DATA⁺AI SUMMIT BY S databricks

Benchmarking Data and AI Platforms

Shannon Barrow, Lead Solutions Architect, Databricks Joe Harris, Sr Software Engineer, Databricks



DATA⁺AI SUMMIT

©2024 Databricks Inc. — All rights reserved

Benchmarking Data and AI Platforms

Joe Harris, Sr Software Engineer



Shannon Barrow, Lead Solutions Architect



Benchmarking Data and AI Platforme

How Much You Bench, Bro?

Shannon Barrow, Lead Solutions Architect



- Joined Databricks in March 2019
- Previously: Principal Innovation and Thought Leadership, Accenture Applied Analytics
- Despite overarching benchmark discussion, I may:
 - Put extra focus on TPC-DI
 - Put on my Databricks hat for short segments

Today's Scope

Focus on LAKEHOUSE and AI Related Benchmarks

Lakehouse/OLAP

ML/Gen Al



- Primarily TPC but others will be mentioned
- Suggestion: view the following benchmarks through the lens of a full end-to-end Lakehouse architecture
- How can an organization get a "full picture" of an end-to-end TCO?

- Highlights and challenges
- Focus on Gen Al
- What to benchmark?
- Lessons learned from Mosaic

Why Benchmark?

Level-setting on the Value and Limitations of Benchmarks

Lies, Damned Lies, and Benchmarks

This is Part 1 of a <u>5-part series on DW/Analytics Performance Benchmarking</u> and its use in **non-traditional** ways, including DW/Analytics Engineering and Maintenance.

Part 1. Background

Riffing on an old quote attributable (though apparently not originally) to Mark Twain [1] ...

There are three kinds of lies: lies, damned lies, and benchmarks [sic]

We've all seen one-upmanship and "fastest" claims from vendors in the computer technology space using industry-standard benchmarks. From the fastest computers, to the fastest storage systems, to the fastest database systems, there are numerous blogs and papers touting the dominance of one platform or approach over another. In some cases, usually when it suits them, vendors adhere to benchmark specifications and standard reporting requirements. In others, only portions of benchmarks are run and often selectively referenced and co-opted for the desired message. Such is the state of our industry and it will likely not change.

This is the case in the world of Data Warehousing and Analytics. The Transaction Processing Council (TPC) arose out of a need for standard performance metrics and well-defined configuration and pricing guidelines across a number of database vendors and platform offerings. Initially focused on OLTP systems, the organization <u>broadened its work</u> to data warehouse/analytics workloads and other related IT infrastructure areas.

https://rethinkio.com/lies-damned-lies-and-benchmarks/



Why Benchmark?

Level-setting on the Value and Limitations of Benchmarks



Level playing field for all platforms

- Standardization and repeatability
 - To conform to the same practices
 - To conform to common industry operations, use cases, input/output, and scale
 - Industry "agreed upon" testing heuristics
- "Official" submissions



Can be hard to believe any results

- Potential for:
 - Cheating
 - Bias
 - Abuse
- Slow pace of modernization

Lakehouse Benchmarking

TP(P)C - The Ubiquitous Standard

Most prevalent and well-known

Transaction Processing Performance Council

- Formed in 1988
- Benchmarks across multiple domains
 - Decision Support (OLAP)
 - TPC-DI, TPC-H, TPC-DS
 - Transaction Processing (OLTP) These are the control of the contr
 - "Big Data"
 - TPC-HS, TPC-BB
 - Virtualization
 - TPC-V, TPC-HCI
 - Internet of Things (TPC-IOT)
 - AI (TPC-AI)

Active Benchmarks		
Benchmark/Document	Current Version	
TPC-C	5.11.0	
TPC-DI	1.1.0	
TPC-DS	3.2.0	
TPC-E	1.14.0	
TPC-H	3.0.1	
TPCX-AI	1.0.3.1	
TPCX-BB	1.6.2	
TPCX-HCI	1.1.9	
TPCX-HS	2.0.3	
TPCX-IOT	2.1.0	
TPCX-V	2.1.9	

Other OLAP Benchmarks

Are TPC Benchmarks The Only Game in Town?

- SSB (Star Schema Benchmark)
 - Designed to measure the performance of databases in a star schema setup
 - Simpler than TPC benchmarks but focused on specific aspects of OLAP querying
- ClickBench, The No-Join Benchmark
 - Focuses on workloads without joins
 - Simulates scenarios common in clickstream analytics

Lakehouse Focus - 10k Foot View

TPC fragments the Lakehouse Architecture into separate benchmarks

- There is no SQL consumption in TPC-DI
- No transformations in TPC-H or TPC-DS
 - Most "unofficial" results even skip the data loading step altogether



ETL: TPC-DI





TPC-DI: Data Integration

The Ingestion and ETL One

- ZERO official submissions
 - Was Databricks first to code it?
- I originally presented completed benchmark at DAIS 2022
 - Not **submitted** (not for lack of trying)
- Extremely short TL;DR:
 - ZERO code given
 - Ingest: TXT, CSV, XML
 - Transform: based upon 100+ pages of business rules
 - Load: all 3 medallion layers

TPC-DI

TPC-DI is a benchmark for Data Integration

Historically, the process of synchronizing a decision support s ETL tools. Recently, ETL was replaced by the more comprehe a unified data model representation and loading it into a data and loads it into a data warehouse. The source and destinatio

Specification

- The current TPC-DI specification can be found on the **TPC** More Information
- TPC-DI: the first industry benchmark for Data Integration

Results

There are no TPC-DI results published yet.

