BigBench Specification V0.1

BigBench: An Industry Standard Benchmark for Big Data Analytics

Tilmann Rabl[1], Ahmad Ghazal[2], Minqing Hu[2], Alain Crolotte[2], Francois Raab[3], Meikel Poess[4], Hans-Arno Jacobsen[1]

[1] University of Toronto
[2] Teradata Corp.
[3] InfoSizing Inc.
[4] Oracle Corp.

Abstract. In this article, we present the specification of BigBench, an end-to-end big data benchmark proposal. BigBench models a retail product supplier. The benchmark proposal covers a data model and a set of big data specific queries. BigBench's synthetic data generator addresses the variety, velocity and volume aspects of big data workloads. The structured part of the BigBench data model is adopted from the TPC-DS benchmark. In addition, the structured schema is enriched with semi-structured and unstructured data components that are common in a retail product supplier environment. This specification contains the full query set as well as the data model.

1 Introduction

Big data (BD) is about increasing volume of data from a variety of sources including structured, semi-structured and unstructured data. Some of the BD sources are typically generated with high velocity like click streams and sensors logs. This wealth of data provides a lot of new analytic and business intelligence (BI) opportunities like fraud, churn and customer loyalty analysis.

Many commercial and open source systems were built or extended to store and process BD. These tools are mostly parallel database management systems or MapReduce (MR) based systems. There are no standards yet on BD processing, but for the most part these systems provide SQL, UDF, MR or a mix of these as an interface.

Even though there are no standards for BD yet, still there is a need to measure and compare the performance of the different systems that claim to support BD. Recently, there are quite a few efforts in the area of big data benchmarking (e.g. PigMix[5], GridMix[6], GraySort [7]). Some of these benchmarks are focused on one component of the system and others are focused on specific MR systems.

[5] PigMix – `https://cwiki.apache.org/confluence/display/PIG/PigMix`
[6] GridMix – `http://hadoop.apache.org/docs/mapreduce/current/gridmix.html`
[7] Sort Benchmark Home Page – `http://sortbenchmark.org`

In this article, we present the specification of the end to end big data benchmark BigBench. BigBench is based on a fictitious retailer who sells products to customers via physical and online stores. This specification completes our previous publication that covered details on the data model, synthetic data generator, workload description, and metrics [1]. The workload queries are specified in English and in Teradata Aster's SQL-MR syntax [2, 3]. We introduce new metrics specific to BD data loading and workload execution. The feasibility of the proposal is shown by applying it on the Teradata Aster DBMS (TAD). This experiment involves generating 200 gigabyte of data and loading it into TAD. The workload is executed as a single stream of queries.

The rest of this article is structured as follows. In Section 2, we describe the BigBench data model. In Section 3, we give a short overview of the BigBench data generation. We describe the BigBench workload in Section 4. Section 5 shows the results of our proof-of-concept evaluation of BigBench on Teradata Aster. We conclude in Section 6. In Appendix A, we list all 30 BigBench queries and Appendix B contains the complete schema for BigBench.

2 Data Model

BD is not about volume only. Douglas Laney described the 3 Vs of BD referring to volume, velocity and variety [4]. Velocity is an important issue in BD since such data like clicks or sensor information are produced at an increasing rate. Also, data comes in different forms like structured relational tables, semi-structured key-value web clicks or unstructured social data text. Our data model has the volume, variety and velocity elements as described in the following.

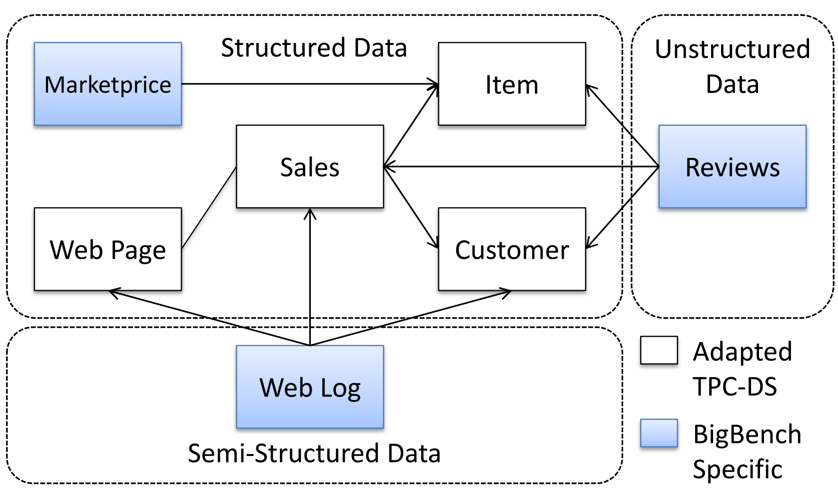

Fig. 1. Simplified BigBench Data Model

The variety property of our model is illustrated in Figure 1. The structured part of BigBench is an adaption of the TPC-DS model which also depicts a product retailer [5]. We borrowed the store and online sales part from that model and added a table for competitor prices of the retailer.

The structured part is enriched with semi-structured and unstructured data shown in the lower and right part of Figure 1. The semi-structured part's content is composed by clicks made by customers and guest users visiting the retailer site. Some of these clicks are for completing a customer order. As shown in Figure 1, the semi-structured data is logically related to the Web Page, Customer and Sales tables in the structured part. Our design assumes the semi-structured data to be a key-value format similar to Apache web server log format.

Typically, database and MR systems would convert such format to a table/file with a schema like (DateID, TimeID, SalesID, WebPageID, UserID). However, we do not require such conversion since some systems may choose to run analytics on the native key-value format. Product reviews is a growing source of online retail data. We found such source to be an excellent representation for the unstructured data in our model. Figure 1 shows product reviews in the right part and its relationship to Date, Time, Item, Users and Sales tables in the structured part. One implementation of the product reviews is a single table/file with a structure like (DateID, TimeID, SalesID, ItemID, ReviewRating, ReviewText). The full schema is specified in SQL in Apendix B.

3 Data Generation

Our work also provides a design and implementation of a data generator for the proposed BigBench data model. Our data generator is based on an extension of PDGF [6]. PDGF is a parallel data generator that is capable of producing large amounts of data for an arbitrary schema. The existing PDGF can be used to generate the structured part of the BigBench model. However, it is not capable of generating the unstructured product reviews text. First, PDGF is enhanced to produce a key-value data set for a fixed set of required and optional keys. This is sufficient to generate the weblogs part of BigBench.

The main challenge in product reviews is producing the unstructured text. We developed and implemented an algorithm that produces synthetic text based on sample input text. The algorithm uses a Markov Chain technique that extracts key words and builds a dictionary based on these key words. The new algorithm is applied for our use case by using some real product reviews from an online retailer for the initial sample data. PDGF interacts with the review generator through an API sending a product category as input and receiving a product review text for that category.

The volume dimension of our model is far simpler than the variety discussion and previous data generators had a good handle on that. PDGF handles the volume well since it can scale the size of the data based on a scale factor. It also runs efficiently for large scale factors since it runs in parallel and can leverage large systems dedicated for the benchmark.

For our proof-of-concept system, the tables that are originating from TPC-DS are generated using DSdgen, the TPC-DS standard data generator[8].

4 Workload

The second major component of BigBench is the specification of workload queries applied on the BigBench data model. In terms of business questions, we found the big data retail analytics by McKinsey serves our purpose given that BigBench is about retail [7]. In [7] five major areas of big data analytics are described namely: marketing, merchandising, operations, supply chain and new business models. These areas are further broken down into sub-functions. For example, marketing can be broken down into cross selling, sentiment analysis, etc. We used these 5 areas and added reporting as a sixth area. We postulate that a big data benchmark should have some traditional business intelligence or reporting type of queries.

In addition to the big data retail business levers above, we looked at the different technical aspects the BigBench queries should measure. We identified the following three areas:

- The type of the input data the query is addressing. We made sure each of the structured, semi-structured, unstructured and their combinations are covered in the queries. Out of the 30 queries 18 (60%) are exclusively on the structured data, 7 (23.3%) incorporated semi-structured data, and 5 (16.7%) additionally incorporated unstructured data.
- The type of processing appropriate for the query. This dimension targets the two common paradigms of SQL (and similar constructs like HQL) and MR. Thus, our queries can be answered by SQL, others by MR or a mix of both. Note that some of the perceived MR queries can also be written through complex SQL constructs like window functions and therefore we identify the two classes in this dimension as simple SQL and MR or complex SQL. In this definition, 12 (40%) queries are declarative (pure SQL), 5 (16.7%) queries are procedural (MR), and 13 (43.3%) are a mix.
- The third important technical dimension is the different algorithms of MR processing as described by the Apache MAHOUT system. Classes of algorithms used in the BigBench queries are statistical analysis (6 queries), path analysis (5 queries), text analysis (4 queries), association mining (4 queries), classification (1 query), clustering (3 queries), reporting (8 queries).

The categorization along technical dimensions with corresponding query numbers is shown in Table 4. The implementation technique is either declarative, procedural, or mixed. Declarative queries are pure SQL queries, that could also be processed by stock relational database systems. Procedural queries are pure MapReduce implementations that do not need joins. Mixed queries contain MapReduce functions along with relational operations, such as joins or views.

[8] TPC-DS and DSDgen is available at `http://www.tpc.org/tpcds/default.asp`

Query Type	Queries	Percent	Data Type	Queries	Percent
Declarative	6, 7, 9, 13, 14, 16, 17, 19, 21, 22, 23, 24	40%	Structured	1, 6, 7, 9, 13, 14, 15, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 29	60%
Mixed	1, 4, 5, 8, 11, 12, 15, 18, 20, 25, 26, 29, 30	43%	Semi-Structured	2, 3, 4, 5, 8, 12, 30	23%
Procedural	2, 3, 10, 27, 28	17%	Unstructured	10, 11, 18, 27, 28	17%

The queries were specified to cover the areas of big data analytics as well as the technical dimensions of big data processing. Below is an overview of the business functions as proposed by McKinsey [7] and the associated BigBench queries:

Marketing 18.6%
- **Cross-selling** 1,2,3,29,30
- **Customer micro-segmentation** 4,5,6,9,25,26
- **Sentiment analysis** 8,10,11,18,28
- **Enhancing multichannel consumer experiences** 12,13

Merchandising 16.7%
- **Assortment optimization** 14,21,27
- **Pricing optimization** 16,17

Operations 13.3%
- **Performance transparency** 7,15
- **Customer return analysis** 19,20

Supply chain 6.7%
- **Inventory management** 22,23

New business models 3.3%
- **Price comparison** 24

In Appendix A, we list all 30 BigBench queries. It has to be noted that some of the queries are identical to TPC-DS queries, this is true for the SQL code as well as the English description. For those queries, we list the original template number in brackets in the description below.

5 Evaluation

We chose to initially run BigBench on the Teradata Aster DBMS. TAD has all features needed to store and process big data. Data can be stored as tables and queries can be executed using the SQL-MR interface that extends declarative SQL with MR processing.

TAD is based on the nCluster technology. nCluster is a shared-nothing parallel database, optimized for data warehousing and analytic workloads [2]. nCluster manages a cluster of commodity server nodes, and is designed to scale out to hundreds of nodes and scale up to petabytes of active data.

The test was executed on a 8 node Teradata Aster appliance. Each node is a Dell server with two quad-core Xeon 5500 at 3.07Ghz and hardware RAID 1 with 8 2.5" drives.

For the test, DSdgen is used to produce the selected TPC-DS tables included in our data model. We used PDGF to generate the additional parts of the data model. The new parts produced by PDGF include the Item_marketprice table, an Apache-style web server log, and the XML configuration for the online review generator. PDGF is also configured to generate references (PK-FK relationships) in the new data that matches the TPC-DS data. In the future, we plan on extending PDGF to handle the generation of TPC-DS tables without the need for DSdgen.

