Accelerating QuestDB: Lessons from A 6x Query Performance Boost

Jaromir Hamala

Core Engineer at QuestDB

Javier Ramirez

Database Advocate at QuestDB



The database is the bottleneck

- Every developer in the 90s

Common use case for a time-series database

Real-time dashboards on recent data.

Real-time decision making (i.e. payment fraud).

> Historical queries aggregated by time chunks.

Latest trades 🕕 🕘 Last 1	minute				
timestamp ↓	asset	counter	quantity	consideration	new
2024-08-08 14:48:31.221	BTC	USD	0.0300	-1724	x
2024-08-08 14:48:31.221	BTC	USD	0.0200	-1148	x
2024-08-08 14:48:30.914	ETH	USD	0.0218	-53.4	x
2024-08-08 14:48:30.913	ETH	DAI	0.00290	-7.12	x
2024-08-08 14:48:30.909	ETH	USD	0.0104	-25.5	x
2024-08-08 14:48:30.909	ETH	USD	0.00712	-17.5	x
2024-08-08 14:48:30.909	ЕТН	USD	0.00427	-10.5	x
2024-08-08 14:48:30.909	ETH	USD	0.0232	-57.0	x

Volume heatmap ① ② Last 5 minutes



Real-time trades (1) (2) Last 1 minute

14.47.40

- UNI-USD - XLM-USD

- ADA-USD - AVAX-USD - BTC-USD -- ETH-USD - LTC-USD - MATIC-USD -

11.17.50

11.19

30000

20000

10000

-10000 -20000

-30000

Meet QuestDB: OSS time-series database

- <u>https://github.com/questdb/questdb</u> (Apache License 2.0)
- High-speed ingestion: InfluxDB line protocol over TCP or HTTP
- Columnar storage format (native or Parquet), partitioned and ordered by time
- Written in Java (90%) and C++/Rust (10%)
- Uses in-house replacement of Java's standard library
- Zero GC, SIMD, parallel SQL execution, SQL JIT compiler
- SQL with time-series extensions: PGWire, HTTP API

Time-series databases high level overview

 Time Series Databases specialise in very fast ingestion, very fast queries over nascent data, and powerful time-based analytical queries.

• They focus on nascent data, deleting, downsampling, or slowing-down older data.

Time-series SQL extensions

```
SELECT pickup_datetime, count() FROM trips
WHERE pickup_datetime in '2016-06-13;1M;1y;3'
SAMPLE BY 1w;
```

```
select timestamp, avg(price) from
(read_parquet('trades.parquet')
timestamp(timestamp)) sample by 15m;
```

```
SELECT
```

```
timestamp, symbol, side, sum(amount) as volume
FROM trades
WHERE side = 'sell' AND timestamp IN today()
SAMPLE BY 1m FILL(NULL);
```

```
SELECT * FROM trades
WHERE symbol in ('BTC-USDT', 'ETH-USDT')
LATEST ON timestamp PARTITION BY symbol;
```

SELECT pickup_datetime, fare_amount, timestamp, tempF, windDir FROM trips ASOF JOIN weather WHERE pickup_datetime in '2018-06-01';

QuestDB in action: quick showcase

https://dashboard.demo.questdb.io/d/fb13b4ab-b1c9-4a54-a920-b60c5fb036 3f/public-dashboard-questdb-io-use-cases-crypto?orgId=1&refresh=750ms

https://demo.questdb.io

https://github.com/questdb/time-series-streaming-analytics-template





What makes a decent analytical database?

- SQL
- Columnar storage format
- All HW resources (CPU & RAM) are available for faster query execution
- Complex queries with GROUP BY / JOIN / filter over large volumes of data, not necessarily accessed over time

How do you improve analytical DB capabilities?

- ClickBench <u>https://github.com/ClickHouse/ClickBench</u>
 - Results accepted by ClickHouse: <u>https://benchmark.clickhouse.com</u>
- db-benchmark <u>https://github.com/duckdblabs/db-benchmark</u>
 - Results accepted by DuckDB: <u>https://duckdblabs.github.io/db-benchmark</u>
- TPC benchmarks <u>https://www.tpc.org</u>
- TSBS <u>https://github.com/timescale/tsbs</u>
 - Time-series specific, not maintained

ClickBench

- Created by ClickHouse team in 2022
- Single table with 105 columns and 99M rows (Yandex search events)
- Includes data import, e.g. in CSV, but the main focus is on queries
- 43 queries with complex GROUP BY, WHERE, and ORDER BY clauses
- Only a few of the queries make use of time (QuestDB was already fast there)
- Run on different machines, but most popular are AWS EC2 instances with EBS volumes

Some sample queries

SELECT COUNT(*) FROM hits; SELECT COUNT(*) FROM hits WHERE AdvEngineID <> 0; SELECT count_distinct(UserID) FROM hits; SELECT count_distinct(SearchPhrase) FROM hits; SELECT UserID FROM hits WHERE UserID = 435090932899640449;

SELECT ClientIP, ClientIP - 1, ClientIP - 2, ClientIP - 3, COUNT(*) AS c FROM hits GROUP BY ClientIP, ClientIP - 1, ClientIP - 2, ClientIP - 3 ORDER BY c DESC LIMIT 10;

SELECT TraficSourceID, SearchEngineID, AdvEngineID, CASE WHEN (SearchEngineID = 0 AND AdvEngineID = 0) THEN Referer ELSE '' END AS Src, URL AS Dst, COUNT(*) AS PageViews FROM hits WHERE CounterID = 62 AND EventTime >= '2013-07-01T00:00:00Z' AND EventTime <= '2013-07-31T23:59:59Z' AND IsRefresh = 0 GROUP BY TraficSourceID, SearchEngineID, AdvEngineID, Src, Dst ORDER BY PageViews DESC LIMIT 1000, 1010;

SELECT TraficSourceID, SearchEngineID, AdvEngineID, CASE WHEN (SearchEngineID = 0 AND AdvEngineID = 0) THEN Referer ELSE '' END AS Src, URL AS Dst, COUNT(*) AS PageViews FROM hits WHERE CounterID = 62 AND EventTime >= '2013-07-01T00:00:00Z' AND EventTime <= '2013-07-31T23:59:59Z' AND IsRefresh = 0 GROUP BY TraficSourceID, SearchEngineID, AdvEngineID, Src, Dst ORDER BY PageViews DESC LIMIT 1000, 1010;

