Distributed Database Design Methodologies

STEFANO CERI, BARBARA PERNICI, AND GIO WIEDERHOLD, MEMBER, IEEE

Invited Paper

This paper surveys methodological approaches for distributed database design. The design of distribution can be performed top-down or bottom-up; the first approach is typical of a distributed database developed from scratch, while the second approach is typical of the development of a multi-database as the aggregation of existing databases. We review the design problems and methodologies along both directions, and we describe DATAID-D, a top-down methodology for distribution design. We indicate how the methodology is part of a global approach to database design; how to collect the requirements about the distribution of data and applications; and how to progressively build the distribution of a schema. Our approach is exemplified through one case study.

I. INTRODUCTION

Due to demand for system availability and autonomy, and enabled by advances in database and communication technology, distributed database systems are becoming widespread. Many database management systems now support extensions for distribution, and at the same time structurally distributed transaction management systems are available on the market [11], [19]. The designers of distributed database applications are now facing a new and relevant problem: how to distribute the data and programs on different computers to obtain the intended performance, reliability, and availability. Distribution design is thus emerging as a new problem area of database design, which requires its own theory, design methodologies, and support tools.

The degree of sophistication of distributed database management systems (DBMS) is often measured by the degree of distribution transparency provided to the users. In an ideal situation, the user does not need to be aware of data distribution, and the system takes the responsibility of distributing access operations to the databases at different sites. However, the actual distribution of data affects the overall performance of the system: time and cost required for accessing multiple data objects differ greatly depending on whether all data objects are stored at the same site or are spread over multiple sites. Replication of data affects the overall reliability and availability of the system, because several copies of the same information become available with independent failure modes. At the same time, data replication affects performance, due to the requirement of maintaining the consistency of copies. Thus the database designer must carefully consider the distribution of data, even when the system supports a high degree of transparency.

One key principle in distribution design is to achieve maximum locality of data and applications. While distributed databases enable more sophisticated communication between sites, the major motivation for developing a distributed database is to reduce communication by allocating data as close as possible to the applications which use them. Thus in a well-designed distributed database "90 percent of the data should be found at the local site, and only 10 percent of the data should be accessed on a remote site" [41]. However, it rarely occurs that data and applications can be cleanly partitioned and assigned to a particular site; more often, the designer is faced by tradeoffs because several applications need to access the same data from different locations. In this case, the most effective design is the one which ensures locality to the largest number of applications. A basic assumption of this paper is that the database designer is able to predict both the logical properties which characterize the "locality" of data with respect to applications and the quantitative information measuring the "load" of applications, in terms of frequency of execution requests at each site. The difficulty of designing the distribution of a database is, in fact, due to the interference of logical and quantitative considerations.

A. Top-Down and Bottom-Up Approaches to Distribution Design

Distribution design can be performed top-down or bottom-up. The former approach is typical of distributed data-

Manuscript received November 26, 1985; revised October 30, 1986. This research was partially conducted within the KBMS Project at Stanford University, which is supported by DARPA under Contract N39-84-C-11. It draws heavily on research performed under the DATAID Project in Italy, supported by the CNR within the Progetto Finalizzato Informatica. The conclusions drawn remain the responsibility of the authors.

S. Ceri is with the Dipartimento di Matematica, Università di Modena, Modena, Italy.

B. Pernici is with CSISEI-CNR, Dipartimento di Elettronica, Politecnico di Milano, 20133 Milano, Italy.

G. Wiederhold is with the Computer Science Department, Stanford University, Stanford, CA 94305, USA.

IEEE Log Number 8714302.

0018-9219/87/0500-0533$01.00 © 1987 IEEE

bases developed from scratch, while the latter approach is typical of the development of multi-database systems as an aggregation of existing databases.

The top-down approach assumes that the designer understands the requirements of a database application from the user, and transforms them into formal specifications. During this process, the designer performs *conceptual*, *logical*, and *physical* design phases, which progressively refine high-level, system-independent specifications of the database into low-level, system-dependent specifications. During conceptual design, the designer is expected to ignore any detail concerning the physical implementation (in particular, data distribution). The result is a global database schema which incorporates, at an abstract level, all the data elements of the database and the patterns of their use. A design phase specific to distributed databases, called *distribution design*, maps the global schema to several, possibly overlapping subschemas, each one representing the subset of information which is associated with one site. Then the design of each individual database is completed.

The bottom-up approach assumes, instead, that a specification of the databases at each site exists already, either because there are existing databases that have to be interconnected to form a multi-database (or federated) system ([18], [32]), or because the conceptual specification of the databases has been done for each site independently. In either case, the site specifications have to be *integrated* in order to generate a global specification.

While top-down and bottom-up approaches appear to represent two extremes, in many practical cases the designer proceeds partially bottom-up and partially top-down. We will present the two approaches in separate sections and then discuss the interaction between them.

B. Structure of the Paper

In Section II, we illustrate the top-down approach to distribution design. We indicate how distribution is incorporated into centralized database design methodologies, then we classify the design problems; we also review previous work on the top-down approach.

In Section III we describe DATAID-D, a methodology for the top-down design of data distribution.

In Section IV we illustrate the bottom-up approach to distribution design. We classify the problems, present some solutions, and review previous work on the bottom-up approach. We also discuss how top-down and bottom-up approaches relate to each other.

II. The Top-Down Approach

A general method for designing centralized databases includes four phases: requirements analysis, conceptual design, logical design, and physical design [8], [30], [43], [47].

Requirements Analysis deals with the collection of users' unstructured specifications of the database application, and produces an unambiguous definition and classification of the elements to be considered in the design of the database. The information is collected in a design data dictionary.

Conceptual Design, sometimes further divided in *View Design* and *View Integration*, produces a conceptual specification of a global, integrated database schema and of the applications that are performed on it.

Local Design transforms the integrated conceptual schema into a database schema of a given DBMS type (Relational, Network, or Hierarchical). The choice of DBMS type will be affected by the requirements of the conceptual model as well as by pragmatic considerations.

Physical Design is performed according to the capabilities and features of the particular DBMS chosen, and produces the definition of physical access structures which implement the database.

The design of distribution adds to the above phases an additional one, called *Distribution Design*, which assumes as input a global, site-independent schema and produces as a result the subschemas for each site of the distributed database. In principle, distribution design can be applied to any of the global conceptual, logical, or physical schemas. This choice is subject to the following tradeoff:

- Details about implementation should be decided only when the database distribution is given, to allow concentrating on the physical design of each local database independently. Independent physical design is mandatory if the site DBMSs are heterogeneous.
- On the other hand, a precise description of data and operations helps in estimating the performance of various distributions.

This tradeoff suggests that the distribution design should be performed at the beginning of the logical design phase. At that time, data and operations are described precisely and the first implementation problems are considered.

A. Problem Classification

Database distribution requires determining the *fragmentation* and *allocation* of data.

- *Fragmentation* is the process of subdividing a global object (entity or relation) into several pieces, called fragments.
- *Allocation* is the process of mapping each fragment to one or more sites.

Fragments must be appropriate units of allocation. Thus the database designers should define fragments as homogeneous collections of information from the viewpoint of transaction access [10], i.e., such that all instances of fragments are uniformly accessed by the transactions.

Two types of fragmentations are possible:

- *Horizontal fragments* consist of subsets of the instances (or tuples) of a global object. Each fragment is associated with a predicate (called *qualification*) which indicates the distinguishing property possessed by the instances or tuples of that fragment. It is always possible to reduce an overlapping horizontal fragmentation to a nonoverlapping fragmentation by redefining horizontal fragments in an appropriate way; therefore, we can assume without loss of generality that horizontal fragments are disjoint.
- *Vertical fragments* are derived by projecting global objects over subsets of their attributes. In order for the projection to be lossless [44], it is required that each fragment include a key attribute of the global object, or at least an internal tuple identifier.

Mixed fragmentations can be built by alternating horizontal and vertical fragmentations.

The rationale of horizontal fragmentation is to produce fragments with the maximum potential locality with respect

to operations, i.e., such that each fragment is located where it is mostly used.

The possibility of not partitioning an entity at all should be considered. In particular, if the benefit observed by fragmenting an entity is weak, the overhead due to horizontal partitioning might not be compensated by the advantage of local processing.

The rationale of vertical fragmentation is to cluster attributes frequently used together. An ideal vertical fragmentation exists when each application uses just one subset of attributes; otherwise, some applications will be harmed, since they will need to access both fragments. In this general situation, one has to balance potential benefits (due to the possibility of placing each fragment close to the applications which mostly use it) against potential disadvantages (due to the same applications accessing two fragments).

In the fragmentation process, the designer applies repeatedly horizontal and vertical fragmentation to each object until appropriate fragments are obtained.

