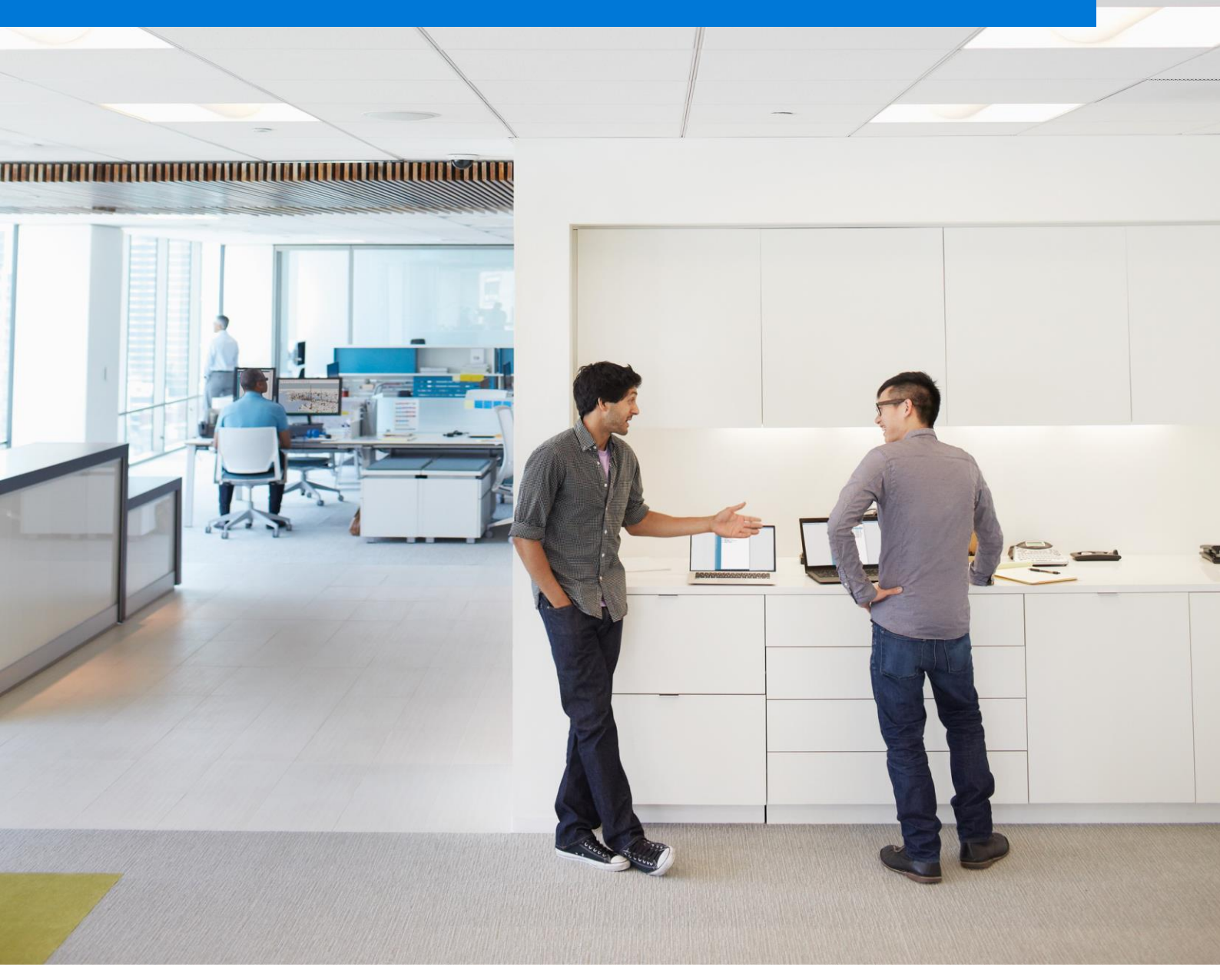
Microsoft

TPC-H を用いた Azure SQL Data Warehouse の性能検証



目次

1 はじめに

本書は Azure SQL Data Warehouse の性能や、スケールアウトによる性能への影響、あるいは、Azure 以外のパブリッククラウドに実装されているデータウェアハウスサービスとの性能比較に関する情報をまとめたものである。

従来、オンプレミスで構築されてきたデータウェアハウスに関しては、高価なアプライアンス製品導入などにより莫大な初期投資を必要としてきた。一方で近年ではクラウド技術の発展に伴い、Azure SQL Data Warehouse に代表されるようなクラウド型のデータウェアハウスサービス等も出現もしてきた。Azure SQL Data Warehouse をはじめとしたクラウド型のデータウェアハウスサービスの多くは、高額なライセンスや大量のハードウェアなどの初期投資を必要とせず、従量課金制となっており、導入に要する期間もオンプレミス型の製品に比べると非常に短いことが特徴である。

データウェアハウスの選定で、性能の観点は非常に重要なポイントである。Azure SQL Data Warehouse と他のクラウド型のデータウェアハウスサービスと比較した時、性能はどうであろうか。本書では TPC-H のデータモデル、および分析クエリを利用し、Azure SQL Data Warehouse をはじめとするクラウド型データウェアハウスの性能に関して検証を行った結果をまとめる。

2 検証概要

2.1 目的

検証対象となるクラウド型データウェアハウス上に TPC-H のデータモデルに準拠したテーブルを作成し、データを投入し、クエリ実行環境を構築する。

TPC-H で定義されている 22 本の分析クエリを実行し、クエリ性能を計測する。

クエリ性能に関しては、いくつかのパターンでデータベースのスケールアウトを行うことで、クエリ性能がどの程度高速化するのか、性能検証を実施する。

その他、クラウド型のデータウェアハウスサービスに関しては、Amazon WEB Services の提供する Amazon Redshift（以下、AWS Redshift）を用いる。AWS Redshift では、2017 年 3 月現在、DS1 および、DS2 ノードタイプと、DC1 ノードタイプが存在している。DS1 および、DS2 ノードタイプはより大容量ワークロードに対して適しており、DC1 ノードタイプはパフォーマンスを多用するワークロードに適している。DS2 は DS1 よりも使用可能なメモリおよび CPU が増加されており、DS1 と比べて高いパフォーマンスを得る事が出来る。本検証では AWS Redshift は DS2 を利用して検証を行うこととする。

Azure SQL Data Warehouse はいくつかのパターンでスケールアウトし検証を行うが、AWS Redshift に関しては Azure SQL Data Warehouse の検証パターンと比較した時、時間当たりのデータベースの課金額がおよそ同じになるような環境にてクエリ実行環境を構築し、クエリ性能の計測を行う。これにより Azure SQL Data Warehouse と AWS Redshift の性能に関する比較検証を行う。この時、AWS Redshift の時間当たりの課金額に関しては円に換算したもの（$1=110 円）で計算を行う。

2.2 検証シナリオ

TPC-H のデータ量に関しては、全ての検証パターンにおいて一律とし、Scale Factor を 1000（約 1TB）にてデータを作成する。TPC-H の分析用クエリの実行には Apache JMeter を利用し計測を行う。

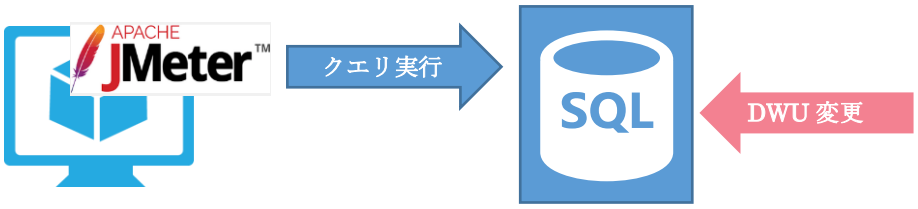


図 2 – 1 ：性能評価環境（Azure）

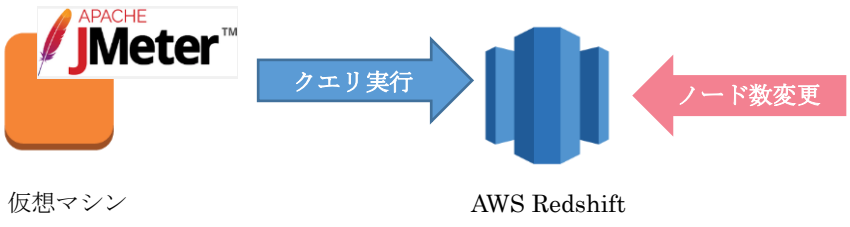


図 2 – 2 ：性能評価環境（Redshift）

分析用クエリは Apache JMeter から、22 本のクエリが重複なくシリアルに流れるように設定する。また、22 本で 1 セットとする。ウォーミングアップとして、1 セット実行した後、同様の分析用クエリを 3 セット実行し、この 3 セットの実行による、各クエリの平均値を計測値として採用することとする。

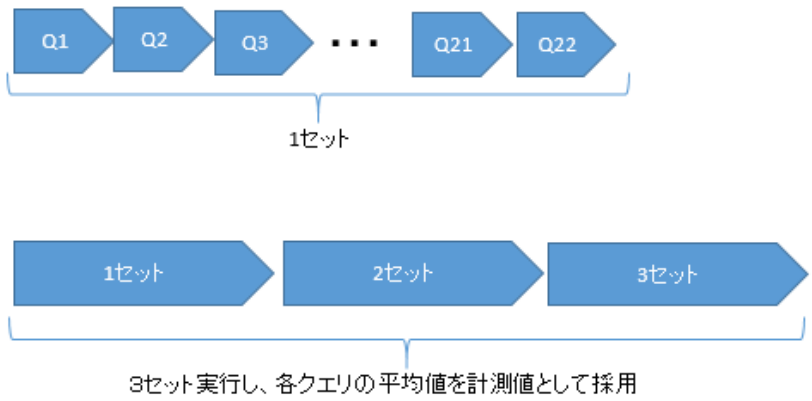


図２－３：Apache JMeter による分析クエリ 22 本の実行方法

Azure SQL Data Warehouse に関しては以下の DWU のパターンにて検証を行う。

・400DWU
・1000DWU
・2000DWU
・3000DWU
・60000DWU

また、Azure SQL Data Warehouse に関しては、一度上記パターンを検証後、チューニングを行い、再度計測を行うこととする。実施するチューニングに関しては本書に後述する。

