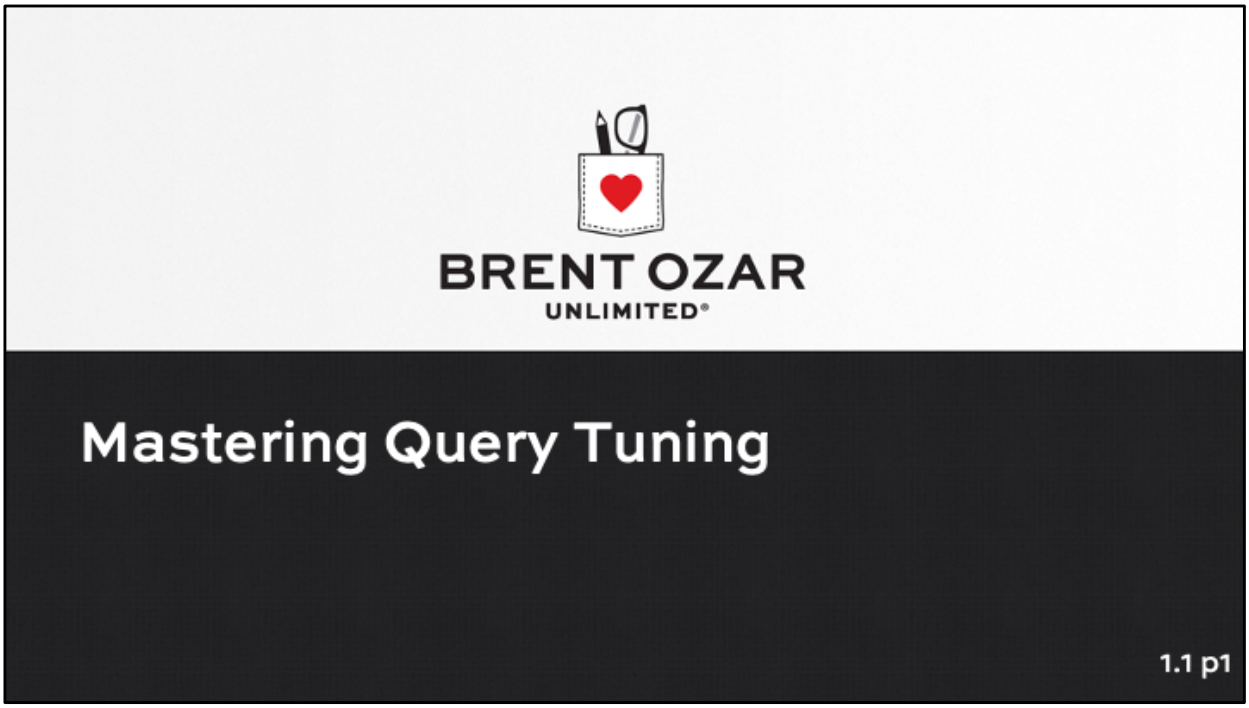


Introduce yourself in Slack:

	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

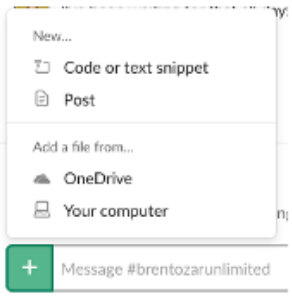
1.1 p2

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?

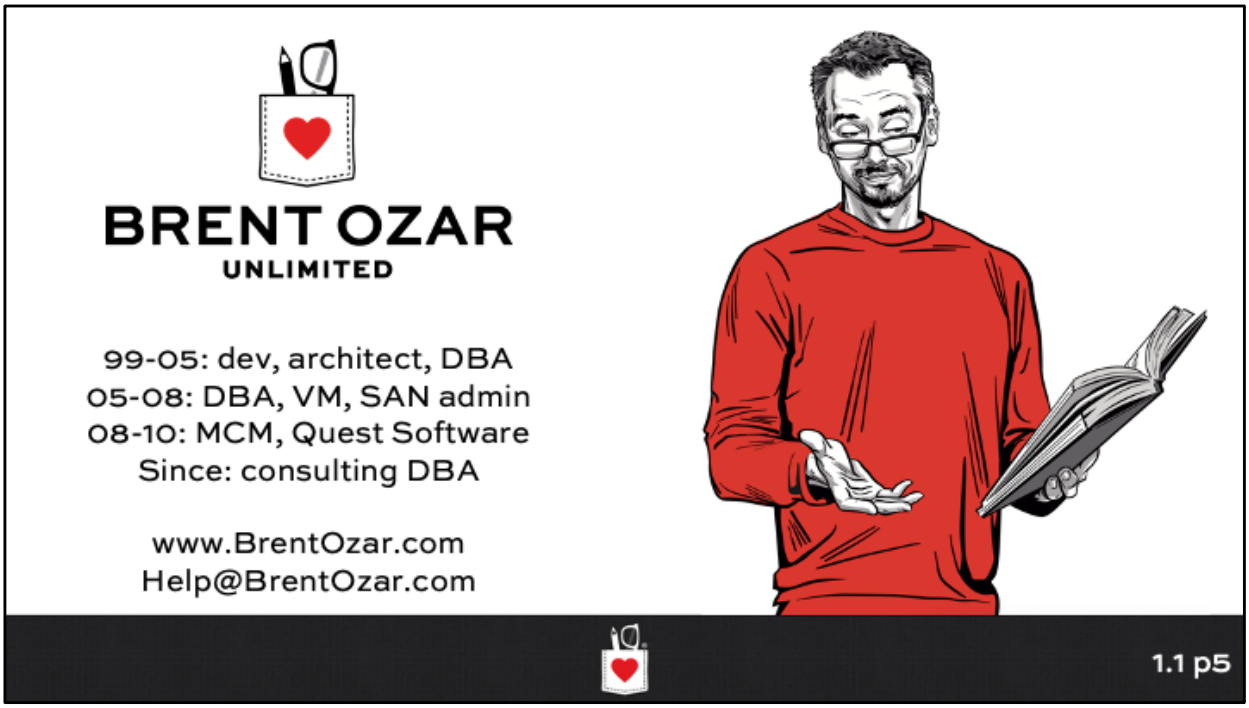
Wondering how other students solved it?

