WHITE PAPER

Performance Comparison Between ASE 15.0 and MySQL 5.0

DHIMANT CHOKSHI
SERVER PERFORMANCE AND ENGINEERING DEVELOPMENT GROUP
SYBASE, INC.

SYBASE®

TABLE OF CONTENTS

1.0 INTRODUCTION

Adaptive Server® Enterprise (ASE) is a highly scalable, mission critical database server that provides a portable, multi-platform system for high performance data management. The new features of ASE 15.0 deliver an operational advantage with lower cost and risk, and they achieve higher performance on mixed workload systems.

Testing with ASE 15.0 has shown an advantage in overall performance for transaction processing over MySQL 5.0, and many complex queries show significant improvements.

The new query optimizer and execution engine greatly enhance the performance of complex queries. ASE 15.0 features patented query-processing technology that increases query performance and, when coupled with features such as Computed Columns and Function Indexes, significantly reduces reporting time on mission critical reports.

Sybase ASE 15.0 provides new features and functionality that can handle severe user demands for performance and economy. This paper will discuss how ASE 15.0 delivers advanced DSS performance on data sets of all sizes—from small to the VERY large—while controlling your costs. Some of the features that improve query performance are hash-based algorithms, merge joins, and N-ary joins.

2.0 QUERY PROCESSING ENHANCEMENTS

Significant work has been done with ASE 15.0 to enhance and optimize the new query-processing engine. The following section describes the design features that result in substantial performance benefits for applications with complex query requirements.

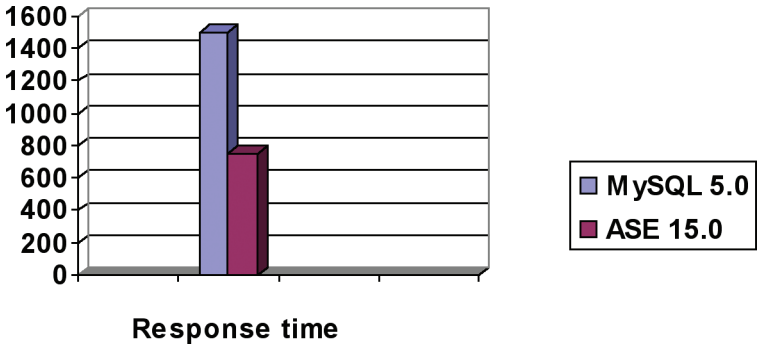
2.1 Hash-based Algorithms

Hashing is a technique used to match rows from one or more data stream by using a formula to compute a hash lookup key as a function of some set of attributes of each row.

ASE 15.0 query processing architecture uses various hashing techniques such as hash **joins**, hash **distinct**, hash **aggregations**, and hash **partitioning** to improve query performance. To demonstrate the efficiency of new hash-based algorithms, the following query was executed against ASE 15.0 and MySQL 5.0. The syntax of the query is provided in Appendix A.

2.1.1 Query 1 – Efficient hash aggregation and sort improvement

In this query, ASE 15.0 is choosing non-clustered index on lineitem table, and uses hash aggregate and sort operator, while mySQL 5.0 uses non-clustered index, file sort operation for order by clause, and temporary table for group by clause. This results in a better query response time of 737ms for ASE 15.0 as opposed to 1480 ms for MySQL 5.0. This clearly demonstrates that hash aggregate, sort, and scan operators are implemented efficiently in ASE 15.0.



2.2 Merge Join

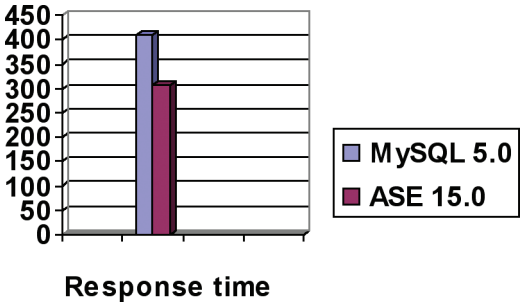
Merge joins involve synchronizing the scan of the joined tables on the merge key, i.e. on the join columns. They rely on their input derived tables being ordered on the merge key. As long as the merge keys in the current rows of the two children do not match, the input value is advanced to the next row. When we have a match, we produce an output row. Inner rows that match the current outer merge key are cached and re-played in case the next output row has the same key value. Merge joins are important since they are the cheapest join algorithms if ordering is available.