The data was loaded into TAD as tables. The web logs were parsed and converted to a table similar to the structure shown in Appendix B. Product reviews are also interpreted as a table assuming the review text as a VARCHAR(5000).

As a proof of concept, we executed the workload as a single stream without velocity on a ca. 130 GB data set. This corresponds to a scale factor 100 in TPC-DS. Since we adapt the velocity methodology from TPC-DS, it and can easily be implemented with a simple driver that periodically adds data to the system and re-submits a new stream of queries. Furthermore, the addition of concurrent query streams can be handled similarly to benchmarks such as TPC-H.

The query processing times for the individual queries can be seen below.

Query	run-time (sec)	Query	run-time (sec)
A1	200	A16	8700.045
A2	12.529	A17	146.879
A3	19.948	A18	1507.33
A4	33.345	A19	11.368
A5	9.462	A20	345
A6	11.652	A21	109.817
A7	1.176	A22	114.555
A8	12.581	A23	1113.373
A9	8.698	A24	11.714
A10	24.847	A25	254.474
A11	2713.042	A26	2708.261
A12	918.575	A27	4.617
A13	1572	A28	381.005
A14	7.952	A29	7.201
A15	41.747	A30	6208

6 Conclusion

In summary, we present the first end-to-end benchmark for big data analytics. While previous work focused on one type of data or processing, we produced

30 queries that address all the three technical dimensions described above. The queries cover all the six major business areas of DB analytics mentioned earlier. We developed and implemented a novel technique for producing unstructured text data and integrated it with traditional structured data generators. We conducted a proof of concept of the proposal by executing it on the Teradata Aster DBMS.

Currently, all queries are translated to the Hadoop eco-system. The complete data generator will be migrated to PDGF, which will make it possible to generate more complex dependencies consistently across the different parts of the schema. This will add correlations that are desirable for exercising analytical queries. Although, basic metrics were specified in [1], we will extend this part of the specification with additional approaches, directly targeting big data related questions.

References

1. Ghazal, A., Rabl, T., Hu, M., Raab, F., Poess, M., Crolotte, A., Jacobsen., H.A.: BigBench: Towards an industry standard benchmark for big data analytics. In: Proceedings of the ACM SIGMOD Conference. (2013)
2. Friedman, E., Pawlowski, P., Cieslewicz, J.: SQL/MapReduce: A Practical Approach to Self-Describing, Polymorphic, and Parallelizable User-Defined Functions. PVLDB **2**(2) (2009) 1402–1413
3. Teradata Aster: Teradata Aster Big Analytics Appliance 3H - Analytics Foundation User Guide. Release 5.0.1 edn. (2012) `http://www.info.teradata.com/edownload.cfm?itemid=123060004`.
4. Laney, D.: 3D Data Management: Controlling Data Volume, Velocity and Variety. Technical report, Meta Group (2001)
5. Nambiar, R.O., Poess, M.: The Making of TPC-DS. In: VLDB. (2006) 1049–1058
6. Rabl, T., Frank, M., Sergieh, H.M., Kosch, H.: A Data Generator for Cloud-Scale Benchmarking. In: TPCTC. (2010) 41–56
7. Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Byers, A.H.: Big data: The Next Frontier for Innovation, Competition, and Productivity. Technical report, McKinsey Global Institute (2011) `http://www.mckinsey.com/insights/mgi/research/technology_and_innovation/big_data_the_next_frontier_for_innovation`.

A BigBench Queries

Below all 30 queries of the BigBench proposal are shown. The queries are specified in English, to give a high-level understanding what the business question of each query is. Additionally, an SQL-MR syntax-based description is given [2, 3].

Query 1 Find products are sold together frequently in given stores. Only products in certain categories sold in specific stores are considered, and "sold together frequently" means at least 50 customers bought these products together in a transaction.

```sql
SELECT pid1 AS item1, pid2 AS item2, COUNT(*) AS cnt
  FROM basket_generator (ON
         (SELECT s.ss_ticket_number AS oid, s.ss_item_sk AS pid
            FROM store_sales100 s
           INNER JOIN item100 i ON s.ss_item_sk = i_item_sk
           WHERE i.i_category_id in (1,4,6) and s.ss_store_sk = 10
          PARTITION BY oid
          basket_size(2)
          basket_item('pid')
          item_set_max(500)
         )
 GROUP BY 1,2
HAVING COUNT(pid1) > 49
 ORDER BY 1,3,2;
```

Listing 1.1. Query 1

Query 2 Find the top 30 products that are mostly viewed together with a given product in online store. Note that the order of products viewed does not matter.

```sql
SELECT pid1 AS item1, pid2 AS item2, COUNT(1) AS cnt
  FROM basket_generator (ON
         (SELECT wcs_user_sk AS cid, wcs_item_sk AS pid
            FROM web_clickstreams
           WHERE wcs_item_sk IS NOT NULL
             AND wcs_user_sk IS NOT NULL
         )
         PARTITION BY cid
         basket_size(2)
         basket_item('pid')
         item_set_max(500)
       )
 WHERE pid1 IN (1416,9082,1547)
 GROUP BY 1,2
 ORDER BY 1,3,2
 LIMIT 30;
```

Listing 1.2. Query 2

Query 3 Find the last 5 products that are mostly viewed before a given product was purchased online. Only products in certain categories and viewed within 10 days before the purchase date are considered.

```sql
SELECT lastviewed_item, purchased_item, COUNT(*)
  FROM nPath (ON web_clickstreams
         PARTITION BY wcs_user_sk
             ORDER BY wcs_click_date_sk, wcs_click_time_sk
              MODE ('NONOVERLAPPING')
           PATTERN ('A+.B')
           SYMBOLS (true AS A, wcs_sales_sk IS NOT NULL AS B)
            RESULT (
                LAST (wcs_item_sk OF A) AS lastviewed_item,
                LAST (wcs_click_date_sk OF A) AS lastviewed_date,
               FIRST (wcs_item_sk OF B) AS purchased_item,
               FIRST (wcs_click_date_sk OF B) AS purchased_date
            )
       )
 WHERE purchased_item = 16891
   AND purchased_date - lastviewed_date < 11
 GROUP BY 1,2;
```

Listing 1.3. Query 3

Query 4 Shopping cart abandonment analysis: For users who added products in their shopping carts but did not check out in the online store, find the average number of pages they visited during their sessions.

```
DROP VIEW sessions;

CREATE VIEW sessions AS (
  SELECT *
    FROM sessionize (ON
         (SELECT c.wcs_user_sk as uid, c.wcs_item_sk as item,
                 w.wp_type as wptype,
                 d.d_date + t.t_time*INTERVAL '1␣second' as tstamp
            FROM web_clickstreams c, web_page w, date_dim d, time_dim t
           WHERE c.wcs_web_page_sk = w.wp_web_page_sk
             AND c.wcs_click_date_sk = d.d_date_sk
             AND c.wcs_click_time_sk = t.t_time_sk
             AND c.wcs_user_sk IS NOT NULL
         ) AS clicks
         PARTITION BY uid
             ORDER BY tstamp
           timecolumn ('tstamp')
              timeout ('300')
       )
   ORDER BY uid, tstamp
);

DROP VIEW cart_abadon;
CREATE VIEW cart_abadon AS (
  SELECT *
    FROM nPath(ON sessions
       PARTITON BY sessionid
          ORDER BY tstamp
           MODE ('NONOVERLAPPING')
        PATTERN ('C*.A.B*$')
        SYMBOLS (wptype = 'dynamic' AS A, true as C, wptype <> 'order' AS B)
         RESULT (FIRST_NOTNULL (sessionid OF C) AS sid,
                 LAST_NOTNULL (tstamp OF B) AS end_s,
                 FIRST_NOTNULL (tstamp OF C) AS start_s
         )
    )
);

SELECT c.sid, COUNT(*) AS s_pages
  FROM cart_abadon c, sessions s
 WHERE s.sessionid = c.sid
 GROUP BY 1;
```

Listing 1.4. Query 4

Query 5 Build a model using logistic regression: based on existing users online activities and demographics, for a visitor to an online store, predict the visitors likelihood to be interested in a given category.

```
DROP VIEW logstic_reg_t;
CREATE VIEW logstic_reg_t AS (
  SELECT c_customer_sk, college_education, male,
         CASE WHEN clicks_in_category > 2 THEN true ELSE false END AS label
    FROM (
       SELECT c_customer_sk,
              CASE WHEN (cd_education_status = 'Advanced␣Degree'
                     OR cd_education_status = 'College'
                     OR cd_education_status = '4␣yr␣Degree'
                     OR cd_education_status = '2␣yr␣Degree␣')
                   THEN TRUE ELSE FALSE END AS college_education,
              CASE WHEN cd_gender = 'M' THEN TRUE ELSE FALSE END AS male,
```

```
              SUM (CASE WHEN i_category='Books' THEN 1 ELSE 0 END) AS
                   clicks_in_category
         FROM customer, customer_demographics, item, web_clickstreams
        WHERE wcs_user_sk = c_customer_sk
          AND c_current_cdemo_sk = cd_demo_sk
          AND wcs_item_sk = i_item_sk
        GROUP BY 1,2,3) C
);

DROP TABLE books_interests;
SELECT *
  FROM log_regression (
    ON (SELECT 1)
    PARTITION BY 1
    DATABASE('benchmark')
    USERID('benchmark')
    PASSWORD('benchmark')
    INPUTTABLE('logstic_reg_t')
    OUTPUTTABLE('books_interests')
    COLUMNNAMES('label','college_education','male')
  );
```

