Deployment of Parallel Data Warehouse Joint Design in Teradata

Ladjel Bellatreche[1], Soumia Benkrid[2], Ahmad Ghazal[3], Alain Crolotte[3], and Alfredo Cuzzocrea[4]

[1] LISI/ENSMA Poitiers University
Futuroscope, France
bellatreche@ensma.fr

[2] National High School for Computer Science (ESI)
Algiers, Algeria
s_benkrid@esi.dz

[3] Teradata Corporation, San Diego, CA, U.S.A.
(Ahmad.Ghazal, Alain.Crolotte)@teradata.com

[4] ICAR-CNR and University of Calabria, Italy
cuzzocrea@si.deis.unical.it

Abstract. Parallel databases play an important role in delivering performance required by data warehousing applications. Such performance can not be accomplished without a good scheme for fragmentation and allocation of data. Both problems are NP-hard problems and optimal solutions are impractical in real life situations. These two problems were addressed extensively in research and there are a lot of heuristic based efficient solutions that provide good results. In these solutions, the fragmentation and allocation were done *separately*. In our previous work, we proposed a genetic solution that solves both problems *simultaneously* which we refer to as a the *joint solution*. The joint solution was tested on some known benchmarks using *internal simulations of a parallel database*. These simulations are not *reliable* since it lacked the real life aspects of *parallelism*. This paper addresses this issue by applying our joint solution on the Teradata DBMS. We experiment the SSB benchmark (TPC-H like) on a Teradata appliance running TD 13.10. The results shows a significant improvement over previous results that performs fragmentation and allocation sequentially.

1 Introduction

Data warehousing is becoming more complex in terms of applications and data size. The parallelism is one of the relevant solutions to deal with mountains of data managed by warehouse and complex queries. More and more organizations are relying on parallel processing technologies to achieve the performance, scalability, and reliability they need [13]. Most of the major commercial database systems support parallelism (Teradata, Oracle, IBM, Microsoft, Sybase, etc.). Rather than relying on a single monolithic processor, parallel systems exploit fast and inexpensive micro processors to achieve high performance.

Designing parallel databases was widely studied in the contexts of OLTP (On-Line Transaction Processing) [1, 7, 11, 12, 14, 15, 17, 19] and OLAP (On-Line Analytical Processing) [2, 4, 10, 13, 21, 22, 20]. Most of these studies are usually performed from theoretical point of view. On the other words, none deployment on real parallel database machine is given.

The main important steps that designers need to perform to construct parallel data warehouses are: **(i)** the *choice of the hardware platform*, **(ii)** the *partitioning of data warehouse schema*, **(iii)** the *allocation of the generated fragments*, **(iv)** the *load balancing over the nodes* of the chosen parallel machine, **(v)** the *query processing* and **(vi)** the deployment of solutions of the previous phases. All these steps got a lot of attention of data warehouse research community, since most of them are inherited from traditional parallel database design, except the last one.

(i) There are three widely used architectures for parallelizing work: (a) *shared memory* (b) *shared disk* and (c) *shared nothing.* In a *shared-memory approach*, all of the CPUs share a single memory and a single collection of disks. This approach is relatively easy to implement, since the lock manager and buffer pool are both stored in the memory system where they can be easily accessed by all the processors. Unfortunately, it has fundamental scalability limitations, as all I/O and memory requests have to be transferred over the same bus that all of the processors share, causing the bandwidth of this bus to rapidly become a bottleneck [8, 12]. In *shared-disk platform*, there are a number of independent processor nodes, each with its own memory. These nodes all access a single collection of disks. This architecture has been used to design parallel data warehouse by [24]. It has a number of drawbacks that severely limit scalability. First, the interconnection network that connects each of the CPUs to the shared-disk subsystem can become an I/O bottleneck. Second, since there is no pool of memory that is shared by all the processors, there is no obvious place for the lock table or buffer pool to reside. In a *shared-nothing approach*, each processor has its own set of disks. This architecture is used by *Teradata.* Data is horizontally partitioned across nodes, such that each node has a subset of the rows from each table in the database. Each node is then responsible for processing only the rows on its own disks. Such architectures are especially well suited to the star queries running on data warehouses modelled using a star schema, as only a very limited amount of communication bandwidth is required to join one or more (typically small) dimension tables with the (typically much larger) fact table [8].

