

Data Warehouse Benchmark:

Redshift

Me



VS **Snowflake**

VS **BigQuery**



CEO

@



Fivetran



Applications

Asana	Instagram	Stripe
Bing Ads	Intercom	Xero
Braintree Payments	iTunes	Zendesk
Desk.com	Jira	Zendesk Chat (Zopim)
DoubleClick	Magento	Zuora
Dynamics (365, GP, AX)	MailChimp	
Eloqua	Mandrill	
Facebook Ad Insights	Marketo	
Freshdesk	Mixpanel	
FrontApp	NetSuite SuiteAnalytics	
Github	Pardot	
Google Adwords	QuickBooks Online	
Google Analytics	Recurly	
Google Analytics 360	Sailthru	
Google Play	Salesforce	
Help Scout	SalesforceIQ	
HubSpot	SAP Business One	
Hybris	Shopify	



Databases

Amazon Aurora	Amazon Cloudfront
Amazon RDS	Amazon Kinesis Firehose
Azure SQL Database	Amazon S3
DynamoDB	Azure Blob Storage
Google Cloud SQL	CSV
Heroku	Dropbox
MariaDB	FTP
MongoDB	FTPS
MySQL	Google Cloud Storage
Oracle DB	Google Sheets
PostgreSQL	JSON
SQL Server	SFTP



Files



Events

Segment	Webhooks
Snowplow	

Online Transaction Processing (**OLTP**)

```
select *  
from github.commit  
where sha = 'feec5a81da13e95a1911b09773f8228f8c0db76'
```

is very different from
Online Analytical Processing (**OLAP**)

```
select author_email, count()  
from github.commit  
group by 1
```

This talk is about **OLAP**!

Row Store:

commit	file	added	removed	changed
xxx	file1.txt	1	10	11
xxx	file2.txt	100	0	100
xxx	file3.txt	50	50	50
yyy	file1.txt	1	10	11

xxx, file1.txt, 1, 10, 11, xxx, file2.txt, 100, 0, 100, xxx, file3.txt, 50, 50, 50, yyy, file1.txt, 1, 10, 11

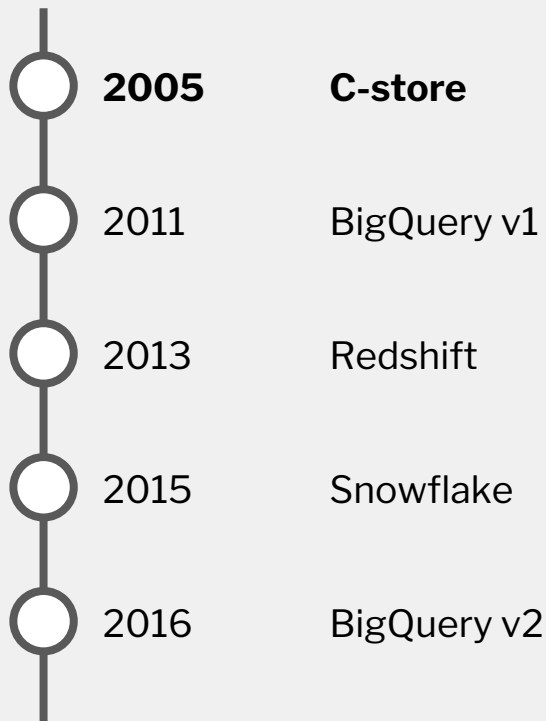
Column Store:

commit	file	added	removed	changed
xxx	file1.txt	1	10	11
xxx	file2.txt	100	0	100
xxx	file3.txt	50	50	50
yyy	file1.txt	1	10	11

xxx, xxx, xxx, yyy, file1.txt, file2.txt, file3.txt, 1, 100, 50, 1, 10, 0, 50, 10, 11, 100, 50, 11

```
select file, sum(changed)
from github.commit
group by 1
```

C-store: the data warehouse that changed everything



C-Store: A Column-oriented DBMS

Mike Stonebraker^{*}, Daniel J. Abadi^{*}, Adam Batkin⁺, Xuedong Chen[†], Mitch Cherniack⁺, Miguel Ferreira^{*}, Edmond Lau^{*}, Amerson Lin^{*}, Sam Madden^{*}, Elizabeth O'Neil[†], Pat O'Neil[†], Alex Rasin[‡], Nga Tran⁺, Stan Zdonik[‡]

^{*}MIT CSAIL
Cambridge, MA

⁺Brandeis University
Waltham, MA

[†]UMass Boston
Boston, MA

[‡]Brown University
Providence, RI

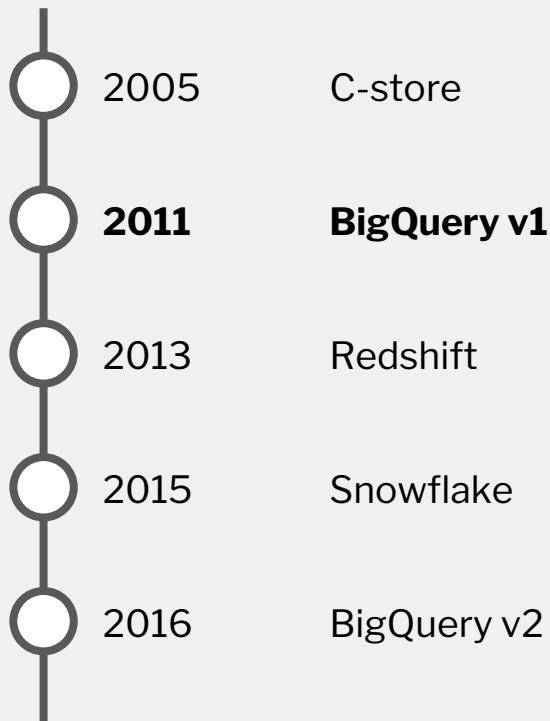
Abstract

This paper presents the design of a read-optimized relational DBMS that contrasts sharply with most current systems, which are write-optimized.

