Section 4: Managing Organizational Memory

Every one complains of his memory, none of his judgment.

François Duc de La Rochefoucauld “Sentences et Maximes,” Morales No. 89 1678

In order to manage data, you need a database architecture, a design for the storage and processing of data. Organizations strive to find an architecture that simultaneously achieves multiple goals:

1. Responds in a timely manner.
2. Minimizes the cost of

processing data,

storing data,

data delivery,

application development.

1. Highly secure.

The section deals with the approaches that can be used to achieve these goals. It covers database structure and storage alternatives. It provides the knowledge necessary to determine an appropriate data storage structure and device for a given situation. It also addresses the fundamental questions of where to store the data and where they should be processed.

Java has become a popular choice for writing programs that operate on multiple operating systems. Combining the interoperability of Java with standard SQL provides software developers with an opportunity to develop applications that can run on multiple operating systems and multiple DBMSs, and this typically reduces the cost of application development. Thus, for a comprehensive understanding of data management, it is important to learn how Java and SQL interface, which is the subject of the third chapter in this section.

As you now realize, organizational memory is an important resource requiring management. An inaccurate memory can result in bad decisions and poor customer service. Some aspects of organizational memory (e.g., chemical formulae, marketing strategy, and R&D plans) are critical to the well-being of an organization. The financial consequences can be extremely significant if these memories are lost or fall into the hands of competitors. Consequently, organizations must develop and implement procedures for maintaining data integrity. They need policies to protect the existence of data, maintain its quality, and ensure its confidentiality. Some of these procedures may be embedded in organizational memory technology, and others may be performed by data management staff. Data integrity is also addressed in this section.

When organizations recognize a resource as important to their long-term viability, they typically create a formal mechanism to manage it. For example, most companies have a human resources department responsible for activities such as compensation, recruiting, training, and employee counseling. People are the major resource of nearly every company, and the human resources department manages this resource. Similarly, the finance department manages a company’s financial assets.

In the information age, data—the raw material of information—need to be managed. Consequently, data administration has become a formal organizational structure in many enterprises. It is designed to administer the data management and exploitation resources of an organization.

20. Data Structure and Storage

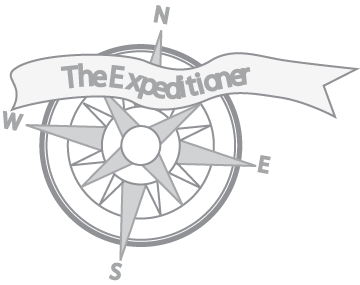
The modern age has a false sense of superiority because it relies on the mass of knowledge that it can use, but what is important is the extent to which knowledge is organized and mastered.

Goethe, 1810

## Learning objectives

Students completing this chapter will, for a given situation, be able to recommend

* understand the implications of the data deluge;
* a data storage structure;
* a storage device.

Every quarter, The Expeditioner’s IS group measures the quality of its service. It asks its clients to assess whether their hardware and software are adequate for their jobs, whether IS service is reliable and responsive, and whether they thought the IS staff was helpful and knowledgeable. The most recent survey revealed that some were experiencing unreasonably long delays for what were relatively simple queries. How could the IS group tune the database to reduce response time?

As though some grumpy clients were not enough, Alice dumped another problem on Ned’s desk. The Marketing department had complained to her that the product database had been down for 30 minutes during a peak selling period. What was Ned going to do to prevent such an occurrence in the future?

Ned had just finished reading Alice’s memo about the database problem when the Chief Accountant poked his head in the door. Somewhat agitated, he was waving an article from his favorite accounting journal that claimed that data stored on magnetic tape decayed with time. So what was he to do with all those financial records on magnetic tapes stored in the fireproof safe in his office? Were the magnetic bits likely to disappear tonight, tomorrow, or next week? “This business had lasted for centuries with paper ledgers. Why, you can still read the financial transactions for 1527,” which he did whenever he had a few moments to spare. “But, if what I read is true, I soon won’t be able to read the balance sheet from last year!”

It was just after 10 a.m. on a Monday, and Ned was faced with finding a way to improve response time, ensure that databases were continually available during business hours, and protect the long-term existence of financial records. It was going to be a long week.

# The data deluge

With petabytes of new data being created daily, and the volume continuing to grow, many IS departments and storage vendors struggle to handle this data flood. “Big Data,” as the deluge is colloquially known, arises from the flow of data created by Internet searches, Web site visits, social networking activity, streaming of videos, electronic health care records, sensor networks, large-scale simulations, and a host of other activities that are part of everyday business in today’s world. The deluge requires a continuing investment in the management and storage of data.

A byte size table

| Abbreviation | Prefix | Factor | Equivalent to |
| --- | --- | --- | --- |
| k | kilo | 103 |  |
| M | mega | 106 |  |
| G | giga | 109 | A digital audio recording of a symphony |
| T | tera | 1012 |  |
| P | peta | 1015 | 50 percent of all books in U.S. academic libraries |
| E | exa | 1018 | 5 times all the world’s printed material |
| Z | zetta | 1021 |  |
| Y | yotta | 1024 |  |

The following pages explore territory that is not normally the concern of application programmers or database users. Fortunately, the relational model keeps data structures and data access methods hidden. Nevertheless, an overview of what happens under the hoodis part of a well-rounded education in data management and will equip you to work on some of the problems of the data deluge.

Data structures and access methods are the province of the person responsible for physically designing the database so that it responds in a timely manner to both queries and maintenance operations. Of course, there may be installations where application programmers have such responsibilities, and in these situations you will need to know physical database design.

# Data structures

An in-depth consideration of the internals of database architecture provides an understanding of the basic structures and access mechanisms underlying the technology. As you will see, the overriding concern of the internal level is to minimize disk access. In dealing with this level, we will speak in terms of files, records, and fields rather than the relational database terms of tables, rows, and columns. We do this because the discussion extends beyond the relational model to file structures in general.

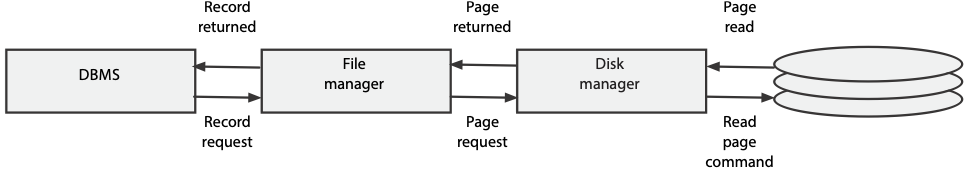
The time required to access data on a magnetic disk, the storage device for many databases, is relatively long compared to that for main memory. Disk access times are measured in milliseconds (10–3), and main memory access times are referred to in nanoseconds (10–9). There are generally around five orders of magnitude difference between disk and main memory access—it takes about 105 times longer. This distinction is more meaningful if placed in an everyday context; it is like asking someone a question by phone or writing them a letter. The phone response takes seconds, and the written response takes days.