Is the TPC-DI Valuable?

A Frustrating Benchmark that hides some real valuable insight



×

The **best** official ETL benchmark available

The worst official ETL benchmark available

Is the TPC-DI Valuable?

A Frustrating Benchmark that hides some real valuable insight



The **best** official ETL benchmark available

- Robust even though built for legacy DWs
- Business rules make for realistic test
 - Though it suffers DQ issues with data generator at higher scale factors
- Flexibility in how rules are coded
 - Allows practitioners to optimize to their platform



The worst official ETL benchmark

- ayaisable one aware of another "official" ETL benchmark?
 - No "official" submittals
 - Scoring metrics are confusing and do not even allow for cloud platforms
 - No provided code means it is extremely frustrating to attempt this benchmark
 - Made worse by long, confusing business rules

Excuse Me While I Digress...

I will speak longer on TPC-DI than originally planned

Why?

A) Because me

2) Because everybody

D) Because Joe Abandoned Us

From Initial Implementation to Scorching Performance Today

This is a slide from the DAIS 2022 Session in which we announced the TPC-DI had been finally implemented

Photon price per billion rows:

\$1.51

Traditional Notebook Workflow Results
Performance Dashboard

- The TPC-DI has a rather confusing benchmark algorithm
- Simplified: TCO approach based on cost per row processed

These were the best performing combinations with On-Demand Pricing:

Run Time (minutes)	Worker	Total Costs	Price per Billion Rows	Photon	Graviton
36.4	m6gd.8xlarge	\$23.28	\$1.44	No	Yes
24.0	m6gd.4xlarge	\$24.47	\$1.51	Yes	Yes

DATATAI • SPOT instances drops this price to as low as 85 CENTS!

From Initial Implementation to Scorching Performance Today

In April 2023, we published a blog, <u>How We Performed ETL on One</u> <u>Billion Records For Under a Dollar,</u> to tout the power and TCO of **Delta Live Tables** on this benchmark.

Photon price per billion rows:



Cost and Time to Process 1 Billion Rows in TPC-DI (Complex Enterprise ETL Benchmark)

\$0.96



From Initial Implementation to Scorching Performance Today

This video compiled in **September 2023** compares a **dbt** implementation against CDW competitors

Photon price per billion rows: **\$0.73**



©2024 Databricks Inc. — All rights reserved

A Prominent CDW Was missing...

The Truth is Out There...



A Prominent CDW Was missing...

Some can handle large file sizes, others can't

- We tried benchmarking the other CDW but found it wholly intractable at larger scale factors since it is the only one that is unable to split raw files natively
- We weren't the only ones to notice

From Initial Implementation to Scorching Performance Today

©2024 Databric

- Last month (May 2024) we expanded the benchmark to test non-DWs– which required moving from dbt
- Since AWS has been improving EMR over the last few years this became the obvious first choice for non-dbt tests

Task name	3:51:57 PM 3:57:28 PM	Job ID 701634745375479
ingest_BatchDate	[14.2s]	
ingest_DimDate	[13.7s]	1039057252964353 匠
Silver_DimBroker	20s	Launched
ingest_DimTime	[<mark>14.2s</mark>]	Manually
ingest_FinWire	1m 20s	Started
ingest_ProspectIncremental	1m 5s	00/07/2024, 03:51:57 FM
ingest_StatusType	<mark>[13.9s]</mark>	Ended 06/07/2024, 04:02:40 PM
ingest_TaxRate	<mark>[13.9s]</mark>	Duration (i)
Silver_DimCustomer	28.25	10m 44s 9 Graviton
Silver_DimAccount	40.5s	Queue duration (16-core workers
Gold_FactCashBalances	→ <mark>2m 3s</mark>	- = 144 cores
Silver_Prospect	→ <mark>40.9s</mark>	Status
ingest_TradeType	[<mark>13.8s</mark>]	
ingest_industry	[14.2s]	Lineage (†) 29 upstream tables, 21 downstream tables
Silver_DimCompany	10.6s	
Silver_DimSecurity	26s	Compute
Gold_FactWatches	3m 5s	Shannon Barrow's Cluster
Silver_DimTrade	5m 2s	Driver: m7gd.2xlarge · Workers: m7gd.4xlarge · 9 workers · On-demand · DBP: 15.2 · - photon-scala2.12
Silver_FactHoldings	→ <mark>1m 16s</mark>	auto
Silver_Financial_CIK	→ <mark>46.6s</mark>	View details Spark UI Logs Metrics
Silver_Financial_CONAME	→ <mark>49</mark> s	
Gold_FactMarketHistory	↔ 8m 23s	Job parameters 🕞

DATA⁺AI SUMMIT

From Initial Implementation to Scorching Performance Today

- 2.2x faster on ¹/₄ the cores as 2 years • ago!
 - 24 minutes down to 10.75 minutes
 - 576 cores down to 144 cores •
- Improvements from: \bullet
 - PHOTON shifting into overdrive •
 - Gradual code and orchestration • improvements
 - No code is provided optimize code to match the • platform
 - Newer generation VMs •

DATA⁺AI SUMMIT

- Other platform enhancements •
- 1 year ago: "A billion rows for under a \bullet dollar"



Today: as low as **20**¢ on spot (27¢ on-



So TPC-H... this still a thing?