in which periodically a bulk load of new data is performed, followed by a relatively long period of ad-hoc queries. Other read-mostly applications include customer relationship management (CRM) systems, electronic library card catalogs, and other ad-hoc inquiry systems. In such environments, a column store architecture, in which

2011: Early BigQuery

Not so great at joins

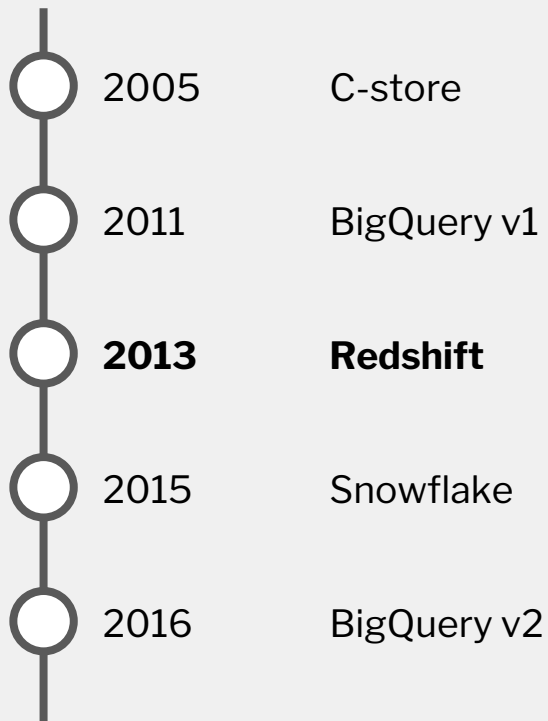


```
select foo, bar
from large_table
join other_large_table
```

Nonstandard SQL-like language

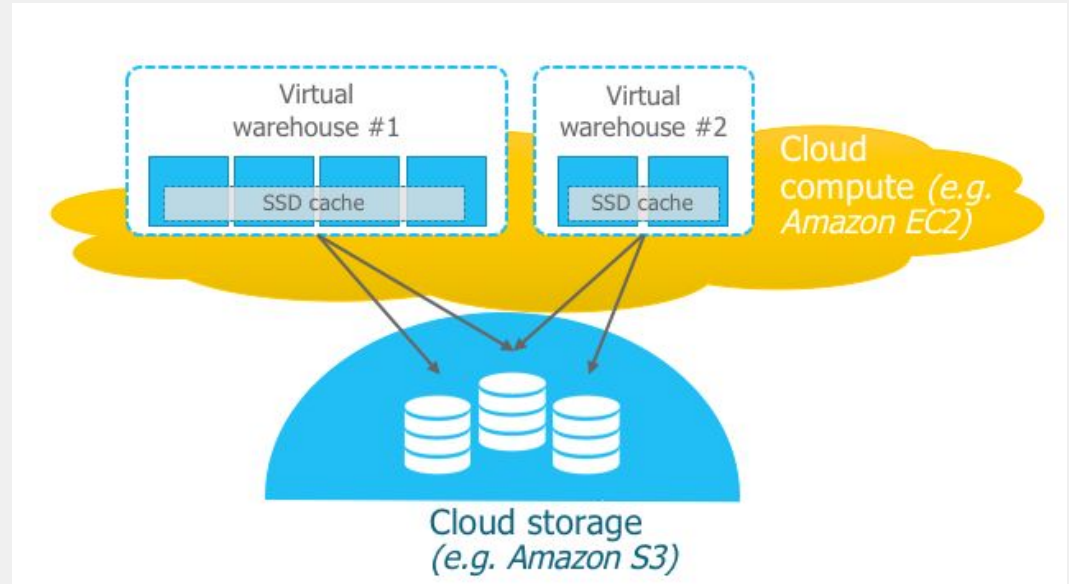
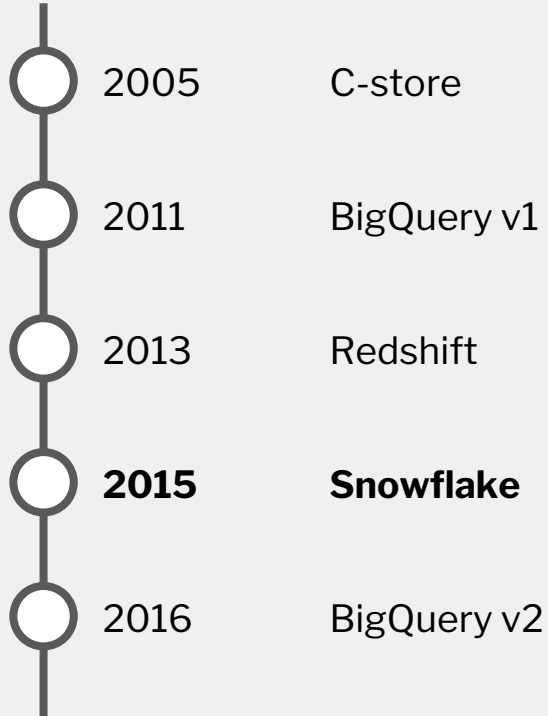
```
select why, did, you,
invent, your, own, sql
from google
```

