

Behind the Scenes of SQL: Understanding SQL Query Execution

Taking A Step Back

SEATTLEDATAGUY
SEP 17, 2024

 77

 3

 10

Share



Here is something school probably didn't teach you about SQL

Or at the very least you likely forgot.

When you write a query, hit submit, and then run the query or that little triangle in DBBeaver...

...what exactly happens?

Sure, you likely understand that data is pulled from multiple tables, data is filtered, and aggregations occur.

But behind the scenes, what is going on?

How does SQL go from English into the lingua franca of data?

In this article we will answer that question.

Query Execution Components

Let's go through how your query gets parsed, and concepts like an execution plan.

My goal is to try to make this more exciting than my professor did in my [database](#) course.

Also, I am going to use the term SQL engine instead of RDBMS just because you have solutions like Trino or other similar solutions that are not a RDBMS.

Parser and Validation

The very first step in the process is lexical analysis(another term for this is tokenization). The goal of lexical analysis is to break down the SQL query into a series of tokens. Tokens are the smallest units of meaning in the SQL query, such as keywords, identifiers, operators, literals, and punctuation marks. This means your SELECT, FROM and WHERE for keywords and tables, and columns for identifiers, and <, = and <= for operators.

From there, you'll likely have two more steps, depending on the SQL engine you're using.

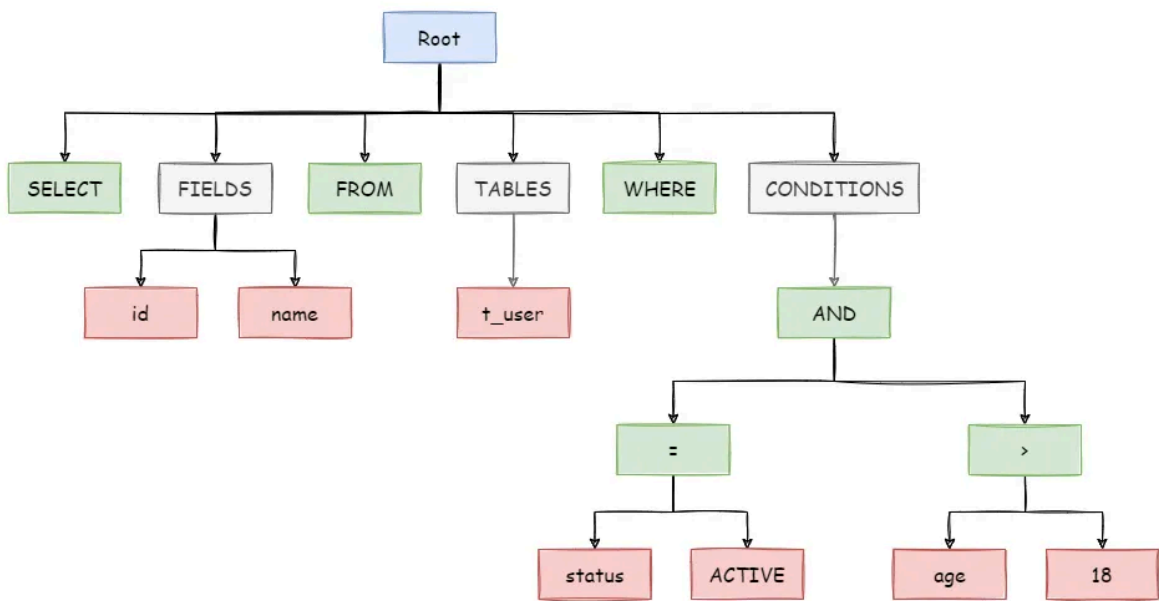
- **Syntax Check** - This ensures that the SQL query follows the correct grammar and structure as defined by the SQL language. It identifies syntax errors like missing commas, misplaced keywords, or unmatched parentheses and that keywords are in the proper order.
- **Semantics Check** - The SQL engine checks the query for semantic correctness. This involves checking whether the tables, columns, and other database objects referenced in the query exist and whether the operations are valid (e.g., ensuring data types are compatible).

Now as I said, this is dependent on what SQL engine you're using. For example, here is an expert from [Understanding MySQL Internals](#),

MySQL's parser, like many others, consists of two parts: the *lexical scanner* and the *grammar rule module*. The lexical scanner breaks the entire query into tokens (elements that are indivisible, such as column names), while the

grammar rule module finds a combination of SQL grammar rules that produce this sequence, and executes the code associated with those rules. In the end, a parse tree is produced, which can now be used by the optimizer. - [Understanding MySQL Internals](#).