For many business applications, slow disk drives are a bottleneck. The computer often must wait for a disk to retrieve data before it can continue processing a request for information. This delay means that customers are also kept waiting. Appropriate selection of data structures and data access methods can considerably reduce delays. Database designers have two options: decrease disk read/write head movement or reduce disk accesses. Before considering these options, we need a general model of database access.

## Database access

A three-layer model provides a framework for thinking about minimization of data access. This is a generic model, and a particular DBMS may implement the approach using a different number of layers. For simplicity, the discussion is based on retrieving a single record in a file, although the principles also apply to the retrieval of multiple records or an entire file.

Database access layers



1. The DBMS determines which record is required and passes a request to the file manager to retrieve a particular record in a file.
2. The file manager converts this request into the address of the unit of storage (usually called a page) containing the specified record. A page is the minimum amount of storage accessed at one time and is typically around 1–4 kbytes. A page will often contain several short records (e.g., 200 bytes), but a long record (e.g., 10 kbytes) might be spread over several pages. In this example, we assume that records are shorter than a page.
3. The disk manager determines the physical location of the page, issues the retrieval instructions, and passes the page to the file manager.
4. The file manager extracts the requested record from the page and passes it to the DBMS.

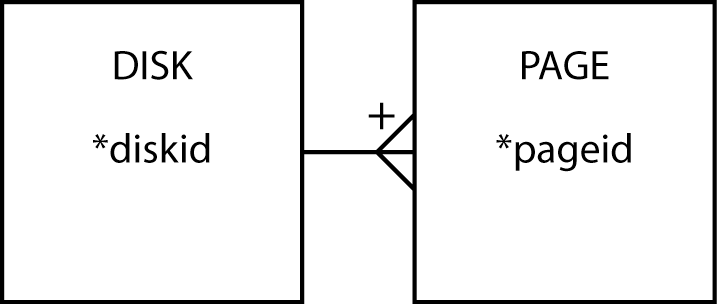
## The disk manager

The disk manager is that part of the operating system responsible for physical input and output (I/O). It maintains a directory of the location of each page on the disk with all pages identified by a unique page number. The disk manager’s main functions are to retrieve pages, replace pages, and keep track of free pages.

Page retrieval requires the disk manager to convert the page number to a physical address and issue the command to read the physical location. Since a page can contain multiple records, when a record is updated, the disk manager must retrieve the relevant page, update the appropriate portion containing the record, and then replace the page without changing any of the other data on it.

The disk manager thinks of the disk as a collection of uniquely numbered pages. Some of these pages are allocated to the storage of data, and others are unused. When additional storage space is required, the disk manager allocates a page address from the set of unused page addresses. When a page becomes free because a file or some records are deleted, the disk manager moves that page’s address to the unallocated set. Note, it does not erase the data, but simply indicates the page is available to be overwritten. This means that it is sometimes possible to read portions of a deleted file. If you want to ensure that an old file is truly deleted, you need to use an erase program that writes random data to the deleted page.

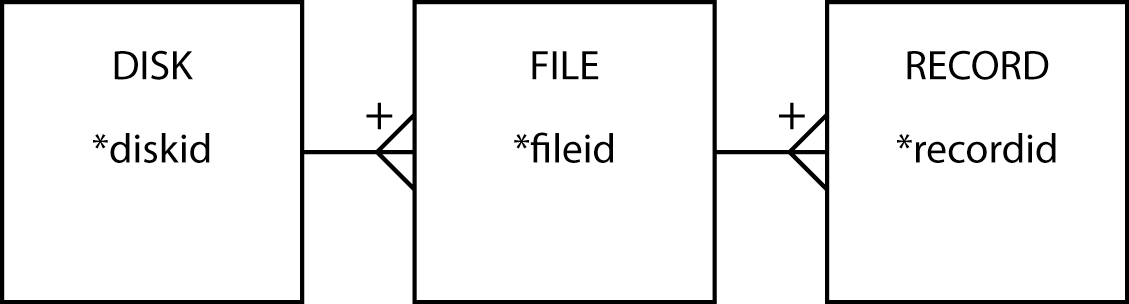
Disk manager’s view of the world



## The file manager

The file manager, a level above the disk manager, is concerned with the storage of files. It thinks of the disk as a set of stored files. Each file has a unique file identifier, and each record within a file has a record identifier that is unique within that file.

File manager’s view of the world



The file manager can

* Create a file
* Delete a file
* Retrieve a record from a file
* Update a record in a file
* Add a new record to a file
* Delete a record from a file

## Techniques for reducing head movement

All disk storage devices have some common features. They have one or more recording surfaces. Typically, a magnetic disk drive has multiple recording surfaces, with data stored on tracks on each surface.

The key characteristics of disk storage devices that affect database access are rotational speedandaccess arm speed. The rotational speed of a magnetic disk is in the range of 3,000 to 15,000 rpm. Reading or writing a page to disk requires moving the read/write head to the destination track and waiting for the storage address to come under the head. Because moving the head usually takes more time (e.g., about 9 msec) than waiting for the storage address to appear under it (e.g., about 4 msec), data access times can be reduced by minimizing the movement of the read/write head or by rotating the disk faster. Since rotational speed is set by the disk manufacturer, minimizing read/write head movement is the only option available to database designers.

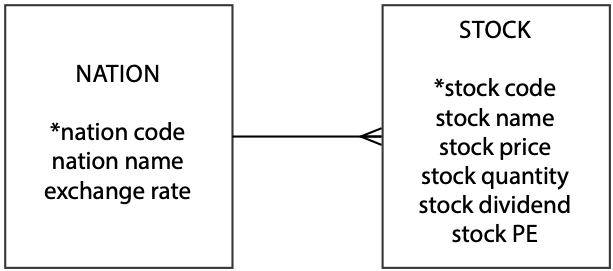
### Cylinders

Head movement is reduced by storing data that are likely to be accessed at the same time, such as records in a file, on the same track on a single surface. When a file is too large to fit on one track, then it can be stored on the same track on different surfaces (i.e., one directly above or below the current track); such a collection of tracks is called a cylinder. The advantage of cylinder storage is that all tracks can be accessed without moving the read/write head. When a cylinder is full, remaining data are stored on adjacent cylinders. Adjacent cylinders are ideal for sequential file storage because the record retrieval pattern is predefined—the first record is read, then the second, and so on.