The "OG" OLAP benchmark

- Released in April 1999 to "fix" issues with TPC-D
- However, the following year TPC moved to develop a new decision support benchmark to better reflect modern OLAP implementations
- In January 2012, the TPC-DS was released begging the question why the TPC-H is still used by organizations

How is TPC-H constructed?

With the "Easy Button"

- Inmon-style DW model with 1 very large table (lineitem) and 7 smaller tables
- All tables contain DATE and STRING columns that are joined using numeric business keys
- Low Query Complexity: 22 queries with only 1 LEFT JOIN, simple aggregates and subqueries, no nested CTEs, and predicates applied directly to large tables
- Easier Tuning Complexity: often "super-tuned" and each query gets a perfectly covering index
- Does not require a sophisticated optimizer: needs join reordering and

TPC-DS: The Popular Kid in School

The one we all fight over

- First published in 2012 to counter aging TPC-H's limitations and tackle current OLAP trends
- A benchmark that is often misused/misrepresented by skipping one or more of the 3 components designed to ensure operational considerations aren't forgotten for the sake of over-indexing on this benchmark's SQL queries
 - Load Test, Throughput Test, and Data Maintenance Test
 - A multi-cloud data warehouse platform even publishes a highly-tuned, preloaded TPC-DS dataset in all deployed warehouses for users to consume
- Still valuable in a vacuum when results can be trusted and validated

How is TPC-DS constructed

Retailer selling goods via 3 different distribution channels: Store, Catalog, Internet

- Based on Kimball dimensional modeling (<u>The Data Warehouse Toolkit</u>)
 - Replaced TPC-H 3NF approach with hybrid approach between 3NF and star schema, or a "multiple snowflake schema"
- Significantly more complicated than TPC-H
 - Heavy on advanced SQL features/functions and lopsided filters
 - 99 queries compared to meager 22 in TPC-H
- 4 query classes:
 - pure reporting queries
 - pure ad-hoc queries
 - iterative OLAP queries
 - extraction or data mining queries



Feature Comparison: TPC-H & TPC-DS

Easier to consume cheat sheet

DA

Feature	ТРС-Н	TPC-DS
Data Model	Simpler schema, uses Inmon style DW model	Complex schema, Kimball style dimensional model.
Schema	1 very large table (lineitem) 7 smaller tables	6 fact tables (3 _sales, 3 _returns) 18 dimension tables
Data Types	All tables contain DATE and STRING columns. Tables are joined using numeric business keys	Fact tables use only INTEGER and NUMERIC columns. Only dimension tables use TIMESTAMP and STRING Tables are joined using numeric surrogate keys
Query Complexi ty	 22 queries: Low complexity Only uses 1 LEFT JOIN Only uses simple aggregates Subqueries are simple, no nested CTEs Predicates applied directly to large tables 	 99 queries: High complexity 9 queries use LEFT JOIN, 3 use a cross join Complex aggregates, 15 queries use window functions Complex nested CTEs used in most queries Predicates applied <u>only</u> to dimension tables
Tuning Complexi ty	 Easier for vendors to tune Often "super-tuned" with perfect indexes Does <u>not</u> require a sophisticated optimizer: 	 Harder for vendors to tune Optimizing a specific query can make others slower <u>Requires</u> a sophisticated query optimizer: must be

Is the TPC-H Valuable?

Best for: Ad -Hoc Manual Benchmarking



Easy Peasy Man

- Simpler schema, easier to understand and manage.
- Fewer benchmark queries and they are easy to understand
- Tables contain DATE and STRING columns that are used as predicates



- Too Simple and Easy to Shortcut. Been replaced! more complex modern data warehousing needs.
 - Simple queries do no reflect hypercomplex real world queries from tools like Tableau and dbt
 - Simple schema does not reflect best

Is the TPC-DS Valuable?

Best for: Vendor Supported POC Evaluations



Most modern of the Common SQL

- Benchmarks Complex, realistic schema that better mimics enterprise data warehouses.
 - Covers a broad spectrum of query types, SQL operators, and complex joins.
- Requires a sophisticated optimizer, testing more capabilities.



Complexity & Popularity Result in Missed Stages • Higher complexity in setup and longer time to implement and tune.

- Many complex queries can make the results hard to evaluate.
- Can require significant resources to fully utilize and understand

©2024 Databricks Inc. — All rights reserved

...TPC-LH?



