Distributed Database Design: A Practical Approach and Example

Toby J. Teorey

Electrical Engineering & Computer Science
The University of Michigan
Ann Arbor, MI 48109-2122 USA
(313) 998-7479

July 1989

Abstract

A distributed database design problem is presented that involves the development of a global model, a fragmentation, and a data allocation. The student is given a conceptual entity-relationship model for the database and a description of the transactions and a generic network environment. A stepwise solution approach to this problem is shown, based on mean value assumptions about workload and service.

1. A Distributed Database Design Problem

Textbooks and courses on distributed database systems lack concrete examples of how the database design process can be extended to include data fragmentation and data allocation strategies. The problem presented here has been carefully designed to provide enough complexity to challenge the advanced database student, but small enough to avoid the drudgery of repetitive hand computation. On this latter point, as the student develops a solution it should become obvious where software tools would be helpful to evaluate large-scale designs.

1.1 The Problem

Given the database description below [Spro76] and the ER diagram representation of that description in Fig. 1, develop a global schema of 3NF (or BCNF) relations. Then design a fragmentation and an initial nonredundant allocation based on the stated needs of individual users at remote sites given in the network topology in Fig. 2. Finally, design an optimal data allocation schema using the "all beneficial sites" method for redundant data [CePe84, CPW87,TCOU89], comparing the design decisions with those from an exhaustive enumeration of transaction costs and feasible allocations.

1.2 Database Description

A customer places an order through a particular salesperson for a given quantity of a specific product that is to be shipped by a certain shipping date. Once the order is filled it is saved for future reference, possibly at a site that is different from the active (unfilled) orders. Customers are considered to be located in certain marketing regions, and a salesperson serves customers within a particular region. The headquarters (HQ) of the company is located at a site that is separate from any regional office.

The database serves to provide information for management decision making, e.g. marketing and sales questions, as well as tracking orders, customers, and salespersons so that customer service is maximized. It also helps management make decisions regarding shipping.

Cardinalities (initially)
Regions=6
Salespersons=1,200
Customers=240,000
Orders (filled)=2,000,000
Orders (unfilled) average level of 2,000 per week per region
Products=10,000

Entities and their attributes(field

width)
Region.....reg-no(2), reg-name(15), manager(20), addr(30), phone(10)
Salesperson.....ales-id(6), sales-name(20), addr(30), phone(10)
Customer.....cust-id(8), cust-name(20), addr(30), phone(10), company(20)
Product.....prod-no(10), prod-type(15), prod-name(20), price(10)
Order.....order-no(12), date(6), prod-no(10), quantity(5), shipping-date(6), date-filled(6), total-price(10)

Note: the designer is allowed to extend these entities and attributes in any way, e.g. adding foreign keys.

1.3 Database Transactions

Database transactions are given as relatively simple queries and updates. After each transaction a frequency is specified, and the source of each tranaction is given as either the headquarters (HQ) or from each of the regions. If the frequency for each region is given, the total frequency is that number multiplied by six, the total number of regions. Assume that the search key value is known when the query is made.

Database Queries

1. Which salespersons service a particular region? [20/day/HQ]
2. What are the unfilled orders for a particular salesperson? [50/day/region]
3. Who are the customers in a particular region? [20/day/HQ]
4. Which customers has a particular salesperson sold to in a particular region?[50/day/reg.]
5. What are the details of the unfilled orders for a particular customer? [100/day/region]
6. How many new orders have there been this past month for all products in each region?[1/mo/HQ]
7. What unfilled orders are for more than $10,000 and who sold those orders? [1/day/HQ]
8. For a given order, filled or unfilled, what is the name and address of the customer and the salesperson? [150/day/region]
9. For a given customer, who is the salesperson? [200/day/region]
10. Which unfilled orders are currently past any of their shipping dates, and for what products? [1/day/HQ]

Database Updates

1. Add a new customer. [one batch of 50/day/region]
2. Move a salesperson to a different region. [2/week/HQ]
3. Place a new order. [400/day/region]
4. Mark an order as filled (shipped). [2400/day/HQ]
5. Update the product catalog with new products. [one batch of 100/quarter/HQ]

Note 1: assume that a "delete" results in setting the data to null, but no reorganization.
Note 2: week=5 days, month=21 days, quarter=63 days, year=252 days (shipping dates).

1.4 Network and Local Site Specifications

A variety of simplifying assumptions are given so that gross estimates of transaction response times can be made. Network contention is assumed to be nil because database design decisions are normally independent of total network load.

1. Generic (ARPANET-like)packet-switched network including an overseas site, S5, at 56 Kbps (see Fig. 2). Pick the shortest distance between two sites and assume a simple protocol with no overhead: send a one packet query or update and receive a one or more packet result with no processing overhead, only disk and transmission delays. Packet size = block size = 2000 Bytes.

