DATA BASE DESIGN TOOLS:

A Survey of Current Research, an Opportunity for Customer Input

Abe Lederman
Hewlett-Packard
Information Networks Division
Cupertino, California

ABSTRACT:

Designing a data base can be a time consuming, tedious, and complex task. The ad-hoc methods used to design a data base often lead to data bases that are poorly designed and which adversely affect overall system performance.

In this paper we identify and briefly describe tools that can assist with the different phases of data base design, introducing methodology to the process and automating some of the more tedious aspects of data base design.

The purpose of this paper, and the accompanying presentation is to introduce our audience to current data base design tools research and give them an opportunity to give us at Hewlett-Packard some feedback as to the usefulness of such tools. A survey on data base design will be distributed at the presentation.

In writing this paper, the author has drawn on work that has conducted at Hewlett-Packard part of an investigation into data base design tools. At this time Hewlett-Packard has not made any commitment to actually develop any of the tools described in this paper

1. INTRODUCTION

Designing a data base can be a time consuming, tedious, and complex task. This is especially true when large data bases, consisting of hundreds of items and many data sets, are designed. In most cases data bases are designed in an ad-hoc trial-and-error manner without the benefit of methodologies or tools to assist in the design process.

The ad-hoc methods used to design a data base often lead to data bases that are poorly designed. A poor data base design will likely result in an information system with poor performance. In addition a poor data base design will make it harder to develop the application software to operate on information in the data base. Once a poorly designed data base is up and running, it can be very costly to restructure the data base to correct the design errors that should have been detected during the design phase.

For the past several months I have been involved in an investigation at Hewlett-Packard to try and determine what kinds of tools Hewlett-Packard can develop to help our customers design both network (IMAGE) and relational data bases. In this paper I am going to share with you some of what I have learned in this investigation.

This paper divides the data base design process into six phases. For each phase I identify and briefly describe tools that can assist with the task of data base design. It is not possible in a paper this brief to go into as much detail as is necessary to do justice to the material presented. The goal of this paper is to introduce the reader to data base design tools research. A select number of papers addressing different areas of data base design tools research are referenced throughout the paper.

I feel, and will try to show in this paper, that the use of data base design tools can :

- **Improve design quality**
- **Shorten design cycle**
- **Improve data base performance**

We are interested in learning how you currently design your databases, the difficulties you encounter, and what tools (other than those described in this paper) will help you be more productive and more successful in your data base design activities. At the conference we will distribute a survey on data base design which we strongly encourage you to fill out and return to Hewlett-Packard.

One thing to keep in mind in reading through this paper is that I anticipate that some of the data base design tools described in this paper will be mostly useful in projects involving the design of large data bases.

2. DESIGN PHASES

I have chosen to divide the data base design process into six phases :

REQUIREMENT SPECIFICATION - Information-oriented and processing-oriented requirements are collected and analyzed.

VIEW MODELLING - Local Entity-Relationship models are constructed from the information-oriented requirements. The Entity-Relationship (E-R) model is a conceptual organization of the data which allows the user to simply think of data objects (entities) and the relationships between them. The E-R model is described in section 2.2.1.

VIEW INTEGRATION - Local E-R models constructed in the previous phase are integrated into one consistent global conceptual schema.

SCHEMA MAPPING - Global conceptual schema is translated into a normalized logical schema for the target DBMS.

PHYSICAL DESIGN - Transaction models are constructed for the most active transactions. These models are input to physical design tools which assist with such design decisions as record segmentation, record joining, and the selection of primary and secondary indices.

DESIGN EVALUATION - Data base performance is estimated through the use of analytical modelling tools and/or data base prototyping tools.

The process of data base design is iterative in nature, both within a given phase, and among the different phases. For example, an inconsistency in the view integration phase may require a change to a local E-R model developed in the view modelling phase. This in turn may require modifying some requirements specified in the requirement specification phase. Similarly, the physical design and design evaluation phases form an iterative loop. A physical design change will require the design to be re-evaluated, which in turn may require additional changes to the physical design, and so on. Data base design tools must facilitate this iterative nature of the data base design process.

Let us now take a look at what kinds of tools can be developed to assist in each of these phases of the data base design process.

2.1 Requirement Specification

The requirement specification phase provides the initial input to the other phases of the design process. This input as proposed by Kahn [Kahn] should consist of both information-oriented and processing-oriented requirements. Information-oriented requirements describe the structure of the information that one is trying to model, and are used to construct the local E-R models in the view modelling phase. Processing-oriented requirements describe the transactions that will be performed against the data base, and are used to construct transaction models (see section 2.5.1) in the physical design phase.