### Clustering

Cylinder storage can also be used when the record retrieval pattern has some degree of regularity to it. Consider the following familiar data model of the following figure. Converting this data model to a relational database creates two tables. Conceptually, we may think of the data in each of the tables as being stored in adjacent cylinders. If, however, you frequently need to retrieve one row of nation and all the corresponding rows of stock, then nation and stock rows should be intermingled to minimize access time.

Data model for nation and stock



The term **clustering** denotes the concept that records that are frequently used together should be physically close together on a disk. Some DBMSs permit the database designer to specify clustering of different files to tune the database to reduce average access times. If usage patterns change, clustering specifications should be altered. Of course, clustering should be totally transparent to application programs and clients.

## Techniques for reducing disk accesses

Several techniques are used to accelerate retrieval by reducing disk accesses. The ideal situation is when the required record is obtained with a single disk access. In special circumstances, it may be possible to create a file where the primary key can convert directly to a unique disk address (e.g., the record with primary key 1 is stored at address 1001, the record with primary key 2 is stored at address 1002, and so on). If possible, this method of direct addressing should be used because it is the fastest form of data access; however, it is most unusual to find such a direct mapping between the primary key and a disk address. For example, it would not be feasible to use direct addressing with a student file that has a Social Security number as the primary key because so many disk addresses would be wasted. Furthermore, direct addressing can work only for the primary key. What happens if there is a need for rapid retrieval on another field?

In most cases, database designers use features such as indexing, hashing, and linked lists. Indexing, a flexible and commonly used method of reducing disk accesses when searching for data, centers on the creation of a compact file containing the index field and the address of its corresponding record. The B-tree is a particular form of index structure that is widely used as the storage structure of relational DBMSs. Hashing is a direct access method based on using an arithmetic function to compute a disk address from a field within a file. A linked lis**t** is a data structure that accelerates data access by using pointers to show relationships existing between records.

## Indexing

Consider the item file partially shown in the following table. Let’s assume that this file has 10,000 records and each record requires 1 kbyte, which is a page on the particular disk system used. Suppose a common query is to request all the items of a particular type. Such a query might be

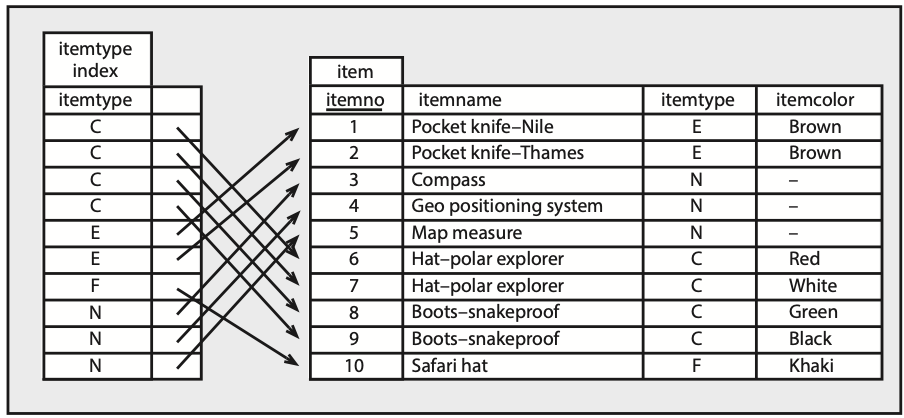
* Find all items of type E.

Regardless of the number of records with itemtype = 'E', this query will require 10,000 disk accesses. Every record has to be retrieved and examined to determine the value of itemtype. For instance, if 20 percent of the items are of type E, then 8,000 of the disk accesses are wasted because they retrieve a record that is not required. The ideal situation would be to retrieve only those 2,000 records that contain an item of type E. We get closer to this ideal situation by creating a small file containing just the value of itemtype for each record and the address of the full record. This small file is called an index.

Portion of a 10,000 Record File

| itemno | itemname | itemtype | itemcolor |
| --- | --- | --- | --- |
| 1 | Pocket knife—Nile | E | Brown |
| 2 | Pocket knife—Thames | E | Brown |
| 3 | Compass | N | — |
| 4 | Geopositioning system | N | — |
| 5 | Map measure | N | — |
| 6 | Hat—polar explorer | C | Red |
| 7 | Hat—polar explorer | C | White |
| 8 | Boots—snake proof | C | Green |
| 9 | Boots—snake proof | C | Black |
| 10 | Safari chair | F | Khaki |

Part of the itemtype index



The itemtype index is a file. It contains 10,000 records and two fields. There is one record in the index for each record in item. The first columns contains a value of itemtype and the second contains a pointer, an address, to the matching record of item. Notice that the index is in itemtype sequence. Storing the index in a particular order is another means of reducing disk accesses, as we will see shortly. The index is quite small. One byte is required for itemtype and four bytes for the pointer. So the total size of the index is 50 kbytes, which in this case is 50 pages of disk space.

Now consider finding all records with an item type of E. One approach is to read the entire index into memory, search it for type E items, and then use the pointers to retrieve the required records from item. This method requires 2,050 disk accesses—50 to load the index and 2,000 accesses of item, as there are 2,000 items of type E. Creating an index for item results in substantial savings in disk accesses for this example. Here, we assume that 20 percent of the records in item contained itemtype = 'E'. The number of disk accesses saved varies with the proportion of records meeting the query’s criteria. If there are no records meeting the criteria, 9,950 disk accesses are avoided. At the other extreme, when all records meet the criteria, it takes 50 extra disk accesses to load the index.

The SQL for creating the index is

CREATE INDEX itemtypeindx ON item (itemtype);

The entire index need not be read into memory. As you will see when we discuss tree structures, we can take advantage of an index’s ordering to reduce disk accesses further. Nevertheless, the clear advantage of an index is evident: it speeds up retrieval by reducing disk accesses. Like many aspects of database management, however, indexes have a drawback. Adding a record to a file without an index requires a single disk write. Adding a record to an indexed file requires at least two, and maybe more, disk writes, because an entry has to be added to both the file and its index. The trade-off is between faster retrievals and slower updates. If there are many retrievals and few updates, then opt for an index, especially if the indexed field can have a wide variety of values. If the file is very volatile and updates are frequent and retrievals few, then an index may cost more disk accesses than it saves.

Indexes can be used for both sequential and direct access. Sequential access means that records are retrieved in the sequence defined by the values in the index. In our example, this means retrieving records in itemtype sequence with a range query such as

* Find all items with a type code in the range E through K.

Direct access means records are retrieved according to one or more specified values. A sample query requiring direct access would be

* Find all items with a type code of E or N.

