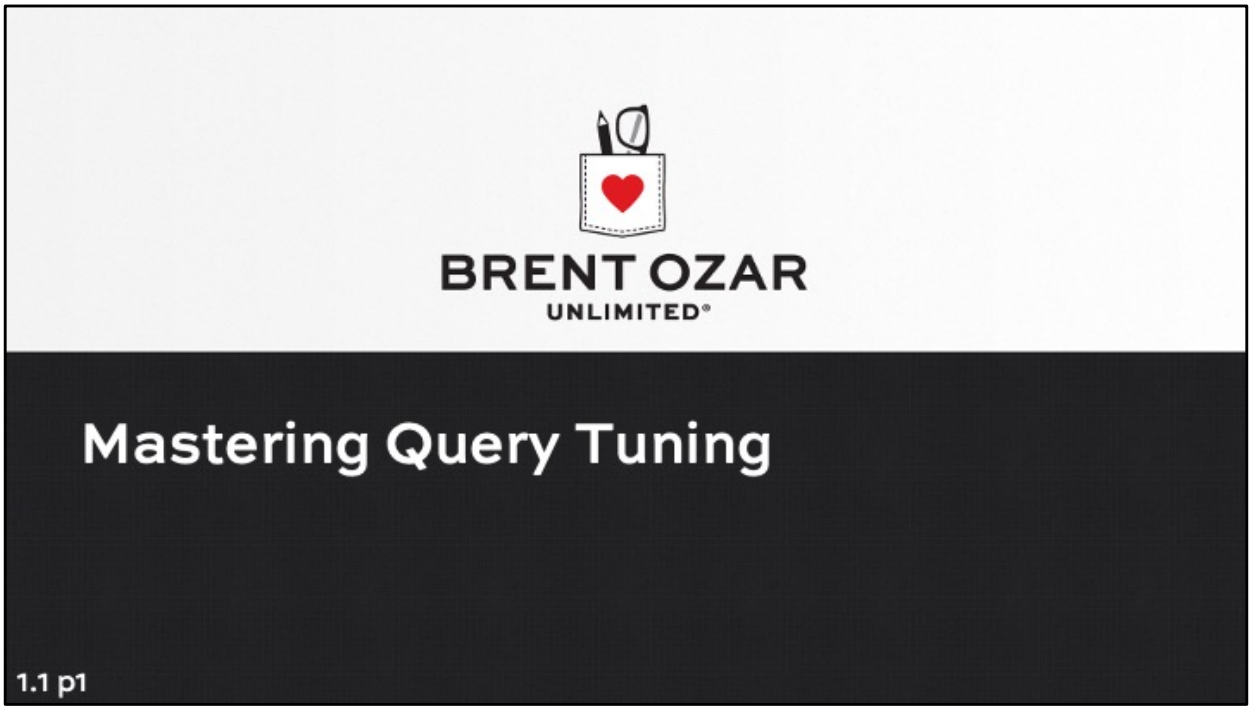
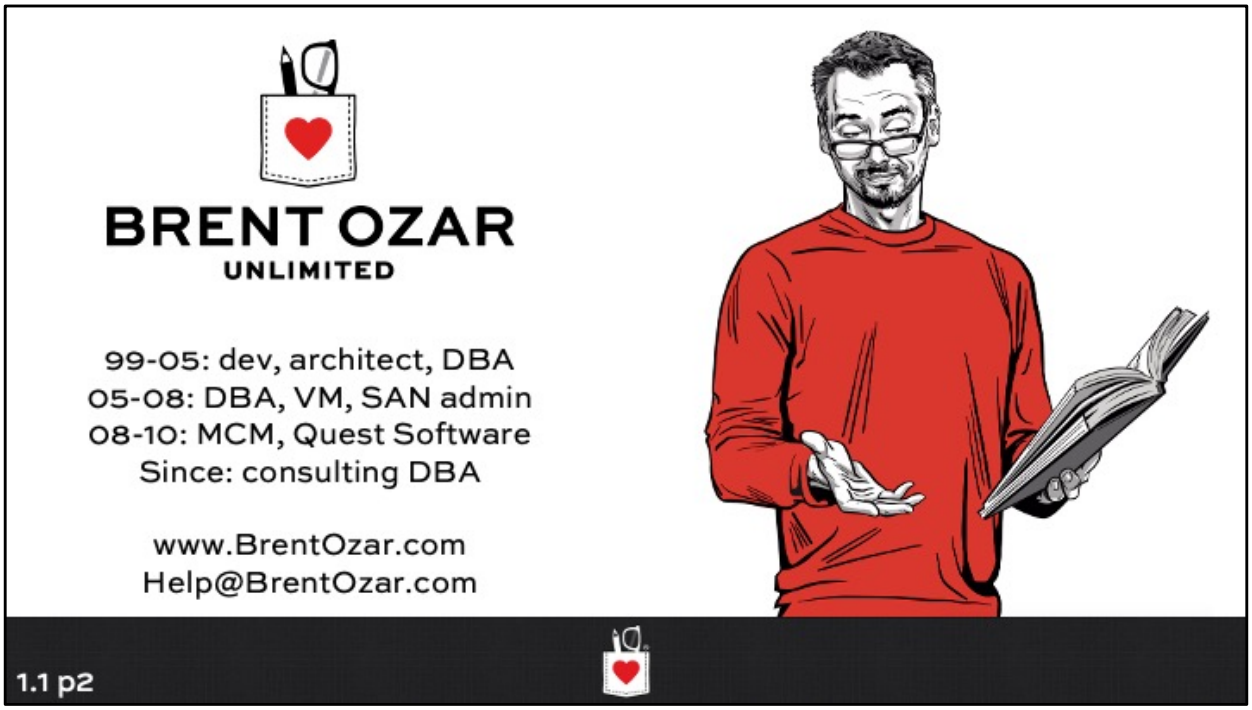




