To address the specialized needs of data warehousing applications (often called "Online Analytical Processing" or OLAP), specialized databases began to appear. These databases were optimized for OLAP workloads in several different ways. Their performance was tuned for complex, read-only query access. They supported advanced statistical and other data functions, such as built-in time-series processing. They supported precalculation of database statistical data, so that retrieving averages and totals could be dramatically faster. Some of these specialized databases did not use SQL, but many did (leading to the companion term "ROLAP," for Relational Online Analytic Processing). As with so many segments of the database market, SQL's advantages as a standard proved to be a powerful force. Data warehousing has become a one-billion-dollar plus segment of the database market, and SQL-based databases are firmly entrenched as the mainstream technology for building data warehouses.

Summary

This chapter described the development of SQL and its role as a standard language for relational database management:

- SQL was originally developed by IBM researchers, and IBM's strong support of SQL is a key reason for its success.
- There are official ANSI/ISO SQL standards and several other SQL standards, each slightly different from the ANSI/ISO standards.
- Despite the existence of standards, there are many small variations among commercial SQL dialects; no two SQLs are exactly the same.
- SQL has become the standard database management language across a broad range of computer systems and applications areas, including mainframes, workstations, personal computers, OLTP systems, client/server systems, data warehousing, and the Internet.

Chapter 4: Relational Databases

Overview

Database management systems organize and structure data so that it can be retrieved and manipulated by users and application programs. The data structures and access techniques provided by a particular DBMS are called its *data model*. A data model determines both the "personality" of a DBMS and the applications for which it is particularly well suited.

SQL is a database language for relational databases that uses the *relational data model*. What exactly is a relational database? How is data stored in a relational database? How do relational databases compare to earlier technologies, such as hierarchical and network databases? What are the advantages and disadvantages of the relational model? This chapter describes the relational data model supported by SQL and compares it to earlier strategies for database organization.

Early Data Models

As database management became popular during the 1970s and 1980s, a handful of popular data models emerged. Each of these early data models had advantages and disadvantages that played key roles in the development of the relational data model. In many ways the relational data model represented an attempt to streamline and simplify the earlier data models. In order to understand the role and contribution of SQL and the relational model, it is useful to briefly examine some data models that preceded the development of SQL.

File Management Systems

Before the introduction of database management systems, all data permanently stored on a computer system, such as payroll and accounting records, was stored in individual files. A *file management system*, usually provided by the computer manufacturer as part of the computer's operating system, kept track of the names and locations of the files. The file management system basically had no data model; it knew nothing about the internal contents of files. To the file management system, a file containing a word processing document and a file containing payroll data appeared the same.

Knowledge about the contents of a file—what data it contained and how the data was organized—was embedded in the application programs that used the file, as shown in Figure 4-1. In this payroll application, each of the COBOL programs that processed the employee master file contained a *file description* (FD) that described the layout of the data in the file. If the structure of the data changed—for example, if an additional item of data was to be stored for each employee—every program that accessed the file had to be modified. As the number of files and programs grew over time, more and more of a data processing department's effort went into maintaining existing applications rather than developing new ones.

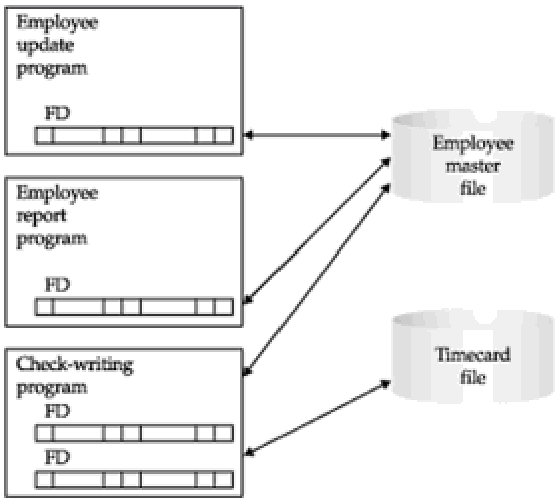


Figure 4-1: A payroll application using a file management system

The problems of maintaining large file-based systems led in the late 1960s to the development of database management systems. The idea behind these systems was simple: take the definition of a file's content and structure out of the individual programs, and store it, together with the data, in a database. Using the information in the database, the DBMS that controlled it could take a much more active role in managing both the data and changes to the database structure.

Hierarchical Databases

One of the most important applications for the earliest database management systems was production planning for manufacturing companies. If an automobile manufacturer decided to produce 10,000 units of one car model and 5,000 units of another model, it needed to know how many parts to order from its suppliers. To answer the question, the product (a car) had to be decomposed into assemblies (engine, body, chassis), which were decomposed into subassemblies (valves, cylinders, spark plugs), and then into sub-subassemblies, and so on. Handling this list of parts, known as a *bill of materials*, was a job tailor-made for computers.

The bill of materials for a product has a natural hierarchical structure. To store this data, the *hierarchical* data model, illustrated in Figure 4-2, was developed. In this model, each *record* in the database represented a specific part. The records had *parent/child relationships*, linking each part to its subpart, and so on.