AWS Redshift に関しては、Azure SQL Data Warehouse の計測パターンの従量課金のコストより、以下の環境にて、計測を実施する。

・ds2.large(4vCPU 31GB)-5node　400DWU 相当
・ds2.large(4vCPU 31GB)-14node　1000DWU 相当
・ds2.8xlarge(36vCPU 244GB)-3node　2000DWU 相当
・ds2.8xlarge(36vCPU 244GB)-5node　3000DWU 相当
・ds2.8xlarge(32vCPU 244GB)-10node　6000DWU 相当

2.3 結果サマリ

計測結果は以下の通り。

DWU	Azure SQL-DWH (未チューニング)	Azure SQL-DWH （チューニング）	AWS Redshift
400	1.00	0.65	2.58
1000	0.39	0.21	0.29
2000	0.19	0.10	0.19
3000	0.13	0.07	0.10
6000	0.09	0.05	0.06

※ 結果は Azure SQL Data Warehouse の 400DWU での実行結果（未チューニング）を 1 とした時の相対値にて記載。

3 検証内容詳細

3.1 環境と事前準備

3.1.1 検証環境

検証環境は各環境ともに、対象となるデータベースと、SQL を実行する Windows Server の 2 階層で構成する。また、対象となるデータベースと Windows Server に関しては同一のリージョン等に配置し、ロケーションが離れることによる処理の遅延が発生しないように可能な限り配慮された構成とする。

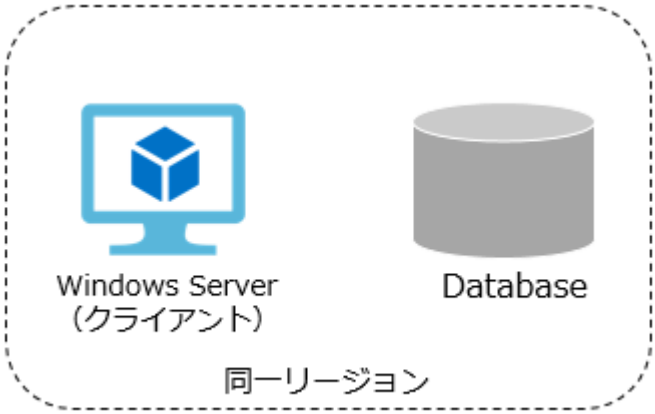


図３－１－１：検証環境イメージ図

3.1.1.1 検証環境（Azure SQL Data Warehouse）

Azure SQL Data Warehouse の検証環境は Windows 仮想マシン(以下、Windows VM)と Azure SQL Data Warehouse で構成する。各スペックに関しては以下の通り。

・Windows VM

項目	スペック
オペレーションシステム	Windows Server 2012 R2
サイズ	Standard DS2 v2
CPU スペック	2 コア
メモリ	7GB
リージョン	東日本

・Azure SQL Data Warehouse

項目	スペック
スペック	400DWU 1000DWU 2000DWU 3000DWU 6000DWU ※ 各 DWU にて検証を行う。
照合順序	SQL_Latin1_General_CP1_CI_AS
リージョン	東日本

Windows VM と Azure SQL Data Warehouse に関しては同一リージョン、同一リソースグループ内に配置する構成とする。また、TPC-H のデータはすべて英語となり日本語は無い為、照合順序は SQL Data Warehouse のデフォルト値（SQL_Latin1_General_CP1_CI_AS）を使用している。

Windows VM に導入する Apache JMeter の情報は以下の通り。

・Apache JMeter

項目	スペック
バージョン	apache-jmeter-3.0

Apache JMeter から SQL Data Warehouse へ接続する際のドライバーには SQL Server の JDBC ドライバーを利用する。

・JDBC ドライバー

項目	スペック
バージョン	Microsoft JDBC Driver 4.0 for SQL Server

3.1.1.2 検証環境（AWS Redshift）

AWS Redshift の検証環境は AWS Elastic Compute Cloud (以下、EC2)と AWS Redshift で構成する。各スペックに関しては以下の通り。

・EC2

項目	スペック
オペレーションシステム	Windows Server 2012 R2
インスタンスタイプ	C4.xlarge
CPU スペック	4vCPU
メモリ	7.5GB
リージョン	日本

・AWS Redshift

項目	スペック
スペック	ds2.large(4vCPU 31GB)-5node ds2.large(4vCPU 31GB)-14node ds2.8xlarge(36vCPU 244GB)-3node ds2.8xlarge(36vCPU 244GB)-5node ds2.8xlarge(32vCPU 244GB)-10node ※ インスタンスタイプ（CPU スペック メモリ）-ノード数 ※ 各クラスター構成にて検証を行う
リージョン	日本

パラメータ名	値
datestyle	ISO, MDY
enable_user_activity_logging	false
extra_float_digits	0
max_cursor_result_set_size	default
query_group	default
require_ssl	false
search_path	$user, public
statement_timeout	0

EC2 と AWS Redshift は同一リージョン内、同一 Virtual Private Cloud 内、同一セグメント内に配置する。

EC2 に導入する Apache JMeter の情報は以下の通り。

・Apache JMeter

項目	スペック
バージョン	apache-jmeter-3.0

Apache JMeter から AWS Redshift へ接続する際のドライバーには PostgreSQL JDBC ドライバー4.0 互換の AWS Redshift カスタム JDBC ドライバーを利用する。

・AWS Redshift カスタム JDBC ドライバー

項目	スペック
バージョン	AWS Redshift カスタム JDBC ドライバー 4.0

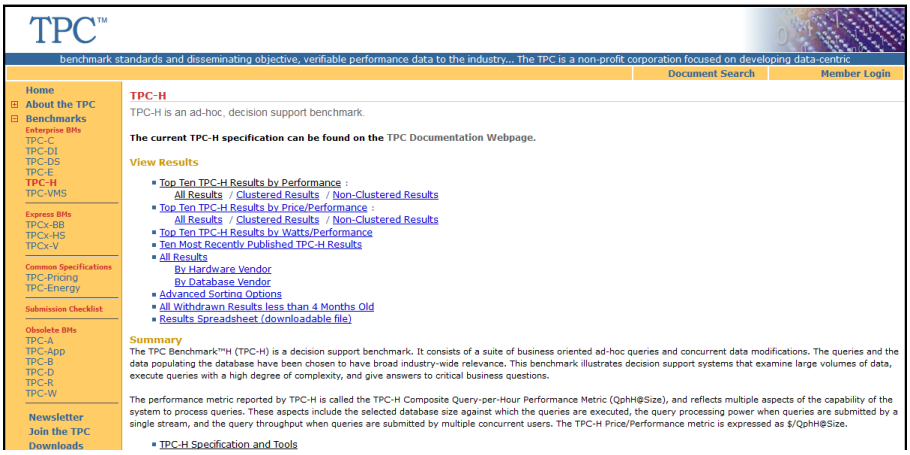
3.1.2 検証用データ、データモデリング

本検証では TPC-H データモデリング、および、22 本の分析用クエリを利用し検証を行う。

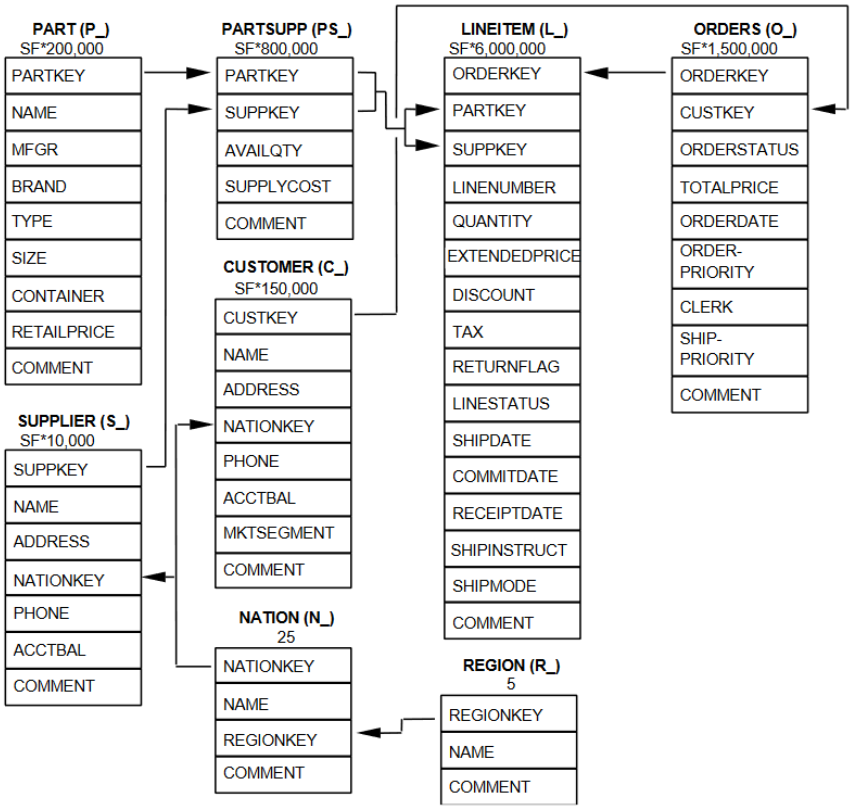
TPC-H は RDBMS ベンチマーク仕様の一つで、売上分析データウェアハウスのデータモデリングを使い、意思決定支援システムの性能を測定、計測するもので、クエリは 22 種類定義されている。

TPC-H の詳細に関しては、下記 URL を参照の事。

http://www.tpc.org/tpch/



図表３－１－２－１：TPC-H



図表３－１－２－２：TPC-H データモデル図

各テーブルに求められるテーブルレイアウトに関しては、下記「TPC BENCHMARK H(Decision Support)Standard Specification」の「1.3 DATATYPE DEFINITIONS」を参照の事。
http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.1.pdf

3.1.2.1 検証用のテーブル

Azure SQL Data Warehouse および AWS Redshift 上で作成する検証用のテーブルは以下の通り作成する。

・PART テーブル

カラム名	データタイプ（SQL DWH）	データタイプ（Redshift）
p_partkey	bigint - HASH KEY	bigint - DISTKEY
p_partkey	varchar(55)	varchar(55)
p_mfgr	char(25)	char(25)
p_brand	char(10)	char(10)
p_type	varchar(25)	varchar(25)
p_size	integer	integer
p_container	char(10)	char(10)
p_retailprice	float	float
p_comment	varchar(23)	varchar(23)
備考	クラスター化カラムストアインデックスおよび HASH KEY によるデータ分散	DISTKEY によるデータ分散。