2013: AWS Redshift takes off

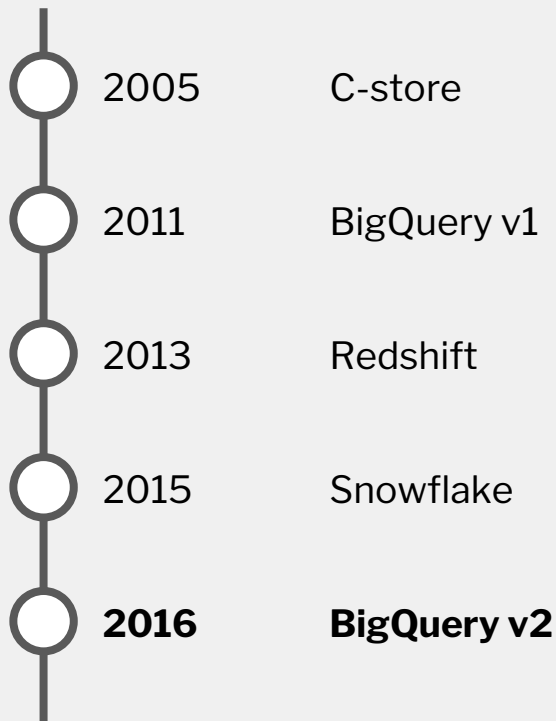


Snowflake: store the data in S3!

(similar to BigQuery)



2016: BigQuery gets way better



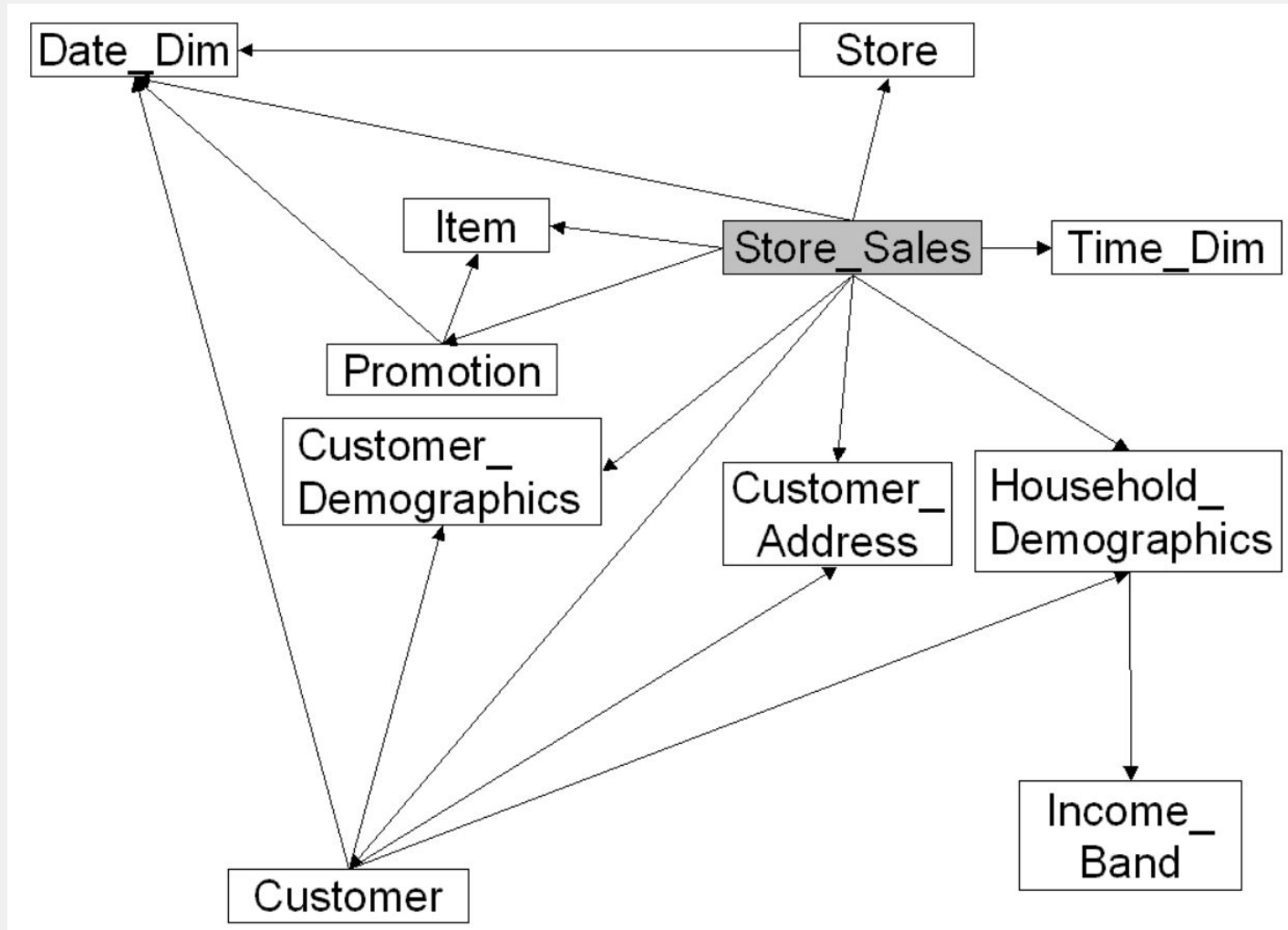
Fact-to-fact joins work!
Standard SQL!
DELETE and UPDATE!

```
update mytable  
set name = 'Hello world!'  
where id = 1
```

Benchmark time!