Some sample queries

SELECT SUM(ResolutionWidth), SUM(ResolutionWidth + 1), SUM(ResolutionWidth + 2), SUM(ResolutionWidth + 3), SUM(ResolutionWidth + 4), SUM(ResolutionWidth + 5), SUM(ResolutionWidth + 6), SUM(ResolutionWidth + 7), SUM(ResolutionWidth + 8), SUM(ResolutionWidth + 9), SUM(ResolutionWidth + 10), SUM(ResolutionWidth + 11), SUM(ResolutionWidth + 12), SUM(ResolutionWidth + 13), SUM(ResolutionWidth + 14), SUM(ResolutionWidth + 15), SUM(ResolutionWidth + 16), SUM(ResolutionWidth + 17), SUM(ResolutionWidth + 18), SUM(ResolutionWidth + 19), SUM(ResolutionWidth + 20), SUM(ResolutionWidth + 21), SUM(ResolutionWidth + 22), SUM(ResolutionWidth + 23), SUM(ResolutionWidth + 24), SUM(ResolutionWidth + 25), SUM(ResolutionWidth + 26), SUM(ResolutionWidth + 27), SUM(ResolutionWidth + 28), SUM(ResolutionWidth + 29), SUM(ResolutionWidth + 30), SUM(ResolutionWidth + 31), SUM(ResolutionWidth + 32), SUM(ResolutionWidth + 33), SUM(ResolutionWidth + 34), SUM(ResolutionWidth + 35), SUM(ResolutionWidth + 36), SUM(ResolutionWidth + 37), SUM(ResolutionWidth + 38), SUM(ResolutionWidth + 39), SUM(ResolutionWidth + 40), SUM(ResolutionWidth + 41), SUM(ResolutionWidth + 42), SUM(ResolutionWidth + 43), SUM(ResolutionWidth + 44), SUM(ResolutionWidth + 45), SUM(ResolutionWidth + 46), SUM(ResolutionWidth + 47), SUM(ResolutionWidth + 48), SUM(ResolutionWidth + 49), SUM(ResolutionWidth + 50), SUM(ResolutionWidth + 51), SUM(ResolutionWidth + 52), SUM(ResolutionWidth + 53), SUM(ResolutionWidth + 54), SUM(ResolutionWidth + 55), SUM(ResolutionWidth + 56), SUM(ResolutionWidth + 57), SUM(ResolutionWidth + 58), SUM(ResolutionWidth + 59), SUM(ResolutionWidth + 60), SUM(ResolutionWidth + 61), SUM(ResolutionWidth + 62), SUM(ResolutionWidth + 63), SUM(ResolutionWidth + 64), SUM(ResolutionWidth + 65), SUM(ResolutionWidth + 66), SUM(ResolutionWidth + 67), SUM(ResolutionWidth + 68), SUM(ResolutionWidth + 69), SUM(ResolutionWidth + 70), SUM(ResolutionWidth + 71), SUM(ResolutionWidth + 72), SUM(ResolutionWidth + 73), SUM(ResolutionWidth + 74), SUM(ResolutionWidth + 75), SUM(ResolutionWidth + 76), SUM(ResolutionWidth + 77), SUM(ResolutionWidth + 78), SUM(ResolutionWidth + 79), SUM(ResolutionWidth + 80), SUM(ResolutionWidth + 81), SUM(ResolutionWidth + 82), SUM(ResolutionWidth + 83), SUM(ResolutionWidth + 84), SUM(ResolutionWidth + 85), SUM(ResolutionWidth + 86), SUM(ResolutionWidth + 87), SUM(ResolutionWidth + 88), SUM(ResolutionWidth + 89) FROM hits;

ClickBench — a Benchmark For Analytical DBMS

Methodology | Reproduce and Validate the Results | Add a System | Report Mistake | Hardware Benchmark

System:	All Athe	ena (partitioned)	Athena (single)	Aurora for	MySQL Auro	ra for PostgreSQI	ByteHouse	Citus		
	clickhouse	local (partitione	d) clickhouse-l	ocal (single)	ClickHouse	ClickHouse (zst	d) CrateDB	Databend	datafusion	Druid
	DuckDB	Elasticsearch	Elasticsearch (tur	ned) Green	olum HeavyA	Infobright	MariaDB Column	Store Ma	riaDB Mon	etDB
	MongoDB	MySQL (MyIS	AM) MySQL I	Pinot Postg	reSQL Ques	tDB (partitioned)	QuestDB Re	dshift Sir	ngleStore	
	Snowflake	SQLite Star	Rocks (tuned)	StarRocks	limescaleDB (c	compression) T	imescaleDB			
Туре:	All state	eless managed	Java colum	n-oriented	C++ MySQL	compatible row	v-oriented C	PostgreS	QL compatible	3
	ClickHouse	e derivative en	nbedded Rust	search do	cument time	e-series				
Machine:	All serve	erless 16acu	L M S XS	c6a.4xlarg	ge, 500gb gp2	c6a.metal, 500	gb gp2 f16s v	2 c6a.4x	arge, 1500gb	gp2
	ra3.16xlarg	ra3.4xlarge	ra3.xlplus S2	4 <mark>S2</mark> 2X	L 3XL 4XL	. XL				
Cluster size:	All 1 2	2 4 8 12	16 32 64	128 server	less undefin	ed				
Metric:	Cold Run	Hot Run Lo	ad Time Storad	e Size						

System & Machine Relative time (lower is better) ClickHouse (c6a.metal, 500gb gp2): ×1.30 StarRocks (c6a.metal, 500gb gp2): ×1.69 StarRocks (c6a.4xlarge, 500gb gp2): ×3.34 ClickHouse (c6a.4xlarge, 500gb gp2): ×3.54 ByteHouse (L): ×3.77 Redshift (4×ra3.4xlarge): ×3.86 Snowflake (8×L): ×4.50 SingleStore (2×S2)*: ×5.30 ByteHouse (M): ×5.88 Snowflake (4×M): ×6.16 Redshift (4×ra3.xlplus): ×6.92 ByteHouse (S): ×7.53 Snowflake (2×S): ×8.62 MonetDB (c6a.4xlarge, 500gb gp2): ×8.81 SingleStore (c6a.4xlarge, 500gb gp2)*: ×9.40 ByteHouse (XS): ×12.55 Snowflake (XS): ×12.55 DuckDB (c6a.4xlarge, 500gb gp2)*: ×22.44 Pinot (c6a.4xlarge, 500gb gp2)*: ×22.54 Greenplum (c6a.4xlarge, 500gb gp2): ×26.33 datafusion (f16s v2): ×31.84 QuestDB (c6a.4xlarge, 500gb gp2): ×33.89 Databend (c6a.4xlarge, 500gb gp2): ×33.93 MariaDB ColumnStore (c6a.4xlarge, 500gb gp2)*: ×46.38 CrateDB (c6a.4xlarge, 500gb gp2)*: ×47.81 Elasticsearch (c6a.4xlarge, 1500gb gp2): ×57.81 TimescaleDB (compression) (c6a.4xlarge, 500gb gp2): ×68.01 Druid (c6a.4xlarge, 500gb gp2)*: ×117.77 HeavyAl (c6a.4xlarge, 500gb gp2)*: ×127.44 Citus (c6a.4xlarge, 500gb gp2): ×173.17