Requirement specification tools should facilitate the input, edit, output, and verification of requirements. These tools should also enable designers to document the data base requirements and generate requirement analysis reports in a form that can be reviewed by end-users. Further

investigation is required to determine whether tools should be provided to assist the designer in mapping requirement specifications into E-R models and transaction models. Alternatively, the initial input of requirements to the data base design tools might occur in the view modelling and transaction modelling phases.

Data base requirement specification may be viewed as a subset of gathering requirements for the information system as a whole. Although a great deal of research [IEEE] into high level languages for requirement specification and analysis of information systems has been conducted, the specific problem of data base requirement specification has not been addressed to any significant extent. One possibility is to provide tools to automatically extract data base requirements from information system requirements.

2.2 View Modelling

In the view modelling phase of the data base design process we take the information-oriented requirements collected in the previous phase and represent them in a form that can be manipulated by the design tools. In this section I describe an E-R model that is used to model the information-oriented requirements.

The design of a large centralized data base requires the input of many users. One way in which this task can be simplified is by independently specifying the requirements for the different functions of an organization. Each such set of requirements is called a user view. For example, user views might be specified for the payroll, personnel, inventory and accounting functions of a company. From each user view independent local E-R models are developed.

The number of user views required to model a data base is dependent on the complexity of the data base design, and on the number of transactions that are encompassed by each user view. At one extreme [Hubbard, Yao], a local E-R model is constructed for each transaction against the data base. At the other extreme, the system analyst or DBA manually performs the task of view integration and only constructs one E-R model for the entire data base. A more common approach is for a view to encompass multiple transactions, which are typically the transactions that a user is responsible for, or the transactions for a logical subdivision of an enterprise. This approach, of modelling the information structure for a set of transactions with one E-R model, requires less input during the modelling phase, and results in less problems during the view integration phase, since a smaller number of views need to be integrated.

2.2.1 The E-R model

The Entity-Relationship-Attribute (E-R) model developed by Chen [Chen] provides a fairly natural way for modelling the information-oriented requirements. The E-R model, with various extensions, has been used in much of the research into computer-assisted data base design tools [Batini].

The E-R model consists of entities, attributes, and relationships among entities. An entity is any identifiable "thing". Entities of the same type are grouped into an entity-set. EMPLOYEE, DEPARTMENT, and CAR are some examples of entities.

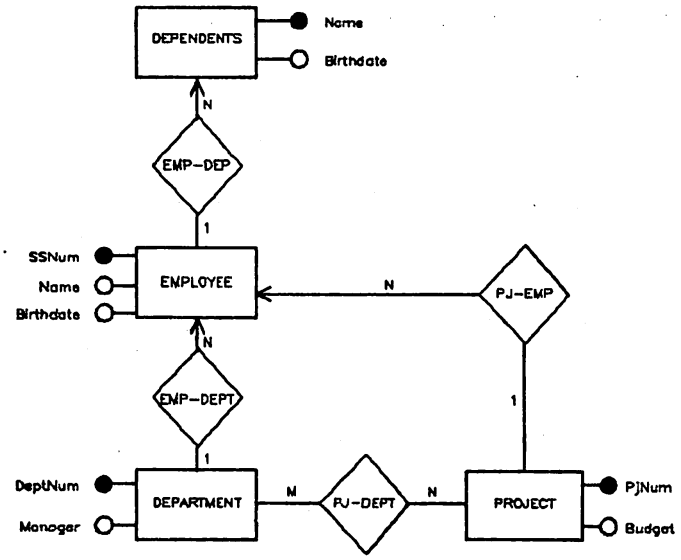
Entities have properties, called attributes. For example, the EMPLOYEE entity may have the following attributes : name, social security number, address, and birth-date An attribute or set of attributes that uniquely identify an entity in an entity-set is called a key.

A **relationship** expresses an association among two (typically) or more entities. PJ-EMP is an example of a relationship between EMPLOYEE and PROJECT. Relationships can be classified into three categories : **one-to-one (1:1), one-to-many (1:N),** and **many-to-many (M:N).**

A graphical technique, the Entity-Relationship Diagram (ERD), may be employed to visually depict entities and relationships. In an ERD :

1. Entity sets are represented by a rectangular box.
2. Attributes are represented by circles.
3. Key attributes are represented by filled-in circles.
4. Relationships are represented by a diamond-shaped box. The relationship type (1:1, 1:N, M:N) is labelled along the lines connecting the entity boxes to the relationship diamond. For 1:N relationships an arrow points from the 1-entity to the N-entity.