Listing 1.5. Query 5

Query 6 (TPC-DS 4) Find customers who spend more money via web than in stores for a given year. Report customers first name, last name, their country of origin and identify if they are preferred customer.

```
BEGIN;
DROP TABLE IF EXISTS q04_year_total_8;

CREATE TEMP TABLE q04_year_total_8 (
  customer_id            VARCHAR(16),
  customer_first_name    CHAR(20),
  customer_last_name     CHAR(30),
  c_preferred_cust_flag  CHAR(1),
  c_birth_country        VARCHAR(20),
  c_login                CHAR(13),
  c_email_address        CHAR(50),
  dyear                  INTEGER,
  year_total             DECIMAL(15,2),
  sale_type              VARCHAR(2)
  ) DISTRIBUTE BY HASH (customer_id) AS (
    SELECT c_customer_id::VARCHAR AS customer_id,
           c_first_name           AS customer_first_name,
           c_last_name            AS customer_last_name,
           c_preferred_cust_flag,
           c_birth_country,
           c_login,
           c_email_address,
           sv.d_year              AS dyear,
           sv.year_total          AS year_total,
           's'::VARCHAR           AS sale_type
      FROM customer,
           (SELECT ss.ss_customer_sk AS customer_sk,
                   dt.d_year         AS d_year,
                   SUM(((ss_ext_list_price - ss_ext_wholesale_cost
                       - ss_ext_discount_amt) + ss_ext_sales_price) / 2)
                                     AS year_total
              FROM store_sales ss, date_dim dt
             WHERE ss.ss_sold_date_sk = dt.d_date_sk
             GROUP BY ss.ss_customer_sk, dt.d_year) sv
     WHERE c_customer_sk = sv.customer_sk
    UNION ALL
    SELECT c_customer_id::VARCHAR AS customer_id,
           c_first_name           AS customer_first_name,
```

```
           c_last_name             AS customer_last_name,
           c_preferred_cust_flag,
           c_birth_country,
           c_login,
           c_email_address,
           cv.d_year               AS dyear,
           cv.year_total           AS year_total,
           'c'::VARCHAR            AS sale_type
      FROM customer,
           (SELECT ws.ws_bill_customer_sk AS customer_sk
                   dt.d_year              AS d_year
                   SUM(((ws_ext_list_price - ws_ext_wholesale_cost
                       - ws_ext_discount_amt) + ws_ext_sales_price) / 2)
                                   AS year_total
              FROM web_sales ws,
                   date_dim dt
             WHERE ws.ws_sold_date_sk = dt.d_date_sk
             GROUP BY ws.ws_bill_customer_sk, dt.d_year) cv
     WHERE c_customer_sk = cv.customer_sk);

ANALYZE q04_year_total_8;

SELECT t_s_secyear.customer_id,
       t_s_secyear.customer_first_name,
       t_s_secyear.customer_last_name,
       t_s_secyear.c_preferred_cust_flag,
       t_s_secyear.c_birth_country,
       t_s_secyear.c_login
  FROM q04_year_total_8 t_s_firstyear,
       q04_year_total_8 t_s_secyear,
       q04_year_total_8 t_c_firstyear,
       q04_year_total_8 t_c_secyear
 WHERE t_s_secyear.customer_id = t_s_firstyear.customer_id
   AND t_s_firstyear.customer_id = t_c_secyear.customer_id
   AND t_s_firstyear.customer_id = t_c_firstyear.customer_id
   AND t_s_firstyear.sale_type   = 's'
   AND t_c_firstyear.sale_type   = 'c'
   AND t_s_secyear.sale_type     = 's'
   AND t_c_secyear.sale_type     = 'c'
   AND t_s_firstyear.dyear       = 1999
   AND t_s_secyear.dyear         = 1999 + 1
   AND t_c_firstyear.dyear       = 1999
   AND t_c_secyear.dyear         = 1999 + 1
   AND t_s_firstyear.year_total  > 0
   AND t_c_firstyear.year_total  > 0
   AND CASE WHEN t_c_firstyear.year_total > 0
            THEN t_c_secyear.year_total / t_c_firstyear.year_total
            ELSE NULL END >
       CASE WHEN t_s_firstyear.year_total > 0
            THEN t_s_secyear.year_total / t_s_firstyear.year_total
            ELSE NULL END
 ORDER BY t_s_secyear.customer_id,
          t_s_secyear.customer_first_name,
          t_s_secyear.customer_last_name,
          t_s_secyear.c_preferred_cust_flag,
          t_s_secyear.c_birth_country,
          t_s_secyear.c_login
 LIMIT 100;

DROP TABLE IF EXISTS q04_year_total_8;
END;
```

Listing 1.6. Query 6

Query 7 (TPC-DS 6) List all the stores with at least 10 customers who during a given month bought products with the price tag at least 20% higher than the average price of products in the same category.

```
BEGIN;
DROP TABLE IF EXISTS q06_specific_month_88;
DROP TABLE IF EXISTS q06_cat_avg_price_88;

CREATE DIMENSION TABLE q06_specific_month_88 AS
  SELECT DISTINCT (d_month_seq) AS d_month_seq
    FROM date_dim
   WHERE d_year = 2002
     AND d_moy  = 7;

CREATE DIMENSION TABLE q06_cat_avg_price_88 AS
    SELECT i_category AS i_category,
           AVG (i_current_price) * 1.2 AS avg_price
      FROM item
     GROUP BY i_category;

SELECT a.ca_state AS state, count(*) as cnt
  FROM customer_address a, customer c,
       store_sales s, date_dim d, item i,
                         q06_specific_month_88 m, q06_cat_avg_price_88  p
 WHERE a.ca_address_sk   = c.c_current_addr_sk
   AND c.c_customer_sk   = s.ss_customer_sk
   AND s.ss_sold_date_sk = d.d_date_sk
   AND s.ss_item_sk      = i.i_item_sk
   AND d.d_month_seq     = m.d_month_seq
   AND p.i_category      = i.i_category
   AND i.i_current_price > p.avg_price
 GROUP BY a.ca_state
HAVING COUNT(*) >= 10
 ORDER BY cnt
 LIMIT 100;

DROP TABLE IF EXISTS q06_specific_month_88;
DROP TABLE IF EXISTS q06_cat_avg_price_88;
END;
```

Listing 1.7. Query 7

Query 8 For online sales, compare the total sales in which customers checked online reviews before making the purchase and that of sales in which customers did not read reviews. Consider only online sales for a specific category in a given year.

```
BEGIN;
DROP VIEW clicks;
CREATE VIEW clicks AS (
  SELECT c.wcs_item_sk AS item,
         c.wcs_user_sk AS uid,
         c.wcs_click_date_sk AS c_date,
         c.wcs_click_time_sk AS c_time,
         c.wcs_sales_sk AS sales_sk,
         w.wp_type AS wpt
    FROM web_clickstreams c, web_page w
   WHERE c.wcs_web_page_sk = w.wp_web_page_sk
     and c.wcs_user_sk IS NOT NULL
);

DROP VIEW sales_review;
CREATE VIEW sales_review AS (
  SELECT s_sk
    FROM nPath(ON clicks
```

```
          PARTITION BY uid
          ORDER BY c_date, c_time
          MODE ('NONOVERLAPPING')
          PATTERN ('A+.C*.B')
          SYMBOLS (wpt = 'review' AS A, TRUE AS C,
                   sales_sk IS NOT NULL AS B)
          RESULT (FIRST (c_date OF B) AS s_date,
                  FIRST (sales_sk OF B) AS s_sk))
   WHERE s_date > 2451424 AND s_date <2451424+365
);

SELECT SUM (CASE WHEN ws.ws_sk IN (SELECT * FROM sales_review)
                 THEN ws_net_paid
                 ELSE 0 END) AS review_sales_amount,
       SUM (ws_net_paid) -
       SUM (CASE WHEN ws.ws_sk IN (SELECT * FROM sales_review)
                THEN ws_net_paid
                ELSE 0 END) AS no_review_sales_amount
  FROM web_sales ws
 WHERE ws.ws_sold_date_sk > 2451424
   AND ws.ws_sold_date_sk <2451424+365;
END;
```

Listing 1.8. Query 8

Query 9 (TPC-DS 48) Calculate the total sales by different types of customers (e.g., based on marital status, education status), sales price and different combinations of state and sales profit.

```
SELECT SUM (ss_quantity)
  FROM store_sales, store, customer_demographics,
       customer_address, date_dim
 WHERE s_store_sk = ss_store_sk
   AND  ss_sold_date_sk = d_date_sk
   AND d_year = 1998
   AND ((cd_demo_sk = ss_cdemo_sk
     AND cd_marital_status = 'M'
     AND cd_education_status = '4␣yr␣Degree'
     AND ss_sales_price between 100.00 AND 150.00)
    OR
       (cd_demo_sk = ss_cdemo_sk
     AND cd_marital_status = 'M'
     AND cd_education_status = '4␣yr␣Degree'
     AND ss_sales_price between 50.00 AND 100.00)
    OR
       (cd_demo_sk = ss_cdemo_sk
     AND cd_marital_status = 'M'
     AND cd_education_status = '4␣yr␣Degree'
     AND ss_sales_price between 150.00 AND 200.00))
   AND ((ss_addr_sk = ca_address_sk
     AND ca_country = 'United␣States'
     AND ca_state in ('KY', 'GA', 'NM')
     AND ss_net_profit between 0 AND 2000)
    OR
       (ss_addr_sk = ca_address_sk
     AND ca_country = 'United␣States'
     AND ca_state in ('MT', 'OR', 'IN')
     AND ss_net_profit between 150 AND 3000)
    OR
       (ss_addr_sk = ca_address_sk
     AND ca_country = 'United␣States'
     AND ca_state in ('WI', 'MO', 'WV')
     AND ss_net_profit between 50 AND 25000));
```

Listing 1.9. Query 9

Query 10 For all products, extract sentences from its product reviews that contain positive or negative sentiment and display the sentiment polarity of the extracted sentences.

```
SELECT pr_item_sk, out_content, out_polarity, out_sentiment_words
  FROM ExtractSentiment
       (ON product_reviews100
        TEXT_COLUMN ('pr_review_content')
        MODEL ('dictionary')
        LEVEL ('sentence')
        ACCUMULATE ('pr_item_sk')
       )
 WHERE out_polarity = 'NEG'
    OR out_polarity = 'POS';
```

Listing 1.10. Query 10

Query 11 For a given product, measure the correlation of sentiments, including the number of reviews and average review ratings, on product monthly revenues.

```
BEGIN;
DROP VIEW IF EXISTS review_stats;
CREATE VIEW review_stats AS(
  SELECT p.pr_item_sk AS pid,
         CAST(p.r_count AS INT) AS reviews_count,
         CAST(p.avg_rating AS INT) AS avg_rating,
         CAST(s.revenue AS INT) AS m_revenue
    FROM (SELECT pr_item_sk, COUNT(*) AS r_count,
                 AVG(pr_review_rating) AS avg_rating
            FROM product_reviews
           WHERE pr_item_sk IS NOT NULL
           GROUP BY 1) p
         JOIN
         (SELECT ws_item_sk, SUM(ws_net_paid) AS revenue
            FROM web_sales
           WHERE ws_sold_date_sk > 2452642-30
             AND ws_sold_date_sk < 2452642
             AND ws_item_sk IS NOT NULL
           GROUP BY 1) s
         ON p.pr_item_sk = s.ws_item_sk);

SELECT *
  FROM corr_reduce (ON
         corr_map (ON
           review_stats
           COLUMNS ('[m_revenue:reviews_count],[m_revenue:avg_rating]')
           KEY_NAME('k')
         )
         PARTITION BY k);

DROP VIEW review_stats;
END;
```

Listing 1.11. Query 11

Query 12 Find all customers, who viewed items of a given category on the web in a given month and year that was followed by an in-store purchase in the three consecutive months.

```
SELECT *
  FROM nPath (
     ON (SELECT c.wcs_item_sk AS item,
                c.wcs_user_sk AS uid,
```

```
                c.wcs_click_date_sk AS c_date,
                c.wcs_click_time_sk AS c_time
           FROM web_clickstreams c, item i
          WHERE c.wcs_item_sk = i.i_item_sk
            AND i.i_category in ('Books', 'Electronics')
            AND c.wcs_user_sk IS NOT NULL
            AND c.wcs_click_date_sk > 2451424
            AND c.wcs_click_date_sk < 2451424+30) AS click
     PARTITION BY uid
     ORDER BY c_date, c_time
     ON (SELECT s.ss_item_sk AS item,
                s.ss_customer_sk AS uid,
                s.ss_sold_date_sk AS s_date,
                s.ss_sold_time_sk AS s_time
           FROM store_sales s, item i
          WHERE s.ss_item_sk = i.i_item_sk
            AND i.i_category in ('Books', 'Electronics')
            AND s.ss_customer_sk IS NOT NULL
            AND s.ss_sold_date_sk > 2451424
            AND s.ss_sold_time_sk < 2451424+120) AS sale
     PARTITION BY uid order by s_date, s_time
     MODE ('NONOVERLAPPING')
     PATTERN ('(c+).(s)')
     SYMBOLS (click.uid IS NOT NULL AS c,
              sale.uid IS NOT NULL AS s)
     RESULT (FIRST(c_date OF c) AS c_date,
             FIRST(s_date OF s) AS s_date,
             FIRST(sale.uid OF s) AS user_sk)
);
```