What data did we query?



What queries did we run?

```
-- query12
SELECT
    i_item_id ,
    i_item_desc ,
    i_category ,
    i_class ,
    i_current_price ,
    Sum(ws_ext_sales_price)
    Sum(ws_ext_sales_price)*100/Sum(Sum(ws_ext_sales_price))
FROM
    web_sales ,
    item ,
    date_dim
WHERE
    ws_item_sk = i_item_sk
AND
    i_category IN ('Home',
                  'Men',
                  'Women')
AND
    ws_sold_date_sk = d_date_sk
AND
    Cast(d_date AS DATE) BETWEEN Cast('2000-05-11' AS DATE)
    Cast('2000-06-11' AS DATE))
GROUP BY i_item_id ,
         i_item_desc ,
         i_category ,
         i_class ,
         i_current_price
ORDER BY i_category ,
         i_class ,
         i_item_id ,
         i_item_desc ,
         revenueratio
LIMIT 100;
```

```
-- query5
WITH ssr AS
(
    SELECT
        s_store_id,
        Sum(sales_price) AS sales,
        Sum(profit) AS profit,
        Sum(return_amt) AS returns1,
        Sum(net_loss) AS profit_loss
    FROM
        (
            SELECT
                ss_store_sk AS store_sk,
                ss_sold_date_sk AS date_sk,
                ss_ext_sales_price AS sales_price,
                ss_net_profit AS profit,
                0 AS return_amt,
                0 AS net_loss
            FROM
                store_sales
            UNION ALL
            SELECT
                sr_store_sk AS store_sk,
                sr_returned_date_sk AS date_sk,
                0 AS sales_price,
                0 AS profit,
                sr_return_amt AS return_amt,
                sr_net_loss AS net_loss
            FROM
                store_returns ) salesreturns,
        date_dim,
        store
    WHERE
        date_sk = d_date_sk
    AND
        Cast(d_date AS DATE) BETWEEN Cast('2002-08-22' AS DATE) AND
        Cast('2002-09-05' AS DATE))
    AND
        store_sk = s_store_sk
GROUP BY s_store_id) , csr AS
(
    SELECT
        cp_catalog_page_id,
        sum(sales_price) AS sales,
        sum(profit) AS profit,
        sum(return_amt) AS returns1,
        sum(net_loss) AS profit_loss
    FROM
        (
            SELECT
                cs_catalog_page_sk AS page_sk,
                cs_sold_date_sk AS date_sk,
                cs_ext_sales_price AS sales_price,
                cs_net_profit AS profit,
                0 AS return_amt,
                0 AS net_loss
            FROM
                catalog_sales
            UNION ALL
            SELECT
                cr_catalog_page_sk AS page_sk,
                cr_returned_date_sk AS date_sk,
                0 AS sales_price,
                0 AS profit,
                cr_return_amount AS return_amt,
                cr_net_loss AS net_loss
            FROM
                catalog_returns ) salesreturns,
        date_dim,
        catalog_page
    WHERE
        date_sk = d_date_sk
    AND
        Cast(d_date AS DATE) BETWEEN cast('2002-08-22' AS date) AND
        Cast('2002-09-05' AS DATE))
    AND
        page_sk = cp_catalog_page_sk
GROUP BY cp_catalog_page_id) , wsr AS
(
    SELECT
        web_site_id,
        sum(sales_price) AS sales,
        sum(profit) AS profit,
        sum(return_amt) AS returns1,
        sum(net_loss) AS profit_loss
    FROM
        (
            SELECT
                ws_web_site_sk AS wsr_web_site_sk,
                ws_sold_date_sk AS date_sk,
                ws_ext_sales_price AS sales_price,
                ws_net_profit AS profit
```

What is TPC-DS?

TPC TPC - Who We Are

www.tpc.org/information/who/howeare.asp

TPC™

benchmark standards and disseminating objective, verifiable performance data to the industry... The TPC is a non-profit corporation











Home
About the TPC
Benchmarks
Newsletter
Join the TPC
Downloads
Technical Articles
TPCTC

Who We Are

- [Full Members](#)
- [Associate Members](#)
- [Professional Affiliates](#)
- [TPC Auditors](#)
- [Honor Roll](#)

If you would like to reach a representative from a member company, please contact the TPC Administrator at: Admin@TPC.org

Full Members

How to run TPC-DS without cheating

DON'T run the same query twice

DON'T use dist keys

DON'T use sort/partition keys

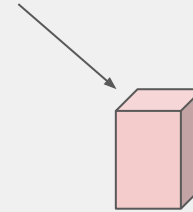
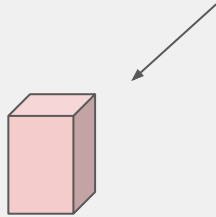
DO apply compression encoding

DO use a realistic (small) scale

DO compare cost

DON'T use dist keys

```
select *  
from web_sales  
join item on ws_item_sk = i_item_sk
```