Indexes are also handy for existence testing. Remember, the EXISTS clause of SQL returns true or false and not a value. An index can be searched to check whether the indexed field takes a particular value, but there is no need to access the file because no data are returned. The following query can be answered by an index search:

* Are there any items with a code of R?

### Multiple indexes

Multiple indexes can be created for a file. The item file could have an index defined on itemcolor or any other field. Multiple indexes may be used independently, as in this query:

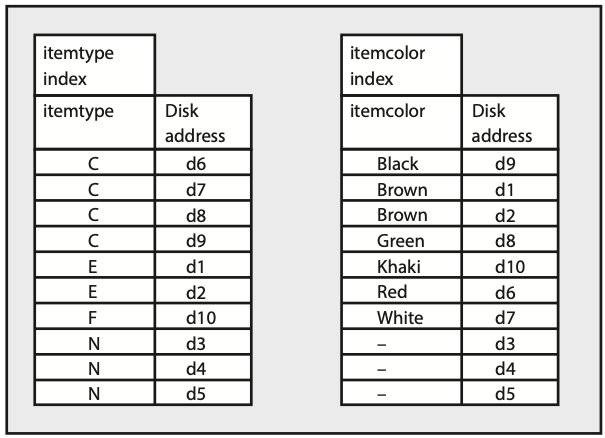
* List red items.

or jointly, with a query such as

* Find red items of type C.

The preceding query can be solved by using the indexes for itemtype and itemcolor.

Indexes for fields itemtype and itemcolor



Examination of the itemtype index indicates that items of type C are stored at addresses d6, d7, d8, and d9. The only red item recorded in the itemcolor index is stored at address d6, and since it is the only record satisfying the query, it is the only record that needs to be retrieved.

Multiple indexes, as you would expect, involve a trade-off. Whenever a record is added or updated, each index must also be updated. Savings in disk accesses for retrieval are exchanged for additional disk accesses in maintenance. Again, you must consider the balance between retrieval and maintenance operations.

Indexes are not restricted to a single field. It is possible to specify an index that is a combination of several fields. For instance, if item type and color queries were very common, then an index based on the concatenation of both fields could be created. As a result, such queries could be answered with a search of a single index rather than scanning two indexes as in the preceding example. The SQL for creating the combined index is

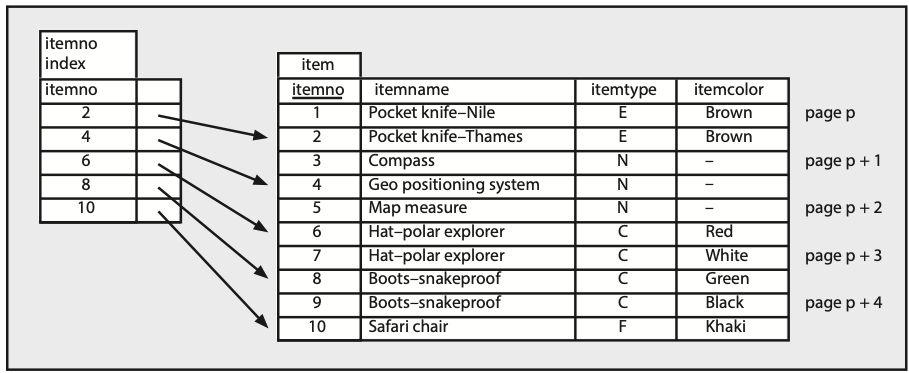
CREATE INDEX typecolorindx ON item (itemtype, itemcolor);

### Sparse indexes

Indexes are used to reduce disk accesses to accelerate retrieval. The simple model of an index introduced earlier suggests that the index contains an entry for each record of the file. If we can shrink the index to eliminate an entry for each record, we can save more disk accesses. Indeed, if an index is small enough, it, or key parts of it, can be retained continuously in primary memory.

There is a physical sequence to the records in a file. Records within a page are in a physical sequence, and pages on a disk are in a physical sequence. A file can also have a logical sequence, the ordering of the file on some field within a record. For instance, the item file could be ordered on itemno. Making the physical and logical sequences correspond is a way to save disk accesses. Remember that the item file was assumed to have a record size of 1,024 bytes, the same size as a page, and one record was stored per page. If we now assume the record size is 512 bytes, then two records are stored per page. Furthermore, suppose that item is physically stored in itemno sequence. The index can be compressed by storing itemno for the second record on each page and that page’s address.

A sparse index for item



Consider the process of finding the record with itemno = 7. First, the index is scanned to find the first value for itemno that is greater than or equal to 7, the entry for itemno = 8. Second, the page on which this record is stored (page p + 3) is loaded into memory. Third, the required record is extracted from the page.

Indexes that take advantage of the physical sequencing of a file are known as sparse or non-dense because they do not contain an entry for every value of the indexed field. (A dense index is one that contains an entry for every value of the indexed field.)

As you would expect, a sparse index has pros and cons. One major advantage is that it takes less storage space and so requires fewer disk accesses for reading. One disadvantage is that it can no longer be used for existence tests because it does not contain a value for every record in the file.

A file can have only one sparse index because it can have only one physical sequence. This field on which a sparse index is based is often called the primary key. Other indexes, which must be dense, are called secondary indexes.

In SQL, a sparse index is created using the CLUSTER option. For example, to define a sparse index on item the command is

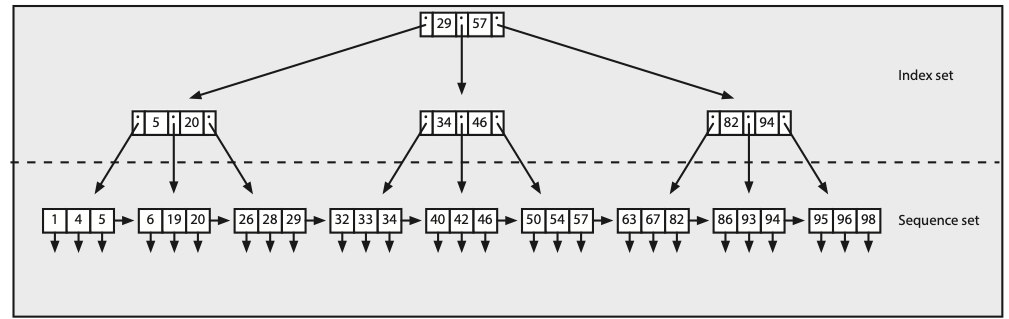
CREATE INDEX itemnoindx ON item (itemno) CLUSTER;

## B-trees

The B-tree is a particular form of index structure that is frequently the main storage structure for relational systems. It is also the basis for IBM’s VSAM (Virtual Storage Access Method), the file structure underlying DB2, IBM’s relational database. A B-tree is an efficient structure for both sequential and direct accessing of a file. It consists of two parts: the sequence set and the index set.

