

Query Processing in PostgreSQL Database

Performance Benchmarking

Master of Computing & Information Systems Project
Advisor: Dr. Feng Yu



**YOUNGSTOWN
STATE
UNIVERSITY**

Tasha Wells

Purpose

To obtain hands on experience working with PostgreSQL database benchmarking.

Summary

Benchmarking conducted between Hadoop with Hive and PostgreSQL database. Fifteen queries and the 1 GB dataset from TPCH were used as well as a sample dataset derived from the TPCH 1 GB dataset using CS2 method for sampling. Additionally, EXPLAIN SQL statement used to review Postgres query plan.

Project Overview

Experiment Setup

- Access Hadoop with Hive and PostgreSQL clusters
- Install Postgres software on PostgreSQL cluster
- Extract, Transform and Load databases from Hive to Postgres

Benchmarking

- Execute fifteen TPC-H benchmarking queries five times each on each dataset
- Analyze and compare results from Hive to Postgres

Postgres Optimizer & Query Tuning

- Review query plan of example query

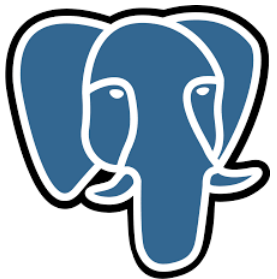
Section 1: Background Information

- Hadoop with Hive versus PostgreSQL
- Transaction Protocol Council Benchmark H (TPC-H)
- Correlated Sample Synopsis (CS2)
- DBeaver 5.2.1

Hadoop with Hive vs Postgres



Hive is an open source data warehouse system built on Hadoop for querying and managing large distributed datasets^[1]



PostgreSQL is a powerful, open source object relation database system that uses and extends the SQL language combined with many features that safely store and scale the most complicated data workloads ^[2]

[1] <https://db-engines.com/en/system/Hive%3BPostgreSQL> [2] <https://www.postgresql.org/about/>

Hadoop with Hive vs Postgres

	Hive	PostgreSQL
Primary Database Model	Relational DBMS	Relational DBMS
Website	hive.apache.org	www.postgresql.org
Developer	Apache Software Foundation	PostgreSQL Global Development Group
Implementation Language	Java	C
Server Operating Systems	All OS with a Java VM	FreeBSD, HP-UX, Linux, NetBSD, OpenBSD, OS X, Solaris, Unix, Windows
MapReduce	Yes	No
Foreign Keys	No	Yes

TPC-H Benchmark

The Transaction Processing Council Benchmark™H (TPC-H) is a decision support benchmark. It consists of a suite of business oriented ad-hoc queries and concurrent data modifications. The queries and the data populating the database have been chosen to have broad industry-wide relevance. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions.

- Fifteen of the TPC-H queries used
- 1 GB TPC-H dataset used (tpch1g)

Correlated Sample Synopsis (CS2)

Correlated Sampling aims to gather a sample of joinable sample tuples from joinable relations in the database to form a synopsis. Consists of correlated sample tuples that retain join relationships with less storage. A statistical technique, called reverse sample, and design a powerful estimator, called reverse estimator, was used to fully utilize correlated sample tuples for query estimation.

- s_tpch1g dataset produced from the 1 GB TPC-H dataset using CS2

DBeaver



DBeaver is a free, open source, graphical database management tool for database developers and administrators. Can use to create and manage databases across a wide range of database management systems including Hadoop, PostgreSQL, MySQL, MariaDB, SQLite, Oracle, DB2, SQL Server, Teradata and more. It's a multi platform tool that works on Windows, Linux, Mac and Solaris.

- One of limited few DBMS that allows you to see all different types of databases in one tool
- Can do ETL to and from different types of databases
- Graphical User Interface makes it easy to see list of databases and their objects, edit data, etc.

Section 2: Experiment Setup

- Access Hadoop with Hive and PostgreSQL clusters
- Install Postgres software on PostgreSQL cluster
- Extract, Transform and Load databases from Hive to Postgres

Access Hive with Hadoop

```
tmwells — tmwells@hdp2-m2:~ — ssh tmwells@10.70.0.179 — 80x24

# authority, or in excess of their authority, are subject #
# to having all of their activities on this system #
# monitored and recorded by system personnel. In the #
# course of monitoring individuals improperly using this #
# system, or in the course of system maintenance, the #
# activities of authorized users may also be monitored. #
# Anyone using this system expressly consents to such #
# monitoring and is advised that if such monitoring #
# reveals possible evidence of criminal activity, system #
# personnel may provide the evidence of such monitoring #
# to law enforcement officials. #
#####

tmwells@10.70.0.179's password: ]
Last login: Wed Oct 10 17:26:15 2018 from 10.8.3.235
[tmwells@hdp2-m2 ~]$ lsb_release -a ]
LSB Version: :core-4.1-amd64:core-4.1-noarch:cxx-4.1-amd64:cxx-4.1-noarch:des
ktop-4.1-amd64:desktop-4.1-noarch:languages-4.1-amd64:languages-4.1-noarch:print
ing-4.1-amd64:printing-4.1-noarch
Distributor ID: CentOS
Description: CentOS Linux release 7.3.1611 (Core)
Release: 7.3.1611
Codename: Core
[tmwells@hdp2-m2 ~]$
```

Access PostgreSQL Cluster

tmwells — tmwells@csis-psql-1:~ — ssh tmwells@10.70.0.141 — 80×24

```
Last login: Wed Oct 10 17:26:02 on ttys000
[ts18709:~ tmwells$ ssh tmwells@10.70.0.141
[tmwells@10.70.0.141's password:
Last login: Fri Oct  5 12:05:04 2018 from 10.8.60.179
[[tmwells@csis-psql-1 ~]$ cat /etc/os-release
NAME="CentOS Linux"
VERSION="7 (Core)"
ID="centos"
ID_LIKE="rhel fedora"
VERSION_ID="7"
PRETTY_NAME="CentOS Linux 7 (Core)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:centos:centos:7"
HOME_URL="https://www.centos.org/"
BUG_REPORT_URL="https://bugs.centos.org/"

CENTOS_MANTISBT_PROJECT="CentOS-7"
CENTOS_MANTISBT_PROJECT_VERSION="7"
REDHAT_SUPPORT_PRODUCT="centos"
REDHAT_SUPPORT_PRODUCT_VERSION="7"

[tmwells@csis-psql-1 ~]$
```

Postgres Software Installation

CentOS default repository contains Postgres packages, so installed easily using yum package system.

```
$ sudo yum install postgresql-server postgresql-contrib
```