https://tinyurl.com/clickbench-2022-10

System:	All	Athena	a (partition	ed) Ath	ena (single	e) Aurora	a for MySG	L Au	rora for Po	stgreSQL	ByConi	ty Byt	eHouse	chDB	Citus
	Click	(House (data lake,	partitione	d) Click	House (Par	quet, parti	tioned)	ClickHo	use (Parq	uet, single	e) Click	KHouse (v	veb)	ClickHouse
	Click	(House ((tuned)	ClickHous	e (zstd)	ClickHous	e Cloud (A	WS)	ClickHouse	e Cloud (G	CP) Cra	ateDB	Databend	t l	
	Data	Fusion (Parquet, s	ingle) A	pache Do	ris Druid	DuckD	B (Parqu	iet, partitio	ned) Du	uckDB E	Elasticsea	arch Ela	asticsea	arch (tuned)
	Gree	enplum	HeavyAl	Hydra	Infobrig	ht Kinetic	a Maria	DB Col	umnStore	MariaDE	B Monet	tDB M	ongoDB	MySG	L (MyISAM)
	MyS	QL Pi	not Pos	tgreSQL (1	tuned) P	ostgreSQL	QuestD	B (parti	tioned)	QuestDB	Redshift	Selec	tDB Sir	ngleSto	е
	Snow	wflake	SQLite	StarRock	s Times	caleDB (co	mpression) Tim	escaleDB						
Type:	All	statele	ess man	aged Ja	ava colu	imn-oriente	d C++	MySG	L compatil	ble row	-oriented	C P	ostgreSQ	L comp	atible
	Click	House	derivative	embed	ded serv	verless a	ws gcp	Rust	search	docume	ent time	-series			
Machine:	All	server	less 16a	icu c6a	.4xlarge, 5	00gb gp2	LM	S X	s c6a.m	etal, 500g	b gp2 d	5n.4xlar	ge, 500gl	b gp2	
	c5.4	xlarge, 5	500gb gp2	192GB	24GB	360GB	48GB 7	20GB	96GB	708GB	m5d.24xla	arge m	6i.32xlar	ge	
	c6a.	4xlarge,	1500gb g	p2 dc2.	8xlarge	ra3.16xlarg	e ra3.4	klarge	ra3.xlplus	S2 5	S24 2X	L 3XL	4XL	XL	
Cluster size:	All	1 2	4 8	16 32	64 12	8 server	less und	lefined							
Metric:	Cold	Run	Hot Run	Load Ti	me Stor	rage Size									

System & Machine Relative time (lower is better) StarRocks (c6a.metal, 500gb gp2): ×1.77 ClickHouse (c6a.metal, 500gb gp2): ×1.77 Databend (c6a.metal, 500gb gp2): ×1.83 SelectDB (c6a.metal, 500gb gp2): ×2.28 DuckDB (c6a.metal, 500gb gp2): ×3.32 Databend (c6a.4xlarge, 500gb gp2): ×3.39 SelectDB (c6a.4xlarge, 500gb gp2): ×3.43 ClickHouse (c6a.4xlarge, 500gb gp2): ×3.86 Apache Doris (c6a.4xlarge, 500gb gp2): ×4.27 StarRocks (c6a.4xlarge, 500gb gp2): ×4.30 DuckDB (c6a.4xlarge, 500gb gp2): ×4.64 Snowflake (16×XL): ×5.29 ByConity (c6a.4xlarge, 500gb gp2): ×5.77 ByteHouse (L): ×5.83 chDB (c6a.metal, 500gb gp2): ×6.18 Snowflake (8×L): ×6.97 SingleStore (S2)*: ×8.20 ByteHouse (M): ×9.09 Snowflake (4×M): ×9.53 Redshift (4×ra3.xlplus): ×10.71 chDB (c6a.4xlarge, 500gb gp2): ×11.56 ×11.64 ByteHouse (S): ×11.96 DataFusion (Parquet, single) (c6a.4xlarge, 500gb gp2)*: Snowflake (2×S): ×13.34 MonetDB (c6a.4xlarge, 500gb gp2): ×13.63 SingleStore (c6a.4xlarge, 500gb gp2)*: ×14.55 ×15.85 QuestDB (partitioned) (c6a.metal, 500gb gp2)*: ByteHouse (XS): ×19.42 Snowflake (XS): ×19.42 QuestDB (c6a.4xlarge, 500gb gp2): ×20.33 Greenplum (c6a.4xlarge, 500gb gp2): ×32.41 Pinot (c6a.4xlarge, 500gb gp2)*: ×34.88 Hudra (ofa Avlarga 500ab an2). ×45.98 https://github.com/ClickHouse/ClickBench/blob/main/questdb-partitioned/results/c6a.metal.json