The sequence set is a single-level index to the file with pointers to the records (the vertical arrows in the lower part of the following figure). It can be sparse or dense, but is normally dense. Entries in the sequence set are grouped into pages, and these pages are linked together (the horizontal arrows) so that the logical ordering of the sequence set is the physical ordering of the file. Thus, the file can be processed sequentially by processing the records pointed to by the first page (records with identifiers 1, 4, and 5), the records pointed to by the next logical page (6, 19, and 20), and so on.

Structure of a simple B-tree



The index set is a tree-structured index to the sequence set. The top of the index set is a single node called the root. In this example, it contains two values (29 and 57) and three pointers. Records with an identifier less than or equal to 29 are found in the left branch, records with an identifier greater than 29 and less than or equal to 57 are found in the middle branch, and records with an identifier greater than 57 are found in the right branch. The three pointers are the page numbers of the left, middle, and right branches. The nodes at the next level have similar interpretations. The pointer to a particular record is found by moving down the tree until a node entry points to a value in the sequence set; this value can be used to retrieve the record. Thus, the index set provides direct access to the sequence set and then the data.

The index set is a B-tree. The combination of index set and sequence set is generally known as a B+ tree (B-plus tree). The B-tree simplifies the concept in two ways. First***,*** the number of data values and pointers for any given node is not restricted to 2 and 3, respectively. In its general form, a B-tree of order n can have at least n and no more than 2n data values. If it has k values, the B-tree will have k+ 1 pointers (in the example tree, nodes have two data values, k= 2, and there are three, k+ 1 = 3, pointers). Second, B-trees typically have free space to permit rapid insertion of data values and possible updating of pointers when a new record is added.

As usual, there is a trade-off with B-trees. Retrieval will be fastest when each node occupies one page and is packed with data values and pointers. This lack of free space will slow down the addition of new records, however. Most implementations of the B+ tree permit a specified portion of free space to be defined for both the index and sequence set.

## Hashing

Hashing reduces disk accesses by allowing direct access to a file. As you know, direct accessing via an index requires at least two or more accesses. The index must be loaded and searched, and then the record retrieved. For some applications, direct accessing via an index is too slow. Hashing can reduce the number of accesses to almost one by using the value in some field (the hash field, which is usually the primary key) to compute a record’s address. A hash function converts the hash field into a hash address.

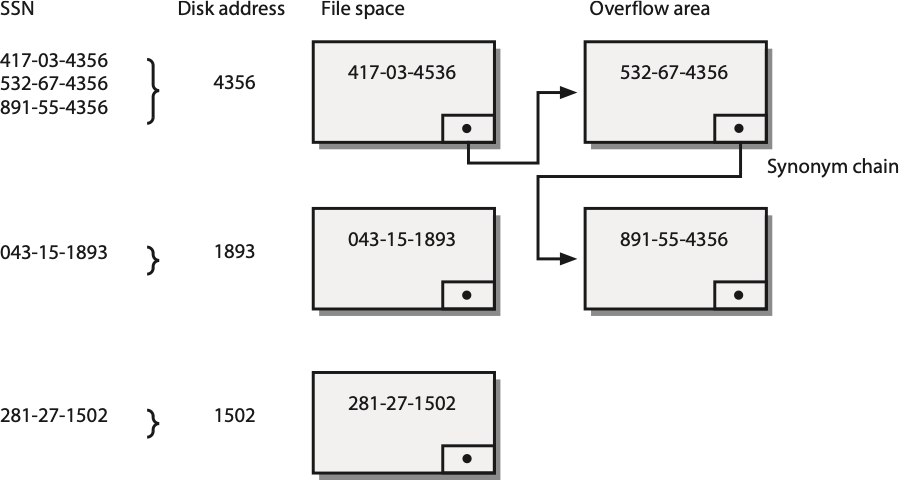
Consider the case of a university that uses the nine-digit Social Security number (SSN) as the student key. If the university has 10,000 students, it could simply use the last four digits of the SSN as the address. In effect, the file space is broken up into 10,000 slots with one student record in each slot. For example, the data for the student with SSN 417-03-4356 would be stored at address 4356. In this case, the hash field is SSN, and the hash function is

hash address = remainder after dividing SSN by 10,000.

What about the student with SSN 532-67-4356? Unfortunately, the hashing function will give the same address because most hashing schemes cannot guarantee a unique hash address for every record. When two hash fields have the same address, they are called synonyms, and a collision is said to have occurred.

There are techniques for handling synonyms. Essentially, you store the colliding record in an overflow area and point to it from the hash address. Of course, more than two records can have the same hash address, which in turn creates a synonym chain. The following figure shows an example of hashing with a synonym chain. Three SSNs hash to the same address. The first record (417-03-4356) is stored at the hash address. The second record (532-67-4356) is stored in the overflow area and is connected by a pointer to the first record. The third record (891-55-4356) is also stored in the overflow area and connected by a pointer from the second record. Because each record contains the full key (SSN in this case), during retrieval the system can determine whether it has the correct record or should follow the chain to the next record.

An example of hashing



If there are no synonyms, hashing gives very fast direct retrieval, taking only one disk access to retrieve a record. Even with a small percentage of synonyms, retrieval via hashing is very fast. Access time degrades, however, if there are long synonym chains.

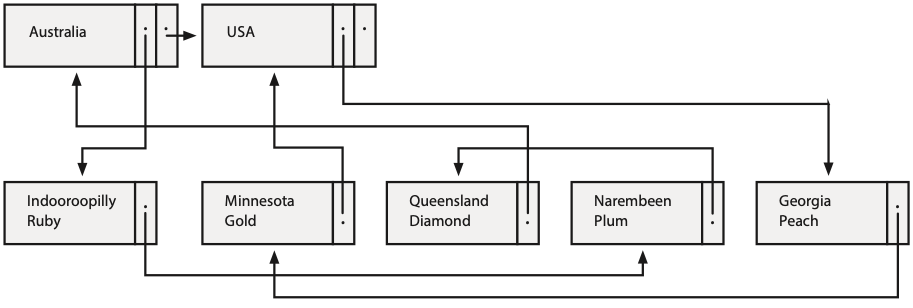
There are a number of different approaches to defining hashing functions. The most common method is to divide by a prime and use the remainder as the address. Before adopting a particular hashing function, test several functions on a representative sample of the hash field. Compute the percentage of synonyms and the length of synonym chains for each potential hashing function and compare the results.

Of course, hashing has trade-offs. There can be only one hashing field. In contrast, a file can have many indexed fields. The file can no longer be processed sequentially because its physical sequence loses any logical meaning if the records are not in primary key sequence or are sequenced on any other field.