The *allocation* of fragments can be either nonredundant or redundant:

- in a *nonredundant allocation*, each fragment is mapped to exactly one site;
- in a *redundant allocation*, each fragment is mapped to one or more sites.

With a redundant allocation, the designer must decide the degree of replication of each fragment. The benefit of replication grows with the ratio between retrieval and updates because maintaining the consistency of the database requires distributing the updates to all the copies. However, the system may permit temporary inconsistencies, and in this case replication becomes more convenient. The benefit of replication decreases with the increase of storage costs, since replicated copies require more space. Moreover, replication increases resiliency from failures, since

- the independent loss of several copies of the same information is unlikely;
- applications can access alternative copies when some failure affects the copies usually accessed.

Given the above classification of problems, the top-down approach to distribution design consists of solving for each global object the following design problems:

horizontal partitioning (H)
vertical partitioning (V)
nonredundant allocation (A)
redundant allocation (R).

While these problems are clearly related, most of the work in the literature has attacked these problems independently.

B. Previous Work on Top-Down Distribution Design

The early work performed on the problem of data distribution was addressed to the file allocation problem; a file was considered to be the "unit of allocation," thus disregarding the fragmentation problem. A knowledgeable user could, of course, present fragments as files to this problem definition. More recently, work in "distributed database design" has centered on both the fragmentation and allocation problems, and their interactions.

1) File Allocation Problem: The file allocation problem was originally investigated by Chu [16], who presented an integer programming formulation of the problem for a fixed number of copies of the file. Casey [6] addressed similarly the problem with a variable degree of replication of each file. Eswaran [23] proved that Casey's formulation is NP-complete.

The file allocation problem was generalized by Mahmoud and Riordon [31] to incorporate the determination of channel capacity, and by Morgan and Levin [35] to incorporate program allocation. Fisher and Hochbaum [24] improved Morgan and Levin's optimization algorithm. Finally, Irani and Khabbaz [28] studied the file allocation problem for distributed supercomputer systems.

Reference [20] provides an excellent survey of several file allocation methods.

2) Distributed Database Design: Research in distributed database design typically assumes the relational model of data; this choice is appropriate because links among remote files are not too useful: remote links are not as beneficial as local links; it is a heavy task to utilize and maintain remote links which violate site autonomy.

Chang and Cheng [14] indicated the way in which a relation can be fragmented by giving a methodology for relation decomposition into fragments. Note that they used the terms horizontal and vertical for describing fragmentation exactly in the opposite way as they are used in all the subsequent literature on fragmentation (and in this paper).

The theory of horizontal fragmentation was studied by Maier and Ullman [33], Paredaens and De Bra [39], and Ceri and Pelagatti [7]. Dependency-preserving horizontal decompositions are discussed in [25].

The problem of determining an optimal horizontal fragmentation for a distributed database has been investigated by Ceri, Navathe, and Wiederhold [9]: the formulation of the nonredundant allocation problem is given by a linear integer program, and heuristics are presented for decomposing the problem into smaller subproblems and for determining a redundant allocation starting from the optimal nonredundant allocation.

Vertical fragmentation was addressed for centralized databases by Hoffer and Severance [27] and by Hammer and Niamir [26]; an application of vertical fragmentation to systems with memory hierarchies (i.e., with a "fast" and a "slow" memory) is presented by Eisner and Severance [21]. March and Scudder have discussed how vertical fragmentation can be useful for the recovery of databases in [34]. Navathe, Ceri, Wiederhold, and Dou have studied the application of vertical fragmentation to distributed databases in [37].

In [42], the problem of distributing the database on a cluster of processors is considered. The motivation of such an architecture is high performance, availability, and reliability. The problem deviates from the standard distribution design because applications also need to be allocated to processors; thus an object to be allocated can be either an application or a fragment. The model considers capacity constraints for the CPUs, processors' input-output, and network communications.

Apers addressed in his thesis the design and allocation of both horizontal and vertical fragments of relations [1],

[2]. In this work, design and allocation are based on query processing strategies, treating nonredundant and redundant allocation of fragments in a uniform way.

III. The DATAID-D Methodology

This section reviews the DATAID-D methodology for distributed database design, developed at the Politecnico di Milano [12]. The emphasis in DATAID-D is on providing a methodological framework to the designer, indicating the relevant design problems, which parameters are required to solve each of them, and how each problem can be approached.

Several tools for distribution design have been developed, providing facilities for documentation and for automatic selection among design alternatives in some subproblems. Design tools for deciding about the horizontal and vertical fragmentation of relations are presented in [7], [9], [37], [42]; their integration within a design support environment is discussed in [13]. All these tools can be used consistently with the DATAID-D approach. In this paper, we stress the methodological approach of DATAID-D and we deliberately omit any discussion about tools; in this way, the results presented here can be immediately applied to design problems, even if the design team has no tools available. Decision processes for selecting among design alternatives are based on simple heuristics rather than complex mathematical formulations; and therefore can be handled manually. We refer to [13] for a discussion about tool support to distribution design.

DATAID-D is built as an extension of the DATAID-1 methodology for centralized database design, which is divided into 4 phases [8]:

- requirements analysis
- conceptual design
- logical design
- physical design.

DATAID-D requires the addition of two phases to the original DATAID-1 architecture (see Fig. 1):

- *Analysis of Distribution Requirements:* This phase is required in order to collect information about distribution, such as partitioning predicates for horizontal fragmentations and the frequency of activation of each application from each site. Since the structure of data and applications must be known in order to collect information about their distribution, distribution requirements are collected starting from some of the results of the conceptual design phase.
- *Distribution Design:* This phase starts from the specification of the global database schema and from the collected distribution requirements, and produces several database schemata, one for each site of the distributed database, each one describing the portion of data that will be allocated to that site.

The input for Distribution Design is a specification of data and applications produced by the previous design phase, in the form of global database schema and logical access tables. Global database schemas are described using a simple version of the ER model [15], which includes just binary relationships; logical access tables indicate the type of database access operations performed by each application on each entity. In DATAID-1, conceptual modeling is performed on a more sophisticated model, called the Extended ER model [8], which also includes generalization hierarchies and n-ary relationships. The first part of Logical Design, Global Logical Design, transforms EER schemata into simplified ER schemata and is performed before distribution design. The final part of Logical Design, which transforms an ER schema into either a relational or a Codasyl schema, is deferred until after distribution design, and is performed independently for each site.

In this paper, we concentrate on distribution; thus we deliberately omit all discussion of requirements analysis, conceptual design, and global logical design (which are performed before distribution design), or of local logical design and physical design (which are performed separately for each site after distribution design). In our example, we will assume that the results of phases preceding distribution design are all available. Refer to [8] for those aspects of the DATAID-1 methodology which are not related to distribution.

Obviously, our clean separation of the design process into phases is an idealization introduced here for simplifying the presentation and also for abstracting the decision process. The actual course of the design of a distributed database will require several feedbacks between phases.

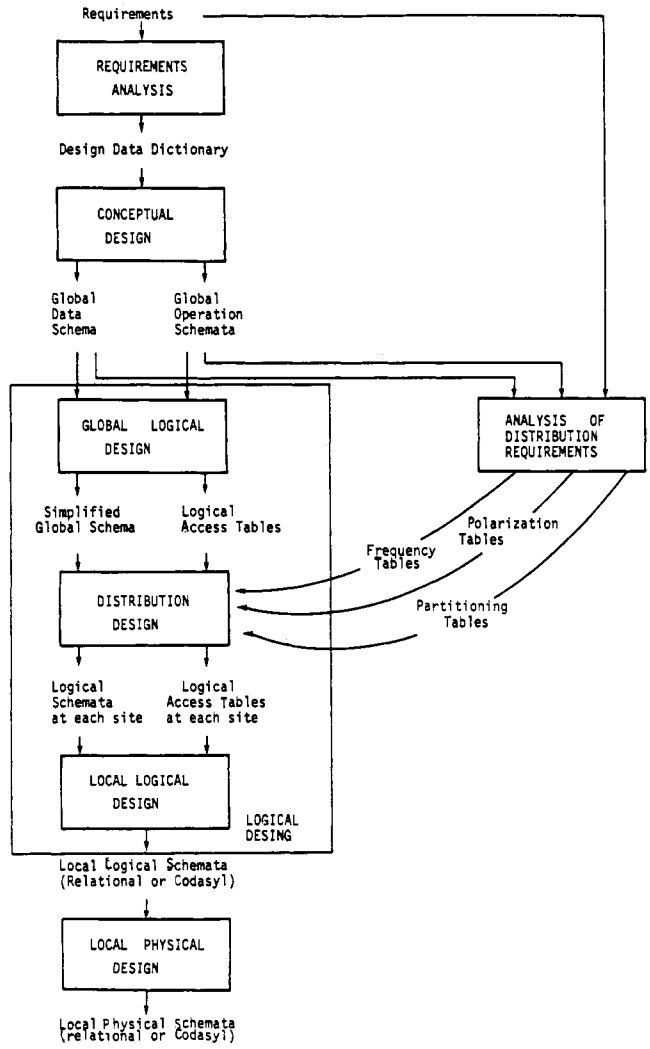


Fig. 1. Architecture of DATAID-D.

