Star Schema Benchmark

Revision 3, June 5, 2009

Pat O'Neil, Betty O'Neil, Xuedong Chen

{poneil, eoneil, xuedchen}@cs.umb.edu

UMass/Boston

1. Star Schema Based on TPC-H

This section provides an explanation of design decisions made in creating the *Star Schema benchmark* or *SSB*. The SSB is designed to measure performance of database products in support of classical data warehousing applications, and is based on the TPC-H benchmark [TPC-H], modified in a number of ways explained in this section.

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, the authors are not attempting to make this benchmark bulletproof by listing illegal tuning approaches. However, any product capability used in one product database design to improve performance must be matched in the database design for other products by an attempt to use the same type of capability, assuming such a capability exists and improves performance.

In outline, here are some of the schema changes we use to change the Normalized TPC-H schema (see Figure 1.1) to the efficient star schema form of SSB (see Figure 1.2). Many reasons for these changes are taken from [Kimball], q.v. More detailed explanations of changes will be provided in Section 2.

1. We combine the TPC-H LINEITEM and ORDERS tables into one sales fact table that we name *LINEORDER*. This denormalization is standard in warehousing, as explained in [Kimball], pg. 121, and makes many joins unnecessary in common queries.

2. We drop the PARTSUPP table since it would belong to a different data mart than the ORDERS and LINEITEM information. This is because PARTSUPP has different temporal granularity, as explained in Section 2.1.

3. We drop the comment attribute of a LINEITEM (27 chars), the comment for an order (49 chars), and the shipping instructions for a LINEITEM (25 chars), because a warehouse does not store such information in a fact table (they can't be aggregated, and take significant storage). See [Kimball], pg. 18. Note this change tends

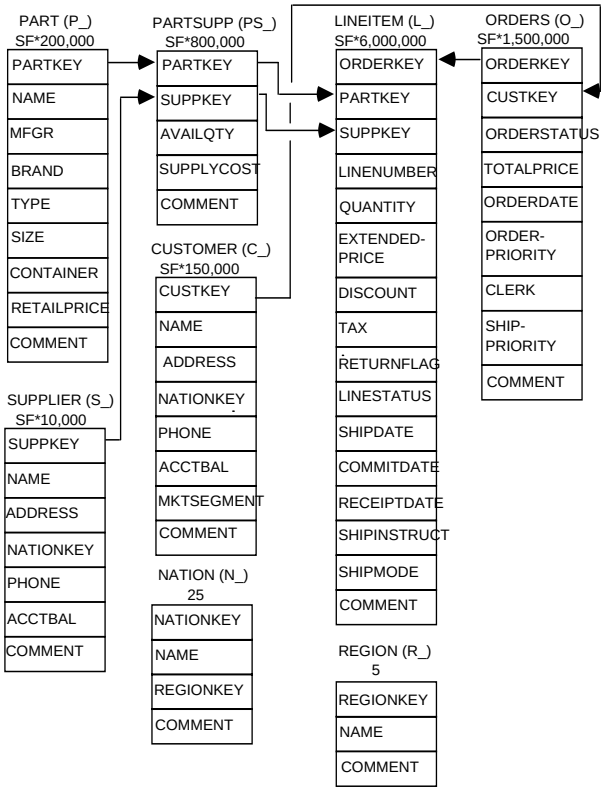

Figure 1.1 TPC-H Schema

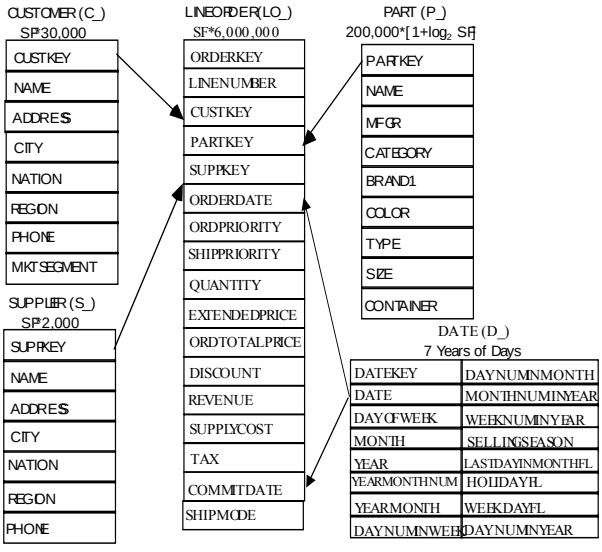

Figure 1.2 SSB Schema

to favor row stores, but is appropriate based on warehouse design principles.