The diagram below is an example of a very simple ERD.



There are two main reasons why the E-R model (as opposed to a network or relational model) is used to model the local views. First, at this early stage in the design process, it is important that the design be DBMS independent. The designer should defer the decision of selecting a target DBMS until the schema mapping phase. The E-R model provides a neutral model that is natural to use, straightforward to translate into a relational schema, and only a bit trickier to translate into a network schema. Second, the E-R model permits more of the semantics of the information system being developed to be captured than is possible with the network or relational model.

Graphic-oriented tools, similar to CAD (Computer Aided Design) tools used in various engineering disciplines, might provide a suitable interface for de eloping the E-R models. A graphic-oriented interface might allow system analysts and end-users to work together in developing their E-R models.

2.3 View Integration

The objective of the view integration phase is the merging of the local E-R models, developed in the previous phase, into a single, consistent global schema. This phase is obviously not required if only one view was created in the view modelling phase.

The first step in view integration involves identifying and resolving inconsistencies and redundancies among the local E-R models. Although design tools will not actually resolve inconsistencies or redundancies, they can assist the DBA or system analyst by identifying potential integration problems.

Some examples of inconsistencies that might occur include :

- SYNONYMS - The use of different names for the same object (entity, relationship or attribute).
- HOMONYMS - Using the same name for different objects.
- Type/size inconsistencies.
- Relationship inconsistencies. For example, defining a relationship between two entities as 1:N in one instance and as M:N in another instance.
- An entity is defined as an attribute of another entity.

Two examples of redundancies that might occur include :

- Redundant relationships. For example, A (an entity) is related 1:1 to B, B is related 1:1 to C, and A is related 1:1 to C. This last relationship is not necessary since it can be derived from the other two relationships.
- Redundant attributes. Attributes that appear in more than one entity. For example, NAME is defined as an attribute of both EMPLOYEE and EMPLOYEE-BENEFITS (with SS-NO as the key).

Once all the inconsistencies and redundancies have been resolved, the next step is to actually merge the local E-R models into the global conceptual schema. A design tool can perform this task, although the designer may wish to control how the merging is performed where alternative ways of merging exist.

2.4 Schema Mapping

In this phase, the global conceptual schema is mapped into a schema for the target DBMS. This phase might be viewed as part of the physical design phase. Physical design decisions such as index selection, however, are deferred until the physical design phase. In this phase we are mapping from the E-R model, expressing the global conceptual schema, into a "vanilla" schema of the target DBMS.

Mapping into a relational model is straightforward, while mapping into a network model may be somewhat trickier Tools can be provided to perform the schema mapping. As with view integration, alternative mappings may exist, requiring the designer to interact with the mapping tool in choosing among the different alternatives.

As part of the process of schema mapping, we want to generate a schema that is normalized. The algorithms to generate a normalized schema may be incorporated into the schema mapping tool. I now briefly describe normalization, an important concept in data base design.

2.4.1 Normalization

Normal forms [Kent] were first defined for relational data bases, although they are also applicable to the design of network data bases such as IMAGE. The normal forms can be viewed as guidelines to be followed in designing a data base in order to avoid update problems resulting from redundant data. These guidelines are for the most part common sense rules that most data base designers follow, perhaps without knowing that they are performing normalization.

Take as an example the SUPPLY table below. The key is the composite (S#, PART#). CITY, however, is only dependent on S#.

S#	PART#	QTY	CITY
S1	P1	300	New York
S1	P2	200	New York
S1	P3	400	New York
S2	P2	250	Palo Alto
S2	P4	500	Palo Alto
S3	P1	300	Sunnyvale

The above table design has several problems:

- CITY is repeated for every record having a PART# supplied by the same S#, resulting in wasted storage.
- If CITY changes for a given S#, then every record specifying a PART# supplied by the same supplier must be updated. In addition to the cost of doing all these updates, inconsistencies might arise with different records showing different cities for the same supplier.
- If at some point there are no parts supplied by a supplier, then there is no record in which to store the supplier's CITY.

The solution, obvious enough, is to normalize the above table by decomposing it into the two normalized tables shown below.