Listing 1.12. Query 12

Query 13 (TPC-DS 74) Display customers with both store and web sales in consecutive years for whom the increase in web sales exceeds the increase in store sales for a specified year.

```
BEGIN;
DROP TABLE IF EXISTS q74_customer_year_total_880;

CREATE TEMP TABLE q74_customer_year_total_880(
  customer_id          VARCHAR(16),
  customer_first_name  CHAR(20)
  customer_last_name   CHAR(30)
  year                 INTEGER
  year_total           DECIMAL(15,2)
  sale_type            VARCHAR(2))
  DISTRIBUTE BY hash (customer_id) AS
  SELECT c_customer_id    customer_id,
         c_first_name     customer_first_name,
         c_last_name      customer_last_name,
         d_year           year,
         SUM(ss_net_paid) year_total,
         's'::VARCHAR     sale_type
    FROM customer, store_sales, date_dim
   WHERE c_customer_sk = ss_customer_sk
     AND ss_sold_date_sk = d_date_sk
     AND d_year IN (1999 ,1999 + 1)
   GROUP BY c_customer_id, c_first_name,
            c_last_name, d_year
  UNION ALL
  SELECT c_customer_id    customer_id,
         c_first_name     customer_first_name,
         c_last_name      customer_last_name,
         d_year           year,
         SUM(ws_net_paid) year_total,
         'w'::VARCHAR     sale_type
```

```
     FROM customer, web_sales, date_dim
    WHERE c_customer_sk = ws_bill_customer_sk
      AND ws_sold_date_sk = d_date_sk
      AND d_year IN (1999 ,1999 + 1)
    GROUP BY c_customer_id, c_first_name,
             c_last_name, d_year;

SELECT t_s_secyear.customer_id, t_s_secyear.customer_first_name,
       t_s_secyear.customer_last_name
  FROM q74_customer_year_total_880 t_s_firstyear,
       q74_customer_year_total_880 t_s_secyear,
       q74_customer_year_total_880 t_w_firstyear,
       q74_customer_year_total_880 t_w_secyear
 WHERE t_s_secyear.customer_id   = t_s_firstyear.customer_id
   AND t_s_firstyear.customer_id = t_w_secyear.customer_id
   AND t_s_firstyear.customer_id = t_w_firstyear.customer_id
   AND t_s_firstyear.sale_type   = 's'
   AND t_w_firstyear.sale_type   = 'w'
   AND t_s_secyear.sale_type     = 's'
   AND t_w_secyear.sale_type     = 'w'
   AND t_s_firstyear.year        = 1999
   AND t_s_secyear.year          = 1999 + 1
   AND t_w_firstyear.year        = 1999
   AND t_w_secyear.year          = 1999 + 1
   AND t_s_firstyear.year_total  > 0
   AND t_w_firstyear.year_total  > 0
   AND CASE WHEN t_w_firstyear.year_total > 0
            THEN t_w_secyear.year_total / t_w_firstyear.year_total
            ELSE NULL END
     > CASE WHEN t_s_firstyear.year_total > 0
            THEN t_s_secyear.year_total / t_s_firstyear.year_total
            ELSE NULL END
 ORDER BY 1
 LIMIT 100;

DROP TABLE IF EXISTS q74_customer_year_total_880;
END;
```

Listing 1.13. Query 13

Query 14 (TPC-DS 90) What is the ratio between the number of items sold over the internet in the morning (8 to 9am) to the number of items sold in the evening (7 to 8pm) of customers with a specified number of dependents. Consider only websites with a high amount of content.

```
SELECT CAST(amc AS DECIMAL(15,4)) / CAST(pmc AS DECIMAL(15,4)) am_pm_ratio
  FROM (SELECT COUNT(*) amc
          FROM web_sales, household_demographics, time_dim, web_page wp
         WHERE ws_sold_time_sk = time_dim.t_time_sk
           AND ws_ship_hdemo_sk = household_demographics.hd_demo_sk
           AND ws_web_page_sk = wp.wp_web_page_sk
           AND time_dim.t_hour BETWEEN 8 AND 8+1
           AND household_demographics.hd_dep_count = 5
           AND wp.wp_char_count BETWEEN 5000 AND 5200) at,
       (SELECT COUNT(*) pmc
          FROM web_sales, household_demographics , time_dim, web_page wp
         WHERE ws_sold_time_sk = time_dim.t_time_sk
           AND ws_ship_hdemo_sk = household_demographics.hd_demo_sk
           AND ws_web_page_sk = wp.wp_web_page_sk
           AND time_dim.t_hour BETWEEN 19 AND 19+1
           AND household_demographics.hd_dep_count = 5
           AND wp.wp_char_count BETWEEN 5000 AND 5200) pt
 ORDER BY am_pm_ratio ;
```

Listing 1.14. Query 14

Query 15 Find the categories with flat or declining sales for in store purchases during a given year for a given store.

```
BEGIN;
DROP VIEW IF EXISTS category_coefficient;
DROP VIEW IF EXISTS time_series_category;

CREATE VIEW time_series_category AS (
  SELECT i.i_category_id AS cat,
         s.ss_sold_date_sk AS d,
         SUM(s.ss_net_paid) AS sales
    FROM store_sales s, item i
   WHERE s.ss_item_sk = i.i_item_sk
     AND i.i_category_id IS NOT NULL
     AND s.ss_sold_date_sk > 2451424
     AND s.ss_sold_date_sk < 2451424+365
     AND s.ss_store_sk = 10
   GROUP BY 1,2
);

CREATE VIEW category_coefficient AS (
  SELECT 1 AS category, coefficient_index, value AS slope
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_category
               WHERE cat = 1)
           ) PARTITION BY 1
         )
   WHERE coefficient_index = 1
  UNION ALL
  SELECT 2, coefficient_index, value
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_category
               WHERE cat = 2)
           ) PARTITION BY 1
         )
   WHERE coefficient_index = 1
  UNION ALL
  SELECT 3, coefficient_index, value
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_category
               WHERE cat = 3)
           ) PARTITION BY 1
         )
   WHERE coefficient_index = 1
  UNION ALL
  SELECT 4, coefficient_index, value
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_category
               WHERE cat = 4)
           ) PARTITION BY 1
         )
   WHERE coefficient_index = 1
  UNION ALL
  SELECT 5, coefficient_index, value
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_category
               WHERE cat = 5)
           ) PARTITION BY 1
```

```
        )
    WHERE coefficient_index = 1
  UNION ALL
  SELECT 6, coefficient_index, value
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_category
               WHERE cat = 6)
           ) PARTITION BY 1
         )
    WHERE coefficient_index = 1
  UNION ALL
  SELECT 7, coefficient_index, value
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_category
               WHERE cat = 7)
           ) PARTITION BY 1
         )
    WHERE coefficient_index = 1
  UNION ALL
  SELECT 8, coefficient_index, value
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_category
               WHERE cat = 8)
           ) partition by 1
         )
    WHERE coefficient_index = 1
  UNION ALL
  SELECT 9, coefficient_index, value
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_category
               WHERE cat = 9)
           ) partition by 1
         )
    WHERE coefficient_index = 1
  UNION ALL
  SELECT 10, coefficient_index, value
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_category
               WHERE cat = 10)
           ) partition by 1
         )
    WHERE coefficient_index = 1;

SELECT * FROM category_coefficient WHERE slope < 0;

DROP VIEW category_coefficient;
DROP VIEW time_series_category;
END;
```

Listing 1.15. Query 15

Query 16 (TPC-DS 40) Compute the impact of an item price change on the store sales by computing the total sales for items in a 30 day period before and after the price change. Group the items by location of warehouse where they were delivered from.

```
SELECT w_state, i_item_id,
      SUM (CASE WHEN (CAST (d_date AS DATE) < CAST ('1998-03-16' AS DATE))
                THEN ws_sales_price - coalesce(wr_refunded_cash,0) ELSE 0
                    END)
                AS sales_before,
      SUM (CASE WHEN (CAST (d_date AS DATE) >= CAST ('1998-03-16' AS DATE))
                THEN ws_sales_price - coalesce(wr_refunded_cash,0) ELSE 0
                    END)
                AS sales_after
  FROM web_sales LEFT OUTER JOIN web_returns
       ON (ws_order_number = wr_order_number
       AND ws_item_sk = wr_item_sk),
       warehouse, item, date_dim
 WHERE i_item_sk       = ws_item_sk
   AND ws_warehouse_sk = w_warehouse_sk
   AND ws_sold_date_sk = d_date_sk
   AND d_date BETWEEN (CAST ('1998-03-16' AS DATE) - INTERVAL '30␣day')
                  AND (CAST ('1998-03-16' AS DATE) + INTERVAL '30␣day')
 GROUP BY w_state,i_item_id
 ORDER BY w_state,i_item_id;
```

Listing 1.16. Query 16

Query 17 (TPC-DS 61) Find the ratio of items sold with and without promotions in a given month and year. Only items in certain categories sold to customers living in a specific time zone are considered.

```
SELECT promotions, total,
       CAST(promotions AS DECIMAL(15,4)) /
       CAST(total AS DECIMAL(15,4)) * 100
  FROM (SELECT SUM (ss_ext_sales_price) promotions
          FROM store_sales, store, promotion, date_dim,
               customer, customer_address, item
         WHERE ss_sold_date_sk = d_date_sk
           AND ss_store_sk = s_store_sk
           AND ss_promo_sk = p_promo_sk
           AND ss_customer_sk= c_customer_sk
           AND ca_address_sk = c_current_addr_sk
           AND ss_item_sk = i_item_sk
           AND ca_gmt_offset = -7
           AND i_category = 'Jewelry'
           AND (p_channel_dmail = 'Y' OR p_channel_email = 'Y'
                                      OR p_channel_tv = 'Y')
           AND s_gmt_offset = -7
           AND d_year = 2001
           AND d_moy  = 12) promotional_sales,
       (SELECT sum(ss_ext_sales_price) total
          FROM store_sales, store, date_dim,
               customer, customer_address, item
         WHERE ss_sold_date_sk = d_date_sk
           AND ss_store_sk = s_store_sk
           AND ss_customer_sk= c_customer_sk
           AND ca_address_sk = c_current_addr_sk
           AND ss_item_sk = i_item_sk
           AND ca_gmt_offset = -7
           AND i_category = 'Jewelry'
           AND s_gmt_offset = -7
           AND d_year = 2001
           AND d_moy  = 12) all_sales
 ORDER BY promotions, total;
```