## Linked lists

A linked list is a useful data structure for interfile clustering. Suppose that the query, Find all stocks of country X***,*** is a frequent request. Disk accesses can be reduced by storing a nation and its corresponding stocks together in a linked list.

A linked list



In this example, we have two files: nation and stock. Records in nation are connected by pointers (e.g., the horizontal arrow between Australia and USA). The pointers are used to maintain the nation file in logical sequence (by nation, in this case). Each record in nation is linked to its stock records by a forward-pointing chain. The nation record for Australia points to the first stock record (Indooroopilly Ruby), which points to the next (Narembeen Plum), which points to the final record in the chain (Queensland Diamond). Notice that the last record in this stock chain points to the nation record to which the chain belongs (i.e., Australia). Similarly, there is a chain for the two USA stocks. In any chain, the records are maintained in logical sequence (by firm name, in this case).

This linked-list structure is also known as a parent/child structure. (The parent in this case is nation and the child is stock.) Although it is a suitable structure for representing a one-to-many (1:m) relationship, it is possible to depict more than one parent/child relationship. For example, stocks could be grouped into classes (e.g., high, medium, and low risk). A second chain linking all stocks of the same risk class can run through the file. Interfile clustering of a nation and its corresponding stocks will speed up access; so, the record for the parent Australia and its three children should be stored on one page. Similarly, all American stocks could be clustered with the USA record and so on for all other nations.

Of course, you expect some trade-offs. What happens with a query such as Find the country in which Minnesota Gold is listed? There is no quick way to find Minnesota Gold except by sequentially searching the stock file until the record is found and then following the chain to the parent nation record. This could take many disk accesses. One way to circumvent this is to build an index or hashing scheme for stock so that any record can be found directly. Building an index for the parent, nation, will speed up access as well.

Linked lists come in a variety of flavors:

* Lists that have two-way pointers, both forward and backward, speed up deletion.
* For some lists, every child record has a parent pointer. This helps prevent chain traversal when the query is concerned with finding the parent of a particular child.

## Bitmap index

A bitmap index uses a single bit, rather than multiple bytes, to indicate the specific value of a field. For example, instead of using three bytes to represent red as an item’s color, the color red is represented by a single bit. The relative position of the bit within a string of bits is then mapped to a record address.

Conceptually, you can think of a bitmap as a matrix. The following figure shows a bitmap containing details of an item’s color and code. An item can have three possible colors; so, three bits are required, and two bits are needed for the two codes for the item. Thus, you can see that, in general, n bits are required if a field can have n possible values.

A bitmap index

| itemcode | color | | | code | | Disk address |
| --- | --- | --- | --- | --- | --- | --- |
|  | Red | Green | Blue | A | N |  |
| 1001 | 0 | 0 | 1 | 0 | 1 | d1 |
| 1002 | 1 | 0 | 0 | 1 | 0 | d2 |
| 1003 | 1 | 0 | 0 | 1 | 0 | d3 |
| 1004 | 0 | 1 | 0 | 1 | 0 | d4 |

When an item has a large number of values (i.e., n is large), the bitmap for that field will be very sparse, containing a large number of zeros. Thus, a bitmap is typically useful when the value of n is relatively small. When isn small or large? There is no simple answer; rather, database designers have to simulate alternative designs and evaluate the trade-offs.

In some situations, the bit string for a field can have multiple bits set to on (set to 1). A location field, for instance, may have two bits on to represent an item that can be found in Atlanta and New York.

The advantage of a bitmap is that it usually requires little storage. For example, we can recast the bitmap index as a conventional index. The core of the bitmap in the previous figure (i.e., the cells storing data about the color and code) requires 5 bits for each record, but the core of the traditional index requires 9 bytes, or 72 bits, for each record.

An index

| itemcode | color  char(8) | code  char(1) | Disk address |
| --- | --- | --- | --- |
| 1001 | Blue | N | d1 |
| 1002 | Red | A | d2 |
| 1003 | Red | A | d3 |
| 1004 | Green | A | d4 |

Bitmaps can accelerate query time for complex queries.

## Join index

Many RDBMS queries frequently require two or more tables to be joined. Indeed, some people refer to the RDBMS as a join machine. A join index can be used to improve the execution speed of joins by creating indexes based on the matching columns of tables that are highly likely to be joined. For example, natcode is the common column used to join nation and stock, and each of these tables can be indexed on natcode.

Indexes on natcode for nation and stock

| nation index |  |  | stock index |  |
| --- | --- | --- | --- | --- |
| natcode | Disk address |  | natcode | Disk address |
| UK | d1 |  | UK | d101 |
| USA | d2 |  | UK | d102 |
|  |  |  | UK | d103 |
|  |  |  | USA | d104 |
|  |  |  | USA | d105 |

A join index is a list of the disk addresses of the rows for matching columns. All indexes, including the join index, must be updated whenever a row is inserted into or deleted from the nation or stock tables. When a join of the two tables on the matching column is made, the join index is used to retrieve only those records that will be joined. This example demonstrates the advantage of a join index. If you think of a join as a product with a WHERE clause, then without a join index, 10 (2\*5) rows have to be retrieved, but with the join index only 5 rows are retrieved. Join indexes can also be created for joins involving several tables. As usual, there is a trade-off. Joins will be faster, but insertions and deletions will be slower because of the need to update the indexes.

Join Index

| join index |  |
| --- | --- |
| nation disk address | stock disk address |
| d1 | d101 |
| d1 | d102 |
| d1 | d103 |
| d2 | d104 |
| d2 | d105 |

# Data coding standards

The successful exchange of data between two computers requires agreement on how information is coded. The American Standard Code for Information Interchange (ASCII) and Unicode are two widely used data coding standards.

## ASCII

ASCII is the most common format for digital text files. In an ASCII file, each alphabetic, numeric, or special character is represented by a 7-bit code. Thus, 128 (27) possible characters are defined. Because most computers process data in eight-bit bytes or multiples thereof, an ASCII code usually occupies one byte.

## Unicode

Unicode, officially the Unicode Worldwide Character Standard, is a system for the interchange, processing, and display of the written texts of the diverse languages of the modern world. Unicode provides a unique binary code for every character, no matter what the platform, program, or language. The Unicode standard currently contains 34,168 distinct coded characters derived from 24 supported language scripts. These characters cover the principal written languages of the world.

Unicode provides for two encoding forms: a default 16-bit form, and a byte (8-bit) form called UTF-8 that has been designed for ease of use with existing ASCII-based systems. As the default encoding of HTML and XML, Unicode is required for Internet protocols. It is implemented in all modern operating systems and computer languages such as Java. Unicode is the basis of software that must function globally.