https://tinyurl.com/clickbench-2023-08-18

System:	All	Alloy	DB A	lloyDB	(tuned)	Athe	na (part	itioned	A) (b)	thena	(single)	Auror	a for N	/ySQL	Auro	ra for	Postgre	SQL	ByConity
	Byte	House	chD	B (Data	Frame)	chDB	(Parqu	et, par	titione	d) (hDB	Citus	ClickH	louse C	loud (a	ws)			
	Click	House	e Cloud	(aws) F	Parallel F	Replicas	ON	ClickH	ouse C	Cloud	(Azure)	Click	House	Cloud (Azure)	Parall	el Replic	a ON	
	Click	House	e Cloud	(Azure)	Paralle	I Replica	as ON	Click	House	Cloud	d (gcp)	ClickH	louse	Cloud (gcp) Pa	arallel	Replicas	S ON	
	Click	House	e (data l	ake, pa	rtitioned	i) Clie	kHouse	e (data	lake,	single) Clici	kHouse	(Parqu	et, part	itioned) Cl	ickHous	e (Paro	quet, single)
	Click	House	e (web)	Click	House	Click	House (tuned)	Clie	ckHou	se (tune	d, mem	ory)	Cloud	perry	Crate	DB		
	Crun	chy B	ridge fo	r Analy	tics (Par	quet)	Databa	end	DataF	usion	(Parquet	t, partitic	oned)	Data	usion	(Parqu	let, sing	le) A	pache Doris
	Drui	d Di	uckDB (DataFra	me) [DuckDB	(Parque	et, par	titione	d) [DuckDB	Elastic	csearc	h Ela	sticsea	arch (t	uned)	Glare	DB
	Gree	nplum	Hea	vyAl	Hydra	Infobri	ight k	Cinetic	a M	ariaDE	Colum	nStore	Maria	DB	MonetD	BN	IongoD	B Mo	otherduck
	MyS	QL (M	yISAM)	MyS	QL O	xla Pa	andas (I	DataFr	ame)	Para	deDB (F	Parquet,	partitio	oned)	Parad	IeDB (I	Parquet	, single) Pinot
	Pola	rs (Da	taFrame) Pos	stgreSQ	L (tuned	i) Pos	stgreS	QL	Quest	DB (part	itioned)	Que	stDB	Redsh	ift S	SingleSt	ore	Snowflake
	SQLi	te	StarRock	s Ta	blespac	e Ter	nbo OL	AP (co	lumna	r) T	imescal	eDB (cor	mpres	sion)	Times	caleDE	3 Uml	bra	
Type:	All	С	column	-oriente	ed Po	stgreSG	L comp	atible	mai	naged	gcp	statele	ess	Java	C++	MySO	QL com	oatible	
	row-	orient	ed Cl	ickHou	se deriv	ative	embed	ded	serve	rless	datafr	ame a	aws	parallel	replica	as A	zure	analyti	cal Rust
	sear	ch d	docume	nt so	mewhat	Postgr	eSQL co	ompati	ble	time-s	eries								
Machine:	All	16 v	CPU 128	GB 8	VCPU	64GB	server	less	16acu	c6	a.4xlarg	e, 500g	b gp2		MS	XS	c6a.m	netal, 5	00gb gp2
	1920	B 2	4GB	360GB	48GE	3 720	GB 9	6GB	1430	GB	dev 7	08GB	c5n.4	xlarge,	500gb	gp2			
	Anal	ytics-:	256GB (64 vCo	res, 256	GB)	c5.4xla	rge, 5	00gb g	jp2	c6a.4xl	arge, 150	00gb g	gp2 c	loud	dc2.8	xlarge	ra3.1	6xlarge
	ra3.4	xlarg	e ra3.	xiplus	S2 :	S24 2	XL 3	SXL	4XL	XL	L1 - 160	PU 32G	BC	6a.4xla	rge, 50	0gb gj	p3		
Cluster size:	All	1	2 4	8 16	32	64	128 s	erverle	ess	dedica	ated								
Metric:	Cold	Run	Hot R	un L	oad Tim	ne St	orage S	lize											

System & Machine	Relative time (lower is better)	
Umbra (c6a.metal, 500gb gp2):	×	<1.57
Apache Doris (c6a.metal, 500gb gp2):	×	<2.10
ClickHouse (c6a.metal, 500gb gp2):	×	<2.15
StarRocks (c6a.metal, 500gb gp2):	×	2.32
Umbra (c6a.4xlarge, 500gb gp2):	×	<2.34
Databend (c6a.metal, 500gb gp2):	×	2.40
DuckDB (c6a.metal, 500gb gp2):	×	<2.60
QuestDB (partitioned) (c6a.metal, 500gb gp2)*:	×	<3.18
SingleStore (S24)*:	×	<3.90
DuckDB (c6a.4xlarge, 500gb gp2):	×	4.29
ClickHouse (c6a.4xlarge, 500gb gp2):	×	4.34
Databend (c6a.4xlarge, 500gb gp2):	×	4.45
chDB (DataFrame) (c6a.metal, 500gb gp2):	×	4.81
Databend (c5.4xlarge, 500gb gp2):	x	4.82
DuckDB (c5.4xlarge, 500gb gp2):	×	<5.07
Apache Doris (c6a.4xlarge, 500gb gp2):	×	<5.59
StarRocks (c6a.4xlarge, 500gb gp2):	×	<5.63
Snowflake (32×2XL):	×	\$.91
chDB (c6a.metal, 500gb gp2):	×	<6.09
Tablespace (L1 - 16CPU 32GB):	×	6.75
QuestDB (c6a.4xlarge, 500gb gp2):	×	(7.15
ByConity (c6a.4xlarge, 500gb gp2):	×	×7.56
ByteHouse (8×L):	x	<7.64

https://tinyurl.com/clickbench-2024-09

											-		
System:	All Alloy	DB Alloy	DB (tuned)	Athena (pa	rtitioned)	Athena	(single)	Aurora f	or MySQL	Aurora for	r PostgreS	QL ByC	Conity
	ByteHouse	chDB (D	ataFrame)	chDB (Parq	uet, partiti	oned)	chDB	Citus Clie	ckHouse C	oud (aws)	ClickHou	ise Cloud	(azure)
	ClickHouse	e Cloud (gc	p) ClickHo	use (data lak	ke, partition	ned) C	lickHous	e (data lake	e, single)	ClickHouse	e (Parquet,	partition	ed)
	ClickHouse	e (Parquet, s	single) Cli	ckHouse (we	eb) Click	House	ClickHo	use (tuned)) ClickHo	ouse (tuned,	, memory)	Cloud	erry
	CrateDB	Crunchy B	ridge for Ana	alytics (Parqu	uet) Data	bend	DataFusi	on (Parque	t, partitione	ed) DataF	usion (Pare	quet, sing	le)
	Apache Do	oris Drill	Druid Du	ickDB (DataF	Frame) D	uckDB (I	memory)	DuckDB	(Parquet,	partitioned)	DuckDE	B Elasti	csearch
	Elasticsear	ch (tuned)	GlareDB	Greenplum	HeavyA	Hydra	a Sale:	sforce Hyp	er (Parquet) Salesfo	rce Hyper	Infobri	ght
	Kinetica	MariaDB C	olumnStore	MariaDB	MonetDB	Mong	oDB N	lotherDuck	MySQL	(MyISAM)	MySQL	OctoSG	L
	Opteryx	Oxla Par	ndas (DataFra	ame) Para	deDB (Parc	quet, part	titioned)	ParadeD	B (Parquet	, single)			
	pg_duckdb	(MotherDu	ick enabled)	pg_duckd	b Postgi	reSQL wi	th pg_mo	oncake	Pinot Po	lars (DataFr	ame) Po	lars (Pare	quet)
	PostgreSQ	L (tuned)	PostgreSQL	QuestDB	Redshif	t Selec	tDB S	ingleStore	Snowflak	ke Spark	SQLite	StarRoc	ks
	Tablespace	a Tembo	OLAP (colun	nnar) Time	escale Clou	d Tim	escaleDE	(no colum	instore)	TimescaleD	B Tinybi	rd (Free 1	rial)
	Umbra \	/ictoriaLogs											
Type:	All C	column-ori	ented Pos	tgreSQL com	patible	manageo	d dcb	stateless	Java	C++ MvS	QL compa	tible	
.,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	row-orient	ed Clickh	louse deriva			rverless	datafra	ame aws	azure	analytical		_	document
		and the second	reSQL com			parquet	time-s			,			
Machine:		CPU 128GB	8 vCPU 6				_	e. 500ab a	D2 L M	A S XS	660 mg	tal. 500gl	0 002
Machine:		GiB 120Gi						, , ,		tics-256GE		, ,	51
								e, 500gb g					
		, 500gb gp2		rge, 1500gb	51		.8xlarge				a3.xlplus	S2 S2	24 2XL
	3XL 4XL	XL L1	- 16CPU 32	GB c6a.4x	large, 500g	gb gp3	16 vCPL	J 64GB	4 vCPU 16G	B 8 VCPL	J 32GB		
Cluster size:	All 1	2 4 8	16 32	64 128	serverless	1 2	3 u	ndefined					
Metric:	Cold Run	Hot Run	Load Time	Storage	Size								

