Converting TPC-H Query Templates to Use DSQGEN for Easy Extensibility

John M. Stephens Jr.[1] and Meikel Poess[2]

[1] Gradient Systems, 643 Bair Island Road #103, Redwood City, CA-94063
jms@gradientsystems.com
[2] Oracle Corporation, 500 Oracle Parkway, Redwood Shores, CA-94107
meikel.poess@oracle.com

Abstract. The ability to automatically generate queries that are not known a-priory is crucial for ad-hoc benchmarks. TPC-H solves this problem with a query generator, QGEN, which utilizes query templates to generate SQL queries. QGEN's architecture makes it difficult to maintain, change or adapt to new types of query templates since every modification requires code changes. DSQGEN, a generic query generator, originally written for the TPC-DS benchmark, uses a query template language, which allows for easy modification and extension of existing query templates. In this paper we show how the current set of TPC-H query templates can be migrated to the template language of DSQGEN without any change to comparability of published TPC-H results. The resulting query template model provides opportunities for easier enhancement and extension of the TPC-H workload, which we demonstrate.

Keywords: Benchmark Development, Databases, Performance Analysis.

1 Introduction

TPC-H [4][6]has been a very successful benchmark for the Transaction Processing Performance Council (TPC), with 147 results published as of June 2009. It relies on a pair of executables for data and query generation (DBGEN and QGEN, respectively) that were originally developed for its predecessor, TPC-D [5], which was released in 1994. QGEN is a command-line utility that uses pattern matching to expand the 22 query templates defined in TPC-H into fully qualified Structured Query Language (SQL). While the substitutions defined in the TPC-H query set have proven adequate, they have not been updated since five new templates were added in 1999, when TPC-D morphed into TPC-H. Further, the substitutions are hard-coded into the QGEN executable. As a result, any refinement or expansion of the query set requires additional software development. The required costs for code modifications and code testing have hindered further evolution of the benchmark.

The underlying design of QGEN remains valid. Its template-based query model and common and well-understood business questions provide TPC-H with a high degree of comparability between benchmark executions. At the same time the precise values or targets of a given instance of a query are random, assuring appropriate variability and limiting the amount of foreknowledge that a test sponsor can employ. The result is a query set that provides consistent and meaningful results, while mimicking ad-hoc queries. However, the TPC-H query model has two inherent problems: The query substitutions are hard coded into the query generator and cannot be modified without additional software development and the query templates themselves use a narrow range of syntax and substitution types, and no longer capture the breadth of common decision support systems.

This paper details the migration of the QGEN query template model to the DSQGEN query template model without any changes to TPC-H's current query template set. This preserves the investment that test sponsors have made in TPC-H, and, simultaneously provides the opportunity for an updated query set which employs a richer set of query operations and syntax. In addition, it leaves further enhancement in the hands of the benchmark designers, without requiring further software development.

The remainder of this paper is organized as follows: Section 2 gives a brief overview of TPC-H focusing on how queries are currently generated with QGEN; section 3 introduces the essential syntax of DSQGEN, including both functions needed to write current TPC-H query templates in DSQGEN's query template language and additional functionality that exceeds the current needs of TPC-H; section 4 demonstrates the changes required to migrate the current set of 22 TPC-H queries to DSQGEN's query template language; section 5 shows how the TPC-H query set can be extended using DSQGEN.

2 TPC-H

Since its introduction in 1999 by the Transaction Performance Council, TPC-H has been the industry standard benchmark for data warehouse applications. This section briefly introduces those elements of TPC-H, which are necessary for the understanding the next sections.

2.1 Background

TPC-H models the activity of any industry, which manages, sells, and distributes products worldwide (e.g., car rental, food distribution, parts, suppliers, etc.). It uses a 3rd normal form schema consisting of eight base tables. They are populated with synthetic data, scaled to an aggregate volume or scale factor (SF). For example, in a database with SF=100, the base tables hold 100 gigabytes of generated data. Fig. 1 illustrates the entity relationship (ER) diagram of the TPC-H schema. The two largest tables, Lineitem and Orders contain about 83 percent of the data. Sizes of all tables, except for nation and region scale linearly with the scale factor.

The TPC-H workload consists of database load, execution of 22 read-only queries in both single and multi-user mode and two refresh functions. The queries are intended to test the most common query capabilities of a typical decision support system. In order to facilitate the understanding of TPC-H queries and the mapping of the benchmark queries to real world situations, each query is described in terms of a business question. This business question is formulated in English explaining the