The State of "Lakehouse" Benchmarks

How can the industry do better?

No Real "Official" Lakehouse Benchmark

- Each benchmark focuses on a portion of end-to-end Lakehouse platform
 - Favors bias and "shortcuts" to improve performance
- Reveals flaws in keeping benchmarks current
 - Example: TPC-DI has no way to calculate its benchmarked metric for cloud platforms

LHBench: Berkeley white paper implementing a Lakehouse benchmark on EMR

- Composed of 4 tests:
 - TPC-DS
 - TPC-DS Refresh
 - Merge Microbenchmark
 - Large File Count
- Pattern appears sound and it's a great start - but not "official"
 - Community may want to move beyond TPC-DS as the core of the benchmark.

How costly is optimizing and what can be learned to "balance" a Lakehouse benchmark?

- Modified for OPTIMIZE on all fact tables
- Adjusting for cluster start times, it takes
 44\$6.42100eDemand nodes (\$4.92 on spot)
- Despite tuning the tables this price is stil less than:
 - Half the price of EMR
 - 1/3 the price of Big Query
- Over 15x cheaper than DATATAI SUBMITY other CDW we

akes	optimize_dimcustomer
	Silver_DimAccount
and nodes	optimizedimaccount
\ \	Gold_FactCashBalances
)	Silver_Prospect
tha	optimize_dimsecurity
uie	Gold_FactWatches
o ic ctill	Silver_DimTrade
	optimize_dimtrade
	Silver_FactHoldings
	optimize_factcashbalances
of EMR	optimize_factholdings
	optimize_factwatches
Big Query	optimize_financial
her than	Gold_FactMarketHistory
	optimize_factmarkethistory
'we	

optimize dimbroker

optimize_dimcompany

	Job ID	
9:00 AM 8:57:45 AM	539843215102897 匠	
20.15		Job ID
20.13	Job run ID	<u>701634745375479</u>
	934318395147884 🕞	lab sus ID
30.95	Launched	JOD FUN ID 1039057252964353
	Manually	
39.5s.	Manadity	Launched
49.45	Started	Manually
	06/08/2024, 08:49:00 AM	Started
22.6s	Ended	06/07/2024, 03:51:57 PM
37.2s		
-	00,00/2024, 00:00:12 / 10	Ended
32.15	Duration 🛈	06/07/2024, 04:02:40 PM
→ 1m 49s	17m 13s 15 29s *adjust for cluster start diff	
		10m 44s
37.75		
5.8s	-	Queue duration (i)
2m 17s	Status	
	Succeeded	Status
5m 19s		Succeeded
2m 47s	Lineage ()	
	18 upstream tables, 2 downstream tables	
→ <u>1m 33s</u>		29 upstream tables, 21 downst
→ <u>1m 5s</u>	Compute	
10060	TPCDI-SE10000-CLUSTER 9 m7ad 4x1 PHOTON	Compute
		Shannon Barrow's Cluster
→ 1m 45s	Driver: m/gd.2xlarge · Workers: m/gd.4xlarge · 9	Driver: m7ad 2xlarge . Workers
1m 3s	workers · On-demand · DBR: 15.1.x-photon-scalaz.12 ·	workers · On-demand · DBR: 1
		auto
9m 5s	View details Spark UI Logs Metrics	
2m 54s		View details Spark UI
Gold_FactMa	rketHistory 8m 23s	Iob parameters B

eam tables

m7qd.4xlarge · 9

Logs

2.x-photon-scala2.12

Metrics

How costly is optimizing and what can be learned to "balance" a Lakehouse benchmark?

- Modified for OPTIMIZE
 on all fact tables
- Adjusting for cluster start times, it takes
 44% longer

Is this worth it?

The answer is always the same: **if** consumption **savings** are **greater than** than the **costs** to optimize the data

Task name	8:49:00 AM 8:57:45 AM
optimize_dimbroker	20.1s
optimize_dimcompany	21.85
Silver_DimSecurity	30.9s
Silver_Financial_CIK	39.5s
Silver_Financial_CONAME	49.4s
optimize_dimcustomer	22.6s
Silver_DimAccount	37.25
optimizedimaccount	32.15
Gold_FactCashBalances	→ <mark>1m 49s</mark>
Silver_Prospect	37.75
optimize_dimsecurity	5.85
Gold_FactWatches	2m 17s
Silver_DimTrade	5m 19s
optimize_dimtrade	→ 2m 47s
Silver_FactHoldings	→ <mark>1m 33s</mark>
optimize_factcashbalances	→ <u>1m 5s</u>
optimize_factholdings	→ <mark>1m 6s</mark>
optimize_factwatches	l→ 1m 45s
optimize_financial	1m 3s
Gold_FactMarketHistory	9m 5s
optimize_factmarkethistory	→ <u>2m</u> 54s

<u>539843215102897</u>	Ъ
Job run ID 934318395147884	6
Launched	

Manually Started 06/08/2024, 08:49:00 AM

Ended 06/08/2024, 09:06:12 AM

Duration (i) 17m 13s 15 29s *adjust for cluster start diff

Queue duration (i)