System & Machine	Relative time (lower is better)	
Umbra (c6a.metal, 500gb gp2):		×1.51
Salesforce Hyper (c6a.metal, 500gb gp2):		×1.58
ClickHouse (c6a.metal, 500gb gp2):		×2.39
Umbra (c6a.4xlarge, 500gb gp2):		×2.43
SelectDB (c6a.metal, 500gb gp2):		×2.46
Apache Doris (c6a.metal, 500gb gp2):		×2.49
DuckDB (c6a.metal, 500gb gp2):		×2.73
StarRocks (c6a.metal, 500gb gp2):		×2.74
Databend (c6a.metal, 500gb gp2):		×2.84
chDB (c6a.metal, 500gb gp2):		×2.97
QuestDB (c6a.metal, 500gb gp2)*:		×3.04
Salesforce Hyper (Parquet) (c6a.metal, 500gb gp2):		×3.19
Salesforce Hyper (c6a.4xlarge, 500gb gp2):		×3.65
SingleStore (S24) ⁺ :		×4.62
DuckDB (c6a.4xlarge, 500gb gp2):		×4.89
Databend (c6a.4xlarge, 500gb gp2):		×5.26
ClickHouse (c6a.4xlarge, 500gb gp2):		×5.32
chDB (c6a.4xlarge, 500gb gp2):		×5.53
Databend (c5.4xlarge, 500gb gp2):		×5.71
QuestDB (c6a.4xlarge, 500gb gp2):		×6.35
Apache Doris (c6a.4xlarge, 500gb gp2):		×6.62

https://tinyurl.com/clickbench-2025-01-29

https://tinyurl.com/clickbench-2025-01-29

Same hardware (c6a.metal 500gb gp2)

System:	All Allo	yDB Alloy	/DB (tuned)	Athena (par	titioned)	Athena (single)	Aurora fo	r MySQL	Aurora	a for Postgre	SQL B	yConity	
	ByteHous	e chDB (I	DataFrame)	chDB (Parqu	uet, partitic	oned) c	hDB C	Clic Clic	kHouse (loud (av	vs) ClickH	ouse Clo	ud (azure)	
	ClickHous	se Cloud (go	p) ClickH	louse (data lak	e, partition	ed) Cli	ClickHouse (data lake, single) ClickHouse					e (Parquet, partitioned)		
	ClickHous	se (Parquet,	single) C	lickHouse (we	b) Click	House	ClickHouse (tuned) ClickH				ned, memory	y) Clou	dberry	
	CrateDB	Crunchy E	Bridge for Ar	nalytics (Parqu	et) Data	bend D	ataFusio	on (Parquet	partition	ed) Da	ataFusion (P	arquet, si	ngle)	
	Apache D	oris Drill	Druid D	DuckDB (DataF	rame) D	uckDB (m	emory)	DuckDB	(Parquet,	partition	ed) Duck	DB Ela:	sticsearch	
	Elasticsea	arch (tuned)	GlareDB	Greenplum	HeavyAl	Hydra	Sales	force Hype	r (Parque	t) Sale	esforce Hype	er Infok	pright	
	Kinetica	MariaDB C	olumnStore	MariaDB	MonetDB	Mongo	DB M	otherDuck	MySQI	(MyISA	M) MySQI	Octo	SQL	
	Opteryx	Oxla Pa	ndas (DataF	rame) Parac	deDB (Parq	uet, parti	ioned)	ParadeDE	8 (Parque	t, single)				
	pg_duckd	b (MotherD	uck enabled	i) pg_duckdi	Postgr	eSQL with	n pg_mo	oncake F	Pinot P	olars (Da	taFrame)	Polars (P	arquet)	
	PostgreS	QL (tuned)	PostgreSQ	QL QuestDB	Redshift	Select	DB Si	ngleStore	Snowfla	ike Sp	ark SQLite	StarR	ocks	
	Tablespac	ce Tembo	OLAP (colu	imnar) Time	scale Clou	d Time:	scaleDB	(no column	nstore)	Timesca	leDB Tiny	bird (Free	e Trial)	
	Umbra	VictoriaLog	S											
Туре:	All	column-or	iented Po	stgreSQL com	patible r	nanaged	gcp	stateless	Java	C++	MySQL com	patible		
	row-orien	ted Click	House deriv	ative embed	ided ser	verless	datafra	me aws	azure	analyti	cal Rust	search	document	
	Go son	newhat Post	greSQL com	npatible Data	Frame	parquet	time-se	eries						
Machine:	All 16 v	CPU 128GB	8 vCPU	64GB serve	rless 16a	acu c6a	.4xlarge	e, 500gb gp	2 L	M S	XS c6a.n	netal, 500	gb gp2	
	12GiB 8	BGiB 120G	iB 16GiB	236GiB 3	2GiB 64	GiB c5	n.4xlarg	e, 500gb gp	2 Ana	lytics-25	6GB (64 vC	ores, 256	GB)	
	c5.4xlarg	e, 500gb gp	2 c6a.4xl	large, 1500gb g	gp2 clou	d dc2.8	Bxlarge	ra3.16xlar	rge ra3	.4xlarge	ra3.xlplus	s S2	S24 2XL	
	3XL 4X	L XL L	I - 16CPU 32	2GB c6a.4xl	arge, 500g	b gp3	16 vCPU	64GB 4	vCPU 16	GB 8 v	CPU 32GB			
Cluster size:	All 1	2 4 8	16 32	64 128 9	serverless	1 2	3 un	defined						
Metric:	Cold Run	Hot Run	Load Tim	ne Storage	Size									

