



Relational Database Management Systems

Those who are in love with practice without knowledge are like the sailor who gets into a ship without rudder or compass and who never can be certain whether he is going. Practice must always be founded on sound theory.

—*The Discourse on Painting* by Leonardo da Vinci

When I was a junior programmer, quite early in my career, my friends and I were assigned to work on a big software development project for which we would have to use unfamiliar technologies, though we were promised that training would be provided before the project started. All we knew in advance was that the operating system was something called VAX/VMS; we did not know which programming language or database would be used. The very first thing the instructor said was (paraphrasing) “First you have to insert your definitions into the CDD,” and he walked to the chalkboard and wrote the commands that we needed for the purpose. Needless to say, we were quite flustered because we had no idea what those “definitions” might be or what a “CDD” was and how it fit into the big picture.

I’ve been told that the first thing I should tell you is how to create an Oracle 11g database. Well, if you really must know, the necessary command is `CREATE DATABASE` followed by your choice of name for the database—anybody can type that command and create an Oracle 11g database. But the mere knowledge of a few Oracle commands (or even a lot of Oracle commands) will not make anyone an Oracle database administrator. What Leonardo said is so important that I’ll quote it again: “*Those who are in love with practice without knowledge are like the sailor who gets into a ship without rudder or compass and who never can be certain whether he is going. Practice must always be founded on sound theory.*” How can you competently administer a relational database management system like Oracle if you don’t really know what makes a “relational” database relational or what a database management system manages for you?

What Is a Database?

Chris Date was the keynote speaker at one of the educational conferences organized by the Northern California Oracle Users Group (NoCOUG), of whose journal I am the editor. The local television news station sent out a crew to cover the event because Chris Date is a well-known database theoretician and one of the associates of Dr. Edgar Codd, the inventor of relational database theory. The news reporter cornered me and asked me if I was willing to answer a few questions for the camera. I was quite flattered but when the reporter pointed the camera at me and asked “Why are databases important to society?” all I could think of to say was (paraphrasing) “Well, they’re important because they’re, like, *really* important, you know.” Ten years of database administration under my belt and I still flunked the final exam!

I’d therefore like us to spend just a few minutes at the outset considering what the word *database* signifies. An understanding of the implications of the word and the responsibilities that go along with them will serve you well as a good database administrator.

We might begin by saying that databases can contain data that is confidential and must be protected from prying eyes. Only authorized users should be able to access the data, their privileges must be suitably restricted, and their actions must be logged. Even if the data in the databases is for public consumption, we might still need to restrict who can update the data, who can delete from it, and who can add to it. Competent *security management* is therefore part of your job.

We might also say that databases can be critical to the ability of the organization to function properly. Organizations such as banks and e-commerce web sites require their databases to be available around the clock. Competent *availability management* is therefore an important part of your job. In the event of a disaster such as flood or fire, the databases may have to be relocated to an alternative location using backups. Competent *continuity management* is therefore another important part of your job. We also need competent *change management* to protect the database from unauthorized or badly tested changes, *incident management* to detect problems and restore service quickly, *problem management* to provide permanent fixes for known issues, *configuration management* to document infrastructure components and their dependencies, and *release management* to bring discipline to the never-ending task of applying patches and upgrades to software and hardware.

We might also observe that databases can be very big. The first database I worked with, for the semiconductor manufacturing giant Intel, was less than 100 megabytes in size and only had a few dozen data tables. Today, databases used by enterprise application suites like Peoplesoft, Siebel, and Oracle Applications are tens or hundreds of gigabytes in size and might have ten thousand tables or more. One reason databases are now so large is that advancements in magnetic disk storage technology have made it feasible to efficiently store and retrieve large quantities of nontextual data such as pictures and sound.

We might also note that databases can grow rapidly and that we need to plan for growth. We might also see that database applications might consume huge amounts of computing resources. *Capacity management* is therefore another important part of your job, and you need a capacity plan that accommodates both continuous data growth and increasing needs for computing resources.