(ii) Once the architecture is chosen, data warehouse designer partitions its schema. Fragmentation[5] is a *pre condition* of parallel data warehouse design. It may be *horizontal*, where table instances are decomposed into disjoint partitions or *vertical*, where tables are split in disjoint sets of attributes. The horizontal partitioning is mainly used for designing parallel data warehouses [24, 25, 10, 13, 2]. Two main types of horizontal partitioning exist [5]: *mono table partitioning* and *table-dependent partitioning.* In the mono table partitioning, a table is partitioned using its own attributes. It is quite similar to the *primary horizontal*

[5] In this paper, we use fragmentation and partitioning interchangeably.

partitioning proposed in traditional databases [17]. Several modes exist to support mono table partitioning: *Range, List, Hash, Round Robin* (supported by Sybase), *Composite* (Range-Range, Range-List, List-List), etc.), *Virtual Column partitioning* recently proposed by *Oracle11G*. In *table-dependent partitioning*, a table inherits the partitioning characteristics from other table. For instance a fact table may be partition based on the fragmentation schemes of dimension tables. This partitioning is feasible if a parent-child relationship among these tables exists [6, 9]. Two main implementations of this partitioning are possible: *native referential partitioning* and *simulated referential partitioning*. The native referential partitioning is recently supported by Oracle11G to equi-partition tables connected by a parent child referential constraint. A native DDL is given to perform this partitioning [9] (*Create Table ... Partition by Reference ...*). This *diversity* of existing modes poses problem of deploying existing research studies on parallel data warehouse design, since a direct deployment is hard to perform. To summarize, we can notice that fragmentation schemes obtained by some partitioning algorithms can be directly implemented in a priori known DBMS (this situation is called a *turnkey solutions*). Others need to be adapted according the target DBMS (*non turnkey solutions*). [2] is an example of turnkey solutions, where algorithms were proposed to *referential partition* a data warehouse. This partitioning was initially supported by Oracle [9]. [10] is an example of non turnkey solutions, where referential partitioning is implemented manually in the context of parallel database machine as follows: (i) a dimension table is first horizontally partitioned using its primary key, then the fact table is decomposed based on its foreign key referencing that dimension table.
(iii) The data allocation is the process that places generated fragments over nodes of parallel machine. This allocation may be either *redundant* (with replication) or *non redundant* (without replication). **(iv)** Once fragments are placed, global queries are then rewritten over fragments and executed on the parallel machine. During their execution phase, **(v)** the load balancing should be verified. Load balancing refers to workload allocation over nodes [19]. **(vi)** The deployment is usually done in simulated environments or using mathematical cost models quantifying the quality of parallel design.

In [2], we propose a parallel data warehouse design approach, where fragmentation and allocation are done in joint way in order to capture the interdependency between these two steps. The decision of allocation fragments in done during the fragmentation process. The quality of this method is measured by the means of a cost model estimating the query processing cost in terms of inputs outputs required for executing a set of queries. The main objective of this paper is to verify our results on a real life parallel DBMS. Based on our collaboration with the Teradata labs, we managed to run our results on a Teradata appliance running TD 13.0. Teradata is a known MPP DBMS and have been in the Gartner's "Data Warehouse DBMS Magic Quadrant" for many years.

The paper is organized as follows. Section 2 summarizes existing approaches for designing parallel data warehouses. In Section 3, we give background related to the joint design methodology for parallel data warehouse. Section 4 describes the validation of our joint approach on *Teradata* machine using star schema benchmark data set [16]. Finally, Section 5 concludes the paper summarizing the main findings of our research, and proposing directions for future work.

2 Related Work

This section reviews the most important studies on parallel data warehouse design from academic [2, 4, 10, 13, 21, 22] and industrial perspective [20].

Academic studies were essentially focused on proposing solutions for designing data warehouses for a given parallel machine architecture. Furtado [10] discusses partitioning strategies for *node-partitioned data warehouses*. The main suggestion coming from [10] can be synthesized in a "best-practice" recommendation stating to partition the fact table on the basis of the *larger* dimension tables (given a ranking threshold). In more detail, each larger dimension table is first partitioned by means of the *Hash mode* approach via its primary key. Then, the fact table is again partitioned by means of the Hash mode approach via foreign keys referencing the larger dimension tables. Finally, the so-generated fragments are allocated according to two alternative strategies, namely *round robin* and *random*. Smaller dimension tables are instead fully-replicated across the nodes of the target data warehouse. In [13], Lima *et al.* focus the attention on data allocation issues for database clusters. Authors recognize that how to place data/fragments on the different PC of a database cluster in the dependence of a given criterion/goal (e.g., query performance) plays a critical role, hence the following two straightforward approaches can be advocated: (*i*) full replication of the target database on *all* the PC, or (*ii*) meaningful partition of data/fragments across the PC. Starting from this main intuition, authors propose an approach that combines partition and replication for OLAP-style workloads against database clusters. In more detail, the fact table is partitioned and replicated across nodes using the so-called *chained de-clustering*, while dimension tables are fully-replicated across nodes. This comprehensive approach enables the *middleware layer* to perform load balancing tasks among replicas, with the goal of improving query response time. Furthermore, the usage of chained de-clustering for replicating fact table partitions across nodes allows the designer not to detail the way of selecting the number of replicas to be used during the replication phase. In [21], the allocation of relational data warehouses based on a star schema and utilizing bitmap index structures in a *shared disk architecture* is proposed. The fragments are generated by the means of multi-dimensional hierarchical data fragmentation of the fact table. The proposal is validated by a *simulation model* [22]. In these studies, fragmentation and allocation are done in sequential way. In [2, 4], another trend of parallel data warehouse design was proposed in which partitioning and allocation processes are done simultaneously. These works were done in a shared nothing architecture [2] and heterogeneous database cluster [4].