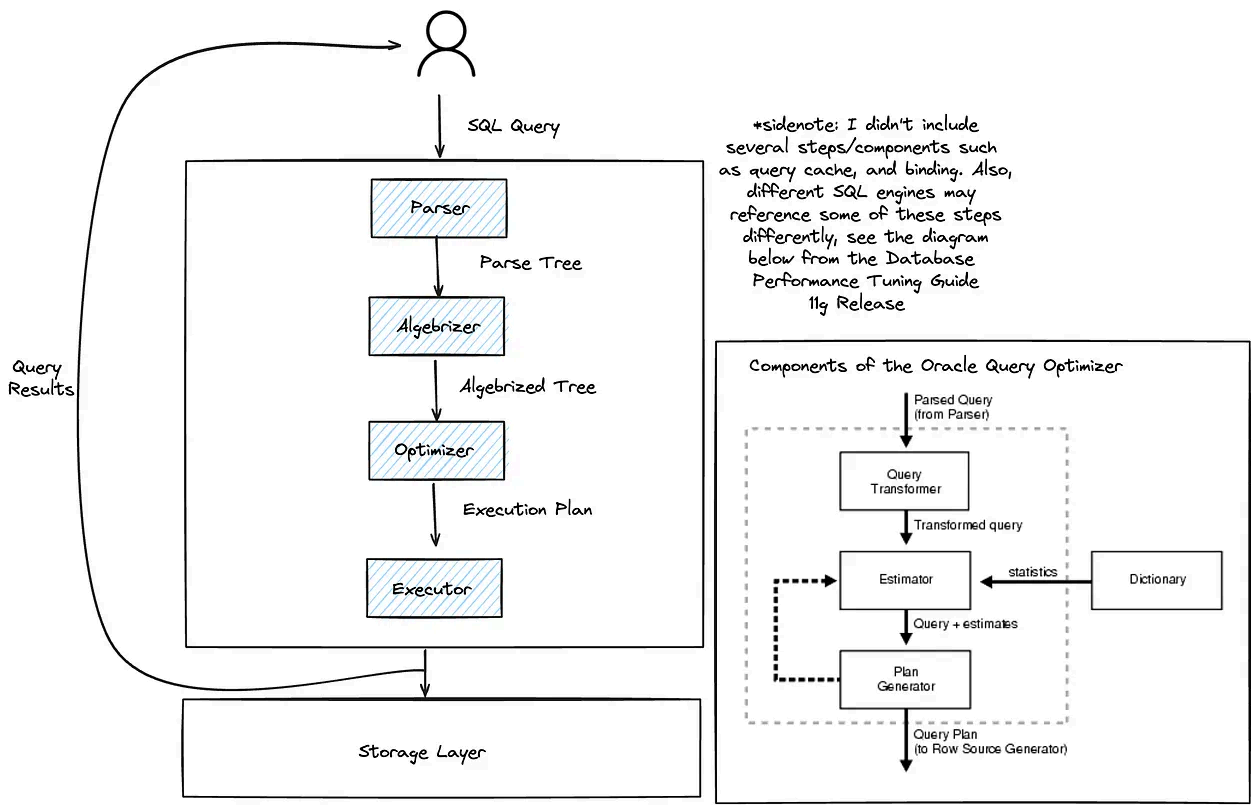
The end result will be something like the parse tree below.



Source

Now on to the next step!

Translator/Algebrizer/Query Transformer



Depending on what SQL engine you’re using will impact the next step in the process. For example, SQL Server has an Algebrizer component. Oracle has their Query Transformer, and still other query engines have other names for this component.

Circling back to Microsoft SQL Server and it’s Algebrizer, it’s responsible for converting the logical SQL query into an algebraic representation(e.g. [relational algebra](#))

Now if you're like me, the only time you've ever seen relational algebra was in your [database](#) course where we had to hand-write out relational algebra for far more complex queries. But if you're really curious about the relational algebra for your query, you can just use this [online tool](#).

I do want to point out a few quick follow-ups here.

- There can be multiple equivalent relational algebra expressions; these will be evaluated in a future step.
- Relational algebra is not always explicitly used in the implementation of every SQL database.