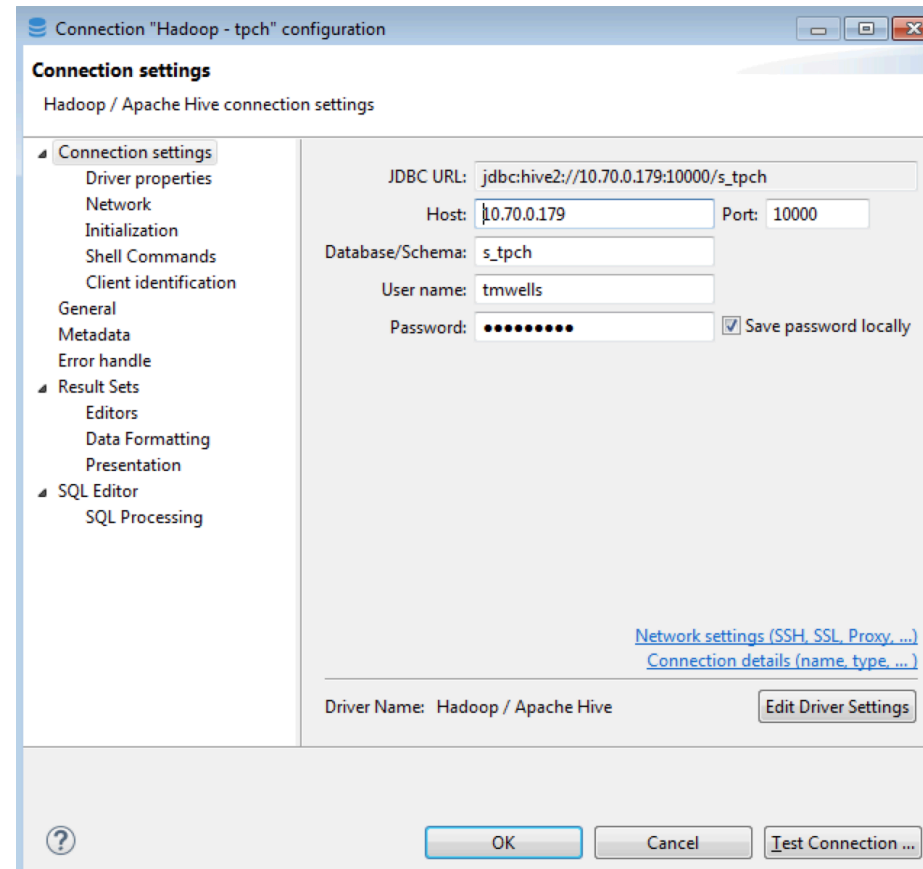
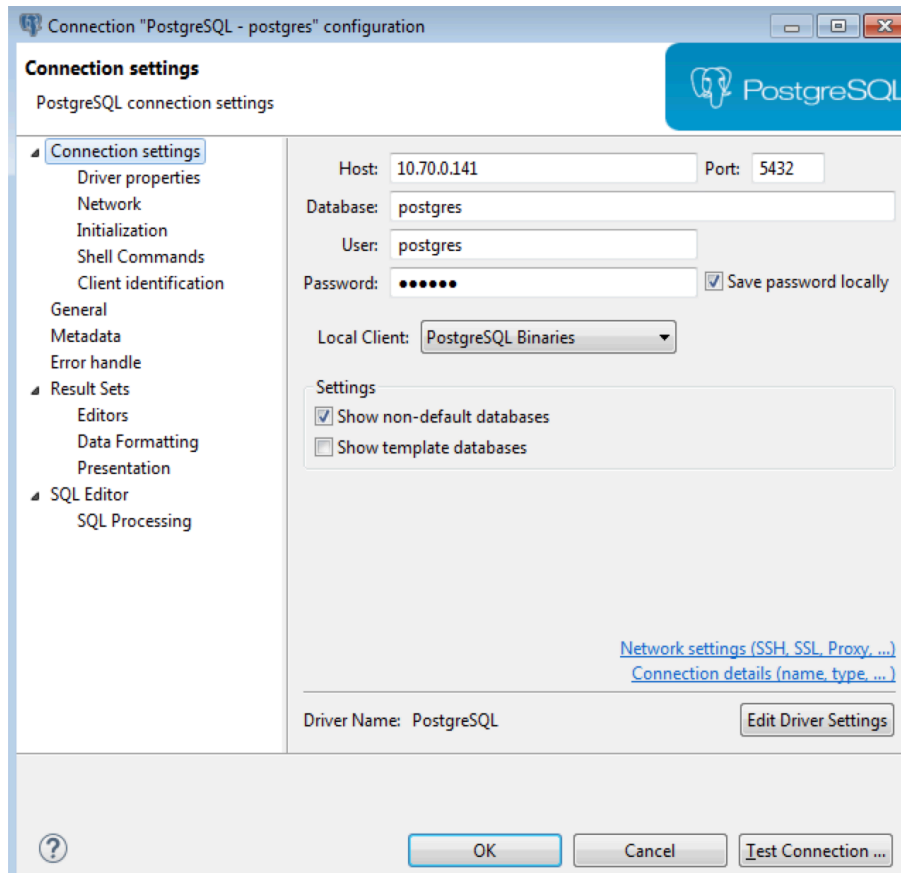
Created new PostgreSQL database cluster

```
$ sudo postgresql-setup initdb
```

Start and enabled PostgreSQL

```
$ sudo systemctl start postgresql  
$ sudo systemctl enable postgresql
```

DBeaver Installation & Connections



DBeaver User Interface

The screenshot displays the DBeaver 5.2.1 interface with a PostgreSQL database connection. The left sidebar shows the Database Navigator with a tree view of the database schema, including tables like `lineitem`, `partsupp`, `orders`, and `customer`. The main SQL Editor window contains a multi-statement query. The Results panel at the bottom shows the execution of the first query, returning a single row with a count of 6,000,003.

SQL Editor Content:

```
select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey;
select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and ps_availqty > 9000;
select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and ps_supplycost < 100;
select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and l_quantity >= 20;
select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and l_quantity >= 40000;

select count (*) from lineitem,orders where l_orderkey = o_orderkey;
select count (*) from lineitem,orders where l_orderkey = o_orderkey and o_totalprice >= 40000;
select count (*) from lineitem,orders where l_orderkey = o_orderkey and l_quantity < 20;
select count (*) from lineitem,orders where l_orderkey = o_orderkey and l_discount = 0.04;
select count (*) from lineitem,orders where l_orderkey = o_orderkey and l_shipmode = 'AIR';

select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey;
select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and c_acctbal > 500;
select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and c_mktsegment = 'AUTOMOBILE';
select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and l_returnflag = 'A';
select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and l_returnflag = 'N' and c_mktsegment = 'HOUSEHOLD';
```

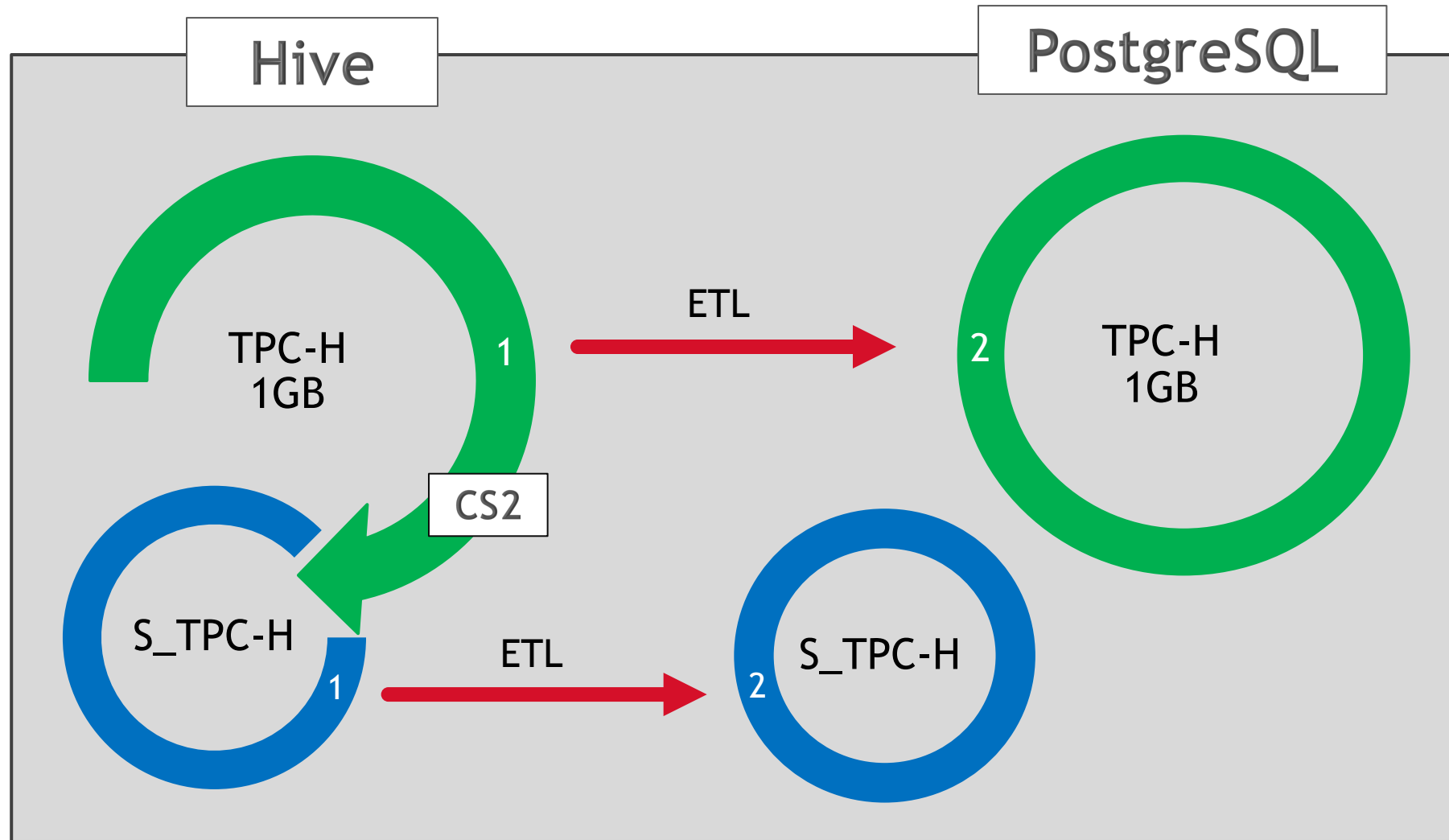
Results Panel:

Value
123 count 6,000,003

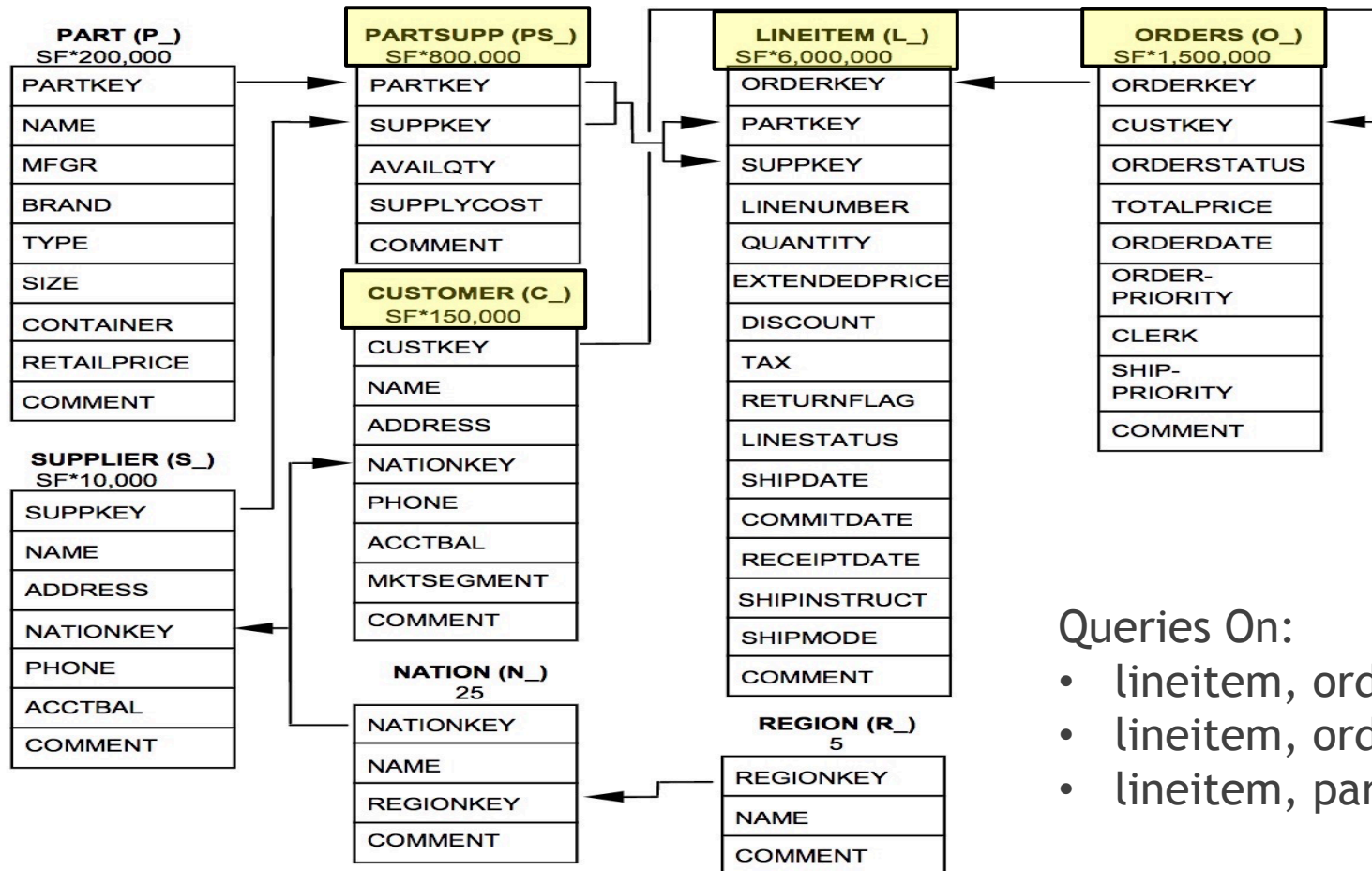
Query Manager:

Time	Type	Text	Duration (ms)	Rows	Result
Nov-24 14:...	SQL / User	select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey	13,510	1	Success

Databases from Hive to Postgres



TPC-H Schema



Queries On:

- lineitem, orders
- lineitem, orders, customer
- lineitem, partsupp

Section 3: Benchmarking

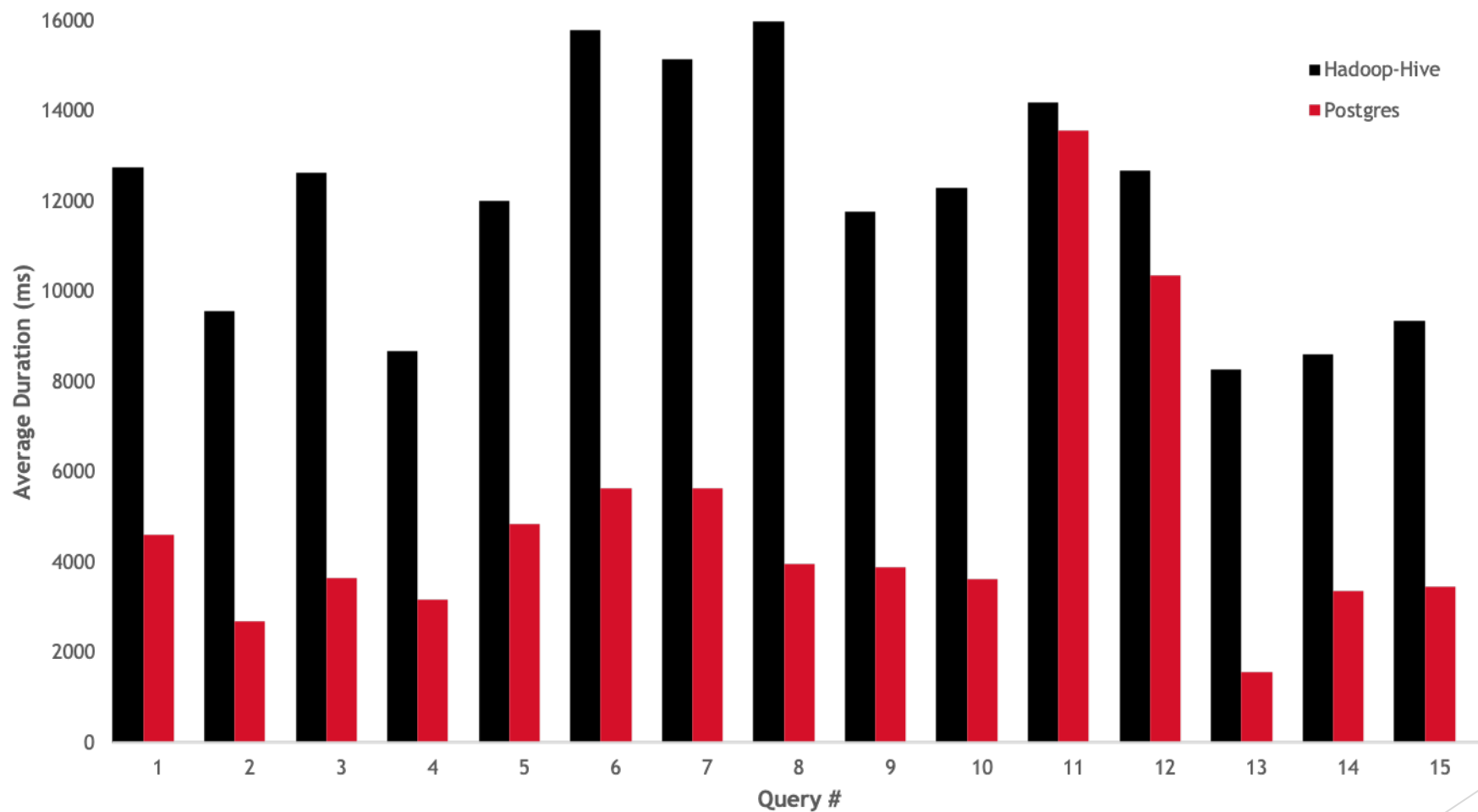
- Executed fifteen TPC-H benchmarking queries five times each on each dataset
- Analyzed and compared results from Hive to Postgres

Query Results TPC-H 1GB

	Text	Data Source	Count Value	Time 1	Time 2	Time 3	Time 4	Time 5	Avg Time	Relative Speedup (%)
1	select count (*) from lineitem,orders where l_orderkey = o_orderkey	Hadoop - tpch	6000003	10,780	11,185	10,515	11,064	20,189	12746.6	276%
		PostgreSQL - postgres	6000003	4,622	4,602	4,602	4,618	4,618	4612.4	
2	select count (*) from lineitem,orders where l_orderkey = o_orderkey and l_discount = 0.04	Hadoop - tpch	545515	9,141	9,547	9,438	9,594	10,156	9575.2	354%
		PostgreSQL - postgres	545515	2,714	2,731	2,698	2,683	2,683	2701.8	
3	select count (*) from lineitem,orders where l_orderkey = o_orderkey and l_quantity < 20	Hadoop - tpch	2280109	10,311	11,122	10,829	19,344	11,562	12633.6	345%
		PostgreSQL - postgres	2280109	3,682	3,666	3,652	3,635	3,666	3660.2	
4	select count (*) from lineitem,orders where l_orderkey = o_orderkey and l_shipmode = 'AIR'	Hadoop - tpch	858891	8,892	8,738	8,877	8,377	8,534	8683.6	275%
		PostgreSQL - postgres	858891	3,042	3,043	3,560	3,186	2,979	3162	
5	select count (*) from lineitem,orders where l_orderkey = o_orderkey and o_totalprice >= 40000	Hadoop - tpch	5798870	11,653	11,971	12,184	12,159	12,033	12000	247%
		PostgreSQL - postgres	5798870	4,760	4,789	4,758	4,696	5,257	4852	
6	select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey	Hadoop - tpch	6000003	14,680	16,661	15,473	16,146	15,990	15790	280%
		PostgreSQL - postgres	6000003	5,741	5,585	5,600	5,602	5,710	5647.6	
7	select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and c_acctbal > 500	Hadoop - tpch	5187412	15,657	15,491	14,319	15,663	14,586	15143.2	268%
		PostgreSQL - postgres	5187412	5,382	6,755	5,369	5,367	5,335	5641.6	
8	select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and c_mktsegment = 'AUTOMOBILE'	Hadoop - tpch	1190865	14,243	23,115	13,011	17,084	12,498	15990.2	404%
		PostgreSQL - postgres	1190865	3,947	3,963	3,978	3,948	3,963	3959.8	
9	select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and l_returnflag = 'A'	Hadoop - tpch	1476912	11,077	12,137	12,153	11,193	12,308	11773.6	303%
		PostgreSQL - postgres	1476912	3,837	3,838	4,118	3,824	3,822	3887.8	
10	select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and l_returnflag = 'N' and c_mktsegment = 'HOUSEHOLD'	Hadoop - tpch	609190	12,963	12,142	12,855	11,811	11,731	12300.4	338%
		PostgreSQL - postgres	609190	3,650	3,606	3,635	3,666	3,619	3635.2	
11	select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey	Hadoop - tpch	6000003	12,698	13,402	12,058	13,587	19,206	14190.2	105%
		PostgreSQL - postgres	6000003	13,354	13,354	14,104	13,463	13,526	13560.2	
12	select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and l_quantity >= 20	Hadoop - tpch	3719894	13,049	12,558	12,293	12,339	13,182	12684.2	123%
		PostgreSQL - postgres	3719894	10,390	10,538	10,296	10,249	10,265	10347.6	
13	select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and l_quantity >= 40000	Hadoop - tpch	5798870	7,550	8,268	9,656	8,143	7,722	8267.8	527%
		PostgreSQL - postgres	5798870	1,528	1,529	1,514	1,513	1,763	1569.4	
14	select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and ps_availqty > 9000	Hadoop - tpch	603036	8,662	8,549	8,705	8,522	8,564	8600.4	256%
		PostgreSQL - postgres	603036	3,369	3,338	3,385	3,354	3,322	3353.6	
15	select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and ps_supplycost < 100	Hadoop - tpch	592794	9,157	10,298	8,863	8,908	9,469	9339	271%
		PostgreSQL - postgres	592794	3,416	3,432	3,569	3,403	3,416	3447.2	