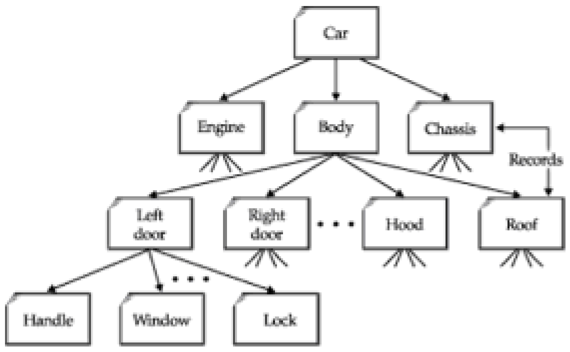


Figure 4-2: A hierarchical bill-of-materials databse

To access the data in the database, a program could:

- find a particular part by number (such as the left door),
- move "down" to the first child (the door handle),
- move "up" to its parent (the body), or
- move "sideways" to the next child (the right door).

Retrieving the data in a hierarchical database thus required *navigating* through the records, moving up, down, and sideways one record at a time.

One of the most popular hierarchical database management systems was IBM's Information Management System (IMS), first introduced in 1968. The advantages of IMS and its hierarchical model follow.

- *Simple structure.* The organization of an IMS database was easy to understand. The database hierarchy paralleled that of a company organization chart or a family tree.
- *Parent/child organization.* An IMS database was excellent for representing parent/child relationships, such as "A is a part of B" or "A is owned by B."
- *Performance.* IMS stored parent/child relationships as physical pointers from one data record to another, so that movement through the database was rapid. Because the structure was simple, IMS could place parent and child records close to one another on the disk, minimizing disk input/output.

IMS is still a very widely used DBMS on IBM mainframes. Its raw performance makes it the database of choice in high-volume transaction processing applications such as processing bank ATM transactions, verifying credit card numbers, and tracking the delivery of overnight packages. Although relational database performance has improved dramatically over the last decade, the performance requirements of applications such as these have also increased, insuring a continued role for IMS.

Network Databases

The simple structure of a hierarchical database became a disadvantage when the data had a more complex structure. In an order-processing database, for example, a single order might participate in three *different* parent/child relationships, linking the order to the customer who placed it, the salesperson who took it, and the product ordered, as shown in Figure 4-3. The structure of this type of data simply didn't fit the strict hierarchy of IMS.

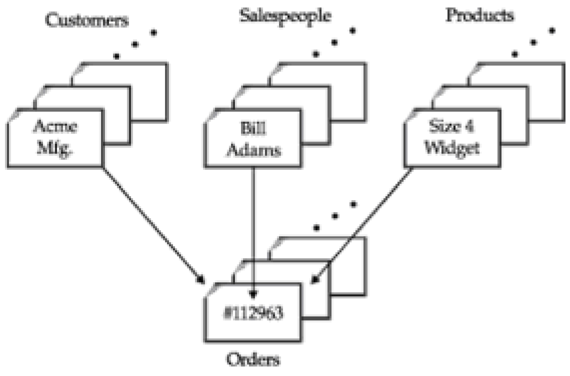


Figure 4-3: Multiple parent/child relationships

To deal with applications such as order processing, a new *network* data model was developed. The network data model extended the hierarchical model by allowing a record to participate in multiple parent/child relationships, as shown in Figure 4-4. These relationships were known as *sets* in the network model. In 1971 the Conference on Data Systems Languages published an official standard for network databases, which became known as the CODASYL model. IBM never developed a network DBMS of its own, choosing instead to extend IMS over the years. But during the 1970s independent software companies rushed to embrace the network model, creating products such as Cullinet's IDMS, Cincom's Total, and the Adabas DBMS that became very popular.

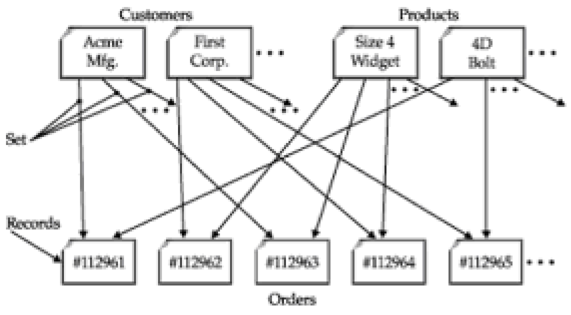


Figure 4-4: A network (CODASYL) order-processing database

For a programmer, accessing a network database was very similar to accessing a hierarchical database. An application program could:

- find a specific parent record by key (such as a customer number),
- move down to the first child in a particular set (the first order placed by this customer),
- move sideways from one child to the next in the set (the next order placed by the same customer), or
- move up from a child to its parent in another set (the salesperson who took the order).

Once again the programmer had to navigate the database record-by-record, this time specifying which relationship to navigate as well as the direction.