6. We add the DATE dimension table, as is standard for a warehouse on sales.

The result of the table simplifications is a proper star schema data mart, with LINEORDER as a central fact table and dimension tables for customer, part, supplier, and date. A series of tables for shipdate, receiptdate, and returnflag, as mentioned in point 5, above could also be constructed, but would result in too complicated a schema for our simple star schema benchmark.

As regards queries we support in SSBM, we concentrate on queries that select from the LINEORDER table exactly once (no self-joins or subqueries or table queries also involving LINEORDER). The classic warehouse query selects from the fact table with restrictions on the dimension table attributes. We also support queries that appear in TPC-H and restrict on fact table attributes. We depart from the TPC-H query format for a number of reasons, most commonly to make an attempt to provide the *Functional Coverage* and *Selectivity Coverage* features explained in [SETQ].

Functional Coverage. The benchmark queries are chosen as much as possible to span the tasks performed by an important set of Star Schema queries, so that prospective users can derive a performance rating from the weighted subset they expect to use in practice.

It is difficult to provide true functional coverage with a small number of queries, but we at least try to provide queries that have 1, 2, 3, and 4 dimensional restrictions.

Selectivity Coverage. The idea here is that the total number of fact table rows retrieved will be 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 aggregated) to queries where a relatively small number of rows are retrieved.

The SSBM Queries are specified in Section 3.1, and a short analysis showing how multiple sort-orders for LINEORDER will make for efficient queries is provided in Section 3.1.

One other issue arises in running the Star Schema Benchmark queries, and that is the caching effect that reduces the number of disk accesses necessary when query Q2 follows query Q1, because of overlap of data accessed between Q1 and Q2. The approach we will try to take is to minimize this overlap. In situations where this cannot be done, if such arise, we will take whatever steps are needed to reduce caching effects of one query on another.

Reporting requirements for SSBM are covered in Section 5: we will want to report lots of things: query plans, numbers of rows accessed, CPU time in queries, disk I/O, etc.

2. Detail on SSB Format

In this section, we will specify the schemas of the various tables to be used in the Star Schema. Note that in Appendix A, we provide a listing of the original TPC-H tables on which the definitions that follow are based.

2.1 We drop the PARTSUPP table

Here is an argument why this is appropriate, based on principles in [KIMBALL]. The problem is that the LINEITEM and ORDERS tables (combined in SSBM to make a LINEORDER table) have the finest Transaction Level temporal grain, while the PARTSUPP table has a Periodic Snapshot grain. This means that transactions that add new rows over time to LINEORDER do not modify rows in PARTSUPP, which is frozen in time (presumably at the CURRENT date).

This would be fine if PARTSUPP and LINEORDER were treated as SEPARATE FACT TABLES (i.e., separate Data Marts in terms of Kimball), queried separately and not joined together. This is done in all but one of the Queries where PARTSUPP is in the WHERE clause: Q1, Q11, Q16 and Q20, but not in Q9, where PARTSUPP, ORDERS, and LINEITEM all appear. Query Q9 is intended to find, for each nation and year, the profits for certain parts ordered that year. Profit is calculated as sum of [(l_extendedprice*(1 - l_discount) - (ps_supplycost*l_quantity)], and the sum is grouped by the o_orderdate for the LINEITEM columns and the s_nationkey for the part supplied to the order by the PARTSUPP table.

The problem, of course, is that it is beyond the bounds of reason that the ps_supplycost would have remained constant during all these past years. This difference in grain between PARTSUPP and LINEORDER is what causes the problem.