Optimizer

The next step in the process is for all of those logical operations plans to map to physical operations. Now sometimes, I find that the terms *logical* and *physical* get a little abstract.

Like [logical and physical data modeling](#).

The way you can think about it is that when we refer to logical, it's more like the blueprint of a house. Blueprints don't necessarily know all the implementation details.

Whereas physical references the actual construction process and building of the house. Now, you're dealing with the concrete details: the materials (bricks, wood, nails), the plumbing, wiring, and each room's actual dimensions. The focus is on how to practically build the house, ensuring that it stands strong and functions as intended.

And here is the thing; just because something might have seemed to be well planned out logically, sometimes someone puts a bolt in a weird corner that a carpenter can't actually hammer in.

Similarly, just because one logical operation flow might seem best, in implementation, it might not be.

To give an example, when you write the word "JOIN", the query engine has a few options on how to implement it. It could be a Nested Loops Join, Merge Join, or Hash Join.

The query optimizer will generate as many possible candidate execution plans in some constrained space. From there, the Optimizer selects the query which is having a low cost; this Optimizer uses the statistics about the data.

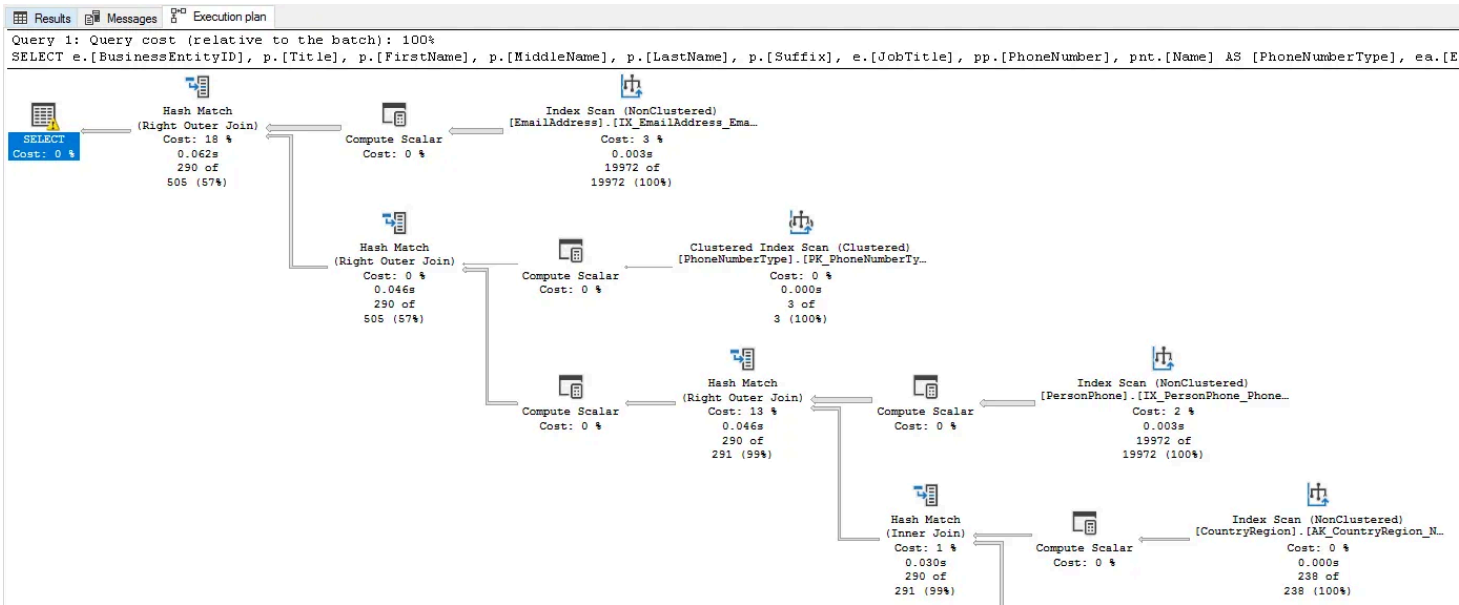
Execution Plans

Now I wanted to talk a little bit about Execution plans. After all, it's one of the few concepts here that you can likely actually examine.

Once you've worked with [SQL](#) for a while, you've likely heard of the term "execution plan." It might be somewhat dependent on what SQL engine you're working with but most of them allow you to see both the actual and estimated plans your query engine will/has taken to query your data.

Kind of like we have been discussing above.