A. Analysis of Distribution Requirements

The goal of analyzing distribution requirements is to collect all the information that will later be used to drive distribution designs. Collecting requirements is a difficult process which entails communicating with users to understand their needs; however, in this paper we just stress the results which should be produced, i.e., the type of information that should be collected.

Inputs to this phase are users' requirements for distribution and the global data and operation schemata. Three types of tables are built as output of this phase: a frequency table for the applications, partitioning tables for entities, and polarization tables which relate data and applications.

The *frequency table* gives the number of activations of each application at each site. We assume that all applications are potentially executable at all sites; clearly, when one application is never executed at one site, then the frequency entry in the corresponding position is zero.

The *partitioning tables* indicate potential horizontal partitioning criteria which apply to each entity in the schema. In practice, each partitioning criterion indicates a potential reason for introducing horizontal fragmentation, and is induced by one or more applications which access a given subset of data at a given location.

Two types of paritioning are defined:

- *Primary partitioning*, where the partitioning of an entity E is expressed using several disjoint predicates on the attributes of E. Let p be a generic predicate of some attribute of E; then, a binary partitioning of E is given by the fragments $F1$ and $F2$, such that p holds for tuples of $F1$ and (not p) holds for tuples of $F2$.
- *Derived partitioning*, where the partitioning of an entity $E2$ is determined by a binary relationship (of the ER model) between $E2$ and another entity $E1$ which is already partitioned. Let $E1$ be an entity fragmented into $F11$ and $F12$ and let $E2$ be a second entity connected to $E1$ by the binary relationship R. Then, the derived partitioning of $E2$ is defined as follows:

$F21$ is the set of tuples related to tuples of $F11$ by R;
$F22$ is the set of tuples related to tuples of $F12$ by R.

We also constrain $F21$ and $F22$ to be *disjoint*: hence, not all binary relations R can be used to derive a partitioning; in order for this constraint to hold, it is sufficient (but not necessary) that R be a 1:1 or 1:n relationship from $E1$ to $E2$.

The polarization table indicates, on a quantitative basis, how the partitions influence the locality of processing of applications. A polarization value indicates the probability that a given fragment is accessed by a given application issued by a given site; we indicate only those values which deviate from a uniform distribution.

B. Distribution Design

The goal of Distribution Design is to allocate data at sites, starting from the global data schema, logical access tables, and the distribution requirements.

The output of the Distribution Design phase is a logical schema and logical access tables for each site. These are used during the following local logical design phase and during physical design phases performed independently at each site.

Distribution design in DATAID-D is subdivided into four phases:

- fragmentation design
- nonredundant allocation
- redundant allocation
- reconstruction of local schemata.

Fragmentation design applies horizontal and vertical fragmentation to entities in order to determine possible units of allocation for the subsequent design phases. Each fragment, to be a good unit of allocation, must contain instances which are accessed roughly in the same way (i.e., with the same frequency) by each application executed at each site. This uniform behavior allows considering fragments as uniform units of allocation, during the subsequent allocation phases. Obviously, a too strict application of this criterion would result in fragmentation down to the level of an individual attribute or an individual tuple; there exists a threshold beyond which further fragmentation is not feasible. Fragmentation design is mostly a logical decision, which is performed by selecting some of the predicates from polarization tables and composing them into logically defined fragments.

Nonredundant allocation is performed by mapping each fragment to the site where it is mostly used. The potential frequency of use of each fragment at each site is obtained as the summation over all transactions issued from that site of the product between polarizations and frequencies of use of the fragment. It is thus possible to identify the site which accesses the fragment most frequently, and hence allocate the fragment to that site.

A quantitative measure of the number of accesses to a given fragment from a given site is trivially obtained from frequency and polarization tables. Let

f_{ij} frequency of application i from site j
p_{ijk} polarization of fragment k for application i from site j.

Then the number of accesses to fragment k from site j is given by

$$n_{kj} = \sum_{i:i\,\text{uses}\,k} f_{ij} P_{ijk}.$$

Therefore, fragment k is allocated to the site $\bar{j}$ such that

$$n_{k\bar{j}} = \max_{\text{all}\,j} n_{kj}.$$

Redundant allocation is performed by selecting additional sites for each fragment with respect to the initial nonredundant allocation, using "greedy" heuristics. At each iteration, the most beneficial site for storing a copy of the fragment (if any) is added to the set of allocations. However, the benefit in retrieval accesses must outbalance the major costs and complexity due to updating of redundant copies.

The above benefit is difficult to quantify, since measuring the complexity of managing redundant copies is very difficult. However, it is possible to evaluate the difference between the number of retrieval accesses that become local due to addition of one copy versus the number of addtional remote update accesses required to maintain that copy. This number should be largely positive to justify the increase of redundancy. The difference is evaluated for an object k and a potential new copy stored at site $\bar{j}$ as follows:

$$d_{kj} = \sum_{i:i\,\text{reads}\,k} f_{ij} p_{ijk} - \sum_{i:i\,\text{writes}\,k} \sum_{h \neq j} f_{ih} p_{ihk}.$$

The *reconstruction of local schemata* builds local schemata from the final fragment allocation; this phase is also responsible for the allocation of relationships of the ER global schema. Assuming that most relationships are implemented as associations among the identifiers of the corresponding entities, the DATAID-D methodology suggests placing relationships at the site of the entities or fragments with the largest cardinality, so that few entity identifiers will have to be transmitted.

C. Case Study: Airline Reservation System

1) Case Study Description: In this section, we present an example of the application of DATAID-D methodology to an airline reservation system. The airline maintains a database distributed over three sites, i.e., airports 1, 2, and 3, each of which is at the center of a geographical area; all other airports which are serviced by the company are in one of these areas. To help in visualizing the system, think of a company operating in the U.S., with 1 = Denver, CO, located in the West, 2 = New York, NY, located in the North, and 3 = Atlanta, GA, located in the South. A database stores data about airport regulations, flight schedules, flight availabilities, and passenger reservations.

We do not present here the conceptual and logical design phases; instead, we present their results: the Global Data Schema and the Global Operation Schemata. A Global Data Schema of the database using the ER model [15] is shown in Fig. 2. We assume that each flight is direct from the departure airport to the arrival airport, without intermediate stops. All entities, attributes, and relationships are self-explanatory; we have kept our example as small and simple as possible, though it is sufficiently complex to show some of the problems which arise in distribution design.

Distribution design also requires the collection of knowledge about the most relevant applications that are performed on data. The 80/20 heuristic assures us that we will obtain 80 percent of the accesses by analyzing the most frequent 20 percent of the applications. DATAID-1 collects this information into Global Operation Schemata, which show the use of entities and relationships for each application. Examples of operation schemata for 3 applications are shown in Fig. 3.

The *Make-Reservation* application (Fig. 3(a)) is activated every time that a new passenger (i.e., one not holding another reservation) wants to get a reservation on one flight. In this case, the access to the database is through Departure and Arrival airport, Departure and Arrival time, and the flight's Date. These attributes are marked in the diagram with a "*K*," which indicates that they are used as keys in accessing data. Arrows indicate that the access proceeds from the Airport entities to the Flight entity via the two relations From and To. Numbers in the left and right lower corners of entities indicate, respectively, the total number of instances and the average number of instances selected by the application. Once a flight has been identified, a new instance of the Passenger entity is created, as well as an instance of the relationship Reservation; data about the passenger's Name, Telephone, and Class (corresponding to the ticket's fare) are also written ("*W*") in the database. Notice that the AvailableSeats attribute is first read and then written ("*O*, *W*"; *O* indicates output).

A similar application, called *Continue-Reservations*, allows taking reservations for passengers whose data have already been collected and stored into the Passenger entity; we do not show this application.

The *Check-In* application is executed whenever a passenger actually boards a plane: based on the passenger's Name and on the Number and Date of the flight, the involved Passenger and Flight instances are detected ("*K*" attributes). Then, the Class information is retrieved ("*O*"), and based on this information and on the flight's SeatMap, a SeatNumber is assigned to the passenger. Both SeatMap and SeatNumber attributes are written, together with the number of CheckedBags by the passenger.

The *Airport-Departures* application builds a report describing the departure information for the next 30 flights departing from the airport, to be displayed on TV monitors. The Airport symbol and present Date and Time are used to identify the involved Airport and Flight entities. For each flight, the Number, Departure-Time, Gate, Delay, and destination airport Symbol and City are extracted from the database. Information about the destination airport is determined using the To relationship.