2. Propagation delay: speed of light, approximately 300 km per millisecond. Note: this ranslates to 3.3 microseconds per km,

which is idealistic. Alternatively, a common assumption of 5 microseconds per km is used to take cable degradation into account. This model also assumes that station latency overhead is negligible.

3. Trba=40 ms (random block access), Tsba=10 ms (sequential block access) for a 2000 Byte block. Local disk capacity: assume it is sufficient to hold all the allocated data.

4. Local query and update: compute access times based on disk I/O for read and rewrite.

5. Remote query and update: compute access times based on disk I/O for read and ewrite, plus network propagation and transmission delays for the query/update, including the actual data being transferred to answer the query/update. Pick an intelligent (near-optimal) query processing strategy for each query and update; absolute optimality is not necessary.

6. Assume that all relations are initially sorted by the primary key.

7. Assume that the local database systems contain sufficient indexing that all searches for individual records (tuples) based on the key value take one random block access, and if a rewrite is necessary for an update it takes one sequential block access. When joins of relations are required, do appropriate selections and projections first to reduce the cost of a join. If necessary, one or both relations may need be sorted on the attribute used in the join; however, you can specify any attribute for sorting a relation before it is initially stored. Assume that all data is uniformly distributed over the six regions.

2. Global Schema and Fragmentation Design

2.1 Analysis of the Transactions

Two approaches are commonly used to determine which transactions to consider in the design process: either all transactions or a dominant subset. A dominant subset is frequently used when an exhaustive enumeration would be prohibitive; and it is selected on such criteria as high frequency of execution, high volume of data accessed, response time constraints, and explicit priority. We will take the exhaustive enumeration approach first in order to specify a standard for comparison with approximation methods such as dominant subsets of transactions.

Since at this point we are not considering time constraints or priorities of transactions, our selection of a dominant subset will be based primarily on frequency of execution and volume of data searched. Our initial analysis of the transactions will be to estimate the number of tuples accessed to execute each transaction, given some simplifying assumptions about database retrieval and update (see Table 1). Later, as we know more details about physical storage and network distribution parameters, the analysis can be refined.

The dominant transactions in this group are queries Q2 - Q5 and update U1. All other transactions appear to have a significantly lower data volume that this subset. We will test this method and its refinements against exhaustive enumeration in Sec. 3.3.

2.2 Global Schema Design Decisions

The transformation from the entity-relationship model to the relational model is very straightforward in most cases, since the relationships in Fig. 1 are for the most part binary and one-to-many. In such cases we create a relation from each entity in the relationship and add a foreign key in each relation that represents the "many" side. The

foreign key we use is the primary key of the "parent" entity, that is the entity on the "one" side of the relationship [TYF86].

The only additional relationship to consider is the generalization of order from unfilled-order and filled-order. In this case we make the simplifying decision that both unfilled-order and filled-order will have the same scheme, with the attribute date-filled set to null in unfilled-order. We will also assume that filled-orders will be physically near each other and unfilled-orders phyically near each other on each local computer's disk subsystem. Alternatively we could have consolidated both types of order into a single relation, order, to be separated physically by whether or not date-filled has a null value. Analysis of the tradeoffs between these two alternative conceptual designs would occur at the physical level, using the parameters discussed in Sec. 3.

A feasible set of relation schemes are as follows. All relations are assumed to be kept sorted by primary key as noted in the assumptions. Note that primary keys are underlined and foreign keys are shown with an asterisk*:

Initial relation definitions

region.....reg-no(2), reg-name(15), manager(20), addr(30), phone(10)
salesperson.....sales-id(6), sales-name(20), addr(30), phone(10), reg-no*(2)
customer.....cust-id(8), cust-name(20), addr(30), phone(10), company(20), reg-no*(2), sales-id*(6)
product.....rod-no(10), prod-type(15), prod-name(20), price(10)
unfilled-order.....order-no(12), date(6), quantity(5), total-price(10), shipping-date(6), date-filled(6), prod-no*(10), cust-id*(8), sales-id*(6)
filled-order.....order-no(12), date(6), quantity(5), total-price(10), shipping-date(6), date-filled(6), prod-no*(10), cust-id*(8), sales-id*(6)

2.3 Normalization of the Global Schema

Normal forms were derived from the following functional dependencies:

Relation	Functional dependencies
region	reg-no -> reg-name, manager, addr, phone
salesperson	sales-id -> sales-name, addr, phone, reg-no
customer	cust-id -> cust-name, addr, phone, company, sales-id, reg-no sales-id -> reg-no
product	prod-no -> prod-type, prod-name, price
unfilled-order(or filled-order)	order-no -> date, quantity, total-price, shipping-date, date-filled, prod-no, cust-id, sales-id cust-id -> sales-id

Thus, all relations except for **customer**, **unfilled-order** and **filled-order** are in BCNF. The **order** and **customer** relations are considered to be in 2NF because of the existence of a transitive functional dependency. There is, however, no delete anomaly because of the replication of the dependent attributes elsewhere.