My job: 2-day SQL Critical Care®

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 p3

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

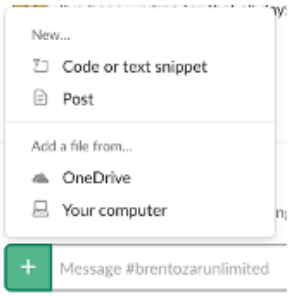
1.1 p4

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.



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 p6

What we covered in Fundamentals

Building a query plan

Finding the most resource-intensive queries with sp_BlitzCache

How parameters influence plans

Improving cardinality estimation

Common T-SQL antipatterns

1.1 p7

What we'll cover this week

More ways to improve cardinality estimation

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, 2022

1.1 p8

Things we won't cover, but I love:

Monitoring tools, Query Store

Red Gate SQL Prompt:
better than IntelliSense

Azure Data Studio:
the new SSMS for T-SQL

1.1 p9

General agenda

9AM-Noon: lectures, short lab exercises

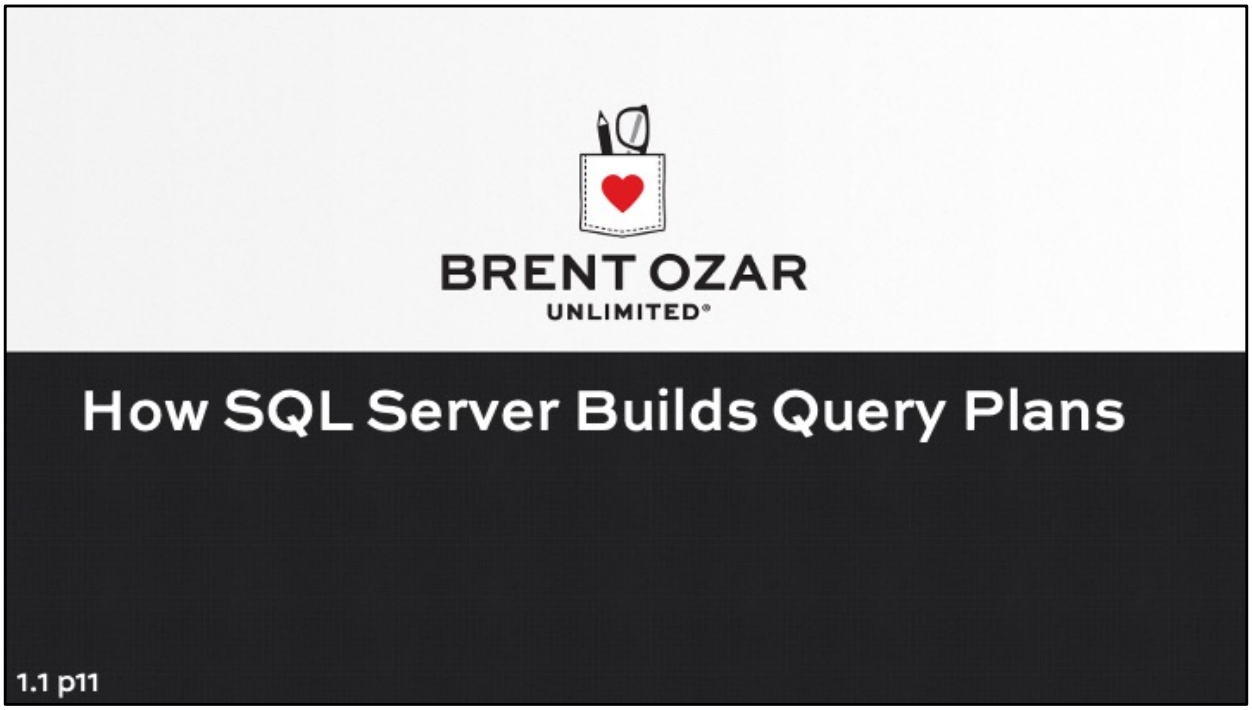
Noon-2PM: big lab & lunch

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

3PM-4PM: lecture

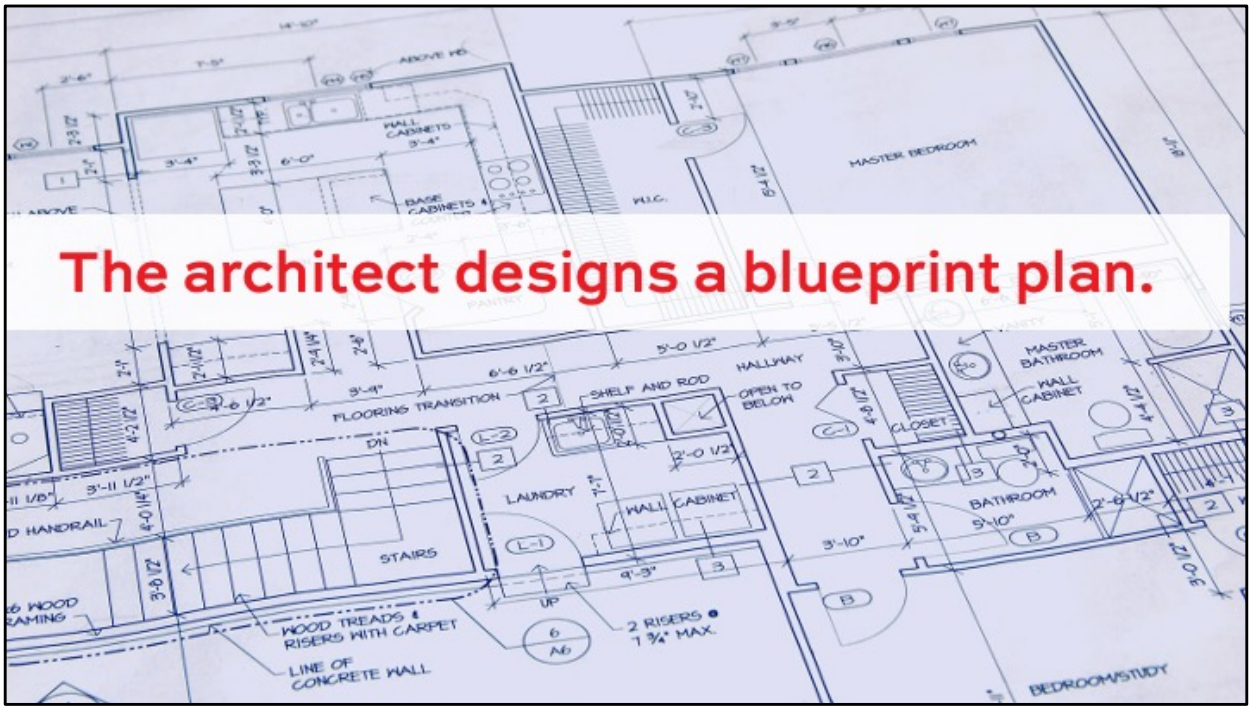
4PM-5PM: big lab homework

1.1 p10



Say you want
a custom house.

1.1 p12



"I want a 3-bedroom, 2-bath house"	SELECT * FROM dbo.Sales WHERE Location = 'NYC'
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 p17

“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 p18

Architect	Query Optimizer
builds a blueprint	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	Query Processor
executes the plan	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.

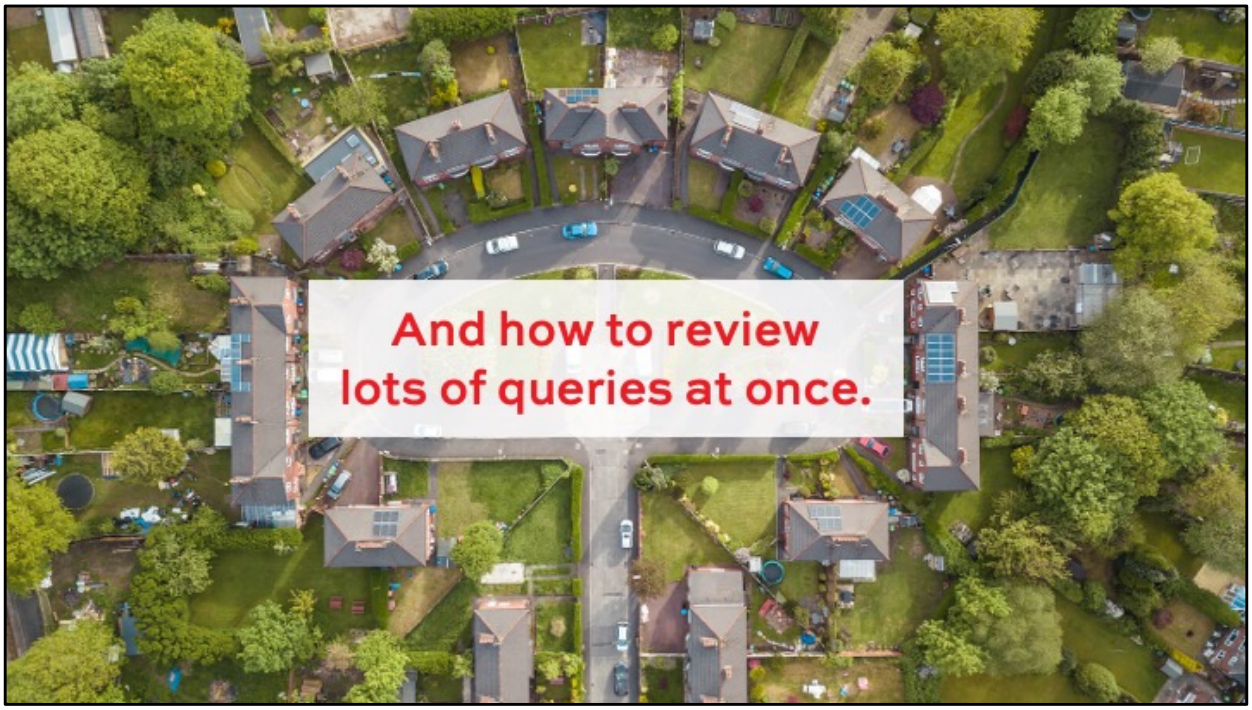
Supply store	Storage Engine
delivers the materials	fetches the pages