Status

Lineage () 18 upstream tables, 2 downstream tables

Compute

TPCDI-SF10000-CLUSTER_9_m7gd_4xl_PHOTON
Driver: m7gd.2xlarge · Workers: m7gd.4xlarge · 9
workers · On-demand · DBR: 15.1.x-photon-scala2.12 ·
us-west-2a

View details Spark UI Logs Metrics

8m 23s

Job run ID 1039057252964353 🖷

Launched Manually

> Started 06/07/2024, 03:51:57 PM

Ended 06/07/2024, 04:02:40 PM

Duration i 10m 44s

Queue duration 🛈

Status

Lineage (i) 29 upstream tables, 21 downstream tables

Compute

Shannon Barrow's Cluster

Driver: m7gd.2xlarge · Workers: m7gd.4xlarge · 9 workers · On-demand · DBR: 15.2.x-photon-scala2.12 · auto View details Spark UI Logs Metrics

DATA⁺AI SUMMIT

Gold_FactMarketHistory

Job parameters 🛛 🔂

How costly is optimizing and what can be learned to "balance" a Lakehouse benchmark?

 Autostats feature: Stats on Write! 亩

 Leverages Liquid Clustering

cdi.tpcdi_sf100	JUO_clustered.factmarkethistory ☆ Create ✓ Create ✓
riew Sample Data	Details Permissions History Lineage Insights Quality
be	MANAGED
rage Location	s3://databricks-e2demofieldengwest/b169b504-4c54-49f2-bc3a-adf4b128f36d/tables/f9a3cb13- 2ce5-45e2-bd86-32be42e03405
perties	spark.sql.statistics.colStats.sk_fiftytwoweekhighdate.createdBy=root delta.lastCommitTimestamp=1717850980000 spark.sql.statistics.colStats.sk_companyid.createdBy=root spark.sql.statistics.colStats.sk_fiftytwoweekkowdate.min=20150706 spark.sql.statistics.colStats.sk_fiftytwomekkowdate.min=20150706 spark.sql.statistics.colStats.btdid.nullCount=0 spark.sql.statistics.colStats.volume.createdAt=1717851792839 spark.sql.statistics.colStats.volume.createdAt=1717851792839 spark.sql.statistics.colStats.cylatel.waten=8 spark.sql.statistics.colStats.sk_dateld.distinctCount=594 spark.sql.statistics.colStats.k_dateld.distinctCount=594
	spark.cal.statistics.colStats.daylow.nullCount=0 spark.cal.statistics.version=7 spark.sql.statistics.colStats.fffytwoweeklow.createdBy=root spark.sql.statistics.colStats.fffytwoweeklow.createdAt=717851792839 spark.sql.statistics.colStats.sk_dateid_nx=20170708 clusteringColumns=[[Tek_dateid"],[Tek_securityid"]] delta.lastUpdateVersion=0 spark.sql.statistics.colStats.sk_dateid.nullCount=0 spark.sql.statistics.colStats.sk_dateid.nullCount=0 spark.sql.statistics.colStats.batchid.nullCount=4
	delta.arakuré.v2checkpoint=supported delta.ariik/interviersion=7 spark.sql.statistics.colStats.sk_companyid.max=201707074421557 spark.sql.statistics.colStats.sk_companyid.max=201707074421557 spark.sql.statistics.colStats.sk_fiftytwoweekkighdate.delta.tictCount=694 spark.sql.statistics.colStats.sk_fiftytwoweekkighdate.version=2 delta.feature.rowTracking=supported spark.sql.statistics.colStats.dayhigh.createdBy=root spark.sql.statistics.colStats.sk_fiftytwoceekkievdate.rowCount=5429185582 spark.sql.statistics.colStats.sk_fiftytwoweekkievdate.rowCount=5429185582
	spark.sql.statistics.colStats.sksecurityid.min=0 spark.sql.statistics.avxilleryind=c?source*?xVT0_STATS*) spark.sql.statistics.colStats.volume.maxLen=4 spark.sql.statistics.colStats.sk_securityid version=2 spark.sql.statistics.colStats.sk_securityid version=2

Catalog Explorer > tpcdi > tpcdi_sf10000_unclustered >					
Transfilled to the text text text text text text text	0_unclustered.factmarkethistory ☆ : Open in a dashboard 🗸 Create 🗸				
Overview Sample Data	Details Permissions History Lineage Insights Quality				
Туре	MANAGED				
Storage Location	s3://databricks-e2demofieldengwest/b169b504-4c54-49f2-bc3a- adf4b128f36d/tables/d843f2f4-c966-48bc-bded-309847543bbc				
Properties	delta.lastCommitTimestamp=1715798454000 delta.lastUpdateVersion=0 delta.minWriterVersion=7 delta.amibBoletionVectors=true delta.minReaderVersion=3 delta.feature.deletionVectors=supported delta.feature.invariant=supported				
Created At	5/15/2024, 2:40:54 PM				
Created By	shannon.barrow@databricks.com				
Table Id	d84312f4-c966-48bc-bded-309847543bbc				
Delta Runtime Properties Kvpairs	(empty)				
Predictive Optimization Flag	DISABLE (inherited from SCHEMA tpcdi_sf10000_unclustered)				

NO STATS

How costly is optimizing and what can be learned to "balance" a Lakehouse benchmark?

