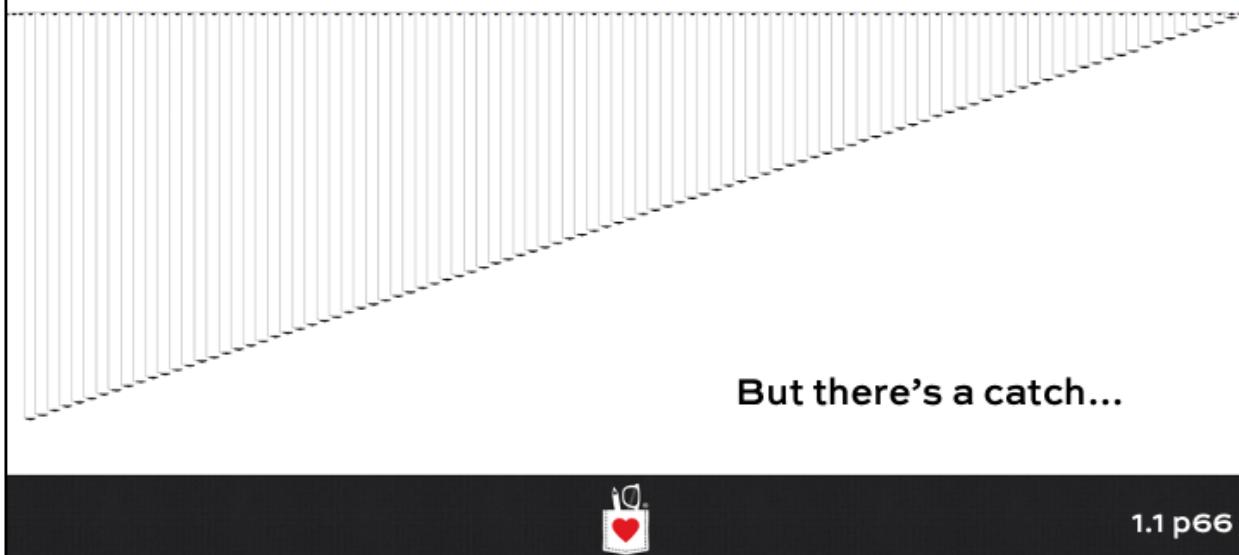
Technique to build queries like this:

<https://www.brentozar.com/archive/2016/08/bad-idea-jeans-dynamically-generating-ugly-queries-task-manager-graffiti/>



1.1 p65

The architect builds a plan quickly



But there's a catch...



1.1 p66

**Time out.
Go sit in the corner.**

The architect started to do a full optimization, but...

At some point, it gave up.

It ran out of time – even though we only spent about a second building the plan.

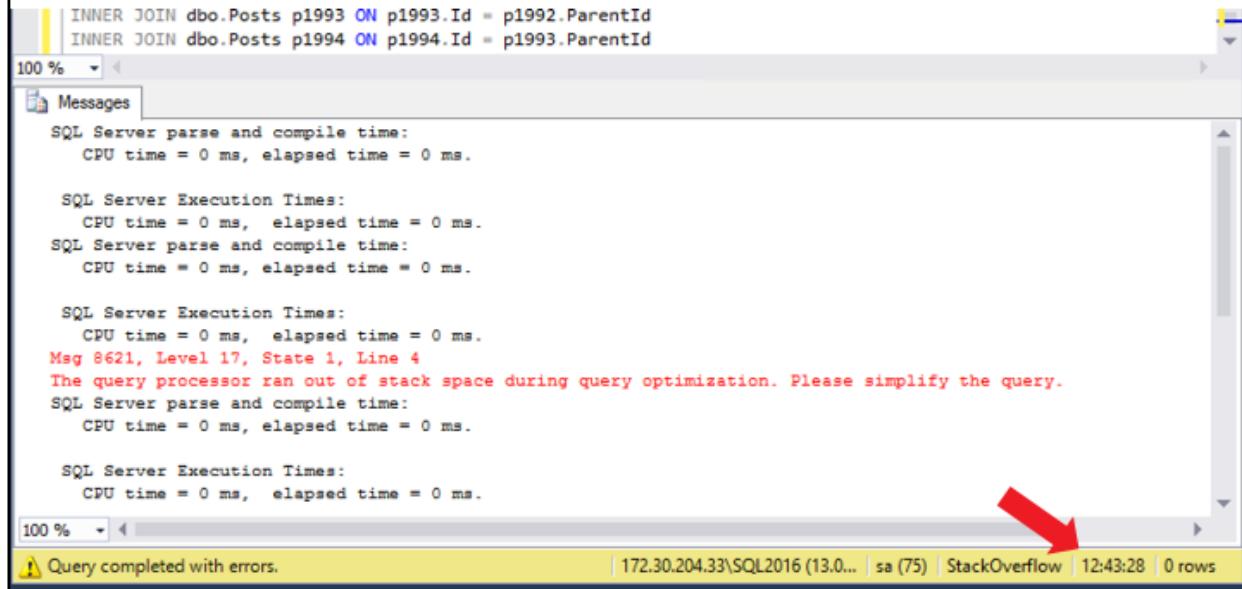
You don't get knobs for timeout controls. It's managed by SQL.