ASE 15.0 optimizer would consider merge join for queries of more than two tables if ordering exists in the joining columns.

2.2.1 Query 2 – Merge join performance

Returned Item reporting query is used to understand performance of merge join. This query finds the top 20 customers, in terms of their effect on lost revenue for a given quarter, who have returned parts. The query considers only parts that were ordered in the specified quarter.

In this query, ASE 15.0 uses merge join while MySQL 5.0 uses nested loop join. As mentioned earlier, merge join is the cheapest join if proper ordering is available. Response time for ASE 15.0 is 308 ms versus 410 ms for MySQL 5.0.



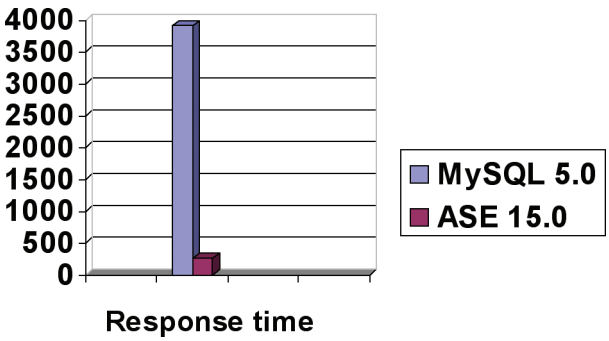
2.3 N-Ary Join

The N-ary NLJ is an optimized variant on the nested loop join. It has been patented by Sybase and provides at least the same performance as NLJ. As it in many cases outperforms the standard NLJ, ASE will automatically choose N-ary NLJ for two or more adjacent nested loop joins.

To understand the performance of N-Ary join, different flavors of the same query are executed against ASE 15.0 and MySQL 5.0. The following graphs illustrate the results obtained from ASE 15.0 and MySQL 5.0. The queries are shown in **Appendix A**.

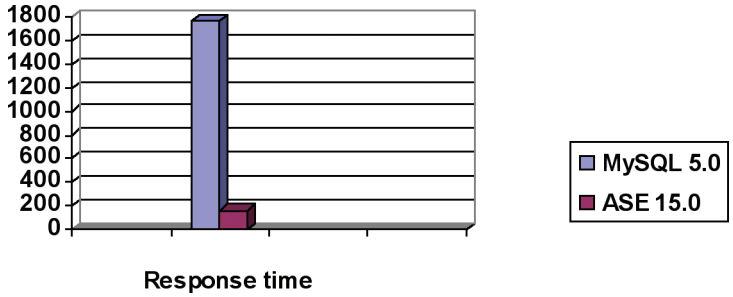
2.3.1 Query 3 – N-Ary join performance

In this query, MySQL 5.0 chooses the nested loop join, while ASE chooses a combination of N-Ary nested loop join and nested loop join. The advantage of the N-ary join is that when the join condition fails, this operator will get the next qualifying row from the outer table, skipping all inner tables that satisfy the conditions, resulting in fewer I/O's. The following chart shows the response time for ASE 15.0 and MySQL 5.0.



2.3.2 Query 3.1 – N-Ary join performance

In this case, the sub query is removed from the original query. MySQL 5.0 uses nested loop join, while ASE 15.0 uses N-Ary join. The response time for MySQL 5.0 is 1700 ms as opposed to 156 ms for ASE 15.0. This clearly demonstrates that ASE 15.0 chooses better plans than MySQL 5.0.



3.0 ADDITIONAL QUERY ENHANCEMENT FEATURES

This section describes additional features that enhance query performance. These features are either not used in the benchmark test, or analysis of performance for this features is not done against MySQL 5.0. They are included in this paper for the sake of completeness.

3.1 Improved Index Usage

Numerous architectural enhancements have been made in the ASE 15.0 optimizer to increase the instances where indexes are used to boost performance. Some of the examples are queries involving mixed data types, joins under OR clause, and RID joins.