S#	PART#	QTY
S1	P1	300
S1	P2	200
S1	P3	400
S2	P2	250
S2	P4	500
S3	P1	300

S#	CITY
S1	New York
S2	Palo Alto
S3	Sunnyvale

It should be noted, however, that it is not always desirable to strictly follow the normalization guidelines. There may be times where for performance reasons, it may be advantageous to maintain redundant data in a table. The normalized schema generated by the schema mapping tool is input to physical design tools where controlled denormalization is done when performance tradeoffs require it.

2.5 Physical Design

In the physical data base design phase, we take the "vanilla" schema generated in the schema mapping phase together with a description of the most active transactions that will be running against the data base and design the physical structure and access paths to the data base that will optimize the performance of the specified transactions.

This task, although it sounds simple, is in practice very complex. The designer must be able to evaluate the storage/CPU usage tradeoffs as well as tradeoffs in the performance of different transactions of many alternative data base designs.

The design evaluation phase which follows physical design in this paper is not really separate from the physical design phase. We have separated the two phases, since a separate set of tools can be identified to assist in the two phases. The physical design tools, however, will have to invoke some of the design evaluation tools in order to be able to optimize data base performance.

2.5.1 Transaction Modelling

In order for the physical design tools to be able to assist with producing an optimized physical design, they must be provided with a description of the workload that is going to be performed against the data base. In most cases the 80/20 rule applies. This rule states that the 20% most active transactions account for 80% of the workload on the data base. Therefore, it is only necessary to describe a few of the most active transactions to the physical design tools to provide a fairly accurate description of the data base workload.

The transaction models are constructed from the processing-oriented requirements collected during the initial requirement specification phase. I have, however, deferred discussing the transaction modelling step until now because constructing these models requires thinking in terms of physical data base structures. Physical design considerations should be avoided until the physical design phase.

The table below shows a transaction model for a report that outputs for every project is overbudget a list of all employees on the project and their departments. This transaction model is similar to models proposed in [Ceri, Teorey].

OBJECT	ACCESS MODE	RECORDS PROCESSED	RECORDS SELECTED	OPERATION	LINK USED NEXT
PROJECT	SERIAL	100	15	GET	PJ-EMP
PJ-EMP	INDEX	10	10	GET	EMPLOYEE
EMPLOYEE	DIRECT	1	1	GET	EMP-DEPT
EMP-DEPT	INDEX	1	1	GET	

FREQ: 2/MONTH WEIGHT: 1 MODE: BATCH

For each transaction we specify (at bottom of table):

FREQ - Frequency of execution.

WEIGHT - Weight factor for frequency. Typically 1. As an example, we might assign a weight factor greater than 1 to a transaction that is executed on an infrequent basis, but requires fast response time.

MODE - Batch or Online. Batch transactions are given a somewhat lower weight factor than online transactions.

For each access operation in the transaction we specify:

OBJECT - The object to be accessed, either an entity or a relationship.

ACCESS MODE - Serial, direct (hashed), or indexed. This is only a hint to the design tools. When considering overall data base performance, an indexed access, for example, may have to be converted into a serial access if it is not cost-effective to provide the required index.

RECORDS PROCESSED - Estimate of the average number of records that will have to be processed.

RECORDS SELECTED - Estimate of the average number of records that will be selected from those processed. In the above example, we estimate that 15 of the 100 project records accessed will be overbudget, requiring access of the PJ-EMP, EMPLOYEE, and EMP-DEPT records.

OPERATION - Get, put, update, or delete.

LINK USED NEXT - Object to be accessed following current access.

The above transaction model is only a first attempt at developing a model that can be used for describing transactions to the design tools. There are still some problems with this model that need to be resolved. One major problem is that the model needs to be able to describe the data that the

transaction is going to operate on in as physical-structure independent a manner as possible in order to give the physical design tools the freedom to do their job.

Perhaps another approach to this problem, which deserves further investigation, is the use of a high-level non-procedural language such as a relational DBMS query language as the language to model transactions in.

2.5.2 Physical Design Assistance

Some of the physical design decisions that tools can assist with include :

- **Record segmentation.** Since some transactions access just a subset of a record, it might make sense to segment the record along such usage lines, improving the performance of the transactions that only need to access the record subset. Smaller record sizes reduce the number of I/Os performed (assuming that we access the records in the order they are clustered), since it is possible to block more records together. Also smaller record sizes result in less wasted storage.