Notice that the last operation schema is linearized, fol-

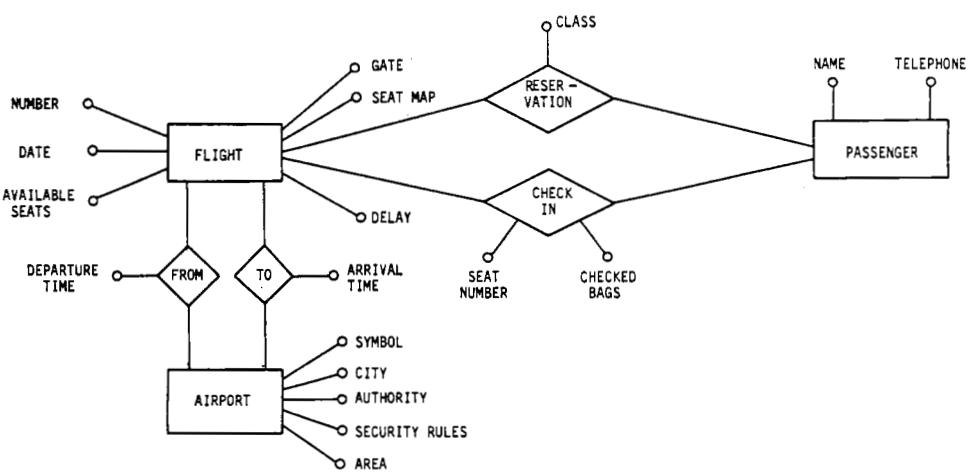


Fig. 2. Global Data Schema of the airline reservation database.

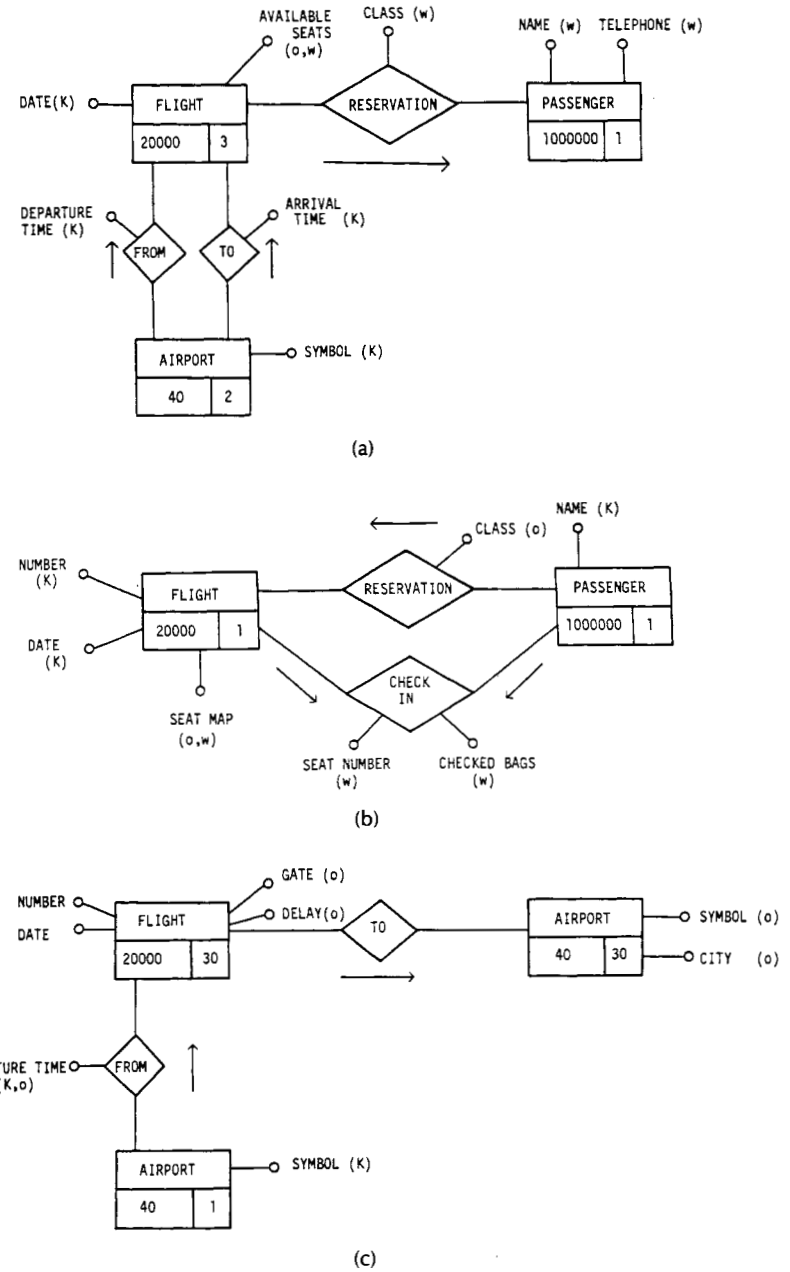


Fig. 3. Global Operation Schemata of the airline reservation database. (a) Make-Reservation. (b) Check-In. (c) Airport Departures.

lowing the application's access path through data. As a more general consideration, notice that operation schemata are "procedural," i.e., they indicate precisely the access path of an application through the database; however, they are built at a logical level, without postulating the existence of access methods. Subsequent design phases (i.e., local logical design and physical design) will decide which access methods have to be provided, based on these schemata and on quantitative information about the applications' load [8].

Finally, quantitative data about applications are summarized for each entity, building the logical access tables. Fig. 4(a) shows the *logical access table* for the entity Flight. Columns correspond to operations, rows correspond to the entity's attributes; positions in the matrix indicate the type of action ("*O*," "*W*," "*K*") performed on objects. The row denoted by *RA* (*relationship access*) indicates whether a relationship was used for accessing the entity, while the row denoted by *AN* (*accesses number*) gives the total number of instances involved in the operation. For a more detailed description, see [8].

2) Analysis of Distribution Requirements in the Airline Reservation System: In this section we examine how distribution requirements are represented by the DATAID-D methodology after the analysis of distribution requirements phase.

The *frequency table* illustrated in Fig. 4(b) shows the frequency of applications *a*, *b*, *c* described in the Global Operation Schemata in Fig. 3 at sites 1 (Denver), 2 (New York), 3 (Atlanta).

Partitioning tables for the Airline Reservation Database are shown in Fig. 5(a) and (b). Fig. 5(b) shows the primary partitioning table for entities Airport and Passenger. The

Entity : FLIGHT

Attributes	Operations		
	a	b	c
NUMBER		K	0
DATE	K	K	K
SEAT MAP		O/W	
GATE			0
DELAY			0
AVAILABLE SEATS	O/W		
RA	FROM TO	RESERVATION	FROM
AN	3	1	30

(a)

Frequency table	Sites		
	1	2	3
Operations			
a	10000	20000	10000
b	8000	12000	8000
c	100	120	100

(b)

Entity access table (a) and frequency table (b).

designer chooses the Area attribute as a partitioning criterion for the Airport entity and the first three digits of telephone number (area code) as a partitioning attribute for the Passenger entity. For each possible value of the partitioning attribute, predicate selectivity gives the percentage of the tuples of the entity with that value.

The designer then considers potential derived partitioning induced by the primary partitioning of Airports into Areas. Four derived partitioning choices are considered in Fig. 5(b):

- The Flight entity can be partitioned in two ways, based on the relationship From (with the departure airport) or To (with the arrival airport) and the partitioning of airports into areas. Notice that the same primary partitioning gives rise to two different derived partitionings based on the use of two different relationships.
- The last two rows show two alternative ways of partitioning the Passenger entity based on the relationships from Reservation to Flight, partitioned by departure areas. Notice that in this case the derivation mechanism is in two steps, since it is applied from Airport to Flight and then from Flight to Passenger. In the former case, passengers are partitioned based on the flight on which they hold the first reservation; in the latter case, passengers are partitioned based on *all* the reservations that they hold. Seven cases are possible (illustrated in the NOTE table): passengers can hold reservations on flights that leave only from one area (*A*, *B*, *C*), or from two areas (*AB*, *BC*, *AC*), or finally from all three areas (*ABC*). The above seven cases are required because Reservation is a many-to-many relationship (each passenger can hold multiple reservations) and yet in the definition of the fragmentation we want to map each passenger in the *real world* to exactly one instance of the Passenger entity. In the first case, each Passenger instance is statically assigned to a partition when the first reservation is made, while in the second case the mapping of Passenger instances to partitions is dynamic: whenever a passenger changes reservation, the corresponding entity instance might move from one partition to another.