・SUPPLIER テーブル

カラム名	データタイプ（SQL DWH）	データタイプ（Redshift）
s_suppkey	bigint	bigint
s_name	char(25)	char(25)
s_address	varchar(40)	varchar(40)
s_nationkey	integer	integer
s_phone	char(15)	char(15)
s_acctbal	float	float
s_comment	varchar(101)	varchar(101)
備考	クラスター化カラムストアインデックスおよび ROUND ROBIN による分散	EVEN によるデータ分散。

・PARTSUPP テーブル

カラム名	データタイプ（SQL DWH）	データタイプ（Redshift）
ps_partkey	bigint – HASH KEY	bigint - DISTKEY
ps_suppkey	bigint	bigint
ps_availqty	integer	integer
ps_supplycost	float	float
ps_comment	varchar(199)	varchar(199)
備考	クラスター化カラムストアインデックスおよび HASH KEY によるデータ分散	DISTKEY によるデータ分散。

・CUSTOMER テーブル

カラム名	データタイプ（SQL DWH）	データタイプ（Redshift）
c_custkey	bigint – HASH KEY	bigint - DISTKEY
c_name	varchar(25)	varchar(25)
c_address	varchar(40)	varchar(40)
c_nationkey	integer	integer
c_phone	char(15)	char(15)
c_acctbal	float	float
c_mktsegment	char(10)	char(10)
c_comment	varchar(117)	varchar(117)
備考	クラスター化カラムストアインデックスおよび HASH KEY によるデータ分散	DISTKEY によるデータ分散。

・ORDERS テーブル

カラム名	データタイプ（SQL DWH）	データタイプ（Redshift）
o_orderkey	bigint – HASH KEY	bigint - DISTKEY
o_custkey	bigint	bigint
o_orderstatus	char(1)	char(1)
o_totalprice	float	float
o_orderdate	date	date
o_orderpriority	char(15)	char(15)
o_clerk	char(15)	char(15)
o_shippriority	integer	integer
o_comment	varchar(79)	varchar(79)
備考	クラスター化カラムストアインデックスおよび HASH KEY によるデータ分散	DISTKEY によるデータ分散。

・LINEITEM テーブル

カラム名	データタイプ（SQL DWH）	データタイプ（Redshift）
l_orderkey	bigint – HASH KEY	bigintt - DISTKEY
l_partkey	bigint	bigint
l_suppkey	bigint	bigint
l_linenumber	bigint	bigint
l_quantity	float	float
l_extendedprice	float	float
l_discount	float	float
l_tax	float	float
l_returnflag	char(1)	char(1)
l_linestatus	char(1)	char(1)
l_shipdate	date	date
l_commitdate	date	date

l_receiptdate	date	date
l_shipinstruct	char(25)	char(25)
l_shipmode	char(10)	char(10)
l_comment	varchar(44)	varchar(44)
備考	クラスター化カラムストアインデックスおよび HASH KEY によるデータ分散	DISTKEY によるデータ分散。

・NATION テーブル

カラム名	データタイプ（SQL DWH）	データタイプ（Redshift）
n_nationkey	integer	integer
n_name	char(25)	char(25)
n_regionkey	integer	integer
n_comment	varchar(152)	varchar(152)
備考	クラスター化カラムストアインデックスおよび ROUND ROBIN による分散	EVEN によるデータ分散。

・REGION テーブル

カラム名	データタイプ（SQL DWH）	データタイプ（Redshift）
r_regionkey	integer	integer
r_name	char(25)	char(25)
r_comment	varchar(152)	varchar(152)
備考	クラスター化カラムストアインデックスおよび ROUND ROBIN による分散	EVEN によるデータ分散。

データ投入後、各テーブルの件数は以下の通り。

テーブル名	データ件数
customer	150,000,000 件
lineitem	5,999,989,709 件
nation	25 件
oeders	1,500,000,000 件
part	200,000,000 件
partsupp	800,000,000 件
region	5 件
supplier	10,000,000 件

また、データ投入後は統計情報の取得を行う。

Azure SQL Data Warehouse では統計情報に関して、以下のカラムにて列統計の取得を行う。

テーブル名	カラム名	備考
part	p_partkey	SAMPLE 100 PERCENT
part	p_name	SAMPLE 100 PERCENT
Part	p_brand	SAMPLE 100 PERCENT

part	p_type	SAMPLE 100 PERCENT
part	p_size	SAMPLE 100 PERCENT
part	p_container	SAMPLE 100 PERCENT
supplier	s_suppkey	SAMPLE 100 PERCENT
supplier	s_name	SAMPLE 100 PERCENT
supplier	s_nationkey	SAMPLE 100 PERCENT
supplier	s_acctbal	SAMPLE 100 PERCENT
partsupp	ps_partkey	SAMPLE 100 PERCENT
partsupp	ps_suppkey	SAMPLE 100 PERCENT
partsupp	ps_availqty	SAMPLE 100 PERCENT
partsupp	ps_supplycost	SAMPLE 100 PERCENT
customer	c_custkey	SAMPLE 100 PERCENT
customer	c_name	SAMPLE 100 PERCENT
customer	c_address	SAMPLE 100 PERCENT
customer	c_phone	SAMPLE 100 PERCENT
customer	c_acctbal	SAMPLE 100 PERCENT
customer	c_mktsegment	SAMPLE 100 PERCENT
customer	c_comment	SAMPLE 100 PERCENT
orders	o_orderkey	SAMPLE 100 PERCENT
orders	o_custkey	SAMPLE 100 PERCENT
orders	o_orderstatus	SAMPLE 100 PERCENT
orders	o_totalprice	SAMPLE 100 PERCENT
orders	o_orderpriority	SAMPLE 100 PERCENT
orders	o_shippriority	SAMPLE 100 PERCENT
orders	o_comment	SAMPLE 100 PERCENT
lineitem	l_partkey	SAMPLE 100 PERCENT
lineitem	l_quantity	SAMPLE 100 PERCENT
lineitem	l_discount	SAMPLE 100 PERCENT
lineitem	l_returnflag	SAMPLE 100 PERCENT
lineitem	l_shipmode	SAMPLE 100 PERCENT
lineitem	l_receiptdate	SAMPLE 100 PERCENT
lineitem	l_linestatus	SAMPLE 100 PERCENT
lineitem	l_commitdate	SAMPLE 100 PERCENT
lineitem	l_shipinstruct	SAMPLE 100 PERCENT
nation	n_nationkey	SAMPLE 100 PERCENT
nation	n_name	SAMPLE 100 PERCENT
nation	n_regionkey	SAMPLE 100 PERCENT
nation	n_comment	SAMPLE 100 PERCENT
region	r_regionkey	SAMPLE 100 PERCENT
region	r_name	SAMPLE 100 PERCENT

AWS Redshift は COPY コマンド（STATSUPDATE=ON）にてデータを投入した場合、自動で統計情報は取得されるので、明示的な統計情報の取得は行わない。

3.1.3 検証ツール（Apache JMeter）

SQL を実行するツールは Apache JMeter とする。

Apache JMeter を起動し、「テスト計画」の中に「図３－１－３－１：Apache JMETER の設定」の通りに作成する。

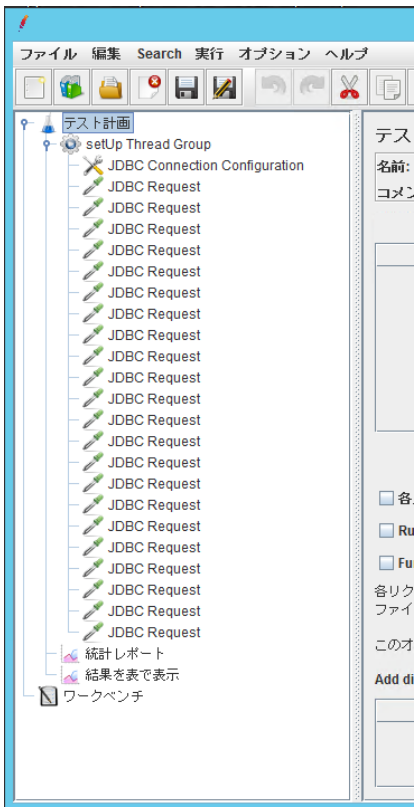


図３－１－３－１：Apache JMeter の設定

3.1.3.1 Apache JMeter の設定

Apache JMeter の設定は以下の通り設定する。また、以下以外の設定値に関してはすべてデフォルト値のまま使用する。

① スレッドグループ

名前：スレッドグループ

・スレッドプロパティ

設定項目	値
スレッド数	1
Ramp-Up 期間（秒）	1
ループ回数	3※

※ ただし、ウォーミングアップの時は 1 に指定する。

② JDBC Connection Configure

名前：JDBC Connection Configure

・Variable Name Bound to Pool

設定項目	値
Variable Name	q1

・Connection Pool Configuration

設定項目	値
Max Number of Connection	100
Max Waite(ms)	10000
Time Betwaeen Eviction Runs(ms)	60000
Auto Commit	True
Transaction Isolation	DEFAULT

・Connection Validation by Pool

設定項目	値
Test While Idle	True
Soft Min Evictable Idle Time(ms)	1000
Validation Query	Select 1

・Database Connection Configuration

設定項目	値（SQL DWH）	値（Redshift）
Database URL	jdbc:sqlserver://<サーバー名>;databaseName=<データベース名>;integratedSecurity=FALSE;	Jdbc:redshift://<endpoint 名>:<ポート番号>/<データベース名>
JDBC Driver class	com.microsoft.sqlserver.jdbc.SQLServerDriver	com.amazon.redshift.jdbc4.Driver
Username	<ユーザ名>	<ユーザ名>
Password	＜パスワード＞	＜パスワード＞

【注意】 接続ユーザに関して

Apache JMeter から Azure SQL Data Warehouse へ接続を行うユーザに関しては、リソースクラスを「xlargerc」にて設定する。この時、デフォルトで作成されるユーザのリソースクラスは変更を行えない為、新規でユーザを作成し、リソースクラスの変更を行う。

③ JDBC Request

22 個作成されている JDBC Request に関しては、TPC-H の分析用クエリ 22 本をそれぞれ記載する。ここでは、Azure SQL Data Warehouse で実行する、クエリ No.1、クエリ No.2 に関して、例として記載するが、実際は 22 本のクエリそれぞれ作成が必要である。

22 本の各クエリに関しては本書「5.1.1 Azure SQL Data Warehouse の TPC-H クエリ」、「5.1.2 AWS Redshift の TPC-H クエリ」に記載する。

