Session 1658

Efficient Database Design

John H. Ristroph
University of Louisiana at Lafayette

Databases are a major foundation of the information age, but specifying their tables and fields can be a daunting challenge for designers. This paper presents a pedagogy that rapidly enables students to design small to medium sized systems. It recognizes that a database management system (DBMS) is a tool, and students must understand how the tool works before they can design something for its use. An example illustrates the concepts of entities and processes, tables, relations, indexes, and queries with a simple report generation problem. Then procedures are provided that make the design of databases more efficient by requiring fairly few changes as implementation progresses. Final steps include techniques for checking the quality of the design prior to implementation.

Process Overview

Table 1 provides an overview of the steps required to develop a database system. The first step defines the desired capabilities of the system. Steps 2 and 3 are pivotal steps that affect all of the following ones. They require creative, conceptual level thinking that challenges even excellent students. This paper presents methods that help all students learn how to perform these crucial steps in an efficient manner.

The first thing to recognize is that Table 1 initially is completely meaningless to students. They need to see how a database management system (DBMS) uses tables, relationships, and indexes before they can design an effective schema. One excellent way to do this is to go through a comparatively simple, but complete, example, such as the one presented in the appendix.

Table 1. Database Development

1. Identify information needs and define final products.
2. Determine entities and processes.
3. Identify tables, attributes, and relationships.
4. Specify important field properties, including primary keys and indexes.
5. Create tables with major field properties and declare relationships.
6. Enter a trial data base.
7. Check formats, add minor field properties, and test system.
8. Produce final products.
9. Document the system.

Schema Design

After illustrating the use of a DBMS, attention can shift to efficiently designing a schema. A commonly used introductory approach is to identify entities and processes, make a guess at the tables, and check the tables to make sure that they are in normal form. Repeatedly checking and changing the tables that have a poor initial design can be very laborious. This section presents procedures for rapidly teaching students how to obtain a good initial design and techniques for improving that design that go beyond checking normal forms.

Types of Tables

Tables need to be developed to describe entities and processes, but which tables? A good approach to specifying tables is first to be aware of the different types of tables, and then use the table types as a checklist to see which tables are needed for each entity or process. Tables can be categorized as descriptive, transaction, child, reference, summary, or historical, as well as tables serving multiple purposes. From the onset, it must be stressed that the ***only*** fields that should be in more than one table are identifiers, such as for employees (EmplD) or projects (ProjID).

1. *Descriptive or master* tables typically contain fairly static data typically dealing with entities, such as employee, product, or customer tables. A descriptive table might store identifiers such as ID numbers, full names, abbreviations or short names for use in reports, verbal descriptions, addresses, dimensions, weights, and so forth.
2. *Transaction* tables record *low-level* (non-aggregated) data about events or activities referred to as transactions. For example, an invoice table typically contains ID's of salespersons and customers, payment methods, dates, and other basic information about a sale. A work log might include dates, regular hours, and overtime hours (*not* just total hours) worked by each employee on each job in each skill class. Inventory tables might include data about shipments, receipts, transfers, and so forth. Storing low-level data allows it to be combined in many different ways for a variety of uses, whereas aggregated data is usually very difficult to recombine in different ways.
3. *Child* tables extend descriptive and transaction tables (referred to as *parent* tables) by allowing an indefinite number of entries. For example, a paper invoice contains basic data about a sale and then lists data for an indefinite number of sale items. The fields of an invoice parent table can be an invoice number used as a primary key, the customer ID, the salesperson ID, payment method, and the date. Each record of the child table contains the invoice number, an item ID, its quantity, and its price. The child table contains the primary key of the parent table, thereby allowing the two tables to be related. Concatenating the invoice number and the item number can provide a primary key for the child table. Notice that extended prices and the total amount of each sale are not stored. As a general rule, items that are easy to calculate are computed as needed, rather than stored, thereby reducing the volume of data.
4. *Reference* tables record basic data used to check entries in other tables (i.e., their data integrity) or to provide names for different codes. Examples include tables containing all allowable codes for types of: payments, inventory transactions, skill classes, project status, and so forth. Such tables typically contain the allowable codes, their corresponding names (possibly both short and long versions), and sometimes definitions.
5. *Summary* tables contain periodic summaries (e.g., monthly or yearly) or totals that are updated with transaction data. For example, a summary payroll table can contain year-to-date regular and overtime hours, total wages, deductions, leave, etc., but not the details in a transaction file. A summary inventory table can provide year-to-date issues and receipts, quantity on-hand, or quantity on-order. As a general rule, summary tables include items that are used frequently but that are too time consuming to calculate on demand. It is not unusual to put summary fields in a descriptive or master table instead of creating a separate table.
6. *Historical* tables store old, rarely accessed data from any of the foregoing types of tables. For example, large transaction tables such as work logs or inventory actions can become expensive to maintain due to the disk space and processing time that they require. Old transactions

typically are transferred to historical tables, particularly if the data in a summary table is adequate. Using the same table design for both the historical and transaction tables allows convenient transfers between tables.