2.4 Fragmentation and Nonredundant Allocation

Only horizontal fragmentation is considered here. Vertical fragmentation involves modifications to the relational schema, and approaches to it are well documented in [CePe84, TeFr82]. In general, horizontal fragmentation decisions can be made by studying the relationships between transactions and the relational schema. In a variation of the "best fit" method [CePe84], we analyze each relation in terms of the data

volume (tuples/day) for all transactions that use that relation, and note where the transactions originate. From Table 1 we can make the following nonredundant allocations, noting in parenthesis the transactions that use each relation.

The **region(no transactions)** relation is very small and is used only by the HQ; therefore it is left whole and unfragmented at the headquarters site.

The **product(Q6,U3,U5)** is used by both the HQ and regions, so the initial placement is not clear. There is no obvious need to partition it because each application needs to access the whole relation. Thus it is left unfragmented at either the HQ or one of the regions.

The **salesperson(Q1,U2)** relation should remain unfragmented at the HQ since it is only used for a full relation scan in Q1 by the headquarters. Although the salesperson's id is imbedded in other relations in the various regions for use in queries Q2, Q4, Q8, and Q9; any retrieval of further salesperson information can be made in a single random access since the primary key is known in each case. The HQ allocation would have the additional benefit of making update U2 much easier, since only the region number (regno) would have to be changed to affect a location change for a salesperson.

The **filled-order(Q8,U4)** relation should remain unfragmented at the HQ because the data volume for U4 at the HQ dominates the data volume for Q8 at the regions.

The **customer(Q3,Q4,Q9,U1)** relation is used more at the regional level (Q4,Q9,U1) than at the HQ level(Q3). Therefore we initially fragment the customers by regions.

The **unfilled-order (Q2, Q5, Q7, Q8, Q10, U3, U4)** relation has many transactions at the HQ and at the regional level. If we total the data volume for HQ(Q7,Q10,U4) and regions (Q2, Q5, Q8, U3) we find that the regional activity is much higher. Therefore we initially fragment the unfilled-orders by regions.

Fragmentation done in this manner is clearly a heuristic. In this problem we wish to study the most obvious alternative data allocation strategies for **salesperson, customer, unfilled-order,** and **filled-order**:

1. keep unfragmented and store only at headquarters (HQ)
2. fragment and store by region
3. fragment and store by region; replicate at headquarters (HQ)
4. keep unfragmented at a single region

3. Redundant Data Allocation Methods

3.1 Cost/benefit Analysis: Basic Performance Statistics

To produce a redundant allocation, one only needs to consider allocating additional copies of a relation at sites where there are queries which use that relation. Thus, one need not consider, for example, placing a copy of region 1's salesperson relation at site 2, since site 2 would never query or update this information. This results in the following regional fragments being considered for redundant allocation: **salesperson, customer, unfilled-order,** and **filled-order**.

First we convert the logical schema specification into 2000 Byte block allocations (see Table 2). Let us analyze the I/O time to execute query Q1 "Which salespersons service a particular region?" as an example of how the computations are conducted:

Case 1: Salesperson stored unfragmented at the headquarters.

This query requires a scan of 1200

salesperson tuples (67 blocks) at the HQ and selecting only those from the targeted region. Ignoring the processing overhead and system contention, we calculate simple elapsed time in terms of I/O service time for the 67 blocks:

Simple elapsed time = 67 blocks * Tsba =
67*10ms = 670 ms

Case 2: Salesperson fragmented and stored by region only.

Since the query is initiated by the HQ, the targeted region must be accessed and its fragment fully scanned and transmitted to HQ. This involves the transmission of a single packet from HQ to that region, followed by transmission of 12 packets of salesperson data from the region to HQ. Our simplifying model of wide area networks assumes that each regional request and reply is done sequentially, and that within each regional reply, the propagation delay occurs exactly once and multiple packet replies occur serially. Propagation and transmission delays for this simple network are summarized in Table 3; the average propagation delay is 9.2 milliseconds.

Simple elapsed time = propagation delay from HQ to the targeted region
+ request packet transmission delay from HQ to the targeted region
+ local disk I/O time for the query at the targeted region
+ propagation delay back to HQ
+ reply packet transmission delay back to HQ

= 9.2 ms + 285.7 ms + 12 blocks * 10 ms/block
+ 9.2 ms + 12 packets * 285.7 ms

= 3852.5 ms

Case 3: Salesperson stored at the HQ and replicated in fragments at each region.

This query takes the minimum of Cases 1 and 2 because Case 3 provides for two paths to the same data.