To summarize, we figure out that the academic studies are validated either by the means of simple cost models estimating the number of inputs outputs required for executing a set of queries or by simulators. None deployment in a real machine is given.

From industrial perspective, DB2 DBMS [20] proposed a solution for data partitioning in shared nothing architecture. Based on this work, data partitioning advisor is developed to recommend user the number of partitions of each fragments. As academic studies, this work considers fragmentation and allocation are done sequentially.

3 Background

In this section, we review the joint methodology for designing parallel data warehouses, where fragmentation and allocation are done simultaneously. To facilitate the understanding of our methodology, we give a formalization of the parallel data warehouse design problem [2]:

– a data warehouse schema composed by d dimension tables $\mathcal{D} = \{D_0, \ldots, D_{d-1}\}$ and one fact table $\mathcal{F}$ – as in [10, 13];
– a shared nothing architecture with M nodes $\mathcal{N} = \{N_0, N_1, \ldots, N_{M-1}\}$;
– a set of star queries $\mathcal{Q} = \{Q_1, Q_2, \ldots, Q_{L-1}\}$ to be executed over the warehouse schema, being each query Q_l, with $0 \leq l \leq L-1$;
– a *maintenance constraint* $\mathcal{W} : W > M$ representing the number of fragments W that the designer considers relevant for his/her target allocation process, called *fragmentation threshold*;

The problem of designing a parallel data warehouse consists in *fragmenting the fact table $\mathcal{F}$ into N_F fragments and allocating them over different* nodes *such that the total cost of executing all the queries in $\mathcal{Q}$ can be minimized while processing constraints are satisfied across nodes*, *under the maintenance constraint $\mathcal{W}$*.

Based on the formal statement above, it follows that our investigated problem is composed by two sub-problems, namely data partitioning and fragment allocation. Each one of these problems is known to be *NP-complete* [3, 23, 11]. In order to deal with the parallel design problem, two main classes of methodologies are possible: *sequential design* and *joint design*. Sequential design methodology has been proposed in the context of traditional distributed and parallel database design research. The basic idea underlying this methodology consists in first fragmenting the data warehouse using *any* partitioning algorithm, and then allocating the so-generated fragments by means of *any* allocation algorithm. In the most general case, each partitioning and allocation algorithm has its own cost model. The main advantage coming from these traditional methodologies is represented by the fact that they are straightforwardly applicable to a large number of even-heterogeneous parallel and distributed environments. Contrary to this, their main limitation is represented by the fact that they neglect the inter-dependency between the data partitioning and the fragment allocation phase, respectively. Another limitation of this approach is the fact that it uses two different cost models: one for fragmentation process and another for allocation process.

To take into account de inter-dependency between and partitioning and fragments allocation, the joint approach is proposed. During the fragmentation phase, each potential solution is tested for allocation process. At the end, the solution with minimum cost is chosen. Only one integrated cost model is used for both processes: fragmentation and allocation. Figure 1 summarizes the steps of joint design methodology [2].

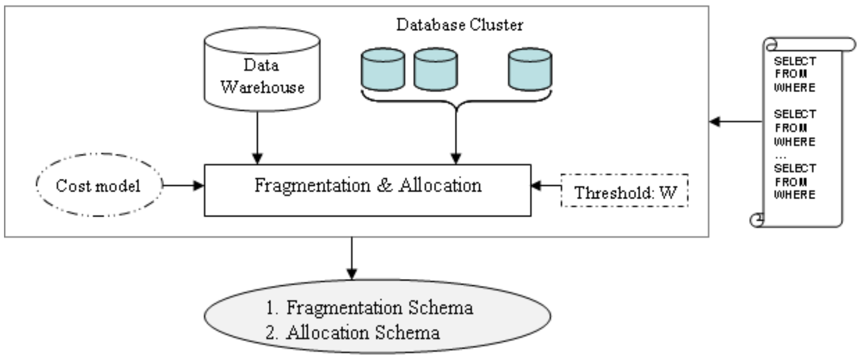

Fig. 1: Joint Design Methodology

4 Validation on Teradata

Empirical results in previous work [2] were based on custom simulation for a distributed system using a single CPU machine. These simulations lacked the real life aspect to demonstrate the efficacy of the new results. In collaboration with *Teradata* Labs, we verified our results on a *Teradata* system running TD 13.10 software.