Relative Speedup % = (Postgres avg time / Hadoop avg time) x 100%

Query Results TPC-H 1GB

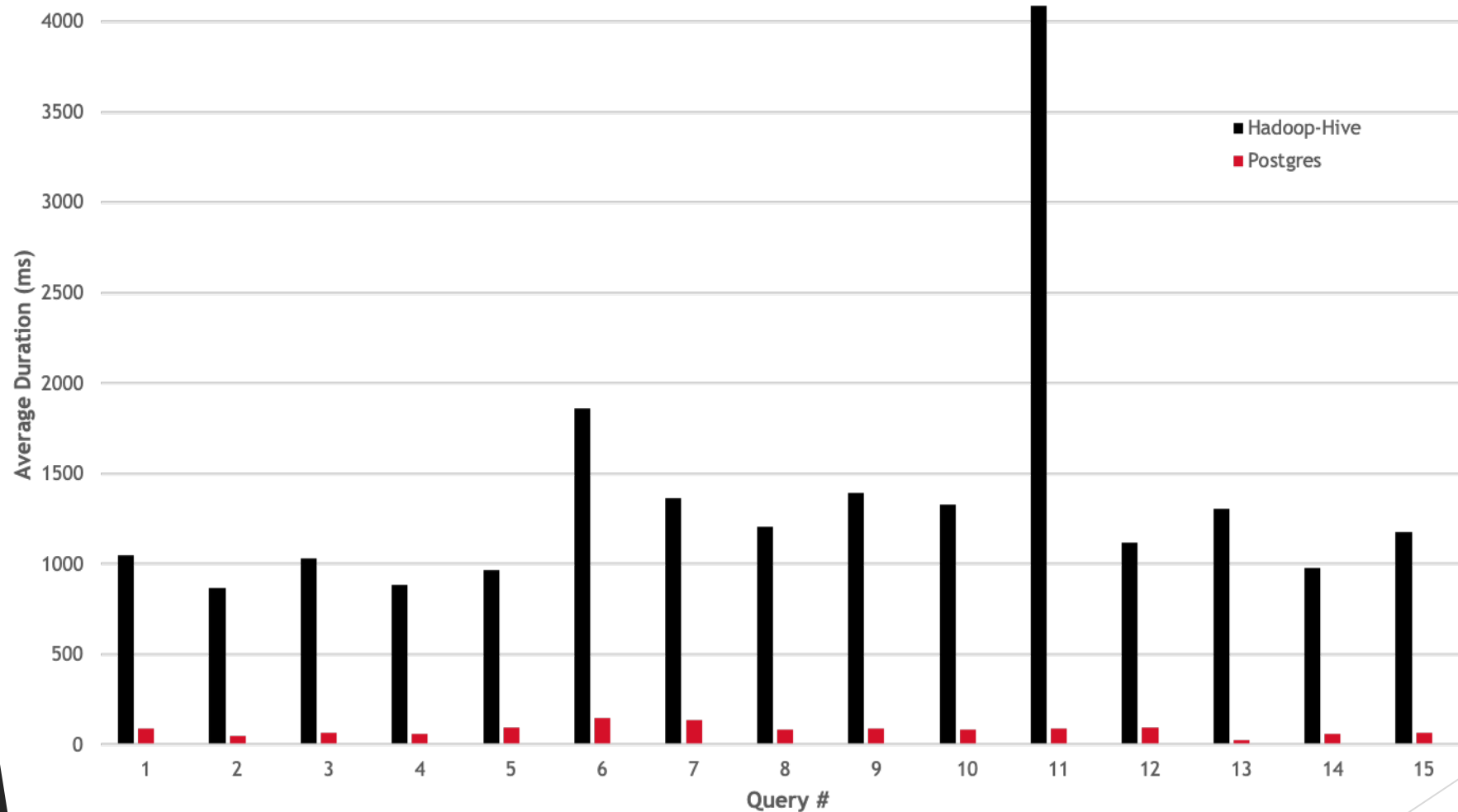


Query Results Sample TPC-H

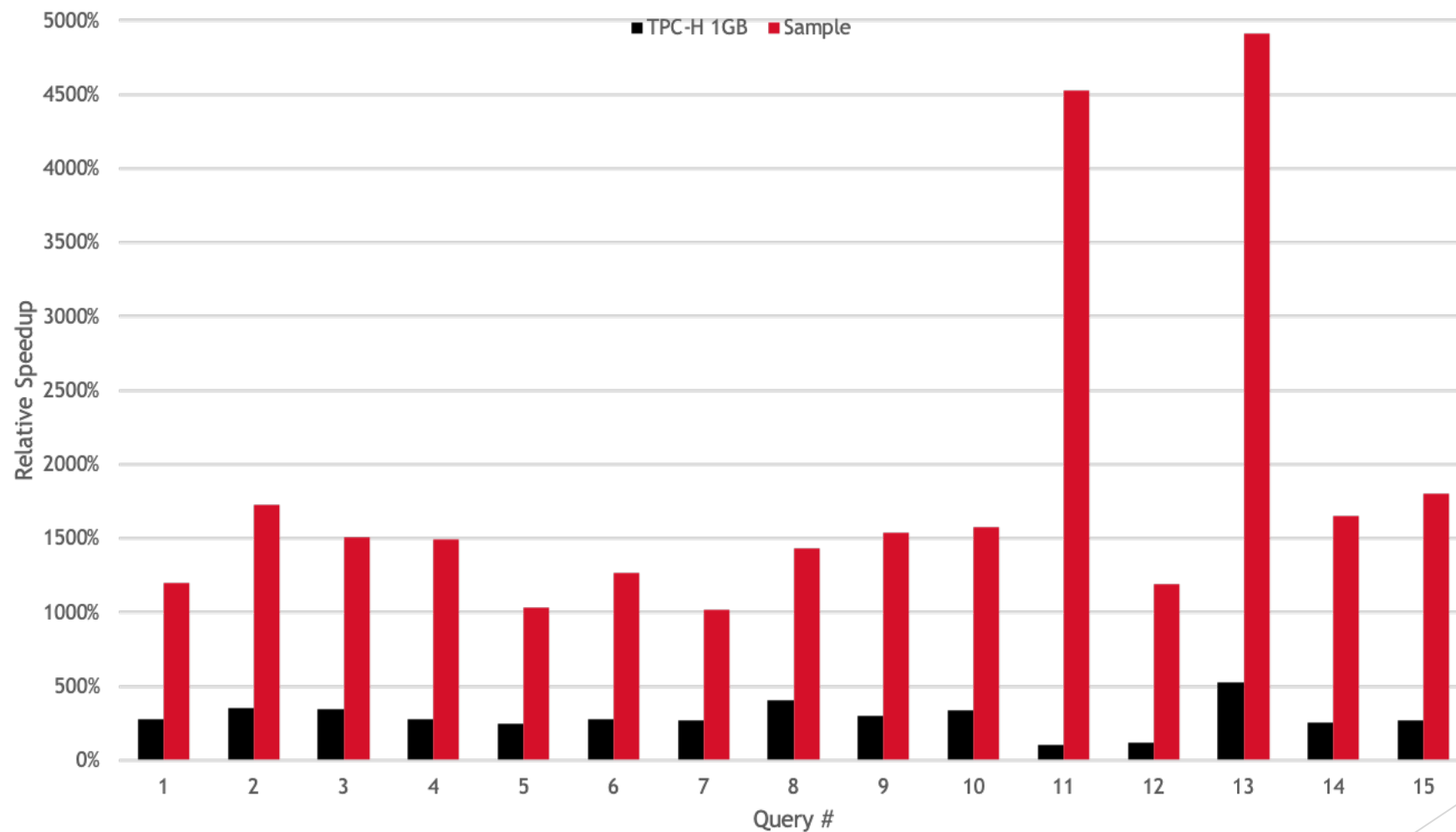
	Text	Data Source	Count Value	Time 1	Time 2	Time 3	Time 4	Time 5	Avg Time	Relative Speedup (%)
1	select count (*) from lineitem,orders where l_orderkey = o_orderkey	Hadoop - tpch	60381	1,513	1,045	858	874	951	1048.2	1197%
		PostgreSQL - postgres	60381	94	94	94	78	78	87.6	
2	select count (*) from lineitem,orders where l_orderkey = o_orderkey and l_discount = 0.04	Hadoop - tpch	5545	749	811	998	904	873	867	1727%
		PostgreSQL - postgres	5545	47	47	47	63	47	50.2	
3	select count (*) from lineitem,orders where l_orderkey = o_orderkey and l_quantity < 20	Hadoop - tpch	22963	1,669	890	905	827	873	1032.8	1506%
		PostgreSQL - postgres	22963	78	63	62	62	78	68.6	
4	select count (*) from lineitem,orders where l_orderkey = o_orderkey and l_shipmode = 'AIR'	Hadoop - tpch	8697	889	670	640	983	1,232	882.8	1491%
		PostgreSQL - postgres	8697	63	78	46	62	47	59.2	
5	select count (*) from lineitem,orders where l_orderkey = o_orderkey and o_totalprice >= 40000	Hadoop - tpch	58361	920	889	967	982	1,061	963.8	1030%
		PostgreSQL - postgres	58361	94	93	93	110	78	93.6	
6	select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey	Hadoop - tpch	60381	2,153	1,935	2,106	1,591	1,513	1859.6	1270%
		PostgreSQL - postgres	60381	156	140	156	140	140	146.4	
7	select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and c_acctbal > 500	Hadoop - tpch	52176	1,326	1,451	1,201	1,388	1,450	1363.2	1016%
		PostgreSQL - postgres	52176	140	140	141	125	125	134.2	
8	select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and c_mktsegment = 'AUTOMOBILE'	Hadoop - tpch	11881	1,201	1,233	1,217	1,076	1,295	1204.4	1434%
		PostgreSQL - postgres	11881	78	78	109	62	93	84	
9	select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and l_returnflag = 'A'	Hadoop - tpch	15057	2,246	1,108	1,186	1,232	1,202	1394.8	1540%
		PostgreSQL - postgres	15057	94	78	78	125	78	90.6	
10	select count (*) from lineitem,orders,customer where l_orderkey = o_orderkey and o_custkey = c_custkey and l_returnflag = 'N' and c_mktsegment = 'HOUSEHOLD'	Hadoop - tpch	6110	1,466	1,248	1,311	1,264	1,342	1326.2	1575%
		PostgreSQL - postgres	6110	78	78	78	109	78	84.2	
11	select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey	Hadoop - tpch	60381	2,013	1,419	2,386	7,519	7,113	4090	4524%
		PostgreSQL - postgres	60381	109	93	94	78	78	90.4	
12	select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and l_quantity >= 20	Hadoop - tpch	37418	1,076	1,014	1,311	983	1,201	1117	1191%
		PostgreSQL - postgres	37418	78	141	94	78	78	93.8	
13	select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and l_quantity >= 40000	Hadoop - tpch	0	858	811	1,591	1,529	1,747	1307.2	4914%
		PostgreSQL - postgres	0	31	31	24	16	31	26.6	
14	select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and ps_availqty > 9000	Hadoop - tpch	6178	873	936	1,076	889	1,124	979.6	1649%
		PostgreSQL - postgres	6178	47	63	62	62	63	59.4	
15	select count (*) from lineitem,partsupp where l_partkey = ps_partkey and l_suppkey = ps_suppkey and ps_supplycost < 100	Hadoop - tpch	6076	952	936	1,591	1,186	1,217	1176.4	1804%
		PostgreSQL - postgres	6076	62	62	62	62	78	65.2	