Fig. 6 shows the *polarization table*. Columns are associated with activations of applications at each site, rows with partitioning predicates. Each entry indicates the percentage of accesses to a fragment if the particular partitioning criterion is selected. In practice, only a few entries are spec-

Primary partitioning table

Entity	Partition Name	Predicates	Predicate Selectivity
AIRPORT	AREA	AREA = 'W'	30
		AREA = 'N'	45
		AREA = 'S'	25
PASSENGER	TELEPHONE (FIRST 3 DIGITS)	TELEPHONE = '415*' OR '408*', OR...	35
		TELEPHONE = '904*' OR '713*' OR...	35
		TELEPHONE = '212*' OR '617*' OR...	30

(a)

Derived partitioning table

Entity	Association for the derivation	Base Entity	Partition Name	Predicate Selectivity
FLIGHT	FROM	AIRPORT	AREA	SAME
FLIGHT	TO	AIRPORT	AREA	SAME
PASSENGER	RESERVATION	FLIGHT	FIRST RESERVATION LOCATION	SAME
PASSENGER	RESERVATION	FLIGHT	FLIGHT DEPARTURE AREA	NOTE

NOTE:FLIGHT DEPARTURE Partitioning is based on 7 predicates:	Predicate Selectivity
P1 : All flights reserved by the passenger depart from area A	20
P2 : All flights reserved by the passenger depart from area B	20
P3 : All flights reserved by the passenger depart from area C	20
P4 : All flights reserved by the passenger depart from areas A and B	10
P5 : All flights reserved by the passenger depart from areas A and C	10
P6 : All flights reserved by the passenger depart from areas B and C	10
P7 : All flights reserved by the passenger depart from all areas	10

(b)

Fig. 5. (a) Primary partitioning table. (b) Derived partitioning table.

		a			b			c		
		1	2	3	1	2	3	1	2	3
AIRPORT by AREA	P1	80						100		
	P2		75						100	
	P3			80						100
PASSENGER by PHONE	P1	60			60					
	P2		65			65				
	P3			60			65			
PASSENGER by FIRST RESERVATION LOCATION	P1	100			70					
	P2		100			75				
	P3			100			70			
PASSENGER by DEPARTURE	P1	40			40					
	P2		40			40				
	P3			30			30			
	P4	30	20		30	20				
	P5	10		30	10		30			
	P6		20	10		20	10			
	P7	20	20	20	20	20	20			
FLIGHT by DEPARTURE	P1	70			100			80		
	P2		75			100			80	
	P3			70			100			80
FLIGHT by ARRIVAL	P1	100			70			60		
	P2		100			75			60	
	P3			100			70			60

Fig. 6. Polarization table.

ified; the remaining entries can be computed by assuming uniform distribution of the remaining instances. Some submatrices are not relevant because the application does not use the entity; these are crossed out in the table.

To illustrate, let us discuss the first submatrix associated with the Make-Reservation application (Fig. 3(a)) and using the partitioning of the Airport entity by areas (respectively, *P*1 for area 1, *P*2 for area 2, and *P*3 for area 3). The matrix indicates that if we partition airports by areas, then a query about reservations issued at area 1 has 80 percent probability of being related to airports in area 1. For the other two positions in column 1 we assume uniformity over the remaining 20 percent of access.

3) Distribution Design in the Airline Reservation System: Distribution design consists of four steps: the selection of fragmentation criteria for each entity, the determination of nonredundant allocation, the introduction of redundancy over the nonredundant allocation, and, finally, the reconstruction of local schemata at each site (see Fig. 7).

a) Fragmentation design: Given the potential partitioning criteria contained in the polarization tables, the designer must here select the most appropriate one for each entity, and validate that the partitioning itself is convenient. This requires the quantitative analysis of the relevant applications, which can be classified in three classes: those which are made easier by the partitioning, those which are made more difficult, and, finally, those which are not affected. Then, the partitioning is convenient if the first class is

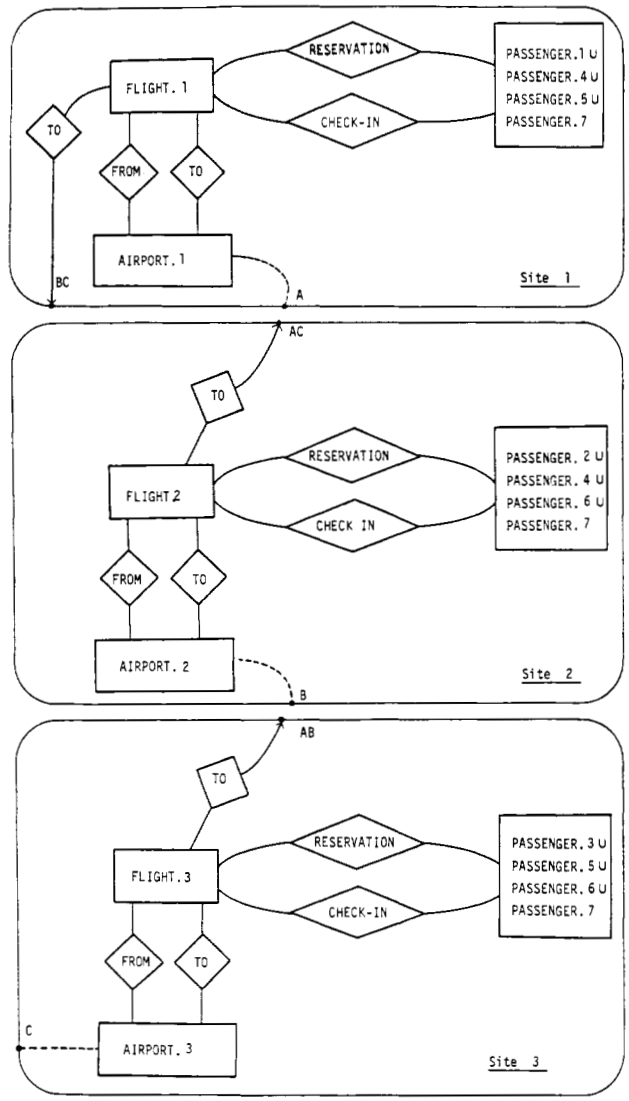


Fig. 7. Result of distribution design: local schemata for sites 1-3.

"larger" than the second class. In our case study:

1) vertical partitioning is not useful to determine units of allocation, in fact, no application exists that can be clearly eased by a vertical partitioning;
2) conversely, all entities have a horizontal fragmentation:
   - the Airport entity has a primary horizontal fragmentation based on Area (fragments Airport.1, Airport.2, Airport.3);
   - the Flight entity has a derived horizontal fragmentation based on the Departure airport (fragments Flight.1, Flight.2, Flight.3);
   - the Passenger entity has a derived horizontal fragmentation based on Departure of all Flights on which the passenger is booked (fragments Passenger.1 to Passenger.7).

b) Nonredundant allocation: In some cases, nonredundant allocation follows easily from the selection of the partitioning criterion. For instance, Airport.1, Flight.1, and Passenger.1 are immediately allocated at site 1, and likewise Airport.2, Flight.2, and Passenger.2 are allocated at site 2

and Airport.3, Flight.3, and Passenger.3 are allocated at site 3. For other fragments of the entity Passenger, we need to select the site which mostly uses the fragment according to the polarization tables and frequency tables; thus we allocate Passenger.4, Passenger.6, and Passenger.7 to site 2, Passenger.5 to site 3.

c) Redundant allocation: Redundancy is then considered; in many cases, costs associated with redundancy outbalance benefits for all fragments of the same entity. This is the case in the Airport and Flight entities. For the fragments of the Passenger entity we introduce instead a limited degree of redundancy: Passenger.4, Passenger.5, and Passenger.6, which correspond to passengers holding reservations on flights departing from two areas, are stored at the two corresponding sites. Likewise, Passenger.7, which includes passengers with flights leaving from all three areas, is stored at all sites.

d) Reconstruction of local schemata: This step is mostly concerned with the fragmentation and allocation of relationships of the ER global data model. According to the above allocation of fragments, the relationships Reservation and Check-In happen to have a natural allocation such that all links are local (because each passenger is associated to one Passenger instance at each site where he holds a reservation or checks in). The From relationship is also naturally allocated, since Airport and Flight entities are partitioned horizontally according to the From relationship; hence, all links are local. Instead, the distribution of the To relationship needs to be discussed, since it links entity instances which may be stored at two different sites. In our solution, the instances of the To relationship are stored at the same site as the corresponding instances of the Flight entity.

The most relevant property of this solution is that all requests about flights can be answered by looking only at the data which are allocated at the departure site of the flight; no remote information is required for preparing for the flight's departure when the database is heavily used. As a drawback, passenger information is replicated, and a careful management of passenger information must be done when reservations are taken.

IV. Bottom-Up Distribution Design

In the bottom-up approach, schemata representing the portion of data stored at each individual site constitute the starting point in the design, and distribution design consists of identifying the data which are common to them, as well as their differences.