In this section we provide a high level description of the Teradata DBMS and the SSB benchmark and how customized it. Finally, we present the results of the SSB benchmark on both the joint and sequential methods.

4.1 Teradata Description

Teradata is a massively parallel processing system running shared nothing architecture. The *Teradata* DBMS is linearly and predictably scalable in all dimensions of a database system workload (data volume, breadth, number of users, complexity of queries). The basic unit of parallelism in *Teradata* is a virtual processor (called Access Module Processor or AMP) which is assigned a data portion. Each AMP executes DBMS functions on its own data. This allows locks and buffers do not have to be shared which ensures scalability. Figure 2 illustrates the *Teradata* architecture by a two node system. A node is a multi-core system with disks and memory. It provides a pool of resources (disk, memory, etc) for the AMPS in that node. BYNET is the network used to link different AMPs within a node and across different nodes as well.

Data entering a *Teradata* Database are processed through a sophisticated hashing algorithm and automatically distributed across all AMPs in the system. In addition to being a distribution technique, this hash approach serves as an indexing strategy. This significantly reduces the amount of DBA work normally required to set up direct access. To define a *Teradata* Database, the DBA simply chooses a column or set of columns as the primary index for each table. The value contained in these indexed columns is used to determine the AMP, which owns the data, as well as a logical storage location within the AMP's associated disk space, all without performing a separate CREATE INDEX operation. To retrieve a row, the primary index value is again passed to the hash algorithm, which generates the two hash values, AMP and Hash-ID. These values are used to immediately determine which AMP owns the row and where the data are stored.

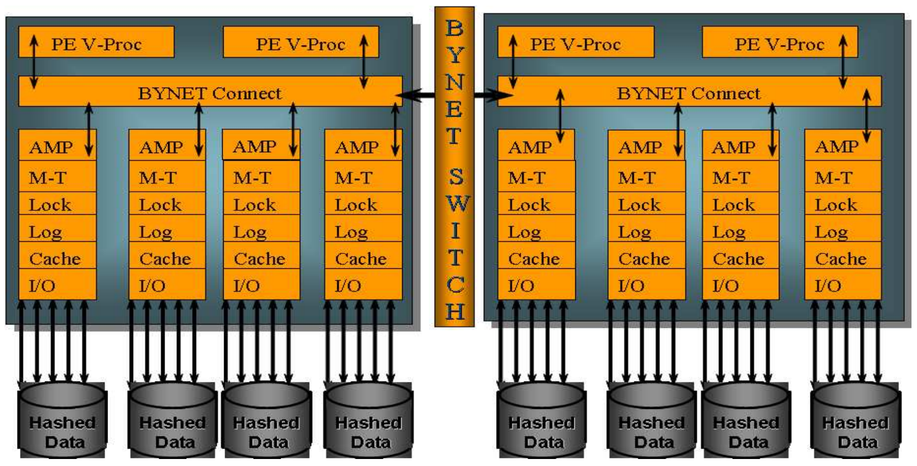

Fig. 2: Teradata Internal Architecture (MPP)

4.2 Experiments

The experiments were designed around the SSB benchmark [16]. This benchmark is based on a star schema derived from the TPC-H schema [18]. Like TPC-H the SSB benchmark comes with a data generation utility (dbgen) which is scalable. For our experiments we used 10 scale factor (10GB). The SSB benchmark is built around a fact table called *lineorder* and 4 dimension tables: *part*, *supplier*, *customer* and *date*. All tables are derived from the TPC-H database except the date dimension which is new. The main table is the fact table *lineorder* with the following DDL:

```
CREATE TABLE dbo10_sq.LINEORDER(
        LO_ORDERKEY int NOT NULL,
        LO_LINENUMBER int NOT NULL,
        LO_CUSTKEY int NOT NULL, -> FK to CUSTOMER
        LO_PARTKEY int NOT NULL, -> FK to PART
        LO_SUPPKEY int NOT NULL, -> FK to SUPPLIER
        LO_ORDERDATE int NOT NULL,
        LO_ORDERPRIORITY char(15) NOT NULL,
        LO_SHIPPRIORITY char(1) NOT NULL,
        LO_QUANTITY int NOT NULL,
        LO_EXTENDEDPRICE int NOT NULL,
        LO_ORDTOTALPRICE int NOT NULL,
        LO_DISCOUNT int NOT NULL,
        LO_REVENUE int NOT NULL,
        LO_SUPPLYCOST int NOT NULL,
        LO_TAX int NOT NULL,
        LO_COMMITDATE int NOT NULL,
        LO_SHIPMODE char(10) NOT NULL);
```