Wondering how *quickly* other students work?

Take a peek in the Slack room.

Otherwise, don't look (spoilers.)

1.1 p4



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 p10

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

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

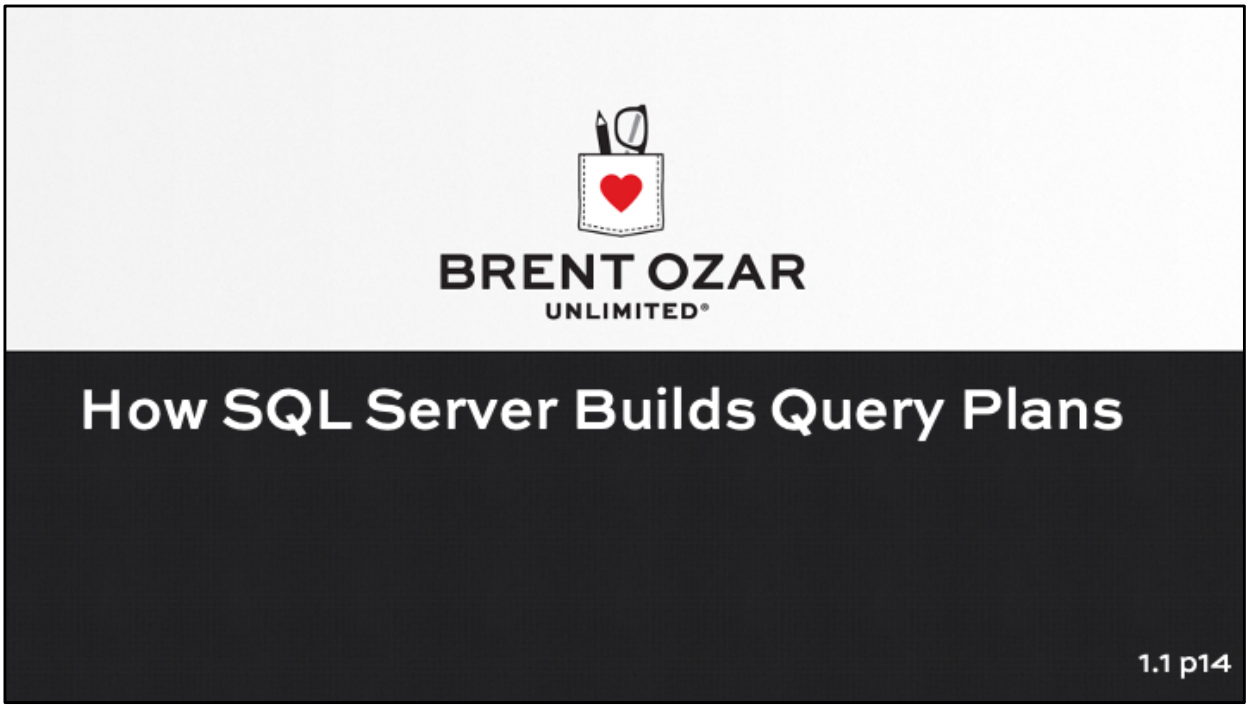
Red Gate SQL Prompt: better than IntelliSense