1.1 p21

"I want a 3-bedroom, 2-bath house"	SELECT * FROM dbo.Sales WHERE Location = 'NYC'
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.



Building our first blueprint

Trivial plans

1.1 p26

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 p27

Let's start with a blank slate.

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

1.1 p28

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 p29

He built the plan FAST.

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

100 %

Messages | Execution plan

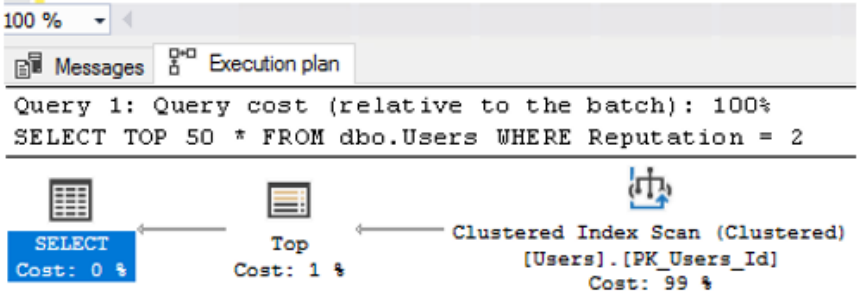
```
SQL Server parse and compile time:
   CPU time = 0 ms, elapsed time = 0 ms.

 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.

It's just not a fast plan.

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

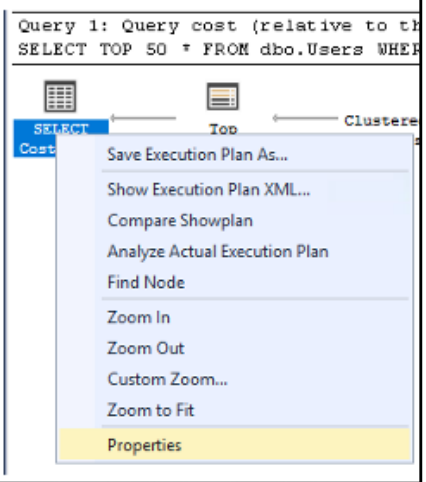
It's a clustered index scan: starting at the beginning, and reading until he finds 50 users with Reputation = 2.



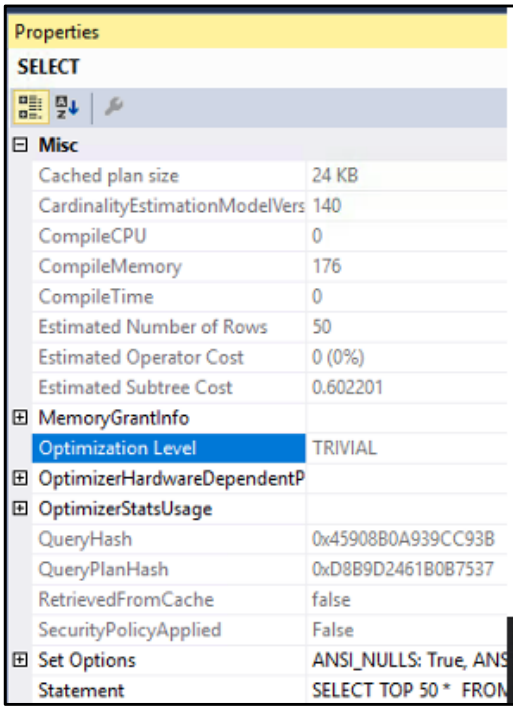
Why? Right-click on the select.

Right-click on the select operator, click Properties.

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



1.1 p32



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.

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 p34

Trivial queries/plans

Really easy to build a plan fast

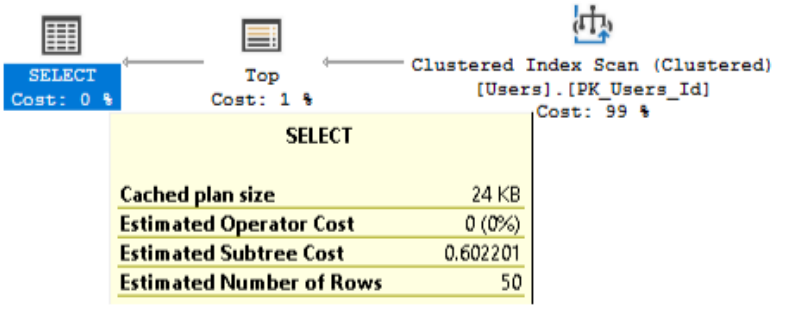
It just may not be a fast plan

- 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
```