When we stop thinking in terms of Oracle commands such as CREATE DATABASE and start thinking in terms such as security management, availability management, continuity management, change management, incident management, problem management, configuration management, release management, and capacity management, the business of database administration begins to make coherent sense and we become more effective database administrators. These terms are the part of the standard jargon of the IT Infrastructure Library (ITIL), a suite of best practices used by IT organizations throughout the world.

Now would you like to take a stab at answering the question that floored me in the television interview: *Why are databases important to society?*

What Is a *Relational* Database?

Relational database theory was invented by Dr. Edgar Codd in 1970 in a paper titled “A Relational Model for Data for Large Shared Data Banks.”¹ He based his theory on rigorous mathematical principles and used the correct mathematical term *relation* to describe what we loosely refer to as a *table*. The word *table* is not a mathematical term, but *relation* is a precisely defined mathematical term, and a lot of good mathematics can be built around its definition.

Definition of the Term *Relation*

In simple terms, a relation is an association of the members of two or more sets. Here is the precise definition found in Dr. Codd’s paper:

Given sets S_1, S_2, \dots, S_n (not necessarily distinct), R is a relation on these n sets if it is a set of n -tuples each of which has its first element from S_1 , its second element from S_2 , and so on.

1. Codd, Edgar. “A Relational Model for Data for Large Shared Data Banks,” *Communications of the ACM*, Volume 13, Issue 6 (June 1970).

Well, this just seems to be a boring mathematical way of complicating a simple concept like a table and does not explain why the relational approach swept aside all that came before it. To understand why the relational approach was revolutionary, we have first to understand the technologies that came before it and how they were deficient. We must then study the “relational operators” that produce new relations from old.

Network Databases

An example of a pre-relational database technology was the “network database” technology, one of the best examples of which was DEC/DBMS, created by Digital Equipment Corporation for the VAX/VMS and OpenVMS platforms—it still survives today as Oracle/DBMS. Yes, it’s strange but it’s true—Oracle Corporation, the maker of the world’s dominant relational database technology, also sells a prerelational database technology. According to Oracle Corporation, Oracle/DBMS is a very powerful, reliable and sophisticated database technology that has continued relevance and that Oracle is committed to supporting. Here are some quotes from Oracle Corporation’s web site.

CODASYL DBMS is a multiuser, CODASYL-compliant database management system for OpenVMS operating systems. CODASYL DBMS is designed for databases of all levels of complexity, ranging from simple hierarchies to sophisticated networks with multilevel relationships. CODASYL DBMS provides a reliable operating platform for application environments where stability, high availability, and throughput are essential. ... Oracle’s strategy for CODASYL DBMS beyond Release 7 is continued emphasis on availability, VLDB capabilities, and performance. ... Our overall objective is that of continuous support and enhancement to ensure DBMS keeps its reputation for stability and quality.²

In a network database, data records are linked together in chains. Consider an example involving three record types: SUPPLIER, PART, and QUOTE; each SUPPLIER record stores information about a supplier, each PART record stores information about a part, and each QUOTE record stores the price quoted by a supplier for a part. Figure 1-1 shows how records might be linked together. We see that hammers are supplied by three different suppliers and that New Yankee Workshop, Inc. has quoted the lowest price.

2. CODASYL is the Conference on Data Systems Languages, an industry consortium that wrote the specifications for the COBOL programming language.