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

sp_WhoIsActive: better than Activity Monitor

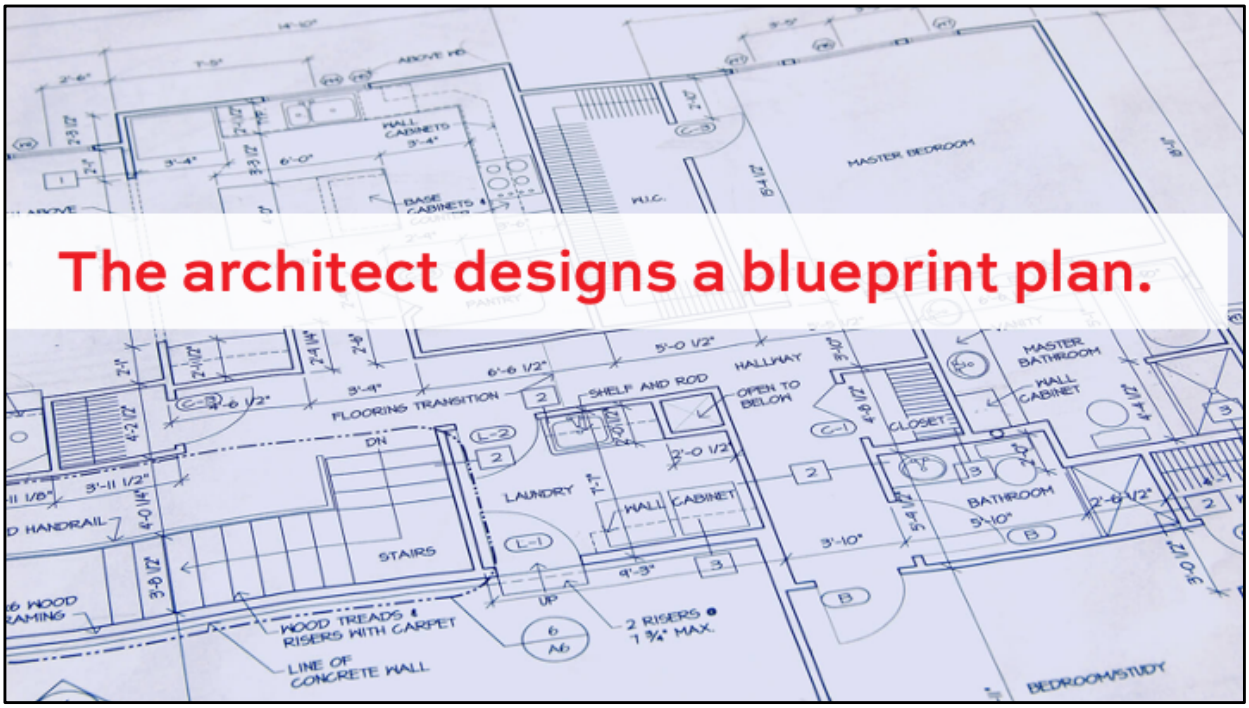
Azure Data Studio: the new SSMS for T-SQL

1.1 p13



Say you want
a custom house.