Network databases had several advantages:

- *Flexibility.* Multiple parent/child relationships allowed a network database to represent data that did not have a simple hierarchical structure.
- *Standardization.* The CODASYL standard boosted the popularity of the network model, and minicomputer vendors such as Digital Equipment Corporation and Data General implemented network databases.
- *Performance.* Despite their greater complexity, network databases boasted performance approaching that of hierarchical databases. Sets were represented by pointers to physical data records, and on some systems, the database administrator could specify data clustering based on a set relationship.

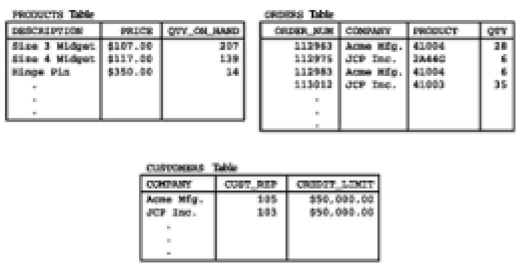
Network databases had their disadvantages, too. Like hierarchical databases, they were very rigid. The set relationships and the structure of the records had to be specified in advance. Changing the database structure typically required rebuilding the entire database.

Both hierarchical and network databases were tools for programmers. To answer a question such as "What is the most popular product ordered by Acme Manufacturing?" a programmer had to write a program that navigated its way through the database. The backlog of requests for custom reports often stretched to weeks or months, and by the time the program was written, the information it delivered was often worthless.

The disadvantages of the hierarchical and network models led to intense interest in the new *relational* data model when it was first described by Dr. Codd in 1970. At first the relational model was little more than an academic curiosity. Network databases continued to be important throughout the 1970s and early 1980s, particularly on the minicomputer systems that were surging in popularity. However, by the mid-1980s the relational model was clearly emerging as the "new wave" in data management. By the early 1990s, network databases were clearly declining in importance, and today they no longer play a major role in the database market.

The Relational Data Model

The relational model proposed by Dr. Codd was an attempt to simplify database structure. It eliminated the explicit parent/child structures from the database, and instead represented all data in the database as simple row/column tables of data values. Figure 4-5 shows a relational version of the network order-processing database in Figure 4-4.



PRODUCTS Table

DESCRIPTION	PRICE	QTY_ON_HAND
Size 3 Widget	$107.00	207
Size 4 Widget	$117.00	139
Hinge Pin	$350.00	14
.		
.		
.		

ORDERS Table

ORDER_NUM	COMPANY	PRODUCT	QTY
112963	Acme Mfg.	41004	28
112975	JCP Inc.	2A44G	6
112983	Acme Mfg.	41004	6
113012	JCP Inc.	41003	35
.			
.			
.			

CUSTOMERS Table

COMPANY	CUST_REP	CREDIT_LIMIT
Acme Mfg.	105	$50,000.00
JCP Inc.	103	$50,000.00
.		
.		
.		

Figure 4-5: A relational order-processing database

Unfortunately, the practical definition of "What is a relational database?" became much less clear-cut than the precise, mathematical definition in Codd's 1970 paper. Early relational database management systems failed to implement some key parts of Codd's model, which are only now finding their way into commercial products. As the relational concept grew in popularity, many databases that were called "relational" in fact were not.

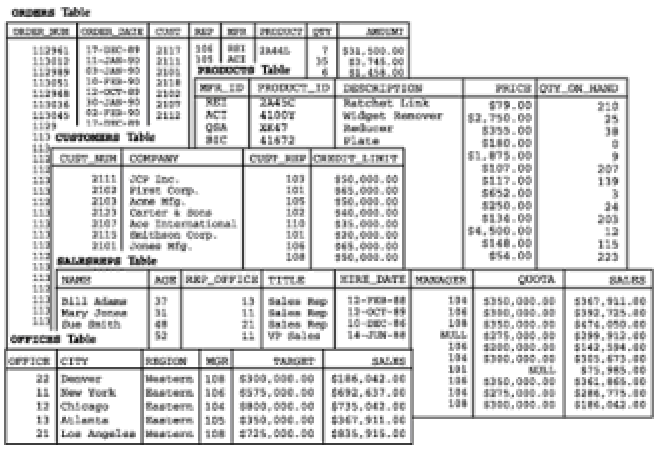
In response to the corruption of the term "relational," Dr. Codd wrote an article in 1985 setting forth 12 rules to be followed by any database that called itself "truly relational." Codd's 12 rules have since been accepted as *the* definition of a truly relational DBMS. However, it's easier to start with a more informal definition:

> A relational database is a database where all data visible to the user is organized strictly as tables of data values, and where all database operations work on these tables.

The definition is intended specifically to rule out structures such as the embedded pointers of a hierarchical or network database. A relational DBMS can represent parent/child relationships, but they are represented strictly by the data values contained in the database tables.

The Sample Database

Figure 4-6 shows a small relational database for an order-processing application. This sample database is used throughout this book and provides the basis for most of the examples. Appendix A contains a complete description of the database structure and its contents.