System & Machine	Relative time (lower is better)
Umbra (c6a.metal, 500gb gp2):	×1.31
Salesforce Hyper (c6a.metal, 500gb gp2):	×1.37
ClickHouse (c6a.metal, 500gb gp2):	×2.07
SelectDB (c6a.metal, 500gb gp2):	×2.14
Apache Doris (c6a.metal, 500gb gp2):	×2.15
DuckDB (c6a.metal, 500gb gp2):	×2.37
StarRocks (c6a.metal, 500gb gp2):	×2.38
Databend (c6a.metal, 500gb gp2):	×2.46
chDB (c6a.metal, 500gb gp2):	×2.57
QuestDB (c6a.metal, 500gb gp2) [†] :	×2.64
Salesforce Hyper (Parquet) (c6a.metal, 500gb gp2):	×2.77
Polars (Parquet) (c6a.metal, 500gb gp2):	×14.14
Polars (DataFrame) (c6a.metal, 500gb gp2):	×14.39
GlareDB (c6a.metal, 500gb gp2):	×97.33
Pandas (DataFrame) (c6a.metal, 500gb gp2):	×260.04

QuestDB in ClickBench: how it started

System & Machine	Relative time (lower is better)
ClickHouse (c6a.metal, 500gb gp2):	×1.65
DuckDB (c6a.metal, 500gb gp2):	×2.00
QuestD8 8.0 (c6a.metal, 500gb gp2)*:	×2.45
DuckDB (c6a.4xlarge, 500gb gp2):	×3.30
ClickHouse (c6a.4xlarge, 500gb gp2):	×3.34
QuestD8 8.0 (c6a.4xlarge, 500gb gp2):	×5.49
DataFusion (Parquet) (c6a.4xlarge, 500gb gp2)*:	×10.44
SingleStore (c6a.4xlarge, S00gb gp2)*:	×14.66
Pinot (c6a.4xiarge, 500gb gp2)*:	×35.14
PostgreSQL (tuned) (c6a.4xlarge, 500gb gp2):	×48.56
CrateDB (c6a.4xlarge, 500gb gp2)*:	×74.52
TimescaleDB (compression) (c6a.4xlarge, 500gb gp2):	×106.02
Druid (c6a.4xlarge, 500gb gp2)*:	×183.58
QuestDB 6.4.1 (c6a.4xlarge, 500gb gp2) [†] :	×213.03
MongoDB (c6a.4xlarge, 500gb gp2):	×489.09
TimescaleDB (c6a.4xlarge, 500gb gp2):	×1039.91
SQLite (c6a.4xlarge, 500gb gp2):	×1052.69

The Journey, or There and Back Again

- ~2 years of calendar time
- Done along with major features: Write-Ahead-Log (WAL), replication, window functions, Parquet and JSON support, etc.
- ~80 patches, including community contributions
- A number of failed optimization attempts
- Even more plans for further steps

Trivial steps

- Added missing SQL functions, e.g. count_distinct() for integer column types or max()/min() on strings
- Reduced memory footprint of some SQL functions to avoid OOM crashes

SELECT RegionID, count_distinct(UserID) AS u
FROM hits
GROUP BY RegionID
ORDER BY u DESC
LIMIT 10;

QuestDB's JIT compiler

- SQL JIT compiler for filters (WHERE clauses)
- Backend is written in C++ with asmjit library, frontend is in Java
- Emits SIMD (AVX-2) instructions for a subset of filters
- JIT compiled (and Java) filter execution is multi-threaded

SELECT count(*)
FROM hits
WHERE AdvEngineID <> 0;

Predicate Compile Time:

SQL (Java) -> AST (Java) -> IR (Java) -> machine code (C++)

Predicate Execution Time:

public	static native long callFunction(
	long fnAddress,
	long colsAddress,
	long colsSize,
	<pre>long varSizeIndexesAddress,</pre>
	<pre>long varsAddress,</pre>
	long varsSize,
	<pre>long rowsAddress,</pre>
	long rowsSize,
	<pre>long rowsStartOffset</pre>
);	

inline Ymm cmp_eq_double(Compiler &c, data_type_t type, const Ymm &lhs, const Ymm &rhs) {

```
Ymm lhs_copy = c.newYmm();
Ymm rhs_copy = c.newYmm();
c.vmovapd(lhs_copy, lhs);
c.vmovapd(rhs_copy, rhs);
Ymm dst = c.newYmm();
Ymm nans = mask_and(c, is_nan(c, type, lhs_copy), is_nan(c, type, rhs_copy));
Mem sign_mask = vec_sign_mask(c, type);
c.vsubpd(lhs_copy, lhs_copy, rhs_copy); //(lhs - rhs)
c.vpand(lhs_copy, lhs_copy, sign_mask); // abs(lhs - rhs)
double eps[4] = {DOUBLE_EPSILON, DOUBLE_EPSILON, DOUBLE_EPSILON, DOUBLE_EPSILON};
Mem epsilon = c.newConst(ConstPool::kScopeLocal, &eps, 32);
c.vcmppd(dst, lhs_copy, epsilon, Predicate::kCmpLT);
c.vpor(dst, dst, nans);
return dst;
```

JIT compiler improvements

• Expanded supported operators and types

```
SELECT URL, count(*) AS PageViews
FROM hits
WHERE CounterID = 62
  AND EventTime >= '2013-07-01T00:00:00Z'
  AND EventTime <= '2013-07-31T23:59:59Z'
  AND DontCountHits = 0
  AND IsRefresh = 0
  AND URL IS NOT NULL
GROUP BY URL
ORDER BY PageViews DESC
LIMIT 10;
```

SQL rewrites

```
SELECT count_distinct(SearchPhrase)
FROM hits;
```

```
-- gets rewritten into:
SELECT count(*)
FROM (
   SELECT SearchPhrase
   FROM hits
   WHERE SearchPhrase IS NOT NULL
   GROUP BY SearchPhrase
```

);