Listing 1.17. Query 17

Query 18 Identify the stores with flat or declining sales in 3 consecutive months, check if there are any negative reviews regarding these stores available online.

```sql
BEGIN;
DROP VIEW IF EXISTS store_coefficient;
DROP VIEW IF EXISTS time_series_store;

CREATE VIEW time_series_store AS (
  SELECT ss_store_sk AS store, ss_sold_date_sk AS d,
         SUM(ss_net_paid) AS sales
    FROM store_sales
   WHERE ss_sold_date_sk > 2451424
     AND ss_sold_date_sk < 2451424+90
   GROUP BY 1,2);

CREATE VIEW store_coefficient AS (
  SELECT 1 AS store, coefficient_index, value AS slope
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_store
               WHERE store = 1)
           ) PARTITION BY 1
         )
   WHERE coefficient_index = 1
  UNION ALL
  SELECT 2 AS store, coefficient_index, value AS slope
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_store
               WHERE store = 2)
           ) PARTITION BY 1
         )
   WHERE coefficient_index = 1
  UNION ALL
  SELECT 3 AS store, coefficient_index, value AS slope
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_store
               WHERE store = 3)
           ) PARTITION BY 1
         )
   WHERE coefficient_index = 1
  UNION ALL
  SELECT 4 AS store, coefficient_index, value AS slope
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_store
               WHERE store = 4)
           ) PARTITION BY 1
         )
   WHERE coefficient_index = 1
  UNION ALL
  SELECT 5 AS store, coefficient_index, value AS slope
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_store
               WHERE store = 5)
           ) PARTITION BY 1
         )
   WHERE coefficient_index = 1
  UNION ALL
  SELECT 6 AS store, coefficient_index, value AS slope
    FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_store
```

```sql
              WHERE store = 6)
          ) PARTITION BY 1
        )
    WHERE coefficient_index = 1
   UNION ALL
   SELECT 7 AS store, coefficient_index, value AS slope
     FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_store
              WHERE store = 7)
          ) PARTITION BY 1
        )
    WHERE coefficient_index = 1
   UNION ALL
   SELECT 8 AS store, coefficient_index, value AS slope
     FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_store
              WHERE store = 8)
          ) PARTITION BY 1
        )
    WHERE coefficient_index = 1
   UNION ALL
   SELECT 9 AS store, coefficient_index, value AS slope
     FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_store
              WHERE store = 9)
          ) PARTITION BY 1
        )
    WHERE coefficient_index = 1
   UNION ALL
   SELECT 10 AS store, coefficient_index, value AS slope
     FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_store
              WHERE store = 10)
          ) PARTITION BY 1
        )
    WHERE coefficient_index = 1
   UNION ALL
   SELECT 11 AS store, coefficient_index, value AS slope
     FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_store
              WHERE store = 11)
          ) PARTITION BY 1
        )
    WHERE coefficient_index = 1
   UNION ALL
   SELECT 12 AS store, coefficient_index, value AS slope
     FROM linreg (ON
           linregmatrix (ON
             (SELECT d, sales
                FROM time_series_store
              WHERE store = 12)
          ) PARTITION BY 1
        )
    WHERE coefficient_index = 1);

 SELECT s_store_name, pr_review_date, out_content,
        out_polarity, out_sentiment_words
   FROM ExtractSentiment (ON
```

```
        (SELECT s_store_name, pr_review_content, pr_review_date
           FROM store_coefficient c, store s, product_reviews
          WHERE c.slope < 0
            AND s.s_store_sk = c.store
            AND pr_review_content like '%'||s_store_name||'%')
        TEXT_COLUMN('pr_review_content')
        MODEL('dictionary')
        LEVEL('DOCUMENT')
        ACCUMULATE('s_store_name','pr_review_date'))
 WHERE out_polarity = 'NEG';

DROP VIEW store_coefficient;
DROP VIEW time_series_store;
END;
```

Listing 1.18. Query 18

Query 19 Retrieve the items with the highest number of returns where the number of returns was approximately equivalent across all store and web channels (within a tolerance of +/- 10%), within the week ending a given date. Analyze the online reviews for these items to see if there are any major negative reviews.

```
BEGIN;

CREATE VIEW sr_items AS
  (SELECT i_item_sk item_id,
          SUM(sr_return_quantity) sr_item_qty
     FROM store_returns, item, date_dim
    WHERE sr_item_sk = i_item_sk
      AND d_date IN
          (SELECT d_date
             FROM date_dim
            WHERE d_week_seq IN
                    (SELECT d_week_seq
                       FROM date_dim
                      WHERE d_date IN
                           ('1998-01-02','1998-10-15','1998-11-10')))
      AND sr_returned_date_sk = d_date_sk
    GROUP BY i_item_sk
   HAVING SUM (sr_return_quantity) > 0);

CREATE VIEW wr_items AS
  (SELECT i_item_sk item_id, SUM(wr_return_quantity) wr_item_qty
     FROM web_returns, item, date_dim
    WHERE wr_item_sk = i_item_sk
      AND d_date IN (SELECT d_date
  FROM date_dim
  WHERE d_week_seq in
        (SELECT d_week_seq
           FROM date_dim
          WHERE d_date IN ('2001-03-10' ,'2001-08-04' ,'2001-11-14')))
    AND wr_returned_date_sk   = d_date_sk
  GROUP BY i_item_sk
 HAVING SUM(wr_return_quantity) > 0);

CREATE VIEW return_items AS
  (SELECT sr_items.item_id item, sr_item_qty,
          100.0 * sr_item_qty / (sr_item_qty + wr_item_qty) / 2.0 sr_dev,
          wr_item_qty, 100.0 * wr_item_qty /
          (sr_item_qty + wr_item_qty) / 2.0 wr_dev,
          (sr_item_qty  +wr_item_qty) / 2.0 "average"
     FROM sr_items, wr_items
    WHERE sr_items.item_id = wr_items.item_id
    ORDER BY average DESC
    LIMIT 100) ;
```

```
SELECT pr_item_sk, out_content, out_polarity, out_sentiment_words
  FROM ExtractSentiment (ON
         product_reviews
         TEXT_COLUMN ('pr_review_content')
         MODEL ('dictionary')
         LEVEL ('sentence')
         ACCUMULATE ('pr_item_sk')
       )
 WHERE out_polarity = 'NEG'
   AND pr_item_sk IN (SELECT item FROM return_items);

DROP VIEW return_items;
DROP VIEW wr_items;
DROP VIEW sr_items;
END;
```

Listing 1.19. Query 19

Query 20 Customer segmentation for return analysis: Customers are separated along the following dimensions: return frequency, return order ratio (total number of orders partially or fully returned versus the total number of orders), return item ratio (total number of items returned versus the number of items purchased), return amount ration (total monetary amount of items returned versus the amount purchased), return order ratio. Consider the store returns during a given year for the computation.

```
CREATE VIEW sales_returns AS (
  SELECT s.ss_sold_date_sk AS s_date,
         r.sr_returned_date_sk AS r_date,
         s.ss_item_sk AS item,
         s.ss_ticket_number AS oid,
         s.ss_net_paid AS s_amount,
         r.sr_return_amt AS r_amount,
         (CASE WHEN s.ss_customer_sk IS NULL
               THEN r.sr_customer_sk ELSE s.ss_customer_sk END) AS cid,
         s.ss_customer_sk AS s_cid,
         sr_customer_sk AS r_cid
    FROM store_sales s LEFT JOIN store_returns100 r ON
             s.ss_item_sk = r.sr_item_sk
         AND s.ss_ticket_number = r.sr_ticket_number
   WHERE s.ss_sold_date_sk IS NOT NULL);

CREATE VIEW clusters AS (
  SELECT cid,
         100.0 * COUNT (DISTINCT (CASE WHEN r_date IS NOT NULL
                                  THEN oid ELSE NULL END))
               / COUNT (DISTINCT oid) AS r_order_ratio,
         SUM (CASE WHEN r_date IS NOT NULL THEN 1 ELSE 0 END)
               / COUNT (item) * 100 AS r_item_ratio,
         SUM (CASE WHEN r_date IS NOT NULL THEN r_amount ELSE 0 END)
               / SUM (s_amount) * 100 AS r_amount_ratio,
         COUNT (DISTINCT (CASE WHEN r_date IS NOT NULL
                               THEN r_date ELSE NULL END))
                               AS r_freq
    FROM sales_returns
   WHERE cid IS NOT NULL
   GROUP BY 1
  HAVING COUNT (DISTINCT (CASE WHEN r_date IS NOT NULL
                               THEN r_date ELSE NULL END)) > 1);

SELECT *
  FROM kmeans (ON
       (SELECT 1)
```

```
       PARTITION BY 1
       DATABASE ('benchmark')
       USERID ('benchmark')
       PASSWORD ('benchmark')
       INPUTTABLE ('clusters AS c')
       OUTPUTTABLE ('user_return_groups')
       NUMBERK('4'));

SELECT clusterid, cid
  FROM kmeansplot (ON
       clusters AS c
       PARTITION BY ANY
       ON user_return_groups dimension
       CENTROIDSTABLE ('user_return_groups'))
 ORDER BY clusterid, cid;

DROP TABLE user_return_groups;
DROP VIEW clusters;
DROP VIEW sales_returns;
```

Listing 1.20. Query 20

Query 21 (TPC-DS 29) Get all items that were sold in stores in a given month and year and which were returned in the next six months and re-purchased by the returning customer afterwards through the web sales channel in the following three years. For those these items, compute the total quantity sold through the store, the quantity returned and the quantity purchased through the web. Group this information by item and store.

```
SELECT i_item_id, i_item_desc, s_store_id, s_store_name,
       sum(ss_quantity) AS store_sales_quantity,
       sum(sr_return_quantity) AS store_returns_quantity,
       sum(ws_quantity) AS web_sales_quantity
  FROM store_sales, store_returns, web_sales, date_dim d1,
       date_dim d2, date_dim d3, store, item
 WHERE d1.d_moy            = 4
   AND d1.d_year           = 1998
   AND d1.d_date_sk        = ss_sold_date_sk
   AND i_item_sk           = ss_item_sk
   AND s_store_sk          = ss_store_sk
   AND ss_customer_sk      = sr_customer_sk
   AND ss_item_sk          = sr_item_sk
   AND ss_ticket_number    = sr_ticket_number
   AND sr_returned_date_sk = d2.d_date_sk
   AND d2.d_moy            BETWEEN 4 AND  4 + 3
   AND d2.d_year           = 1998
   AND sr_customer_sk      = ws_bill_customer_sk
   AND sr_item_sk          = ws_item_sk
   AND ws_sold_date_sk     = d3.d_date_sk
   AND d3.d_year           IN (1998,1998+1,1998+2)
 GROUP BY i_item_id, i_item_desc, s_store_id, s_store_name
 ORDER BY i_item_id, i_item_desc, s_store_id, s_store_name;
```

Listing 1.21. Query 21

Query 22 (TPC-DS 21) For all items whose price was changed on a given date, compute the percentage change in inventory between the 30-day period before the price change and the 30-day period after the change. Group this information by warehouse.

```sql
SELECT *
  FROM (SELECT w_warehouse_name, i_item_id,
               SUM (CASE WHEN (CAST (d_date AS DATE) < CAST ('2000-05-08' AS
                    DATE))
                         THEN inv_quantity_on_hand
                         ELSE 0 END) AS inv_before
               SUM (CASE WHEN (CAST (d_date AS date) >= CAST ('2000-05-08'
                    AS DATE))
                         THEN inv_quantity_on_hand
                         ELSE 0 END) AS inv_after
          FROM inventory, warehouse, item, date_dim
         WHERE i_current_price BETWEEN 0.99 AND 1.49
           AND i_item_sk = inv_item_sk
           AND inv_warehouse_sk = w_warehouse_sk
           AND inv_date_sk = d_date_sk
           AND d_date BETWEEN (CAST ('2000-05-08' AS DATE) - 30)
                          AND (CAST ('2000-05-08' AS DATE) + 30)
         GROUP BY w_warehouse_name, i_item_id) x
 WHERE (CASE WHEN inv_before > 0
             THEN inv_after / inv_before
             ELSE NULL END) BETWEEN 2.0/3.0 AND 3.0/2.0
 ORDER BY w_warehouse_name, i_item_id;
```

Listing 1.22. Query 22

Query 23 (TPC-DS 39) This query contains multiple, related iterations:

1. Calculate the coefficient of variation and mean of every item and warehouse of two consecutive months.
2. Find items that had a coefficient of variation in the first months of 1.5 or larger.

```sql
BEGIN;

CREATE VIEW inv AS
  (SELECT w_warehouse_name, w_warehouse_sk, i_item_sk,
          d_moy, stdev, mean,
          CASE mean WHEN 0 THEN NULL ELSE stdev/mean END cov
     FROM (SELECT w_warehouse_name, w_warehouse_sk, i_item_sk,
                  d_moy, stddev_samp(inv_quantity_on_hand) stdev,
                  avg(inv_quantity_on_hand) mean
             FROM inventory, item, warehouse, date_dim
            WHERE inv_item_sk = i_item_sk
              AND inv_warehouse_sk = w_warehouse_sk
              AND inv_date_sk = d_date_sk
              AND d_year = 1998
            GROUP BY w_warehouse_name, w_warehouse_sk,
                     i_item_sk, d_moy) foo
    WHERE CASE mean WHEN 0 THEN 0 ELSE stdev/mean END > 1);

SELECT inv1.w_warehouse_sk, inv1.i_item_sk, inv1.d_moy, inv1.mean,
       inv1.cov, inv2.w_warehouse_sk, inv2.i_item_sk, inv2.d_moy,
       inv2.mean, inv2.cov
  FROM inv inv1,inv inv2
 WHERE inv1.i_item_sk = inv2.i_item_sk
   AND inv1.w_warehouse_sk =  inv2.w_warehouse_sk
   AND inv1.d_moy=1
   AND inv2.d_moy=1+1
 ORDER BY inv1.w_warehouse_sk, inv1.i_item_sk,
          inv1.d_moy, inv1.mean, inv1.cov,
          inv2.d_moy,inv2.mean, inv2.cov;

DROP VIEW IF EXISTS inv;
CREATE VIEW inv AS
```

```
  (SELECT w_warehouse_name, w_warehouse_sk, i_item_sk,
          d_moy, stdev, mean,
          CASE mean WHEN 0 THEN NULL ELSE stdev/mean END cov
     FROM (SELECT w_warehouse_name, w_warehouse_sk, i_item_sk,
                  d_moy, stddev_samp(inv_quantity_on_hand) stdev,
                  avg(inv_quantity_on_hand) mean
             FROM inventory, item, warehouse, date_dim
            WHERE inv_item_sk = i_item_sk
              AND inv_warehouse_sk = w_warehouse_sk
              AND inv_date_sk = d_date_sk
              AND d_year = 1998
            GROUP BY w_warehouse_name, w_warehouse_sk,
                  i_item_sk,d_moy) foo
    WHERE CASE mean WHEN 0 THEN 0 ELSE stdev/mean END > 1);

SELECT inv1.w_warehouse_sk, inv1.i_item_sk, inv1.d_moy,
       inv1.mean, inv1.cov, inv2.w_warehouse_sk, inv2.i_item_sk,
       inv2.d_moy, inv2.mean, inv2.cov
  FROM inv inv1, inv inv2
 WHERE inv1.i_item_sk = inv2.i_item_sk
   AND inv1.w_warehouse_sk = inv2.w_warehouse_sk
   AND inv1.d_moy= 2
   AND inv2.d_moy= 2 + 1
   AND inv1.cov > 1.5
 ORDER BY inv1.w_warehouse_sk, inv1.i_item_sk, inv1.d_moy,
          inv1.mean,inv1.cov, inv2.d_moy, inv2.mean, inv2.cov;

DROP VIEW inv;
END;
```

Listing 1.23. Query 23

Query 24 For a given product, measure the effect of competitors' prices on products' in-store and online sales. (Compute the cross-price elasticity of demand for a given product).

```
BEGIN;

CREATE VIEW competitor_price_view AS
  (SELECT i_item_sk, (imp_competitor_price - i_current_price)
          / i_current_price AS price_change, imp_start_date,
          imp_end_date - imp_start_date AS no_days
     FROM item, item_marketprices
    WHERE imp_item_sk = i_item_sk
      AND i_item_sk in (7,17)
      AND imp_competitor_price < i_current_price);

CREATE VIEW self_ws_view AS
  (SELECT ws_item_sk,
          SUM (CASE WHEN ws_sold_date_sk >= c.imp_start_date
                     AND ws_sold_date_sk < c.imp_start_date + c.no_days
                    THEN ws_quantity ELSE 0 END) AS current_ws,
          SUM (CASE WHEN ws_sold_date_sk >= c.imp_start_date - c.no_days
                     AND ws_sold_date_sk < c.imp_start_date
                    THEN ws_quantity ELSE 0 END) AS prev_ws
     FROM web_sales, competitor_price_view c
    WHERE ws_item_sk = c.i_item_sk
    GROUP BY 1);

CREATE VIEW self_ss_view AS
  (SELECT ss_item_sk,
          SUM (CASE WHEN ss_sold_date_sk >= c.imp_start_date
                     AND ss_sold_date_sk < c.imp_start_date + c.no_days
                    THEN ss_quantity ELSE 0 END) AS current_ss,
          SUM (CASE WHEN ss_sold_date_sk >= c.imp_start_date - c.no_days
                     AND ss_sold_date_sk < c.imp_start_date
```

```
                    THEN ss_quantity ELSE 0 END) AS prev_ss
     FROM store_sales, competitor_price_view c
    WHERE c.i_item_sk = ss_item_sk
    GROUP BY 1);

SELECT i_item_sk, (current_ss + current_ws-prev_ss-prev_ws)
       / ((prev_ss + prev_ws) * price_change) AS cross_price_elasticity
  FROM competitor_price_view, self_ws_view, self_ss_view
 WHERE i_item_sk = ws_item_sk
   AND i_item_sk = ss_item_sk;

DROP VIEW self_ws_view;
DROP VIEW self_ss_view;
DROP VIEW competitor_price_view;
END;
```

Listing 1.24. Query 24

Query 25 Customer segmentation analysis: Customers are separated along the following key shopping dimensions: recency of last visit, frequency of visits and monetary amount. Use the store and online purchase data during a given year to compute.

```
DROP VIEW usersegments;
CREATE VIEW usersegments AS
  (SELECT ss_customer_sk AS cid, ss_ticket_number AS oid,
          ss_sold_date_sk AS dateid, sum(ss_net_paid) AS amount
     FROM store_sales
    WHERE ss_sold_date_sk > 2452277
      AND ss_customer_sk IS NOT NULL
    GROUP BY 1,2,3
   UNION ALL
   SELECT ws_bill_customer_sk AS cid, ws_order_number AS oid,
          ws_sold_date_sk AS dateid, SUM(ws_net_paid) AS amount
     FROM web_sales
    WHERE ws_sold_date_sk > 2452277
      AND ws_bill_customer_sk is not null
          GROUP BY 1,2,3);

DROP VIEW clusteringtable;
CREATE VIEW clusteringtable AS
  (SELECT cid AS id,
          CASE WHEN 2452642 - MAX(dateid) < 60
               THEN 1.0 ELSE 0.0 END as recency,
          COUNT(oid) AS frequency,
          SUM(amount) AS totalspend
     FROM usersegments
    GROUP BY 1);

DROP TABLE user_shopping_groups;
SELECT *
  FROM kmeans (ON
       (SELECT 1)
       PARTITION BY 1
       DATABASE('benchmark')
       USERID('benchmark')
       PASSWORD('benchmark')
       INPUTTABLE ('clusteringtable␣AS␣c')
       OUTPUTTABLE ('user_shopping_groups')
       NUMBERK('8'));
```

Listing 1.25. Query 25

Query 26 Cluster customers into book buddies/ club groups based on their in store book purchasing histories.

```
CREATE VIEW clusteringtable AS
  (SELECT ss.ss_customer_sk AS cid,
          COUNT(CASE WHEN i.i_class_id=1 THEN 1 ELSE NULL END) AS id1,
          COUNT(CASE WHEN i.i_class_id=3 THEN 1 ELSE NULL END) AS id3,
          COUNT(CASE WHEN i.i_class_id=5 THEN 1 ELSE NULL END) AS id5,
          COUNT(CASE WHEN i.i_class_id=7 THEN 1 ELSE NULL END) AS id7,
          COUNT(CASE WHEN i.i_class_id=9 THEN 1 ELSE NULL END) AS id9,
          COUNT(CASE WHEN i.i_class_id=11 THEN 1 ELSE NULL END) AS id11,
          COUNT(CASE WHEN i.i_class_id=13 THEN 1 ELSE NULL END) AS id13,
          COUNT(CASE WHEN i.i_class_id=15 THEN 1 ELSE NULL END) AS id15,
          COUNT(CASE WHEN i.i_class_id=2 THEN 1 ELSE NULL END) AS id2,
          COUNT(CASE WHEN i.i_class_id=4 THEN 1 ELSE NULL END) AS id4,
          COUNT(CASE WHEN i.i_class_id=6 THEN 1 ELSE NULL END) AS id6,
          COUNT(CASE WHEN i.i_class_id=8 THEN 1 ELSE NULL END) AS id8,
          COUNT(CASE WHEN i.i_class_id=10 THEN 1 ELSE NULL END) AS id10,
          COUNT(CASE WHEN i.i_class_id=14 THEN 1 ELSE NULL END) AS id14,
          COUNT(CASE WHEN i.i_class_id=16 THEN 1 ELSE NULL END) AS id16
     FROM store_sales ss, item i
    WHERE ss.ss_item_sk = i.i_item_sk
      AND i.i_category = 'Books'
      AND ss.ss_customer_sk IS NOT NULL
    GROUP BY 1
   HAVING COUNT(ss.ss_item_sk) > 5);

SELECT *
  FROM kmeans (ON
       (SELECT 1)
       PARTITION BY 1
       DATABASE('benchmark')
       USERID('benchmark')
       PASSWORD('benchmark')
       INPUTTABLE ('clusteringtable␣AS␣c')
       OUTPUTTABLE ('book_club_groups')
       NUMBERK('2'));

SELECT clusterid, cid
  FROM kmeansplot (
       ON clusteringtable AS c
       PARTITION BY ANY
       ON book_club_groups dimension
       CENTROIDSTABLE ('book_club_groups'))
 ORDER BY clusterid, cid;

DROP TABLE IF EXISTS book_club_groups;
DROP VIEW IF EXISTS clusteringtable;
```

Listing 1.26. Query 26

Query 27 Extract competitor product names and model names (if any) from online product reviews for a given product.

```
SELECT DISTINCT *
  FROM FindNamedEntity (
    ON (SELECT pr_review_sk, pr_item_sk, pr_review_content
          FROM product_reviews
         WHERE pr_item_sk = 10653) AS p
    PARTITION BY ANY
    ON nameFind_configure AS "ConfigureTable" DIMENSION
    TEXT_COLUMN ('pr_review_content')
    MODEL('organization')
    OUTPUT_COLUMNS('pr_review_sk', 'pr_item_sk'));
```

Listing 1.27. Query 27

Query 28 Build text classifier for online review sentiment classification (positive, negative, neutral), using 60% of available reviews for training and the remaining 40% for testing. Display classifier accuracy on testing data.

```
CREATE FACT TABLE a32_trainingt (
  pr_review_sk BIGINT,
  pr_rating CHAR(3),
  pr_review_content TEXT,
  pr_item_sk BIGINT
) DISTRIBUTE BY HASH (pr_review_sk) AS
  SELECT pr_review_sk,
         (CASE pr_review_rating
          WHEN 1 THEN 'NEG'
          WHEN 2 THEN 'NEG'
          WHEN 3 THEN 'NEU'
          WHEN 4 THEN 'POS'
          WHEN 5 THEN 'POS' END) AS pr_rating,
         pr_review_content, pr_item_sk
    FROM product_reviews
   WHERE MOD (pr_review_sk, 5) IN (1,2,3);

CREATE FACT TABLE a32_testingt (
  pr_review_sk BIGINT,
  pr_rating CHAR(3),
  pr_review_content text,
  pr_item_sk BIGINT
) DISTRIBUTE BY HASH (pr_review_sk) AS
  SELECT pr_review_sk,
         (CASE pr_review_rating
          WHEN 1 THEN 'NEG'
          WHEN 2 THEN 'NEG'
          WHEN 3 THEN 'NEU'
          WHEN 4 THEN 'POS'
          WHEN 5 THEN 'POS' END) AS pr_rating,
         pr_review_content, pr_item_sk
    FROM product_reviews
   WHERE MOD (pr_review_sk, 5) IN (0, 4);

SELECT *
  FROM TextClassifierTrainer (
    ON (SELECT 1)
    PARTITION BY 1
    DATABASE('benchmark')
    USERID('benchmark')
    PASSWORD('benchmark')
    INPUTTABLE('a32_trainingt')
    TEXTCOLUMN('pr_review_content')
    CATEGORYCOLUMN('pr_rating')
    MODELFILE('senti_classifier.mod')
    CLASSIFIERTYPE('MaxEnt')
    NLPPARAMETERS('useStem:true'));

SELECT *
  FROM TextClassifier (
    ON InputTable('a32_testingt')
    TEXTCOLUMN('pr_review_content')
    MODEL('senti_classifier.mod')
    ACCUMULATE('pr_review_sk','pr_rating'));

SELECT *
  FROM TextClassifierEvaluator (
    ON TextClassifier (
       ON InputTable('a32_trainingt')
       TEXTCOLUMN('pr_review_content')
       MODEL('senti_classifier.mod')
       ACCUMULATE('pr_review_sk','pr_rating'))
    PARITION BY 1
```

```
    EXPECTCOLUMN('pr_rating')
    PREDICTCOLUMN('out_category'));

DROP TABLE a32_trainingt;
DROP TABLE a32_testingt;
```