ORDERS Table

ORDER_NUM	ORDER_DATE	CUST	REP	MFR	PRODUCT	QTY	AMOUNT
112961	17-DEC-89	2117	106	REI	2A44L	7	$31,500.00
113012	11-JAN-90	2111	105	ACI		35	$3,745.00
112989	03-JAN-90	2101				6	$1,458.00
113051	10-FEB-90	2118					
112968	12-OCT-89	2102					
113036	30-JAN-90	2107					
113045	02-FEB-90	2112					
1129	17-DEC-89						

PRODUCTS Table

MFR_ID	PRODUCT_ID	DESCRIPTION	PRICE	QTY_ON_HAND
REI	2A45C	Ratchet Link	$79.00	210
ACI	4100Y	Widget Remover	$2,750.00	25
QSA	XK47	Reducer	$355.00	38
BIC	41672	Plate	$180.00	0
			$1,875.00	9
			$107.00	207
			$117.00	139
			$652.00	3
			$250.00	24
			$134.00	203
			$4,500.00	12
			$148.00	115
			$54.00	223

CUSTOMERS Table

CUST_NUM	COMPANY	CUST_REP	CREDIT_LIMIT
2111	JCP Inc.	103	$50,000.00
2102	First Corp.	101	$65,000.00
2103	Acme Mfg.	105	$50,000.00
2123	Carter & Sons	102	$40,000.00
2107	Ace International	110	$35,000.00
2115	Smithson Corp.	101	$20,000.00
2101	Jones Mfg.	106	$65,000.00
		108	$50,000.00

SALESREPS Table

NAME	AGE	REP_OFFICE	TITLE	HIRE_DATE	MANAGER	QUOTA	SALES
Bill Adams	37	13	Sales Rep	12-FEB-88	104	$350,000.00	$367,911.00
Mary Jones	31	11	Sales Rep	12-OCT-89	106	$300,000.00	$392,725.00
Sue Smith	48	21	Sales Rep	10-DEC-86	108	$350,000.00	$474,050.00
	52	11	VP Sales	14-JUN-88	NULL	$275,000.00	$299,912.00
					106	$200,000.00	$142,594.00
					104	$300,000.00	$305,673.00
					101	NULL	$75,985.00
					106	$350,000.00	$361,865.00
					104	$275,000.00	$286,775.00
					108	$300,000.00	$186,042.00

OFFICES Table

OFFICE	CITY	REGION	MGR	TARGET	SALES
22	Denver	Western	108	$300,000.00	$186,042.00
11	New York	Eastern	106	$575,000.00	$692,637.00
12	Chicago	Eastern	104	$800,000.00	$735,042.00
13	Atlanta	Eastern	105	$350,000.00	$367,911.00
21	Los Angeles	Western	108	$725,000.00	$835,915.00

Figure 4-6: The sample database

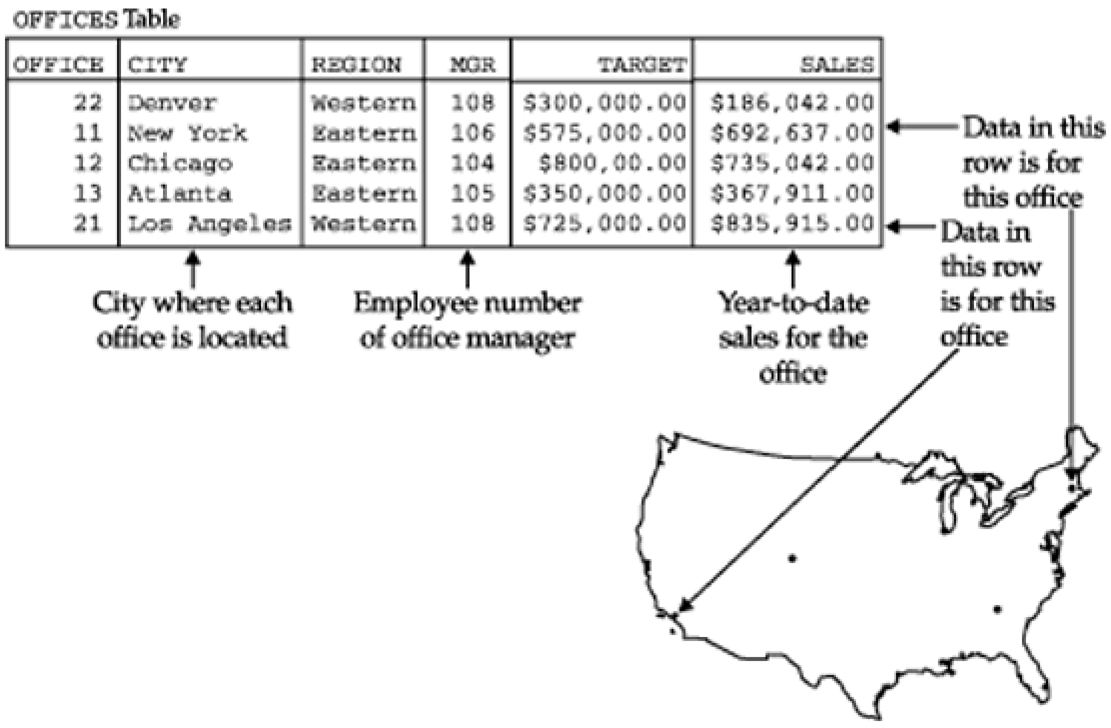
The sample database contains five tables. Each table stores information about one particular *kind* of entity:

- The `CUSTOMERS` table stores data about each customer, such as the company name, credit limit, and the salesperson who calls on the customer.
- The `SALESREPS` table stores the employee number, name, age, year-to-date sales, and other data about each salesperson.
- The `OFFICES` table stores data about each of the five sales offices, including the city where the office is located, the sales region to which it belongs, and so on.
- The `ORDERS` table keeps track of every order placed by a customer, identifying the salesperson who took the order, the product ordered, the quantity and amount of the order, and so on. For simplicity, each order is for only one product.
- The `PRODUCTS` table stores data about each product available for sale, such as the manufacturer, product number, description, and price.

Tables

The organizing principle in a relational database is the *table,* a rectangular, row/column arrangement of data values. Each table in a database has a unique *table name* that identifies its contents. (Actually, each user can choose their own table names without worrying about the names chosen by other users, as explained in Chapter 5.)

The row/column structure of a table is shown more clearly in Figure 4-7, which is an enlarged view of the `OFFICES` table. Each horizontal *row* of the `OFFICES` table represents a single physical entity—a single sales office. Together the five rows of the table represent all five of the company's sales offices. All of the data in a particular row of the table applies to the office represented by that row.



OFFICES Table

OFFICE	CITY	REGION	MGR	TARGET	SALES
22	Denver	Western	108	$300,000.00	$186,042.00
11	New York	Eastern	106	$575,000.00	$692,637.00
12	Chicago	Eastern	104	$800,00.00	$735,042.00
13	Atlanta	Eastern	105	$350,000.00	$367,911.00
21	Los Angeles	Western	108	$725,000.00	$835,915.00

City where each office is located
Employee number of office manager
Year-to-date sales for the office
Data in this row is for this office
Data in this row is for this office

Figure 4-7:. The row/column structure of a relational table

Each vertical *column* of the `OFFICES` table represents one item of data that is stored in the database for each office. For example, the `CITY` column holds the location of each office. The `SALES` column contains each office's year-to-date sales total. The `MGR` column shows the employee number of the person who manages the office.

Each row of a table contains exactly one data value in each column. In the row representing the New York office, for example, the `CITY` column contains the value "New York." The `SALES` column contains the value "$692,637.00," which is the year-to-date sales total for the New York office.

For each column of a table, all of the data values in that column hold the same type of data. For example, all of the `CITY` column values are words, all of the `SALES` values are money amounts, and all of the `MGR` values are integers (representing employee numbers). The set of data values that a column can contain is called the *domain* of the column. The domain of the `CITY` column is the set of all names of cities. The domain of the `SALES` column is any money amount. The domain of the `REGION` column is just two data values, "Eastern" and "Western," because those are the only two sales regions the company has!

Each column in a table has a *column name,* which is usually written as a heading at the top of the column. The columns of a table must all have different names, but there is no prohibition against two columns in two different tables having identical names. In fact, frequently used column names, such as NAME, ADDRESS, QTY, PRICE, and SALES, are often found in many different tables of a production database.

The columns of a table have a left-to-right order, which is defined when the table is first created. A table always has at least one column. The ANSI/ISO SQL standard does not specify a maximum number of columns in a table, but almost all commercial SQL products do impose a limit. Usually the limit is 255 columns per table or more.

Unlike the columns, the rows in a table do *not* have any particular order. In fact, if you use two consecutive database queries to display the contents of a table, there is no guarantee that the rows will be listed in the same order twice. Of course you can ask SQL to sort the rows before displaying them, but the sorted order has nothing to do with the actual arrangement of the rows within the table.

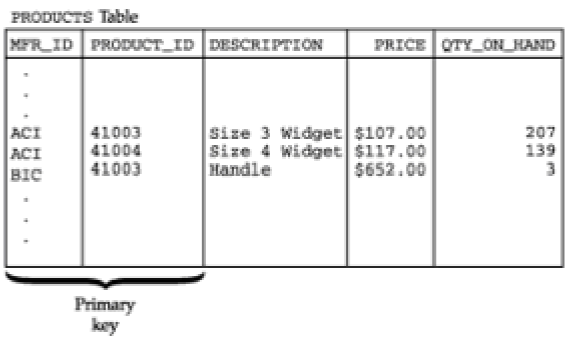
A table can have any number of rows. A table of zero rows is perfectly legal and is called an *empty* table (for obvious reasons). An empty table still has a structure, imposed by its columns; it simply contains no data. The ANSI/ISO standard does not limit the number of rows in a table, and many SQL products will allow a table to grow until it exhausts the available disk space on the computer. Other SQL products impose a maximum limit, but it is always a very generous one—two billion rows or more is common.