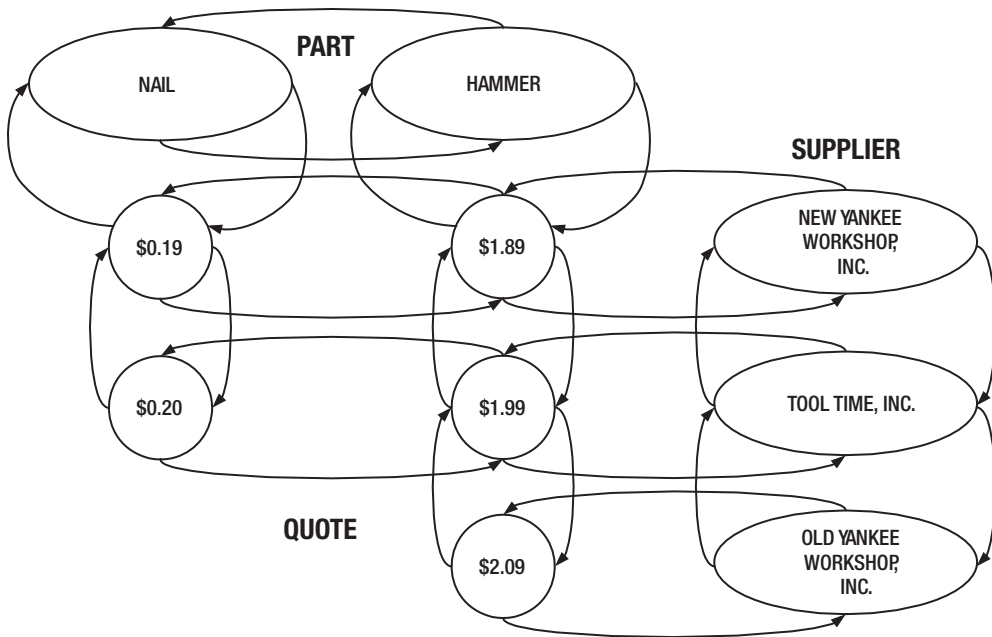


Figure 1-1. Relationship between suppliers and parts

Assuming that the chains of quotes are sorted in ascending order and that individual PART records and SUPPLIER records can be quickly located using the *hash* technique,³ the database organization shown in Figure 1-1 allows us to accomplish the following tasks.

1. List all the information available for a specified part—the hash indexing method allows us to retrieve the required PART record quickly.
2. List all the information available for a specified supplier—the hash indexing method also allows us to retrieve the required SUPPLIER record quickly.
3. List all parts—this can be answered by traversing the chain of PART records from the beginning to the end.
4. List all suppliers—this can be answered by traversing the chain of SUPPLIER records from the beginning to the end.
5. List all suppliers of a specified part—this can be answered by traversing the chain of QUOTE records linked to the specific PART record.

3. *Hash* techniques are used to compute a numeric value from a nonnumeric value such as a part code. A data record can then be stored at the address corresponding to this numeric value and can be found at this address at a future time. Hash techniques thus serve the same purpose as an index.

6. List all parts supplied by a specified supplier—this can be answered by traversing the chain of QUOTE records linked to the specific SUPPLIER record.
7. List the supplier who has quoted the lowest price for a specified part—this can be answered by finding the first QUOTE record linked to the specific PART record.

But other tasks such as “list the suppliers who supply all parts,” “list the parts that are supplied by all suppliers,” and “list the suppliers who supply all the parts supplied by a specified supplier at cheaper prices” cannot be easily accomplished using the network database structure diagrammed in Figure 1-1. But they are easily accomplished with a relational database, as we shall soon see.

Definition of a *Relational* Database

Relational database technology swept aside the older technologies precisely because it proved flexible enough to answer all kinds of questions; not just a small set of questions. This is because relational databases come with *relational operators* that produce new relations from old. Here, then, is a “rough and ready” definition (slightly paraphrased) of a relational database from C.J. Date’s *An Introduction to Database Systems, Eighth Edition* (Addison-Wesley, 2003):

A relational database is a database in which: The data is perceived by the user as tables (and nothing but tables)⁴ and the operators available to the user for (for example) retrieval are operators that derive “new” tables from “old” ones.

Relational Operators

Let’s examine some relational operators and use them to answer the question: *Which suppliers supply all parts?* But first, we need to organize our data into tables. Tables 1-1 through 1-3 list the contents of a sample database that we’ll use to explore the workings of relational operators.

Table 1-1. *The Part Table*

PartName
HAMMER
NAIL

4. Having explained the origin of the word *relation*, we can start using the more common term *table* wherever we are referring to a relation.

Table 1-2. *The Supplier Table*

SupplierName
NEW YANKEE WORKSHOP, INC.
OLD YANKEE WORKSHOP, INC.
TOOL TIME, INC.