- Autostats feature: Stats on Write!
- Leverages Liquid \bullet
- **Edisteriolgup** adightarrowhoc type query
- 1 set use cached result = false;
- 2 select * from factmarkethistory
- 3 where sk securityid = 42949723972 and sk dateid = 20170401

Raw results V +				
	1 ² 3 sk_securityid	1 ² 3 sk_companyid	1 ² 3 sk_dateid	
1	42949723972	19700316303537	20170401	

Optimized

Wall-clock duration		
Total wall-clock duration		350 ms
🗸 🍵 Scheduling 🛈	35 m	^s 10%
Waiting for compute ①	0 m	s
Waiting in queue 🛈	35 m	s
✓ ● Running ^①	315 m	s90%
Optimizing query & pruning files) 155 m	s
Executing ①	160 m	s
Start time	2024-06-09 21:44:17.64	0 -04:00
End time	2024-06-09 21:44:17.99	0 -04:00
Result fetching by client 🛈		49 ms
Tasks total time Tasks time in Photon		39 ms 82 %
Rows returned		1
Rows read	04	1
Bytes read	94	
Bytes read from cache 35s V	e 10 2e 🎬	00 00
Bytes written	5 10.25	0 bytes
		U Dytes
Files & partitions	_	
Files read	UX	2
Files pruned	vomont 2,7	31
Partitions read	vement	0

Not Optimized

Wall-clock duration 🕕		
Total wall-clock duration	10 s :	235 m
✓ ● Scheduling ①	43 ms	0%
Waiting for compute 🛈	0 ms	
Waiting in queue 🛈	43 ms	
∽ ● Running ①	10 s 192 ms	1009
Optimizing query & pruning files 🕕	139 ms	
Executing ①	10 s 53 ms	
Start time	2024-06-09 21:43:35.606	-04:0
End time	2024-06-09 21:43:45.841	-04:0
		40 11
Aggregated task time 🕕		
Tasks total time	3.5	9 m
Tasks time in Photon	9	8 %
10		
Rows returned		1
Rows read		1
Lites road	2 5	
Bytes read Bytes pruned	2.5 716.1	7 GB 8 MB
Bytes read Bytes pruned Bytes read from cache	2.5 716.1 6	7 GB 8 MB 2 %
Bytes read Bytes pruned Bytes read from cache Bytes written	2.5 716.1 6	7 GB 8 MB 2 % 0 byte
Bytes read Bytes read from cache Bytes written Files & partitions	2.5 716.1 6	7 GB 8 MB 2 % 0 byte
Bytes read Bytes pruned Bytes read from cache Bytes written Files & partitions Files read	2.5 716.1 6 44	7 GB 8 MB 2 % 0 byte 8
Bytes read Bytes pruned Bytes read from cache Bytes written Files & partitions Files read Files pruned	2.5 716.1 6 44 15	7 GB 8 MB 2 % 0 byte 8 5

How costly is optimizing and what can be learned to "balance" a Lakehouse benchmark?

- Autostats feature: Stats on Write!
- Leverages Liquid
- Edisteriolgup adhoc type query
- BI-like Query using Dimensional filtering and dynamic file