名前	1
Variable Name	q1
Query Type	Select Statement
Query	

```
/* TPC_H Query 1 - Pricing Summary Report */
SELECT TOP 1 L_RETURNFLAG, L_LINESTATUS, SUM(L_QUANTITY) AS SUM_QTY,
 SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS SUM_DISC_PRICE,
 SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE, AVG(L_QUANTITY) AS AVG_QTY,
 AVG(L_EXTENDEDPRICE) AS AVG_PRICE, AVG(L_DISCOUNT) AS AVG_DISC, COUNT_BIG(*) AS COUNT_ORDER
FROM LINEITEM
WHERE L_SHIPDATE <= dateadd(dd, -90, cast('1998-12-01' as datetime))
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG,L_LINESTATUS
```

名前	2
Variable Name	q1
Query Type	Select Statement
Query	

```
/* TPC_H Query 2 - Minimum Cost Supplier */
SELECT TOP 100 S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS, S_PHONE, S_COMMENT
FROM PART, SUPPLIER, PARTSUPP, NATION, REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15 AND
P_TYPE LIKE '%%BRASS' AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND
R_NAME = 'EUROPE' AND
PS_SUPPLYCOST = (SELECT MIN(PS_SUPPLYCOST) FROM PARTSUPP, SUPPLIER, NATION, REGION
 WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY
 AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND R_NAME = 'EUROPE')
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY
```

クエリの詳細に関しては、「前項 : 3.1.2 検証用データ、データモデリング」で記載した、「TPC BENCHMARK H(Decision Support)Standard Specification」を参照の事。

3.2 検証方法

3.2.1 検証方法

Azure SQL Data Warehouse、AWS Redshift それぞれの環境にて Apache JMeter より「前項：3.1.3 検証ツール（Apache JMeter）」で作成したテスト計画を実行する。

実行は以下の通り行う

① ウォーミングアップとして、テスト計画を実行

スレッドグループのループ回数を 1 に設定し、テスト計画の実行を行う。

この実行に関してはウォーミングアップのため、計測値は破棄する。

② 本番計測として、テスト計画を実行

計測値の取得の為に、スレッドグループのループ回数を 3 に設定し、テスト計画の実行を行う。この実行で得られた値を計測値として採用する。

計測値として②で得られた値を採用する。②実行時には、22 本のクエリがシリアルに実行され、それが 3 回繰り返される。このため、各クエリ 3 回ずつ実行することになる。計測値として、3 回のクエリ実行の平均時間を用いる。

また、検証を行う各データベースに関しては、以下の検証パターンでそれぞれ同様の計測を行う。

	Azure SQL Data Warehouse	AWS Redshift
比較 1	400DWU	ds2.xlarge(4core 31GB)-5node
比較 2	1000DWU	ds2.xlarge(4Core 31GB)-14node
比較 3	2000DWU	ds2.8xlarge (36Core 244GB)-3node
比較 4	3000DWU	ds2.8xlarge (36Core 244GB)-5node
比較 5	6000DWU	ds2.8xlarge (36Core 244GB)-10node

3.2.2 計測値の取得方法

計測値は Apache JMeter の統計レポートの Average より、各クエリの値を取得する。この値は各クエリの 3 回実行時の平均実行時間を示すものである。

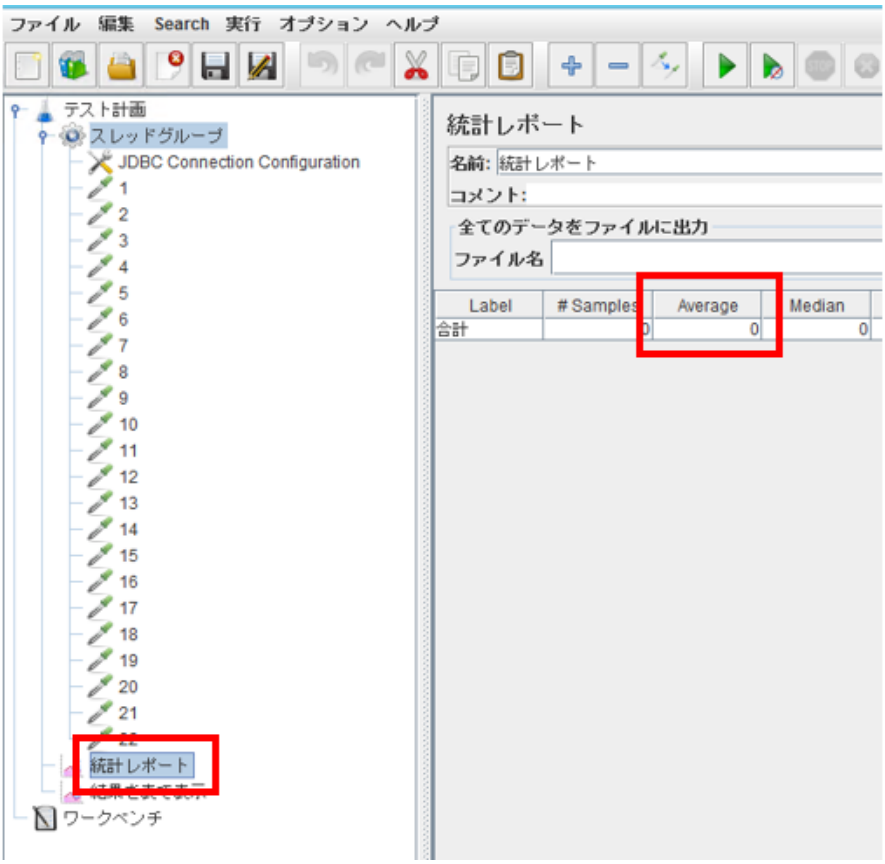


図３－１－５：Apache JMeter の統計レポート

3.3 チューニング

Azure SQL Data Warehouse ではパーティショニングや非クラスター化インデックスを作成する事が可能である。

3.3.1 パーティション化

データ件数の多い以下 2 テーブルに対してはパーティション化を実施する。この時、1 パーティションあたり 100 万件を上回るようにパーティション化する事が一般的な推奨値である。

・lineitem テーブル

l_shipdate 列をパーティション化列としてパーティション化を実施。DDL は以下の通り。

```
CREATE TABLE [lineitem]
(
        [l_orderkey] [bigint] NOT NULL,
        [l_partkey] [bigint] NOT NULL,
        [l_suppkey] [bigint] NOT NULL,
        [l_linenumber] [bigint] NOT NULL,
        [l_quantity] [float] NOT NULL,
        [l_extendedprice] [float] NOT NULL,
        [l_discount] [float] NOT NULL,
        [l_tax] [float] NOT NULL,
        [l_returnflag] [char](1) NOT NULL,
        [l_linestatus] [char](1) NOT NULL,
        [l_shipdate] [date] NOT NULL,
        [l_commitdate] [date] NOT NULL,
        [l_receiptdate] [date] NOT NULL,
        [l_shipinstruct] [char](25) NOT NULL,
        [l_shipmode] [char](10) NOT NULL,
        [l_comment] [varchar](44) NOT NULL
)
WITH
(
        DISTRIBUTION = HASH ( [l_orderkey] ),
    CLUSTERED COLUMNSTORE INDEX,
        PARTITION ([l_shipdate] RANGE RIGHT FOR VALUES(
      '9999',
      '1992',
      '1993',
      '1994',
      '1995',
      '1996',
      '1997',
      '1998',
      '1999',
      '2000')
    )
)
```

・orders テーブル

l_shipdate 列をパーティション化列としてパーティション化を実施。DDL は以下の通り。

```
CREATE TABLE [orders]
(
        [o_orderkey] [bigint] NOT NULL,
        [o_custkey] [bigint] NOT NULL,
        [o_orderstatus] [char](1) NOT NULL,
        [o_totalprice] [float] NOT NULL,
        [o_orderdate] [date] NOT NULL,
        [o_orderpriority] [char](15) NOT NULL,
        [o_clerk] [char](15) NOT NULL,
        [o_shippriority] [int] NOT NULL,
        [o_comment] [varchar](79) NOT NULL
)
WITH
(
        DISTRIBUTION = HASH ( [o_orderkey] ),
    CLUSTERED COLUMNSTORE INDEX,
        PARTITION ([o_orderdate] RANGE RIGHT FOR VALUES (
      '9999',
      '1992',
      '1993',
      '1994',
      '1995',
      '1996',
      '1997',
      '1998',
      '1999',
      '2000')
    )
)
```

3.3.2 HASH KEY の変更

テーブルの結合を行う際、クエリの結合条件となっている列に HASH KEY を指定する事で、クエリ実行時のデータ移動サービス（DMS）を抑制する事が可能である。この為、下記テーブルに関しては、HASH KEY の見直しを行う。

・nation テーブル

ROUND ROBIN 分散から n_regionkey を HASH KEY に指定する HASH 分散へ変更。DDL は以下の通り。

```
CREATE TABLE [nation]
(
    [n_nationkey] [integer] NOT NULL,
    [n_name] [char](25) NOT NULL,
    [n_regionkey]       [integer] NOT NULL,
    [n_comment]         [varchar](152) NOT NULL
)
WITH (
            DISTRIBUTION = HASH (n_regionkey)
            ,CLUSTERED COLUMNSTORE INDEX
)
```

3.3.3 ヒープテーブルへの変更

クラスター化カラムストア インデックスは 1 億件以上のテーブルで初めて最適な圧縮が行われる。この為、比較的件数の少ないテーブルに関してはヒープテーブルを利用する方が効率的に処理する事が可能になる。以下のテーブルに関してはクラスター化カラムストアインデックスからヒープテーブルへ変換を行う。

・supplier テーブル

クラスター化カラムストアインデックスからヒープテーブルへ変更。DDL は以下の通り。

```
CREATE TABLE [supplier]
(
            [s_suppkey] [bigint] NOT NULL,
            [s_name] [char](25) NOT NULL,
            [s_address] [varchar](40) NOT NULL,
            [s_nationkey] [int] NOT NULL,
            [s_phone] [char](15) NOT NULL,
            [s_acctbal] [float] NOT NULL,
            [s_comment] [varchar](101) NOT NULL
)
WITH
(
            DISTRIBUTION = ROUND_ROBIN,HEAP
)
```

3.3.4 クラスター化インデックスの作成

supplier テーブルに関しては、テーブルのヒープテーブル化するだけではなく、クラスター化インデックスを作成する事で、処理の高速化を行う。

・supplier テーブル

```
CREATE CLUSTERED INDEX [PK_SUPPLIER] ON [supplier] ([s_suppkey] ASC)
```

