The Star Schema Benchmark (SSB)

January 2007

Pat O'Neil, Betty O'Neil, Xuedong Chen
{poneil,eoneil,xuedchen}@cs.umb.edu: UMass/Boston

Abstract.

The *Star Schema benchmark*, or *SSB*, was devised to evaluate database system performance of star schema data warehouse queries. The schema for SSB is based on the TPC-H benchmark, but in a highly modified form. We believe the details of modification to be instructive in answering an important question: given a database schema that is not in star schema form, how can it be transformed to star schema form without loss of important query information? The SSB has been used to measure a number of major commercial database products on Linux to evaluate a new product. We also intend to use SSB to compare star schema query performance of three major commercial database products running on Windows, which will be reported separately.

1. Introduction

The Schema of the Star Schema Benchmark (SSB) is based on that of the TPC-H benchmark. The queries are also based on a few of the TPC-H queries, but the number of queries is rather small to make it easy for individuals to run SSB on different platforms. There is no attempt to make SSB bulletproof against intentional misuse by providing an exhaustive list of tuning approaches that are illegal, but a few guidelines are provided in Section 5.

2. Deriving The Star Schema Benchmark

Figure 1 (below) outlines the Schema layout of the TPC-H benchmark, taken from [TPC-H]. We presume the reader is somewhat familiar with TPC-H schema conventions: for example: P_NAME is a column in the PART table, SF stands for the Scale Factor of the benchmark, and the LINEITEM table has 6,000,000 rows in a benchmark with SF=1, but 600,000,000 in a benchmark with SF=100.

Figure 2 (on the same page as Figure 1) outlines the Schema layout of SSB, the Star Schema Benchmark defined here. We explain below the various factors that led us to define the schema for SSB based on the schema of [TPC-H]. ***Definitions of columns of SSB (cardinalities, types, and lengths) are given in Appendix A*** (Q.V.).
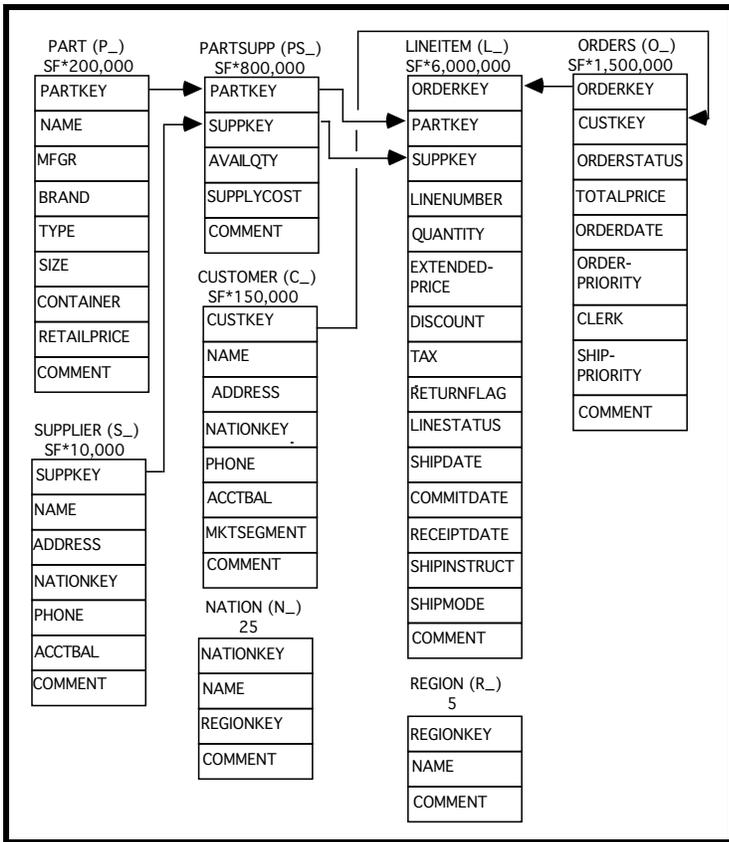
2.1 Achieving a Star Schema and Fact Table Changes

We are guided in the transformations that allow us to derive SSB from TPC-H by principles explained in [KIMBALL02]. Here is a list of transformations made.
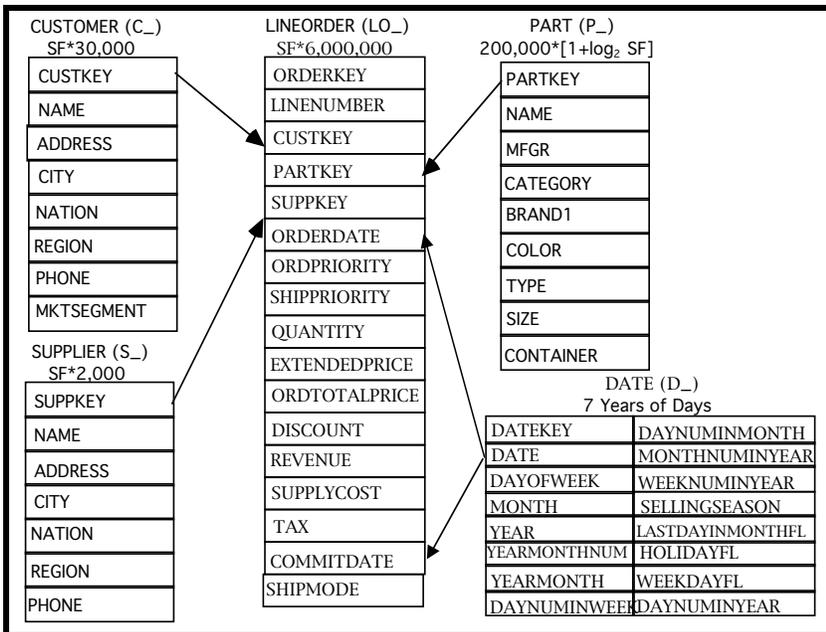
1. **Create SSB LINEORDER Table**. We combine the LINEITEM and ORDER table in SSB to make a LINEORDER table. This denormalization is standard in data warehousing (see [KIMBALL02], page 121), and makes many joins unnecessary in common queries. Look at Appendix A for a reasonably detailed explanation of the columns in LINEORDER and all other SSB tables. We discuss these issues further below.