Primary Keys

Because the rows of a relational table are unordered, you cannot select a specific row by its position in the table. There is no "first row," "last row," or "thirteenth row" of a table. How then can you specify a particular row, such as the row for the Denver sales office?

In a well-designed relational database every table has some column or combination of columns whose values uniquely identify each row in the table. This column (or columns) is called the *primary key* of the table. Look once again at the OFFICES table in Figure 4-7. At first glance, either the OFFICE column or the CITY column could serve as a primary key for the table. But if the company expands and opens two sales offices in the same city, the CITY column could no longer serve as the primary key. In practice, "ID numbers," such as an office number (OFFICE in the OFFICES table), an employee number (EMPL_NUM in the SALESREPS table), and customer numbers (CUST_NUM in the CUSTOMERS table), are often chosen as primary keys. In the case of the ORDERS table there is no choice—the only thing that uniquely identifies an order is its order number (ORDER_NUM).

The PRODUCTS table, part of which is shown in Figure 4-8, is an example of a table where the primary key must be a *combination* of columns. The MFR_ID column identifies the manufacturer of each product in the table, and the PRODUCT_ID column specifies the manufacturer's product number. The PRODUCT_ID column might make a good primary key, but there's nothing to prevent two different manufacturers from using the same number for their products. Therefore, a combination of the MFR_ID and PRODUCT_ID columns must be used as the primary key of the PRODUCTS table. Every product in the table is guaranteed to have a unique combination of data values in these two columns.



PRODUCTS Table

MFR_ID	PRODUCT_ID	DESCRIPTION	PRICE	QTY_ON_HAND
.				
.				
.				
ACI	41003	Size 3 Widget	$107.00	207
ACI	41004	Size 4 Widget	$117.00	139
BIC	41003	Handle	$652.00	3
.				
.				
.				

Primary key

Figure 4-8: A table with a composite primary key

The primary key has a different unique value for each row in a table, so no two rows of a table with a primary key are exact duplicates of one another. A table where every row is different from all other rows is called a *relation* in mathematical terms. The name "relational database" comes from this term, because relations (tables with distinct rows) are at the heart of a relational database.

Although primary keys are an essential part of the relational data model, early relational database management systems (System/R, DB2, Oracle, and others) did not provide explicit support for primary keys. Database designers usually ensured that all of the tables in their databases had a primary key, but the DBMS itself did not provide a way to identify the primary key of a table. DB2 Version 2, introduced in April 1988, was the first of IBM's commercial SQL products to support primary keys. The ANSI/ISO standard was subsequently expanded to include a definition of primary key support.

Relationships

One of the major differences between the relational model and earlier data models is that explicit pointers, such as the parent/child relationships of a hierarchical database, are banned from relational databases. Yet obviously these relationships exist in a relational database. For example, in the sample database, each of the salespeople is assigned to a particular sales office, so there is an obvious relationship between the rows of the `OFFICES` table and the rows of the `SALESREPS` table. Doesn't the relational model "lose information" by banning these relationships from the database?

As shown in Figure 4-9, the answer to the question is "no." The figure shows a close-up of a few rows of the `OFFICES` and `SALESREPS` tables. Note that the `REP_OFFICE` column of the `SALESREPS` table contains the office number of the sales office where each salesperson works. The domain of this column (the set of legal values it may contain) is *precisely* the set of office numbers found in the `OFFICE` column of the `OFFICES` table. In fact, you can find the sales office where Mary Jones works by finding the value in Mary's `REP_OFFICE` column (11) and finding the row of the `OFFICES` table that has a matching value in the `OFFICE` column (in the row for the New York office). Similarly, to find all the salespeople who work in New York, you could note the `OFFICE` value for the New York row (11) and then scan down the `REP_OFFICE` column of the `SALESREPS` table looking for matching values (in the rows for Mary Jones and Sam Clark).

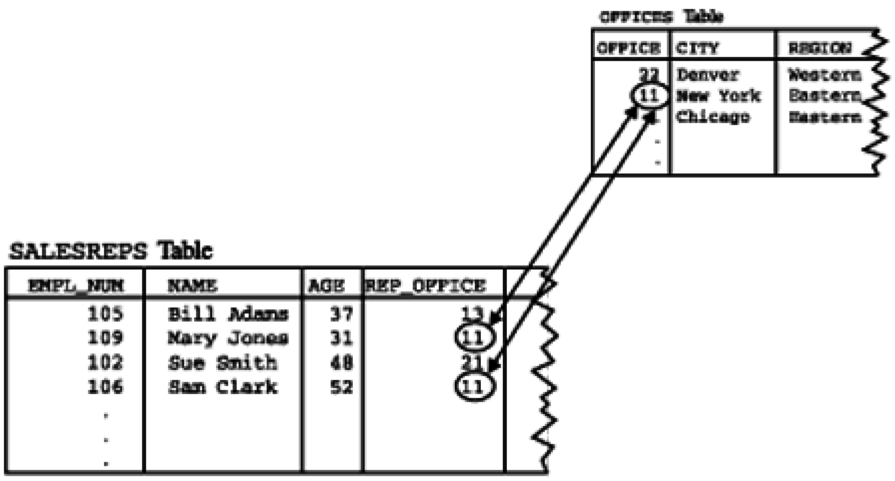


Figure 4-9: A parent/child relationship in a relational database

The parent/child relationship between a sales office and the people who work there isn't lost by the relational model, it's just not represented by an explicit pointer stored in the database. Instead, the relationship is represented by *common data values* stored in the two tables. All relationships in a relational database are represented this way. One of the main goals of the SQL language is to let you retrieve related data from the database by manipulating these relationships in a simple, straightforward way.