3.3.5 非クラスター化インデックスの作成

テーブルの結合項目を意識し、以下の４テーブルに関しては非クラスター化インデックスを作成する。

・customer テーブル

```
CREATE NONCLUSTERED INDEX [IX_customer_nationandcustkey] ON [customer]
(
	[c_nationkey] ASC,[c_custkey] ASC
)
```

・lineitem テーブル

```
CREATE NONCLUSTERED INDEX [IX_lineitem_part_mixkey] ON [lineitem]
(
	[l_orderkey] ASC,[l_suppkey] ASC,[l_discount] ASC,[l_extendedprice] ASC
)
CREATE NONCLUSTERED INDEX [IX_lineitem_part_mixkey2] ON [lineitem]
(
	[l_suppkey] ASC,[l_orderkey] ASC,[l_discount] ASC,[l_extendedprice] ASC
)
```

・orders テーブル

```
CREATE NONCLUSTERED INDEX [IX_orders_part_mixkey] ON [orders]
(
	[o_orderdate] ASC,[o_custkey] ASC,[o_orderkey] ASC
)
```

・supplier テーブル

```
CREATE NONCLUSTERED INDEX [IX_SUPPLIER] ON [supplier]
(
	[s_nationkey] ASC
)
CREATE NONCLUSTERED INDEX [IX_SUPPLIER_2] ON [tpch4].[supplier]
(
	[s_comment] ASC
)
```

3.3.6 統計情報の再取得

テーブル定義の変更に伴い、テーブルの再作成等を行っている為、データの投入後、統計情報の再作成を「前項：3.1.2.1 検証用のテーブル」で記載した通り実施する。

3.4 検証結果

3.4.1 実行時間

22 本のクエリを Apache JMeter より実行した時、全てのクエリの実行に要した時間は以下の結果となった。結果は Azure SQL Data Warehouse 400DWU のチューニング前の実行時間を 1 とした時の相対値で表しており、値が大きいほど実行時間が長く、値が小さいほど実行時間は短い。

実行時間に関しては、400DWU の時、Azure SQL Data Warehouse（チューニング前）は AWS Redshift（ds2.xlarge(4core 31GB)-5node）と比較し、AWS Redshift の方が 2.58 倍 実行時間がかかっている。また、400DWU の SQL Data Warehouse のチューニング前後で確認をすると、チューニング後の方が、チューニング前に比べ、0.65 倍の実行時間は短くなっている。

チューニング前の Azure SQL Data Warehouse の 2000DWU と 400DWU で比較した場合には、2000DWU は 0.19 倍 実行時間は短くなっており、実行時間は約 1/5 に短縮された。

・実行時間

	チューニング前	チューニング後	Redshift
400DWU	1.00	0.65	2.58
1000DWU	0.39	0.21	0.29
2000DWU	0.19	0.10	0.19
3000DWU	0.13	0.07	0.10
6000DWU	0.09	0.05	0.06

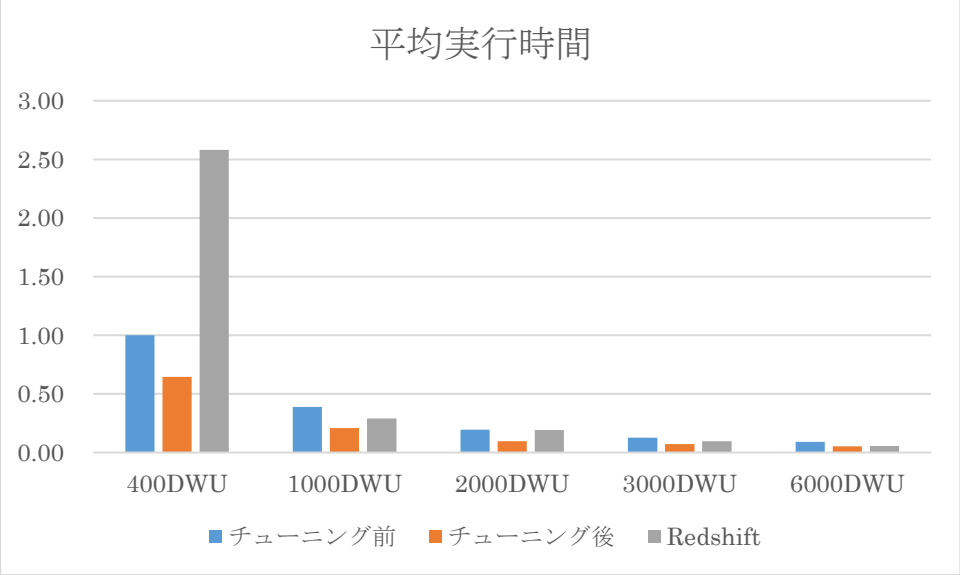


図３－４－１：シナリオごとの平均実行時間（相対値）

3.4.2 TPC-H Query per Hour Performance Metric（QphH@1000GB）

今回の検証環境における TPC-H Query per Hour Performance Metric（QphH@1000）の結果は以下の通りとなった。SQL Data Warehouse 400DWU のチューニング前の QphH@1000 を 1 とした時の相対値で表しており、値が大きいほど、パフォーマンスが良く、値が小さいほど、パフォーマンスが悪い事を表している。

QphH@1000 に関しては、400DWU の時、SQL Data Warehouse（チューニング前）は AWS Redshift（ds2.xlarge(4core 31GB)-5node）と比較した場合、AWS Redshift は 0.56 倍の Azure SQL Data Warehouse に比べてパフォーマンス劣化となった。一方で、400DWU で Azure SQL Data Warehouse のチューニング前後で比較した場合には、チューニング後ではチューニング前に比べ、1.56 倍程度パフォーマンスが向上している。

チューニング前の SQL Data Warehouse 同士で比較すると 2000DWU は 400DWU と比較し、QphH@1000 の値は 5.23 倍となり、TPC-H 上では約 5 倍以上の性能となっている。

・Query-per-Hour Performance Metric(QphH@1000GB)

	チューニング前	チューニング後	Redshift
400DWU	1.00	1.56	0.56
1000DWU	2.53	5.03	2.42
2000DWU	5.23	11.59	3.74
3000DWU	10.06	14.79	7.57
6000DWU	12.11	17.24	13.67

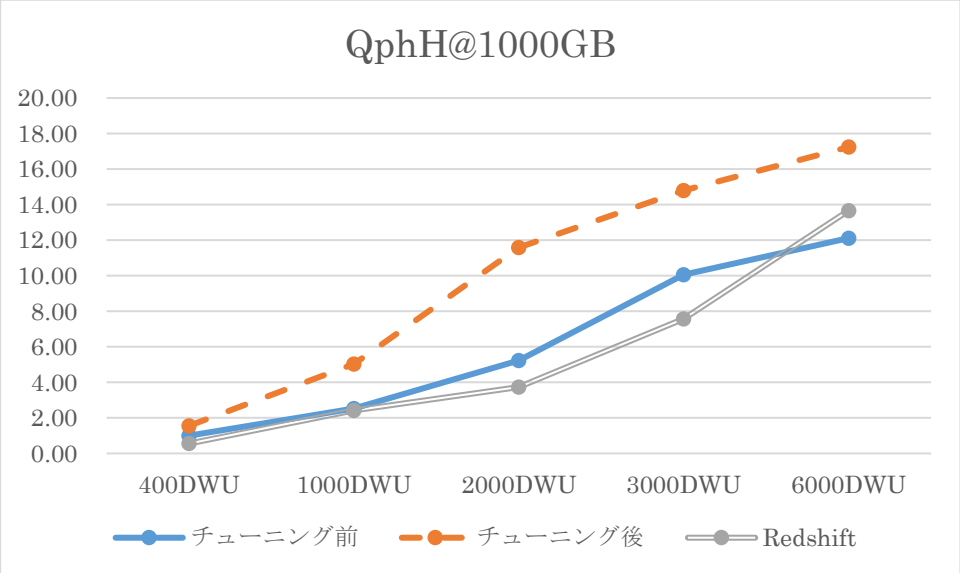


図３－４－２：シチュエーションごとの QphH@1000GB（相対値）

4 まとめ

今回の計測では、Azure SQL Data Warehouse は AWS Redshift にくらべ、TPC-H 22 本のクエリの実行時間は各 DWU の平均で、チューニング前で約 113%程度の時間がかかっている状況が確認できた。しかしながら、QphH@1000 で比較をすれば各 DWU の数値は AWS Redshift に比べ、129%の性能となっており、チューニング前でも Azure SQL Data Warehouse 性能は AWS Redshift と比較しほぼ同性能かそれ以上であると言える結果となった。同一の価格帯で同等の性能が出ている場合、Azure SQL Data Warehouse は 3 層構造アーキテクチャから、停止、起動やスケーリングが即座に可能なため性能単位のコストメリットは AWS Redshift よりも得やすいと考えられる。

また、チューニングを実施した場合には実行時間は、各 DWU の平均で、AWS Redshift の 63%程度の実行時間で完了しており、QphH@1000 も平均で AWS Redshift の約 224%良い結果となっている。この為、チューニングを実施した Azure SQL Data Warehouse は、AWS Redshift に比べ、1.5 倍～2 倍程度の性能であると推察される。

一方で SQL Data Warehouse が 6000DWU でスケールアウトは頭打ちとなっている。しかしながら、AWS Redshift ではさらに多くのスケールアウトが可能である。この為、超巨大なデータに対する高速なクエリ処理に関しては、AWS Redshift の方がサービスのキャパシティの観点からその恩恵を享受しやすいと考える。

Azure SQL Data Warehouse は格納されるデータ容量が 80TB～160TB 程度で 6000DWU の処理能力が妥当とアナウンスされており、今回の結果から鑑みると 160TB 前後までは Azure SQL Data Warehouse の方が性能および性能単位のコストメリットの観点から選定する事が望ましく、それ以上のデータ量になるとサービスのキャパシティの観点から AWS Redshift を選定する事が望ましいと言える。