2. **Drop PARTSUPP Table**. We drop the PARTSUPP table of TPC-H because of a grain mismatch. While TPC-H LINEITEM and ORDER tables (combined in SSB as the LINEORDER table) have the finest Transaction Level grain, the PARTSUPP table has what is called a Periodic Snapshot grain. (These terms are from [KIMBALL02].) Basically, this means that refreshes adding new rows over time to LINEORDER do not add rows to PARTSUPP, which is frozen in time.

Now this is OK as long as PARTSUPP and LINEORDER are treated as SEPARATE FACT TABLES (i.e., separate Data Marts in Kimball terms), queried separately, and never joined together. This is true in most TPC-H Queries where PARTSUPP is in the FROM clause, i.e., in Q1, Q11, Q16 and Q20.; however in Q9, PARTSUPP, ORDERS, and LINEITEM all appear in the FROM clause. Query Q9 calculates profit for a part in a LINEITEM row (identifying PS for

PART (P_)
SF*200,000

PARTKEY
NAME
MFGR
BRAND
TYPE
SIZE
CONTAINER
RETAILPRICE
COMMENT

PARTSUPP (PS_)
SF*800,000

PARTKEY
SUPPKEY
AVAILQTY
SUPPLYCOST
COMMENT

LINEITEM (L_)
SF*6,000,000

ORDERKEY
PARTKEY
SUPPKEY
LINENUMBER
QUANTITY
EXTENDED-PRICE
DISCOUNT
TAX
RETURNFLAG
LINESTATUS
SHIPDATE
COMMITDATE
RECEIPTDATE
SHIPINSTRUCT
SHIPMODE
COMMENT

ORDERS (O_)
SF*1,500,000

ORDERKEY
CUSTKEY
ORDERSTATUS
TOTALPRICE
ORDERDATE
ORDER-PRIORITY
CLERK
SHIP-PRIORITY
COMMENT

CUSTOMER (C_)
SF*150,000

CUSTKEY
NAME
ADDRESS
NATIONKEY
PHONE
ACCTBAL
MKTSEGMENT
COMMENT

SUPPLIER (S_)
SF*10,000

SUPPKEY
NAME
ADDRESS
NATIONKEY
PHONE
ACCTBAL
COMMENT

NATION (N_)
25

NATIONKEY
NAME
REGIONKEY
COMMENT

REGION (R_)
5

REGIONKEY
NAME
COMMENT

Figure 1. TPC-H Benchmark Schema

CUSTOMER (C_)
SF*30,000

CUSTKEY
NAME
ADDRESS
CITY
NATION
REGION
PHONE
MKTSEGMENT

SUPPLIER (S_)
SF*2,000

SUPPKEY
NAME
ADDRESS
CITY
NATION
REGION
PHONE

LINEORDER (LO_)
SF*6,000,000

ORDERKEY
LINENUMBER
CUSTKEY
PARTKEY
SUPPKEY
ORDERDATE
ORDPRIORITY
SHIPPRIORITY
QUANTITY
EXTENDEDPRICE
ORDTOTALPRICE
DISCOUNT
REVENUE
SUPPLYCOST
TAX
COMMITDATE
SHIPMODE

PART (P_)
200,000*[1+log₂ SF]

PARTKEY
NAME
MFGR
CATEGORY
BRAND1
COLOR
TYPE
SIZE
CONTAINER

DATE (D_)
7 Years of Days

DATEKEY	DAYNUMINMONTH
DATE	MONTHNUMINYEAR
DAYOFWEEK	WEEKNUMINYEAR
MONTH	SELLINGSEASON
YEAR	LASTDAYINMONTHFL
YEARMONTHNUM	HOLIDAYFL
YEARMONTH	WEEKDAYFL
DAYNUMINWEEK	DAYNUMINYEAR

Figure 2. Star Schema Benchmark (SSB) Schema

See Appendix A for Definitions of Columns shown here and discussed below!

the given part and supplier) as:

$$[(L\_EXTENDEDPRICE*(1-L\_DISCOUNT))-(PS\_SUPPLYCOST*L\_QUANTITY)]$$

The problem, of course, is that that the PS_SUPPLYCOST could never be expected to remain constant during the seven -year history of accumulating LINEORDER rows. The difference in grain between PARTSUPP and LINEORDER causes this problem.

If we were to ignore Query Q9 (as we do in SSB), we could create a second Data Mart with SUPPLYCOST as the only Fact table, but we don't do this in SSB. The presence of a Snapshot PARTSUPP table in TPC-H design seems of little use in any event, and seems included simply to create a complex join schema; it is very much what one would expect in an update transactional design, where in adding an order lineitem for some part, we would access PARTSUPP to find the minimal cost supplier, and would then correct PS_AVAILQTY after filling the order. All of this would be happening in current time, however, with PS_SUPPLYCOST and PS_AVAILQTY kept up to date. But in the TPC-H benchmark, PS_AVAILQTY is never updated, even during Refresh of ORDERS. In a Star Schema Data Warehouse, it's more reasonable to leave out the PARTSUPP table and perhaps (as we do) create a column SUPPLYCOST for each LINEORDER row for this information. A data warehouse, of course, contains derived data only, so there is no need to normalize to guarantee one fact in one place: the next order for the same part and supplier might not repeat this price, and if some part is not ordered we might lose information on some old price charged, but that's fine with derived data.

3. **Drop Some TPC-H Columns of LINEITEM and ORDER and Add Some to LINEORDER**. **(a)** We drop the COMMENT and SHIPINSTRUCT attributes of LINEITEM (27 chars and 25 chars respectively), and COMMENT of ORDER (49 chars). A warehouse doesn't store unparsed text information in a fact table, since it can't be aggregated and takes significant storage. See [Kimball], pg. 18, first full paragraph.