Logic of Table Assignment

The following procedure produces fairly well-designed initial tables, and subsequent checks improve them. Each step ideally is applied to one entity or process at a time, going through all of steps before going onto the next entity or process. However, a certain amount of back-tracking or skipping ahead is inevitable.

1. Check to see which type of table is required. Entities typically need a descriptive table, and processes generally use transactions tables.
2. List the most important attributes for each entity or process, generally beginning with an identifier that is used as the primary key. It usually is easier to combine data than to separate it, so store each data element separately, such as last name, first name, and middle initial for a person or regular hours and overtime hours for a task.
3. Each time a code or identifier is entered, ask how its referential integrity can be checked. If it cannot be validated from a table that will be created for other entities and processes, then a special reference table is needed.
4. If a field or group of fields must be repeated an indefinite number of times, then use one or more child tables. For example, a customer descriptive table might require an indefinite number of entries for contacts' names, phone numbers, and so forth. A project descriptive table has an indefinite number of jobs, so each job becomes a record in a job child table with fields that include the project number and the job number. In turn, each job uses an unknown number of resource (skill or equipment) types, so the activity table becomes a parent table to a resource child table with fields that include the project number, job number, resource code, and the number of units of that resource required by the activity.
5. Check to see if any reference tables are needed to insure the referential integrity of codes that are not present in a non-reference table. For example, suppose that a Task table uses EmpID and SkillClass. The Employee table provides a validation check for EmpID, but a reference table might be required to contain all allowable entries of SkillClass.
6. If there is any summary data that is difficult to calculate, then create a summary table or possibly merge the summary fields into a descriptive table.
7. Examine all of the fields to insure that the only fields present in more than one table are codes, such as EmpID. If data associated with the code, such as a name, are needed, then they should be looked up in other tables.
8. Examine the need for historical tables. Even if they should be missed initially, they usually can be added to the schema later without much difficulty.
9. Update a list of any primary keys, relationships, or indexes that have been identified thus far. Whenever a field or concatenation of fields must be looked up in a table, it should be indexed and possibly be a primary key if it is unique.
10. Update a list of unanswered questions, issues that must be checked, or simply notes for future use. Trying to remember all of the details associated with a schema is a certain recipe for a headache.

This procedure usually provides a good starting point, but the tables should be checked as thoroughly as possible before implementing the system. The time spent in design is far less than the time spent making changes to a poorly designed database after implementation has begun. Provided below are some methods for checking the quality of the initial table specification.

Completeness

Data is processed into information products, such as reports. The first check insures that all necessary data is recorded. One very direct way to do this is to examine each information product to see if its required data is available. For example, examine each item in a report to see if its requisite data is in a table. If data is missing, see if it should be part of an existing table before creating any new ones. Consider marking each data element that is used, so that the potential need for any unmarked data can be questioned.

Normal Forms

Once the tables are complete, then examine each one to see if it is organized correctly by insuring that it is in *normal form*. Three steps for doing this are provided below, in the context of a database for a video rental store. Notice that the foregoring table assignment logic usually results in tables that are in normal form.

1NF. Remove any repeating field or group of repeating fields, and develop appropriate parent-child tables. Consider a rental transaction table with its fields in parentheses and the primary key underlined:

Rental(<u>RentID</u>, CustID, Date, VideoID1, CopyNum1, DueDate1,
VideoID2, CopyNum2, DueDate2, …)

The more common way of indicating repeating fields is with parentheses:

Rental(<u>RentID</u>, CustID, Date, Total, (VideoID, CopyNum, DueDate))

A child table will store the data more effectively, since DBMSs generally are programmed so that data is easier to manage when it is in a child table instead of repeating fields.

2NF. The entire primary key in a 1NF table should be needed to identify what is in each of the other fields. Suppose that the rental child table had been defined as

RentChild(<u>RentID</u>,<u>VideoID</u>, CopyNum, Title, DueDate, ReturnDate)

Notice that the Title field depends only on the VideoID component of the primary key. Placing the Title field in the Video descriptive file and just using the VideoID to reference it avoids wasting space and prevents problems that can occur with misspellings or slightly different names.