5 参考資料

5.1 Azure SQL Data Warehouse の TPC-H クエリ

本検証で使用した Azure SQL Data Warehouse の TPC-H クエリは以下の通り。

```
/* TPC_H Query 1 - Pricing Summary Report */
SELECT L_RETURNFLAG, L_LINESTATUS, SUM(L_QUANTITY) AS SUM_QTY,
 SUM(L_EXTENDEDPRICE) AS SUM_BASE_PRICE, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS SUM_DISC_PRICE,
 SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)*(1+L_TAX)) AS SUM_CHARGE, AVG(L_QUANTITY) AS AVG_QTY,
 AVG(L_EXTENDEDPRICE) AS AVG_PRICE, AVG(L_DISCOUNT) AS AVG_DISC, COUNT(*) AS COUNT_ORDER
FROM LINEITEM
WHERE L_SHIPDATE <= dateadd(dd, -90, cast('1998-12-01' as datetime))
GROUP BY L_RETURNFLAG, L_LINESTATUS
ORDER BY L_RETURNFLAG,L_LINESTATUS

/* TPC_H Query 2 - Minimum Cost Supplier */
SELECT TOP 100 S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS, S_PHONE, S_COMMENT
FROM PART, SUPPLIER, PARTSUPP, NATION, REGION
WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY AND P_SIZE = 15 AND
P_TYPE LIKE '%%BRASS' AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND
R_NAME = 'EUROPE' AND
PS_SUPPLYCOST = (SELECT MIN(PS_SUPPLYCOST) FROM PARTSUPP, SUPPLIER, NATION, REGION
 WHERE P_PARTKEY = PS_PARTKEY AND S_SUPPKEY = PS_SUPPKEY
 AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY AND R_NAME = 'EUROPE')
ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY

/* TPC_H Query 3 - Shipping Priority */
SELECT TOP 10 L_ORDERKEY, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE, O_ORDERDATE, O_SHIPPRIORITY
FROM CUSTOMER, ORDERS, LINEITEM
WHERE C_MKTSEGMENT = 'BUILDING' AND C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND
O_ORDERDATE < '1995-03-15' AND L_SHIPDATE > '1995-03-15'
GROUP BY L_ORDERKEY, O_ORDERDATE, O_SHIPPRIORITY
ORDER BY REVENUE DESC, O_ORDERDATE

/* TPC_H Query 4 - Order Priority Checking */
SELECT O_ORDERPRIORITY, COUNT(*) AS ORDER_COUNT FROM ORDERS
WHERE O_ORDERDATE >= '1993-07-01' AND O_ORDERDATE < dateadd(mm,3, cast('1993-07-01' as datetime))
AND EXISTS (SELECT * FROM LINEITEM WHERE L_ORDERKEY = O_ORDERKEY AND L_COMMITDATE < L_RECEIPTDATE)
GROUP BY O_ORDERPRIORITY
ORDER BY O_ORDERPRIORITY
```

```
/* TPC_H Query 5 - Local Supplier Volume */
SELECT N_NAME, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE
FROM CUSTOMER, ORDERS, LINEITEM, SUPPLIER, NATION, REGION
WHERE C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND L_SUPPKEY = S_SUPPKEY
AND C_NATIONKEY = S_NATIONKEY AND S_NATIONKEY = N_NATIONKEY AND N_REGIONKEY = R_REGIONKEY
AND R_NAME = 'ASIA' AND O_ORDERDATE >= '1994-01-01'
AND O_ORDERDATE < DATEADD(YY, 1, cast('1994-01-01' as datetime))
GROUP BY N_NAME
ORDER BY REVENUE DESC

/* TPC_H Query 6 - Forecasting Revenue Change */
SELECT SUM(L_EXTENDEDPRICE*L_DISCOUNT) AS REVENUE
FROM LINEITEM
WHERE L_SHIPDATE >= '1994-01-01' AND L_SHIPDATE < dateadd(yy, 1, cast('1994-01-01' as datetime))
AND L_DISCOUNT BETWEEN .06 - 0.01 AND .06 + 0.01 AND L_QUANTITY < 24

/* TPC_H Query 7 - Volume Shipping */
SELECT SUPP_NATION, CUST_NATION, L_YEAR, SUM(VOLUME) AS REVENUE
FROM ( SELECT N1.N_NAME AS SUPP_NATION, N2.N_NAME AS CUST_NATION, datepart(yy, L_SHIPDATE) AS L_YEAR,
 L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME
 FROM SUPPLIER, LINEITEM, ORDERS, CUSTOMER, NATION N1, NATION N2
 WHERE S_SUPPKEY = L_SUPPKEY AND O_ORDERKEY = L_ORDERKEY AND C_CUSTKEY = O_CUSTKEY
 AND S_NATIONKEY = N1.N_NATIONKEY AND C_NATIONKEY = N2.N_NATIONKEY AND  ((N1.N_NAME = 'FRANCE' AND N2.N_NAME = 'GERMANY') OR
 (N1.N_NAME = 'GERMANY' AND N2.N_NAME = 'FRANCE')) AND
 L_SHIPDATE BETWEEN '1995-01-01' AND '1996-12-31' ) AS SHIPPING
GROUP BY SUPP_NATION, CUST_NATION, L_YEAR
ORDER BY SUPP_NATION, CUST_NATION, L_YEAR

/* TPC_H Query 8 - National Market Share */
SELECT O_YEAR, SUM(CASE WHEN NATION = 'BRAZIL' THEN VOLUME ELSE 0 END)/SUM(VOLUME) AS MKT_SHARE
FROM (SELECT datepart(yy,O_ORDERDATE) AS O_YEAR, L_EXTENDEDPRICE*(1-L_DISCOUNT) AS VOLUME, N2.N_NAME AS NATION
 FROM PART, SUPPLIER, LINEITEM, ORDERS, CUSTOMER, NATION N1, NATION N2, REGION
 WHERE P_PARTKEY = L_PARTKEY AND S_SUPPKEY = L_SUPPKEY AND L_ORDERKEY = O_ORDERKEY
 AND O_CUSTKEY = C_CUSTKEY AND C_NATIONKEY = N1.N_NATIONKEY AND
 N1.N_REGIONKEY = R_REGIONKEY AND R_NAME = 'AMERICA' AND S_NATIONKEY = N2.N_NATIONKEY
 AND O_ORDERDATE BETWEEN '1995-01-01' AND '1996-12-31' AND P_TYPE= 'ECONOMY ANODIZED STEEL') AS ALL_NATIONS
GROUP BY O_YEAR
ORDER BY O_YEAR
```

```
/* TPC_H Query 9 - Product Type Profit Measure */
SELECT NATION, O_YEAR, SUM(AMOUNT) AS SUM_PROFIT
FROM (SELECT N_NAME AS NATION, datepart(yy, O_ORDERDATE) AS O_YEAR,
 L_EXTENDEDPRICE*(1-L_DISCOUNT)-PS_SUPPLYCOST*L_QUANTITY AS AMOUNT
 FROM PART, SUPPLIER, LINEITEM, PARTSUPP, ORDERS, NATION
 WHERE S_SUPPKEY = L_SUPPKEY AND PS_SUPPKEY= L_SUPPKEY AND PS_PARTKEY = L_PARTKEY AND
 P_PARTKEY= L_PARTKEY AND O_ORDERKEY = L_ORDERKEY AND S_NATIONKEY = N_NATIONKEY AND
 P_NAME LIKE '%%green%%') AS PROFIT
GROUP BY NATION, O_YEAR
ORDER BY NATION, O_YEAR DESC

/* TPC_H Query 10 - Returned Item Reporting */
SELECT TOP 20 C_CUSTKEY, C_NAME, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS REVENUE, C_ACCTBAL,
N_NAME, C_ADDRESS, C_PHONE, C_COMMENT
FROM CUSTOMER, ORDERS, LINEITEM, NATION
WHERE C_CUSTKEY = O_CUSTKEY AND L_ORDERKEY = O_ORDERKEY AND O_ORDERDATE>= '1993-10-01' AND
O_ORDERDATE < dateadd(mm, 3, cast('1993-10-01' as datetime)) AND
L_RETURNFLAG = 'R' AND C_NATIONKEY = N_NATIONKEY
GROUP BY C_CUSTKEY, C_NAME, C_ACCTBAL, C_PHONE, N_NAME, C_ADDRESS, C_COMMENT
ORDER BY REVENUE DESC

/* TPC_H Query 11 - Important Stock Identification */
SELECT PS_PARTKEY, SUM(PS_SUPPLYCOST*PS_AVAILQTY) AS VALUE
FROM PARTSUPP, SUPPLIER, NATION
WHERE PS_SUPPKEY = S_SUPPKEY AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'GERMANY'
GROUP BY PS_PARTKEY
HAVING SUM(PS_SUPPLYCOST*PS_AVAILQTY) > (SELECT SUM(PS_SUPPLYCOST*PS_AVAILQTY) * 0.0001000000
 FROM PARTSUPP, SUPPLIER, NATION