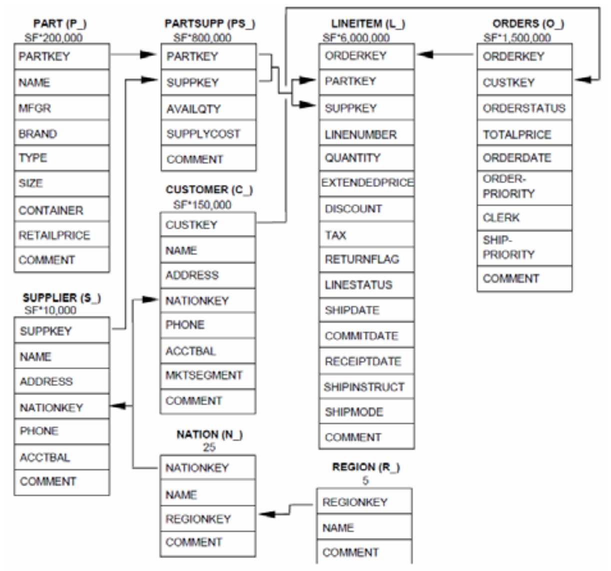


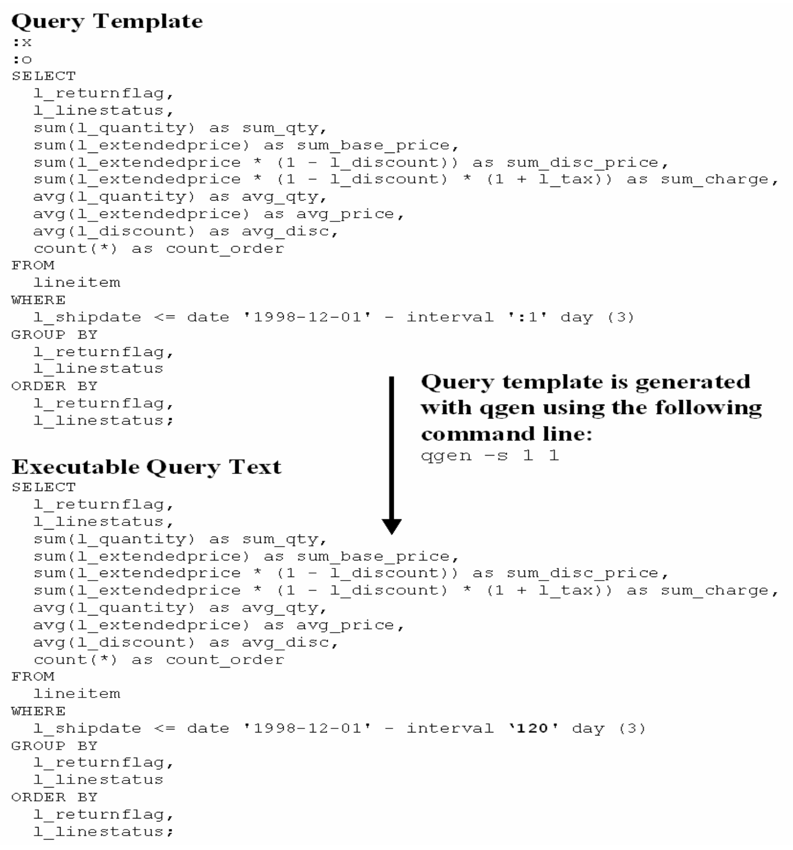
Fig. 1. TPC-H Entity Relationship Diagram (Source: TPC-H Version 2.8.0)

result of the query in context of TPC-H's business model. The business questions are translated into functional query definitions that define the queries using the SQL-92 query language. TPC-H queries are chosen to perform operations that are relevant to common data warehouse applications. Accordingly, the demands a query places on the hardware (processor, IO-subsystem) and software (Operating System, Database Management System) of the tested system varies from query to query. To assure that the benchmark remains dynamic, each TPC-H query contains substitution parameters that are randomly chosen by the benchmark driver immediately before its execution, to mimic ad-hoc workloads.

One TPC-H performance run consists of one execution of a Power Test (see Clause 6.3.3 of [6]), followed by one execution of a Throughput Test described (see Clause 6.3.4. of [6]). The Power Test measures single-user performance. Single-user performance measures a systems ability to parallelize queries across all available resources (memory, CPU, I/O) in order to deliver the result in the least amount of time. In TPC-H's Power Test the single-user performance measurement is implemented as one stream of queries. This stream contains all 22 queries in a pre-defined order (see Appendix A in [6]): The Throughput Test measures multi-user performance. A multi-user test measures a system's ability to execute multiple concurrent queries, allocate resources efficiently across all users to maximize query throughput. In TPC-H's throughput test multi-user measurement is implemented with n concurrent streams. Each stream contains all 22 queries ordered in a different permutation.

2.2 TPC-H Data Generator QGEN

QGEN produces the query streams required by TPC-H. The templates each contain between one and five substitution tokens, each with a static set of possible values.



Query Template

```
:x
:o
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 ':1' day (3)
GROUP BY
  l_returnflag,
  l_linestatus
ORDER BY
  l_returnflag,
  l_linestatus;
```

Query template is generated with qgen using the following command line:

```
qgen -s 1 1
```

Executable Query Text

```
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 '120' day (3)
GROUP BY
  l_returnflag,
  l_linestatus
ORDER BY
  l_returnflag,
  l_linestatus;
```

Fig. 2. Sample QGEN Template Usage

QGEN replaces the substitution token with a randomly selected value from the permissible domain to produce fully qualified SQL. Fig. 2 illustrates the transformation of the query template of query 2 to a valid entry in a query stream.