Foreign Keys

A column in one table whose value matches the primary key in some other table is called a *foreign key.* In Figure 4-9 the `REP_OFFICE` column is a foreign key for the `OFFICES` table. Although `REP_OFFICE` is a column in the `SALESREPS` table, the values that this column contains are office numbers. They match values in the `OFFICE` column, which is the primary key for the `OFFICES` table. Together, a primary key and a foreign key create a parent/child relationship between the tables that contain them, just like the parent/child relationships in a hierarchical database.

Just as a combination of columns can serve as the primary key of a table, a foreign key can also be a combination of columns. In fact, the foreign key will *always* be a compound (multi-column) key when it references a table with a compound primary key. Obviously, the number of columns and the data types of the columns in the foreign key and the primary key must be identical to one another.

A table can contain more than one foreign key if it is related to more than one other table. Figure 4-10 shows the three foreign keys in the `ORDERS` table of the sample database:

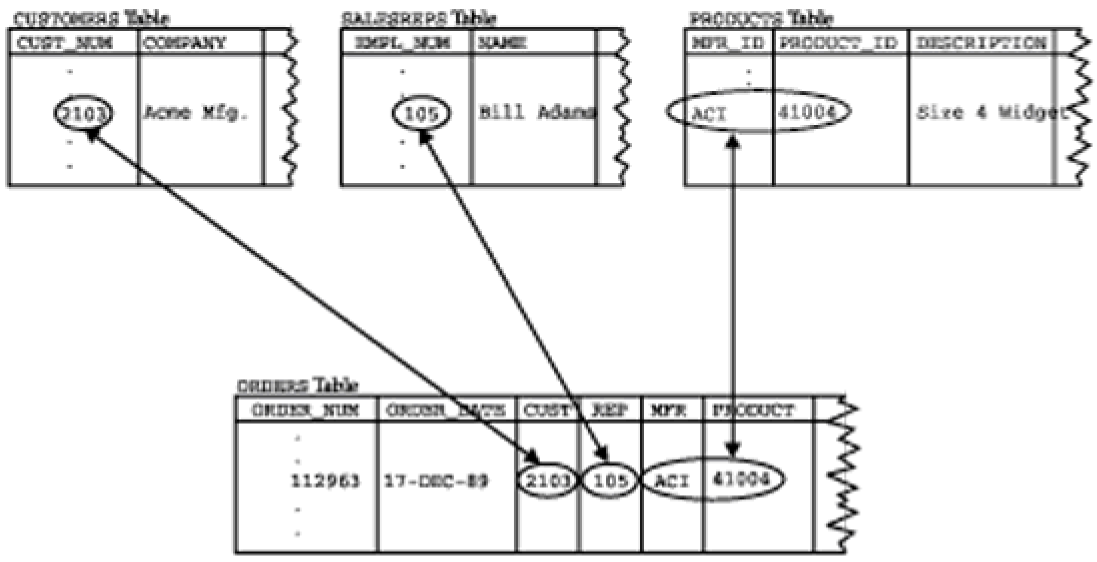


Figure 4-10: Multiple parent/child relationships in a relational database

- The `CUST` column is a foreign key for the `CUSTOMERS` table, relating each order to the customer who placed it.
- The `REP` column is a foreign key for the `SALESREPS` table, relating each order to the salesperson who took it.
- The `MFR` and `PRODUCT` columns together are a composite foreign key for the `PRODUCTS` table, relating each order to the product being ordered.

The multiple parent/child relationships created by the three foreign keys in the `ORDERS` table may seem familiar to you, and they should. They are precisely the same relationships as those in the network database of Figure 4-4. As the example shows, the relational data model has all of the power of the network model to express complex relationships.

Foreign keys are a fundamental part of the relational model because they create relationships among tables in the database. Unfortunately, as with primary keys, foreign key support was missing from early relational database management systems. They were added to DB2 Version 2, have since been added to the ANSI/ISO standard, and now appear in many commercial products.

Codd's Twelve Rules *

In his 1985 *Computerworld* article, Ted Codd presented 12 rules that a database must obey if it is to be considered truly relational. Codd's 12 rules, shown in the following list, have since become a semi-official definition of a relational database. The rules come out of Codd's theoretical work on the relational model and actually represent more of an ideal goal than a definition of a relational database.

1. *The information rule.* All information in a relational database is represented explicitly at the logical level and in exactly one way—by values in tables.
2. *Guaranteed access rule.* Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.
3. *Systematic treatment of null values*. Null values (distinct from an empty character string or a string of blank characters and distinct from zero or any other number) are

supported in a fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of the data type.