select dicti	inct sk securitvid			
from tocdi.1	unct sk_securityiu Incdi sf10000 cluste	red.dimsecurity sec		
join tpcdi.	tpcdi_sf10000_cluste	red.dimcompany comp on	comp.sk_companyid = sec.	sk_companyid
join tpcdi.	tpcdi_sf10000_cluste	red.industry ind on in	d.in_name = comp.indust	ry
join tpcdi.	tpcdi_sf10000_cluste	red.statustype status_c	omp <mark>on</mark> status_comp.st_na	ame = comp.status
join tpcdi.	tpcdi_sf10000_cluste	red.statustype status_s	ec on status_sec.st_name	e = sec.status
ind.in_i	ld = 'WU' Retail	(Apparel)		
and stat	tus_comp.st_id = 'ACT	V' Only active comp V' Only active secur	dilles	
and comp	.iscurrent is t		CD Type 2 records	
and comp	.stateprov = 'AL' -			
and comp	o.sprating = 'AAA' -			
and sec.	iscurrent is the			
and sec.	exchangeid = 'NASDA			
and sec.	issue = 'COMMON'	UNLI common issue stoc		
elect trade *				
here dd.fiscaly	yearid = 2015 all			
here dd.fiscaly sults ∨ +	yearid = 2015 all 1 ² a sk brokerid	trades for 1 year	1 ² 3 sk createtimeid	1 ² 2 sk closedateid
here dd.fiscaly sults \checkmark + 1^{2}_{3} tradeid 605927611	rearid = 2015 all 1 ² 3 sk_brokerid 102/3683	trades for 1 year 1 ² 3 sk_createdateid 20141105	i ² ₀ sk_createtimeid	¹² 3 sk_closedateid 201411
here dd.fiscaly sults ∨ + 1 ² 3 tradeid 605927611 661288248	yearid = 2015 → all i ² 3 sk_brokerid 10243683 35482844	trades for 1 year 1 ² 3 sk_createdateid 20141105 20150122	1 ² 3 sk_createtimeid 132425 84917	1 ² 3 sk_closedateid 201411 201503
here dd.fiscaly sults ~ + i ² 3 tradeid 605927611 661288248 660721160	yearid = 2015 all ₄ ² ₃ sk_brokerid 10243683 35482844 35682844	trades for 1 year 1 ² 3 sk_createdateid 20141105 20150122 20150122	132425 132425 84217 133310	1 ² 3 sk_closedateid 201411 201503 201503
here dd.fisca\y sults ∨ + 4 ² 3 tradeid 605927611 661288248 660721160 606908399	rearid = 2015 all 1 ² 3 sk_brokerid 10243683 35482844 35482844 10243683	trades for 1 year *% sk_createdateid 20141105 20150122 20150121 20141106	1 ² 3 sk_createtimeid 132425 84217 133310 222758	1 ² 3 sk_closedateid 201411 201503 201503 201411
here dd.fiscaly sults ∨ + 4 ² ₃ tradeid 605927611 661288248 660721160 606908399 596186927	rearid = 2015 all 1 ² 3 sk_brokerid 10243683 35482844 35482844 10243683 21148125	trades for 1 year 1 ² 3 sk_createdateid 20141105 20150122 20150121 20141106 20141102	1 ² 3 sk_createtimeid 132425 84217 133310 222758 205013	1 ² 3 sk_closedateid 201411 201503 201503 201411 201411
here dd.fiscaly sults	rearid = 2015 all 1 ² 3 sk_brokerid 10243683 35482844 35482844 10243683 21148125 21148125	trades for 1 year 1 ² a sk_createdateid 20141105 20150122 20150121 20141106 20141102 20141022 20141023	± ² 3 sk_createtimeid 132425 84217 133310 222758 205013 44220	1 ² 3 sk_closedateid 201411 201503 201503 201411 201412 201412
here dd.fiscaly sults × + salts × + salta 605927611 661288248 660721160 606908399 596186927 596418065 589507618	rearid = 2015 all 1 ² 3 sk_brokerid 10243683 35482844 35482844 10243683 21148125 20760 <u>6652</u>	trades for 1 year 1 ² 3 sk_createdateid 20141105 20150122 20150121 20141106 20141022 20141023 20141023	± ² ₃ sk_createtimeid 132425 84217 133310 222758 205013 44220 13821	1 ² 3 sk_closedateid 201411 201503 201503 201411 201412 201412 201412 201412
here dd.fiscaly sults ∨ + 4 ² 3 tradeid 605927611 661288248 660721160 606908399 596186927 596418065 589507618 59037618	rearid = 2015 all 10243683 35482844 35482844 10243683 21148125 21148125 20760652 20760652	trades for 1 year 1 ² 3 sk_createdateid 20141105 20150122 20150121 20141106 20141102 201411023 20141013 20141013	1 ² 3 sk_createtimeid 132425 84217 133310 222758 205013 44220 113621 103835	1 ² 3 sk_closedateid 201411 201503 201503 201411 201412 201412 201410 201410 201410
here dd.fiscaly sults v + 1 ² tradeid 605927611 661288248 660721160 606908399 566186927 596418065 589507618 590191010 561946739	earid = 2015 all a ² 3 sk_brokerid 10243683 35492844 10243683 21148125 21148125 20760652 20760652 34101145	trades for 1 year ** sk_createdateid 20141105 20150122 20150121 20141105 20141102 20141102 20141023 20141013 20141014 20140114	x ² ₃ sk_createtimaid 132425 84217 133310 222758 205013 44220 113621 108835 164009	t ⁴ 3 sk_closedateid 201411 201503 201503 201411 201412 201410 201410 201410 201408
erre dd.fiscaly sults ∨ + s ² s tradeid 605927611 661288248 660721160 606908399 596186927 596418065 589507618 590191010 551946739 653527702	earid = 2015 all s ² 3 sk_brokerid 10243683 35482844 35482844 10243683 21148125 21148125 20760652 20760652 20760652 34101145 38231555	trades for 1 year *3 sk_createdateid 20141105 20150122 20150121 20141106 20141023 20141023 20141023 20141013 20141014 201400821 2015011	x ² ₃ sk_createtimeid 132425 84217 133310 222758 205013 44220 113821 103835 164009 105646	1 ² 3 sk_closedateid 201411 201503 201503 201411 201412 201412 201410 201410 201410 201410 201408 201503
here dd.fiscaly sults ∨ + x³ tradeid 605927611 661288248 660721160 606908399 569186927 596418065 589507618 58910101 551946739 653527702 65408212	rearid = 2015 all 1 ² 3 sk_brokerid 10243683 35482844 10243683 21148125 20760652 20760652 34101145 38231555 38231555	trades for 1 year * ² 3 sk_createdateid 20141105 20150122 20150121 20141106 20141023 20141023 20141013 20141013 20141014 20140821 20150111 20150111	x ² 3 sk_createtimeid 132425 84217 133310 222758 225013 44220 113621 103835 164009 105646	1 ² 3 sk_closedateid 201411 201503 201503 201412 201412 201412 201412 201410 201410 201408 201503 201503
here dd.fiscaly sults × + 13 tradeid 605927611 661288248 660721160 606908399 596186927 596418065 589507618 590191010 551946739 653527702 654098412 564992484	rearid = 2015 all 1 ² 3 sk_brokerid 10243683 35482844 35482844 10243683 21148125 20760652 20760652 20760652 34101145 38231555 38231555 38101145	trades for 1 year 1 ² 3 sk_createdateid 20141105 20150122 20150121 20141106 20141102 201411023 20141013 20141014 20150111 20150112 20140212	1 ² 3 sk_createtimeid 132425 84217 133310 222758 205013 44220 113621 103835 1164009 105646 61227 23223	1 ² 3 sk_closedateid 201411 201503 201503 201411 201412 201412 201412 201410 201408 201503 201503 201408
here dd.fiscaly sults × + s ² s tradeid 605927611 661288248 660721160 606908399 596186927 596418065 569507618 590191010 551946739 663527702 654098412 55492464 6431338655	rearid = 2015 all 1 ² 3 sk_brokerid 10243683 35482844 10243683 21148125 21148125 20760652 20760652 34101145 38231555 34101145 8063334	trades for 1 year	± ² ₃ sk_createtimeid 132425 84217 133310 222758 205013 44220 113621 103835 164009 105646 61227 232223	1 ² 3 sk_closedateid 201411 201503 201503 201411 201412 201412 201410 201408 201503 201503 201503 201503 201503