Table 1-3. *The Quote Table*

SupplierName	PartName	Quote
NEW YANKEE WORKSHOP, INC.	HAMMER	\$1.89
NEW YANKEE WORKSHOP, INC.	NAIL	\$0.19
OLD YANKEE WORKSHOP, INC.	HAMMER	\$2.09
TOOL TIME, INC.	HAMMER	\$1.99
TOOL TIME, INC.	NAIL	\$0.20

Table 1-4 lists the definitions of five relational operators, four of which we will need to answer the question: *Which suppliers supply all parts?*

Table 1-4. *Five Relational Operators*⁵

Operator	Definition
Selection	Form another table by extracting a subset of the rows of a table of interest using some criteria.
Projection	Form another table by extracting a subset of the columns of a table of interest. Any duplicate rows that are formed as a result of the projection operation are eliminated.
Union	Form another table by selecting all rows from two tables of interest. If the first table has 10 rows and the second table has 20 rows, then the resulting table will have at most 30 rows, because duplicates will be eliminated from the result.
Difference	Form another table by extracting only those rows from one table of interest that do not occur in a second table.
Join	Form another table by concatenating records from two tables of interest. For example, if the first table has 10 rows and the second table has 20 rows, then the resulting table will have 200 rows—and if the first table has 10 columns and the second table has 20 columns, then the resulting table will have 30 columns.

We can compute the answer to the question “*Which suppliers supply all parts?*” in a sequence of five steps. At each step, we use one of the relational operators just listed and create an intermediate result table.

5. It is possible to create new operations by combining the listed operations. For example, “Natural Join” is the result produced by a Join operation on two tables followed by a Selection operation on the resulting intermediate table.

1. In the first step, we use the Join operation and form an intermediate result table by concatenating records from the Suppliers table and the Parts table. All combinations of SupplierName and PartName occur in this table. Table 1-5 shows the result.

Table 1-5. *All SupplierName and PartName Combinations*

SupplierName	PartName
NEW YANKEE WORKSHOP, INC.	HAMMER
NEW YANKEE WORKSHOP, INC.	NAIL
OLD YANKEE WORKSHOP, INC.	HAMMER
OLD YANKEE WORKSHOP, INC.	NAIL
TOOL TIME, INC.	HAMMER
TOOL TIME, INC.	NAIL

2. In the second step, we use the Projection operation and form another intermediate result table by extracting the SupplierName and PartName columns from the Quotes table. The result in Table 1-6 is the list of valid SupplierName and PartName combinations.

Table 1-6. *Valid SupplierName and PartName Combinations*

SupplierName	PartName
NEW YANKEE WORKSHOP, INC.	HAMMER
NEW YANKEE WORKSHOP, INC.	NAIL
OLD YANKEE WORKSHOP, INC.	HAMMER
TOOL TIME, INC.	HAMMER
TOOL TIME, INC.	NAIL

3. In the third step, we use the Difference operation and form a third intermediate result table, shown in Table 1-7, by extracting only those rows from the intermediate result table created in the first step that are not to be found in the intermediate result table created in the second step. The occurrence of a certain combination of SupplierName and PartName in this new intermediate table indicates that the supplier in question does not supply the indicated part.

Table 1-7. *Invalid SupplierName and PartName Combinations*

SupplierName	PartName
OLD YANKEE WORKSHOP, INC.	NAIL

4. In the fourth step, we use the Projection operation and form yet another intermediate result table by extracting only the first column from the intermediate result table created in the third step. The result, shown in Table 1-8, is the list of suppliers who do not supply at least one part.

Table 1-8. *Suppliers Who Do Not Supply All Parts*

SupplierName

OLD YANKEE WORKSHOP, INC.

5. In the fifth and final step, we use the Difference operation once again and obtain the final result we were seeking by extracting only those rows from the Suppliers table that do not occur in the intermediate result table of the fourth step. Table 1-9 shows the final result, which is the required list of suppliers who do supply all parts!

Table 1-9. *Suppliers Who Supply All Parts*

SupplierName

NEW YANKEE WORKSHOP, INC.