Obviously, if query Q1 were the only transaction, storing the unfragmented salesperson data at HQ (Case 1) would minimize simple elapsed time, disregarding any reliability constraints. On the other hand, if the link were improved to T1 speed (1.544 Mbps), then fragmenting the data at the regions (Case 2) would produce the best performance. However, since there are many more query and update transactions to consider, we present a table (Table 4) that summarizes the cost of each transaction for each of the three data allocation cases. Since update U4 "Mark an order as filled (shipped)" is among the more complex transactions, we show the details of its simple elapsed time computation here.

Case 1: unfilled-orders and filled-orders stored unfragmented at the headquarters.

We still assume that unfilled-orders and filled-orders are physically separated at the HQ. This update involves strictly local accesses, using an index to find the appropriate unfilled-order, marking the deletion, and rewriting the block. Then an index is consulted to find the appropriate point in the filled-orders to insert the new tuple. This involves one access to get the appropriate block and a rewrite of that block with the new tuple.

Simple elapsed time= access unfilled-order block + rewrite block
+ access filled-order block + rewrite block
= 1 Trba + 1 Tsba + 1 Trba + Tsba
= 40 ms + 10 ms + 40 ms + 10 ms
= 100 ms

Case 2: unfilled-orders and filled-orders fragmented and stored by region only.
In this case the local update time is the same as in Case 1 because the updates involve only

random accesses to data followed by sequential rewrites. We assume that the reply from an update is a single packet.

Simple elapsed time= request propagation delay + request transmission delay + local update time + reply propagation delay + reply transmission delay

= 9.2 ms + 285.7 ms + 100 ms + 9.2 ms + 285.7 ms

= 689.8 ms

Case 3: unfilled-orders and filled-orders stored at the HQ and replicated in fragments at each region.

This case requires that both copies of the data must be updated.

Simple elapsed time= simple elapsed time for the HQ update + simple elapsed time for the regional update
= 100 ms + 689.8 ms
= 789 8 ms

Thus update U4 would be best served by the data allocation scheme for Case 1. Table 4 below summarizes all costs for each of the transactions.

3.2 Exhaustive Enumeraton Method

The exhaustive enumeration approach computes the entire cost of executing all transactions that use a particular relation for each allocation strategy and choosing the strategy that minimizes total cost. In this case cost is measured in simple elapsed time for queries and updates, assuming no contention either on the network or at the local sites. Cost computations are summarized for each relation in Table 6. Dominant transactions are shown with an asterisk.

Thus our allocation decision is:

1. Allocate **salesperson** and **filled-order** to the HQ (unfragmented).
2. Replicate **customer** at the HQ(unfragmented) and regions (fragmented by region).
3. Allocate **unfilled-order** to each region (fragmented by region).
4. Allocate product to a single region (unfragmented). Replication at the HQ results in higher cost and is not recommended.

In addition to this, we want to allocate **region** to the HQ (unfragmented), based on previous knowledge.

3.3 Dominating Transactions

We test our hypothesis that the dominating transactions will lead to the same decisions as exhaustive enumeration by recomputing the total costs for the dominating subset (Q2*-Q5*, U1* in Table 6). The **salesperson** relation has no dominating transactions, so no decision is made regarding redundant allocation; thus it remains at HQ. This is consistent with the exhaustive enumeration method. The **customer** relation has dominating transactions Q3, Q4, and U1 which lead to the same decision as exhaustive enumeration. On the other hand, **unfilled-order**, with dominating transactions Q2 and Q5, results in a tie between Cases 2 and 3. If the decision is randomly chosen at this point, a wrong decision could be made. Even worse, **filled-order**, with no dominating transactions, would stay fragmented at the regions when it should be kept unfragmented at the HQ.

We conclude from this test that serious problems exist in the specified method of computing dominant transactions based only on relation tuple counts. A better approach would be to account for packet traffic across the network, since the difference between a remote and local access can be quite large. If

we use the nonredundant cases in Table 5 (Cases 1 & 2) as a more detailed comparison of transaction costs, we can make better allocation decisions.

Let us establish the threshold between dominating and nondominating transactions as the point in which the lower bound cost, i.e. simple elapsed time, of one transaction is more than an order of magnitude greater that the upper bound of another transaction. Then the dominating subset consists of transactions Q2-Q5, Q8, Q9, U1, U3, and U4. Recomputing times from Table 5 using only the newly defined dominating subset, we find that we will make the same redundant allocation decisions as we did with the exhaustive enumeration method. In general, however, it is difficult to specify a good threshold selection policy before looking at the performance data for a specific configuration.

3.4 All Beneficial Sites Method

The "all beneficial sites" method [CePe84, TCOU89] can be used for either redundant or nonredundant data allocation design decisions, and is particularly useful when the number of alternative strategies (cases) for the exhaustive enumeration method is prohibitively large. The all beneficial sites method selects all sites for a fragment allocation where the benefit is greater than the cost for one additional copy of that fragment. One may start with either no copies or one nonredundant copy. In this network individual remote query and update times can be calculated (instead of the average as used in [TCOU89]). The *benefit* at a specific site is measured by the difference in cost to do a remote query, i.e. having no additional copy for a given fragment; and a local query, i.e. having one additional copy of the given fragment so the same query can be done locally. The *cost* at a specific site is the cost of all the additional remote update references for the given fragment at that site.