3.2 Optimization for STAR Schema Joins

Star schema joins are common in decision support environments where several denormalized tables are joined together. The typical star schema consists of a very large fact table and several smaller dimension tables. The query usually involves joining the dimension tables with the fact tables with no join predicates between dimension tables. ASE 15.0 optimizer recognizes Star/Snowflakes patterns, and places fact tables in the end. It may use index intersection if indexes are available on join columns.

3.3 Improved Order and Sort Based Algorithms

In the ASE 15.0 optimizer, an advanced property model allows the optimizer to efficiently use algorithms such as merge join, merge union, group sorted and distinct sorted that require their children to provide ordered data. These algorithms are called order-based algorithms and are the most efficient relational algorithms. Besides providing a platform for generating plans that avoid sorts whenever possible and doing in-memory operations when needed, the new ASE optimizer may also piggyback other operations over the sort itself. For example, when enforcing an ordering while distinctness is needed, distinct sorting is generated instead of sort.

3.4 Improved Aggregations

Normally a materialization step is used for evaluating a "group by" aggregate. Optimizer would create a worktable and clustered index on grouping columns if no index is available. Also, if the number of distinct values in a grouping column is large, then the result may spill to disk in a tempdb worktable, even if there is an index on the column that could provide an ordered scan. Moreover, the time to the first row will increase until the entire table is scanned.

ASE 15.0 aggregation algorithms avoid materialization steps if an appropriate ordering is already available on the grouping columns. ASE 15.0 will use either hashed-based aggregation or order-based aggregation.

Any "group by" query will be affected by this decision and will have tremendous response time improvements since there are opportunities to return the first grouped row quickly, as well as to avoid accessing the tempdb

3.5 Materialization Avoidance

Materialization steps by such operations as a sort, hash build or grouped aggregation can frequently be avoided by careful modeling of properties within the query plan. Eliminating these materialization steps typically reduces "tempdb" usage and improves response time.

The ASE 15.0 architecture is much more of a "data flow" plan in which most of the absolute requirement of a worktable is removed. Worktables are used only if no alternative can be found.

4.0 TPCH BENCHMARK PERFORMANCE COMPARISONS

The TPC-H benchmark is a popular yardstick for comparing DSS query performance of RDBMSs on various hardware and software configurations. It consists of suite of business-oriented ad-hoc queries and data modification queries. This benchmark illustrates decision support systems that examine large volumes of data, execute queries with a high degree of complexity, and give answers to critical business questions.

The following benchmark compares ASE 15.0 to MySQL 5.0 performed on the Solaris 10 operating system, using the TPC-H standard benchmarking kit. Although table schema and queries are extracted from the standard TPC-H kit, the test does not fully comply with the TPC-H benchmark. No special query tuning is performed during the benchmark run. The results demonstrate that ASE 15.0 provides superior performance over MySQL 5.0 in the area of complex query processing.

5.0 CONFIGURATION

5.1 System Configuration

Server machine	Sun-Fire-V490
Total memory	12 GB
Server software	Sun® Solaris™ 10 Sybase Adaptive Server Enterprise 15.0.1 Sybase Open Client 15.0 MySQL 5.0.21

5.2 Server Configuration

Configuration is kept identical for ASE 15.0 and MySQL 5.0. Most of the configuration parameters are kept to default value. In MySQL 5.0, the InnoDB storage engine is enabled by default. MySQL 5.0 uses configuration option innodb_buffer_pool_size to allocate the amount of memory in the buffer pool to cache data and the indexes of its tables. The ASE 15.0 server is built with 4k based pages.

5.2.1 ASE Configuration

Total memory	512Mb
4k I/O Pool	150Mb
16k I/O Pool	150Mb
Max network packet size	2K
Number of sort buffers	2000
Max online engines	1

5.2.2 MySQL 5.0 Configuration

Innodb_buffer_pool_size	512Mb
Innodb_additional_mem_pool_size	20Mb
Innodb_log_buffer_pool_size	8Mb
Key_buffer_size	64M
Sort_buffer_size	4M
Read_buffer_size	2M
Net_buffer_length	2k

6.0 RESULTS

All the queries are executed 100 times, and the total time is reported. Response time is measured in seconds. MySQL 5.0 has a query cache feature that stores the text of SELECT statements together with corresponding results. This feature is not used in the benchmark.