The presence of a Snapshot PARTSUPP table in this design seems suspicious anyway, as if placed there to require a non-trivial normalized join schema; it is very much what we 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, perhaps in some restricted region, and would then correct ps_availqty after filling the order. In the TPC-H benchmark, however, ps_availqty is never updated, not even during the Refresh that inserts new ORDERS. In a Star Schema data warehouse, it's more reasonable to leave out the PARTSUPP table, and create a column supplycost for each LINEORDER Fact row to answer such questions. A data warehouse, of

course, contains derived data only, so there is no reason to normalize to guarantee one fact in one place -- the next order for the same part and supplier might repeat this price, and if we delete the last part of some kind we might lose the price charged, but that's fine since we're trying to simplify queries. In fact, we add the lo_profit column to the LINEORDER table to simplify calculations of this type even further. In general, there are a number of modifications.

See Appendix A for listing of Original TPC-H Table Layouts. Note that all tables in TPC-H and SSB scale from a given size at ***Scale Factor 1*** (**SF = 1**) to 10 times as large (for example) at SF = 10. Typically tables have cardinalities that are multiples of SF (but see the Part table, Section 2.3 in what follows).

2.2 Layout of LINEORDER Fact table.

We combine the LINEITEM and ORDERS tables into one sales fact table that we name *LINEORDER*. This denormalization is standard in warehousing, as explained in [Kimball], pg. 121, and makes many joins unnecessary in common queries. Columns are classified as identifiers (any datatype but unique values for what it is identifying), text (fixed or variable length), and numeric (whole numbers, not floating point.) Numeric identifiers must have unique values and have numeric interpretations which provide unique numbers. Text is in 8-bit ASCII. For numeric columns, the needed range of numbers is indicated.

LINEORDER Table Layout SF*6,000,000
LO_ORDERKEY numeric (int up to SF 300) first 8 of each 32 keys populated
LO_LINENUMBER numeric 1-7
LO_CUSTKEY numeric identifier FK to C_CUSTKEY
LO_PARTKEY identifier FK to P_PARTKEY
LO_SUPPKEY numeric identifier FK to S_SUPPKEY
LO_ORDERDATE identifier FK to D_DATEKEY
LO_ORDERPRIORITY fixed text, size 15 (See pg 91: 5 Priorities: 1-URGENT, etc.)
LO_SHIPPRIORITY fixed text, size 1
LO_QUANTITY numeric 1-50 (for PART)
LO_EXTENDEDPRICE numeric ≤ 55,450 (for PART)
LO_ORDTOTALPRICE numeric ≤ 388,000 (ORDER)
LO_DISCOUNT numeric 0-10 (for PART, percent)
LO_REVENUE numeric (for PART: (lo_extendedprice*(100-lo_discnt))/100)
LO_SUPPLYCOST numeric (for PART)
LO_TAX numeric 0-8 (for PART)
LO_COMMITDATE FK to D_DATEKEY
LO_SHIPMODE fixed text, size 10 (See pg. 91: 7 Modes: REG AIR, AIR, etc.)
Compound Primary Key: LO_ORDERKEY, LO_LINENUMBER

NOTES. (a) We drop all columns in ORDERS and LINEITEMS that make us wait to insert a Fact row after an order is placed on ORDERDATE, For example, we don't want to wait until we know when the order is shipped, when it is received, and whether it is returned before we can query the existence of an order: see pg 96 and 97 of the TPC-H Specification. Thus we drop L_RETURNFLAG, L_LINESTATUS, L_SHIPDATE, L_RECEIPTDATE, and O_ORDERSTATUS. We keep L_COMMITDATE since that is the delivery date promised to the customer at ship time. (b) We drop O_COMMENT (text string [49]), L_COMMENT (text string[27]), and L_SHIPINSTRUCT (text string [25]), since data warehouse queries typically do not parse comments and cannot aggregate them; similarly we drop LO_CLERK (text string[15]); columns such as these are only useful in an operational venue, though some abstraction of this information might well be made available in a data warehouse in a form where a query can return quantitative results. (c) We also add LO_SUPPLYCOST for PART, LO_ORDSUPPLYCOST summing for ORDERS, and bring over O_TOTALPRICE as LO_ORDTOTALPRICE.