Clearly, QGEN depends heavily on the underlying data set defined for TPC-H. A query generator can only exploit data relationships that exist in its target data population. In QGEN, these relationships are captured in the source code of the query generator itself. This means that the query set cannot be modified without modifying the query generator, and that the relationships and domains employed by the queries can only be discovered by referring to the benchmark specification or to the source code of the query generator. Similarly, the query template permutations that define the benchmark's query streams are hard-coded into QGEN itself, and rely on a static, hand-cobbled query ordering which cannot be extended without source code changes to QGEN.

3 Query Generator DSQGEN

DSQGEN (a.k.a. QGEN2 see [6]) was developed by the TPC for a proposed decision support benchmark. It is a command-line utility that translates an arbitrary set of query templates into streams of valid SQL statements. Query templates are defined in a template language, and parsed using an LALR(1) grammar. DSQGEN provides a rich set of query template translation semantics that go far beyond what is required to support the TPC-H query set. For example, if DSQGEN is used in combination with DSDGEN (a.k.a. MUDD see [1][3]), distribution-based predicate substitutions can be defined. The distribution-related substitutions allow a template to use arbitrary distributions, encoded as ASCII histograms. The result is a tight linkage between data generation and query generation without requiring the template designer to know the specifics of the data distributions. For a detailed description of DSQGEN's functionality, including its sophisticated template language, refer to [2]. The following sections only address those parts of the template language that are relevant to this paper.

3.1 Query Template Grammar

A query template is divided into two parts: substitution definitions and SQL Text. The substitution definitions specify the substitution rules for a query as a list of substitution tags. These tags control the translation of the SQL Text portion of the template into a valid SQL query. Once defined, a substitution tag can be used throughout the query template. Each occurrence of the substitution tag is replaced by values generated according to the tag's definition. Multiple occurrences of the same tag are replaced with the same value. If a substitution tag is post-fixed with a number, then each unique tag/number combination receives a unique value. A simplified grammar for substitution tag, limited to the <random> and <text> substitution types used in Section 4, is outlined in Fig. 3.

```
<tag>= <exp>|
       string[30]|
       <substitution type>|
       list(<substitution type>,<exp>)|
       ulist(<substitution type>,<exp>);
<substitution type>=<random> | <text>
<random> = (<min>,<max>,uniform)
<exp> = <exp>-<exp>|<exp>+<exp>|<exp>/<exp>|
        <exp>*<exp>|<exp>%<exp>|<number>|<const>
<number>=<number>|0|1|2|3|4|5|6|7|8|9
<const>=_SCALE|_SEED|_QUERY|_TEMPLATE|_STREAM|_LIMIT
```

Fig. 3. Basic Substitution Declaration Grammar

The Random substitution type allows defining tags to use randomly-generated integers in an inclusive range [min, max] using a uniform distribution. The specific grammar for a <random> substitution tag is:

```
<random> = random(<exp>,<exp>,uniform);
```

Let's call the first expression min and the second expression max. The likelihood P_i of each value to be picked by DSQGEN is identical:

$$P_i = \frac{1}{\max - \min} \tag{1}$$

The location parameter *min* and the scale parameter *max* must be picked such that *min<max*. The designer of a query template must assure that the values picked for *min* and *max* fall within the range of the data distribution of the targeted column. The following examples show how the random substitution tag can be used:

Example 1 `order_quantity = random (1, 10, uniform);`

Example 2 `price_int=random(1,1000,uniform;`
`price_frac=random(1,100,uniform);`

Example 3
```
birthday=random("1929-01-01"
                ,"2009-05-31",uniform);
```

Example 1 defines a tag, which randomly chooses a value between 1 and 10. This can be used as a projection predicate on a quantity column. Example 2 defines two tags, one to generate the integer portion of a price (price_int) and a second (price_frac) to generate the fraction of a price. The price can then be combined in the SQL text as: [price_int]+1/[price_frac]. Example 3 selects a random date between 2009 and 2029, with appropriate allowances for leap years.

The TEXT substitution, which uses the grammar shown in Fig. 4, replaces a particular tag with one of a weighted set of ASCII strings. This substitution type can be employed in many different ways. In its basic form, this can be employed in a projection predicate such as: `column_name = "<string>"`, providing a crude form of text searching. The elements of a TEXT substitution tag must be distinct. The empty string is permissible.

```
<text> =({<subelem>, <weight>}<subelem_weight>);
<subelem_weight>=,{<subelem>, <weight>}|NULL
<subelem> = string[100];
<weight>  = integer;
```