TOOL TIME, INC.

RELATIONAL ALGEBRA EXPRESSIONS

Just as numbers and arithmetical symbols such as addition and multiplication can be combined into an arithmetical expression, so also can tables and table operators be combined into a relational algebra expression. We can specify the previous sequence of steps in a single expression as shown here.

```
Supplier MINUS PROJECTION((Part JOIN Supplier) MINUS PROJECTION(Quote))
```

Structured Query Language

The specification of relational algebra expressions is facilitated by an English-like language called Structured Query Language or SQL. As an example, let's look at the SQL formulation of the query *Which suppliers supply all parts?* Multiple formulations are possible, and the one shown in Listing 1-1 uses a technique called *subquery factoring* to produce the intended result using the same series of short steps that was used in the previous section.

Listing 1-1. *Suppliers Who Supply All Parts*

```

WITH

-- Step 1: Join operation
    supplierpart AS
    (SELECT suppliername, partname
     FROM supplier, part),

-- Step 2: Projection operation
    validsupplierpart AS
    (SELECT suppliername, partname
     FROM quote),

-- Step 3: Difference operation
    invalidsupplierpart AS
    (SELECT suppliername, partname
     FROM supplierpart
     MINUS
     SELECT suppliername, partname
     FROM validsupplierpart),

-- Step 4: Projection operation
    unwantedsupplier AS
    (SELECT suppliername
     FROM invalidsupplierpart),

-- Step 5: Difference operation
    wantedsupplier AS
    (SELECT suppliername
     FROM supplier
     MINUS
     SELECT suppliername
     FROM unwantedsupplier)

SELECT suppliername
FROM wantedsupplier;

```

The SQL statement shown in Listing 1-1 is fairly English-like and self-explanatory, and we will resume the discussion of SQL in the next chapter. For now, note how the formatting improves readability—the elegantly formatted version with vertical “rivers” and

capitalized “reserved words”⁶ shown in Listing 1-1 was produced using a tool called Toad and is completely equivalent to the unreadable version shown in Listing 1-2.

Listing 1-2. *Unreadable SQL Query*

```
with supplierpart as (select suppliername, partname from supplier, part),
validsupplierpart as (select suppliername, partname from quote),
invalidsupplierpart as (select suppliername, partname from supplierpart minus
select suppliername, partname from validsupplierpart), unwantedsupplier as (select
suppliername from invalidsupplierpart), wantedsupplier as (select suppliername from
supplier minus select suppliername from unwantedsupplier) select suppliername from
wantedsupplier;
```

Efficiency of Relational Operators

You may have noticed that the discussion in the previous section made no mention of efficiency. The definitions of the table operations do not explain how the results can be efficiently obtained. This is, in fact, intentional and is one of the greatest strengths of relational database technology—it is left to the database management system to provide efficient implementations of the table operations. In particular, the selection operation depends heavily on indexing schemes and Oracle Database provides a host of such schemes, including B-tree indexes, index-organized tables, partitioned tables, partitioned indexes, function indexes, reverse-key indexes, bitmap indexes, table clusters, and hash clusters. We’ll discuss indexing possibilities as part of physical database design in Chapter 7.

Query Optimization

Perhaps the most important aspect of relational algebra expressions is that, except in very simple cases, they can be rearranged in different ways to gain a performance advantage without changing their meaning or causing the results to change. The following two expressions are equivalent, except perhaps in the order in which data columns occur in the result—a minor presentation detail, not one that changes the meaning of the result.

Listing 1-3. *Joining Two Tables*

```
Table_1 JOIN Table_2
Table_2 JOIN Table_1
```

6. Words that have special meaning in SQL.

The number of ways in which a relational algebra expression can be rearranged increases dramatically as the expressions grow longer. Even the relatively simple expression `(Table_1 JOIN Table_2) JOIN Table_3` can be arranged in the following 12 equivalent ways that produce results differing only in the order in which columns are presented—a cosmetic detail that can be easily remedied before the results are shown to the user.