For me, this was the first type I saw from Microsoft SQL Server:



Source

I used it a lot when I was trying to deal with queries that were taking longer than expected and trying to figure out if there was somewhere I could optimize it.

Also, for those who might be more familiar with a more tabular version of an execution plan, you might see something like the table below, which I pulled from [Oracles Optimizer](#).

Every SQL engine I have used has allowed me to use the [EXPLAIN PLAN](#) command to get this.

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				1140 (100)			
1	HASH GROUP BY		4	80	1140 (45)	00:00:14		
* 2	HASH JOIN		489K	9555K	792 (21)	00:00:10		
3	TABLE ACCESS FULL	PRODUCTS	767	8437	10 (0)	00:00:01		
4	PARTITION RANGE ALL		489K	4300K	741 (17)	00:00:09	1	16
5	TABLE ACCESS FULL	SALES	489K	4300K	741 (17)	00:00:09	1	16

So if you’re looking for a practical takeaway from this article, here it is!

Execution

Now that the optimizer has selected the optimal query. The execution plan will be sent off to the query execution engine. Now perhaps in the future, someone will build an agnostic Optimizer that automatically goes through and looks at [Presto/Trino](#), [Snowflake](#), DuckDB, Spark, etc., and picks the best engine for the job. Depending on whether you’re looking for cost-effective or fast (honestly, perhaps this already exists and if it does, I’ll add it to this article if someone wants to send it over). However, I imagine most individuals will want to be able to personally select which engine is used when vs. having the Optimizer select it.

Once the execution engine has run the execution plan, it will return the results (assuming it doesn’t run out of memory). And now we are back to where you likely are accustomed to thinking of a query.

SQL -> Data.

But hopefully, we’ve filled in a few of the steps in between!

Side note: There are a few other steps and components that I didn’t include such as security and the [query cache](#).

Why I Think It’s Important To Know

- Be Better At Optimizing Queries - If you understand how the optimizer functions, you’ll likely be able to write better queries or a the very least improve the performance of queries

that are slow. Why? Because you'll eventually learn about the different types of joins (and I don't mean left, inner, cross, etc) and understand why your database might pick them. Don't worry, I will be covering this topic in an upcoming article!

- **Start-ups And SQL Replacements** - If you're out there trying to build a Text to SQL start-up, a cost optimization start-up or [trying to replace SQL](#). I'd argue you need to understand what is going on underneath the hood. If your understanding of how SQL functions is limited to the high level queries themselves, then you won't be able to understand the technical space at a granular enough level to see the real causes of your technical challenges.

Not All Query Engines Are The Same

Most query engines have most of the steps discussed above, perhaps by a different name. So I wanted to share a few articles that discuss some of these concepts more in depth and in some cases for more specific engines. I also included Snowflake's white paper; it's less about how a query is parsed and more focused on how to run a distributed query.

- [I spent 6 hours learning how Apache Spark plans the execution for us](#) by [Vu Trinh](#)
- [Database Performance Tuning Guide 11g Release](#)
- [Queryparser, an Open Source Tool for Parsing and Analyzing SQL](#) by Uber
- [How A Query Engine Works](#)
- [The Snowflake Elastic Data Warehouse](#)

SeattleDataGuy's Newsletter is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.

Going Back To The Basics

In my last article, I discussed digging [deeper into the basics](#). So I wanted to go back and focus on something all of us data people use but probably take for granted. After all, I just hit submit and the data comes back.

Every once and a while, as much as I like to harp on business value, I believe it is important to stop and dig into a subject. Enjoy learning about how things work. Even if it's just for the sake of it.

Also, because there are points where understanding things at lower levels does start to provide value, you can start to make more decisions based on more compact areas of knowledge.

As always, thanks for reading!

Join Us For The Data Engineering And Machine Learning Summit 2024!



Join us on October 3rd and 4th for the Data Engineering And Machine Learning Summit 2024.

Immerse yourself in a confluence of data engineering and data science where the spotlight is on those who truly get their hands dirty - the practitioners.

Practitioner-First Philosophy

We believe in the power of practice. At the summit, you won't just hear high-level theories, but practical insights, solutions, and real-world stories. Our meticulously curated lineup includes dozens of speakers, all of whom are leading practitioners in the world of data.

Cutting-Edge Innovations

Explore the latest tools, platforms, and methodologies that are revolutionizing data engineering and data science.