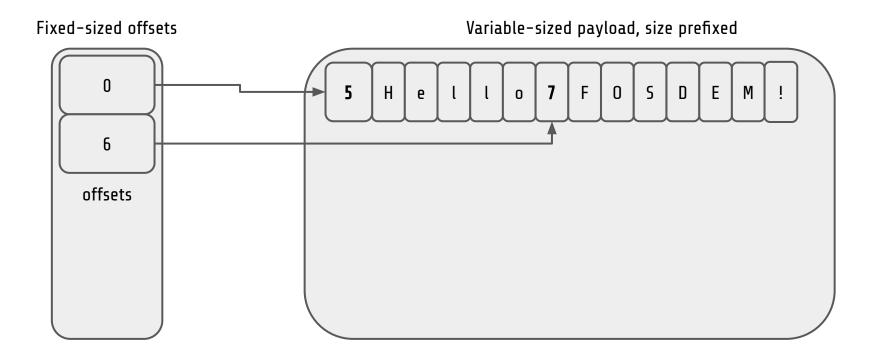
SQL function optimizations #1

-- uses SWAR-based LIKE operator implementation
SELECT count(*)
FROM hits
WHERE URL LIKE '%google%';

SQL function optimizations #2

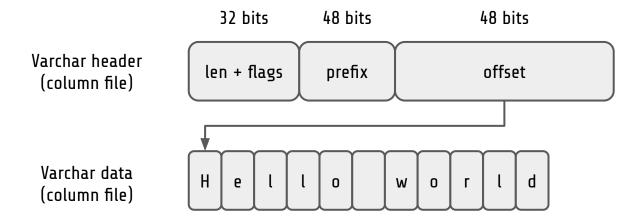
-- regexp_replace() uses Java regular expressions, but with a few fast paths **SELECT** * FROM (SELECT regexp_replace(Referer, '^https?://(?:www\.)?([^/]+)/.*\$', '\$1') AS k, avg(length(Referer)) AS 1, count(*) AS c, min(Referer) FROM hits WHERE Referer IS NOT NULL GROUP BY k WHERE c > 100000 ORDER BY 1 DESC LIMIT 25;

Old STRING column type - UTF-16 encoded



New VARCHAR column type

- Introduced VARCHAR type (UTF-8) instead of old STRING type (UTF-16)
- Layout is similar to what Andy Pavlo calls "German Strings", but with some differences, including an ASCII bit flag



The elephant in the room

• Only a few GROUP BY queries ran parallel (and used SIMD)

```
SELECT sum(AdvEngineID), count(*), avg(ResolutionWidth) FROM hits;
```

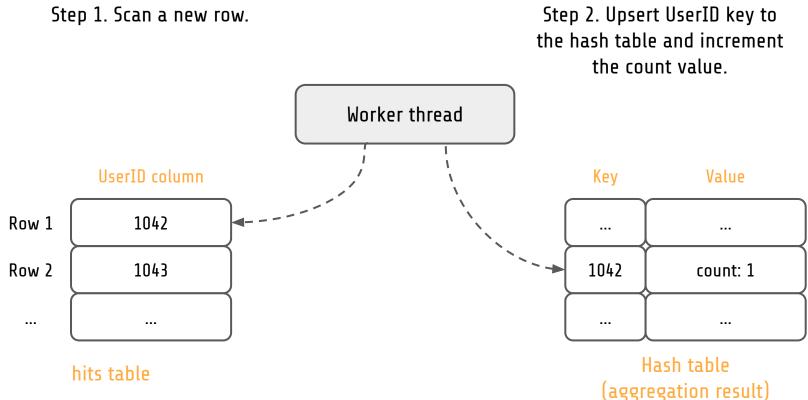
```
SELECT avg(UserID) FROM hits;
```

```
SELECT min(EventDate), max(EventDate) FROM hits;
```

SELECT sum(ResolutionWidth), sum(ResolutionWidth + 1), -- many more sums...
FROM hits;

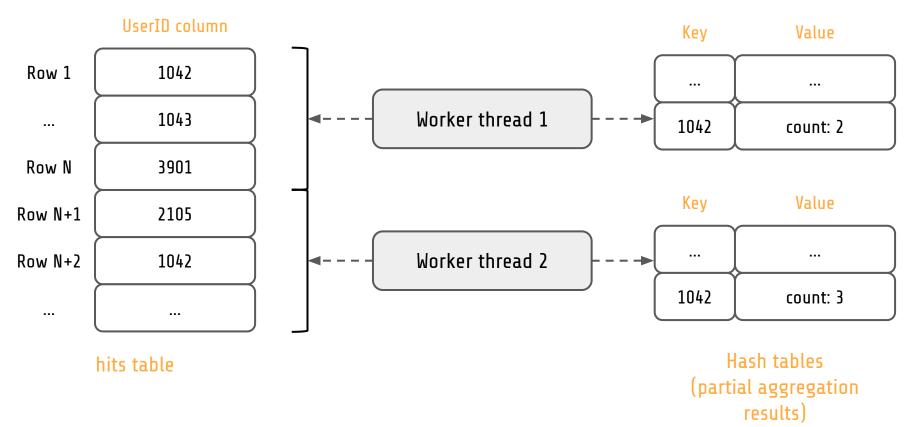
How do you implement a GROUP BY?

SELECT UserID, count(*) AS c
FROM hits
GROUP BY UserID;

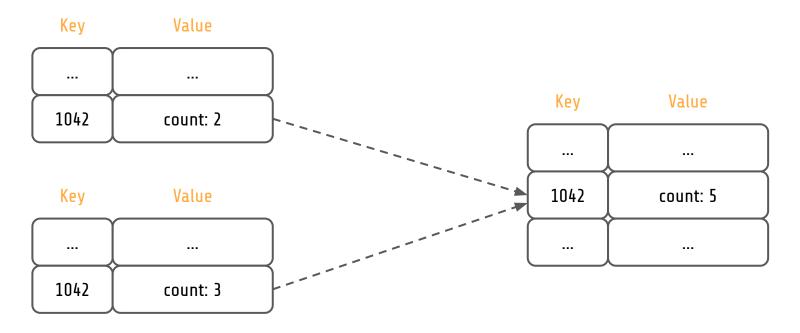


Single-threaded GROUP BY

(-35.654.00.1654



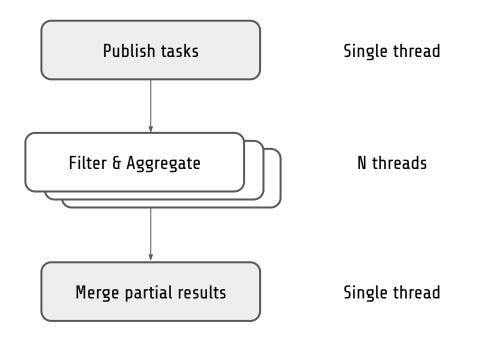
Multi-threaded GROUP BY: aggregation



Hash tables (partial aggregation results)