Fig. 4. TEXT Substitution Type

The likelihood of a particular "subelem" to be picked as a substitution parameter depends on the ratio of its weight (<subelem_weight>) to the sum of all weights. The probability of S_i for the following definition with n elements tag=TEXT({"S_1", W_1},…,{"S_1", W_1}) is defined as :

$$P_{S_i} = \frac{W_i}{\sum_{j=1}^{n} W_j} : 0 \le i \le n : W_i > 0 \tag{2}$$

Table 1. DSQGEN Build-In Functions

Keyword	Value
_SCALE	Scale factor in GB, as set with -scale command line option
_SEED	Current random-number-generator seed
_QUERY	Sequence number within a query set
_TEMPLATE	Template ID
_STREAM	Query stream ID
_LIMITA,_LIMITB,_LIMITC	Used for vendor specific syntax to limit the number of rows
_LIMIT	Maximum number of rows to be returned by the query

Example 4
```
dog=TEXT({"German
sheppard",1},{"poodle",1},{"pug",1});
```
Example 5
```
dog_color=TEXT({"brown",6},{"black",3},{"gre
y",1});
```

Example 4 defines the tag dog, which generates values German Sheppard, poodle and pug with the same likelihood. Example 5 defines dog colors to be brown, back or grey. However, in this example the color brown gets picked six out of ten times, while back gets picked three out of ten times and grey gets only picked one out of ten times.

By default, substitution tags produce singleton values. When combined with the LIST or ULIST operators, each tag produces a list of <number> values that can be referred to by adding a ".<n>", suffix to the substitution tag. ULIST guarantees uniqueness within the list of values, while the LIST operator does not. There are some limitations to the ULIST operator. If the domain from which the ULIST operator picks its value set is smaller than or equal to the size of the requested list (i.e., <number>), the ULIST operator behaves like the LIST operator.

DSQGEN recognizes some keywords and built-in functions as integer constants. Table 1 summarizes some commonly used keyword substitutions and their values. These constants are commonly used to instrument the query stream, provide unique names for temporary table or view definitions, or to access vendor-specific syntax to constrain the size of a result set. For instance, a vendor might need to define a temporary view if they didn't support SQL's common-sub-expression syntax. In order to distinguish the view name between streams, a unique identifier needs to be assigned to it. The _STREAM keyword fulfills this requirement, and can be used in the rewrite of Query 15 of TPC-H. Another example creates a predicate based on the scale factor, which is used in Query 11 of TPC-H. Example 6 prints the number of rows in the part table together with the scale factor.

Example 6
```
SELECT 'part count at scale factor
[_SCALE]'|count(*)
FROM PART;
```

The built-in functions can also be used to access vendor-specific syntax to limit the number of rows returned by a query. Vendors have dialect-specific extensions to SQL that control the number of rows returned by a query, and require those syntactic changes at different points in the query. DSQGEN defines three possible additions (_LIMITA, _LIMITB, and _LIMITC) that, in conjunction with a global limit to the number of rows to be returned (_LIMIT) and vendor specific definitions, allow a single query template to satisfy the requirements of all supported database dialects (currently, ANSI, Oracle, DB2, Netezza, SqlServer). Example 7 shows a sample usage of the LIMIT tags to return the 100 most frequent last names from a customer table. Vendor-specific substitutions (__LIMITA, __LIMITB and __LIMITC) are defined to limit the number of rows returned by a query (Example 7a). The query template (Example 7b) needs only define the desired number of rows (via _LIMIT) and include the potential syntax substitutions (_LIMITA, _LIMITB, _LIMITC). The result is a single query template that can produce appropriate SQL for all defined dialect, as illustrated for ANSI SQL (Example 7c). The call to generate the query is shown in Example 7d.

Example 7 Implementation of the ANSI specific dialect to limit the number of rows returned by a query

7a: ansi.tpl

```
DEFINE __LIMITA = "";
DEFINE __LIMITB = "top %d";
DEFINE __LIMITC = "";
```

7b: query.tpl

```
DEFINE LIMIT=100;
[_LIMITA]
SELECT [_LIMITB] last_name, count(*) as name_count
FROM customer
GROUP BY name_count, order by name_count
[_LIMITC];
```

7c: query_0.sql

```
SELECT top 100 last_name, count(*) as name_count
FROM customer
GROUP BY name_count, order by name_count;
```

7d: Command line call to DSQEN for generating vendor specific syntax

```
DSQGEN –scale 1 –template query.tpl –dialect ansi.tpl
```

3.2 Generating Query Workloads

DSQGEN is capable of generating three different kinds of workload: Single-Template, Single-Stream, and Multi-Stream. Each type of workload requires a set of query templates to be defined. Each template must be stored in a separate file (e.g. T_1.tpl, T_2.tpl,…,T_n.tpl). While a template can contain more than one SQL statement, there can only be one set of substitution tag declarations for all queries included in a given template, and they must occur before the first SQL statement. Having multiple

SQL queries in one template allows for the implementation of business questions that usually occur in the same sequence, such as drill down queries.

The *Single-Template Workload* generates one or multiple versions of the same query. It can be used to stress test the execution of a single query with multiple substitution parameters. This is especially useful to test the query optimizer's ability to generate the most optimal execution plan for every query that can be generated from one template. The syntax to generate 500 queries for scale factor 100 using query template T1 in the ANSI SQL syntax is:

```
DSQGEN -scale 100 -template T1.tpl
       -count 500 -dialect ansi.tpl
```

For the *Single-Stream Workload,* fully qualified paths to a set of template files are listed in an ASCII meta-file (e.g. MF.txt). This workload generates one query for every template included in the meta-file. The Single-Stream query workload is identical to the workload used in TPC-H's Power-Test.