1.1 p15



"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

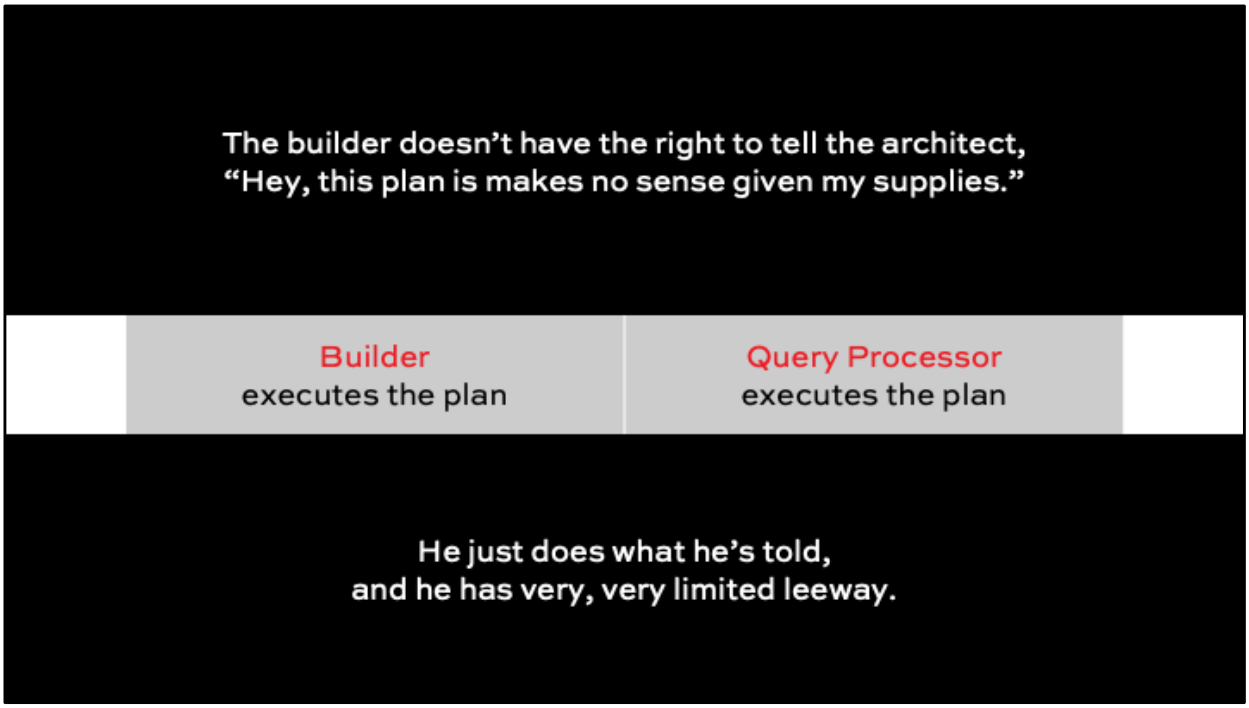
1.1 p20

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 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 p24

“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 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.

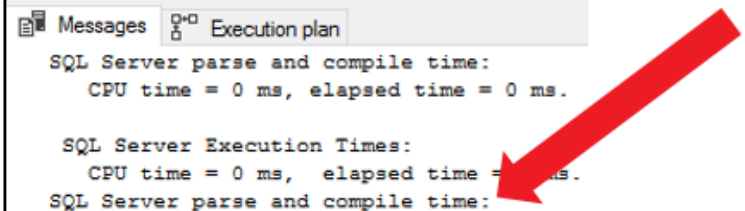
```
SELECT TOP 50 *
  FROM dbo.Users
  WHERE Reputation = 2;
GO
```

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.
```

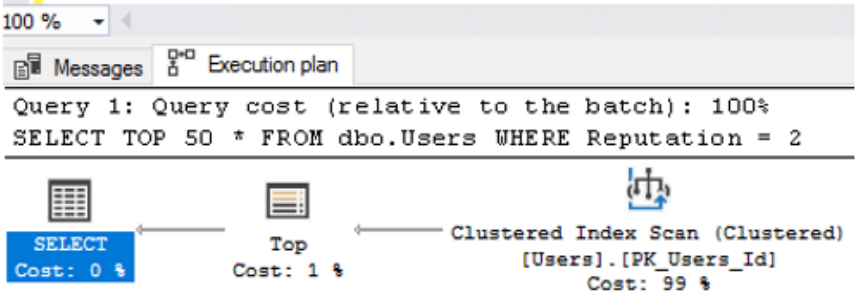
Statistics time shows less than 1 millisecond of time spent parsing and compiling.



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.

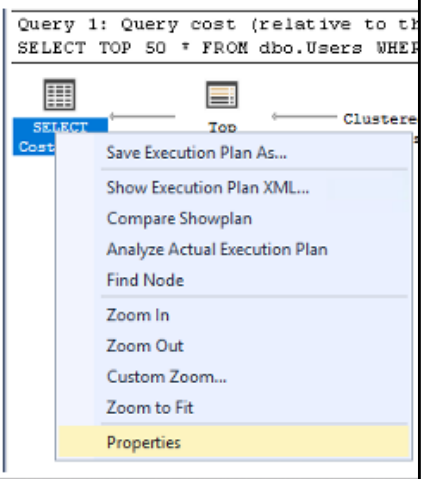


1.1 p34

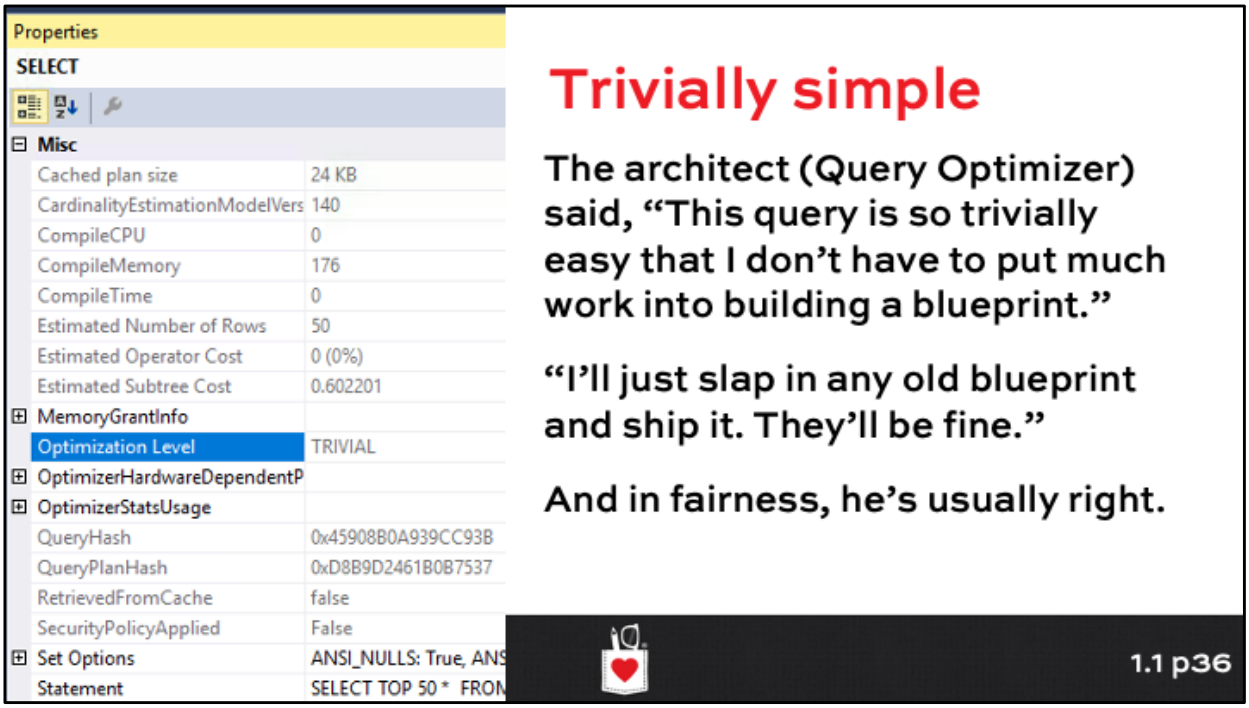
Why? Right-click on the select.

Right-click on the select operator, click Properties.

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



1.1 p35



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

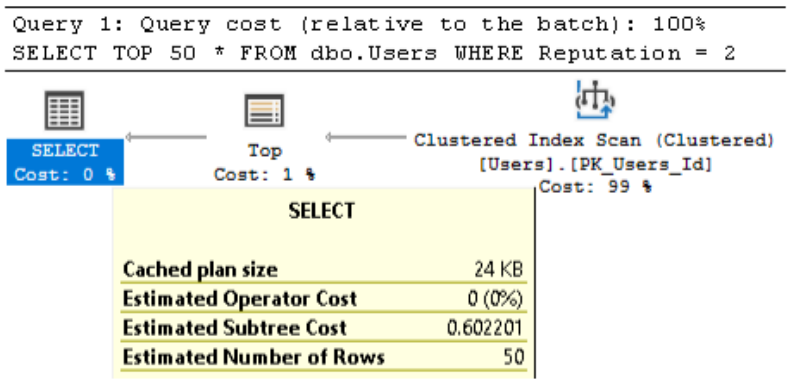
Really easy to build a plan fast

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.



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