- **Record joining.** This is the opposite of record segmentation and is done to improve the performance of high frequency transactions which access data from two or more records together. The tradeoff to consider is that the resulting denormalized design may lead to redundancy problems.

- **Data base partitioning.** For performance reasons, maintainability reasons, or because of limitations such as the 256 item limit in IMAGE, it might be desirable to partition a data base into two or more data bases. Design tools can suggest how to best partition the data base.

- **Index selection.** Design tools can be built to assist in selecting primary and secondary indices based on access paths needed to run the specified transactions. Tradeoffs have to be evaluated between improving the performance of the transaction accessing the data base through the index, worsening the performance of the transactions that have to update the index, and the increase in storage requirements from the added index.

- **Sort item selection.** Design tools can assist in evaluating tradeoffs involving the addition of a sort item to a data set.

- **Prime capacities.** It has been shown that hashing for master data sets in IMAGE improves if prime capacities are used. Prime capacities, slightly higher than required capacity, can be automatically assigned by a design tool.

- **Blocking factors.** Optimal blocking factors can be calculated for each data set, relieving the designer from the task.

There is a great deal of literature describing algorithms used to optimize physical data base design parameters. A good survey paper is [Chen2].

2.6 Design Evaluation

Physical data base design involves making a lot of design decisions. The effects of such decisions on data base performance are not always obvious. There is a need for VISICALC like tools that can

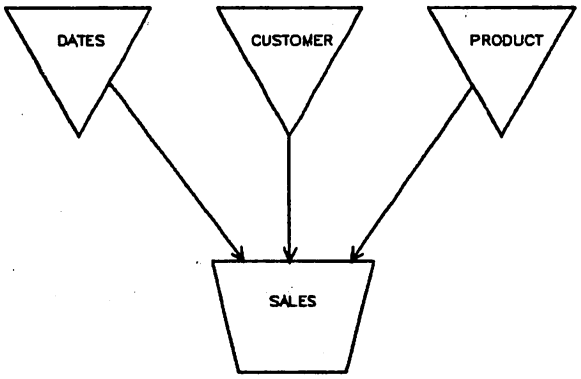
answer "what if" questions about data base performance. These tools need to be able to predict performance given a description of the data base structure and a description of the workload that is going to be run on the data base.

The tools that can be used to evaluate data base performance fall into two classes : analytical modelling tools and data base prototyping tools.

2.6.1 Analytical Modelling Tools

Making some simplifying assumptions, we can calculate the cost of running a transaction in terms of the average number of I/Os required to execute the transaction, ignoring all other costs. A design evaluation tool can be developed that will calculate the total I/O cost for a specified transaction workload. The following example illustrates the calculations that would be performed by such a tool in evaluating a very simple IMAGE design in which two transactions are executed.

The IMAGE design we are evaluating is a simplified version of the STORE data base in the IMAGE manual. This data base consists of two manual masters, one detail data set, and an optional auto master which provides a search path on PURCHASE-DATE. The goal of this evaluation is to determine whether the search path on PURCHASE-DATE should be provided or not.



There are two transactions performed against the STORE data base. The first transaction, SALES-POSTING, is an online transaction, executed 200 times per day. Each time the transaction is executed an average of 5 entries are posted to the SALES data set. A transaction model for this transaction is shown below :

SALES-POSTING

OBJECT	ACCESS MODE	RECORDS PROCESSED	RECORDS SELECTED	OPERATION	LINK USED NEXT
SALES		5		PUT	

FREQ: 200/DAY WEIGHT: 1 MODE: ONLINE

The second transaction, SALES-REPORT, is a batch transaction, executed once a day, which generates a report of sales information, customer information and product information for all sales with a specified PURCHASE-DATE. The transaction model for this transaction is shown below :

SALES-REPORT

OBJECT	ACCESS MODE	RECORDS PROCESSED	RECORDS SELECTED	OPERATION	LINK USED NEXT
SALES	SERIAL	20,000	1,000	GET	CUSTOMER
CUSTOMER	DIRECT	1	1	GET	PRODUCT
PRODUCT	DIRECT	1	1	GET	

FREQ: 1/DAY WEIGHT: 0.8 MODE: BATCH

The following assumptions have been made about the number of I/Os performed by each IMAGE intrinsic call:

	I/Os	
DBGET (master-calculated)	1.7	Assumes that we have to follow some synonym chains.
DBGET (detail-serial)	0.2	Assumes 5 records per block.
DBGET (detail-chain)	1	Does not assume that records are loaded in chain sequence.
DBPUT (detail)	2 + 4.7P	P is the number of search paths to the detail data set. 1 I/O - read a block with free space 1 I/O - write the block to disc 1.7P I/O - read the master record 1P I/O - update the master record 2P I/O - read and update previous end of chain

Let us now calculate the number of I/Os performed by the SALES-POSTING transaction when we do not have a search path on PURCHASE-DATE, and when we do have one.