(b) Similarly we drop LO_CLERK, which seems useful only in operational data, though some abstraction of this information might be included in a data warehouse in a form where a query can return quantitative results.

(c) We add LO_SUPPLYCOST for PART, LO_ORDSUPPLYCOST summing for ORDERS, and bring over O_TOTALPRICE as LO_ORDTOTALPRICE.

(d) We drop the TPC-H attributes for events following the ORDERDATE of an item. I.e.: SHIPDATE, RECEIPTDATE, and RETURNFLAG. Clearly the order information must be queryable prior to shipping, customer receipt and return events many days later. The way such a sequence of dates is normally handled in data warehousing is with a sequence of tables as in [KIMBALL], pg. 94, but we feel these are too complex for our intended benchmark. We retain the COMMITDATE (commit to ship) in SSB, since it is part of the sale negotiation.

4. Drop Tables NATION and REGION. We drop the tables, NATION and REGION, which turn the star schema into a snowflake schema when joined to the SSB dimensions CUSTOMER, PART, and SUPPLIER (see [Kimball], page 55). Such tables, which add joins to TPC-H queries, may be appropriate in an OLTP system to enforce integrity, but not in a warehouse system, where data is cleaned on the way in. It is unlikely that a user will need to browse the NATION or REGION dimension to determine what query to perform, but if this is the case, the tables may exist for browsing purposes without being involved in queries.

2.2 Changes to TPC-H Dimension Tables

5. Further Changes Resulting from Grain Mismatches. As in the mismatch between the snapshot value PS_SUPPLYCOST with seven years of ORDER and LINEITEM rows, we have mismatches in a few dimension columns.

(a) We drop P_RETAILPRICE since the retail price is likely to change too frequently to be held in a dimension (see [KIMBALL], page 20, last paragraph); the price of a part is better determined for a lineorder row many days old as LO_EXTENDEDPRICE/LO_QUANTITY.

(b) We drop C_ACCTBAL, which does not match the grain of LINEORDER (C_ACCTBAL never changes); we also refrain from adding another column to the LINEORDER table to provides the customer account balance at the time the order is placed: we assume that existence of the LINEORDER row implies the customer had sufficient funds (or credit) to place the order, and we don't want to build into our design complex concepts of charges and extended credit, features with policies that might change over time in any event.

6. Dropping, Adding, and Changing Columns.
(a) Shortening Columns. **(i)** We shorten the TPC-H column P_NAME that is unrealistically long (55 bytes, five concatenated "colors"); presumably this length was intended to make the PART table larger and more challenging to query; P_NAME is 22 bytes in SSB (two concatenated "colors"). **(ii)** Although P_MFGR, which is 25 bytes long in TPC-H is not considered two long in [KIMBALL], page 20, we change the values to ["MFGR",M], where M is a random value [1,5], a total of 6 chars, e.g.: "MFGR#2". We make this change only to simplify SSB, so users need not learn complex P_MFGR names.

(b) Dropping Columns. In addition to P_RETAILPRICE and C_ACCTBAL mentioned in Change 5, we also drop P_COMMENT; as with O_COMMENT, we have no use for an unparsed comment in a Data Warehouse query.

(c) Adding Columns and Changing Column Names. TPC-H schema columns all have small cardinalities (numbers of values), with O_ORDERDATE the single exception; thus most possible column predicates result in large filter factors that make index restrictions ineffective. As for O_ORDERDATE, TPC-H predicates restrict this to a minimum of one year, and most restrictions are to five out of seven years. Because of this, major database products running TPC-H create indexes only on primary keys, to aid in joins. We consider these small column cardinalities unrealistic, and make a number of reasonable changes to permit predicates with smaller filter factors.

(i) We change the name P_BRAND in TPC-H to P_CATEGORY in SSB, since the TPC-H P_BRAND only has 25 distinct values, a ridiculously small number. We add a new column, P_BRAND1, with 1000 values, subdividing each P_CATEGORY into 40. (A P_BRAND1 value such as MFGR#5433 rolls up to P_CATEGORY value MFGR#54, which rolls up to P_MFGR value MFGR#5. See [KIMBALL], pg. 21, paragraph 3 for terminology: P_CATEGORY might be 'Hardware Products' and P_BRAND1 the Brand 'Snap-On'.)

(ii) We add S_CITY and C_CITY columns using the first 9 characters of the S_NATION (blank extended if there are fewer than 9) followed by a digit 0-9. This provides CITY columns for SUPPLIER and CUSTOMER with cardinalities 250, where NATION has only 25 values.

(iii) We also add a new column named P_COLOR to use in queries where currently a color must be parsed as a substring from P_NAME.

(d) Changing Scale Sizes of Dimensions. We believe that some of the scale sizes and growth factors of dimension tables in TPC-H are unrealistic, and try to address this.

(i) While PARTS (or PRODUCTS in a typical SALES Date Mart) realistically form a large dimension, they do not grow so fast as to remain in the ratio 2/15 to the number of rows in a large ORDERS table (as they would with SF*200,000 rows in TPC-H). **We change the scaling factor for PARTS in SSB** to $200,000*\text{floor}(1+\log_2 SF)$, giving 200,000 parts for 6,000,000 lineorder rows (SF =1), 400,000 parts when there are 12,000,000 lineorder rows (SF = 2), to 600,000 parts only when there are 24,000,000 lineorder rows (SF = 4), and so on. Note that sublinear scaling is also a feature of the planned benchmark presented in [TPC-DS].

(ii) With SF*150,000 customers and 1,500,000 orders, this means we expect the average customer to place 10 orders in 7 years, an unreasonably small number. **We change the number of customers to SF*30,000**, corresponding to 50 orders in 7 years, about 7 orders a year.

7. Adding Date Dimension. We add a DATE dimension table, standard for a warehouse on sales. Unlike the dropped tables NATION and REGION, this table has a large number of attributes that are valuable in queries, such as DAYOFWEEK, MONTH (name), SELLINGSEASON, etc. For a source of Date columns, see [Kimball] page 39. We leave out Fiscal dates. Note that we keep the DATE dimension in order by date.