```
SELECT *
  FROM dbo.Users
  WHERE 1 = 0;
```

100 %

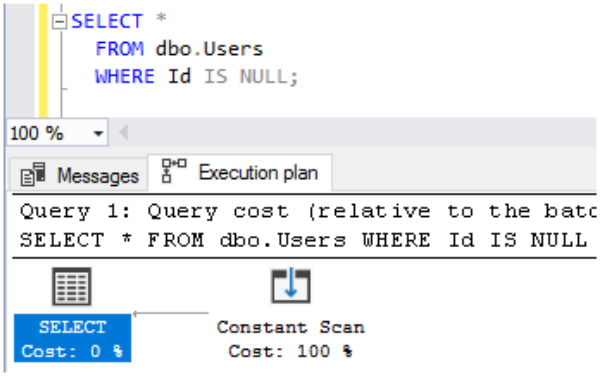
Messages | Execution plan

```
Query 1: Query cost (relative to the
SELECT * FROM dbo.Users WHERE 1 = 0
```

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

1.1 p40

Constraints help too



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!

1.1 p41

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
```

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.

Let's upgrade to a doghouse

Full optimization (non-trivial queries)

1.1 p45

Let's add just a little complexity.

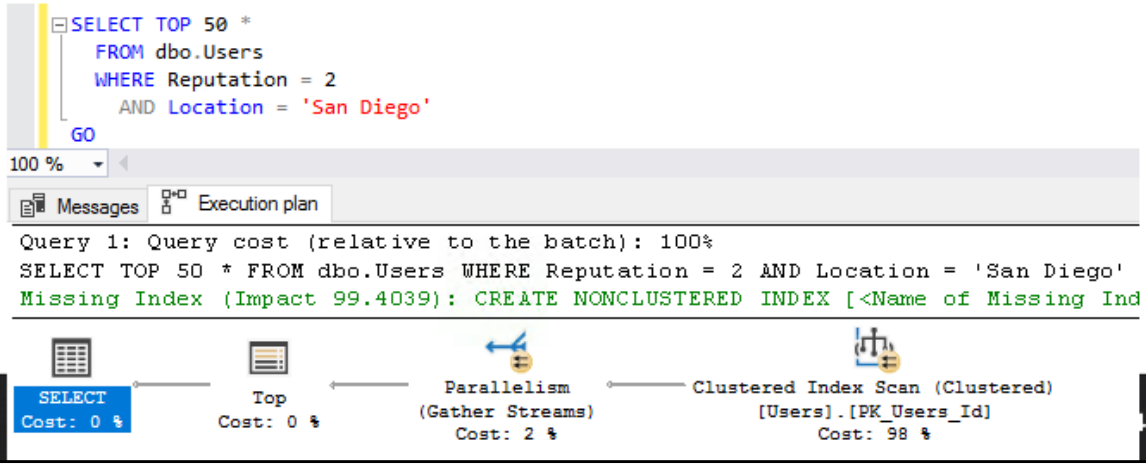
```
SELECT TOP 50 *
  FROM dbo.Users
  WHERE Reputation = 2
  AND Location = 'San Diego'
```