Relative Speedup % = (Postgres avg time / Hadoop avg time) x 100%

Query Results Sample TPC-H



Relative Speedup



Relative Speedup % = (Postgres avg time / Hadoop avg time) x 100%

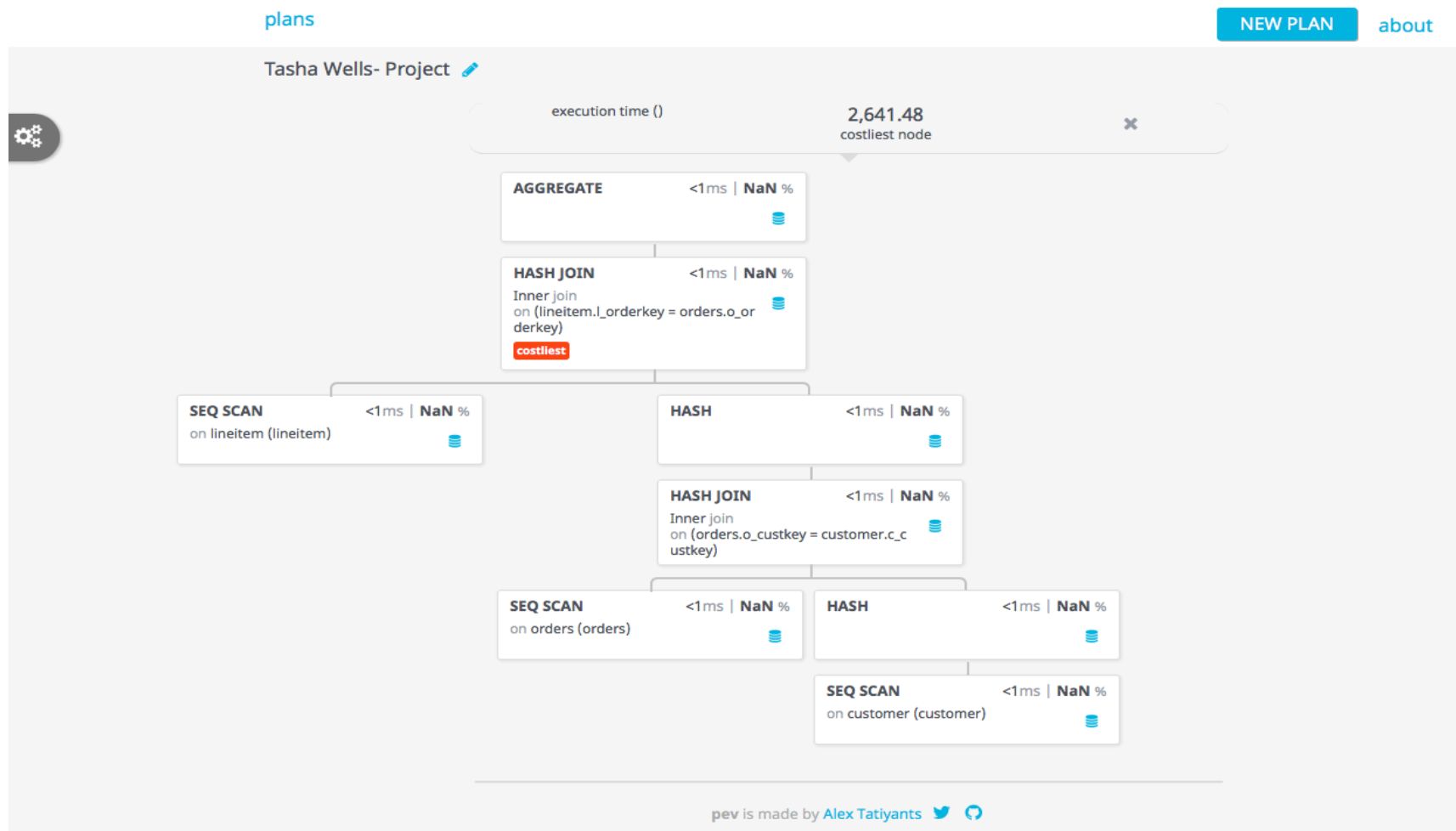
Highlights of Results

- PostgreSQL took less time than Hive for every query processed
- Relative Speedup from PostgreSQL compared to Hive decreased with the larger 1 GB dataset
- Query 11 stands out as a costly query for Hive in both datasets and PostgreSQL in the larger dataset
Relative speedup was significant in the sample dataset
- Postgres processed query 13 the fastest in both datasets and it has the greatest Relative Speedup