4. *Dynamic online catalog based on the relational model*. The database description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as they apply to the regular data.

5. *Comprehensive data sublanguage rule*. A relational system may support several languages and various modes of terminal use (for example, the fill-in-the-blanks mode). However, there must be at least one language whose statements are expressible, per some well-defined syntax, as character strings, and that is comprehensive in supporting all of the following items:

   - Data definition
   - View definition
   - Data manipulation (interactive and by program)
   - Integrity constraints
   - Authorization
   - Transaction boundaries (begin, commit, and rollback)

6. *View updating rule*. All views that are theoretically updateable are also updateable by the system.

7. *High-level insert*, *update*, *and delete*. The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data but also to the insertion, update, and deletion of data.

8. *Physical data independence*. Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.

9. *Logical data independence*. Application programs and terminal activities remain logically unimpaired when information preserving changes of any kind that theoretically permit unimpairment are made to the base tables.

10. *Integrity independence*. Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.

11. *Distribution independence.* A relational DBMS has distribution independence.

12. *Nonsubversion rule.* If a relational system has a low-level (single record at a time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher-level relational language (multiple records at a time).

During the early 1990s, it became popular practice to compile "scorecards" for commercial DBMS products, showing how well they satisfy each of the rules. Unfortunately, the rules are subjective so the scorecards were usually full of footnotes and qualifications, and didn't reveal a great deal about the products. Today, the basis of competition for database vendors tends to revolve around performance, new features, the availability of development tools, the quality of vendor support, and other issues, rather than conformance to Codd's rules. Nonetheless, they are an important part of the history of the relational model.

Rule 1 is basically the informal definition of a relational database presented at the beginning of this section.

Rule 2 stresses the importance of primary keys for locating data in the database. The table name locates the correct table, the column name finds the correct column, and the primary key value finds the row containing an individual data item of interest. Rule 3 requires support for missing data through `NULL` values, which are described in Chapter 5.

Rule 4 requires that a relational database be self-describing. In other words, the database must contain certain *system tables* whose columns describe the structure of the database itself. These tables are described in Chapter 16.

Rule 5 mandates using a relational database language, such as SQL, although SQL is not specifically required. The language must be able to support all the central functions of a DBMS—creating a database, retrieving and entering data, implementing database security, and so on.

Rule 6 deals with views, which are *virtual tables* used to give various users of a database different views of its structure. It is one of the most challenging rules to implement in practice, and no commercial product fully satisfies it today. Views and the problems of updating them are described in Chapter 14.

Rule 7 stresses the set-oriented nature of a relational database. It requires that rows be treated as sets in insert, delete, and update operations. The rule is designed to prohibit implementations that only support row-at-a-time, navigational modification of the database.

Rule 8 and Rule 9 insulate the user or application program from the low-level implementation of the database. They specify that specific access or storage techniques used by the DBMS, and even changes to the structure of the tables in the database, should not affect the user's ability to work with the data.

Rule 10 says that the database language should support integrity constraints that restrict the data that can be entered into the database and the database modifications that can be made. This is another of the rules that is not supported in most commercial DBMS products.

Rule 11 says that the database language must be able to manipulate distributed data located on other computer systems. Distributed data and the challenges of managing it are described in Chapter 20.

Finally, Rule 12 prevents "other paths" into the database that might subvert its relational structure and integrity.

Summary

SQL is based on the relational data model that organizes the data in a database as a collection of tables:

- Each table has a table name that uniquely identifies it.
- Each table has one or more named columns, which are arranged in a specific, left-to-right order.
- Each table has zero or more rows, each containing a single data value in each column. The rows are unordered.

- All data values in a given column have the same data type, and are drawn from a set of legal values called the domain of the column.

Tables are related to one another by the data they contain. The relational data model uses primary keys and foreign keys to represent these relationships among tables:

- A primary key is a column or combination of columns in a table whose value(s) uniquely identify each row of the table. A table has only one primary key.

- A foreign key is a column or combination of columns in a table whose value(s) are a primary key value for some other table. A table can contain more than one foreign key, linking it to one or more other tables.

- A primary key/foreign key combination creates a parent/child relationship between the tables that contain them.

Part II: Retrieving Data

Chapter List

Chapter 5: SQL Basics

Overview

This chapter begins a detailed description of the features of SQL. It describes the basic structure of a SQL statement and the basic elements of the language, such as keywords, data types, and expressions. The way that SQL handles missing data through `NULL` values is also described. Although these are basic features of SQL, there are some subtle differences in the way they are implemented by various popular SQL products, and in many cases the SQL products provide significant extensions to the capabilities specified in the ANSI/ISO SQL standard. These differences and extensions are also described in this chapter.

Statements

The main body of the SQL language consists of about 40 statements, which are summarized in Table 5-1. Each statement requests a specific action from the DBMS, such as creating a new table, retrieving data, or inserting new data into the database. All SQL statements have the same basic form, illustrated in Figure 5-1.