Summary of SSB Schema Derivation

The result of these table modifications is a proper star schema Data Mart, with the fact table LINEORDER in the middle and dimension tables for CUSTOMER, PART, SUPPLIER, and DATE. A series of Data Marts tables of the types mentioned in point **3 (b)**, above could easily be constructed, but this is considered too complex a venture for our current simple benchmark.

3. Benchmark Queries

The classic data warehouse queries appearing in SSB select from the lineorder table exactly once (no self-joins or subqueries), with predicate restrictions on dimension table attributes. SSB also has one relatively rare form of query that appears in TPC-H and restricts fact table attributes. In varying the TPC-H query format for SSB, we make a weak attempt to provide *Functional Coverage* and *Selectivity Coverage* features explained in [SETQ].

Functional Coverage. We want to choose benchmark queries that span the tasks performed by Star Schema queries used in commercial systems. We would like to achieve such coverage so as to allow prospective users to derive a performance rating from a weighted subset of SSB queries to match the query workload they expect to use in practice. This aim is quite difficult, however, given the huge number of possible combinations of predicates on columns of a fact table and four (or more) dimension tables. We can only try our best, by choosing queries with different numbers of predicates on dimension attributes (and a few fact column predicates).

Selectivity Coverage. The total number of fact table rows retrieved is determined by the selectivity (i.e., total Filter Factor FF) of restrictions on dimensions. We wish to vary this selectivity from queries where a lot of fact table rows are retrieved--though the data reported out is normally grouped and aggregated, resulting in a smaller number of values--to queries where a small number of rows are retrieved.

One issue that arises in all benchmarks is the disk buffering effect that reduces the number of disk accesses necessary when one SQL statement follows another with an overlap of accessed data. To minimize this effect, the approach we take is to flush the disk buffers between successive queries. While one may argue that flushing buffers is unrealistic, we believe that the extent to which buffering aids from one query to the next asymptotes to zero as the size of the database increases so that queries access more information than can be held in memory buffers, a common commercial situation which we cannot emulate with a low-cost benchmark of limited size such as SSB.

As in the Set Query Benchmark [SETQ], we strive in this benchmark to provide functional coverage (different common types of Star Schema queries) and Selectivity Coverage (varying fractions of the lineitem table that must be accessed to answer the queries). We only have a small number of queries to provide such coverage, but we do our best. Some model queries will be based on the TPC-H query set, but others we feel needed have no counterpart in TPC-H. In any event need to modify base query formats to vary the selectivity, resulting in what we call *Query Flights* below.

In Section 3.1, we provide the definitions of queries we propose to use in SSB. Reporting requirements for SSB and guidelines for running the SSB are covered in Section 3.2.

3.1 Query Definitions

Many queries in TPC-H will not translate into our schema. For example, **TPCQ1** requires knowledge whether items shipped as of a given date were returned. We have decided that our LINEORDER table will only have ordering information, and that other data marts would be needed for shipping, receipt, and return information (see [KIMBALL], pg. 94). Similarly, **TPCQ2** asks for the minimum cost supplier for parts in various regions, which requires the PARTSUPP table (with the questionable assumption that it is up-to-date). **TPCQ3** requires knowledge that an order is unshipped; **TPCQ4** requires knowledge of receipt date by customer. And so on. Only a few queries from TPC-H can be implemented on our SSB scheme with minimal modification (as we see below).

The SSB Query Definitions are given in Figure 3, below, and the filter factors (FF) for the query predicates are given in Figure 4 of the following page.

Query Flight Q1, based on TPC-H query **TPCQ6**, has a restriction on one dimension and two LINEORDER columns, LO_DISCOUNT and LO_QUANTITY. The query measures the revenue increase from eliminating various ranges of discounts in given product order quantity intervals shipped in a given year. Since our LINEORDER table doesn't list shipdate, we replace shipdate by orderdate in the flight. We consider this a minimal change, since all products ordered are later shipped. Query Flight **Q1** has three queries.

Q1.1 has restrictions d_year = 1993, lo_quantity < 25, and lo_discount between 1 and 3. We can calculate the total LINEORDER Filter Factor FF to be $(1/7)*0.5*(3/11) = 0.0194805$. The reader can do the same Filter Factor calculations for restrictions of other queries given, by looking at Appendix A, where column cardinalities are provided. Figure 4 provides a list of FF vs. RestrictionSource calculations for all SSB Queries.

Q1.2 changes restrictions of **Q1.1** to d_yearmonth = 199401, lo_quantity between 26 and 35, lo_discount between 4 and 6. The Filter factor is smaller: $FF = (1/84)*(3/11)*0.2 = 0.00064935$.

Q1.3 changes the restrictions to d_weeknuminyear = 6 and YEAR = 1994, lo_quantity between 36 and 40, lo_discount between 5 and 7 with smaller FF = .000075.

Query Flight Q2 has restrictions on two dimensions. The query compares revenues for certain product classes and suppliers in a certain region, grouped by more restrictive product classes and all years of orders; since TPC-H has no query of this description, we add it here.

Q2.1 has restrictions p_category = 'MFGR#12' and s_region, so lineorder $FF = (1/25)*(1/5) = 1/125$.

Q2.2 changes restrictions of **Q2.1** to: p_brand1 between 'MFGR#2221' and 'MFGR#2228' and s_region to 'ASIA', resulting in the $FF = (1/125)*(1/5) = 1/625$. (Note: p_brand1 is a roll-up of category, which rolls up MFGR.)

Q2.3 changes restriction to: p_brand1 = 'MFGR#2339' and s_region = 'EUROPE'; $FF = (1/1000)*(1/5) = 1/5000$.

Query Flight Q3, based on TPC-H query **TPCQ5**, has restrictions on three dimensions. The query is intended to retrieve total revenue for lineorder transactions within a given region in a certain time period, grouped by customer nation, supplier nation and year.

Q3.1 has restrictions c_region = 'ASIA', s_region 'ASIA', and restricts d_year to a 6-year period , grouped by c_nation, s_nation, and year: LINEORDER $FF = (1/5)*(1/5)*(6/7) = 6/175$.