Section 4: PostgreSQL Query Plan

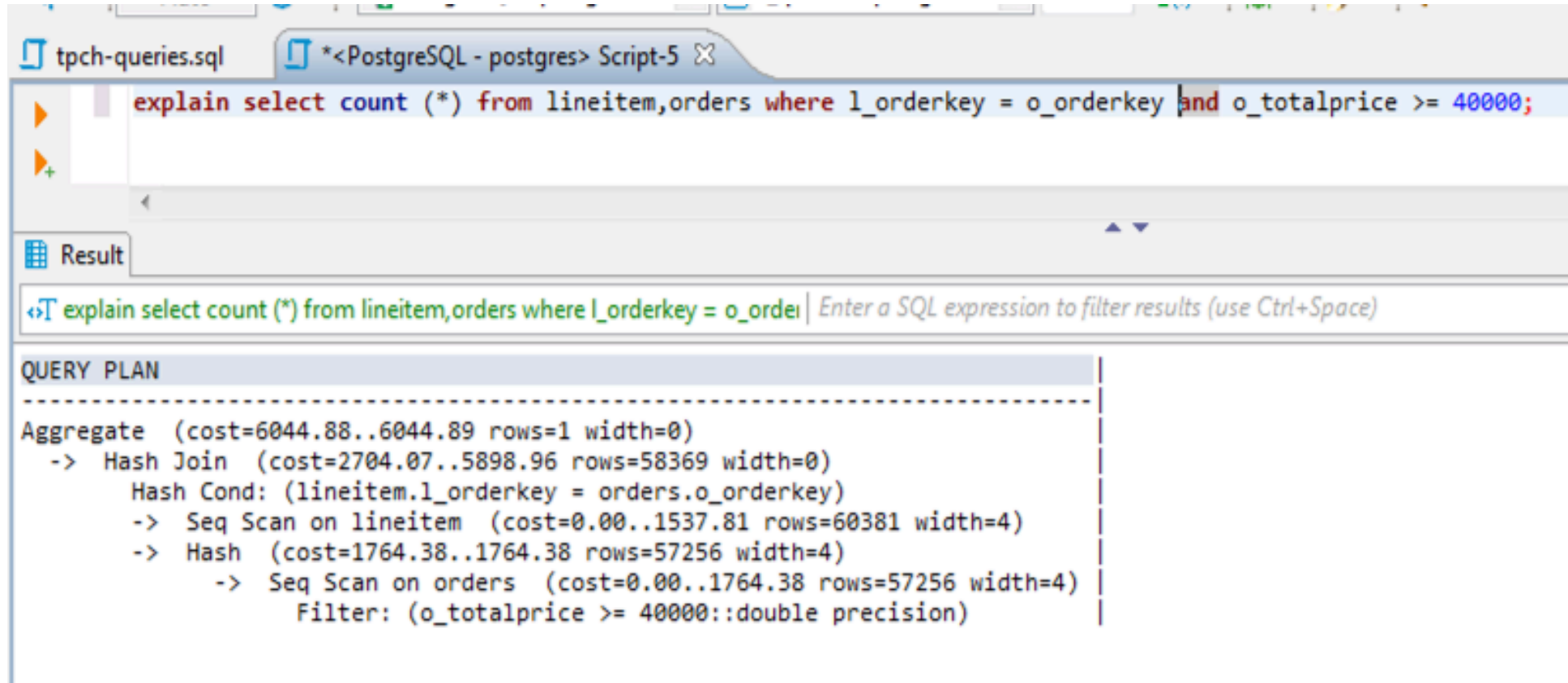
- Review query plan of example query
- PostgreSQL optimizer overview

Visualization of Query Plan



http://tatiyants.com/pev/#/plans/plan_1541530153517

Visualization of Query Plan



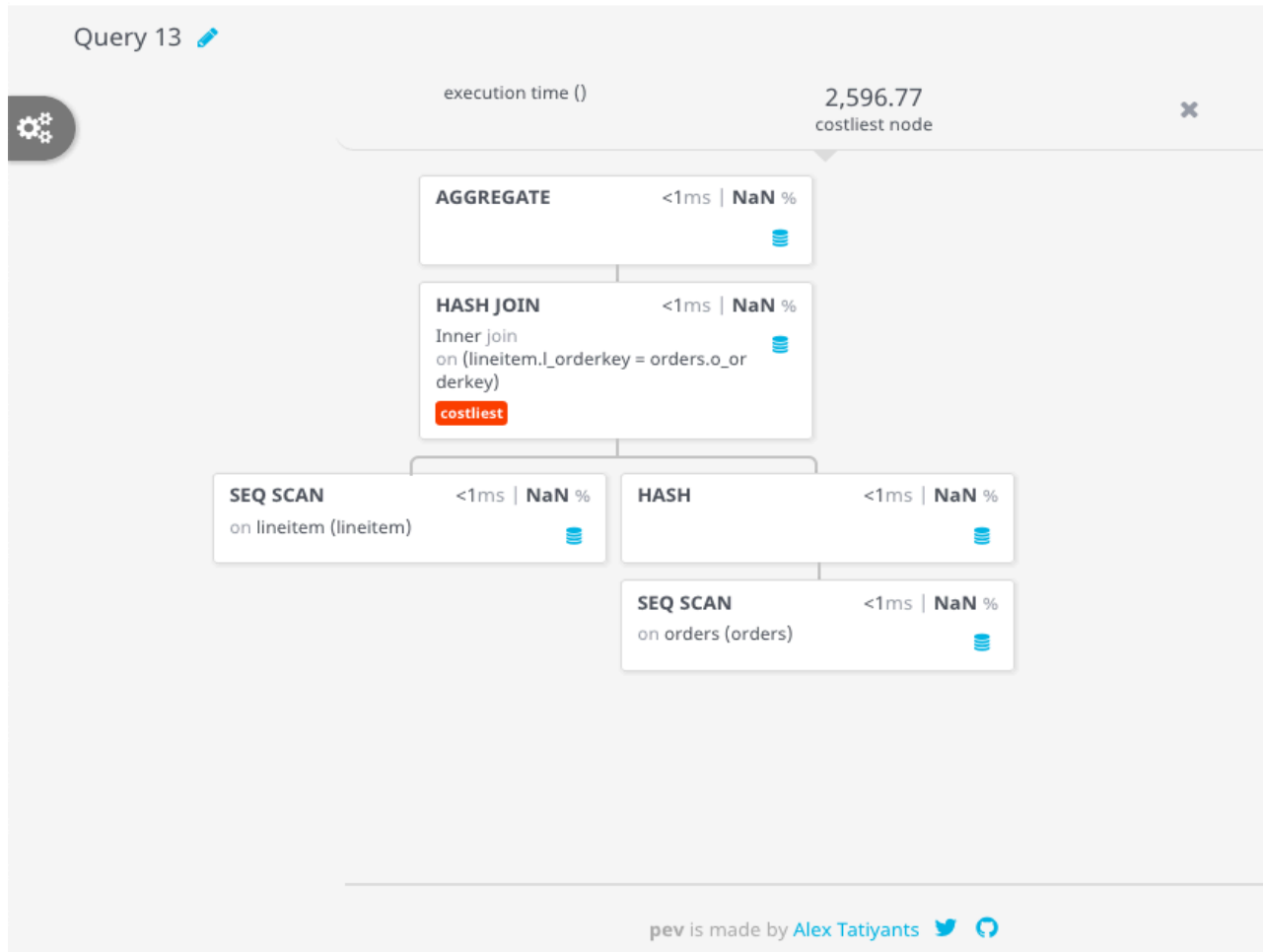
The screenshot shows a PostgreSQL query editor with two tabs: 'tpch-queries.sql' and '*<PostgreSQL - postgres> Script-5'. The active tab contains the following SQL query:

```
explain select count (*) from lineitem,orders where l_orderkey = o_orderkey and o_totalprice >= 40000;
```