Total cost for an additional copy of a given fragment at a specific site is the remote update time per reference multiplied by the total number of update (write) references by all user transactions at that site. Total benefit for the same additional copy of a fragment at that site is the difference between remote and local query time per reference times the total number of queries (reads).

In this example, we determined that the initial nonredundant allocation was:

1. **Region, salesperson, and filled-order** are at the HQ.
2. **Customer and unfilled-order** are fragmented by region.
3. **Product** is unfragmented at a single re gion.

We now proceed to determine whether to replicate data or not by computing costs and benefits for each relation (see Table 7). We see that costs dominate benefits for **salespersons, products, filled-orders, and unfilled-orders**; thus no replication of data is needed. However, for **customer** the benefits exceed costs and replication of the relation at the HQ is justified. This result is consistent with the exhaustive enumeration method, and in fact selects from the same computations in Table 4.

As a minor extension of the original problem, consider the following availability constraint: a crash at one site should not cause the loss of data to any of the other sites who may need it now or in future transactions. Part of this is provided by redundancy in the network ring architecture, but data redundancy is also required. Because the **customer** relation is the only relation replicated in the unconstrained problem, we must now create additional copies of all other relations by fragmenting at the regions and maintaining an unfragmented copy at the HQ. This satisfies the availability constraint and maximizes the local query performance at each site.

3.5 Variations of All Beneficial Sites

Numerous variations of the all beneficial sites method have been suggested. In the previous section we assumed an initial nonredundant allocation and looked at one-step variations from this initial allocation. Using a cumulative approach, we make an allocation decision regarding a particular relation, then assume that the next decision must be made based on the new state of the configuration which is no longer the initial allocation. Both the one-step and cumulative approaches are "local optimization" methods and cannot guarantee global optimal solutions.

A more practical approach involves computing actual wait times in the network and local sites, rather than just I/O service times. This results in more realistic elapsed times instead of "simple elapsed times" as a means of comparing different allocation decisions. However, it requires much more knowledge about system contention, which may not be available.
Extensions of all beneficial sites to account for data availability in a partially reliable network are discussed in [CePe84].

4. Conclusions

A distributed data allocation problem was defined and a step-by-step solution approach was given that included conceptual entity-relationship modeling, transformation to normalized relations, fragmentation for a nonredundant data allocation, and finally a redundant data allocation scheme. A variety of practical data allocation strategies were illustrated and evaluated using a simple block access analysis of local databases and a packet propagation and transmission analysis of a simple network configuration for distributed databases. It was shown that the all beneficial sites method has the potential as an effective substitute for exhaustive enumeration when scaling up for large complex databases is required.

5. References

[CePe84] Ceri, S., and G. Pelagatti, *Distributed Databases: Principles and Systems*, McGraw Hill, 1984.

[CPW87] Ceri, S., Pernici, B., and Wiederhold, G., "Distributed Database Design Methodologies", *Proc. of the IEEE*, May, 1987, pp.533-546.

[Spro76] Sprowls,R.C. *Management Data Bases* , Wiley/Hamilton, Santa Barbara, CA, 1976.

[TeFr82] Teorey,T.J. and Fry,J.P *Design of Database Structures* , Prentice-Hall, Englewood Cliffs, NJ, 1982.

[TYF86] Teorey,T.J., Yang,D., and Fry,J.P. "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model," *ACM Computing Surveys* 18,2 (June 1986), pp. 197-222.

[TCOU89]Teorey,T.J., Chaar,J., Olukotun,K., and Umar,A. "Distributed Database Design: Some Basic Concepts and Strategies," *Database Programming and Design* 2,4 (April 1989), 34-42.