The *Multi-Stream Workload* simulates n users, each running a unique permutation of the query templates defined in a meta-file. The following command line generates n files, query_0.sql through query_<n-1>.sql, each containing a different permutation of the queries defined in M.tpl according to the vendor-specific dialect defined in dialect.tpl:

```
DSQGEN -input M.tpl -stream <n> -dialect <dialect.tpl>
```

4 Modeling Existing TPC-H Queries with DSQGEN

4.1 Substitution Analysis

The 22 TPC-H queries use the substitution parameters listed in Table 2. After eliminating duplicates, we are able to classify all substitutions into the 10 types as listed in the third column.

The **Type 1** substitution type randomly selects one or more numbers from a dense interval. Most queries use substitutions of integer numbers, a straightforward use of the RANDOM function. Query 16 concatenates two independently selected values in [1..5] to identify a value for P_BRAND (Type 1a). Query 6 requires random floating value between 0.02 and 0.09 to build a selectivity predicate on L_DISCOUNT (Type 1b). Another variant of Type 1 is used in Query 16, which applies an in-list predicate on P_SIZE.

The **Type 2** substitution type randomly selects one or more strings from a list of possible items.

The **Type 3** substitution type randomly selects a date. The desired value may be a random day in a static month and year (Type 3a), a static day of a random month and year (Type 3b), a static day of a random month between the January of a static year and October of a static year (Type 3c), or the first of January of a random year (Type 3d).

The **Type 4** substitution type selects the scale factor of the database being queried.

The **Type 5** substitution type selects the number of rows to be returned by the top most SQL statement.

Table 2. TPC-H parameter substitutions and their characterization into types

Table Column	Substitution Domain	Type	Query
P_BRAND	'Brand#MN' where MN is a two character string representing two numbers randomly and independently selected within [1...5]	1a	16
N/A	Randomly selected within [60 ... 120]	1a	16
C_PHONE	Randomly selected within [11 … 35]	1a	16
L_QUANTITY	Randomly selected within [312 … 315]	1a	16
L_DISCOUNT	Randomly selected within [0.02 ... 0.09]	1b	6
P_SIZE	Randomly selected within [1 … 50]	1a	16
P_SIZE	8 numbers randomly selected within [1 … 50] (no duplicates)	1c	16
P_NAME	Randomly selected from the list of P_NAMEs	2	16
P_CONTAINER	Randomly selected from the list defined for P_CONTAINER	2	16
N_NAME	Randomly selected within the list of N_NAME	2	16
R_NAME	Randomly selected within the list of R_NAME	2	16
C_MKT_ SEGMENT	Randomly selected within the list of Segments	2	16,3
L_SHIPMODE	Randomly selected within the list of values defined for Modes	2	16
P_TYPE	Made of the first 2 syllables of a string randomly selected within the list of 3-syllable strings defined for Types	2	16
P_TYPE	Randomly selected within the list Syllable 3 defined for Types	2	16
O_COMMENT	Randomly selected of “special”, “pending”, “unusual”, “express”	2	16
DATE	Randomly selected day [1995-03-01 ... 1995-03-31].	3a	3
DATE	The first day of a random month of years [1993 ... 1997].	3b	
DATE	The first day of a random month between the first month of 1993 and the 10th month of 1997.	3c	4
DATE	The first of January of a random year within [1993 ... 1997].	3d	6
N/A	Chosen as 0.0001 / SF.	4	11
N/A	Limit the number of rows to <n>	5	3

In the following sections, we will use these query substitution types to translate representative TPC-H query templates into the DSQGEN syntax. The accompanying figures outline the QGEN syntax for a given query include the substitution definitions used for that query, but it is worth noting that the substitution definition is not included in the template in the actual QGEN template. The substitution definitions would only be clear to a user who was able to access and understand the source code of QGEN itself. While this paper does not illustrate the translation of the entire TPC-H query set, the process outlined here can be applied to all queries defined for TPC-H.

4.2 Query 16

Query 16 finds out how many suppliers can supply parts with given attributes. It might be used, for example, to determine whether there are a sufficient number of suppliers for heavily ordered parts. Query 16 is an example that uses the substitution types: 1, 1c and 2 (see Table 2).

```
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 <> ':1'
  AND p_type not like ':2%'
  AND p_size in (:3, :4, :5, :6, :7, :8, :9, :10)
```

Fig. 5. Query 16 of TPC-H in QGEN Syntax

```
  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;
```

Fig. 5. (*continued*)

:1 (p_brand) is substituted as Brand#MN, where M and N are two single character strings representing two numbers randomly and independently selected within [1 .. 5];