# Data storage devices

Many corporations double the amount of data they need to store every two to three years. Thus, the selection of data storage devices is a key consideration for data managers. When evaluating data storage options, data managers need to consider possible uses, which include:

1. Online data
2. Backup files
3. Archival storage

Many systems require data to be online—continually available. Here the prime concerns are usually access speed and capacity, because many firms require rapid response to large volumes of data. Backup files are required to provide security against data loss. Ideally, backup storage is high volume capacity at low cost. Archived data may need to be stored for many years; so the archival medium should be highly reliable, with no data decay over extended periods, and low-cost.

In deciding what data will be stored where, database designers need to consider a number of variables:

1. Volume of data
2. Volatility of data
3. Required speed of access to data
4. Cost of data storage
5. Reliability of the data storage medium
6. Legal standing of stored data

The design options are discussed and considered in terms of the variables just identified. First, we review some of the options.

## Magnetic technology

Over USD 20 billion is spent annually on magnetic storage devices.[[1]](#footnote-1) A significant proportion of many units IS hardware budgets is consumed by magnetic storage. Magnetic technology, the backbone of data storage for five decades, is based on magnetization and demagnetization of spots on a magnetic recording surface. The same spot can be magnetized and demagnetized repeatedly. Magnetic recording materials may be coated on rigid platters (hard disks), thin ribbons of material (magnetic tapes), or rectangular sheets (magnetic cards).

The main advantages of magnetic technology are its relative maturity, widespread use, and declining costs. A major disadvantage is susceptibility to strong magnetic fields, which can corrupt data stored on a disk. Another shortcoming is data storage life; magnetization decays with time.

As organizations continue to convert paper to images, they need a very long-term, unalterable storage medium for documents that could be demanded in legal proceedings (e.g., a customer’s handwritten insurance claim or a client’s completed form for a mutual fund investment). Because data resident on magnetic media can be readily changed and decay with time, magnetic storage is not an ideal medium for storing archival data or legal documents.

### Fixed magnetic disk

A fixed magnetic disk, also known as a hard disk drive (HDD), containing one or more recording surfaces is permanently mounted in the disk drive and cannot be removed. The recording surfaces and access mechanism are assembled in a clean room and then sealed to eliminate contaminants, making the device more reliable and permitting higher recording density and transfer rates. Access time is typically between 4 and 10 ms, and transfer rates can be as high as 1,300 Mbytes per second. Disk unit capacities range from gigabytes to terabytes. Magnetic disk units are sometimes called direct-access storage devices or DASD (pronounced dasdee).

Fixed disk is the medium of choice for most systems, from personal computers to supercomputers. It gives rapid, direct access to large volumes of data and is ideal for highly volatile files. The major disadvantage of magnetic disk is the possibility of a head crash, which destroys the disk surface and data. With the read/write head of a disk just 15 millionths of an inch (40 millionths of a centimeter) above the surface of the disk, there is little margin for error. Hence, it is crucial to make backup copies of fixed disk files regularly.

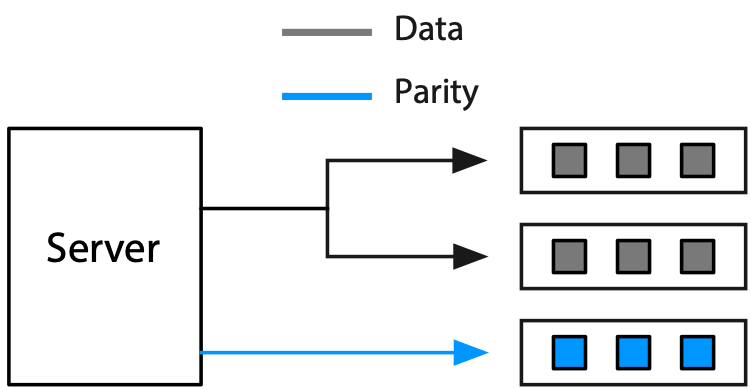
### RAID

RAID (redundant array of independent, or inexpensive, disks) takes advantage of the economies of scale in manufacturing disks for the personal computing market. The cost of drives increases with their capacity and speed. RAIDs use several cheaper drives whose total cost is less than one high-capacity drive but have the same capacity. In addition to lower cost, RAID offers greater data security. All RAID levels (except level 0) can reconstruct the data on any single disk from the data stored on the remaining disks in the array in a manner that is quite transparent to the user.

RAID uses a combination of mirroring and striping to provide greater data protection. When a file is written to a mirrored array, the disk controller writes identical copies of each record to each drive in the array. When a file is read from a mirrored array, the controller reads alternate pages simultaneously from each of the drives. It then puts these pages together in the correct sequence before delivering them to the computer. Mirroring reduces data access time by approximately the number of drives in the array because it interleaves the reading of records. During the time a conventional disk drive takes to read one page, a RAID system can read two or more pages (one from each drive). It is simply a case of moving from sequential to parallel retrieval of pages. Access times are halved for a two-drive array, quartered for a four-drive array, and so on.

If a read error occurs on a particular disk, the controller can always read the required page from another drive in the array because each drive has a full copy of the file. Mirroring, which requires at least two drives, improves response time and data security; however, it does take considerably more space to store a file because multiple copies are created.

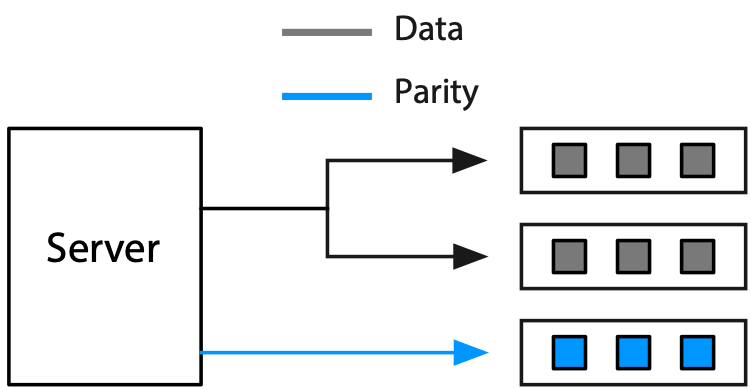
Mirroring



When a file is written to a striping array of three drives, for instance, one half of the file is written to the first drive and the second half to the second drive. The third drive is used for error correction. A parity bit is constructed for each corresponding pair of bits written to drives one and two, and this parity bit is written to the third drive. A parity bit is a bit added to a chunk of data (e.g., a byte) to ensure that the number of bits in the chunk with a value of one is even or odd. Parity bits can be checked when a file is read to detect data errors, and in many cases the data errors can be corrected. Parity bits demonstrate how redundancy supports recovery.