WEB_SALES

ws_sold_date_sk	ws_sales_price	ws_item_sk
Jan 1 2000	\$1	1
Jan 2 2000	\$1	1
Feb 1 2000	\$1	1

ITEMS

i_item_sk	i_product_name
1	Rubber Ducks

WEB_SALES

ws_sold_date_sk	ws_sales_price	ws_item_sk
Feb 1 2000	\$10	2
Aug 1 2000	\$10	2
Jan 1 2001	\$10	2

ITEMS

i_item_sk	i_product_name
2	Pinwheels

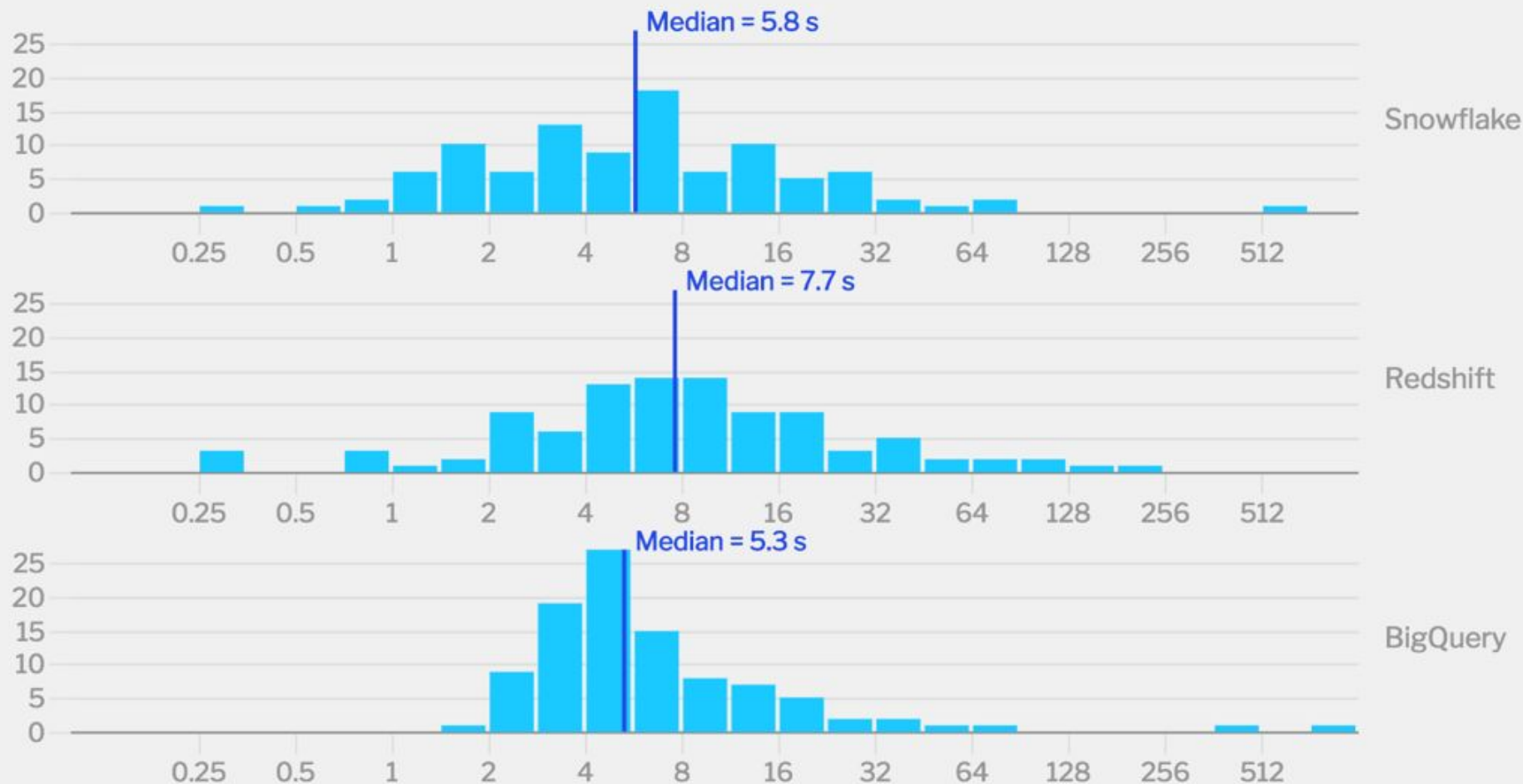
DON'T use sort/partition keys

```
select *  
from web_sales  
join item on ws_item_sk = i_item_sk  
where d_date between '2000-05-11' and '2000-06-11'
```

WEB_SALES		
ws_sold_date_sk	ws_sales_price	ws_item_sk
2000-01-01	\$1	1
2000-01-07	\$10	2
...
2000-05-11	\$1	1
2000-05-20	\$10	2
2000-05-30	\$1	1
2000-06-05	\$10	2
2000-06-11	\$10	2
...
2000-12-01	\$1	1
2000-12-31	\$10	2