Hash table (final aggregation result)

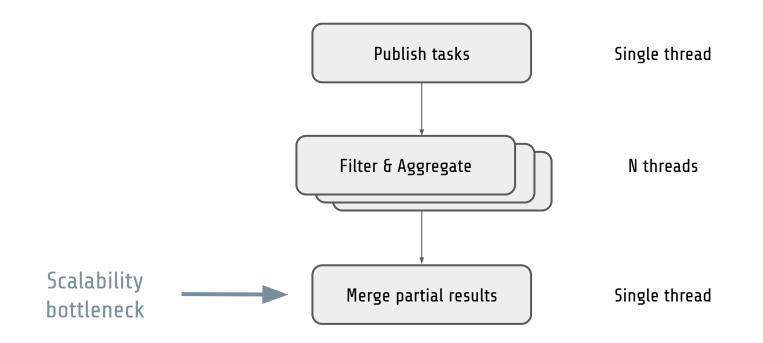
Multi-threaded GROUP BY: merge



Multi-threaded GROUP BY pipeline

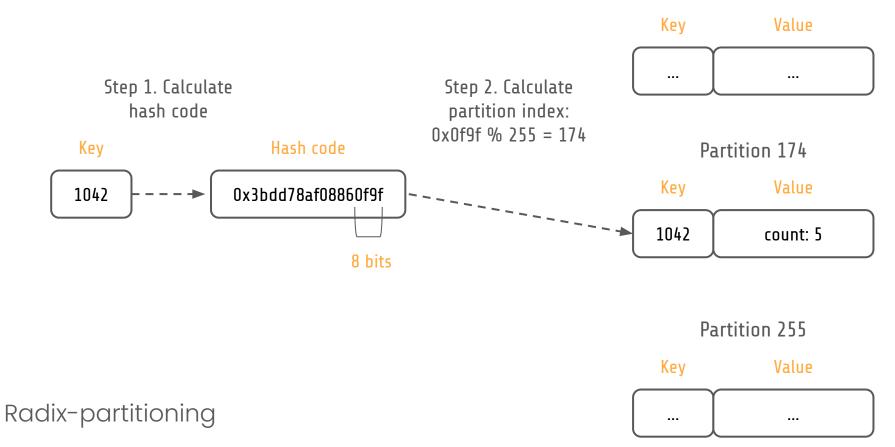
Parallel GROUP BY v1: any good?

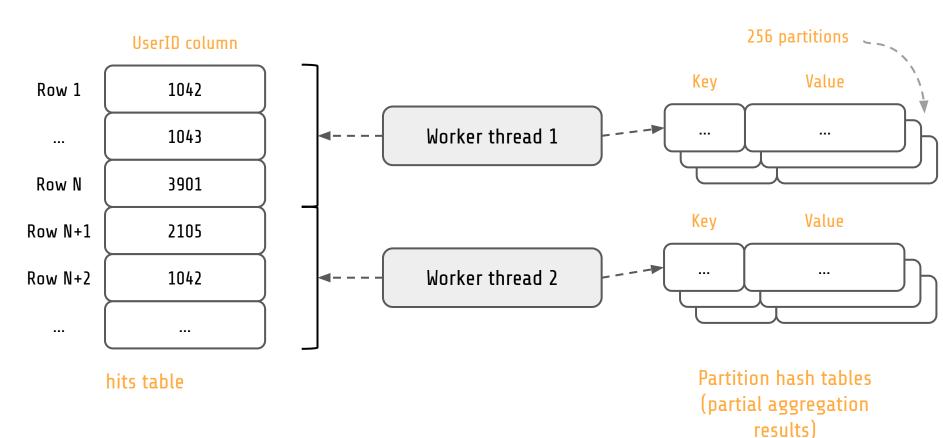
- Simple pipeline, easy to implement
- Scales nicely when there are not so many groups (distinct UserID values)
- Yet, high cardinality (>= 100K groups) is a problem



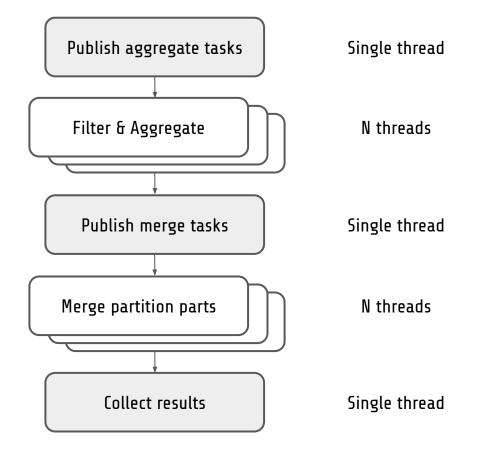
Multi-threaded GROUP BY pipeline: the cardinality problem

Partition 0





High-cardinality multi-threaded GROUP BY



High-cardinality multi-threaded GROUP BY pipeline

Parallel GROUP BY v2

- More complex pipeline, a bit harder to implement
- Scales nicely for any cardinality
- Potentially parallel ORDER BY + LIMIT when the cardinality is high
- Used for multi-threaded GROUP BY and SAMPLE BY

SELECT RegionID, count_distinct(UserID) AS u FROM hits GROUP BY RegionID ORDER BY u DESC LIMIT 10;

The more hash tables, the merrier

- Introduced a number of specialized hash tables
- All use open addressing with linear probing
- Some preserve insertion order

So far, we have:

- A "general purpose" hash table for variable-size keys
- Hash tables with small fixed-size keys (32-bit and 64-bit integers)
- A lookup table for 16-bit keys
- A hash table for single VARCHAR key

Lessons learned

- A fast time-series database must be a good analytical database
- Benchmarks made by 3rd-parties help when deciding what to optimize
- Improving query engine efficiency requires discipline
- As a nice side effect, we made SAMPLE BY run parallel
- We have lots of plans for the next steps.

https://github.com/questdb/questdb https://questdb.io https://demo.questdb.io https://slack.questdb.io/ https://github.com/questdb/time-series-streaming-analytics-template



Jaromir Hamala

Core Engineer at QuestDB

Javier Ramirez

Database Advocate at QuestDB

@supercoco9@supercoco9.bsky.social@j@chaos.sociallinkedin.com/in/ramirez