3NF. No non-primary key field in a 2NF table should be able to identify another one. For example, suppose that the Rental table contained the customer's last name:

Rental(<u>RentID</u>, CustID, LastName, Date)

Then CustID is a non-primary key field that uniquely identifies LastName. Another way of saying this is that LastName is *dependent* on CustID. This wastes space since a look-up can be performed. It also invites spelling and other problems.

Data Entry Procedure

At this stage of development, the system can be too abstract for a new designer. Writing the system's data entry procedure helps to define the system logic, review the table design, understand the relations between tables, and determine indexes and primary keys. Having a written data entry procedure before the system becomes operational is desirable, and this is a good time to do it. The order in which data is entered into tables should recognize the following:

1. Increase data accuracy by requiring *referential integrity* wherever possible. Tables that contain lists of all permissible codes must be updated before using those codes. For example, entering an EmpID or SkillClass in a Task table should be done after including those codes in an Employee descriptive table or a SkillClass reference table.
2. There is a *natural temporal sequence*. For example, data describing a project, such as its number, name, or component jobs, should be known before performing work on the project.

Continue updating the list of all *relationships* during the data entry process, as well as any *indexes* needed to perform look-ups. Also realize that indexes that do not allow duplicates also can improve data accuracy in some cases. For example, each employee number in an Employee table should be a unique entry, and an index on EmpNum without duplicates would insure uniqueness. Such indexes frequently indicate primary keys.

Sample Data Base and Initial Queries

This stage of development is on the borderline of the design and implementation phases of creating a database. At this time, there still might be some questions in the mind of the designer regarding the schema. It is important to focus on potential problem areas immediately, so that as little work as possible needs to be redone. Once the potential problems are identified, perhaps based on notes made earlier, then create the affected tables, specify properties and relationships, and enter just enough data to develop and test queries.

The data should be brief enough to hand check, but otherwise have the essential characteristics of a complete dataset. For example, if sums are to be developed, then more than one record that will be summed needs to be in the tables. Similarly, if there will be logical tests for special conditions, such as null values, then those conditions should be present in the data. It is a good idea to test one characteristic at a time to isolate the cause of any problems.

If the testing indicates problems with the current design, then the modifications must be made with care being taken to trace out all effects of changes. Once again, jotting notes removes the stress of trying to remember everything can improves performance.

Summary and Conclusions

Databases are a cornerstone of the information age, but the critical step of designing a schema is an art that requires creativity and insight in addition to technological knowledge. The objective of this paper is to make the design effort more efficient by developing a design process that does more than just recognize entities and check normal forms.

The design procedure has been found to be effective in practice and useful as a teaching tool. However, students, like everyone else, listen to wonderful lessons from someone else's experience, then promptly forget them. One way to provide reinforcement is to give students case problems and let them do the design.

Initially, the instructor serves as a moderator, recording and displaying (using a blackboard or a LCD) decisions students make on entities, processes, tables, fields, and relationships while gently guiding them through the design steps. Doing one or two simple systems like this should be followed by a system of moderate complexity that will be challenging.

At this time, the instructor should do less guiding. Let students ignore the design process, seek shortcuts, and have problems. Giving partial solutions in stages will keep the class moving while allowing the students to gain experience. Part of the learning experience is to understand that design is more than a quick homework assignment and to recognize the necessity of a structured design process. Students need to learn for themselves that occasional setbacks are a normal part of the design process. This can be frustrating and disheartening, but it is much better for it to happen in the classroom than on the job.

A good way to finish a course is to provide another case study *after* students have internalized the need for the design *process*. Randomly choose students to take turns guiding the design process, as well as recording and displaying the decisions of their classmates. Lend a helping hand only if necessary. The improvement in students' performance will be evident, and a source of satisfaction to everyone in the classroom.

Bibliography

1. Valacich, Joseph S., Joey F. George, and Jeffrey A. Hoffer, *Essentials of Systems Analysis and Design*, Prentice Hall, Upper Saddle River, NJ, 2001.

Biography

Dr. John H. Ristroph is a Professor of Engineering and Technology Management and a registered professional engineer in Louisiana. His B.S. and M.S. are from LSU, and his Ph.D. is from VPI&SU, all in industrial engineering. He has been active in the information systems area as an analyst, teacher, and researcher for over thirty years.