Listing 1.28. Query 28

Query 29 Perform category affinity analysis for products purchased online together.

```
CREATE VIEW c_affinity_input AS
  (SELECT i.i_category_id AS category_cd,
          s.ws_bill_customer_sk AS customer_id
     FROM web_sales s INNER JOIN item i
       ON s.ws_item_sk = i_item_sk
    WHERE i.i_category_id IS NOT NULL);

SELECT *
  FROM cfilter (ON
       (SELECT 1)
       PARTITION BY 1
       DATABASE ('benchmark')
       USERID ('benchmark')
       PASSWORD ('benchmark')
       INPUTTABLE ('benchmark.c_affinity_input')
       OUTPUTTABLE ('c_affinity_out')
       DROPTABLE ('true')
       INPUTCOLUMNS ('category_cd')
       JOINCOLUMNS ('customer_id'));

SELECT * FROM c_affinity_out;

DROP TABLE IF EXISTS c_affinity_out;
DROP VIEW IF EXISTS c_affinity_input;
```

Listing 1.29. Query 29

Query 30 Perform category affinity analysis for products viewed together.

```
DROP VIEW IF EXISTS c_affinity_input;
CREATE VIEW c_affinity_input AS
  (SELECT i.i_category_id AS category_cd,
          s.wcs_user_sk AS customer_id
     FROM web_clickstreams s INNER JOIN item i
       ON s.wcs_item_sk = i_item_sk
    WHERE s.wcs_item_sk IS NOT NULL
      AND i.i_category_id IS NOT NULL
      AND s.wcs_user_sk IS NOT NULL);

SELECT *
  FROM cfilter (ON
       (SELECT 1)
       PARTITION BY 1
       DATABASE ('benchmark')
       USERID ('benchmark')
       PASSWORD ('benchmark')
       INPUTTABLE ('benchmark.c_affinity_input')
       OUTPUTTABLE ('c_affinity_out')
       DROPTABLE ('true')
       INPUTCOLUMNS ('category_cd')
       JOINCOLUMNS ('customer_id'));

SELECT *
  FROM c_affinity_out;
```

```
DROP VIEW IF EXISTS c_affinity_input;
DROP TABLE IF EXISTS c_affinity_out;
```