2.3 Layout of Part Dimension Table. New cardinality growth relative to SF (logarithmic)

PART Table Layout 200,000*floor(1+$\log_2$SF)
P_PARTKEY identifier
P_NAME variable text, size 22 (Not unique)
P_MFGR fixed text, size 6 (MFGR#1-5, CARD = 5)
P_CATEGORY fixed text, size 7 ('MFGR#'||1-5||1-5: CARD = 25)
P_BRAND1 fixed text, size 9 (P_CATEGORY||1-40: CARD = 1000)
P_COLOR variable text, size 11 (CARD = 94)
P_TYPE variable text, size 25 (CARD = 150)
P_SIZE numeric 1-50 (CARD = 50)
P_CONTAINER fixed text, size 10 (CARD = 40)
Primary Key: P_PARTKEY

NOTES. (a) P_NAME is as long as 55 bytes in TPC-H, which is unreasonably large. We reduce it to 22 by limiting to a concatenation of two colors (see [TPC-H], pg 94). We also add a new column named P_COLOR that could be used in queries where currently a color must be chosen by substring from P_NAME. (b) P_MFGR is fixed text, size 25 in TPC-D; we change the values to ["MFGR",M], where M = random value [1,5], e.g.: "MFGR#2", a total of 6 characters. (c) We add a new column P_CATEGORY as a division of P_MFGR (to take the place of P_BRAND in [TPC-H], which has 25 values, an unreasonably small number of brands; we add a new column P_BRAND1, a division of P_CATEGORY (see [KIMBALL], pg 21, paragraph 3: P_CATEGORY might be 'Paper Products' and

P_BRAND1 is a true Brand such as 'Snap-On'). (d) We drop P_RETAILPRICE (this is likely to change too frequently to be in a dimension; the part price is better determined for an order many days old as LO_EXTENDEDPRICE/LO_QUANTITY. (e) We drop P_COMMENT; as with O_COMMENT, we have no use for an unparsed comment in a data warehouse query. (f) While PARTS (or PRODUCTS) typically form a large dimension, they do not grow so fast that they remain in the ratio 2/15 to the number of rows in a large ORDERS table (as they would with SF*200,000 rows). Thus we change the scaling factor to $200{,}000*\text{floor}(1+\log_2 SF)$. There will be 200,000 parts for 6,000,000 LINEORDER rows (SF =1), jumping to 400,000 parts when there are 12,000,000 LINEORDER rows (SF = 2), to 600,000 parts 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].

2.4 Layout of Supplier Dimension Table.

SUPPLIER Table Layout (SF*2,000 are populated):
S_SUPPKEY numeric identifier
S_NAME fixed text, size 25: 'Supplier'||S_SUPPKEY
S_ADDRESS variable text, size 25 (city below)
S_CITY fixed text, size 10 (10/nation: S_NATION_PREFIX||(0-9)
S_NATION fixed text, size 15 (25 values, longest UNITED KINGDOM)
S_REGION fixed text, size 12 (5 values: longest MIDDLE EAST)
S_PHONE fixed text, size 15 (many values, format: 43-617-354-1222)
Primary Key: S_SUPPKEY

NOTES. (a) We reduce the number of suppliers so as to not have too many suppliers per customer. (b) The S_CITY column is created using the first 9 characters of the S_NATION (blank extended if there are fewer than 9) followed by a digit 0-9. This column is added because there is no other column that can be restricted to result in a reasonably small filter factor, an unnatural situation in real applications.