Q3.2 changes region restrictions to c_nation = 'UNITED STATES' and s_nation = 'UNITED STATES', grouping revenue by customer city, supplier city, and year, so LINEORDER $FF = (1/25)*(1/25)*(6/7) = 6/4375$.

Q3.3 Changes restrictions to c_city and s_city to two cities in 'UNITED KINGDOM' and retrieves revenue grouped by c_city, s_city, d_year;, so LINEORDER $FF = (1/125)*(1/125)*(6/7) = 6/109375$

Q3.4 Changes date restriction to a single month, resulting in $FF = (1/125)*(1/125)*(1/84) = 1/1312500$.

Q1.1
select sum(lo_extendedprice*lo_discount) as revenue
  from lineorder, date
  where lo_orderdate = d_datekey
    and d_year = 1993
    and lo_discount between1 and 3
    and lo_quantity < 25;

Q1.2
select sum(lo_extendedprice*lo_discount) as revenue
  from lineorder, date
  where lo_orderdate = d_datekey
    and d_yearmonth = 199401
    and lo_discount between4 and 6
    and lo_quantity between 26 and 35;

Q1.3
select sum(lo_extendedprice*lo_discount) as revenue
  from lineorder, date
  where lo_orderdate = d_datekey
    and d_weeknuminyear = 6
    and d_year = 1994
    and lo_discount between 5 and 7
    and lo_quantity between 26 and 35;

Q2.1
select sum(lo_revenue), d_year, p_brand1
  from lineorder, date, part, supplier
  where lo_orderdate = d_datekey
    and lo_partkey = p_partkey
    and lo_suppkey = s_suppkey
    and p_category = 'MFGR#12'
    and s_region = 'AMERICA'
  group by d_year, p_brand1
  order by d_year, p_brand1;

Q2.2
select sum(lo_revenue), d_year, p_brand1
  from lineorder, date, part, supplier
  where lo_orderdate = d_datekey
    and lo_partkey = p_partkey
    and lo_suppkey = s_suppkey
    and p_brand1between 'MFGR#2221'
     and 'MFGR#2228'
    and s_region = 'ASIA'
  group by d_year, p_brand1
  order by d_year, p_brand1;

Q2.3
select sum(lo_revenue), d_year, p_brand1
  from lineorder, date, part, supplier
  where lo_orderdate = d_datekey
    and lo_partkey = p_partkey
    and lo_suppkey = s_suppkey
    and p_brand1= 'MFGR#2239'
    and s_region = 'EUROPE'
  group by d_year, p_brand1
  order by d_year, p_brand1;

Q3.1
select c_nation, s_nation, d_year, sum(lo_revenue) as revenue
  from customer, lineorder, supplier, date
  where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and c_region = 'ASIA'
    and s_region = 'ASIA'
    and d_year >= 1992 and d_year <= 1997
    group by c_nation, s_nation, d_year
    order by d_year asc, revenue desc;

Q3.2
select c_city, s_city, d_year, sum(lo_revenue) as revenue
  from customer, lineorder, supplier, date
  where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and c_nation = 'UNITED STATES'
    and s_nation = 'UNITED STATES'
    and d_year >= 1992 and d_year <= 1997
    group by c_city, s_city, d_year
    order by d_year asc, revenue desc;

Q3.3
select c_city, s_city, d_year, sum(lo_revenue) as revenue
  from customer, lineorder, supplier, date
  where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and (c_city='UNITED KI1'
     or c_city='UNITED KI5')
    and (s_city='UNITED KI1'
     or s_city='UNITED KI5')
    and d_year >= 1992 and d_year <= 1997
    group by c_city, s_city, d_year
    order by d_year asc, revenue desc;

Q3.4
select c_city, s_city, d_year, sum(lo_revenue) as revenue
  from customer, lineorder, supplier, date
  where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_orderdate = d_datekey
    and (c_city='UNITED KI1'
     or c_city='UNITED KI5')
    and (s_city='UNITED KI1'
     or s_city='UNITED KI5')
    and d_yearmonth = 'Dec1997'
    group by c_city, s_city, d_year
    order by d_year asc, revenue desc;

Q4.1
select d_year, c_nation, sum(lo_revenue - lo_supplycost) as profit
  from date, customer, supplier, part, lineorder
  where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_partkey = p_partkey
    and lo_orderdate = d_datekey
    and c_region = 'AMERICA'
    and s_region = 'AMERICA'
    and (p_mfgr = 'MFGR#1'
     or p_mfgr = 'MFGR#2')
    group by d_year, c_nation
    order by d_year, c_nation;

Q4.2
select d_year, s_nation, p_category, sum(lo_revenue - lo_supplycost) as profit
  from date, customer, supplier, part, lineorder
  where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_partkey = p_partkey
    and lo_orderdate = d_datekey
    and c_region = 'AMERICA'
    and s_region = 'AMERICA'
    and (d_year = 1997 or d_year = 1998)
    and (p_mfgr = 'MFGR#1'
     or p_mfgr = 'MFGR#2')
    group by d_year, s_nation, p_category
    order by d_year, s_nation, p_category;

Q4.3
select d_year, s_city, p_brand1, sum(lo_revenue - lo_supplycost) as profit
  from date, customer, supplier, part, lineorder
  where lo_custkey = c_custkey
    and lo_suppkey = s_suppkey
    and lo_partkey = p_partkey
    and lo_orderdate = d_datekey
    and s_nation = 'UNITED STATES'
    and (d_year = 1997 or d_year = 1998)
    and p_category = 'MFGR#14'
    group by d_year, s_city, p_brand1;
    order by d_year, s_city, p_brand1;

Figure 3. Star Schema Benchmark Query List

Query Flight Q4 provides a "What-If" sequence of queries that might be generated in an OLAP style of exploration. Starting with a query with rather weak constraints on three-dimensional columns, we retrieve aggregate profit, sum(lo_revenue - lo_supplycost), grouped by d_year and c_nation. Successive queries modify predicate constraints by drilling down to find the source of an anomaly.