As shown in the results table, ASE 15.0 outperforms MySQL 5.0 for most of the benchmark queries due to superior plan generation by the ASE 15.0 optimizer and the efficiency of new features. All the queries are listed in Appendix B.

Query Name	MySQL 5.0	ASE 15.0
Query 1	148	74
Query 2	1	1
Query 3	20	15.8
Query 4	218	5.6
Query 5	212	12
Query 6	54	7
Query 7	15	19
Query 8	6	5.3
Query 9	25	59
Query 10	44	31
Query 11	2	1.9
Query 12	189	20
Query 13	87	76
Query 14	65	7.6
Query 15	191	6.8
Query 16	9	17
Query 17	4	0.6
Query 18	3	80
Query 19	1	2.4
Query 20	323	24

7.0 CONCLUSION

ASE 15.0 has demonstrated better overall performance for transaction processing over the MySQL 5.0, and many complex queries show significant improvements without any tuning effort. The new ASE 15.0 optimizer chooses superior plans, and the new execution engine executes these plans very efficiently. Although a small data set is used for this benchmark, we expect similar performance improvements on larger data sets. This benchmark clearly demonstrates that, while maintaining its lead in OLTP environments, the new ASE 15.0 also provides superior performance in complex query environments.

APPENDIX A

Query 1 – Efficient hash aggregation and sort improvement

```
Select
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
        sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
from
        lineitem
where
                l_shipdate <= dateadd(day, 79, '1998-12-01')
group by
        l_returnflag,
        l_linestatus
order by
        l_returnflag,
        l_linestatus
```

Query 2 – Merge join performance

```
select
        c_custkey,
        c_name,
        sum(l_extendedprice * (1 - l_discount)) as revenue,
        c_acctbal,
        n_name,
        c_address,
        c_phone,
        c_comment
from
        customer,
        orders,
        lineitem,
        nation
where
        c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate >= '1994-04-01'
        and o_orderdate < dateadd(month,3, '1994-04-01')
        and l_returnflag = 'R'
        and c_nationkey = n_nationkey
group by
        c_custkey,
        c_name,
        c_acctbal,
        c_phone,
        n_name,
        c_address,
        c_comment
order by
        revenue desc
```

Query 3 – N-Ary join performance

```
select
        s_name,
        count(*) as numwait
from
        supplier,
        lineitem l1,
        orders,
        nation
where
        s_suppkey = l1.l_suppkey
        and o_orderkey = l1.l_orderkey
        and o_orderstatus = 'F'
        and l1.l_receiptdate > l1.l_commitdate
        and s_nationkey = n_nationkey
        and n_name = 'FRANCE'
group by
        s_name
order by
        numwait desc,
        s_name
```

Query 3.1 – N-Ary join performance

```
select
        s_name,
        count(*) as numwait
from
        supplier,
        lineitem l1,
        orders,
        nation
where
        s_suppkey = l1.l_suppkey
        and o_orderkey = l1.l_orderkey
        and o_orderstatus = 'F'
        and l1.l_receiptdate > l1.l_commitdate
        and s_nationkey = n_nationkey
        and n_name = 'FRANCE'
```

APPENDIX B

Query 1

```
select
        l_returnflag,
        l_linestatus,
        sum(l_quantity) as sum_qty,
        sum(l_extendedprice) as sum_base_price,
        sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
        sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
        avg(l_quantity) as avg_qty,
        avg(l_extendedprice) as avg_price,
        avg(l_discount) as avg_disc,
        count(*) as count_order
from
        lineitem
where
                l_shipdate <= dateadd(day, 79, '1998-12-01')
group by
        l_returnflag,
        l_linestatus
order by
        l_returnflag,
        l_linestatus
```