:2 (p_type) is made of the first 2 syllables of a string randomly selected within the list of 3-syllable strings “STANDARD", "ANODIZED", "TIN", "SMALL", "BURNISHED", "NICKEL", "MEDIUM", "PLATED", "BRASS", "LARGE", "POLISHED", "STEEL", "ECONOMY", "BRUSHED", "COPPER", "PROMO”

:3 to :10 (p_size) are eight randomly selected as a set of different values of [1...50];

Query 16 can be rewritten in DSQGEN syntax by utilizing the RANDOM and TEXT substitution types and the ULIST operator as follows.

```
DEFINE PBRAND_A = RANDOM(1,5,uniform);
DEFINE PBRAND_B = RANDOM(1,5,uniform);
DEFINE PTYPE = LIST(TEXT({"STANDARD",1},{"ANODIZED",1}
                        ,{"TIN",1},{"SMALL",1}
                        ,{"BURNISHED",1},{"NICKEL",1}
                        ,{"MEDIUM",1},{"PLATED",1}
                        ,{"BRASS",1},{"LARGE",1}
                        ,{"POLISHED",1},{"STEEL",1}
                        ,{"ECONOMY",1},{"BRUSHED",1}
                        {"COPPER",1},{"PROMO″,1}),8);
DEFINE SIZE = ULIST(RANDOM(1,50,uniform),8);

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 <> '[PBRAND_A][PBRAND_B]'
  AND p_type not like '[PTYPE]%'
  AND p_size in ([SIZE.1],[SIZE.2],[SIZE.3],[SIZE.4]
              ,[SIZE.5],[SIZE.6],[SIZE.7],[SIZE.8])
  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;
```

Fig. 6. Query 16 of TPC-H in DSQGEN Syntax

The substitution parameter for P_BRAND is essentially a two-digit number, each digit from the domain of [1,5]. It can be constructed from the two independent substitution tags PBRAND_A and PBRAND_B, each defined with the RANDOM substitution type as an integer between 1 and 5. The substitution parameter for P_TYPE is a random string from a list of 16 elements (see above). It can be implemented as a TEXT substitution of 16 elements, each with the same weight. P_SIZE requires 8 substitution parameters, each from the domain of [1,50]. Additionally the set of 8 parameters has to be unique. Hence, we implement the substitution parameter using a combination of the RANDOM substitution and the ULIST operator.

4.3 Query 6

Query 6 quantifies the amount of revenue increase for a given year that would have resulted from eliminating discounts. Query 6 is an example that uses the substitution types: 1b and 3d (see Table 2).

```
SELECT sum(l_extendedprice * l_discount) as revenue
FROM   lineitem
WHERE l_shipdate>= date ':1'
  AND l_shipdate<add_months(date':1'+ interval '1' year
  AND l_discount between :2 - 0.01 and :2 + 0.01
  AND l_quantity < :3;
```

Fig. 7. Query 6 of TPC-H with QGEN Syntax

:1 DATE is the first of January of a randomly selected year within [1993 .. 1997];
:2 DISCOUNT is randomly selected within [0.02 .. 0.09];
:3 QUANTITY is randomly selected within [24 .. 25].

Query 6 can be implemented solely with the RANDOM substitution type. The substitution tag on L_SHIPDATE, S_YEAR is implemented as a random number between 1993 and 1997. The month and day portion of the date are statically set to 01. The substitution tag for L_DISCOUNT requires a fraction [0.02,0.09]. Since the RANDOM substitution type only allows for integer values, we use a random number tag between 2 and 9 and build the fraction by prefixing the number with "0.0". The substitution tag for L_QUANTITY is a simple random substitution of [24,25].

```
DEFINE SYEAR     = random(1993,1997,normal);
DEFINE DF        = random(2,9,normal);
DEFINE LQUANTITY = random(24,25,normal);
SELECT sum(l_extendedprice * l_discount) as revenue
FROM  lineitem
WHERE l_shipdate>= date'[SYEAR]-01-01'
  AND l_shipdate< date'[SYEAR]-01-01'+interval '1' year
  AND l_discount between 0.0[DF]-0.01 and 0.0[DF]+0.01
  AND l_quantity < [LQUANTITY];
```

Fig. 8. Query 6 of TPC-H with QGEN Syntax

4.4 Query 3

Query 3 retrieves the ten unshipped orders with the highest value. It is an example that uses the substitution types: 2, 3a and 5 (see Table 2).

```
SELECT l_orderkey
      ,sum(l_extendedprice*(1-l_discount)) as revenue
      ,o_orderdate ,o_shippriority
FROM customer ,orders, lineitem
WHERE c_mktsegment = ':1'
  AND c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND o_orderdate < date ':2'
  AND l_shipdate > date ':2'
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY revenue desc, o_orderdate;
:n
```

Fig. 9. Query 3 in TPC-H qgen sytax

:1 is randomly selected within the list of values defined for Segments
:2 is a randomly selected day within [1995-03-01 .. 1995-03-31].
:**n** defines the maximum number of rows returned by the query (top)