At scale factor 10 the row counts are given in Table 1. More details on the SSB

Table 1: Scale Factor

lineorder	59986052
part	800000
customer	300000
supplier	20000
ddate	2556

benchmark can be found at [16].

4.3 Implementation and Testing Joint and Sequential Approaches

To implement the Joint and Sequential fragmentation/allocation schemes and assess their respective effects in *Teradata* we proceeded as follows:

1. The joint and sequential algorithms is applied to the SSB workload (query descriptions shown later). We used a cost model that is focused on the I/O factor. For more details on these algorithms refer to [2]. Sequential approach starts by partitioning the SSB benchmark schema using a genetic algorithm [2]. The obtained fragmentation schema is then allocated over various nodes.
2. The obtained theoretical results from our algorithms are implemented to *Teradata* as follows:

Table 2: Results (Time in Seconds)

Queries	Joint	Sequential
Q01.1	0.12	0.59
Q01.2	0.11	0.11
Q01.3	0.12	0.11
Q04.1	0.60	0.58
Q04.2	0.54	0.53
Q04.3	1.08	1.13
Q05.0	0.36	0.15
Q06.0	0.53	0.13
Q07.0	0.46	0.58
Q08.0	0.14	2.15
Q09.0	0.08	0.14
Q10.0	0.07	0.16
Q11.0	0.32	0.62
Q12.0	0.34	0.72
Q13.0	0.18	0.63
Q14.0	0.18	0.24
Q15.0	0.18	0.57
Q16.0	0.18	0.51
Q17.0	0.19	0.24
Q18.0	0.29	0.64
Q19.0	0.56	0.66
Q20.0	0.49	0.33
total time	7.12	11.52

- The dimension tables are hash distributed using their primary key field.
- The fact table is partitioned using the results of each of the sequential and joint algorithms. Each fragment is represented as a separate table. These fragments were then allocated to particular AMPs based on the hash function that reflect the allocation scheme.
- Finally, the fact table lineorder is defined as a view with UNION of these fragments

For experiments in this section, we have considered a query workload of 22 queries. They are based on the original 13 queries (except Q1.2, Q1.3) but with varied predicates. We have used 50 selection predicates defined on 11 different attributes: (*d.d_year, p.p_category, d.d_yearmonth, s.s_region, p.p_brand , c.c_region, c.c_nation, s.s_nation, c.c_city, s.s_city, p.p_mfgr*). The domains of these attributes are split into: 7, 2, 2, 7, 3, 5, 2, 6, 3, 3 and 3 sub domains, respectively to perform the genetic algorithms for joint and sequential approaches [2]. Note that each selection predicate has a selectivity factor computed using SQL queries executed on the data set of SSB benchmark (see Appendix A).

This technique was applied for both Joint and Sequential approaches. Table 2 below shows the run time (in seconds) of the queries for both joint and sequential methods. Each method performed better in certain queries. However, the joint method performed 38% better for the overall workload which validates the results since the workload performance is the objective of both algorithms.

5 Conclusion

This work is the fruit of collaboration between academician and industrial representing by *Teradata Labs* established during DAWAK'10 in Bilbao Spain, where we presented our paper on *joint parallel data warehouse design*. In our previous studies, the verification of the joint method is based on a single CPU system simulating parallel and distributed systems. In this paper, we verified the superiority of our joint method over sequential method using the Teradata DBMS running the SSB benchmark. Overall, the joint method performed 38% better than the sequential method.

Future extensions include incorporating the Teradata cost model in the joint solution. This will insure that the actual cost of CPU, I/O and network cost are reflected. Also, other models beyond star schema like the TPC-H model can be benchmarked.