Who Should Attend

Whether you're an aspiring data engineer, an experienced data scientist, or simply a data aficionado, the summit promises a wealth of knowledge, inspiration, and networking opportunities that you can't afford to miss.

Special Thanks to [Databricks](#) for sponsoring DEML 2024 and [Accel Events](#) for making it so easy to manage the event.



databricks

Join My Data Engineering And Data Science Discord

If you’re looking to talk more about data engineering, data science, breaking into your first job, and finding other like minded data specialists. Then you should join the Seattle Data Guy discord!

We are now well over 7000 members!

Join My Technical Consultants Community

If you’re a data consultant or considering becoming one then you should join the Technical Freelancer Community! We have over 700 members!

You’ll find plenty of free resources you can access to expedite your journey as a technical consultant as well as be able to talk to other consultants about questions you may have!

Articles Worth Reading

There are 20,000 new articles posted on Medium daily and that’s just Medium! I have spent a lot of time sifting through some of these articles as well as TechCrunch and companies tech blog and wanted to share some of my favorites!

OLTP Vs OLAP – What Is The Difference

	OLTP	OLAP
Access Patterns	The access pattern of an OLTP system is characterized by a high volume of small, frequent transactions that require fast response times and concurrent access by multiple users.	The access pattern of an OLAP system is characterized by fewer, larger, and more complex queries that require longer response times but provide greater analytical capabilities.
Data Model	OLTP systems typically use a normalized data model, where data is organized into multiple tables and relationships. Normalization reduces redundancy and ensures data consistency.	OLAP data models tend to be more denormalized. This should reduce the number of joins required and generally make it easier for an analyst to understand how to write their query.
Size	OLTPs tend to be smaller in terms of memory since they might only hold the current data and not historical changes.	OLAPs will be larger as they will store historical data as well as data from multiple systems.
Performance Needs	OLTPs need to have fast response times. Otherwise, end-users would be concerned that their tweet didn't go through	OLAP systems can get away with being a little slower. But if your dashboard is taking 10 minutes, DM me.

If you’re relying on your OLTP system to provide analytics, you might be in for a surprise. While it can work initially, these systems aren’t designed to handle complex queries. Adding databases like MongoDB and CassandraDB only makes matters worse, since they’re not SQL-friendly – the language most analysts and data practitioners are used to. Over time, these systems simply can’t keep up with the demands of performing analytics.

Why does this matter? After all, data is data, right? Well, there’s a big difference between online transaction processing (OLTP) and online analytical processing (OLAP). Each has its own unique requirements in terms of software and design. That’s why solutions like Teradata and Vertica have played such a large role in many enterprises. In fact, Teradata was one of the first data warehouse systems to handle a TB of data for Walmart.

But here’s the thing: Why go through the trouble of duplicating your data and creating a separate analytics system? In this article, we’ll explore the reasons why you need to develop an analytical system if you want to answer your business questions effectively.

[Read More Here](#)

Data Model Building Blocks (Intro)

By [Joe Reis](#)

I often find that people working with data don’t understand its building blocks. Data is frequently treated as *something* that’s stored, queried, and cobbled together. Consequently, data models are often an unintentional side effect or artifact created without deliberate thought.

Before we build a data model, we must grasp the critical building blocks forming it.

[Read More Here](#)

End Of Day 143

Thanks for checking out our community. We put out 3-4 Newsletters a week discussing data, tech, and start-ups.




77 Likes · 10 Restacks


Discussion about this post

Comments

Restacks





Write a comment...




Mehran Sep 17


Thank you! It would be really valuable if you could write an article explaining how the indexing mechanism works, for example, in PostgreSQL.

 LIKE (1)

 REPLY

 SHARE

...



Ramona C Truta Ramona C Truta Sep 17

Excellent post, Ben!


After being a TA marking hand-written RA queries, as a teacher instituted a LaTeX-typed one. I provided the students with a small stub with everything they needed, and it worked really well.


I would also include Alex Xu's diagram, or yours if you have it, on how a typical query gets evaluated, the one with the blocks etc.


Not knowing data design and not knowing these deep fundamentals on query evaluations result in so much overhead cost. It's really great you're bringing this into focus!

I do know of a startup working on using AI to optimize queries. I can send that info separately, we had a chance to talk at a low-key event.

Please include index-only queries in your index post :)

 LIKE

 REPLY

 SHARE

...

1 more comment...