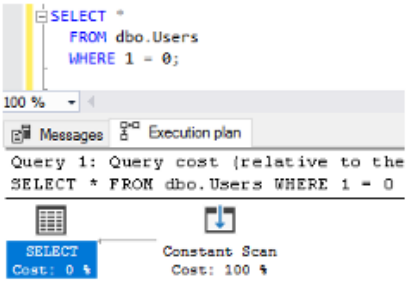


SELECT	
Cached plan size	24 KB
Estimated Operator Cost	0 (0%)
Estimated Subtree Cost	0.602201
Estimated Number of Rows	50

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 p36

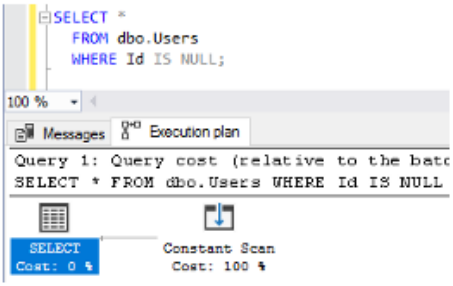
Contradictions get a trivial plan



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

1.1 p37

Non-nullable constraint check



Id is the primary key, non-nullable.
The more the architect (QO) knows,
the more work he can eliminate.

1.1 p38

Reputation is a non-nullable field

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

100 %

Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM dbo.Users WHERE Reputation IS NULL
```

1.1 p39

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 p40

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 p41

Let's upgrade to a doghouse

Full optimization (non-trivial queries)

1.1 p42

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.

Suddenly, a lot changes.

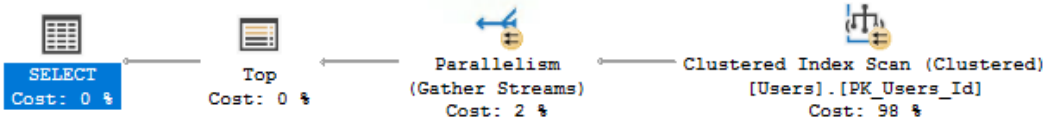
Now we get parallelism and a missing index request.

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

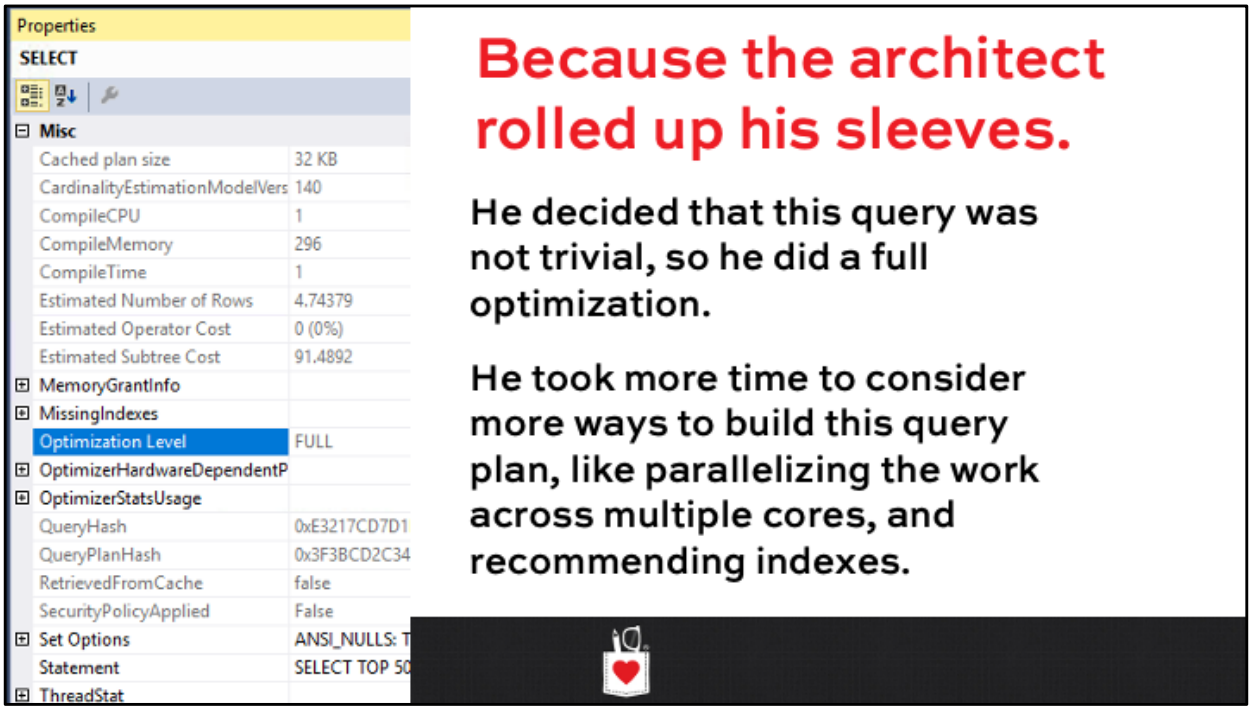
100 %

Messages | Execution plan