During operation, most multi-database systems provide only global query capability and local update capability, so that each local system may only be updated by transactions issued at that site. If the designer cannot modify the local databases of a multi-database system, then conflict resolution has to be incorporated into the query processing capability of the system.

Multi-database support provides an automatic mapping of queries issued according to the global view into queries applicable to the local schemata, and coordinates the execution of queries and the collection of results.

A. Design Problems in Building a Global Schema

Problems in the bottom-up design of a multi-database system are due to the need for *building a global schema* (also called *Superview* in [18], [36]). The integration process recognizes matching entities and their attributes.

To integrate databases we need to select a suitable type of data model for the global schema. Previous work on view integration has demonstrated the power of *generalization hierarchies* to support view integration. A generalization hierarchy allows defining a type-subtype relationship between two entities; this can be useful when two views give a partially overlapping description of the same entity. Then, the classical solution of the view integration problem consists in generating three entities, one with the common attributes and the other two with the nonoverlapping attributes [22]. An example is shown in Fig. 8, where two views of the Flight entity have some common attributes; in the global view, the common attributes are associated with the supertype, and one subtype is generated for each view including the nonoverlapping attributes. Generalization hierarchies are represented by double-line arrows.

The need for generalization hierarchies indicates that conceptual models such as the ER model (extended with generalization hierarchies), the structural model, or the functional model are good candidates for the view integration process. In this paper, we use the ER model extended with generalization hierarchies, as shown in Fig. 8; generalization hierarchies are represented by double-line arrows.

In the example, two views of the Flight entity have some common attributes; in the global view, we generate a generalization hierarchy, with one supertype and two subtypes; the supertype has the common attributes, while the subtypes correspond each to one view; "difference" attributes are related to the subtypes.

Another general question is the *order of integration* of views. When several views are present, integration is typically performed by merging one view at a time with the global schema, which is thus built progressively. Thus the general problem that we consider is how to build the superview of two views. It is, in general, better to integrate first the largest or most important views, followed by the smallest or least important ones.

1) Recognizing Similarities: The first step in the integration of two schemata is to recognize their similarities; these provide the starting point for integration. Matches can be recognized because of the naming or structure similarities of overlapping portions of the schemata. Matches can also be deduced from similarity of data in pre-existing databases or files. Similar value sets indicate overlap. Domains of attributes can be matched by comparing their projections. Observing that files or domains have the same or nearly the same cardinality can be an initial hint.

If different applications on different database locations use copies of the same source data, then there is surely some overlap between the databases.

2) Recognizing Conflicts: Integration should also identify conflicts, i.e., different representations or domain definitions of similiar data in different schemata [3], [22], [38]. Conflicts can be resolved by incorporating differences in the global model or by inducing compromises in the source

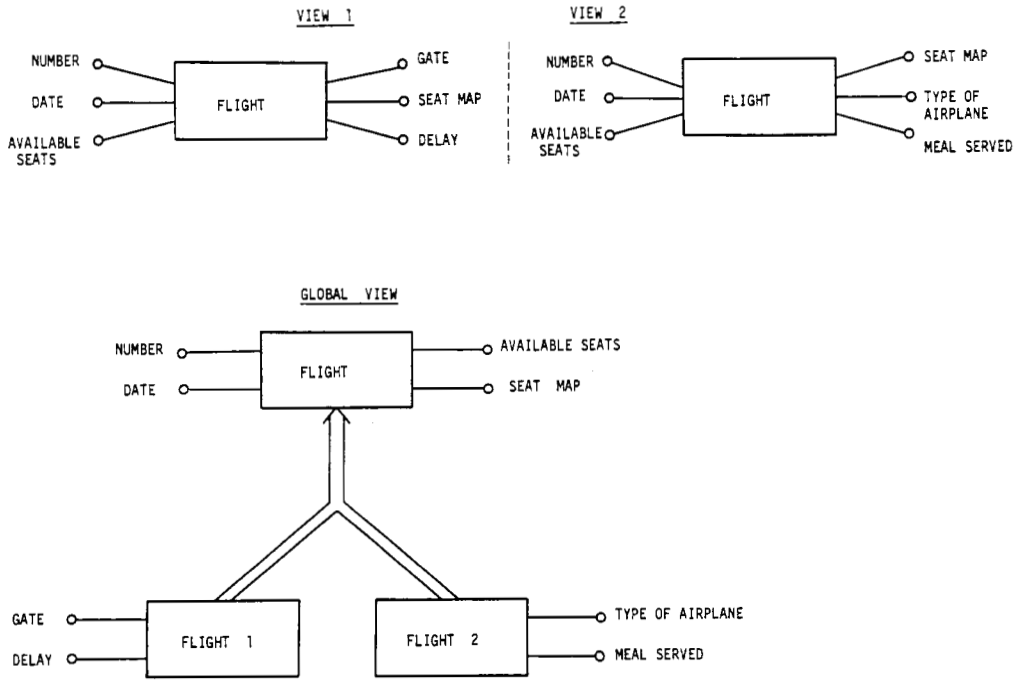


Fig. 8. Merge of two views using generalization hierarchies.

models [47]. Conflicts which cannot be resolved at design time require selection of policies for answering queries with inconsistent data.

Schema differences include naming conflicts, scale differences, and structural differences.

- *Naming conflicts:* There are two types of naming conflicts: *synonyms* occur when data objects which represent the same world objects have been given different names in the two views; *homonyms* occur when data objects with the same name in two views represent different real-world objects. Once naming conflicts are detected, they are easily handled by storing name correspondence tables in the global schema.
- *Domain differences:* The most insidious problems are due to domain differences. A relation Personnel at a site handling the payroll may not include contractor personnel, who are paid indirectly. At the site where projects are managed, contractor personnel are included, but auditors, who are not assignable to any project, are not. Detection of such problems is best achieved by comparing the source databases or files, and noting inconsistencies. Generalization hierarchies can be used for representing the solution to this problem. In the above case, permanent personnel, contractor personnel, and auditors could all be subtypes of the same entity Personnel. Applications, such as payment, are then performed in different ways according to the subtype that they use.
- *Scale differences:* Scale differences may be found in different views of the same numeric values. The data should be retrieved using, if possible, the more precise scale, and joined or output using conversion formulas. Conversion formulas should be stored as a part of the global schema, if the distributed DBMS can support them.
- *Structural differences:* Structural differences may be due to different design choices in each view; for instance, the same real-world object can be modeled as an attribute in one view and as an entity in another view. Only a few of these differences can be dealt with by generalization in the global schema. In view design, structural differences are typically solved by changing one or both views. When dealing with autonomous databases, structural differences typically require writing complex query modification procedures, to be stored in the global schema, if the distributed DBMS can support them [18].

Many of the above conflicts should be reported and fixed before integration, and local systems should be modified to reflect integration whenever possible. Otherwise, the global schema should include information about conflicts, and policies for resolution. Clearly, the management of conversion formulas , query modification procedures, or conflict resolution policies, mentioned in this section, require an extension of the capabilities of traditional DBMS in the direction of more sophisticated multi-database systems; in absence of such extensions, the above features must be carefully supported by application programs.

3) Dealing with Inconsistent Data During Operation: In practice, operational multi-databases have errors; an error rate of up to 1 percent per stored records is not unusual. They may be due to input transcription, omission or failure in synchronization of updates, and improper recovery from system errors. The database designer must decide the *policies* for dealing with inconsistencies that arise during operation of the global database.

Let us consider, for instance, the situation where two instances of the Employee entities stored at different sites happen to have the same identifier, but different values for the salary attribute. This situation can be due to several reasons:

- It is possible that the same identifier corresponds to two different employees; this situation can be solved by sys-

tematically modifying values as they are extracted from the database. For instance, it is possible to introduce system-wide unique identifiers, concatenating all local identifiers with site identifiers.

- Another possibility is that the same employee has two different jobs at the two sites, and the salary attributes just refer to the two different jobs. This may be considered at the level of schema integration, treating the two salary attributes as homonyms. Otherwise, it is possible to manage this data inconsistency by indicating in the *global schema* that all queries interested in the global salary should compute it as the summation of the two local salaries.
- A third reason for discrepancy may be obsolescence. This again can be solved at the schema level, treating the two salaries as homonyms (i.e., old salary and new salary). Otherwise, it is possible to operate at the query modification level, indicating that the most recent value should be used (in this case, the salary attributes must be time-stamped, but obviously this is an additional major cost).
- A fourth reason for the inconsistency presented above is actual incorrectness, due to spurious errors.