Listing 1-4. *Joining Three Tables*

```
(Table_1 JOIN Table_2) JOIN Table_3
(Table_1 JOIN Table_3) JOIN Table_2
(Table_2 JOIN Table_1) JOIN Table_3
(Table_2 JOIN Table_3) JOIN Table_1
(Table_3 JOIN Table_1) JOIN Table_2
(Table_3 JOIN Table_2) JOIN Table_1

Table_1 JOIN (Table_2 JOIN Table_3)
Table_1 JOIN (Table_3 JOIN Table_2)
Table_2 JOIN (Table_1 JOIN Table_3)
Table_2 JOIN (Table_3 JOIN Table_1)
Table_3 JOIN (Table_1 JOIN Table_2)
Table_3 JOIN (Table_2 JOIN Table_1)
```

It is not obvious at this stage what performance advantage, if any, is gained by rearranging relational algebra expressions. Nor is it obvious what criteria should be used while rearranging expressions. Suffice it to say that a relational algebra expression is intended to be a *nonprocedural* specification of an intended result and the *query optimizer* may take any actions intended to improve the efficiency of query processing as long as the result is not changed. Relational query optimization is the subject of much theoretical research, and the Oracle query optimizer continues to be improved in every release of Oracle Database. We shall return to the subject of SQL query tuning in Chapter 17.

What Is a Database Management System?

Database management systems such as Oracle are the interface between users and databases. Database management systems differ in the range of features they provide, but all of them offer certain core features such as *transaction management*, *data integrity*, and *security*. And, of course, they offer the ability to create databases and to define their structure, as well as to store, retrieve, update, and delete the data in the database.

Transaction Management

A *transaction* is a unit of work that may involve several small steps, all of which are necessary in order not to compromise the integrity of the database. For example, a *logical* operation such as inserting a row into a table may involve several *physical* operations such as index updates, *trigger* operations,⁷ and *recursive* operations.⁸ A transaction may also involve multiple logical operations. For, example transferring money from one bank account to another may require that two separate rows be updated. A DBMS needs to ensure that transactions are *atomic*, *consistent*, *isolated*, and *durable*.

The Atomicity Property of Transactions

It is always possible for a transaction to fail at any intermediate step. For example, the user may lose his or her connection to the database or the database may run out of space and may not be able to accommodate new data that the user is trying to store. If a failure occurs, the database management system performs automatic *rollback* of the work that has been performed so far. Transactions are therefore *atomic* or indivisible from a logical perspective. The end of a transaction is indicated by an explicit instruction such as COMMIT.

The Consistency Property of Transactions

Transactions also have the *consistency* property. That is, they do not compromise the integrity of the database. However, it is easy to see that the database may be *temporarily* inconsistent during the operation of the transaction. In the previous example, the database is in an inconsistent state when money has been subtracted from the balance in the first account but has not yet been added to the balance in the second account.

The Isolation Property of Transactions

Transactions also have the *isolation* property; that is, concurrently occurring transactions must not interact in ways that produce incorrect results. A database management system must be capable of ensuring that the results produced by concurrently executing transactions are *serializable*; that is, the outcome must be the same as if the transactions were executed in serial fashion instead of concurrently.

For example, suppose that one transaction is withdrawing money from a bank customer's checking account, and another transaction is simultaneously withdrawing money from

7. Trigger operations are operations that are automatically performed when the triggering event occurs. For example, an attempt to update data in one table may cause a *log record* to be written to another table.

8. Recursive operations are *management operations* that are performed by the database in order to support user operations. For example, an attempt to insert new data into a table might necessitate that additional space be allocated to accommodate the new data.

the same customer's savings account. Let's assume that negative balances are permitted as long as the *sum* of the balances in the two accounts is not negative. Suppose that the operation of the two transactions proceeds in such a way that each transaction determines the balances in both accounts before either of them has had an opportunity to update either balance. Unless the database management system does something to prevent it, this can potentially result in a negative sum. This kind of problem is called *write skew*.