Tr.	Tr./day	Approx. number of tuples accessed(per region)	Data volume: Total tuples/day
Q1.	20	200 salesperson tuples (get all salespersons per region)	24k
Q2.	50*6	2000 unfilled-order tuples(get all unfilled-orders per reg.)	600k
Q3.	20	40,000 customer tuples (get all customers per region)	800k
Q4.	50*6	40,000 customer tuples (scan all customers, check salesperson)	12M
Q5.	100*6	2000 unfilled-order tuples (scan all unfilled-orders, check customer)	1.2M
Q6.	1/21	10,000 product count tuples....assume a 20B count tuple for each product is maintained by U3 for each new order (scan all count tuples, check product number and region number)	.5k
Q7.	1	[2000 unfilled-order tuples]*6 (scan unfilled-orders, check total price and salesperson)	12k
Q8.	150*6	1 order tuple, filled or unfilled + 1 salesperson tuple + 1 customer tuple(access order based on key, check customer and salesperson names and addresses)	2.7k
Q9.	200*6	1 customer tuple + 1 salesperson(access customer based on key, check salesperson)	2.4k
Q10.	1*6	[2000 unfilled-order tuples] (scan unfilled-orders, check shipping date)	12k
U1.	1*6	40,000 + 40,050 customer tuples (scan all customers, rewrite to maintain sort by primary key)	480.3k
U2.	.4	100 salesperson tuples + 1 rewrite + 401 salesperson tuples (scan half of salespersons in a region, delete with one rewrite; scan all salespersons to do the insert, then rewrite whole relation)	.3k
U3.	400*6	1 unfilled-order tuple + 1 rewrite + 1 count tuple + rewrite (access last unfilled-order and rewrite with new order, then update the product count per month) + access product for price	12k
U4.	2400	1 unfilled-order tuple + 1 rewrite + 1 filled-order tuple + 1 rewrite (index to an unfilled-order, delete and rewrite; index to a filled-order and rewrite)	9.6k
U5.	1/63	10,000 + 10,100 product tuples (scan all products and rewrite with 100 more products)	.3k

Table 1. Data volume analysis of all transactions.

Relation	Cardinality	Size (Bytes)	Block. factor (BF)	Total. blks.	Blks/region
region	6	77	25	1	1
salesperson	1,200	66	18	67	12
customer	240,000	88	22	10,910	1,819
unfilled-order	2,000	55	36	56	10
filled-order	2,000,000	55	36	55,556	9,260
product	10,000	55	36	278	47

Table 2. Relation physical characteristics.

Region	Distance to HQ	Propagation delay to HQ	Transmission delay to HQ
1	2400 km	8 ms	285.7 ms
2	3900 km	13 ms	285.7 ms
3	3750 km	12.5 ms	285.7 ms
4	750 km	2.5 ms	285.7 ms
5	5100 km	17 ms	285.7 ms
6	600 km	2 ms	285.7 ms
		(Average = 9.2 ms)	

Table 3. Network topological features and delays

Trans.	Freq/day	Case 1 Unfrag. at HQ	Case 2 Frag. at regions	Case 3 Replicated at HQ & regions
Q1	20 @HQ	67 blk*10 = 670	9.2+285.7+12*10 +9.2+12*285.7 = 3853	Min(1&2) = 670
Q2	50*6 reg	local: 334 blk*10 = 3340 remote: 9.2+285.7 +9.2+10*285.7 = 3161 total: 6501	56 blk*10 = 560	Min(1&2) = 560
Q3	20@HQ	10,910 blk*10 = 109,100	local: 1819 blk*10 =18,190 remote: 9.2+285.7 +1819*285.7+9.2 = 519,992 total = 538,182	Min(1&2) = 109,100
Q4	50*6 regs	local: 10,910*10 =109,100 remote: 9.2+285.7 + 1819*285.7 = 519,992 total = 629,092	1819 blk*10 = 18,190	Min(1&2) = 18,910
Q5	100*6 reg	Same as Q2	Same as Q2	Same as Q2
Q6	1/21@HQ (20B count per product)	100 blk*10 = 1000	local: 6*100 blk*10 = 6000 remote: 6*(9.2+285.7 +9.2+100*285.7) = 173,245, total: 179,245	Min(1&2) = 1000
	Case 4: unfrag. at one region, local=1000, remote=173,245, total=174,245			
Q7	1 @ HQ	334 blk*10 = 3340	local: 6*56*10=3360 remote: 6*(9.2+285.7 +9.2+285.7)=3539 total: 6899	Min(1&2) = 3340

Q8	150*6 reg	local: 3 Trba = 120 remote: 9.2+285.7 +9.2+285.7 = 590 total: 710	3 Trba = 120	Min(1&2) = 120
Q9	200*6 reg	local: 2 Trba = 80 remote: 590 total: 670	2 Trba = 80	Min(1&2) = 80
Q10	1 @ HQ	Same as Q7	Same as Q7	Same as Q7
U1	1@regs	local: (10,910 +10,924)*10 = 218,340 remote: 590 total: 218,930	6*(1819+1822)blks *10 = 218,460	Sum(1&2)=437,390
U2	.4 @ HQ Sum(1&2) = 2710	36 blk*10+10 +144 blk*10 = 1810	local: 6 blk*10 +10 +24 blk*10 = 310 remote: 590 total: 900	
U3	400*6 reg	local: 3 Trba+2 Tsba = 140 remote: 590, total: 730	3 Trba +2 Tsba=140	Sum(1&2) = 870
	Case 4: unfrag. at one region, local= 140*1, remote=590*5+140=3790			
U4	2400@HQ	2*(Trba+Tsba) = 100	local: 9.2+285.7+100 = 395 remote: 9.2+285.7=295 total: 689	Sum(1&2) = 789
U5	1/63 @HQ	(278+281)*10 = 5590	local: 6*(47+48)*10 = 5700 remote: 6*590 = 3540 total: 9240	Sum(1&2) = 14,830

Table 4. Simple elapsed time (in milliseconds) for three data allocation schemes (cases) for a single execution of each transaction.