2.5 Layout of Customer Dimension Table.

CUSTOMER Table Layout (SF*30,000 are populated)
C_CUSTKEY numeric identifier
C_NAME variable text, size 25 'Cutomer'||C_CUSTKEY
C_ADDRESS variable text, size 25 (city below)
C_CITY fixed text, size 10 (10/nation: C_NATION_PREFIX||(0-9)
C_NATION fixed text, size 15 (25 values, longest UNITED KINGDOM)
C_REGION fixed text, size 12 (5 values: longest MIDDLE EAST)
C_PHONE fixed text, size 15 (many values, format: 43-617-354-1222)
C_MKTSEGMENT fixed text, size 10 (longest is AUTOMOBILE)
Primary Key: C_CUSTKEY

NOTES. (a) We drop C_ACCTBAL, which does not match the grain of LINEORDER. (b) 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, or 50 orders in 7 years, about 7 orders a year.

2.6 Layout of (NEW) Date Dimension Table.

DATE Table Layout (7 years of days)
D_DATEKEY identifier, unique id -- e.g. 19980327 (what we use)
D_DATE fixed text, size 18: e.g. December 22, 1998
D_DAYOFWEEK fixed text, size 8, Sunday..Saturday
D_MONTH fixed text, size 9: January, ..., December
D_YEAR unique value 1992-1998
D_YEARMONTHNUM numeric (YYYYMM)
D_YEARMONTH fixed text, size 7: (e.g.: Mar1998
D_DAYNUMINWEEK numeric 1-7
D_DAYNUMINMONTH numeric 1-31
D_DAYNUMINYEAR numeric 1-366
D_MONTHNUMINYEAR numeric 1-12
D_WEEKNUMINYEAR numeric 1-53
D_SELLINGSEASON text, size 12 (e.g.: Christmas)
D_LASTDAYINWEEKFL 1 bit
D_LASTDAYINMONTHFL 1 bit
D_HOLIDAYFL 1 bit
D_WEEKDAYFL 1 bit
Primary Key: D_DATEKEY

NOTES.(a) For source of Date columns, see [Kimball] page 39. We leave out Fiscal dates. (b) Note that we keep the DATE dimension in order by date.

3. Benchmark Queries

As in the Set Query Benchmark [O'NEIL93], 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 flights to use to provide such coverage, but we do our best. Some model queries will be based on the TPC-H query set, but we need to modify these queries to vary the selectivity, resulting in what we call a *Query Flight* below. Other queries that we feel are needed will have no counterpart in TPC-H.

In Section 3.1, we provide the definitions of queries we propose to use in SSBM. Section 3.1 provides a bit of analysis of the benchmark, including an indication of multiple sortorders for LINEITEM that will provide best efficiency.

3.1 Query Definitions

Many queries in TPC-H will not translate into our schema. For example, TPCQ1 requires knowledge of all items shipped as of a given date and whether these items 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 (assuming it's 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 SSBM scheme with minimal modification.

Here are the (Draft) query flights we propose.

Q1. We want to start with a query flight having restrictions on only one dimension. We base Q1 on TPC-H query TPCQ6, which has rather unusual restrictions on the Fact table as well; however the rationale for these Fact table restrictions seems reasonable. The query is meant to quantify the amount of revenue increase that would have resulted from eliminating certain company-wide discounts in a given percentage range for products shipped in a given year. This is a "what if" query to find possible revenue increases. Since our lineorder table doesn't list shipdate, we will replace shipdate by orderdate in the flight.

Q1 select sum(lo_extendedprice*lo_discount) as revenue
from lineorder, date
where lo_orderdate = d_datekey
and d_year = [YEAR] -- Specific values below
and lo_discount between [DISCOUNT] - 1
and [DISCOUNT] + 1 and lo_quantity <
[QUANTITY];

In TPC-H: d_year = [YEAR], random year in [1993..1997] FF = 1/7, lo_quantity < [QUANTITY] a random quantity in [24..25], FF ≈ 47/100, lo_discount value [DISCOUNT] random [2..9], FF = 3/11

In our Q1 Query flight we will restrict lo_quantity, not just to the lower half of the range, but to different ranges with different filter factors. Query flight Q1 has three queries.

Q1.1 YEAR = 1993, DISCOUNT = 2, QUANTITY = 25, so predicates are d_year = 1993, lo_quantity < 25, lo_discount between 1 and 3.