Properties	
SELECT	
<input checked="" type="checkbox"/>	Cache
<input checked="" type="checkbox"/>	Script
<input checked="" type="checkbox"/>	Misc
Cached plan size	1584 KB
CardinalityEstimationModeVersion	70
CompileCPU	956
CompileMemory	26656
CompileTime	956
Estimated Number of Rows	1
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	35.1666
<input checked="" type="checkbox"/>	MemoryGrantInfo
<input checked="" type="checkbox"/>	MissingIndexes
Optimization Level	FULL
<input checked="" type="checkbox"/>	OptimizerHardwareDependentProperties
QueryHash	0xC6280DC34E19246
QueryPlanHash	0x223B18A99D7CFFF
Reason For Early Termination Of Statement Optimization	Time Out
RetrievedFromCache	true



1.1 p67

Hard plans can take a LONG time.



INNER JOIN dbo.Posts p1993 ON p1993.Id = p1992.ParentId
INNER JOIN dbo.Posts p1994 ON p1994.Id = p1993.ParentId

100 % Messages

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

Msg 8621, Level 17, State 1, Line 4
The query processor ran out of stack space during query optimization. Please simplify the query.

SQL Server parse and compile time:
CPU time = 0 ms, elapsed time = 0 ms.

SQL Server Execution Times:
CPU time = 0 ms, elapsed time = 0 ms.

100 %

Query completed with errors. | 172.30.204.33\SQL2016 (13.0...) | sa (75) | StackOverflow | 12:43:28 | 0 rows

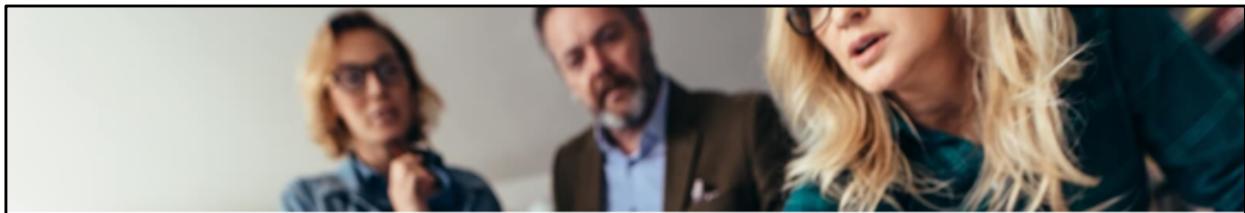
A red arrow points to the 'StackOverflow' status indicator in the bottom status bar.

Optimization levels

	Trivial	Full	Full	Full
Early termination	None	None	Good Enough Plan Found	Timeout
Query is...	Really simple	Moderate	Moderate	Hellscape
Missing index hints	No	Considered	Considered	Considered
Parallelism	No	Considered	Considered	Considered
Columnstore indexes	Depends on SQL build	Considered	Considered	Considered
Tuning will usually involve...	Small tweaks	Tweaks & indexes	Mild rewrites	Complete rewrites



1.1 p69



Optimization levels aren't the takeaway.



Here are the real takeaways.

The architect – the Query Optimizer – builds a plan:

- Starting with what you ask for
- Tries a few revisions with what he knows about the builder and his supply store (indexes, hardware)
- He can – and does – run out of time
- He doesn't know how successful (or unsuccessful) the builder was
- The blueprint keeps getting used, regardless



1.1 p71

This class is about better blueprints.

You can get better query plans by:

- Learning to read the blueprints, and understanding when they're not the house you really wanted
- Rewriting your query to be easier to understand
- Improving the database to tell SQL Server more about your data