Appendix: Example Illustrating DBMS Functions

Employees of a company work on many projects for different clients, and it is necessary to track the number of hours that each employee works and to determine his or her gross pay. Figures 1 and 2 show two reports that that identify information needs and define the final products, step 1 in the development process.

Entities and Processes

Students must understand that examining reports might reveal what data is needed, but it does not help to organize it into the tables needed by a DBMS. This is done by determining the entities and processes of the business system. *Entities* are things about which data is needed. They can be persons, companies, products, parts and so forth. *Processes* are business activities that generate data, such as selling a product, performing a task, and so forth.

Sometimes it can be difficult to decide whether something is an entity or a process. For example, is a project an entity or a process? It is a thing about which data is needed, such as its due date, the amount of the contract, and so on. However, it also consists of tasks that create data, such as the hours worked by different employees at various pay rates. Whether something is classified as an entity or a process is not particularly important. In fact, the term entity is commonly used to describe both. What is important is to identify the things about which data must be

Gross Pay Report from 1/3/2000 to 1/4/2000
Date: 1/5/2000 Time: 10:05 a.m.

Employee	ID	Hours	Pay Rate	Gross Pay
LastA, FirstA A	0002	16.0	$20.00	$320.00
LastB, FirstB B	0001	15.0	$15.00	$225.00
			Total:	$545.00

Figure 1. Gross Pay Report

must be recorded, whatever they are called. Once the entities and processes are identified, then one or more tables can be set up to record the data needed for each one.

Identifying entities and processes is more of an art than a science, and it takes practice before general principles become meaningful and helpful. It is useful to ask students to help in identifying this system's entities and processes: The entity Employee performs the Task process on the entity Project. Then ask which descriptors or *attributes* of each entity or process are needed to produce the reports.

- Employee: the person's unique identifier, name, and pay rate
- Task: its unique identifier, employee, project, date, and hours
- Project: its unique identifier and name

Later exercises will focus on giving students a variety of small systems, and letting them practice identifying entities and processes, and then specifying tables to contain the attributes.

Employee Activity Report from 1/3/2000 to 1/4/2000
Date: 1/5/2000 Time: 10:00 a.m.

Employee	ID	Date	Project	Hours
LastA, FirstA A	0002	1/3	Project1 for ClientA	6.0
			Project2 for ClientA	2.0
		1/4	Project1 for ClientB	3.5
			Project2 for ClientB	4.5
			Total:	16.0
LastB, FirstB B	0001	1/3	Project1 for ClientA	3.0
			Project2 for ClientB	5.0
		1/4	Project2 for ClientA	7.0
			Total:	15.0
			Grand Total:	31.0

Figure 2. Employee Activity Report

Table 2. Employee Table

EmpID	PayRate	EmpLast	EmpFirst	EmpMid
0001	$15.00	LastB	FirstB	B
0002	$20.00	LastA	FirstA	A

Techniques for doing this are presented later, but for the moment a student still needs to know how a DBMS uses its tables.

Data Dictionaries, Tables, Records, and Fields

Table 2 through Table 4 show how tables store the attributes as columns known as *fields* on rows referred to as *records*. Each field is defined in a *data dictionary*, such as Table 5. Students need to examine the tables to see if they contain all of the data necessary to produce the reports. It must be stressed that the ***only*** fields repeated in more than one table are the unique identifiers known as *keys*. This saves space and minimizes problems caused by different names or spellings for the same entity. It also means that if a name changes, then only one entry needs to be changed.

Indexes, Primary Keys, and Field Properties

It is important to explain how a DBMS uses the tables to create the reports, as described in Table 6. Students quickly grasp the concept of looking up data in one table based on a key in another table and see that it can be time consuming, so this is a good time to introduce indexes.

Table 3. Project Table

ProjID	ProjName
0001	Project1 for ClientA
0002	Project2 for ClientA
0003	Project1 for ClientB
0004	Project2 for ClientB

It is helpful for a student to see an index. Table 7 shows an index for the Task table based on the employee ID. Note that the index is sorted by EmpID, and it points to records in the Task to facilitate look-ups. Without the index, it would be necessary to search every record in Task to match each EmpID needed in step 2 for both reports, just like trying to find a word in a dictionary without the words being stored in sorted order. There are very efficient procedures for searching a sorted list, such as an index, so indices are commonly used to facilitate look-ups. The performance improvement afforded by an index typically more than compensates for the resources used to store the index and update it.