Below the query editor, the 'Result' tab is selected, displaying the query plan for the executed statement. The query plan is as follows:

```
QUERY PLAN
-----
Aggregate (cost=6044.88..6044.89 rows=1 width=0)
-> Hash Join (cost=2704.07..5898.96 rows=58369 width=0)
    Hash Cond: (lineitem.l_orderkey = orders.o_orderkey)
    -> Seq Scan on lineitem (cost=0.00..1537.81 rows=60381 width=4)
    -> Hash (cost=1764.38..1764.38 rows=57256 width=4)
        -> Seq Scan on orders (cost=0.00..1764.38 rows=57256 width=4)
            Filter: (o_totalprice >= 40000::double precision)
```

Visualization of Query Plan



http://tatiyants.com/pev/#/plans/plan_1543501806264

HTML	TEXT	STATS					Settings	Add optimization
#	exclusive	inclusive	rows x	rows	loops	node		
1.	0.158	55.012	↑ 11.4	71	1	→ Sort (cost=146.63..148.65 rows=808 width=138) (actual time=55.009..55.012 rows=71 loops=1) Sort Key: n.nspname, p.proname, (pg_get_function_arguments(p.oid)) Sort Method: quicksort Memory: 43kB "Plan": { "Node Type": "Aggregate", "Strategy": "Plain", "Startup Cost": 6044.88, "Total Cost": 6044.89, "Plan Rows": 1, "Plan Width": 0, "Plans": [
2.	1.378	54.854	↑ 11.4	71	1	→ Hash Join (cost=1.14..107.61 rows=808 width=138) (actual time=42.495..54.854 rows=71 loops=1) Hash Cond: (p.pronamespace = n.oid) "Node Type": "Hash Join", "Parent Relationship": "Outer", "Join Type": "Inner", "Startup Cost": 2704.07, "Total Cost": 5898.96, "Plan Rows": 58369, "Plan Width": 0, "Hash Cond": "(lineitem.l_orderkey = orders.o_orderkey)", "Plans": [],		
3.	53.465	53.465	↓ 3.0	2,402	1	→ Seq Scan on pg_proc p (cost=0.00..89.30 rows=808 width=78) (actual time=0.052..53.465 rows=2,402 loops=1) Filter: pg_function_is_visible(oid)		
4.	0.004	0.011	↑ 1.0	4	1	→ Hash (cost=1.09..1.09 rows=4 width=68) (actual time=0.011..0.011 rows=4 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 1kB "Node Type": "Seq Scan", "Parent Relationship": "Outer",		

<https://explain.depesz.com/s/gwQA>

Postgres Query Handling

Joins are the most difficult relational operators to process and optimize. Possible query plans grow exponentially with the number of joins in the query. Further optimization effort is caused by the support of a variety of join methods (e.g. nested loop, hash join, etc.) to process. ***Genetic Algorithm*** implemented to solve join ordering problem.

Genetic Algorithm Summary:

- 1) Randomly initialize populations p
- 2) Determine fitness of population
- 3) Until convergence repeat:
 - a) Select parents from population
 - b) Crossover and generate new population
 - c) Perform mutation on new population
 - d) Calculate fitness for new population

Section 5: Conclusion

- Experiment & Overall
- Question & Answer

Conclusion

Experiment

PostgreSQL takes less time to execute the TPC-H queries than Hive in both the 1GB database and sample database. For both datasets, PostgreSQL would be the better system to use. For the dataset and queries used, PostgreSQL is the system I recommend. Once you get into larger dataset, this may not be the case. Further testing could be done. PostgreSQL uses a query optimizer that works very well and is difficult to do a better job manually tuning queries

Overall Project

Gained hands on experience:

Installing PostgreSQL software and database cluster

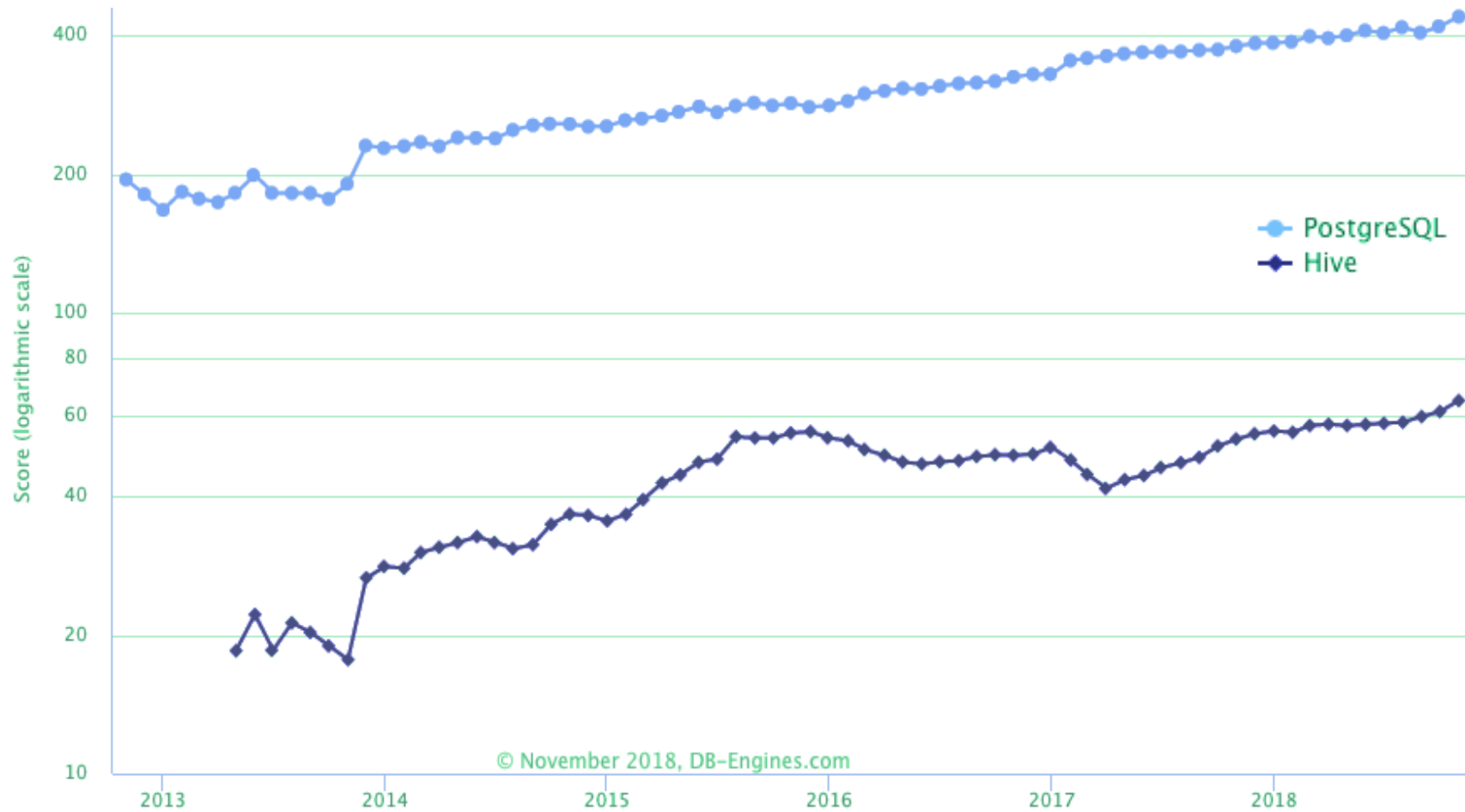
Extracting, Transforming and Loading data from Hive to PostgreSQL

Performing benchmarking between Hive and PostgreSQL

Using EXPLAIN and visualization tools to review PostgreSQL query plan

.

Popularity Ranking of Hive vs. PostgreSQL



https://db-engines.com/en/ranking_trend/system/Hive%3BPostgreSQL

Acknowledgements

David Wilson
Swetha Yelamanchi

Dr. Feng Yu

Dr. Alina Lazar
CSIS Department

Thank you!