The four situations above have shown that the database designer has several options on how to manage inconsistencies. While inconsistencies will be detected at execution time, the determination of policies for solving inconsistencies is a design problem. Policies include:

- Presenting any one of the inconsistent values without notifying the user: the most straightforward but at the same time most dangerous solution.
- Presenting all inconsistent values, and showing to the user the sources of the information. In this case, the user should be able to evaluate the causes of inconsistencies.
- Evaluate some aggregate function on the inconsistent values and present the result of the function to the user. Possible aggregate functions include average, minimum, maximum. This technique was used where observations were expected to differ since they occurred at different times [4].
- Present the most recent value. This policy requires the time-stamping of update operations (which is certainly rather costly). It is based on the assumption that inconsistencies are due to deferred updates, and thus the latest value is also the most likely one.
- Present the value from the most reliable system. This policy is based on the assumption that the designer is able to evaluate the reliability of sites in the distributed database.

B. An Example of Bottom-Up Integration

Assume that two operational airlines decide to have a multi-database system in order to allow queries about flight availabilities from any office of the two companies. Let the conceptual schema illustrated in Fig. 2 and discussed in Section III-C be associated with airline *A*, and let the conceptual schema of Fig. 9 represent the data of airline *B*. Let us look at the features of the schema of airline *B* and compare the schema with the schema of airline *A*. Clearly, the new schema is simpler; in particular, there is no information about airports and check-ins. Information about departure and arrival of flights is represented as attributes of the entity Flight. The basic information about passengers and flights is, however, very similar.

Fig. 10 represents the global schema built after integration. A generalization hierarchy is used to represent two subtypes Flight.*A* and Flight.*B*, while the supertype Flight contains all common attributes. They include the FlightId, called flight Number in schema *A*; here, a synonym is recognized. Common attributes also include the flight Date, which was encoded in the first positions of the attribute Departure Time; here, a more difficult analysis is required. The entity Flight.*A* is related to the entity Passenger through the relationship CheckIn, showing that this information is available only for entities of schema *A*. Finally, the information about the departure and arrival airports is represented in different ways for entities Flight.*A* and Flight.*B*, reflecting the different representations. Notice that a query requesting information about flights departing from a given airport (for instance, the Airport-Departure application of Section III-C) should be translated in very different ways for the two local schemata.

C. Earlier Work in Bottom-Up Distribution Design

Many of the considerations of this section are derived from the paper by Dayal and Hwang [18]. The integration of multi-databases is also discussed by Breitbart and Paolini [5], Litwin [29], Wong and Bazex [48], and Dayal [17]. These papers describe how queries should be decomposed into subqueries to each local system; we have not discussed this topic, since it is more related to the development of a multi-database system than to the design of applications for such systems.

View integration is mainly discussed in the framework of database design for centralized systems by ElMasri and Wiederhold [22], Batini, Lenzerini, and Moscarini [3], and Navathe, Sashidar, and ElMasri [38]. Though generalization hierarchies are modeled in slightly different ways, the use of generalization hierarchies in view integration is suggested by most references in view integration, including [22] (using the Structural Model), [3] (using the EER model), and [18] (using the functional data model).

D. Interaction Between Top-Down and Bottom-Up Approaches

In Section II, we have presented the pure top-down approach, in which the database designer ignores any physical detail (including distribution) when performing conceptual design. While this approach is theoretically valuable, there can be practical situations for which it is

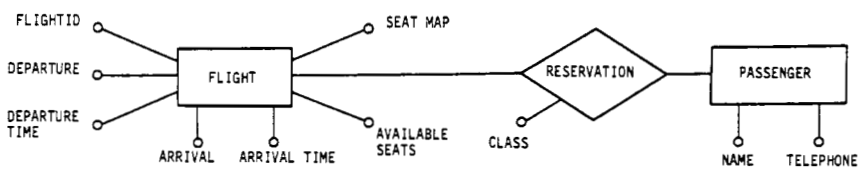


Fig. 9. Conceptual schema of airline *B*.

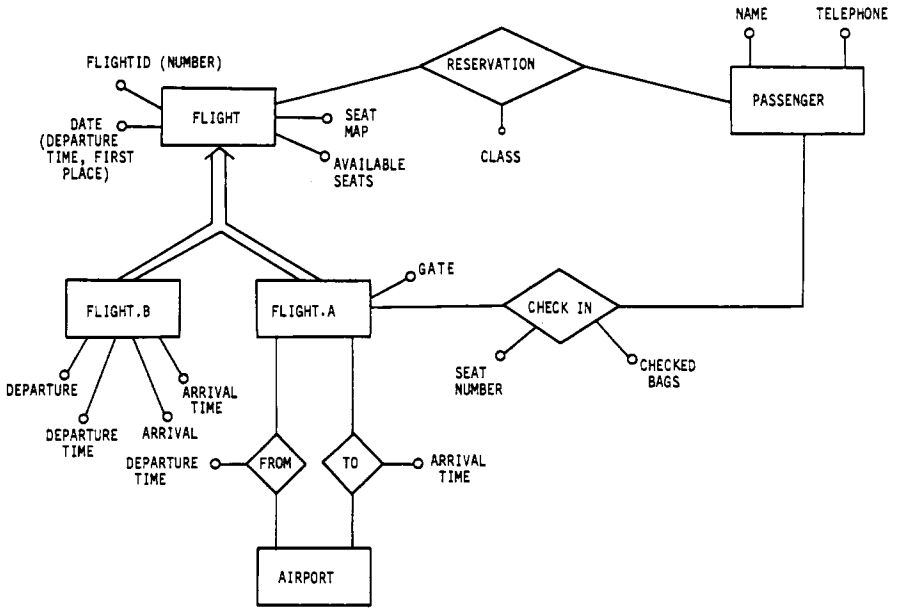


Fig. 10. Global schemata built after integration.

inappropriate. For instance, assume that the database describes an enterprise which is organized by functions, with each function immediately mapped to one database location.

Even if the design is started from scratch, abstracting from distribution information might be unnatural in this case. One possible approach to the design is to proceed partially top-down and partially bottom-up, by collecting requirements and performing conceptual design for each function independently, then integrating all conceptual schemata into a global one, and finally redistributing the information. Clearly, with this approach we retain the notion of the view where each object was originally defined, because in turn this notion will suggest the allocation of the object. However, redistribution is useful for introducing replication and for "moving" some objects from one local schema to another.

In summary, a "pure" top-down approach to distribution is advisable for views that have no correspondence with distribution, while a bottom-up approach is advisable when views have an immediate correspondence to database locations; all intermediate situation are possible and should be dealt with by intermediate approaches.

V. Conclusions

As distributed applications are becoming a reality, distribution design is becoming a new and relevant area within database design. Distribution requires its own theory, problem definitions, solution methods, and methodologies.

This paper has presented a survey of top-down and bottom-up approaches to distribution design and has focused on the DATAID-D top-down methodology; approaches have been exemplified and compared.

Acknowledgment

The authors would like to thank the anonymous referees for providing very useful and detailed comments for revision of this paper.

References

[1] P. M. G. Apers, "Query processing and data allocation in distributed database systems," Ph.D. dissertation, Vrije Universiteit, Amsterdam, The Netherlands, Sept. 1982.

[2] ——, "Data allocation in distributed database systems," to appear in *ACM Trans. Database Syst.*

[3] C. Batini, M. Lenzerini, and M. Moscarini, "Views integration," in *Methodology and Tools for Database Design*, S. Ceri, Ed. Amsterdam, The Netherlands: North-Holland, 1983.

[4] R. L. Blum, *Discovery and Representation of Causal Relationships from a Large Time-Oriented Clinical Database: The RX Project* (Lecture Notes in Medical Informatics, no. 19). New York, NY: Springer-Verlag, 1982.

[5] Y. J. Breitbart and P. Paolini, "Chairmen's report, Session on multi-databases," in *3rd Int. Seminar on Distributed Data Sharing Systems*, F. A. Schreiber and W. Litwin, Eds. Amsterdam, The Netherlands: North Holland, 1985.

[6] R. G. Casey, "Allocation of copies of a file in an information network," in *Proc. 1972 Spring Joint Computer Conf.*, AFIPS, 1972.

[7] S. Ceri, M. Negri, and G. Pelagatti, "Horizontal data partitioning in database design," presented at ACM-SIGMOD, June 1982.

[8] S. Ceri, Ed., *Methodology and Tools for Database Design*. Amsterdam, The Netherlands: North-Holland, 1983.

[9] S. Ceri, S. B. Navathe, and G. Wiederhold, "Distribution design of logical database schemas," *IEEE Trans. Software Eng.*, vol. SE-9, no. 4, pp. 487–504, July 1983.

[10] S. Ceri, B. Pernici, and G. Wiederhold, "An overview in the design of distributed databases," *IEEE Database Eng.* (Special Issue on Database Design), IEEE Computer Society, 1984.

[11] S. Ceri and G. Pelagatti, *Distributed Databases: Principles and Systems*. New York, NY: McGraw-Hill, 1984.

[12] S. Ceri and B. Pernici, "DATAID-D: Methodology for distributed database design," in *Computer Aided Database Design*, A. Albano, V. de Antonellis, and A. di Leva, Eds. Amsterdam, The Netherlands: North-Holland, 1985.