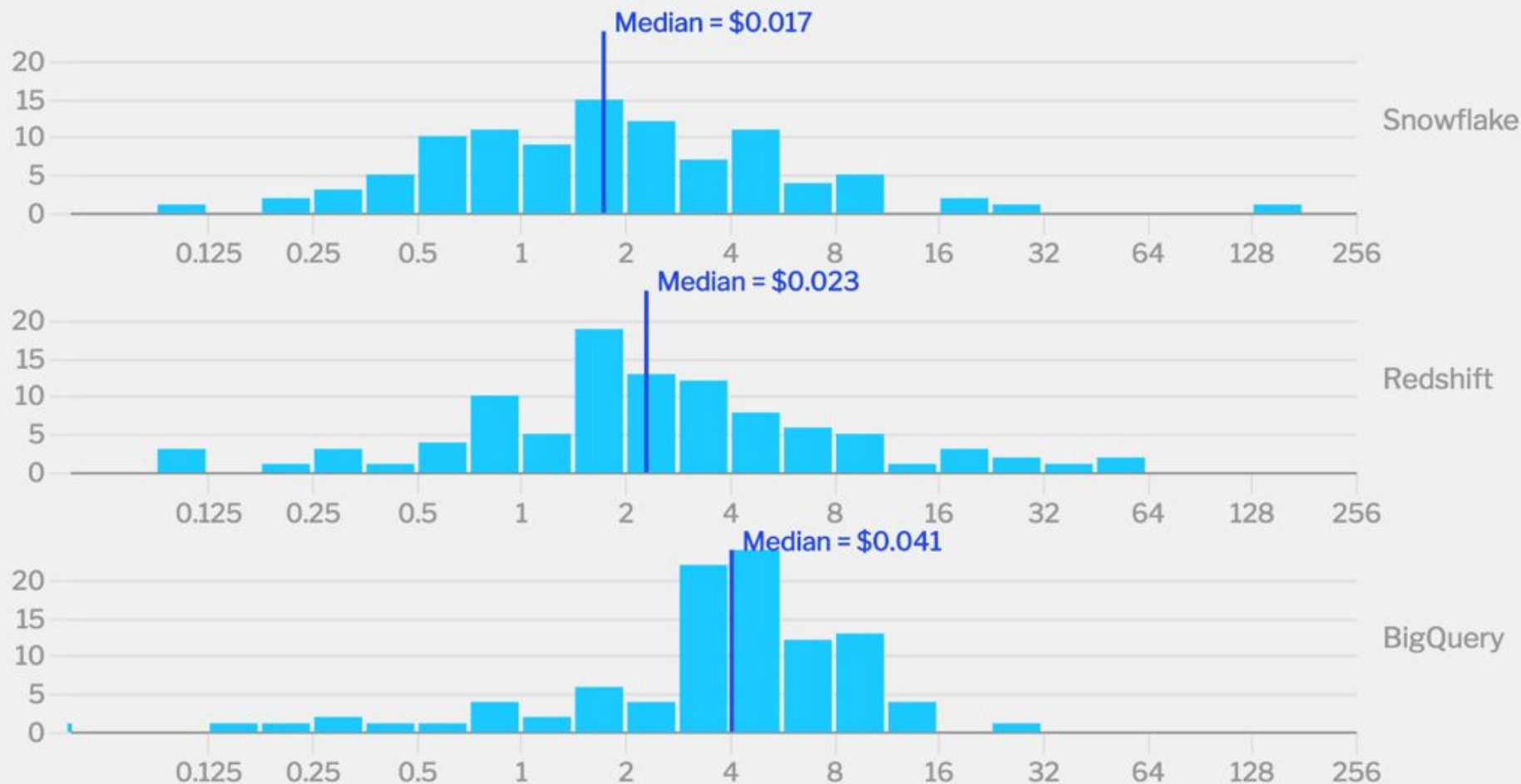
Which data warehouse is the fastest?

Histogram of execution times for 99 TPC-DS queries (seconds, log scale)