 WHERE PS_SUPPKEY = S_SUPPKEY AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'GERMANY')
ORDER BY VALUE DESC

/* TPC_H Query 12 - Shipping Modes and Order Priority */
SELECT L_SHIPMODE,
SUM(CASE WHEN O_ORDERPRIORITY = '1-URGENT' OR O_ORDERPRIORITY = '2-HIGH' THEN 1 ELSE 0 END) AS HIGH_LINE_COUNT,
SUM(CASE WHEN O_ORDERPRIORITY <> '1-URGENT' AND O_ORDERPRIORITY <> '2-HIGH' THEN 1 ELSE 0 END ) AS LOW_LINE_COUNT
FROM ORDERS, LINEITEM
WHERE O_ORDERKEY = L_ORDERKEY AND L_SHIPMODE IN ('MAIL','SHIP')
AND L_COMMITDATE < L_RECEIPTDATE AND L_SHIPDATE < L_COMMITDATE AND L_RECEIPTDATE >= '1994-01-01'
AND L_RECEIPTDATE < dateadd(mm, 1, cast('1995-09-01' as datetime))
GROUP BY L_SHIPMODE
ORDER BY L_SHIPMODE
```

```
/* TPC_H Query 13 - Customer Distribution */
SELECT C_COUNT, COUNT(*) AS CUSTDIST
FROM (SELECT C_CUSTKEY, COUNT(O_ORDERKEY)
 FROM CUSTOMER left outer join ORDERS on C_CUSTKEY = O_CUSTKEY
 AND O_COMMENT not like '%%special%%requests%%'
 GROUP BY C_CUSTKEY) AS C_ORDERS (C_CUSTKEY, C_COUNT)
GROUP BY C_COUNT
ORDER BY CUSTDIST DESC, C_COUNT DESC

/* TPC_H Query 14 - Promotion Effect */
SELECT 100.00* SUM(CASE WHEN P_TYPE LIKE 'PROMO%%' THEN L_EXTENDEDPRICE*(1-L_DISCOUNT)
ELSE 0 END) / SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) AS PROMO_REVENUE
FROM LINEITEM, PART
WHERE L_PARTKEY = P_PARTKEY AND L_SHIPDATE >= '1995-09-01' AND L_SHIPDATE < dateadd(mm, 1, '1995-09-01')

/* TPC_H Query 15 - Create View for Top Supplier Query */
CREATE VIEW REVENUE0 (SUPPLIER_NO, TOTAL_REVENUE) AS
SELECT L_SUPPKEY, SUM(L_EXTENDEDPRICE*(1-L_DISCOUNT)) FROM LINEITEM
WHERE L_SHIPDATE >= '1996-01-01' AND L_SHIPDATE < dateadd(mm, 3, cast('1996-01-01' as datetime))
GROUP BY L_SUPPKEY
GO

/* TPC_H Query 15 - Top Supplier */
SELECT S_SUPPKEY, S_NAME, S_ADDRESS, S_PHONE, TOTAL_REVENUE
FROM SUPPLIER, REVENUE0
WHERE S_SUPPKEY = SUPPLIER_NO AND TOTAL_REVENUE = (SELECT MAX(TOTAL_REVENUE) FROM REVENUE0)
ORDER BY S_SUPPKEY

DROP VIEW REVENUE0

/* TPC_H Query 16 - Parts/Supplier Relationship */
SELECT P_BRAND, P_TYPE, P_SIZE, COUNT(DISTINCT PS_SUPPKEY) AS SUPPLIER_CNT
FROM PARTSUPP, PART
WHERE P_PARTKEY = PS_PARTKEY AND P_BRAND <> 'Brand#45' AND P_TYPE NOT LIKE 'MEDIUM POLISHED%%'
AND P_SIZE IN (49, 14, 23, 45, 19, 3, 36, 9) AND PS_SUPPKEY NOT IN (SELECT S_SUPPKEY FROM SUPPLIER
 WHERE S_COMMENT LIKE '%%Customer%%Complaints%%')
GROUP BY P_BRAND, P_TYPE, P_SIZE
ORDER BY SUPPLIER_CNT DESC, P_BRAND, P_TYPE, P_SIZE
```

```
/* TPC_H Query 17 - Small-Quantity-Order Revenue */
SELECT SUM(L_EXTENDEDPRICE)/7.0 AS AVG_YEARLY FROM LINEITEM, PART
WHERE P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#23' AND P_CONTAINER = 'MED BOX'
AND L_QUANTITY < (SELECT 0.2*AVG(L_QUANTITY) FROM LINEITEM WHERE L_PARTKEY = P_PARTKEY)

/* TPC_H Query 18 - Large Volume Customer */
SELECT TOP 100 C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE, SUM(L_QUANTITY)
FROM CUSTOMER, ORDERS, LINEITEM
WHERE O_ORDERKEY IN (SELECT L_ORDERKEY FROM LINEITEM GROUP BY L_ORDERKEY HAVING
 SUM(L_QUANTITY) > 300) AND C_CUSTKEY = O_CUSTKEY AND O_ORDERKEY = L_ORDERKEY
GROUP BY C_NAME, C_CUSTKEY, O_ORDERKEY, O_ORDERDATE, O_TOTALPRICE
ORDER BY O_TOTALPRICE DESC, O_ORDERDATE

/* TPC_H Query 19 - Discounted Revenue */
SELECT SUM(L_EXTENDEDPRICE* (1 - L_DISCOUNT)) AS REVENUE
FROM LINEITEM, PART
WHERE (P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#12' AND P_CONTAINER IN ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG') AND L_QUANTITY >= 1 AND L_QUANTITY <= 1 + 10
AND P_SIZE BETWEEN 1 AND 5
AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
OR (P_PARTKEY = L_PARTKEY AND P_BRAND ='Brand#23' AND P_CONTAINER IN ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK') AND L_QUANTITY >=10 AND L_QUANTITY <=10 +
10 AND P_SIZE BETWEEN 1 AND 10
AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')
OR (P_PARTKEY = L_PARTKEY AND P_BRAND = 'Brand#34' AND P_CONTAINER IN ( 'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG') AND L_QUANTITY >=20 AND L_QUANTITY <= 20 + 10 AND
P_SIZE BETWEEN 1 AND 15
AND L_SHIPMODE IN ('AIR', 'AIR REG') AND L_SHIPINSTRUCT = 'DELIVER IN PERSON')

/* TPC_H Query 20 - Potential Part Promotion */
SELECT S_NAME, S_ADDRESS FROM SUPPLIER, NATION
WHERE S_SUPPKEY IN (SELECT PS_SUPPKEY FROM PARTSUPP
 WHERE PS_PARTKEY in (SELECT P_PARTKEY FROM PART WHERE P_NAME like 'forest%%') AND
 PS_AVAILQTY > (SELECT 0.5*sum(L_QUANTITY) FROM LINEITEM WHERE L_PARTKEY = PS_PARTKEY AND
   L_SUPPKEY = PS_SUPPKEY AND L_SHIPDATE >= '1994-01-01' AND
   L_SHIPDATE < dateadd(yy,1,'1994-01-01'))) AND S_NATIONKEY = N_NATIONKEY AND N_NAME = 'CANADA'
ORDER BY S_NAME
```

```
/* TPC_H Query 21 - Suppliers Who Kept Orders Waiting */
SELECT TOP 100 S_NAME, COUNT(*) AS NUMWAIT
FROM SUPPLIER, LINEITEM L1, ORDERS, NATION WHERE S_SUPPKEY = L1.L_SUPPKEY AND
O_ORDERKEY = L1.L_ORDERKEY AND O_ORDERSTATUS = 'F' AND L1.L_RECEIPTDATE> L1.L_COMMITDATE
AND EXISTS (SELECT * FROM LINEITEM L2 WHERE L2.L_ORDERKEY = L1.L_ORDERKEY
 AND L2.L_SUPPKEY <> L1.L_SUPPKEY) AND
NOT EXISTS (SELECT * FROM LINEITEM L3 WHERE L3.L_ORDERKEY = L1.L_ORDERKEY AND
 L3.L_SUPPKEY <> L1.L_SUPPKEY AND L3.L_RECEIPTDATE > L3.L_COMMITDATE) AND
S_NATIONKEY = N_NATIONKEY AND N_NAME = 'SAUDI ARABIA'
GROUP BY S_NAME
ORDER BY NUMWAIT DESC, S_NAME

/* TPC_H Query 22 - Global Sales Opportunity */
SELECT CNTRYCODE, COUNT(*) AS NUMCUST, SUM(C_ACCTBAL) AS TOTACCTBAL
FROM (SELECT SUBSTRING(C_PHONE,1,2) AS CNTRYCODE, C_ACCTBAL
  FROM CUSTOMER WHERE SUBSTRING(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17') AND
  C_ACCTBAL > (SELECT AVG(C_ACCTBAL) FROM CUSTOMER WHERE C_ACCTBAL > 0.00 AND
    SUBSTRING(C_PHONE,1,2) IN ('13', '31', '23', '29', '30', '18', '17')) AND
  NOT EXISTS ( SELECT * FROM ORDERS WHERE O_CUSTKEY = C_CUSTKEY)) AS CUSTSALE
GROUP BY CNTRYCODE
ORDER BY CNTRYCODE
```