Query 2

```
select
        s_acctbal,
        s_name,
        n_name,
        p_partkey,
        p_mfgr,
        s_address,
        s_phone,
        s_comment
from region,  part p,partsupp ps,supplier s, nation
where
        p_partkey = ps_partkey
        and s_suppkey = ps_suppkey
        and p_size = 4
        and p_type like '%COPPER'
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = 'ASIA'
        and ps_supplycost = (
                select
                        min(ps_supplycost)
                from region r1, partsupp ps1, supplier s1, nation n1
                where
                          p.p_partkey = ps1.ps_partkey and
                          s1.s_suppkey = ps1.ps_suppkey and
                          s1.s_nationkey = n1.n_nationkey and
                          n1.n_regionkey = r1.r_regionkey and
                          r_name = 'ASIA'
        )
order by
        s_acctbal desc,
        n_name,
        s_name,
        p_partkey
```

Query 3

```
select
       l_orderkey,
       sum(l_extendedprice * (1 - l_discount)) as revenue,
       o_orderdate,
       o_shippriority
from
        customer,
        orders,
        lineitem
where
        c_mktsegment = 'HOUSEHOLD'
        and c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate < '1995-03-06'
        and l_shipdate > '1995-03-06'
group by
        l_orderkey,
        o_orderdate,
        o_shippriority
order by
        revenue desc,
        o_orderdate
```

Query 4

```
select
        o_orderpriority,
        count(*) as order_count
from
        orders
where
        o_orderdate >= '1995-04-01'
        and o_orderdate < dateadd (month, 3, '1995-04-01')
        and exists (
                select
                        *
                from
                        lineitem
                where
                        l_orderkey = orders.o_orderkey
                        and l_commitdate < l_receiptdate
        )
group by
        o_orderpriority
order by
        o_orderpriority desc
```

Query 5

```
select
        n_name,
        sum(l_extendedprice * (1 - l_discount)) as revenue
from
        customer,
        orders,
        lineitem,
        supplier,
        nation,
        region
where
        c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and l_suppkey = s_suppkey
        and c_nationkey = s_nationkey
        and s_nationkey = n_nationkey
        and n_regionkey = r_regionkey
        and r_name = 'MIDDLE EAST'
        and o_orderdate >=  '1995-01-01'
        and o_orderdate <  dateadd(year,1,'1995-01-01')
group by
        n_name
order by
        revenue desc
```

Query 6

```
select
        sum(l_extendedprice * l_discount) as revenue
from
        lineitem
where
        l_shipdate >= '1993-01-01'
        and l_shipdate < dateadd(year, 1, '1993-01-01')
        and l_discount between 0.07 - 0.01 and 0.07 + 0.01
        and l_quantity < 24
```

Query 7

```
create view shipping_vu_q7
as
    select
          n1.n_name as supp_nation,
          n2.n_name as cust_nation,
          year(l_shipdate) l_year,
          l_extendedprice * (1 - l_discount) as volume
    from
          supplier,
          lineitem,
          orders,
          customer,
          nation n1,
          nation n2
    where
          s_suppkey = l_suppkey
          and o_orderkey = l_orderkey
          and c_custkey = o_custkey
          and s_nationkey = n1.n_nationkey
          and c_nationkey = n2.n_nationkey
          and (
                (n1.n_name = 'IRAQ' and n2.n_name = 'INDONESIA')
                or (n1.n_name = 'INDONESIA' and n2.n_name = 'IRA
Q')
                 )
                and l_shipdate between '1995-01-01' and '1996-12-31'

select
        supp_nation,
        cust_nation,
        l_year,
        sum(volume) as revenue
from shipping_vu_q7
group by
        supp_nation,
        cust_nation,
        l_year
order by
        supp_nation,
        cust_nation,
        l_year
```