The SALES-POSTING transaction calls the DBPUT intrinsic five times. Using the estimate :

$$I/Os = 2 + 4.7P$$

with P=2 (search paths on CUSTOMER and PRODUCT), each DBPUT intrinsic will perform 11.4 I/Os, and the SALES-POSTING transaction will perform a total of 57 I/Os.

with P=3 (search path on PURCHASE-DATE added), each DBPUT intrinsic will now perform 16.1 I/Os, and the SALES-POSTING transaction will perform a total of 80.5 I/Os.

Let us now do the same calculations for the SALES-REPORT transaction. Without a search path on PURCHASE-DATE, the transaction will call the DBGET intrinsic 20,000 times. For each of the 1,000 records we estimate will match the specified PURCHASE-DATE, a DBGET call to access the CUSTOMER data set and a DBGET call to access the PRODUCT data set will have to be made. The total number of I/Os performed by the transaction is as follows :

20,000 DBGETs (detail-serial) * 0.2	=	4,000
1,000 DBGETs (master-calculated) * 1.7	=	1,700
1,000 DBGETs (master-calculated) * 1.7	=	1,700
TOTAL	=	7,400

If we add a search item on PURCHASE-DATE, the 20,000 calls to DBGET in serial mode, are replaced by 1,000 calls to DBGET in chain mode, each call performing one I/O. Adding the search item on PURCHASE-DATE saves us 3,000 I/Os (4,000 - 1,000) over doing the serial DBGETs, and the total number of I/Os performed by the SALES-REPORT transaction is now only 4,400 I/Os.

The table below summarizes the remaining calculations that are necessary to determine which of our alternative designs will require the least number of total I/Os for the entire transaction workload.

TRANSACTION	WEIGHTED FREQ (tran/day)	DESIGN #1 I/Os PER TRANS.	DESIGN #1 TOTAL I/O	DESIGN #2 I/Os PER TRANS.	DESIGN #2 TOTAL I/O
SALES-POSTING	200.0	57.0	11,400.0	80.5	16,100.0
SALES-REPORT	0.8	7400.0	5,920.0	4400.0	3,520.0
			17,320.0		19,620.0

The table above shows that design #1, the one without the search path on PURCHASE-DATE will perform less total I/Os than design #2. The example we have just gone over, although overly simplified illustrates one type of calculation that a simple design evaluation tool can perform. Such a design evaluation tool would also calculate storage requirements for two or more alternative designs.

More sophisticated analytical modelling tools, based on queueing theory [Sevcik] are now starting to be developed to predict data base performance. Such tools would be used to more accurately predict data base performance taking into account the effects of different transactions running concurrently on the system, interaction with other activity on the system, data base locking strategies, disc caching, etc.

2.6.2 Data Base Prototyping Tools

Another approach that can be used to evaluate a data base design is to build prototypes of the most active transactions and actually measure the performance of these transactions using performance measurement tools. There are several tools that can assist with the different tasks involved in this approach to data base design evaluation. Such tools are not only useful in the initial data base design process, they are also useful in the ongoing data base performance-tuning and maintenance activities. These tools include :

- **Application prototyping tools.** Currently, there are tools available on the HP3000 such as the TRANSACT language, a part of the RAPID/3000 package, and some third party software packages that allow you to quickly build prototype applications. An effort needs to be made to tie together the transaction models used by the design tools to these prototyping tools.

- **Data base simulation tools.** An example of such a tool is IDEA (IMAGE Data base Evaluative Analyzer), a contributed library software package that simulates data base activity, and generates timing reports. Data base activity is specified through scripts which describe transactions against the data base. IDEA actually generates calls to the IMAGE intrinsics to obtain its performance estimates. It would be useful if scripts for an IDEA-like tool could be automatically generated from the transaction descriptions used as input to the design tools.