How costly is optimizing and what can be learned to "balance" a Lakehouse benchmark?

- How do we balance the Query Load based on the TPC-DI optimize added latency?
- Back of the napkin math...
 - Conservatively assume 2x performance gains for SQL
 - Assume 5 minutes longer to optimize ETL (at 10k scale factor) need to save at 5 minutes in queries
 - $5 = \frac{NonOptimizedTablesQueryTime}{2}$
 - Therefore if there is 2x improvement in the SQL times and we need to make up 5 minutes, we need approximately 10 minutes of non-optimized tables query time
 - 5 minutes on optimized tables

AI Benchmarking • ` OLYMPICS OLYMPICS DLYMPICS OLYMPICS VALEPU

https://harvard-edge.github.io/cs249r_book/contents/benchmarking/benchmarking.html

Why Benchmark in AI/ML?

- Standardized methods allow us to quantitatively know the capabilities of different models, software, and hardware enabling fair comparisons across different solutions.
- Allow ML developers to measure the inference time, memory usage, power consumption, and other metrics that characterize a system.
- Goals and Objectives:
 - Performance assessment
 - Resource evaluation
 - Validation and verification
 - Competitive analysis
 - Credibility
 - Regulation and Standardization

What to Benchmark in AI/ML?

How does one benchmark something so subjective?

- 3 primary categories:
 - Hardware/System
 - Model
 - Data
- Granularity:
 - Micro
 - Macro
 - End to End
- Training vs Inference



https://harvard-edge.github.io/cs249r_book/contents/benchmarking/benchmarking.html

In an LLM Not Far Away...

Keeping Pace and Choosing Wisely

Benchmarks are rapidly created and deprecated, what can Mosaic's Gauntlet teach us?

- According to <u>Stanford's 2024 AI</u> <u>INDEX REPORT</u>
 - 15 benchmarks were deprecated in 2023 alone - many of which were less than 4 years old
 - 18 new benchmarks were added in 2023
- The "Mosaic Evaluation Gauntlet" (blog)
 - Evaluated 39 public benchmarks split across 6 core competencies
 - In order to prioritize the metrics that are most useful for research tasks across model scales, we tested the

DATATAI SUNDENCHMARKS USING A SERIES OF ©2024 Databricks Inc. — All rights reserved

Group 1: Well-behaved metrics robust to few-shot settings

These benchmarks reliably ordered models by training scale and monotonically improved at any number of shots. We believe that these benchmarks can provide a reliable evaluation signal for models in this range.



Figure 2: Monotonically improving benchmarks. These benchmarks include popular tasks like Lambada, BoolQ, Arc, and Hellaswag.

Challenges and Trends

Human evaluation is "in"

- Practitioners are growing incredibly skeptical about Academic Benchmarks
 - Habitual issues overfitting models to existing benchmarks
 - MMLU, HumanEval, Hellaswag are bona fide benchmarks but model creators' game the system for models to do well on them
- Accordingly, practitioners today tend to prefer evaluating their LLM options by human preference in the real-world - like <u>LMSYS</u>
 - The HAI Stanford Report even points out "human evaluation is in" (Chapter 2)
 - LMSYS: Allows users to vote on the better response based on a prompt they provide to the LLMs the user is blind to the choice of the models they're given).