Query 8

```
create view all_nations_vu_q8
as
select
        datepart(year, o_orderdate) as o_year,
        l_extendedprice * (1 - l_discount) as volume,
        n2.n_name as nation
from
        part,
        supplier,
        lineitem,
        orders,
        customer,
        nation n1,
        nation n2,
        region
where
        p_partkey = l_partkey
        and s_suppkey = l_suppkey
```

```
        and l_orderkey = o_orderkey
        and o_custkey = c_custkey
        and c_nationkey = n1.n_nationkey
        and n1.n_regionkey = r_regionkey
        and r_name = 'AMERICA'
        and s_nationkey = n2.n_nationkey
        and o_orderdate between '1995-01-01' and '1996-12-31'
        and p_type = 'STANDARD BURNISHED TIN'
select
        o_year,
        sum(case
                when nation = 'CANADA' then volume
                else 0
        end) / sum(volume) as mkt_share
from
all_nations_vu_q8
group by
        o_year
order by
        o_year

Query 9

create view profit_vu_q9
as
        select
                n_name as nation,
                datepart(year, o_orderdate) as o_year,
                l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity
as amount
        from
                part,
                supplier,
                lineitem,
                partsupp,
                orders,
                nation
        where
                s_suppkey = l_suppkey
                and ps_suppkey = l_suppkey
                and ps_partkey = l_partkey
                and p_partkey = l_partkey
                and o_orderkey = l_orderkey
                and s_nationkey = n_nationkey
                and p_name like '%peach%'
go

select
        nation,
        o_year,
        sum(amount) as sum_profit
from
        profit_vu_q9
group by
        nation,
        o_year
order by
        nation,
        o_year desc
```

Query 10

```
select
        c_custkey,
        c_name,
        sum(l_extendedprice * (1 - l_discount)) as revenue,
        c_acctbal,
        n_name,
        c_address,
        c_phone,
        c_comment
from
        customer,
        orders,
        lineitem,
        nation
where

        c_custkey = o_custkey
        and l_orderkey = o_orderkey
        and o_orderdate >= '1994-04-01'
        and o_orderdate < dateadd(month,3, '1994-04-01')
        and l_returnflag = 'R'
        and c_nationkey = n_nationkey
group by
        c_custkey,
        c_name,
        c_acctbal,
        c_phone,
        n_name,
        c_address,
        c_comment
order by
        revenue desc
```

Query 11

```
select
        ps_partkey,
        sum(ps_supplycost * ps_availqty) as value
from
        partsupp,
        supplier,
        nation
where
        ps_suppkey = s_suppkey
        and s_nationkey = n_nationkey
        and n_name = 'CHINA'
group by
        ps_partkey having
                sum(ps_supplycost * ps_availqty) > (
                        select
                        sum(ps_supplycost * ps_availqty)*  0.0001000000
from
                                partsupp,
                                supplier,
                                nation
                        where
                                ps_suppkey = s_suppkey
                                and s_nationkey = n_nationkey
                                and n_name = 'CHINA'
                )
order by
      value desc
```

```
Query 12

select
        l_shipmode,
        sum(case
                when o_orderpriority = '1-URGENT'
                        or o_orderpriority = '2-HIGH'
                        then 1
                else 0
        end) as high_line_count,
        sum(case
                when o_orderpriority <> '1-URGENT'
                        and o_orderpriority <> '2-HIGH'
                        then 1
                else 0
        end) as low_line_count
from
        orders,
        lineitem
where
        o_orderkey = l_orderkey
        and l_shipmode in ('AIR', 'TRUCK')
        and l_commitdate < l_receiptdate
        and l_shipdate < l_commitdate
        and l_receiptdate >= '1996-01-01'
        and l_receiptdate < date_add('1996-01-01',interval 1 year)
group by
        l_shipmode
order by
        l_shipmode

Query 13

create view c_orders_vu_q13
as
                select
                        c_custkey,
                        count(o_orderkey) c_count
                from
                        customer left outer join orders on
                                c_custkey = o_custkey
                                and o_comment not like '%special%packages%'
                group by
                        c_custkey
go

select
        c_count,
        count(*) as custdist
from
        c_orders_vu_q13
group by
        c_count
order by
        custdist desc,
        c_count desc
```

```
Query 14
select
        100.00 * sum(case
                when p_type like 'PROMO%'
                        then l_extendedprice * (1 - l_discount)
                else 0
        end) / sum(l_extendedprice * (1 - l_discount)) as promo_revenue
from
        lineitem,
        part
where
        l_partkey = p_partkey
        and l_shipdate >= '1996-09-01'
        and l_shipdate < dateadd(month, 1, '1996-09-01')

Query 15

create view revenue0_vu_15
as
        select
                l_suppkey supplier_no,
                sum(l_extendedprice * (1 - l_discount)) total_revenue
        from
                lineitem
        where
                l_shipdate >= '1997-08-01'
                and l_shipdate < dateadd(month, 3, '1997-08-01')
        group by
                l_suppkey
go
select
        s_suppkey,
        s_name,
        s_address,
        s_phone,
        total_revenue
from
        supplier,
        revenue0_vu_15
where
        s_suppkey = supplier_no
        and total_revenue = (
                select
                        max(total_revenue)
                from
                        revenue0_vu_15
        )
order by
        s_suppkey
```