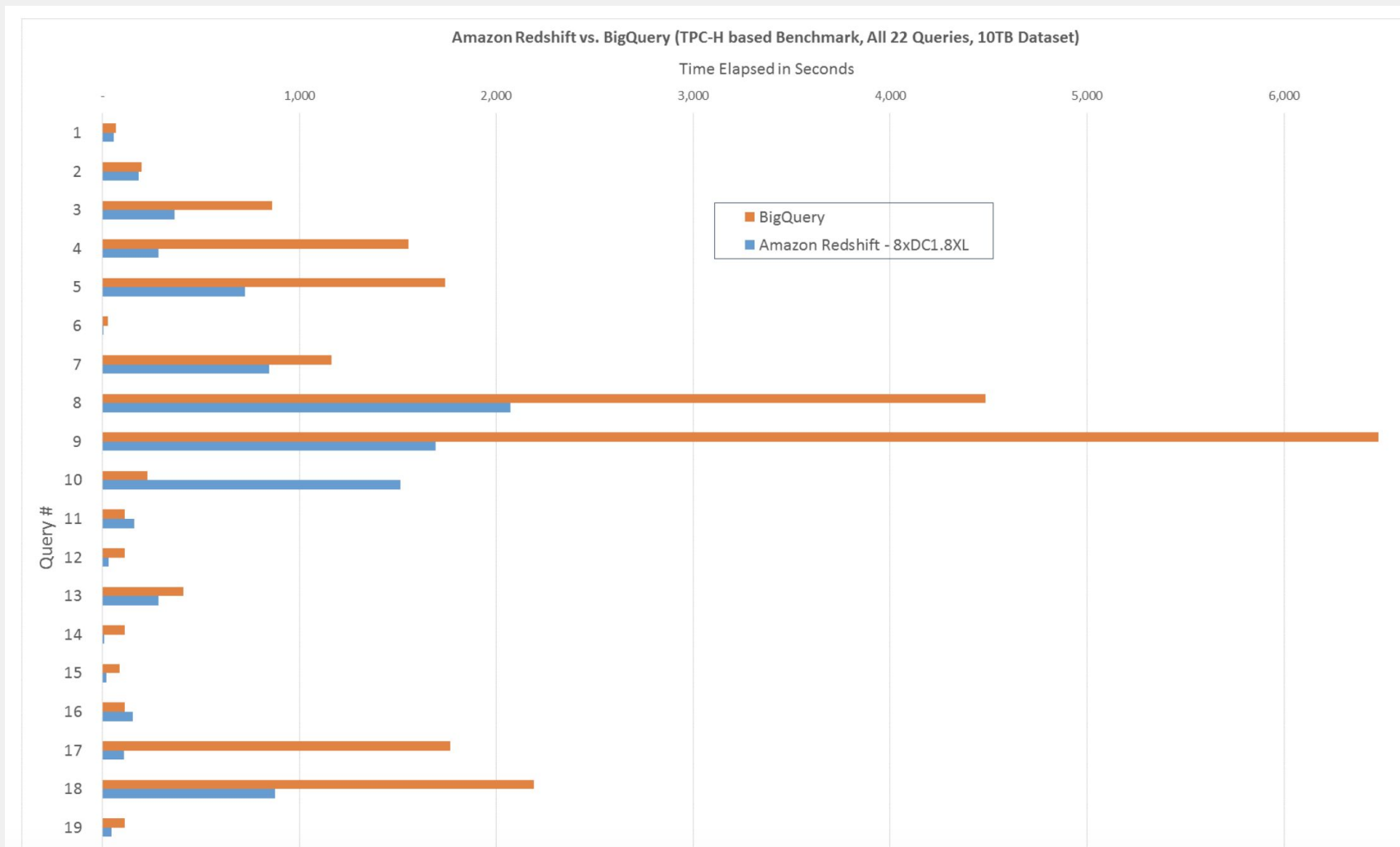
Which data warehouse is the cheapest?

Histogram of cost for 99 TPC-DS queries (cents, log scale)



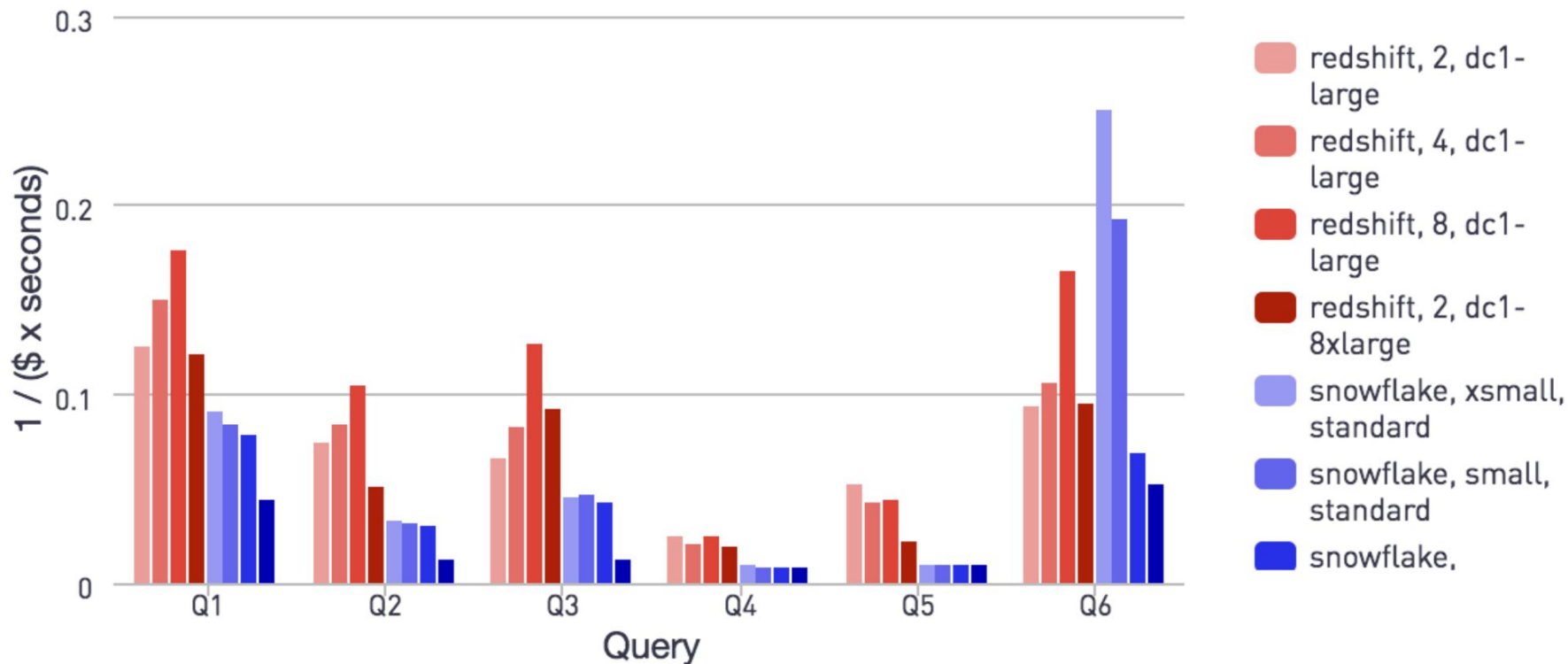
How does this compare to
other benchmarks?