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;

FF = (1/7)*0.5*(3/11) = 0.0194805. Number of lineorder rows selected, for SF = 1, is 0.0194805*6,000,000 ≈ 116,883.

Q1.2 d_yearmonthnum = 199401, lo_quantity between 26 and 35, lo_discount between 4 and 6.

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

FF = (1/84)*(3/11)*0.2 = 0.00064935. Number of lineorder rows selected, for SF = 1: 0.00064935*6,000,000 ≈ 3896.

Q1.3 d_weeknuminyear = 6 and d_year = 1994, lo_quantity between 36 and 40, lo_discount between 5 and 7.

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;

FF = (1/364)*(3/11)*0.1 = .000075. Number of lineorder rows selected, for SF = 1, is .000075*6,000,000 ≈ 450.

NOTE that each of the selections of these three queries is disjoint in lineorder and even in restrictions on columns, so there should be no overlap where caching might make results vary from cold access.

Q2. For a second query flight, we want a query type with restrictions on two dimensions. Our query will compare revenue for some product classes, for 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: p_category = 'MFGR#12', s_region = 'AMERICA'

```
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;
```

p_category = 'MFGR#12', FF = 1/25; s_region, FF=1/5. So LINEORDER FF = (1/25)*(1/5) = 1/125. Number of lineorder rows selected, for SF = 1, is (1/125)*6,000,000 ≈ 48,000

Q2.2 Change p_category = 'MFGR#12' to p_brand1 between 'MFGR#2221' and 'MFGR#2228' and s_region to 'ASIA'.

```
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 between
          'MFGR#2221' and 'MFGR#2228'
    and s_region = 'ASIA'
  group by d_year, p_brand1
  order by d_year, p_brand1;
```

So lineorder FF = (1/125)*(1/5) = 1/625. Number of lineorder rows selected, for SF = 1, is (1/625)*6,000,000 ≈ 9600.

Q2.3 Change p_category = 'MFGR#12' to p_brand1 = 'MFGR#2339' and s_region = 'EUROPE'.

```
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#2221'
     and s_region = 'EUROPE'
  group by d_year, p_brand1
  order by d_year, p_brand1;
```

So lineorder FF = (1/1000)*(1/5) = 1/5000. Number of lineorder rows selected, for SF = 1, is (1/5000)*6,000,000 ≈ 1200. One of the Group By clauses has only one value.

NOTE again, each of the selections of these four queries is disjoint in lineorder and even in restrictions on columns among themselves and also with flight Q1, so there should be no overlap where caching might make results vary from cold access.

Q3. In our third query flight, we want to place restrictions on three dimensions, including the remaining dimension, customer. We base our query on TPCQ5. The query is intended to provide revenue volume for lineorder transactions by customer nation and supplier nation and year within a given region, in a certain time period.

Q3 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;
```

Q3.1 Q3 as written: c_region = 'ASIA' so FF = 1/5 for customer, FF = 1/5 for supplier, and 6-year period FF = 6/7 for d_year; Thus LINEORDER FF = (1/5)*(1/5)*(6/7) = 6/175 and the number of lineorder rows selected, for SF = 1, is (6/175)*6,000,000 ≈ 205,714.

Q3.2 Change restriction to a certain nation, and within that nation, revenue by customer city and supplier city, and year.

```
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;
```

Here the c_nation and s_nation restriction has FF = (1/25); so lineorder FF is (1/25)*(1/25)*(6/7) = 6/4375. The number of lineorder rows selected, for SF = 1, is (6/4375)*6,000,000 ≈ 8,228.

Q3.3 Change restriction to two cities in 'UNITED KINGDOM'; retrieve c_city and group by c_city.

```
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;
```

Here the c_nation and s_nation restriction has FF = (2/10)(1/25)= 1/125; so lineorder FF is (1/125)*(1/125)*(6/7) = 6/109375. The number of lineorder rows selected, for SF = 1, is (6/109375)*6,000,000 ≈ 329.

Q 3.4 Drill down in time to just one month, to create a "needle-in-haystack" query.