Query 3 uses the TEXT substitution type, the RANDOM substitution type and the build-in functions to limit the number of rows returned by the query. As in the P_TYPE substitution of Query 16, this query implements the substitution parameter for C_MKTSEGMENT using the TEXT substitution type with a four item list, each with the same likelihood. The substitution parameters O_ORDERDATE and L_SHIPDATE are implemented with the RANDOM substitution. Since both

```
DEFINE SEGMENT=text({"AUTOMOBILE",1},{"BUILDING",1}
                   ,{"FURNITURE",1},{"MACHINERY",1}
                   ,{"HOUSEHOLD"});
DEFINE SHIPDAY = random(1,31,uniform);
DEFINE _LIMIT=10;
[_LIMITA] select [_LIMITB] l_orderkey
      ,sum(l_extendedprice*(1-l_discount)) as revenue
      ,o_orderdate, o_shippriority
FROM customer, orders, lineitem
WHERE c_mktsegment = '[SEGMENT]'
  AND c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND o_orderdate < date '1995-03-[SHIPDAY]'
  AND l_shipdate > date '1995-03-[SHIPDAY]'
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY revenue desc, o_orderdate
[_LIMITC];
```

Fig. 10. Query 3 in TPC-H DSQGEN syntax

substitution parameters are the same they can be implemented with the same substitution tag, SHIPDAY, which picks a day between 1 and 31 prefixed with the static string "'1995-03-". This query also needs to limit the number of rows to be returned to ten. This is done with three substitution tags, _LIMITA, _LIMITB and _LIMITC. _LIMITA, _LIMITB and _LIMITC are defined in the vendor specific template _LIMIT is defined as 10.

4.5 Query 4

Query 4 determines how well the order priority system is working and gives an assessment of customer satisfaction. It is an example using the substitution type 3c.

```
SELECT o_orderpriority, count(*) as order_count
FROM orders
WHERE o_orderdate >= date ':1'
  AND o_orderdate < date ':1' + 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;
```

Fig. 11. Query 4 in TPC-H QGEN syntax

:1 is the first day of a randomly selected month between the first month of 1993 and the 10th month of 1997.

Query 4 uses the RANDOM substitution type in combination with the build-in arithmetic capability of DSQGEN. There are 58 months between January 1993 and October 1997. In order to choose a random month between those dates, we first generate a random number between 0 and 58 (SEQMO). Then we divide that number by 12 to generate the year (YR). Please note that the result of the division is an integer. In order to generate the months, we take that number modulo 12 (MO). In the query we build the date by concatenating these numbers into: [YR]-[MO]-01

```
DEFINE SEQMO = random(0,57,uniform);
DEFINE YR    = ([SEQMO] / 12) + 1;
DEFINE MO    = ([SEQMO] % 12) + 1;
SELECT o_orderpriority,
       count(*) as order_count
FROM orders
WHERE o_orderdate>=date'[YR]-[MO]-01'
  AND o_orderdate<date'[YR]-[MO]-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;
```

Fig. 12. Query 4 in TPC-H DSQGEN syntax

5 Scope of Possible Expansions to the TPC-H Query Set

Section 4, identified the substitution types that are found in the TPC-H query set. We have also shown that DSQGEN's current functionality is sufficient to generate queries for all 22 TPC-H query templates. It is also possible to extend the TPC-H query set very elegantly using DSQGEN, well beyond the identified substitution types. Since DSQGEN uses textual substitutions, we are able to introduce aggregation substitutions, column substitutions and full date substitutions. In the following sections we will illustrate how new queries can be introduced into TPC-H, creating a new query using the existing substitution types, followed by examples using new substitution types: aggregation substitution, and column substitution. Please note, we are not proposing new queries to TPC-H, but merely illustrating how new queries could be added to TPC-H's query set without any modifications to the query generator.

5.1 Query Using Existing Substitution Types

The following query retrieves unshipped orders with the highest value for customers with specific account balances and located in specific nations. This query uses the SQL ROLLUP operator, grouping by any combination of customer name, customer nation, order date and ship priority. The query uses three substitutions. The ABAL substitution tag, used in a *between predicate*, is defined using the RANDOM

```
DEFINE ABAL=random(0,9000,uniform);
DEFINE NT=text({"ALGERIA",1),{"ARGENTINA",1},{"IRAQ",1}
              ,{"BRAZIL",1},{"CANADA",1},{"RUSSIA",1}
              ,{"ETHIOPIA",1},{"FRANCE",1},{"INDIA",1}
              ,{"GERMANY",1},{"JORDAN",1},{"KENYA",1}
              ,{"INDONESIA",1},{"IRAN",1},{"EGYPT",1}
              ,{"JAPAN",1},{"MOROCCO",1},{"ROMANIA",1}
              ,{"MOZAMBIQUE",1},{"PERU",1},{"CHINA",1}
              ,{"ROMANIA",1},{"SAUDI ARABIA",1}
              ,{"VIETNAM",1},{"UNITED KINGDOM",1});
DEFINE _LIMIT=10;
[_LIMITA] select [_LIMITB] c_name, c_nation
      ,sum(l_extendedprice*(1-l_discount)) as revenue
      ,o_orderdate, o_shippriority
FROM customer, orders, lineitem, nation
WHERE c_acctbal between [ABAL]-999.99 and [ABAL]
  AND c_nationkey = n_nationkey
  AND c_custkey = o_custkey
  AND n_name = '[NT]'
  AND l_orderkey = o_orderkey
GROUP BY ROLLUP (c_name, c_nation
                ,o_orderdate, o_shippriority)
ORDER BY revenue desc, o_orderdate, c_name, c_nation
[_LIMITC];
```