```
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
```



1.1 p



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...

1.1 p46

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
```

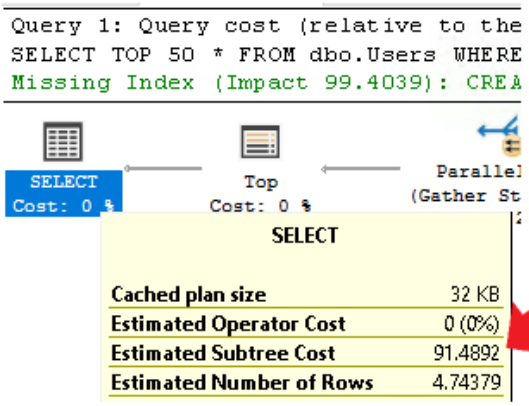
100 %

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.
```

Compile time was 0ms before,
but now it's 2ms.
(This can vary by a lot.)

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



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

1.1 p48

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 p49

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 p50

Estimated costs: almost useless.

Not good for:
comparing which plan was actually better.

Good for:
understanding why SQL Server chose one plan over another before execution started.

1.1 p51

Like how I fully did my chores

Full optimization
that isn't really full

1.1 p52

Query with a join

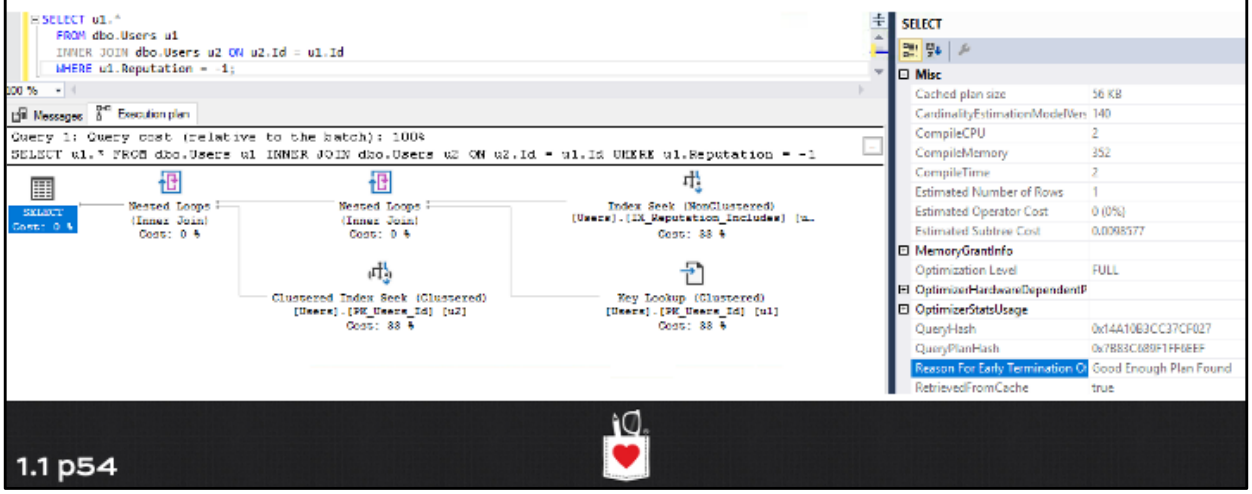
```
SELECT u1.*
  FROM dbo.Users u1
  INNER JOIN dbo.Users u2 ON u2.Id = u1.Id
  WHERE u1.Reputation = -1
```

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

1.1 p53

The query plan is simple.

And SQL Server generates it quickly:



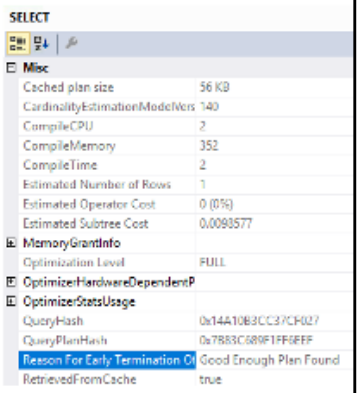
1.1 p54

Good enough.

Reason for Early Termination:
Good Enough Plan Found.

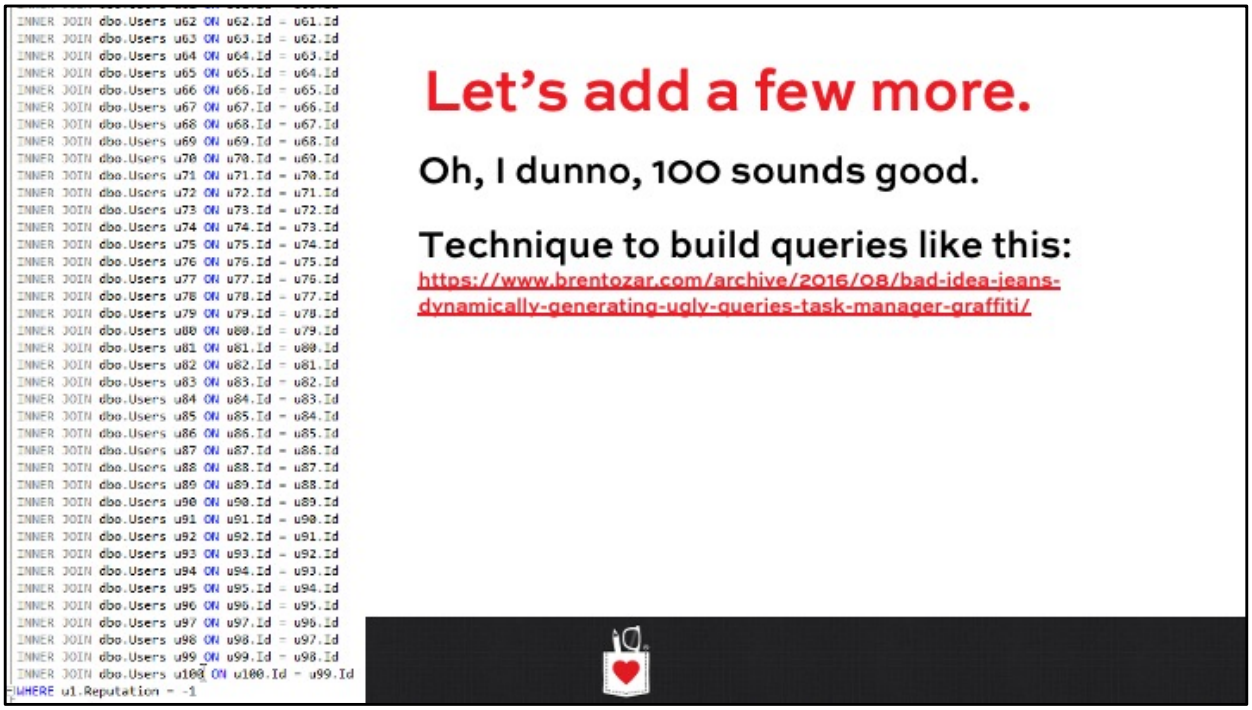
SQL Server spent hardly any CPU time or memory in order to build it.

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.

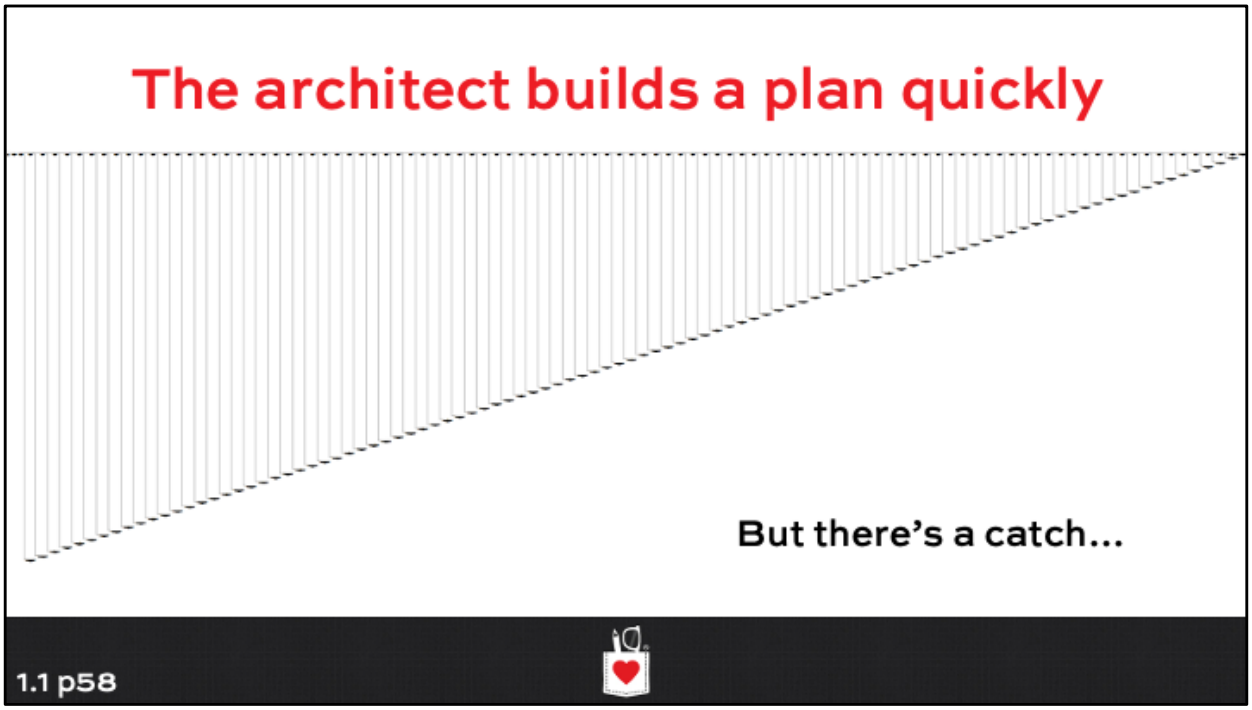


1.1 p55