Listing 1.30. Query 30

B BigBench Schema

Below is the complete schema definition for BigBench in Teradata Aster DBMS syntax.

```
DROP TABLE IF EXISTS customer_simple;
DROP TABLE IF EXISTS customer_addr_simple;
DROP TABLE IF EXISTS inventory_simple;
DROP TABLE IF EXISTS item_simple;
DROP TABLE IF EXISTS store_sales_simple;
DROP TABLE IF EXISTS store_returns_simple;
DROP TABLE IF EXISTS web_sales_simple;
DROP TABLE IF EXISTS web_returns_simple;


DROP TABLE IF EXISTS customer cascade;
DROP TABLE IF EXISTS customer_address cascade;
DROP TABLE IF EXISTS customer_demographics cascade;
DROP TABLE IF EXISTS date_dim cascade;
DROP TABLE IF EXISTS dbgen_version cascade;
DROP TABLE IF EXISTS household_demographics cascade;
DROP TABLE IF EXISTS income_band cascade;
DROP TABLE IF EXISTS item cascade;
DROP TABLE IF EXISTS promotion cascade;
DROP TABLE IF EXISTS reason cascade;
DROP TABLE IF EXISTS ship_mode cascade;
DROP TABLE IF EXISTS store cascade;
DROP TABLE IF EXISTS time_dim cascade;
DROP TABLE IF EXISTS warehouse cascade;
DROP TABLE IF EXISTS web_site cascade;
DROP TABLE IF EXISTS web_page cascade;
DROP TABLE IF EXISTS inventory cascade;
DROP TABLE IF EXISTS store_sales cascade;
DROP TABLE IF EXISTS store_returns cascade;
DROP TABLE IF EXISTS web_sales cascade;
DROP TABLE IF EXISTS web_returns cascade;

CREATE TABLE dbgen_version (
  dv_version       VARCHAR(16),
  dv_create_date   date,
  dv_create_time   time,
  dv_cmdline_args  VARCHAR(200)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE customer_demographics (
  cd_demo_sk            BIGINT NOT NULL,
  cd_gender             CHAR(1),
  cd_marital_status     CHAR(1),
  cd_education_status   CHAR(20),
  cd_purchase_estimate  INTEGER,
  cd_credit_rating      CHAR(10),
  cd_dep_count          INTEGER,
  cd_dep_employed_count INTEGER,
  cd_dep_college_count  INTEGER,
  PRIMARY KEY (cd_demo_sk)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE date_dim (
```

```sql
  d_date_sk           BIGINT   NOT NULL,
  d_date_id           CHAR(16) NOT NULL,
  d_date              DATE,
  d_month_seq         INTEGER,
  d_week_seq          INTEGER,
  d_quarter_seq       INTEGER,
  d_year              INTEGER,
  d_dow               INTEGER,
  d_moy               INTEGER,
  d_dom               INTEGER,
  d_qoy               INTEGER,
  d_fy_year           INTEGER,
  d_fy_quarter_seq    INTEGER,
  d_fy_week_seq       INTEGER,
  d_day_name          CHAR(9),
  d_quarter_name      CHAR(6),
  d_holiday           CHAR(1),
  d_weekend           CHAR(1),
  d_following_holiday CHAR(1),
  d_first_dom         INTEGER,
  d_last_dom          INTEGER,
  d_same_day_ly       INTEGER,
  d_same_day_lq       INTEGER,
  d_current_day       CHAR(1),
  d_current_week      CHAR(1),
  d_current_month     CHAR(1),
  d_current_quarter   CHAR(1),
  d_current_year      CHAR(1),
  PRIMARY KEY (d_date_sk)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE warehouse (
  w_warehouse_sk    BIGINT   NOT NULL,
  w_warehouse_id    CHAR(16) NOT NULL,
  w_warehouse_name  VARCHAR(20),
  w_warehouse_sq_ft INTEGER,
  w_street_number   CHAR(10),
  w_street_name     VARCHAR(60),
  w_street_type     CHAR(15),
  w_suite_number    CHAR(10),
  w_city            VARCHAR(60),
  w_county          VARCHAR(30),
  w_state           CHAR(2),
  w_zip             CHAR(10),
  w_country         VARCHAR(20),
  w_gmt_offset      DECIMAL(5,2),
  PRIMARY KEY (w_warehouse_sk)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE ship_mode (
  sm_ship_mode_sk BIGINT   NOT NULL,
  sm_ship_mode_id CHAR(16) NOT NULL,
  sm_type         CHAR(30),
  sm_code         CHAR(10),
  sm_carrier      CHAR(20),
  sm_contract     CHAR(20),
  PRIMARY KEY (sm_ship_mode_sk)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE time_dim (
  t_time_sk   BIGINT   NOT NULL,
  t_time_id   CHAR(16) NOT NULL,
  t_time      INTEGER,
  t_hour      INTEGER,
  t_minute    INTEGER,
  t_second    INTEGER,
  t_am_pm     CHAR(2),
  t_shift     CHAR(20),
```

```sql
  t_sub_shift CHAR(20),
  t_meal_time CHAR(20),
  PRIMARY KEY (t_time_sk)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE reason (
  r_reason_sk   BIGINT   NOT NULL,
  r_reason_id   CHAR(16) NOT NULL,
  r_reason_desc CHAR(100),
  PRIMARY KEY (r_reason_sk)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE income_band (
  ib_income_band_sk BIGINT NOT NULL,
  ib_lower_bound    INTEGER,
  ib_upper_bound    INTEGER,
  PRIMARY KEY (ib_income_band_sk)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE store (
  s_store_sk         BIGINT   NOT NULL,
  s_store_id         CHAR(16) NOT NULL,
  s_rec_start_date   DATE,
  s_rec_end_date     DATE,
  s_closed_date_sk   BIGINT,
  s_store_name       VARCHAR(50),
  s_number_employees INTEGER,
  s_floor_space      INTEGER,
  s_hours            CHAR(20),
  s_manager          VARCHAR(40),
  s_market_id        INTEGER,
  s_geography_class  VARCHAR(100),
  s_market_desc      VARCHAR(100),
  s_market_manager   VARCHAR(40),
  s_division_id      INTEGER,
  s_division_name    VARCHAR(50),
  s_company_id       INTEGER,
  s_company_name     VARCHAR(50),
  s_street_number    VARCHAR(10),
  s_street_name      VARCHAR(60),
  s_street_type      CHAR(15)
  s_suite_number     CHAR(10),
  s_city             VARCHAR(60),
  s_county           VARCHAR(30),
  s_state            CHAR(2),
  s_zip              CHAR(10),
  s_country          VARCHAR(20),
  s_gmt_offset       DECIMAL(5,2),
  s_tax_precentage   DECIMAL(5,2),
  PRIMARY KEY (s_store_sk)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE web_site (
  web_site_sk               BIGINT   NOT NULL,
  web_site_id               CHAR(16) NOT NULL,
  web_rec_start_date        DATE,
  web_rec_end_date          DATE,
  web_name                  VARCHAR(50),
  web_open_date_sk          BIGINT,
  web_close_date_sk         BIGINT,
  web_class                 VARCHAR(50),
  web_manager               VARCHAR(40),
  web_mkt_id                INTEGER,
  web_mkt_class             VARCHAR(50),
  web_mkt_desc              VARCHAR(100),
  web_market_manager        VARCHAR(40),
  web_company_id            INTEGER,
  web_company_name          CHAR(50),
```

```sql
  web_street_number         CHAR(10),
  web_street_name           VARCHAR(60),
  web_street_type           CHAR(15),
  web_suite_number          CHAR(10),
  web_city                  VARCHAR(60),
  web_county                VARCHAR(30),
  web_state                 CHAR(2),
  web_zip                   CHAR(10),
  web_country               VARCHAR(20),
  web_gmt_offset            DECIMAL(5,2),
  web_tax_percentage        DECIMAL(5,2),
  PRIMARY KEY (web_site_sk)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE household_demographics (
  hd_demo_sk                BIGINT NOT NULL,
  hd_income_band_sk         BIGINT,
  hd_buy_potential          CHAR(15),
  hd_dep_count              INTEGER,
  hd_vehicle_count          INTEGER,
  PRIMARY KEY (hd_demo_sk)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE web_page (
  wp_web_page_sk            BIGINT   NOT NULL,
  wp_web_page_id            CHAR(16) NOT NULL,
  wp_rec_start_date         DATE,
  wp_rec_end_date           DATE,
  wp_creation_date_sk       BIGINT,
  wp_access_date_sk         BIGINT,
  wp_autogen_flag           CHAR(1),
  wp_customer_sk            BIGINT,
  wp_url                    VARCHAR(100),
  wp_type                   CHAR(50),
  wp_char_count             INTEGER,
  wp_link_count             INTEGER,
  wp_image_count            INTEGER,
  wp_max_ad_count           INTEGER,
  PRIMARY KEY (wp_web_page_sk)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE promotion (
  p_promo_sk                BIGINT   NOT NULL,
  p_promo_id                CHAR(16) NOT NULL,
  p_start_date_sk           BIGINT,
  p_end_date_sk             BIGINT,
  p_item_sk                 BIGINT,
  p_cost                    DECIMAL(15,2),
  p_response_target         INTEGER,
  p_promo_name              CHAR(50),
  p_channel_dmail           CHAR(1),
  p_channel_email           CHAR(1),
  p_channel_catalog         CHAR(1),
  p_channel_tv              CHAR(1),
  p_channel_radio           CHAR(1),
  p_channel_press           CHAR(1),
  p_channel_event           CHAR(1),
  p_channel_demo            CHAR(1),
  p_channel_details         VARCHAR(100),
  p_purpose                 CHAR(15),
  p_discount_active         CHAR(1),
  PRIMARY KEY (p_promo_sk)
  ) DISTRIBUTE BY REPLICATION;

CREATE TABLE customer (
  c_customer_sk             BIGINT   NOT NULL,
  c_customer_id             CHAR(16) NOT NULL,
  c_current_cdemo_sk        BIGINT,
```

```sql
  c_current_hdemo_sk        BIGINT,
  c_current_addr_sk         BIGINT,
  c_first_shipto_date_sk    BIGINT,
  c_first_sales_date_sk     BIGINT,
  c_salutation              CHAR(10),
  c_first_name              CHAR(20),
  c_last_name               CHAR(30),
  c_preferred_cust_flag     CHAR(1),
  c_birth_day               INTEGER,
  c_birth_month             INTEGER,
  c_birth_year              INTEGER,
  c_birth_country           VARCHAR(20),
  c_login                   CHAR(13),
  c_email_address           CHAR(50),
  c_last_review_date        CHAR(10),
  PRIMARY KEY (c_customer_sk)
  ) DISTRIBUTE BY HASH (c_customer_sk);

CREATE TABLE customer_address (
  ca_address_sk             BIGINT   NOT NULL,
  ca_address_id             CHAR(16) NOT NULL,
  ca_street_number          CHAR(10),
  ca_street_name            VARCHAR(60),
  ca_street_type            CHAR(15),
  ca_suite_number           CHAR(10),
  ca_city                   VARCHAR(60),
  ca_county                 VARCHAR(30),
  ca_state                  CHAR(2),
  ca_zip                    CHAR(10),
  ca_country                VARCHAR(20),
  ca_gmt_offset             DECIMAL(5,2),
  ca_location_type          CHAR(20),
  PRIMARY KEY (ca_address_sk)
  ) DISTRIBUTE BY HASH (ca_address_sk);

CREATE TABLE inventory (
  inv_date_sk               BIGINT NOT NULL,
  inv_item_sk               BIGINT NOT NULL,
  inv_warehouse_sk          BIGINT NOT NULL,
  inv_quantity_on_hand      INTEGER
  ) DISTRIBUTE BY HASH (inv_item_sk);

CREATE TABLE item (
  i_item_sk                 BIGINT   NOT NULL,
  i_item_id                 CHAR(16) NOT NULL,
  i_rec_start_date          DATE,
  i_rec_end_date            DATE,
  i_item_desc               VARCHAR(200),
  i_current_price           DECIMAL(7,2),
  i_wholesale_cost          DECIMAL(7,2),
  i_brand_id                INTEGER,
  i_brand                   CHAR(50),
  i_class_id                INTEGER,
  i_class                   CHAR(50),
  i_category_id             INTEGER,
  i_category                CHAR(50),
  i_manufact_id             INTEGER,
  i_manufact                CHAR(50),
  i_size                    CHAR(20),
  i_formulation             CHAR(20),
  i_color                   CHAR(20),
  i_units                   CHAR(10),
  i_container               CHAR(10),
  i_manager_id              INTEGER,
  i_product_name            CHAR(50),
  PRIMARY KEY (i_item_sk)
  ) DISTRIBUTE BY HASH (i_item_sk);
```

```
CREATE TABLE store_sales (
  ss_sold_date_sk           BIGINT default 9999999,
  ss_sold_time_sk           BIGINT,
  ss_item_sk                BIGINT NOT NULL,
  ss_customer_sk            BIGINT,
  ss_cdemo_sk               BIGINT,
  ss_hdemo_sk               BIGINT,
  ss_addr_sk                BIGINT,
  ss_store_sk               BIGINT,
  ss_promo_sk               BIGINT,
  ss_ticket_number          BIGINT NOT NULL,
  ss_quantity               INTEGER,
  ss_wholesale_cost         DECIMAL(7,2),
  ss_list_price             DECIMAL(7,2),
  ss_sales_price            DECIMAL(7,2),
  ss_ext_discount_amt       DECIMAL(7,2),
  ss_ext_sales_price        DECIMAL(7,2),
  ss_ext_wholesale_cost     DECIMAL(7,2),
  ss_ext_list_price         DECIMAL(7,2),
  ss_ext_tax                DECIMAL(7,2),
  ss_coupon_amt             DECIMAL(7,2),
  ss_net_paid               DECIMAL(7,2),
  ss_net_paid_inc_tax       DECIMAL(7,2),
  ss_net_profit             DECIMAL(7,2),
  ) DISTRIBUTE BY HASH (ss_item_sk);

CREATE TABLE store_returns (
  sr_returned_date_sk       BIGINT default 9999999,
  sr_return_time_sk         BIGINT,
  sr_item_sk                BIGINT NOT NULL,
  sr_customer_sk            BIGINT,
  sr_cdemo_sk               BIGINT,
  sr_hdemo_sk               BIGINT,
  sr_addr_sk                BIGINT,
  sr_store_sk               BIGINT,
  sr_reason_sk              BIGINT,
  sr_ticket_number          BIGINT NOT NULL,
  sr_return_quantity        INTEGER,
  sr_return_amt             DECIMAL(7,2),
  sr_return_tax             DECIMAL(7,2),
  sr_return_amt_inc_tax     DECIMAL(7,2),
  sr_fee                    DECIMAL(7,2),
  sr_return_ship_cost       DECIMAL(7,2),
  sr_refunded_cash          DECIMAL(7,2),
  sr_reversed_charge        DECIMAL(7,2),
  sr_store_credit           DECIMAL(7,2),
  sr_net_loss               DECIMAL(7,2),
  ) DISTRIBUTE BY HASH (sr_item_sk);

CREATE TABLE web_sales (
  ws_sk                     BIGINT NOT NULL,
  ws_sold_date_sk           BIGINT default 9999999,
  ws_sold_time_sk           BIGINT,
  ws_ship_date_sk           BIGINT,
  ws_item_sk                BIGINT NOT NULL,
  ws_bill_customer_sk       BIGINT,
  ws_bill_cdemo_sk          BIGINT,
  ws_bill_hdemo_sk          BIGINT,
  ws_bill_addr_sk           BIGINT,
  ws_ship_customer_sk       BIGINT,
  ws_ship_cdemo_sk          BIGINT,
  ws_ship_hdemo_sk          BIGINT,
  ws_ship_addr_sk           BIGINT,
  ws_web_page_sk            BIGINT,
  ws_web_site_sk            BIGINT,
  ws_ship_mode_sk           BIGINT,
  ws_warehouse_sk           BIGINT,
  ws_promo_sk               BIGINT,
```

```sql
  ws_order_number           BIGINT NOT NULL,
  ws_quantity               INTEGER,
  ws_wholesale_cost         DECIMAL(7,2),
  ws_list_price             DECIMAL(7,2),
  ws_sales_price            DECIMAL(7,2),
  ws_ext_discount_amt       DECIMAL(7,2),
  ws_ext_sales_price        DECIMAL(7,2),
  ws_ext_wholesale_cost     DECIMAL(7,2),
  ws_ext_list_price         DECIMAL(7,2),
  ws_ext_tax                DECIMAL(7,2),
  ws_coupon_amt             DECIMAL(7,2),
  ws_ext_ship_cost          DECIMAL(7,2),
  ws_net_paid               DECIMAL(7,2),
  ws_net_paid_inc_tax       DECIMAL(7,2),
  ws_net_paid_inc_ship      DECIMAL(7,2),
  ws_net_paid_inc_ship_tax  DECIMAL(7,2),
  ws_net_profit             DECIMAL(7,2),
  ) DISTRIBUTE BY HASH (ws_sk);

CREATE TABLE web_returns (
  wr_returned_date_sk       BIGINT default 9999999,
  wr_returned_time_sk       BIGINT,
  wr_item_sk                BIGINT NOT NULL,
  wr_refunded_customer_sk   BIGINT,
  wr_refunded_cdemo_sk      BIGINT,
  wr_refunded_hdemo_sk      BIGINT,
  wr_refunded_addr_sk       BIGINT,
  wr_returning_customer_sk  BIGINT,
  wr_returning_cdemo_sk     BIGINT,
  wr_returning_hdemo_sk     BIGINT,
  wr_returning_addr_sk      BIGINT,
  wr_web_page_sk            BIGINT,
  wr_reason_sk              BIGINT,
  wr_order_number           BIGINT NOT NULL,
  wr_return_quantity        INTEGER,
  wr_return_amt             DECIMAL(7,2),
  wr_return_tax             DECIMAL(7,2),
  wr_return_amt_inc_tax     DECIMAL(7,2),
  wr_fee                    DECIMAL(7,2),
  wr_return_ship_cost       DECIMAL(7,2),
  wr_refunded_cash          DECIMAL(7,2),
  wr_reversed_charge        DECIMAL(7,2),
  wr_account_credit         DECIMAL(7,2),
  wr_net_loss               DECIMAL(7,2),
  ) DISTRIBUTE BY HASH (wr_item_sk);

DROP TABLE IF EXISTS item_marketprices cascade;
DROP TABLE IF EXISTS web_clickstreams cascade;
DROP TABLE IF EXISTS product_reviews cascade;

CREATE TABLE item_marketprices (
  imp_sk                  BIGINT NOT NULL,
  imp_item_sk             BIGINT NOT NULL,
  imp_competitor          VARCHAR(20),
  imp_competitor_price    DECIMAL(7,2),
  imp_start_date          BIGINT,
  imp_end_date            BIGINT,
  PRIMARY KEY (imp_sk)
  ) DISTRIBUTE BY HASH (imp_sk);

CREATE TABLE web_clickstreams (
  wcs_click_sk            BIGINT NOT NULL,
  wcs_click_date_sk       BIGINT,
  wcs_click_time_sk       BIGINT,
  wcs_sales_sk            BIGINT,
  wcs_item_sk             BIGINT,
  wcs_web_page_sk         BIGINT,
  wcs_user_sk             BIGINT,
```

```
  PRIMARY KEY (wcs_click_sk)
  ) DISTRIBUTE BY HASH (wcs_click_sk);

CREATE TABLE product_reviews (
  pr_review_sk            BIGINT NOT NULL,
  pr_review_date          DATE,
  pr_review_time          CHAR(6),
  pr_review_rating        INT    NOT NULL,
  pr_item_sk              BIGINT NOT NULL,
  pr_user_sk              BIGINT,
  pr_order_sk             BIGINT,
  pr_review_content       TEXT   NOT NULL,
  PRIMARY KEY (pr_review_sk)
  ) DISTRIBUTE BY HASH (pr_review_sk);
```