Trans.	Freq/day	Case 1 Unfrag. at HQ	Case 2 Frag. at regs	Case 3 Replicated	Case 4 Unfrag. at reg.
Q1	20 @HQ	13.4	77.1	13.4	
Q2	50*6 reg	1950.3	168	168	
Q3	20@HQ	2182	10763.6	2182	
Q4	50*6 regs	188727.6	5457	5457	
Q5	100*6 reg	3900.6	336	336	
Q6	1/21@HQ	.05	8.5	.05	8.3
Q7	1 @ HQ	3.3	6.9	3.3	
Q8	150*6 reg	639	108	108	
Q9	200*6 reg	804	96	96	
Q10	1 @ HQ	3.3	6.9	3.3	
U1	1@regs	218.9	218.5	437.4	
U2	.4 @ HQ	.7	.4	1.1	
U3	400*6 reg	1752	336	2088	1516
U4	2400@HQ	240	1653.6	1893.6	
U5	1/63 @HQ	.1	.1	.2	.1

Table 5. Cumulative simple elapsed time (in seconds) for three data allocation schemes for total transaction executions per day.

Relation & trans.	Case 1 (at HQ only)	Case 2 (at regions only)	Case 3 (replicated)
salesperson			
Q1	13.4	77.1	13.4
U2	.7	.4	1.1
	14.1 sec	77.5 sec	14.5 sec
customer			
Q3*	2182.0	10763.6	2182.0
Q4*	188727.6	5457.0	5457.0
Q9	804.0	96.0	96.0
U1*	218.9	218.5	437.4
	191932.5 sec	16535.1 sec	**8172.4 sec**
unfilled-order			
Q2*	1950.3	168.0	168.0
Q5*	1950.3	168.0	168.0
Q7	3.3	6.9	3.3
Q8	639.0	108.0	108.0
Q10	3.3	6.9	3.3
U3	1752.0	336	2088.0
U4	240.0	1653.6	1893.6
	6538.2 sec	**2447.4 sec**	4432.2 sec
filled-order			
Q6	639.0	108.0	108.0
U4	240.0	1653.6	1893.6
	879.0 sec	1761.6 sec	2001.6 sec
product	**Case 1**	**Case 4 (unfrag at reg)**	**Case 5 (rep)**
Q6	.05	8.3	.05
U3	1752.0	1516.0	3268.0
U5	.1	.1	.2
	1752.15	**1524.4**	3268.25

Table 6. Exhaustive enumeration of simple elapsed times for all transactions

Relation(initially at) & transaction(from)	Cost of replicating	Benefit of replicating	Decision
salesperson(HQ)			
Q1(HQ)		0 (no query at regions)	do not replicate at regions
U2(HQ)	900 ms*.4/day		
filled-order(HQ)			
Q8(regions)		590 ms*150/day*6 reg. = 531000 ms	do not replicate at regions
U4(HQ)	689 ms*2400/day = 1,653,600 ms		
customer(regions)			
Q3(HQ)		519,992 ms*20/day	replicate at HQ
Q4(regions)		0 (no queries at HQ)	
Q9(regions)		0 (no queries at HQ)	
U1(regions)	218,930 ms*1/day		
unfilled-order(regs.)			
Q2(regions)		0 (no queries at HQ)	do not replicate at HQ
Q5(regions)		0 (no queries at HQ)	
Q7(HQ)		3539 ms*1/day	
Q8(regions)		0 (no queries at HQ)	
Q10(HQ)		3539 ms*1/day	
U3(regions)	690 ms*400/day*6 reg		
U4(HQ)	395 ms*2400/day		
product (one region)			
Q6(HQ)	0	(173,245-1000)*1/21=8200 ms	
U3(regions)	3790*400/day=1516 sec	0	do not replicate at HQ
U5 (HQ)	6180 ms*1/63 = 98 ms	0	
(pick best of solutions starting with no copies)			