    Q4.1 restricts c_region and s_region both to 'AMERICA', and p_mfgr to one of two possibilities, so FF on lineorder = (1/5)(1/5)*(2/5) = 2/125.

    Assume that **Q4.1** output shows a surprising growth of 40% in profit from year 1997 to year 1998, uniform across c_nation. (This need not be true in the data we actually examine.) We would probably want **Q4.2** to pivot away from grouping by s_nation, restrict d_year to 1997 and 1998, and drill down to group by p_category to see where the profit change arises.

Q4.2 thus has filter factor FF = (2/7)*(1/5)(1/5)*(2/5) = 4/875.

Now assume that as a result of **Q4.2**, a great percentage of the profit increase from year 1997 to 1998 comes from s_nation = 'UNITED STATES' and p_category = 'MFGR1#4'. Now in **Q4.3** we might want to restrict s_nation to 'UNITED STATES' and p_category = 'MFGR1#4', drilling down to group by s_city (in the United States) and p_brand1 (within p_category'MFGR#14').

Q4.3 thus has FF = (2/7)*(1/5)*(1/25)*(1/25) = 2/21875.

3.2 A Discussion of SSB Query Optimization

Figure 4 contains Filter Factors (FFs) for SSB Query predicates, categorized by LINEORDER column predicates or Dimensional column predicates from which they spring. Ultimately, FFs of each query line combine to restrict the number of rows retrieved from the LINEORDER table. An algorithm for optimizing a query with dimension column

Query	FF lineorder restriction	Dimensions: FFs of indexable predicates on dimension columns				Combined FF Effect on lineorder
		FF on time	FF on part: Brand1 roll-up	FF on supplier: city roll-up	FF customer: city roll-up	
Q1.1	.47*3/11	1/7				.019
Q1.2	.2*3/11	1/84				.00065
Q1.3	.1*3/11	1/364				.000075
Q2.1			1/25	1/5		1/125 = .0080
Q2.2			1/125	1/5		1/625 = .0016
Q2.3			1/1000	1/5		1/5000 = .00020
Q3.1		6/7		1/5	1/5	6/175 = .034
Q3.2		6/7		1/25	1/25	6/4375 = .0014
Q3.3		6/7		1/125	1/125	6/109375 =.000055
Q3.4		1/84		1/125	1/125	1/1312500= .000000762
Q4.1			2/5	1/5	1/5	2/125 = .016
Q4.2		2/7	2/5	1/5	1/5	4/875 = .0046
Q4.3		2/7	1/25	1/25	1/5	2/21875 = .000091

Figure 4. Filter Factor Breakdown for SSB Queries of Figure 3

restrictions is outlined in [SCALZO], page 90. The query plan translates each restriction on a dimension to a set of primary keys of rows in that dimension table, then translates from these primary keys to LINEORDER foreign keys for the dimension, and finally ORs the set of LINEORDER rows for each of these indexed foreign key values to achieve a single restriction on LINEORDER, usually represented as a bitmap. The restrictions on all dimensions of the query are then ANDed with any indexed restrictions on LINEORDER columns to result in a combined FF shown in the final column of Figure 4. This restricted set of LINEORDER rows is then joined with dimension columns from the Select list of the query. If the combined FF effect on LINEORDER is very small (as in **Q3.4**), the total effort to retrieve the joined rows is much reduced.

A FF of 1/1000 would have saved a great deal of time retrieving disk pages from the LINEORDER table on MVS DB2: assuming the table had 40 rows on each disk page, we would have only needed to retrieve 40/1000 = 1/25 of all the pages. While sequential access (reading in all disk pages in sequence) was ten times faster on a per page basis than list prefetch (which would be used to pick up an average of 1/25 of all pages in sequence in the most efficient way), that still meant that we could access 1/25 of all pages in only 10/25 = 40% of the time it took to access all pages in sequence. Times have changed, however. Currently, sequential access from a disk is much faster than it used to be compared to random I/O to retrieve 1/25 of the pages.

An indexed restriction on a table with most given FFs is thus of less value than it used to be. The TPC-H benchmark specifically has no columns whose restrictions can lead to small filter factors, so in fact only primary keys and foreign keys are indexed.

3.2 SSB Reporting Requirements

NOTE 1: published SSB Reports need not include information precluding anonymous DBMS references to avoid license restrictions on published performance measurements for specific products. This exemption includes: product-specific tuning details, query plans, and any other facts that would reveal the name of the DBMS product.

Aside from exemptions of **NOTE 1**, SSB Reports should include a Scale Factor rating (and database product name, if not anonymous) in the heading, as well as the following information: the processor model, DBMS name(s) and versions,

memory space broken down by use (e.g., disk buffers, sort space), disk setup, number of processors being used in the test with breakdown of schema by processor, and any other parameters of the system that impinge on performance.

After a load on a DBMS (see below for data generation), the space utilization of all tables, indexes, materialized views, and any other objects that incur space utilization will be listed. The purpose of any defined object used for performance acceleration (such as a materialized view) will be clearly explained. The time expended for all data loads, including sorting data, partitioning or clustering the data, creating and loading indexes (indexes are considered part of the data), generating statistics on the data for query optimization, and any other procedures used to bring the data to a state where efficient query performance can be measured, will be reported in a clear manner.

The 13 queries **Q1.1** through **Q4.3** listed in Figure 3 will be executed, and performance reported will be: Elapsed time, CPU time, and disk volume of I/O (in MBytes or Disk blocks with stated size). Buffers will be flushed between successive queries: more on this in Section 3.3.

3.3 Benchmarking Rules/Guidelines

The authors are not attempting to make the SSB benchmark bulletproof by listing tuning approaches that are illegal (as is done in TPC benchmarks). However, any tuning capability habitually used to improve performance in a database product should be adopted for that product when it was measured and reported for SSB. If a tuning method should prove impossible because of cost limitations in the benchmarking setup, the tuning method that was not adopted and the reason for it should be mentioned in the preface to the report with a bold heading. Note that lack of disk space would normally not be an acceptable reason, since one is expected to buy additional disk to accommodate efficient database product execution; CPU parallelism across multiple processors would be an acceptable exemption, since a single-processor benchmarks should be appropriate for measurement.