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.

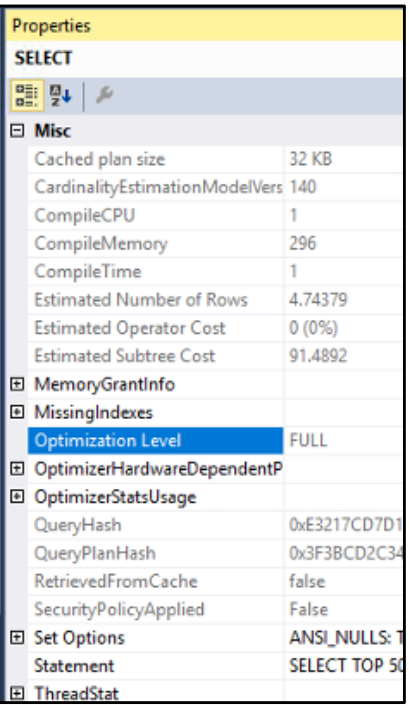
Now we get parallelism and a missing index request.



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.



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 p49

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 %

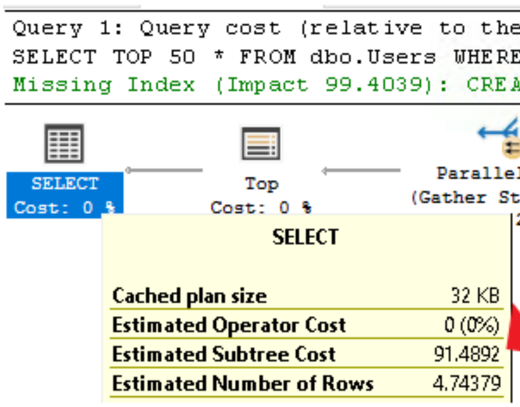
Messages | Execution plan

```
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.)

1.1 p50

The builder's cost is higher, but…



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

1.1 p51

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?

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

100 %

Messages | Execution plan

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

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

1.1 p55

What if the architect doesn't know?

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

100 %

Results | Messages | Execution plan

Id	AboutMe	Age	CreationDate	DisplayName	DownVotes

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.

1.1 p56

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

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

100 %

Results | Messages | Execution plan