1.1 p56



The architect builds a plan quickly



1.1 p58

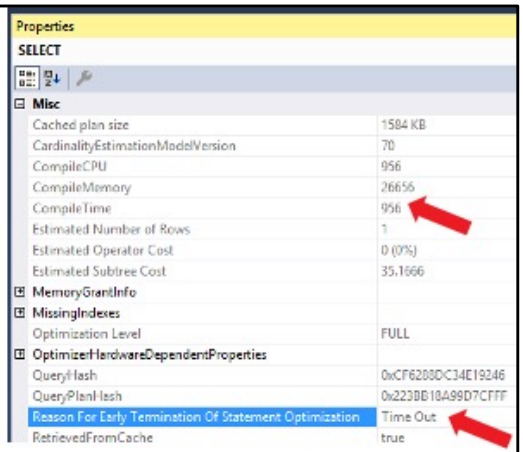
Time out.
Go sit in the corner.

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.



They can hit memory pressure, too

SQL Server needs memory to compile plans.

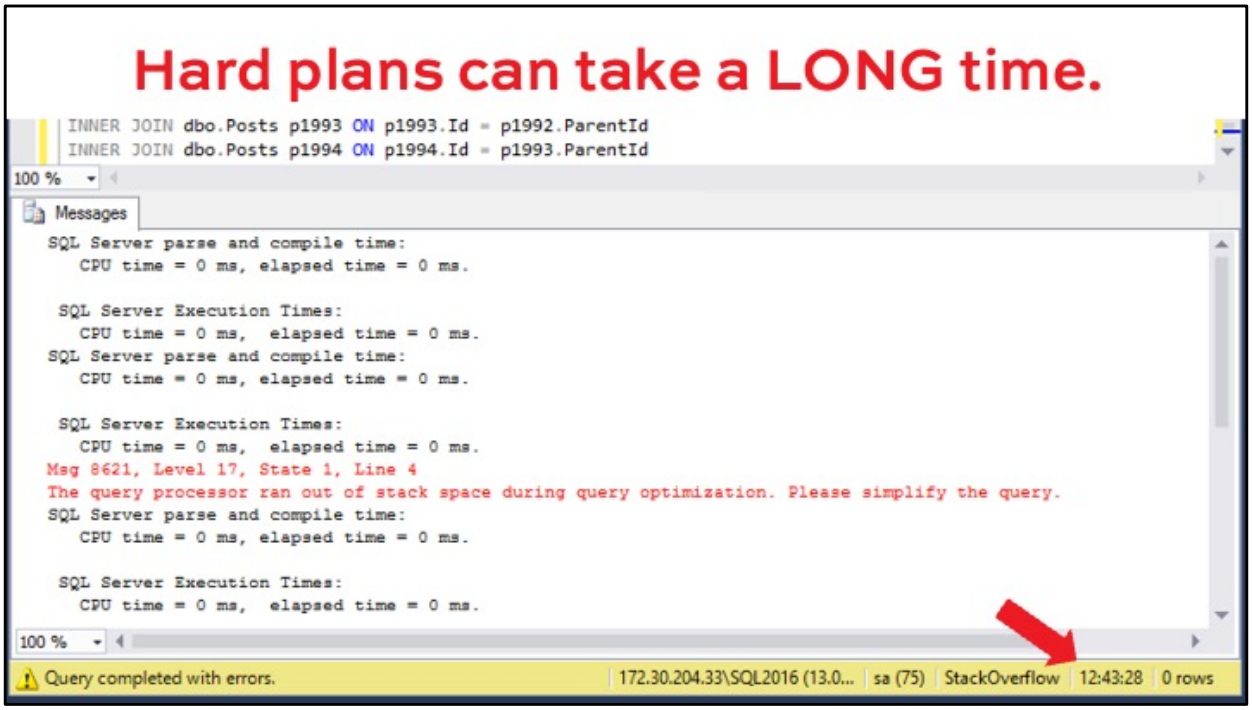
If many complex queries need to be compiled at the same time, they can be forced to wait on memory.

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

In situations like this, you're also more likely to see compilation timeouts.

1.1 p60

Hard plans can take a LONG time.



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 p62

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 p64

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

1.1 p65