Table 4. Task Table

TaskID	EmpID	ProjID	TaskDate	TaskHour
1	0002	0001	1/3/00	6
2	0002	0002	1/3/00	2
3	0001	0001	1/3/00	3
4	0001	0004	1/3/00	5
5	0002	0003	1/4/00	3.5
6	0002	0004	1/4/00	4.5
7	0001	0002	1/4/00	7
8	0002	0001	1/5/00	8

Table 5. Data Dictionary

Name	Description
EmpID	Employee ID: Unique number for each employee
EmpFirst	Employee First Name
EmpLast	Employee Last Name
EmpMid	Employee Middle Initial
PayRate	Pay Rate: Dollars per hour earned by an employee
ProjID	Project ID: Unique number for each project
ProjName	Project Name
TaskID	Task ID: Unique number for each task. A task is the work performed by one employee on one project during one day.
TaskDate	Task Date: Date on which a task is performed.
TaskHour	Task Hours: Hours worked on a task.

Table 6. Basic Logic for Reports

Employee Activity Report

1. Retrieve EmpID, EmpLast, EmpFirst, and EmpMid from the Employee table.
2. For each EmpID in Employee, search the Task table to find each entry with the same EmpID and record its ProjectID, TaskDate, and TaskHour.
3. Retrieve each ProjName by matching the ProjID in the Task table with the ProjID in the Project table.

Gross Pay Report

1. Retrieve EmpID, EmpLast, EmpFirst, EmpMid, and PayRate from the Employee table.
2. For each EmpID in Employee, search Task and find all entries with the same EmpID and sum the values of TaskHour.
3. Multiply the sum of TaskHour by PayRate.

Field Properties

Usually students are not familiar with the need to specify properties of data structures. It must be noted that a DBMS is a very general, automated system that requires a high degree of structure. Table 8 shows some of the major field properties used by Microsoft's Access 2000™.

Table 7. Index

EmpID	Pointer
0001	3
0001	4
0001	7
0002	1
0002	2
0002	5
0002	6
0002	8

Relationships

Using the same name for fields common to more than one table helps humans understand a system, but a DBMS needs an explicit declaration of these relationships, such as those shown in Figure 3. This is a good time to explain how relationships can enforce referential integrity, such as not entering a value of EmpID in Task unless that value also is in Employee.

Table 8. Tables and Fields

Table	Field	Primary Key	Data Type	Field Size	Req'd	Index
Employee	EmpID	Yes	Text	4	Yes	
	PayRate		Currency	8	Yes	
	EmpLast		Text	12	Yes	
	EmpFirst		Text	10	No	
	EmpMid		Text	1	No	
Project	ProjID	Yes	Text	4	Yes	
	ProjName		Text	20	No	
Task	TaskID	Yes	AutoNumber	Long Int.	Yes	
	EmpID		Text	4	Yes	Allow dup.
	ProjID		Text	4	Yes	
	TaskDate		Date	8	Yes	
	TaskHour		Number	Single	Yes	

Queries

Queries use relationships to create temporary tables called *dynaset*. The dynasets can organize data in a manner suitable for creating the reports. A good way to illustrate queries is to create a dynaset containing all of the data in the Employee table, followed by all of the related data in the Task and Project tables. This clearly shows how the new records are formed from the ones in the tables. The difference between inner and outer joins can be shown by adding a new employee to the Empoyee table, and executing the query with both types of joins. Developing queries that will be used to produce the reports shown in Figures 1 and 2 makes a transition to the next step.

Report Generation

Generating reports serves two important instructional purposes. The first is to close the design loop and show how tables lead to queries that are transformed into reports. The second is to illustrate the basics of how a report generator works. Both of these purposes are served by using quick, default formats instead of more edited versions such as Figures 1 and 2. Formatting reports can be tedious and time consuming enough to distract students from the primary focus of schema design.

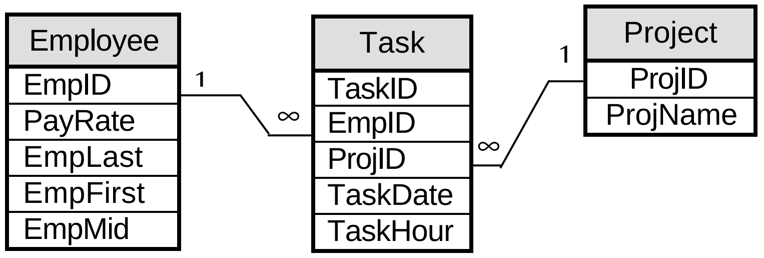

Figure 3. Relationships