Here are a few ground rules. First, the columns in the SSB tables can be compressed by whatever means available in the database system used, as long as reported data retrieved by queries has the values specified in our schemas: e.g., we report values: Monday, Tuesday, . . ., Sunday, rather than 1, 2,..., 7.

Second, Materialized Views, or *MVs*, that might enhance performance, for example by pre-joining some useful dimension columns with the LINEORDER table, are permitted. The point of this is to avoid artificial restrictions that would limit performance unrealistically compared to commercial systems, which have no such restrictions. Of course such MVs will pay a price in longer load time, and it should be noted that the small number of queries contained in SSB would not be the only ones that would occur in an ad-hoc data warehouse query application. However, the queries of SSB can be thought of as commonly occurring in such an application.

Disk buffers must be flushed between queries. This can be done by any means desired (e.g., long table scan), but a test that the flush was effective should be made. This can be achieved by validating that query measurements Q1.1, Q2.1, Q3.1, Q3.4 and Q4.1 remain the same after bringing down the processor and executing the queries on a fresh start-up.

Variant query forms are allowed. Any alternative SQL form that modifies predicate restrictions but retains the same effect on retrieval is fine. For example, it is permissible to leave out a join for a column that has been added to LINEORDER in a Materialized View; similarly, one can provide extra information to a query optimizer that doesn't understand dimension hierarchy roll-ups by adding restrictions upper hierarchy restrictions that reinforce restrictions on lower dimension columns in a hierarchy. For example, one can add a D_YEAR = 1997 restriction to a D_YEARMONTH = 'Dec1997' restriction. A word of caution, however: on a basic SSB Schema with no materialized view, a modified query **3.4** that added the restriction C_NATION = 'UNITED KINGDOM' to the restriction (C_CITY ='UNITED KI1' or C_CITY ='UNITED KI5') caused one of our DBMS products to choose an inferior query plan; the query optimizer erroneously assumed that these two restrictions were independent, and multiplied their filter factors rather than looking for another restriction to reduce the filter factor selectivity.

The underlined FF for each query distinguishes the smallest FF over the indexable dimension column predicate. The most valuable way we can speed up a query with an indexable dimension column restriction is to sort the lineorder table by that column; otherwise, indexes on such columns will probably not limit the number of disk pages that must be accessed. Note that by breaking ties for underlining away from supplier, we can avoid underlines in the supplier city roll-up column in Table 3.1. Thus we can avoid a lineorder sort by s_city. The query set suggests sorts by time, part brand roll-up and (customer roll-up, supplier roll-up).

We see that Q4 shifts from customer-sort to part-sort as best match between Q4.1 and Q4.3. (NOTE that the DB Designer could either choose the supplier roll-up or customer roll-up for a sort order to provide efficiency where they have equal Filter Factors; however the DB Designer should NOT require both sort orders.)

4. Load and Refresh

There is a DBGEN load provided with SSB Specification; it works pretty much as specified in TPC-H, but with data modifications as needed. It will be documented separately.

Refresh (Insert and Delete multiple LINEORDER rows) will also follow TPC-H to reflect accumulated changes.(One one-thousandth of the lineorder table will be deleted and one one-thousandth inserted with each refresh, with the original lineorder table coming back into existence after 1000 refresh pairs.) As with TPC-H, we allow inserts and deletes while queries are running or while queries are quiesced. Refresh is likely to affect What-If analysis query sets if queries are ongoing.

Appendix A. TPC-H and SSB Column Definitions

TPC-H Column Definitions

PART Table Layout
PARTKEY identifier SF*200,000 are populated
NAME variable text, size 55
MFGR fixed text, size 25
BRAND fixed text, size 10
TYPE variable text, size 25
SIZE integer
CONTAINER fixed text, size 10
RETAILPRICE decimal
COMMENT variable text, size 23
Primary Key: PARTKEY

SUPPLIER Table Layout
SUPPKEY identifier SF*10,000 are populated
NAME fixed text, size 25
ADDRESS variable text, size 40
NATIONKEY identifier foreign key reference to NATIONKEY
PHONE fixed text, size 15
ACCTBAL decimal
COMMENT variable text, size 101
Primary Key: SUPPKEY

PARTSUPP Table Layout
PARTKEY identifier foreign key reference to PARTKEY
SUPPKEY identifier foreign key reference to SUPPKEY
AVAILQTY integer
SUPPLYCOST decimal
COMMENT variable text, size 199
Compound Primary Key: PARTKEY, SUPPKEY

CUSTOMER Table Layout
CUSTKEY identifier SF*150,000 are populated
NAME variable text, size 25
ADDRESS variable text, size 40
NATIONKEY identifier foreign key reference to PNATIONKEY
PHONE fixed text, size 15
ACCTBAL decimal
MKTSEGMENT fixed text, size 10
COMMENT variable text, size 117
Primary Key: CUSTKEY

ORDERS Table Layout
ORDERKEY identifier SF*1,500,000 are sparsely populated
CUSTKEY identifier foreign key reference to C_CUSTKEY
ORDERSTATUS fixed text(1)
TOTALPRICE decimal
ORDERDATE date
ORDERPRIORITY fixed text(15)
CLERK fixed text, size 15
SHIPPRIORITY integer
COMMENT variable text, size 79
Primary Key: ORDERKEY

Comment: Orders are not present for all customers. The orders are assigned at random to two-thirds of the customers, leaving one-third with no order. The purpose is to exercise capabilities of the DBMS to handle "dead data" when joining two or more tables.