Fig. 13. Query Using Existing Substitution Types

substitution type. It chooses the upper boundary of the account balance from the interval [0..9000]. The lower boundary of the between predicate is computed by subtracting 999.99 from ABAL. The second tag (NT) is used to choose a nation from a text list rather than the nation key. The last tag, _LIMIT, caps the number of rows returned to 10.

5.2 Query Using Aggregate Substitutions

The following query is based on Query 11 of TPC-H. It lists the most important subset of suppliers' stock in a given nation. In this context "importance" is based on the total, largest or smallest stocking cost. It uses the random substitution type for the NK substitution tag to implement a predicate on nation key. It uses the text substitution type to implement the AGG substitution tag, which chooses between the aggregation functions sum, min and max for calculating a supplier's stock.

```
DEFINE NK = random (0,31, uniform);
DEFINE AGG= text({"sum",1},{"min",1},{"max",1});

SELECT ps_partkey
      ,[AGG](ps_supplycost * ps_availqty) as value
FROM partsupp,supplier
WHERE ps_suppkey = s_suppkey
  AND s_nationkey = [NK]
GROUP BY ps_partkey;
```

Fig. 14. Query using aggregate substitution

5.3 Query Column Substitutions and Full Date Substitution

The final example query is based on Query 3 from TPC-H. It employs column substitution to randomly select the target of an aggregation from a set of statistically equivalent columns. The resulting queries generated by DSQGEN would exercise similar selectivity and computational load, but would increase the cost and complexity of maintaining summary tables or other auxiliary data structure, by increasing the randomness of the eventual SQL.

```
DEFINE SHIPDATE = random(1,31,uniform);
DEFINE LIMIT=10;
DEFINE COL=text({"l_quantity",1},{"l_discount",1}
                ,{"l_extendedprice",1},{"l_tax",1});

[_LIMITA] select [_LIMITB] l_orderkey
      ,sum([COL]), o_orderdate, o_shippriority
FROM customer, orders, lineitem
WHERE c_custkey = o_custkey
  AND l_orderkey = o_orderkey
  AND o_orderdate < date '1995-03-[SHIPDAY]'
  AND l_shipdate > date '1995-03-[SHIPDAY]'
GROUP BY l_orderkey, o_orderdate, o_shippriority
ORDER BY [COL] desc, o_orderdate
[_LIMITC];
```

Fig. 15. Query using Column Substitutions

6 Conclusion

This paper has demonstrated how the enhanced syntax available with the query generator developed for the proposed TPCDS benchmark, DSQGEN, can be used to express the queries defined for TPC-H, which currently uses the older, simpler query generator, QGEN. The migration from one query dialect to the other has no impact on the syntactic formulation of the queries, the selectivity of their predicates, the work they present to the system under test or the answer sets that will be returned. As such, the migration from the old query dialect to the new dialect can be accomplished without any impact on the viability or comparability of existing TPC-H results.

At the same time, moving the TPC-H query set from the existing syntax to that provided by DSQGEN presents the TPC with a two-fold opportunity that could enrich the existing benchmark and extend its useful life. The rephrased queries would reduce the support burden borne by the TPC, since the query templates could be revised or corrected without the need to fund additional software development. The migration would also provide the TPC with the opportunity to explore, and potentially adopt, additional queries that broaden the scope of the functions tested by the TPC's only decision support benchmark, expand the relevance of the workload to modern decision support customers, and increase the relevance of TPC-H results to customers faced with the complex and costly process of selecting a decision support solution, whether in hardware or software.

References

1. Stephens Jr., J.M., Poess, M.: MUDD: a multi-dimensional data generator. In: WOSP 2004, pp. 104–109 (2004)
2. Poess, M., Stephens Jr., J.M.: Generating Thousand Benchmark Queries in Seconds. In: VLDB 2004, pp. 1045–1053 (2004)
3. Poess, M.: Controlled SQL query evolution for decision support benchmarks. In: WOSP 2007, pp. 38–41 (2007)
4. Poess, M., Floyd, C.: New TPC Benchmarks for Decision Support and Web Commerce. ACM SIGMOD RECORD 29(4) (2000)
5. TPC-D Version 2.1: http://www.tpc.org/tpcd/default.asp
6. TPC-H specification 2.8.0, http://www.tpc.org/tpch/spec/tpch2.8.0.pdf
7. Transaction Processing Performance Council Policies Version 5.17, http://www.tpc.org/information/about/documentation/spec/TPC_Policies_v5.17.pdf