A detailed discussion of isolation and serializability properly belongs in an advanced course on application development, not in a beginner text on database administration. The interested reader will find more information online, in the *Oracle 11g Advanced Application Developer's Guide*, available at <http://www.oracle.com/technology/documentation/index.html>.

The Durability Property of Transactions

Transactions also have the *durability* property. This means that once all the steps in a transaction have been successfully completed and the user notified, the results must be considered permanent even if there is a subsequent computer failure, such as a damaged disk. We will return to this topic in the chapters on database backups and recovery; for now, we note that the end of a transaction is indicated by an explicit command such as COMMIT.

Data Integrity

Data loses its value if it cannot be trusted to be correct. A database management system provides the ability to define and enforce *integrity constraints*. The database management system will reject any attempt to violate the integrity constraints when inserting, updating, or deleting data records and will typically display an appropriate error code and message. In fact, the very first Oracle error code, "ORA-00001," relates to attempts to violate an integrity constraint. It is possible to enforce arbitrary constraints using trigger operations; these can include checks that are as complex as necessary, but the more common types of constraints are *check constraints*, *uniqueness constraints*, and *referential constraints*.

Check Constraints

Check constraints are usually simple checks on the values of a data item. For example, a price quote must not be less than \$0.00.

Uniqueness Constraints

A uniqueness constraint requires that some part of a record be unique. For example, two employees may not have the same employee number. A unique part of a record is called a *candidate key*, and one of the candidate keys is designated as the *primary key*. Intuitively, we expect every record to have at least one candidate key; otherwise, we would have no

way of specifying which records we needed. Note that the candidate key can consist of a single item from the data record, a combination of items, or even all the items.

Referential Constraints

Consider the example of an employee database in which all payments to employees are recorded in a table called SALARY. The employee number in a salary record must obviously correspond to the employee number in some employee record; this is an example of a *referential constraint*.⁹

Data Security

A database management system gives the owners of the data a lot of control over their data—they can delegate limited rights to others if they choose to. It also gives the database administrator the ability to restrict and monitor the actions of users. For example, the database administrator can disable the password of an employee who leaves the company, to prevent him or her from gaining access to the database. Relational database management systems use techniques such as *views* (virtual tables defined in terms of other tables) and *query modification* to give individual users access to just those portions of data they are authorized to use.

Oracle offers extensive query modification capabilities under the name of Virtual Private Database (VPD), but here is a simple example of the technique from a database management system called Ingres. A manager named Solomon Grundy is being given permission to retrieve and update the name, age, and salary of just those employees that he manages. When he tries to retrieve records from the employee data table, the following additional clauses are silently appended to his query:

```
employee.departmentnumber = department.departmentnumber  
and department.managername = "Solomon Grundy"
```

9. Database management systems are a perpetual work in progress and each new version offers new features. Strange as it may sound, the Oracle database management system did not enforce “referential integrity” constraints until Version 7 was released in the 1990s (by which time it was already the world’s largest database company). It would not, for example, prevent a salary from being inadvertently paid to a non-existent employee. The following quote from an article published in *Software* magazine in 1989 alludes to a time when network database management systems outclassed relational database management systems in areas such as data integrity:

“About six or seven years ago when I worked for a vendor that made a [network] DBMS called Seed, I spoke at a conference. Also speaking was Larry Rowe, one of the founders of Relational Technology, Inc. and one of the developers of the relational DBMS Ingres. We were about to be clobbered by these new relational systems. He suggested to me that the best way to compete against the relational systems was to point out that they did not support referential integrity.”

This filters out rows of data that Solomon Grundy is not allowed to see by joining each employee record with the corresponding department record and checking that the manager of the department is none other than Solomon Grundy. Furthermore, as shown in Listing 1-5, Solomon Grundy can only get access to the data from terminal “tta2” and only from 8 a.m. to 5 p.m. on weekdays.

Listing 1-5. Query Modification

```
define permit retrieve, replace of employee (employeename, age, salary)
to sgrundy at "tta2" from 8:00 to 17:00 on mon to fri
where employee.departmentnumber = department.departmentnumber
and department.managername = "Solomon Grundy"
```

We shall return to the subject of data security in a future chapter.