```
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;
```

so lineorder FF is (1/125)*(1/125)*(1/84) = 1/1,312,500. The number of lineorder rows selected, for SF = 1, is (1/1,312,500)*6,000,000 ≈ 5.

NOTE again, each of the selections of these queries is disjoint in lineorder and also with flights Q1 and Q2, except for Q3.4 vs. Q 3.3, so there should be no overlap where caching might make results vary from cold access, except for Q3.4.

Q4. The following query flight represents a "What-If" sequence, of the OLAP type. We start with a group by on two dimensions and rather weak constraints on three dimensions, and measure the aggregate profit, measured as (lo_revenue - lo_supplycost).

```
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
```

Q4.1 Query Q4 as written. Restriction on region restriction FFs 1/5 each, p_mfgr restriction 2/5. FF on lineorder = (1/5)(1/5)*(2/5) = 2/125. So the number of lineorder rows selected for SF = 1 is (2/125)*6,000,000 ≈ 96000.

Assume that in Q4.1 output we find 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 to pivot to group by year, s_nation and a further breakdown by p_category to see where the change arises.

```
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
```

This has the same FF as Q4.1 except in time and accesses 2/7 of the same lineorder data; for that data it simply has a different group by dimension breakout. Its FF = (2/7)*(2/125) = 4/875. So the number of lineorder rows selected for SF = 1 is (4/875)*6,000,000 ≈ 27,428.

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 we might want to drill down to cities in the United States and into p_brand1 (within 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 c_region = 'AMERICA'
        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
```

The FF for c_region is 1/5. and for s_nation is 1/25; the FF for d_year remains at 2/7, and the restriction on p_category is now 1/25. Thus the lineorder FF is: (1/5)*(1/25)*(2/7)*(1/25) = 2/21875. The number of

lineorder rows retrieved for SF = 1 is (2/21875)*6,000,000 ≈ 549.

The lineorder rows retrieved by query flight Q4 are disjoint from those of Q1, Q2, and Q3. However successive queries of the Q4 flight retrieve subsets of the rows retrieved in the first flight. This is realistic, however, and measures how well lineorder rows are cached and how efficient the new indexing restrictions can be evaluated.

3.2 Analysis of Queries

Table 3.1 provides Filter Factors (FF) of queries given in Section 3.1, allowing an analysis of the most restrictive indexable dimension column predicates for each query.

Query	FF LINEORDER restriction	Dimensions: FFs of indexable predicates on dimension columns				FF Combined on LINEORDER
		FF time	FF part: brand roll-up	FF 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=.00000076
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

Table 3.1. FF Analysis of Queries in Section 3.1

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 which has an indexable dimension column restriction is to sort the LINEORDER 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.

4. Load and Refresh

There is a DBGEN load provided with SSBM Specification Draft 2; it works pretty much as specified in TPC-H, but with data modifications as specified above. 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 will 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.

5. Performance Measurement

Performance measurement on each DBMS will result in a Report with the name of the DBMS being tested in the title, page numbers, and the following information. First, the processor model, memory space, disk setup, number of processors being used in the test with breakdown of schema by processor, and any other parameter of the system that impinges on performance must be listed.

After a load on a DBMS, the space utilization of all tables, indexes, materialized views, and any other objects that incur space utilization will be listed. The purpose of any object other than a table for performance accelleration will be clearly explained.

The query plan of each of the queries of SSBM will be generated and included in a report.

We will perform all queries, one after another in sequence (this is called a Power Test in TPC-H). For each query, we will list the Query number (e.g., Q3.1), number of rows accessed (do a count in one run), wall clock time to execute, CPU time utilized, and I/O utilization. (We will in at least one run gather CPU time and I/O utilization statistics between queries. This process should be automated to handle multiple measurements after changes in queries, tuning, etc.) We have tried to specify the queries so that memory caching from one query to the next will be minimal, but this will be validated at some point by bringing the system down and starting it up again before executing successive queries.

We also want to think in terms of running the queries on concurrent streams to measure parallelism effects (a Throughput Test in TPC-H). Two streams can run the same sequence to see if inter-query buffer sharing is working properly (piggybacking on each others buffered data). Multiple streams can run sequences that are non-cache-intersecting for a TPCH-like Throughput test.