6 Appendix

Query	Description
Q01.1	select sum(lo_extendedprice*lo_discount) as revenue from lineorder l, DATE d where l.lo_orderdate=d.d_datekey and d.d_year='1993' and l.lo_discount in (1,2,3) and l.lo_quantity=25;
Q01.2	select sum(lo_extendedprice*lo_discount) as revenue from lineorder l, DATE d where l.lo_orderdate=d.d_datekey and d.d_year='1994' and l.lo_discount in (1,2,3) and l.lo_quantity=25;
Q01.3	select sum(lo_extendedprice*lo_discount) as revenue from lineorder l, DATE d where l.lo_orderdate=d.d_datekey and d.d_year='1995' and l.lo_discount in (1,2,3) and l.lo_quantity=25;
Q04.1	select sum(lo_revenue), d_year, p_brand from lineorder l, DATE d, part p, supplier s where l.lo_orderdate=d.d_datekey and l.lo_partkey=p.p_partkey and l.lo_suppkey=s.s_suppkey and p.p_category='MFGR#12'and s.s_region='AMERICA' group by d.d_year, p.p_brand order by d.d_year, p.p_brand;
Q04.2	select sum(lo_revenue), d_year, p_brand from lineorder l, DATE d, part p, supplier s where l.lo_orderdate=d.d_datekey and l.lo_partkey=p.p_partkey and l.lo_suppkey=s.s_suppkey and p.p_category='MFGR#12' and s.s_region='AFRICA' group by d.d_year, p.p_brand order by d.d_year, p.p_brand;
Q04.3	select sum(lo_revenue), d_year, p_brand from lineorder l, DATE d, part p, supplier s where l.lo_orderdate=d.d_datekey and l.lo_partkey=p.p_partkey and l.lo_suppkey=s.s_suppkey and p.p_category='MFGR#12' and (s.s_region='ASIA' or s.s_region='MIDDLE EAST' or s.s_region='EUROPE') group by d.d_year, p.p_brand order by d.d_year, p.p_brand;
Q05.0	select sum(lo_revenue), d_year, p_brand from lineorder l, .DATE d, part p, supplier s where l.lo_orderdate=d.d_datekey and l.lo_partkey=p.p_partkey and l.lo_suppkey=s.s_suppkey and p.p_brand in ('MFGR#2221','MFGR#2222', 'MFGR#2223','MFGR#2224','MFGR#2225', 'MFGR#2226','MFGR#2227','MFGR#2228') and s.s_region='ASIA' group by d.d_year, p.p_brand order by d.d_year, p.p_brand;
Q06.0	select sum(lo_revenue), d_year, p_brandc from lineorder l, DATE d, part p, supplier s where l.lo_orderdate=d.d_datekey and l.lo_partkey= p.p_partkey and l.lo_suppkey= s.s_suppkey and p.p_brand='MFGR#2239' and s.s_region='EUROPE' group by d.d_year, p.p_brand order by d.d_year, p.p_brand;
Q07.0	select c_nation, s_nation, d_year, sum(lo_revenue) as revenue from customer c, lineorder l, supplier s, DATE d where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_orderdate=d.d_datekey and c.c_region='ASIA' and s.s_region='ASIA'and d.d_year in(1992,1993,1994,1995,1996,1997) group by c.c_nation, s.s_nation, d.d_year order by d.d_year asc, revenue desc;
Q08.0	select c_city, s_city, d_year, sum(lo_revenue) as revenue from customer c, lineorder l, supplier s, DATE d where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_orderdate= d.d_datekey and c.c_nation='UNITED STATES' and s.s_nation='UNITED STATES'and d.d_year in(1992,1993,1994,1995,1996,1997) group by c.c_city, s.s_city, d.d_year order by d.d_year asc, revenue desc;
Q09.0	select c_city, s_city, d_year, sum(lo_revenue) as revenue from customer c, lineorder l, supplier s, DATE d where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_orderdate=d.d_datekey and (c.c_city='UNITED KI1' or c.c_city='UNITED KI5') and (s.s_city='UNITED KI1' or s.s_city='UNITED KI5') and d.d_year in(1992,1993,1994,1995,1996,1997) group by c.c_city, s.s_city, d.d_year order by d.d_year asc, revenue desc;
Q10.0	select c_city, s_city, d_year, sum(lo_revenue) as revenue from customer c, lineorder l, supplier s, DATE d where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_orderdate= d.d_datekey and (c.c_city='UNITED KI1' or c.c_city='UNITED KI5') and (s.s_city='UNITED KI1' or s.s_city='UNITED KI5') and d.d_yearmonth = 'Dec1997'group by c.c_city, s.s_city, d.d_year order by d.d_year asc, revenue desc;