What Makes a Relational Database Management System Relational?

Having already discussed the meaning of both *relational database* and *database management system*, it might appear that the subject is settled. But the natural implications of the relational model are so numerous and profound that critics contend that, even today, a “truly relational” database management system does not exist. For example, Dr. Edgar Codd, the inventor of relational database theory, wanted the database management system to treat “views” in the same manner as “base tables” whenever possible but the problem of view updateability is unsolved to the present day. Dr. Codd listed more than 300 separate requirements that a database management system must meet in order to fulfill his vision properly, and we have time for just one of them—*physical data independence*. Here is the relevant quote from Dr. Codd’s book (*The Relational Model for Database Management: Version 2*. Addison Wesley, 1990):

RP-1 Physical Data Independence: The DBMS permits a suitably authorized user to make changes in storage representation, in access method, or in both—for example, for performance reasons. Application programs and terminal activities remain logically unimpaired whenever any such changes are made.

Summary

I hope that you now have an appreciation for the theoretical foundations of Oracle 11g. More information on the subjects we touched upon can be found in the books mentioned in the bibliography at the end of the chapter. Here is a short summary of the concepts we discussed in this chapter.

- A database is an information repository that must be competently administered using the principles laid out in the IT Infrastructure Library (ITIL) including *security management, availability management, continuity management, change management, incident management, problem management, configuration management, release management, and capacity management*.
- A *relation* is a precise mathematical term for what we loosely call a data table. Relational database technology swept aside earlier technologies because of the power and expressiveness of *relational algebra* and because it made performance the responsibility of the database management system instead of the application developer.
- A database management system provides efficient algorithms for the processing of table operations as well as indexing schemes for data storage. The *query optimizer* rearranges relational algebra expressions in the interests of efficiency but without changing the meaning or the results that are produced.
- A *database management system* is defined as a software layer that provides services such as *transaction management, data security, and data integrity*.
- A *transaction* is a logical unit of work characterized by *atomicity, consistency, isolation, and durability*.
- Relational database theory has many consequences including that of *logical data independence*, which implies that changes to the way in which data is stored or indexed should not affect the logical behavior of application programs.

Exercises

The following exercises, based on this chapter's supplier-part example, will help you appreciate the expressive power of relational algebra; in each case, you will need only the table operators that are described in this chapter.

- List the parts that are supplied by all suppliers.
- List the parts that are supplied by OLD YANKEE WORKSHOP, INC. as well as by TOOL TIME, INC.

- List the suppliers who supply at least one part that is not supplied by a specified supplier.
- List the suppliers who do not supply at least one part that is supplied by a specified supplier.
- List the suppliers who supply all the parts supplied by a specified supplier but no others.
- List the suppliers who supply all the parts supplied by a specified supplier at cheaper prices.
- List the parts that are only supplied by a specified supplier.

These exercises are a good preparation for the discussion of Structured Query Language in the next chapter.

Further Reading

Silberschatz, Abraham; Korth, Henry; Sudarshan S. *Database Systems Concepts, Fifth Edition*. McGraw-Hill, 2005. If you had to buy just one book on database theory, this is the one that I would recommend. This standard college textbook, now in its fifth edition, offers not only a fair amount of theory but also coverage of three of the leading commercial relational database management systems: Oracle, IBM DB2 Universal Database, and Microsoft SQL Server. It also covers the leading open-source relational database management system, PostgreSQL.

Date, C. J. *An Introduction to Database Systems, Eighth Edition*. Addison Wesley, 2003. This well-known work, now in its eighth edition, whose author is one of the world's foremost database theoreticians and an associate of Dr. Edgar Codd, takes an extremely rigorous and theoretical approach, which makes the book suitable only for the most dedicated students of relational database technology.

Codd, E. F. *The Relational Model for Database Management: Version 2*. Addison Wesley, 1990. This famous book, now out of print, by the inventor of relational database technology lists more than 300 requirements that a relational database management system must meet in order to fulfill his vision. It makes fascinating reading but, like Date's book, is suitable only for the most dedicated students of relational database technology.