Appendix A. TPC-H Tables [TPC-H]

PART Table Layout
P_PARTKEY identifier SF*200,000 are populated
P_NAME variable text, size 55
P_MFGR fixed text, size 25
P_BRAND fixed text, size 10
P_TYPE variable text, size 25
P_SIZE integer
P_CONTAINER fixed text, size 10
P_RETAILPRICE decimal
P_COMMENT variable text, size 23
Primary Key: P_PARTKEY

SUPPLIER Table Layout
S_SUPPKEY identifier SF*10,000 are populated
S_NAME fixed text, size 25
S_ADDRESS variable text, size 40
S_NATIONKEY identifier Foreign key reference to N_NATIONKEY
S_PHONE fixed text, size 15
S_ACCTBAL decimal
S_COMMENT variable text, size 101
Primary Key: S_SUPPKEY

PARTSUPP Table Layout
PS_PARTKEY identifier Foreign key reference to P_PARTKEY
PS_SUPPKEY identifier Foreign key reference to S_SUPPKEY
PS_AVAILQTY integer
PS_SUPPLYCOST decimal
PS_COMMENT variable text, size 199
Compound Primary Key: PS_PARTKEY, PS_SUPPKEY

CUSTOMER Table Layout
C_CUSTKEY identifier SF*150,000 are populated
C_NAME variable text, size 25
C_ADDRESS variable text, size 40
C_NATIONKEY identifier Foreign key reference to C_NATIONKEY
C_PHONE fixed text, size 15
C_ACCTBAL decimal
C_MKTSEGMENT fixed text, size 10
C_COMMENT variable text, size 117
Primary Key: C_CUSTKEY

ORDERS Table Layout
O_ORDERKEY identifier SF*1,500,000 are sparsely populated
O_CUSTKEY identifier Foreign key reference to C_CUSTKEY
O_ORDERSTATUS fixed text, size 1
O_TOTALPRICE decimal
O_ORDERDATE date
O_ORDERPRIORITY fixed text, size 15
O_CLERK fixed text, size 15
O_SHIPPRIORITY integer
O_COMMENT variable text, size 79
Primary Key: O_ORDERKEY

Comment: Orders are not present for all customers. In fact, one-third of the customers do not have any order in the database. The orders are assigned at random to two-thirds of the customers (see Clause 4). The purpose of this is to exercise the capabilities of the DBMS to handle "dead data" when joining two or more tables.

LINEITEM Table Layout
L_ORDERKEY identifier Foreign key reference to O_ORDERKEY
L_PARTKEY identifier Foreign key reference to P_PARTKEY, Compound
Foreign Key Reference to (PS_PARTKEY, PS_SUPPKEY) with L_SUPPKEY
L_SUPPKEY identifier Foreign key reference to S_SUPPKEY, Compound
Foreign key reference to (PS_PARTKEY, PS_SUPPKEY) with L_PARTKEY

L_LINENUMBER integer
L_QUANTITY decimal
L_EXTENDEDPRICE decimal
L_DISCOUNT decimal
L_TAX decimal
L_RETURNFLAG fixed text, size 1
L_LINESTATUS fixed text, size 1
L_SHIPDATE date
L_COMMITDATE date
L_RECEIPTDATE date
L_SHIPINSTRUCT fixed text, size 25
L_SHIPMODE fixed text, size 10
L_COMMENT variable text size 44
Compound Primary Key: L_ORDERKEY, L_LINENUMBER

NATION Table Layout
N_NATIONKEY identifier 25 nations are populated
N_NAME fixed text, size 25
N_REGIONKEY identifier Foreign key reference to R_REGIONKEY
N_COMMENT variable text, size 152
Primary Key: N_NATIONKEY

REGION Table Layout
R_REGIONKEY identifier 5 regions are populated
R_NAME fixed text, size 25
R_COMMENT variable text, size 152
Primary Key: R_REGIONKEY

References

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

[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