Table 7. All beneficial sites computation of costs and benefits

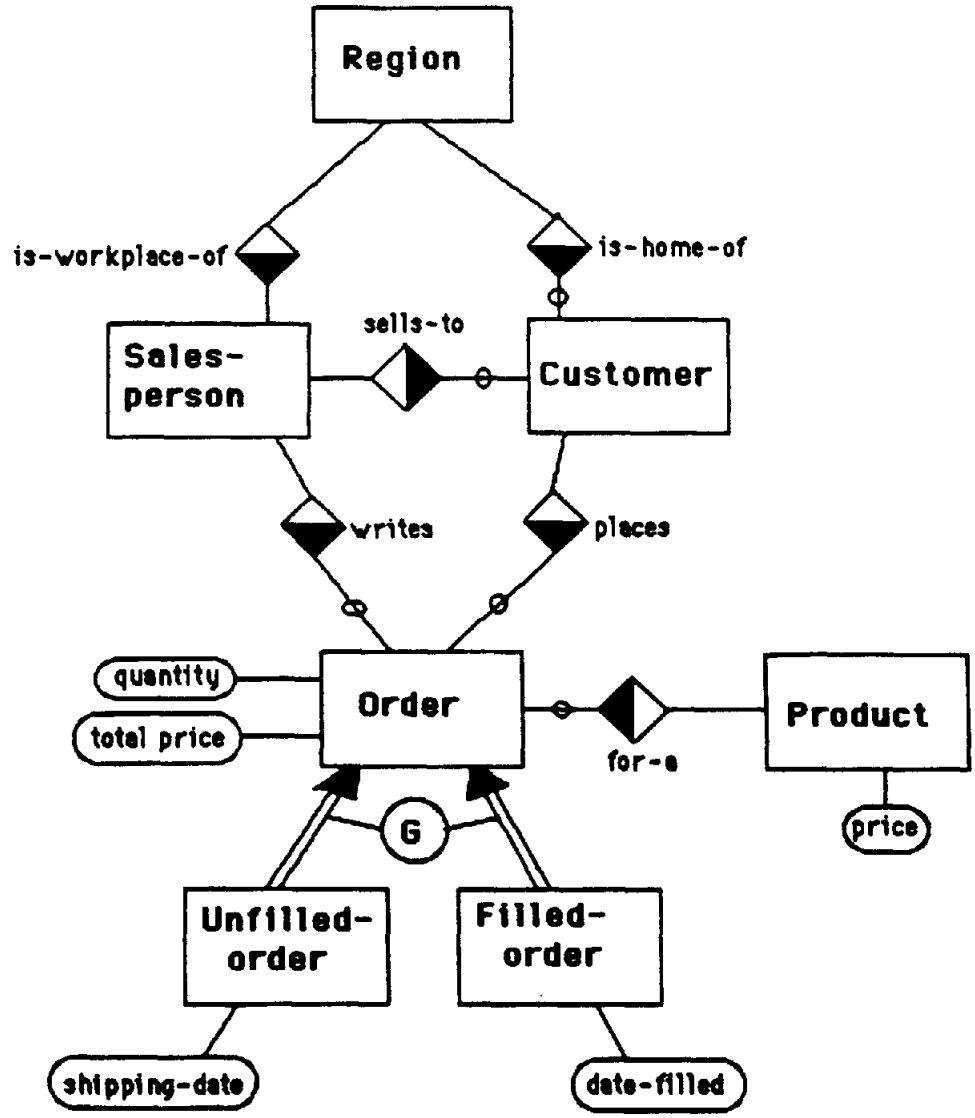

Figure 1 Entity-relationship diagram

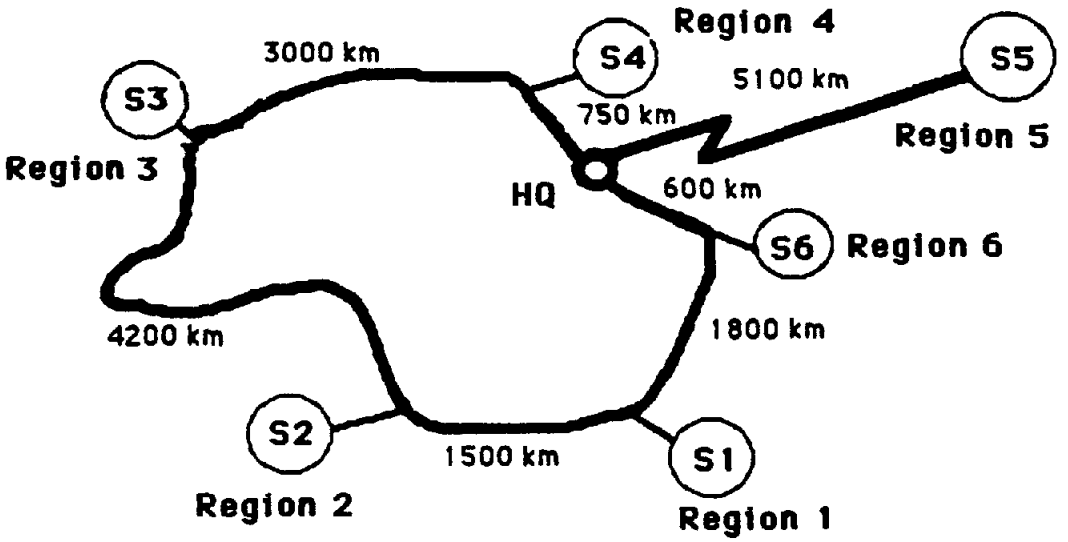

Figure 2 Network topology for the "order" database.