Query 16

```
select sum(l_extendedprice) / 7.0 as avg_yearly
from
        lineitem,
        part
where
        p_partkey = l_partkey
        and p_brand = 'Brand#12'
        and p_container = 'MED DRUM'
        and l_quantity < (
                select
                        0.2 * avg(l_quantity)
                from
                        lineitem
                where
                        l_partkey = part.p_partkey
        )
```

Query 17

```
select
        sum(l_extendedprice* (1 - l_discount)) as revenue
from
        lineitem,
        part
where
        (
                p_partkey = l_partkey
                and p_brand = 'Brand#32'
                and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
                and l_quantity >= 6 and l_quantity <= 6 + 10
                and p_size between 1 and 5
                and l_shipmode in ('AIR', 'AIR REG')
                and l_shipinstruct = 'DELIVER IN PERSON'
        )
or
(
                p_partkey = l_partkey
                and p_brand = 'Brand#24'
                and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
                and l_quantity >= 20 and l_quantity <= 20 + 10
                and p_size between 1 and 10
                and l_shipmode in ('AIR', 'AIR REG')
                and l_shipinstruct = 'DELIVER IN PERSON'
        )
        or
        (
                p_partkey = l_partkey
                and p_brand = 'Brand#25'
                and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
                and l_quantity >= 30 and l_quantity <= 30 + 10
                and p_size between 1 and 15
                and l_shipmode in ('AIR', 'AIR REG')
                and l_shipinstruct = 'DELIVER IN PERSON'
```

Query 18

```
select
        s_name,
        s_address
from
        supplier,
        nation
where
        s_suppkey in (
                select
                        ps_suppkey
                from
                        partsupp ps
                where
                        ps_partkey in (
                                select
                                        p_partkey
                                from
                                        part
                                where
                                        p_name like 'navajo%'
                        )
                   and ps_availqty > (
                            select
                             0.5 * sum(l_quantity)
                             from
                                lineitem
                             where
                              l_partkey = ps.ps_partkey
                              and l_suppkey = ps.ps_suppkey
                              and l_shipdate >= '1996-01-01'
                             and l_shipdate < dateadd(year, 1, '1996-
01-01')
                        )
        )
        and s_nationkey = n_nationkey
        and n_name = 'JAPAN'
order by
        s_name
```

Query 19

```
select
        s_name,
        count(*) as numwait
from
        orders,
        lineitem l1,
        supplier,
        nation

where
        s_suppkey = l1.l_suppkey
        and o_orderkey = l1.l_orderkey
        and o_orderstatus = 'F'
        and l1.l_receiptdate > l1.l_commitdate
        and not exists (
                select
                   *
                from
                        lineitem l3
                where
                        l3.l_orderkey = l1.l_orderkey
```

```
                        and l3.l_suppkey <> l1.l_suppkey
                        and l3.l_receiptdate > l3.l_commitdate
        )
        and exists (
                select
                        *
                from
                        lineitem l2
                where
                        l2.l_orderkey = l1.l_orderkey
                        and l2.l_suppkey <> l1.l_suppkey
        )
        and s_nationkey = n_nationkey
        and n_name = 'FRANCE'
group by
        s_name
order by
        numwait desc,
        s_name

Query 20

select
        c_name,
        c_custkey,
        o_orderkey,
        o_orderdate,
        o_totalprice,
        sum(l_quantity)
from
        customer,
        orders,
        lineitem
where
      o_orderkey in (
                select
                        l_orderkey
                from
                        lineitem
                group by
                        l_orderkey having
                                sum(l_quantity) > 300
        )
        and c_custkey = o_custkey
        and o_orderkey = l_orderkey
group by
        c_name,
        c_custkey,
        o_orderkey,
        o_orderdate,
  o_totalprice
order by
        o_totalprice desc,
        o_orderdate
```