```
Query 1: Query cost (relative to the batch): 100%
SELECT * FROM dbo.Users WHERE AccountId IS NULL
Missing Index (Impact 99.964): CREATE NONCLUSTERED INDEX [<Nar
```

1.1 p57

Query tuning is about...

Giving the architect as much help as possible to:

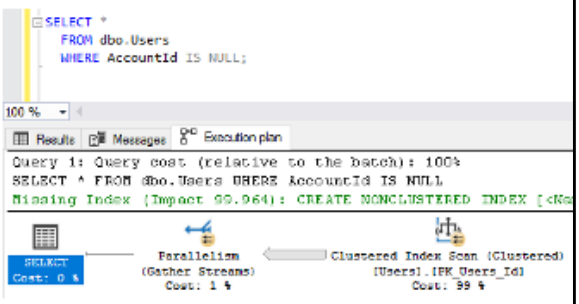
- Tell him as much as we can about the house, the end result that we want, and the supplies we have
- Lower his costs of building a plan
- Later, lower the builder's costs of building results with indexes, better hardware, settings

1.1 p58

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

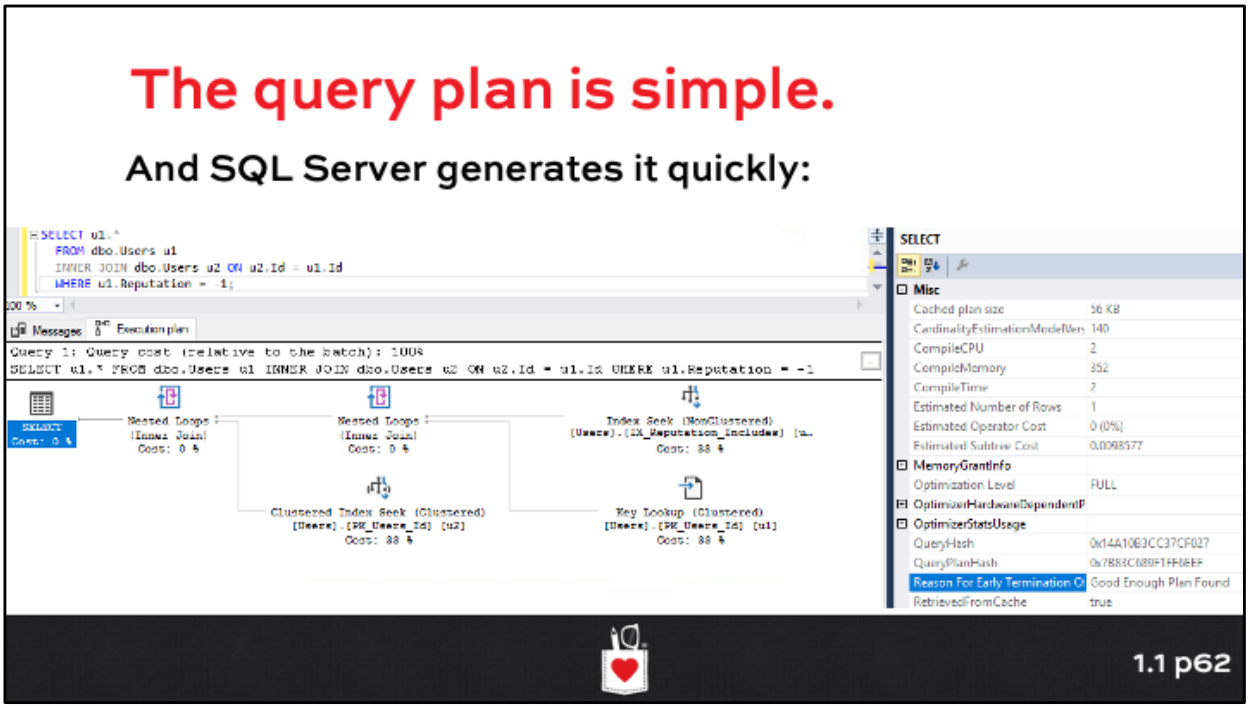
Let's add just one join.

```
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.

The query plan is simple.

And SQL Server generates it quickly:

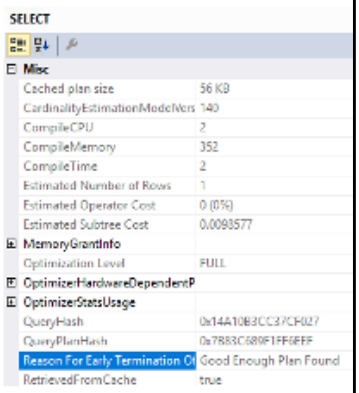


Good enough.

Reason for Early Termination:
Good Enough Plan Found.

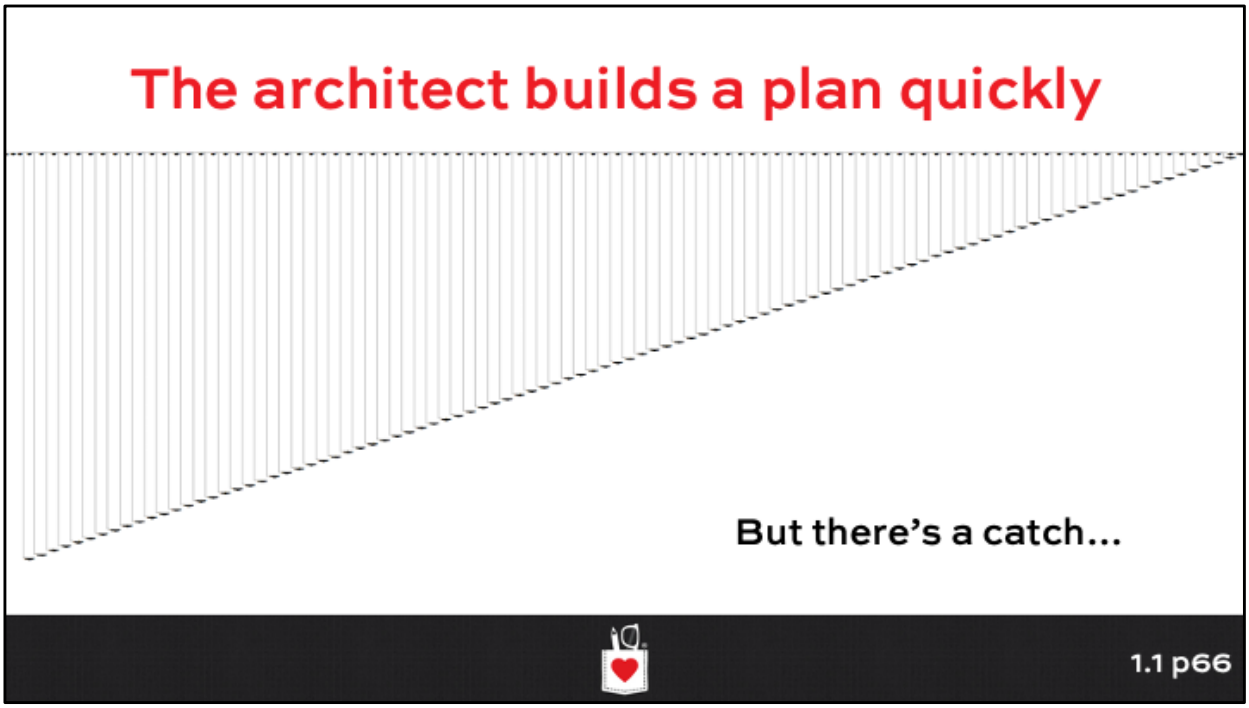
SQL Server spent hardly any CPU time or memory in order to build it.