LINEITEM Table Layout
ORDERKEY identifier foreign key reference to O_ORDERKEY
PARTKEY identifier foreign key reference to P_PARTKEY,
SUPPKEY identifier foreign key reference to S_SUPPKEY,
[Compound Foreign key reference to (PS_PARTKEY,PS_SUPPKEY)]
LINENUMBER integer
QUANTITY decimal
EXTENDEDPRICE decimal
DISCOUNT decimal
TAX decimal
RETURNFLAG fixed text, size 1
LINESTATUS fixed text, size 1
SHIPDATE date
COMMITDATE date
RECEIPTDATE date
SHIPINSTRUCT fixed text, size 25
SHIPMODE fixed text, size 10
COMMENT variable text size 44
Compound Primary Key: ORDERKEY, LINENUMBER

NATION Table Layout
NATIONKEY identifier 25 nations are populated
NAME fixed text, size 25
REGIONKEY identifier foreign key reference to R_REGIONKEY
COMMENT variable text, size 152
Primary Key: NATIONKEY

REGION Table Layout
REGIONKEY identifier 5 regions are populated
NAME fixed text, size 25
COMMENT variable text, size 152
Primary Key: REGIONKEY

SSB Column Definitions

LINEORDER Table Layout (SF*6,000,000 are populated)
ORDERKEY numeric (int up to SF 300) first 8 of each 32 keys used
LINENUMBER numeric 1-7
CUSTKEY numeric identifier foreign key reference to C_CUSTKEY
PARTKEY identifier foreign key reference to P_PARTKEY
SUPPKEY numeric identifier foreign key reference to S_SUPPKEY
ORDERDATE identifier foreign key reference to D_DATEKEY
ORDERPRIORITY fixed text, size 15 (5 Priorities: 1-URGENT, etc.)
SHIPPRIORITY fixed text, size 1
QUANTITY numeric 1-50 (for PART)
EXTENDEDPRICE numeric, MAX about 55,450 (for PART)
ORDTOTALPRICE numeric, MAX about 388,000 (for ORDER)
DISCOUNT numeric 0-10 (for PART) -- (Represents PERCENT)
REVENUE numeric (for PART: (extendedprice*(100-discount))/100)
SUPPLYCOST numeric (for PART, cost from supplier, max = ?)
TAX numeric 0-8 (for PART)
COMMITDATE Foreign Key reference to D_DATEKEY
SHIPMODE fixed text, size 10 (Modes: REG AIR, AIR, etc.)
Compound Primary Key: ORDERKEY, LINENUMBER

Comment: As in TPC-H, Orders are not present for all customers. The orders are assigned at random to two-thirds of the customers.

PART Table Layout (200,000*⌊1+log$_2$SF⌋ populated)
PARTKEY identifier
NAME variable text, size 22 (Not unique per PART but never was)
MFGR fixed text, size 6 (MFGR#1-5, CARD = 5)
CATEGORY fixed text, size 7 ('MFGR#'‖1-5‖1-5: CARD = 25)
BRAND1 fixed text, size 9 (CATEGORY‖1-40: CARD = 1000)
COLOR variable text, size 11 (CARD = 94)
TYPE variable text, size 25 (CARD = 150)
SIZE numeric 1-50 (CARD = 50)
CONTAINER fixed text(10) (CARD = 40)
Primary Key: PARTKEY

SUPPLIER Table Layout (SF*10,000 are populated)
SUPPKEY identifier
NAME fixed text, size 25: 'Supplier'‖SUPPKEY
ADDRESS variable text, size 25 (city below)
CITY fixed text, size 10 (10/nation: nation_prefix‖(0-9))
NATION fixed text(15) (25 values, longest UNITED KINGDOM)
REGION fixed text, size 12 (5 values: longest MIDDLE EAST)
PHONE fixed text, size 15 (many values, format: 43-617-354-1222)
Primary Key: SUPPKEY

CUSTOMER Table Layout (SF*30,000 are populated)
CUSTKEY numeric identifier
NAME variable text, size 25 'Customer'‖CUSTKEY

ADDRESS variable text, size 25 (city below)
CITY fixed text, size 10 (10/nation: NATION_PREFIX‖(0-9))
NATION fixed text(15) (25 values, longest UNITED KINGDOM)
REGION fixed text, size 12 (5 values: longest MIDDLE EAST)
PHONE fixed text, size 15 (many values, format: 43-617-354-1222)
MKTSEGMENT fixed text, size 10 (longest is AUTOMOBILE)
Primary Key: CUSTKEY
DATE Table Layout (7 years of days: 7366 days)
DATEKEY identifier, unique id -- e.g. 19980327 (what we use)
DATE fixed text, size 18, longest: December 22, 1998
DAYOFWEEK fixed text, size 8, Sunday, Monday, ..., Saturday)
MONTH fixed text, size 9: January, ..., December
YEAR unique value 1992-1998
YEARMONTHNUM numeric (YYYYMM) -- e.g. 199803
YEARMONTH fixed text, size 7: Mar1998 for example
DAYNUMINWEEK numeric 1-7
DAYNUMINMONTH numeric 1-31
DAYNUMINYEAR numeric 1-366
MONTHNUMINYEAR numeric 1-12
WEEKNUMINYEAR numeric 1-53
SELLINGSEASON text, size 12 (Christmas, Summer,...)
LASTDAYINWEEKFL 1 bit
LASTDAYINMONTHFL 1 bit
HOLIDAYFL 1 bit
WEEKDAYFL 1 bit
Primary Key: DATEKEY

References

[KIMROSS] Ralph Kimball and Margy Ross, "The Data Warehouse Toolkit", Second Edition, Wiley, 2002.

[SCALZO] Bert Scalzo, "Oracle DBA Guide to Data Warehousing and Star Schemas", Prentice Hall, 2003.

[SETQ] Pat O'Neil, "The Set Query Benchmark", The Benchmark Handbook for Database and Transaction Processing Systems, Jim Gray, Editor, Morgan Kaufmann 1991/1993, pp. 209-245. Download this text from http://www.sigmod.org/dblp/db/books/collections/gray91.html .

[TPC-DS] Meikel Poess, Bryan Smith, Lubor Kollar and Paul Larson, "TPC-DS, Taking Decision Support Benchmarking to the Next Level", ACM SIGMOD 2002, pp. 582-587.

[TPC-H] TPC-H Version 2.4.0 in PDF Form from http://www.tpc.org/tpch/default.asp