- **Performance monitoring tools.** Tools are needed to monitor data base activity. Such tools should monitor and generate reports detailing how often each data set was accessed, what type of access was performed, what search path was followed, how much contention for data base resources was experienced (e.g. how often was a data set being accessed locked by another transaction), and so on.

- **Test data generation tools.** A test data generator should automatically load a data base with random data which satisfies characteristics of the real data as specified by the designer. For example, a designer should be able to specify that a detail data set search item field be loaded with randomly distributed data in the range of 100,000 to 200,000 with an average chain length of five.

- **Data conversion tools.** These tools would simplify the task of loading a new data base with existing data, perhaps contained in a KSAM file, or in another IMAGE data base.

- **Data base restructuring tools.** An example is the ADAGER package which allows IMAGE data bases to be restructured online, without having to unload the data to do the restructuring.

3. DESIGN TOOLSET

In this section I would like to briefly discuss how a design toolset might integrate the different tools described in this paper. Such a toolset would provide a common interface to the different tools, would provide some common functionality across the different tools, and would allow the different tools to communicate with each other.

The central component of such a design toolset is going to be a data dictionary. The data dictionary is the interface through which the different tools communicate. The data dictionary is going to have to maintain different representations (E-R models, transaction models, network models, and relational models) of the meta-data describing the different models that are used by the design tools in the different phases of the design process.

The design toolset should provide (through the data dictionary) some common functionality supporting the design process. This functionality would include :

- **Documentation.** All the tools should allow all design decisions (from requirement specification through physical design) to be thoroughly documented.
- **Logging.** Changes to a design should be logged to permit review of the steps taken to arrive at a current design.
- **Version Management.** Make it easier for designers to explore alternative designs.
- **Archiving.** Ability to store material (e.g. local E-R models) generated during the design process which is not needed in a production dictionary on backup storage.

It is also important that each tool be designed to perform as specific and small a task as possible. This will allow tools to be used either independentlv or in combinations without interactions. For example, it should be possible to use the design evaluation tools without having to first use the modelling, schema mapping, and physical design tools. This allows the design evaluation tools to be used to evaluate the impact of design changes to an existing data base.

4. REFERENCES

Batini — Batini, C., Lenzerini, M., and Santucci, G., "A Computer-Aided Methodology for Conceptual Data Base Design". Information Systems, Vol. 7, No. 3, 1982, pp. 265-280.

Ceri — Ceri, S., Navathe S., and Wiederhold, G., "Distribution Design of Logical Database Schemas". IEEE Transactions on Software Engineering, Vol. SE-9, No. 4, July 1983, pp. 487-504.

Chen — Chen, P.P., "The Entity-Relationship Model - Toward a Unified View of Data". ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976, pp. 9-36.

Chen2 — Chen, P.P., and Yao, S.B., "Design and Performance Tools for Data Base Systems". Proceedings 3rd International Conference on Very Large Data Bases, 1977, pp. 3-15.

Hubbard — Hubbard, G.U., *Computer-Assisted Data Base Design*, Van Nostrand Reinhold Co., 1981

IEEE — IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977.

Kahn — Kahn, B.K., "A Method for Describing Information Required by the Data Base Design Process". Proceedings ACM-SIGMOD International Conference on Management of Data, 1976, pp. 53-64.

Kent — Kent, W., "A Simple Guide to Five Normal in Relational Database Theory". Communications of the ACM, Vol. 26, No. 2, February 1983, pp. 120-125.

Sevcik — Sevcik, K.C., "Data Base System Performance Prediction Using an Analytical Model". Proceedings 7th International Conference on Very Large Data Bases, 1981, pp. 182-198.

Teorey — Teorey, T.J. and Fry, J.P., "The Logical Record Access Approach to Database Design". ACM Computing Surveys, Vol. 12, No. 2, June 1980, pp. 179-211.

Yao — Yao, S.B., Navathe, S.B., and Weldon, J-L, "An Integrated Approach to Database Design". Proceedings 1978 NYU Symposium on Data Base Design (Lecture Series in Computer Science), no. 132, pp. 1-30, Springer-Verlag, 1982.

Abe Lederman was born in Montevideo, Uruguay and immigrated to the U.S. in 1968 at the age of 10. He received the B.S. and M.S. degrees in computer science from the Massachusetts Institute of Technology in 1981.

Upon graduation, Abe Lederman accepted a position as a member of technical staff in Hewlett-Packard's Information Network Division working on maintaining and enhancing the RAPID software package. He is now working in the data base section on data base tools.