Q11.0	select d_year, s_nation, sum(lo_revenue - lo_supplycost) as profit from DATE d, customer c, supplier s, part p, lineorder l where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_partkey=p.p_partkey and l.lo_orderdate=d.d_datekey and c.c_region='AMERICA' and s.s_region='AMERICA' and (p.p_mfgr='MFGR#1' or p.p_mfgr='MFGR#2') group by d.d_year, c.c_nation order by d.d_year, c.c_nation;
Q12.0	select d_year, s_nation, p_category, sum(lo_revenue - lo_supplycost) as profit from DATE d, customer c, supplier s, part p, lineorder l where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_partkey=p.p_partkey and l.lo_orderdate=d.d_datekey and c.c_region='AMERICA' and s.s_region='AMERICA' and (d.d_year=1997 or d.d_year=1998)and(p.p_mfgr='MFGR#1' or p.p_mfgr= 'MFGR#2') group by d.d_year,s.s_nation,p.p_category order by d.d_year,s.s_nation, p.p_category;
Q13.0	select d_year, s_city, p_brand, sum(lo_revenue - lo_supplycost) as profit from DATE d, customer c, supplier s, part p, lineorder l where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_partkey=p.p_partkey and l.lo_orderdate=d.d_datekey and s.s_nation='UNITED STATES' and (d.d_year=1997 or d.d_year=1998) and p.p_category='MFGR#14' group by d.d_year, s.s_city, p.p_brand order by d.d_year, s.s_city, p.p_brand;
Q14.0	select d_year, s_city, p_brand1, sum(lo_revenue - lo_supplycost) as profit from DATE d, customer c, supplier s, part p, lineorder l where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_partkey=p.p_partkey and l.lo_orderdate=d.d_datekey and s.s_nation='EGYPT' and (d.d_year=1997 or d.d_year=1998) and p.p_category='MFGR#14' group by d.d_year, s.s_city, p.p_brand order by d.d_year, s.s_city, p.p_brand;
Q15.0	select d_year, s_city, p_brand, sum(lo_revenue - lo_supplycost) as profit from DATE d, customer c, supplier s, part p, lineorder l where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_partkey=p.p_partkey and l.lo_orderdate=d.d_datekey and s.s_nation='ALGERIA' and (d.d_year=1997 or d.d_year=1998) and p.p_category='MFGR#14' group by d.d_year, s.s_city, p.p_brand order by d.d_year, s.s_city, p.p_brand;
Q16.0	select d_year, s_city, p_brand, sum(lo_revenue - lo_supplycost) as profit from DATE d, customer c, supplier s, part p, lineorder l where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_partkey=p.p_partkey and l.lo_orderdate=d.d_datekey and s.s_nation='ALGERIA' and (d.d_year=1996 or d.d_year=1997) and p.p_category='MFGR#14' group by d.d_year, s.s_city, p.p_brand order by d.d_year, s.s_city, p.p_brand;
Q17.0	select d_year, s_city, p_brand, sum(lo_revenue - lo_supplycost) as profit from DATE d, customer c, supplier s, part p, lineorder l where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_partkey=p.p_partkey and l.lo_orderdate=d.d_datekey and s.s_nation='CANADA' and (d.d_year=1997 or d.d_year=1998) and p.p_category='MFGR#14' group by d.d_year, s.s_city, p.p_brand order by d.d_year, s.s_city, p.p_brand;
Q18.0	select c_nation, s_nation, d_year, sum(lo_revenue) as revenue from customer c, lineorder l, supplier s, DATE d where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_orderdate=d.d_datekey and c.c_region='AMERICA' and s.s_region='AMERICA' and d.d_year in(1992,1993,1994,1995,1996,1997) group by c.c_nation, s.s_nation, d_year order by d.d_year asc, revenue;
Q19.0	select c_nation, s_nation, d_year, sum(lo_revenue) as revenue from customer c, lineorder l, supplier s, DATE d where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_orderdate=d.d_datekey and c.c_region='MIDDLE EAST'and s.s_region='MIDDLE EAST' and d.d_year in(1992,1993,1994,1995,1996,1997) group by c.c_nation, s.s_nation, d_year order by d.d_year asc, revenue desc;
Q20.0	select c_nation, s_nation, d_year, sum(lo_revenue) as revenue from customer c, lineorder l, supplier s, DATE d where l.lo_custkey=c.c_custkey and l.lo_suppkey=s.s_suppkey and l.lo_orderdate=d.d_datekey and c.c_region='EUROPE' and s.s_region='EUROPE' and d.d_year in(1992,1993,1994,1995,1996,1997) group by c.c_nation, s.s_nation, d_year order by d.d_year asc, revenue desc;