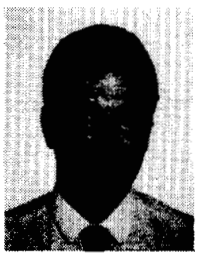
[13] S. Ceri, B. Pernici, and G. Wiederhold, "Optimization problems and solution methods in the design of distributed databases," Politecnico di Milano, Department of Electronics, Computer Laboratory, Internal Rep. 85-7, revised version.

[14] S. K. Chang and W. H. Cheng, "A methodology for structured database decomposition," *IEEE Trans. Software Eng.*, vol. SE-6, no. 2, pp. 205–218, Mar. 1980.

[15] P. P. Chen, "The entity-relationship model: Toward a unified view of data," *ACM Trans. Database Syst.*, vol. 1, no. 1, pp. 9–36, 1976.

[16] W. W. Chu, "Optimal file allocation in a multiple computer system," *IEEE Trans. Comput.*, vol. C-18, no. 10, pp. 885–888,

Oct. 1969.
[17] U. Dayal, "Processing queries over generalization hierarchies in a multi-database system," presented at the 9th Int. Conf. on Very Large Data Bases, Florence, Italy, 1983.
[18] U. Dayal and H. Y. Hwang, "View definition and generalization for database integration in a multibase system," *IEEE Trans. Software Eng.*, vol. SE-10, no. 6, pp. 628–645, Nov. 1984.
[19] *IEEE Database Eng.* (Special Issue on Highly Available Systems), IEEE Computer Society, vol. 6, no. 2, 1983.
[20] L. W. Dowdy and D. V. Foster, "Comparative models of file assignment problem," *ACM Comput. Surv.*, vol. 14, no. 2, 1982.
[21] M. J. Eisner and D. G. Severance, "Mathematical techniques for efficient record segmentation in large shared databases," *J. ACM*, vol. 23, no. 4, Oct. 1976.
[22] R. ElMasri and G. Wiederhold, "Data model integration using the structural model," in *ACM-SIGMOD* (May 1979), pp. 191–202.
[23] K. P. Eswaran, "Placement of records in a file and file allocation in a computer network," *Informat. Processing* (Amsterdam, The Netherlands: North-Holland), 1974.
[24] M. L. Fisher and D. S. Hochbaum, "Database location in computer networks," *J. ACM*, vol. 27, Oct. 1980.
[25] J. Grant, "Constraint preserving and lossless database transformations," *Inform. Syst.*, vol. 9, no. 2, pp. 147–156, 1984.
[26] M. Hammer and B. Niamir, "A heuristic approach to attribute partitioning," presented at ACM-SIGMOD, Boston, MA, 1979.
[27] J. A. Hoffer and D. G. Severance, "The use of cluster analysis in physical database design," in *Proc. 1st VLDB Conf.*, 1975.
[28] K. B. Irani and N. G. Khabbaz, "A methodology for the design of communication networks and the distribution of data in distributed supercomputer systems," *IEEE Trans. Comput.*, vol. C-31, no. 5, 1982.
[29] W. Litwin, "MALPHA, A multi-database manipulation language," in Proc. *IEEE Data Engineering Conf. 1* (Los Angeles, CA, Apr. 1984).
[30] V. Lum *et al.*, "1978 New Orleans Data Base Design Workshop Report," IBM Rep. RJ2554 (33154), IBM Res. Lab., San Jose, CA; and in *Proc. Int. Conf. on Very Large Data Bases*, 1979.
[31] S. Mahmoud and J. S. Riordon, "Optimal allocation of resources in distributed information networks," *ACM Trans. Database Syst.*, vol. 1, no. 1, 1976.
[32] D. McLeod, "A federated architecture for database systems," *NCC*, pp. 283–289, 1980.
[33] D. Maier and J. Ullman, "Fragments of relations: First hack," in *Proc. XP2 Workshop*, 1981.
[34] S. T. March and G. D. Scudder, "On the selection of efficient record segmentations and backup strategies for large shared databases," *ACM Trans. Database Syst.*, vol. 9, no. 3, 1984.
[35] H. L. Morgan and K. D. Levin, "Optimal program and data location in computer networks," *Commun. ACM*, vol. 20, no. 5, pp. 315–322, May 1977.
[36] A. Motro and P. Buneman, "Constructing superviews," in *Proc. SIGMOD '81*, pp. 56–64, 1981.
[37] S. B. Navathe, S. Ceri, G. Wiederhold, and J. Dou, "Vertical partitioning algorithms for database design," *ACM Trans. Database syst.*, vol. 9, no. 4, 1984.
[38] S. B. Navathe, T. Sashidar, and R. ElMasri, "Relationship merging in schema integration," in *Proc. 10th Conf. on Very Large Data Bases* (Singapore), 1984.
[39] J. Paredaens and P. De Bra, "On horizontal decompositions," in *Proc. XP2 Workshop*, 1981.
[40] D. Reiner *et al.*, "The database design and evaluation workbench (DDEW) project at CCA," *IEEE Database Eng.* (IEEE Computer Society), vol. 7, no. 4, Dec. 1984.
[41] J. B. Rothnie and N. Goodman, "A survey of research and development in distributed database management systems," in *Proc. 3rd Int. Conf. on Very Large Data Bases*, 1977.
[42] D. Saccà and G. Wiederhold, "Database partitioning in a cluster of processors," *ACM Trans. Database Syst.*, vol. 10, no. 1, Mar. 1985.
[43] T. J. Teorey and J. P. Fry, *Design of Database Structures*. Englewood Cliffs, NJ: Prentice-Hall, 1982.
[44] J. Ullman, *Principles of Database Systems*, 2nd ed. Rockville, MD: Comput. Sci. Press, 1982.
[45] K.-Y. Whang, G. Wiederhold, and D. Sagalowicz, "Separability. An approach to physical database design," *IEEE Trans. Comput.*, vol. C-33, no. 3, pp. 209–222, Mar. 1984.
[46] G. Wiederhold, "A method for the design of multi-objective databases," in *Computers in Engineering 1982*, vol. 4, R. Raghavan, Ed. New York, NY: Amer. Soc. Mech. Eng., 1982.
[47] ——, *Database Design*, 2nd ed. New York, NY: McGraw-Hill, 1983.
[48] K. K. Wong and P. Bazex, "MRDSM: A relational multidatabases management system," in *Third International Seminar on Distributed Data Sharing Systems*, F. A. Schreiber and W. Litwin, Eds. Amsterdam, The Netherlands: North Holland, 1985.



Stefano Ceri is Professor of Computer Science at the Dipartimento di Matematica Pura ed Applicata, University of Modena, Modena, Italy, and a Visiting Professor at Stanford University, Stanford, CA. While working on this paper, he was at the Dipartimento di Elettronica of the Politecnico di Milano, Milan, Italy. His research interests include distributed databases, database design, and the use of databases in software engineering and logic programming. He is author of numerous articles in these areas and co-author of the book *Distributed Databases, Principles and Systems*, published by McGraw-Hill.

Barbara Pernici is a Research Associate at the CSISEI Research Center of the Italian National Research Council, Department of Electronics, Politecnico di Milano, Milan, Italy. She has been with the Department of Electronics since 1981. Her research interests include distributed database design, office information systems modeling and design.

Dr. Pernici is an affiliate member of the IEEE Computer Society and a member of ACM.

Gio Wiederhold (Member, IEEE) received a degree in aeronautical engineering in the Netherlands in 1957 and the Ph.D. degree in medical information science from the University of California in San Francisco, in 1978.

He is an Associate Professor of Medicine and Computer Science (Research) at Stanford University, Stanford, CA, where he also holds a courtesy appointment in Electrical Engineering. As a member of the Computer Systems Laboratory and the Center for Integrated Systems, he is active in the application and development of knowledge-based techniques to database management. Applications of this work are found in medicine, design, and planning. He has led the development of flexible file and database systems to support clinical research. He consults for government and commercial enterprises, among them the United Nations Development Program, the U.S. Department of Health and Human Services, the U.S. Navy, and several European Banks. He is chairing groups for advanced database development within Ada and for Engineering Information systems. He is also Chairman of the IEEE Technical Committee on Database Engineering and of the IEEE Data Engineering Conference in 1987. He is an Associate Editor of ACM's *Transactions on Databases* and of Springer-Verlag's *MD Computing* magazine. He recently left the editorial board of *IEEE Computer* to help with the *IEEE Expert* magazine. He has over 130 publications in computing and medicine, including an earlier McGraw-Hill textbook *Database Design* and a book *Databases for Health Care*.

Dr. Wiederhold is a member of AAS, AAMS, AIAA, ACL, ACM, ASTM, The Combustion Institute, and TIMS.