5.2 AWS Redshift の TPC-H クエリ

本検証で使用した AWS Redshift の TPC-H クエリは以下の通り。

```
/* TPC_H Query 1 - Pricing Summary Report */
select l_returnflag,l_linestatus,sum(l_quantity) as sum_qty,sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,avg(l_extendedprice) as avg_price,avg(l_discount) as avg_disc,count(*) as count_order
from lineitem
where l_shipdate <= date '1998-12-01' - interval '117' day
group by l_returnflag,l_linestatus
order by l_returnflag,l_linestatus
limit 1;
```

```
/* TPC_H Query 2 - Minimum Cost Supplier */

select s_acctbal,s_name,n_name,p_partkey,p_mfgr,s_address,s_phone,s_comment

from part,supplier,partsupp,nation,region

where

		p_partkey = ps_partkey and s_suppkey = ps_suppkey and p_size = 15 and p_type like '%BRASS'

		and s_nationkey = n_nationkey and n_regionkey = r_regionkey and r_name = 'EUROPE'

		and ps_supplycost = (

				select min(ps_supplycost)

				from partsupp,supplier,nation,region

				where p_partkey = ps_partkey and s_suppkey = ps_suppkey and s_nationkey = n_nationkey

					and n_regionkey = r_regionkey and r_name = 'EUROPE')

order by s_acctbal desc,n_name,s_name,p_partkey

LIMIT 100;


/* TPC_H Query 3 - Shipping Priority */

select l_orderkey,sum(l_extendedprice * (1 - l_discount)) as revenue,o_orderdate,o_shippriority

from customer,orders,lineitem

where c_mktsegment = 'BUILDING' and c_custkey = o_custkey and l_orderkey = o_orderkey

		and o_orderdate < date '1995-03-15' and l_shipdate > date '1995-03-15'

group by l_orderkey,o_orderdate,o_shippriority

order by revenue desc,o_orderdate

limit 10;


/* TPC_H Query 4 - Order Priority Checking */

select o_orderpriority,count(*) as order_count

from orders

where

		o_orderdate >= date '1993-07-01' and o_orderdate < date '1993-07-01' + interval '3' month

		and exists (

				select *

				from lineitem

				where l_orderkey = o_orderkey and l_commitdate < l_receiptdate)

group by o_orderpriority

order by o_orderpriority

limit 1;
```

```
/* TPC_H Query 5 - Local Supplier Volume */

select n_name,sum(l_extendedprice * (1 - l_discount)) as revenue

from customer,orders,lineitem,supplier,nation,region

where c_custkey = o_custkey and l_orderkey = o_orderkey and l_suppkey = s_suppkey and c_nationkey = s_nationkey

                and s_nationkey = n_nationkey and n_regionkey = r_regionkey and r_name = 'ASIA'

                and o_orderdate >= date '1994-01-01' and o_orderdate < date '1994-01-01' + interval '1' year

group by n_name

order by revenue desc

limit 1;


/* TPC_H Query 6 - Forecasting Revenue Change */

select sum(l_extendedprice * l_discount) as revenue

from lineitem

where

                l_shipdate >= date '1994-01-01' and l_shipdate < date '1994-01-01' + interval '1' year

                and l_discount between 0.06 - 0.01 and 0.06 + 0.01 and l_quantity < 24

limit 1;


/* TPC_H Query 7 - Volume Shipping */

select supp_nation,cust_nation,l_year,sum(volume) as revenue

from ( select

                n1.n_name as supp_nation,n2.n_name as cust_nation,extract(year from l_shipdate) as l_year,l_extendedprice * (1 - l_discount) as volume

                from supplier,lineitem,orders,customer,nation n1,nation n2

                where s_suppkey = l_suppkey and o_orderkey = l_orderkey and c_custkey = o_custkey

                and s_nationkey = n1.n_nationkey and c_nationkey = n2.n_nationkey

                and ((n1.n_name = 'FRANCE' and n2.n_name = 'GERMANY') or (n1.n_name = 'GERMANY' and n2.n_name = 'FRANCE'))

                and l_shipdate between date '1995-01-01' and date '1996-12-31'

) as shipping

group by supp_nation,cust_nation,l_year

order by supp_nation,cust_nation,l_year

limit 1;
```

```
/* TPC_H Query 8 - National Market Share */

select o_year,sum(case when nation = 'INDIA' then volume else 0 end) / sum(volume) as mkt_share

from (

		select extract(year from o_orderdate) as o_year,l_extendedprice * (1 - l_discount) as volume,n2.n_name as nation

		from part,supplier,lineitem,orders,customer,nation n1,nation n2,region

		where p_partkey = l_partkey and s_suppkey = l_suppkey and l_orderkey = o_orderkey

		and o_custkey = c_custkey and c_nationkey = n1.n_nationkey and n1.n_regionkey = r_regionkey

		and r_name = 'AMERICA' and s_nationkey = n2.n_nationkey

		and o_orderdate between date '1995-01-01' and date '1996-12-31' and p_type = 'ECONOMY ANODIZED STEEL'

) as all_nations

group by o_year

order by o_year

limit 1;


/* TPC_H Query 9 - Product Type Profit Measure */

select nation,o_year,sum(amount) as sum_profit

from (

		select  n_name as nation,extract(year from o_orderdate) as o_year,l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity as amount

		from part,supplier,lineitem,partsupp,orders,nation

		where s_suppkey = l_suppkey and ps_suppkey = l_suppkey and ps_partkey = l_partkey and p_partkey = l_partkey

		and o_orderkey = l_orderkey and s_nationkey = n_nationkey and p_name like '%green%') as profit

group by nation,o_year

order by nation,o_year desc

limit 1;


/* TPC_H Query 10 - Returned Item Reporting */

select c_custkey,c_name,sum(l_extendedprice * (1 - l_discount)) as revenue,c_acctbal,n_name,c_address,c_phone,c_comment

from customer,orders,lineitem,nation

where c_custkey = o_custkey and l_orderkey = o_orderkey and o_orderdate >= date '1993-10-01'

and o_orderdate < date '1993-10-01' + interval '3' month and l_returnflag = 'R' and c_nationkey = n_nationkey

group by c_custkey,c_name,c_acctbal,c_phone,n_name,c_address,c_comment

order by revenue desc

limit 1;
```

```
/* TPC_H Query 11 - Important Stock Identification */
select ps_partkey,sum(ps_supplycost * ps_availqty) as value
from partsupp,supplier,nation
where ps_suppkey = s_suppkey and s_nationkey = n_nationkey and n_name = 'GERMANY'
group by ps_partkey having
        sum(ps_supplycost * ps_availqty) > (
                select sum(ps_supplycost * ps_availqty) * 0.0001000000
                from partsupp,  supplier, nation
                where ps_suppkey = s_suppkey and s_nationkey = n_nationkey and n_name = 'GERMANY')
order by value desc
limit 1;

/* TPC_H Query 12 - Shipping Modes and Order Priority */
select l_shipmode,sum(case when o_orderpriority = '1-URGENT' or o_orderpriority = '2-HIGH' then 1 else 0 end) as high_line_count,
        sum(case when o_orderpriority <> '1-URGENT' and o_orderpriority <> '2-HIGH' then 1 else 0 end) as low_line_count
from orders,lineitem
where o_orderkey = l_orderkey and l_shipmode in ('MAIL', 'SHIP') and l_commitdate < l_receiptdate
and l_shipdate < l_commitdate and l_receiptdate >= date '1994-01-01' and l_receiptdate < date '1995-09-01' + interval '1' mounth
group by l_shipmode
order by l_shipmode
limit 1;

/* TPC_H Query 13 - Customer Distribution */
select c_count,count(*) as custdist
from (
        select c_custkey,count(o_orderkey)
        from customer left outer join orders on c_custkey = o_custkey and o_comment not like '%special%requests%'
        group by c_custkey
) as c_orders (c_custkey, c_count)
group by c_count
order by custdist desc,c_count desc
limit 1;

/* TPC_H Query 14 - Promotion Effect */
select
        100.00 * sum(case when p_type like 'PROMO%' then l_extendedprice * (1 - l_discount) else 0end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from lineitem,part
where l_partkey = p_partkey and l_shipdate >= date '1995-09-01' and l_shipdate < date '1995-09-01' + interval '1' month
limit 1;
```

```
/* TPC_H Query 15 - Create View for Top Supplier Query */

create view revenue0 (supplier_no, total_revenue) as

	select l_suppkey,sum(l_extendedprice * (1 - l_discount))

	from lineitem

	where l_shipdate >= date '1996-01-01' and l_shipdate < date '1996-01-01' + interval '3' month

	group by l_suppkey;


/* TPC_H Query 15 - Top Supplier */

select s_suppkey,s_name,s_address,s_phone,total_revenue

from supplier,revenue0

where s_suppkey = supplier_no and total_revenue = (select max(total_revenue) from revenue0)

order by s_suppkey;


drop view revenue0

limit 1;


/* TPC_H Query 16 - Parts/Supplier Relationship */

select p_brand,p_type,p_size,count(distinct ps_suppkey) as supplier_cnt

from partsupp,part

where p_partkey = ps_partkey and p_brand <> 'Brand#45' and p_type not like 'MEDIUM POLISHED%' and p_size in (49, 14, 23, 45, 19, 3, 36, 9)

	and ps_suppkey not in (

		select s_suppkey

		from supplier

		where s_comment like '%Customer%Complaints%')

group by p_brand,p_type,p_size

order by supplier_cnt desc,p_brand,p_type,p_size

limit 1;


/* TPC_H Query 17 - Small-Quantity-Order Revenue */

select sum(l_extendedprice) / 7.0 as avg_yearly

from lineitem,part,(select l_partkey as agg_partkey, 0.2 * avg(l_quantity) as avg_quantity from lineitem group by l_partkey) part_agg

where p_partkey = l_partkey and agg_partkey = l_partkey and p_brand = 'Brand#23' and p_container = 'MED BOX' and l_quantity < avg_quantity

limit 1;


/* TPC_H Query 18 - Large Volume Customer */

select c_name,c_custkey,o_orderkey,o_orderdate,o_totalprice,sum(l_quantity)

from customer,orders,lineitem

where o_orderkey in (select l_orderkey from lineitem group by l_orderkey having sum(l_quantity) > 300)

and c_custkey = o_custkey and o_orderkey = l_orderkey

group by c_name,c_custkey,o_orderkey,o_orderdate,o_totalprice

order by o_totalprice desc,o_orderdate

limit 1;
```

```
/* TPC_H Query 19 - Discounted Revenue */
select sum(l_extendedprice* (1 - l_discount)) as revenue
from lineitem,part
where (p_partkey = l_partkey and p_brand = 'Brand#12' and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
and l_quantity >= 1 and l_quantity <= 1 + 10 and p_size between 1 and 5 and l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct = 'DELIVER IN PERSON')
or (p_partkey = l_partkey and p_brand = 'Brand#23' and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
and l_quantity >= 10 and l_quantity <= 10 + 10 and p_size between 1 and 10 and l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct = 'DELIVER IN PERSON')
or (p_partkey = l_partkey and p_brand = 'Brand#34' and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG') and l_quantity >= 20
and l_quantity <= 20 + 10 and p_size between 1 and 15 and l_shipmode in ('AIR', 'AIR REG') and l_shipinstruct = 'DELIVER IN PERSON')
limit 1;

/* TPC_H Query 20 - Potential Part Promotion */
select s_name,s_address
from supplier,nation
where s_suppkey in (select ps_suppkey
        from partsupp,( select l_partkey agg_partkey,l_suppkey agg_suppkey,0.5 * sum(l_quantity) AS agg_quantity
                from lineitem
                where l_shipdate >= date '1994-01-01' and l_shipdate < date '1994-01-01' + interval '1' year
                group by l_partkey, l_suppkey) agg_lineitem
        where agg_partkey = ps_partkey and agg_suppkey = ps_suppkey and ps_partkey in (
                select p_partkey
                from part
                where p_name like 'forest%')
        and ps_availqty > agg_quantity)
and s_nationkey = n_nationkey and n_name = 'CANADA'
order by s_name
limit 1;

/* TPC_H Query 21 - Suppliers Who Kept Orders Waiting */
select s_name,count(*) as numwait
from supplier,lineitem l1,orders,nation
where s_suppkey = l1.l_suppkey and o_orderkey = l1.l_orderkey and o_orderstatus = 'F' and l1.l_receiptdate > l1.l_commitdate
        and exists ( select * from lineitem l2 where l2.l_orderkey = l1.l_orderkey and l2.l_suppkey <> l1.l_suppkey)
        and not exists (select * from lineitem l3 where l3.l_orderkey = l1.l_orderkey and l3.l_suppkey <> l1.l_suppkey and l3.l_receiptdate > l3.l_commitdate)
        and s_nationkey = n_nationkey and n_name = 'SAUDI ARABIA'
group by s_name
order by numwait desc,s_name
limit 1;
```

```
/* TPC_H Query 22 - Global Sales Opportunity */

select cntrycode,count(*) as numcust,sum(c_acctbal) as totacctbal

from ( select substring(c_phone from 1 for 2) as cntrycode,c_acctbal from customer

        where substring(c_phone from 1 for 2) in ('13', '31', '23', '29', '30', '18', '17')

                and c_acctbal > ( select avg(c_acctbal)

                from customer

                where c_acctbal > 0.00 and substring(c_phone from 1 for 2) in ('13', '31', '23', '29', '30', '18', '17'))

                and not exists ( select * from orders where o_custkey = c_custkey)) as custsale

group by cntrycode

order by cntrycode

limit 1;
```

以上