Amazon's Redshift vs BigQuery benchmark



Periscope's Redshift vs Snowflake vs BQ

Snowflake vs. Redshift: Computation per Dollar [Higher is Better]



Mark Litwintshik's 1.1 billion taxi-rides

Query 1	Query 2	Query 3	Query 4	Setup
0.005	0.011	0.103	0.188	<u>BrytlytDB 2.1 & 5-node IBM Minsky cluster</u>
0.009	0.027	0.287	0.428	<u>BrytlytDB 2.0 & 2-node p2.16xlarge cluster</u>
0.021	0.053	0.165	0.51	<u>MapD & 8 Nvidia Pascal Titan Xs</u>
0.027	0.083	0.163	0.891	<u>MapD & 8 Nvidia Tesla K80s</u>
0.028	0.2	0.237	0.578	<u>MapD & 4-node g2.8xlarge cluster</u>
0.034	0.061	0.178	0.498	<u>MapD & 2-node p2.8xlarge cluster</u>
0.036	0.131	0.439	0.964	<u>MapD & 4 Nvidia Titan Xs</u>
0.051	0.146	0.047	0.794	<u>kdb+/q & 4 Intel Xeon Phi 7210 CPUs</u>
0.762	2.472	4.131	6.041	<u>BrytlytDB 1.0 & 2-node p2.16xlarge cluster</u>
1.034	3.058	5.354	12.748	<u>ClickHouse, Intel Core i5 4670K</u>
1.56	1.25	2.25	2.97	<u>Redshift, 6-node ds2.8xlarge cluster</u>
2	2	1	3	<u>BigQuery</u>
4	4	10	21	<u>Presto, 50-node n1-standard-4 cluster</u>
6.41	6.19	6.09	6.63	<u>Amazon Athena</u>
8.1	18.18	n/a	n/a	<u>Elasticsearch (heavily tuned)</u>
10.19	8.134	19.624	85.942	<u>Spark 2.1, 11 x m3.xlarge cluster w/ HDFS</u>
11	10	21	31	<u>Presto, 10-node n1-standard-4 cluster</u>
14.389	32.148	33.448	67.312	<u>Vertica, Intel Core i5 4670K</u>
34.48	63.3	n/a	n/a	<u>Elasticsearch (lightly tuned)</u>
35	39	64	81	<u>Presto, 5-node m3.xlarge cluster w/ HDFS</u>
43	45	27	44	<u>Presto, 50-node m3.xlarge cluster w/ S3</u>
152	175	235	368	<u>PostgreSQL 9.5 & cstore_fdw</u>
264	313	620	961	<u>Spark 1.6, 5-node m3.xlarge cluster w/ S3</u>
1103	1198	2278	6446	<u>Spark 2.2, 3-node Raspberry Pi cluster</u>

* 20 points by georgewfraser 2 hours ago [-]

This is not a good benchmark. There are two problems:

1. It's a simple GROUP BY on a single table. You're basically just measuring the scan speed. Real queries are dominated by shuffles and the probe side of joins; these aren't even present in this benchmark.
2. He runs the query repeatedly and takes the fastest time. This is far too cache-friendly. In this example, the intermediate stages of the query or even the result are probably just sitting in memory on the nodes after the first couple runs.

If you want to measure the performance of a data warehouse, you need to use more complex queries and not run the *exact same* query repeatedly.

edit: Coincidentally, I am giving a talk about data warehouse benchmarking TONIGHT in NYC. If you're in NY and interested in this subject, please come! <https://www.meetup.com/mysqlnyc/>

marklit 2 hours ago [-]

The benchmarks are aimed at OLAP not OLTP workloads.

If I had a large query set to run on each vendor I'd be more likely to hit compatibility issues which could mean fewer benchmarks going out. As it is I spend a lot of time getting hardware and software vendors together for these benchmarks.

If caches were being hit I'd expect a lot more DBs hitting single millisecond times in my benchmarks but as far as I can see, there is a clear delta between the various setups I've tested:

<http://tech.marksblogg.com/benchmarks.html>

What really matters: **ease of use**





Applications

Asana	Instagram	Stripe
Bing Ads	Intercom	Xero
Braintree Payments	iTunes	Zendesk
Desk.com	Jira	Zendesk Chat (Zopim)
DoubleClick	Magento	Zuora
Dynamics (365, GP, AX)	MailChimp	
Eloqua	Mandrill	
Facebook Ad Insights	Marketo	
Freshdesk	Mixpanel	
FrontApp	NetSuite SuiteAnalytics	
Github	Pardot	
Google Adwords	QuickBooks Online	
Google Analytics	Recurly	
Google Analytics 360	Sailthru	
Google Play	Salesforce	
Help Scout	SalesforceIQ	
HubSpot	SAP Business One	
Hybris	Shopify	



Databases

Amazon Aurora	Amazon Cloudfront
Amazon RDS	Amazon Kinesis Firehose
Azure SQL Database	Amazon S3
DynamoDB	Azure Blob Storage
Google Cloud SQL	CSV
Heroku	Dropbox
MariaDB	FTP
MongoDB	FTPS
MySQL	Google Cloud Storage
Oracle DB	Google Sheets
PostgreSQL	JSON
SQL Server	SFTP



Files



Events

Segment	Webhooks
Snowplow	