References

1. P. M. G. Apers. Data allocation in distributed database systems. *ACM Transactions on database systems*, 13(3):263–304, 1988.
2. L. Bellatreche and S. Benkrid. A joint design approach of partitioning and allocation in parallel data warehouses. In *11th International Conference on Data Warehousing and Knowledge Discovery (DAWAK'09)*, pages 99–110, 2009.
3. L. Bellatreche, K. Boukhalfa, and P. Richard. Data partitioning in data warehouses: Hardness study, heuristics and oracle validation. In *International Conference on Data Warehousing and Knowledge Discovery (DaWaK'2008)*, pages 87–96, 2008.
4. Ladjel Bellatreche, Alfredo Cuzzocrea, and Soumia Benkrid. &: A methodology for effectively and efficiently designing parallel relational data warehouses on heterogenous database clusters. In *12th International Conference on Data Warehousing and Knowledge Discovery (DAWAK)*, pages 89–104, 2010.
5. Ladjel Bellatreche and Komla Yamavo Woameno. Dimension table driven approach to referential partition relational data warehouses. In *ACM 12th International Workshop on Data Warehousing and OLAP (DOLAP)*, pages 9–16, 2009.
6. S. Ceri, M. Negri, and G. Pelagatti. Horizontal data partitioning in database design. *Proceedings of the ACM SIGMOD International Conference on Management of Data. SIGPLAN Notices*, pages 128–136, 1982.
7. D. J. DeWitt and J. Gray. Parallel database systems: The future of high performance database systems. *Communnications ofthe ACM*, 35(6):85–98, 1992.
8. D. J. D DeWitt, S. Madden, and M. Stonebraker. How to build a high-performance data warehouse. *http://db.lcs.mit.edu/madden/high_perf.pdf.*
9. G. Eadon, E. I. Chong, S. Shankar, A. Raghavan, J. Srinivasan, and S. Das. Supporting table partitioning by reference in oracle. *SIGMOD'08*, 2008.
10. P. Furtado. Experimental evidence on partitioning in parallel data warehouses. In *DOLAP*, pages 23–30, 2004.
11. K. Karlapalem and N. M Pun. Query driven data allocation algorithms for distributed database systems. *in 8th International Conference on Database and Expert Systems Applications (DEXA'97)*, pages 347–356, September 1997.
12. L.Bouganim, D. Florescu, and P. Valduriez. Dynamic load balancing in hierarchical parallel database systems. In *Proceedings of the International Conference on Very Large Databases*, pages 436–447, 1996.
13. A. B. Lima, C. Furtado, P. Valduriez, and M. Mattoso. Improving parallel olap query processing in database clusters with data replication. *To appear in Distributed and Parallel Database Journal*, 2009.
14. M. Mehta and D. J. DeWitt. Data placement in shared-nothing parallel database systems. *VLDB Journal*, 6(1):53–72, 1997.
15. S. Menon. Allocating fragments in distributed databases. *IEEE Transactions on Parallel and Distributed Systems*, 16(7):577–585, July 2005.
16. P. O'Neil, E. B. O'Neil, and X.uedong Chen. The star schema benchmark: http://www.cs.umb.edu/ poneil/starschemab.pdf. 2007.
17. M. T. Özsu and P. Valduriez. *Principles of Distributed Database Systems : Second Edition*. Prentice Hall, 1999.
18. TPC Home Page. Tpc benchmark^{TM}d (decision support). *http://www.tpc.org.*
19. E. Rahm and R. Marek. Analysis of dynamic load balancing strategies for parallel shared nothing database systems. In *Proceedings of the International Conference on Very Large Databases*, pages 182–193, 1993.

20. Jun Rao, Chun Zhang, Nimrod Megiddo, and Guy M. Lohman. Automating physical database design in a parallel database. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 558–569, 2002.
21. Uwe Röhm, Klemens Böhm, and Hans-Jörg Schek. Olap query routing and physical design in a database cluster. In *7th International Conference on Extending Database Technology (EDBT'00)*, pages 254–268, 2000.
22. Uwe Röhm, Klemens Böhm, and Hans-Jörg Schek. Cache-aware query routing in a cluster of databases. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 641–650, 2001.
23. D. Saccà and G. Wiederhold. Database partitioning in a cluster of processors. *ACM Transactions on Database Systems*, 10(1):29–56, 1985.
24. T. Stöhr, H. Märtens, and E. Rahm. Multi-dimensional database allocation for parallel data warehouses. *Proceedings of the International Conference on Very Large Databases*, pages 273–284, 2000.
25. T. Stöhr and E. Rahm. Warlock: A data allocation tool for parallel warehouses. *Proceedings of the International Conference on Very Large Databases*, pages 721–722, 2001.