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 p63

1.1 p64



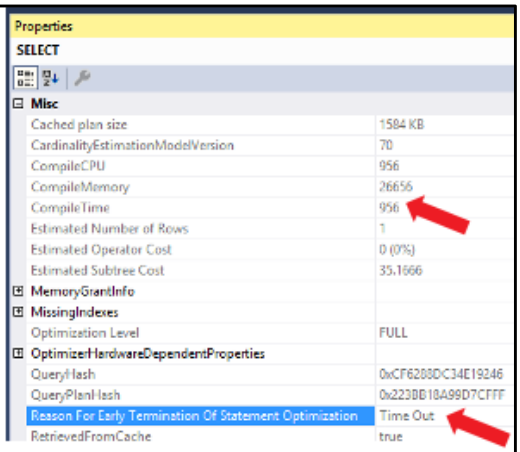
Time out.
Go sit in the corner.

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

At some point, it gave up.

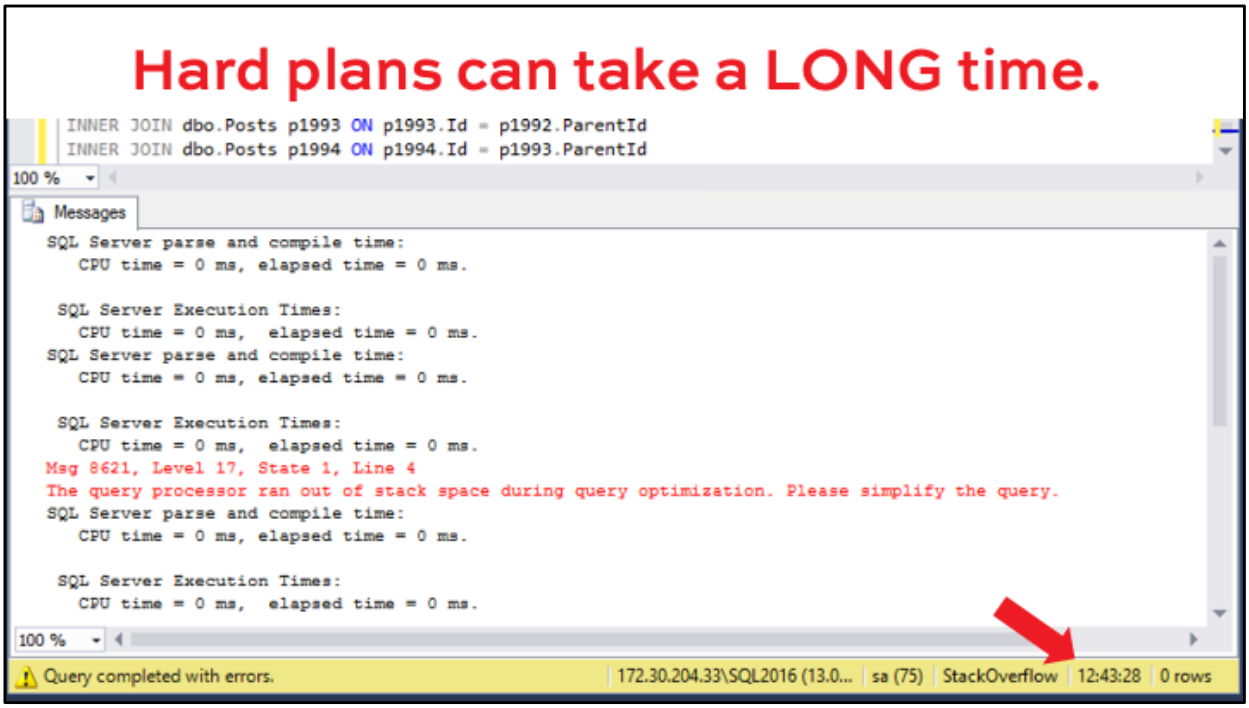
It ran out of time – even though we only spent about a second building the plan.

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



1.1 p67

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 p69

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

1.1 p72