When a file is read by a striping array, portions are retrieved from each drive and assembled in the correct sequence by the controller. If a read error occurs, the lost bits can be reconstructed by using the parity data on the third drive. If a drive fails, it can be replaced and the missing data restored on the new drive. Striping requires at least three drives. Normally, data are written to every drive but one, and that remaining drive is used for the parity bit. Striping gives added data security without requiring considerably more storage, but it does not have the same response time increase as mirroring.

Striping



RAID subsystems are divided into seven levels, labeled 0 through 6. All RAID levels, except level 0, have common features:

* There is a set of physical disk drives viewed by the operating system as a single, logical drive.
* Data are distributed across corresponding physical drives.
* Parity bits are used to recover data in the event of a disk failure.

Level 0 has been in use for many years. Data are broken into blocks that are interleaved or striped across disks. By spreading data over multiple drives, read and write operations can occur in parallel. As a result, I/O rates are higher, which makes level 0 ideal for I/O intensive applications such as recording video. There is no additional parity information and thus no data recovery when a drive failure occurs.

Level 1 implements mirroring, as described previously. This is possibly the most popular form of RAID because of its effectiveness for critical nonstop applications, although high levels of data availability and I/O rates are counteracted by higher storage costs. Another disadvantage is that every write command must be executed twice (assuming a two-drive array), and thus level 1 is inappropriate for applications that have a high ratio of writes to reads.

Level 2 implements striping by interleaving blocks of data on each disk and maintaining parity on the check disk. This is a poor choice when an application has frequent, short random disk accesses, because every disk in the array is accessed for each read operation. However, RAID 2 provides excellent data transfer rates for large sequential data requests. It is therefore suitable for computer-aided drafting and computer-aided manufacturing (CAD/CAM) and multimedia applications, which typically use large sequential files. RAID 2 is rarely used, however, because the same effect can be achieved with level 3 at a lower cost.

Level 3 utilizes striping at the bit or byte level, so only one I/O operation can be executed at a time. Compared to level 1, RAID level 3 gives lower-cost data storage at lower I/O rates and tends to be most useful for the storage of large amounts of data, common with CAD/CAM and imaging applications.

Level 4 uses sector-level striping; thus, only a single disk needs to be accessed for a read request. Write requests are slower, however, because there is only one parity drive.

Level 5, a variation of striping, reads and writes data to separate disks independently and permits simultaneous reading and writing of data. Data and parity are written on the same drive. Spreading parity data evenly across several drives avoids the bottleneck that can occur when there is only one parity drive. RAID 5 is well designed for the high I/O rates required by transaction processing systems and servers, particularly e-mail servers. It is the most balanced implementation of the RAID concept in terms of price, reliability, and performance. It requires less capacity than mirroring with level 1 and higher I/O rates than striping with level 3, although performance can decrease with update-intensive operations. RAID 5 is frequently found in local-area network (LAN) environments.

RAID level 5



RAID 6 takes fault tolerance to another level by enabling a RAID system to still function when two drives fail. Using a method called double parity, it distributes parity information across multiple drives so that, on the fly, a disk array can be rebuild even if another drive fails before the rebuild is complete.

RAID does have drawbacks. It often lowers a system’s performance in return for greater reliability and increased protection against data loss. Extra disk space is required to store parity information. Extra disk accesses are required to access and update parity information. You should remember that RAID is not a replacement for standard backup procedures; it is a technology for increasing the fault tolerance of critical online systems. RAID systems offer terabytes of storage.

### Removable magnetic disk

Removable drives can be plugged into a system as required. The disk’s removability is its primary advantage, making it ideal for backup and transport. For example, you might plug in a removable drive to the USB port of your laptop once a day to backup its hard drive. Removable disk is also useful when applications need not be continuously online. For instance, the monthly payroll system can be stored on a removable drive and mounted as required. Removable drives cost about USD 100 per terabyte.

### Magnetic tape

A magnetic tape is a thin ribbon of plastic coated with ferric oxide. The once commonly used nine-track 2,400-foot (730 m) tape has a capacity of about 160 Mbytes and a data transfer rate of 2 Mbytes per second. The designation nine-track means nine bits are stored across the tape (8 bits plus one parity bit). Magnetic tape was used extensively for archiving and backup in early database systems; however, its limited capacity and sequential nature have resulted in its replacement by other media, such as magnetic tape cartridge.

### Magnetic tape cartridges

Tape cartridges, with a capacity measured in Gbytes and transfer rates of up to 6 Mbytes per second, have replaced magnetic tape. They are the most cost effective (from a purchase price $/GB perspective) magnetic recording device. Furthermore, a tape cartridge does not consume any power when unmounted and stored in a library. Thus, the operating costs for a tape storage archival system are much lower than an equivalent hard disk storage system, though hard disks provide faster access to archived data.

### Mass storage

There exists a variety of mass storage devices that automate labor-intensive tape and cartridge handling. The storage medium, with a capacity of terabytes, is typically located and mounted by a robotic arm. Mass storage devices can currently handle hundreds of petabytes of data. They have slow access time because of the mechanical loading of tape storage, and it might take several minutes to load a file.

## Solid-state memory

A solid-state disk (SSD) connects to a computer in the same way as regular magnetic disk drives do, but they store data on arrays of memory chips. SSDs require lower power consumption (longer battery life) and come in smaller sizes (smaller and lighter devices). As well, SSDs have faster access times. However, they cost around six times as much as equivalent size HDDs, but they declining in prices rapidly, and might be as little as twice as costly by 2021.[[2]](#footnote-2) SSD is rapidly becoming the standard for laptops.

A flash drive, also known as a keydrive or jump drive, is a small, removable storage device with a Universal Serial Bus (USB) connector. It contains solid-state memory and is useful for transporting small amounts of data. Capacity is in the range 1 to 128 Gbytes. The price for low capacity drives is about about USD 1-2 per Gbytes.

While price is a factor, the current major drawback holding back the adoption of SSD is the lack of manufacturing capacity for the NAND chips used in SSDs.[[3]](#footnote-3) The introduction of tablets, such as the iPad, and smartphones is exacerbating this problem.

## Optical technology

Optical technology is a more recent development than magnetic. Its advantages are high-storage densities, low-cost media, and direct access. Optical storage systems work by reflecting beams of laser light off a rotating disk with a minutely pitted surface. As the disk rotates, the amount of light reflected back to a sensor varies, generating a stream of ones and zeros. A tiny change in the wavelength of the laser translates into as much as a tenfold increase in the amount of information that can be stored. Optical technology is highly reliable because it is not susceptible to head crashes.

There are three storage media based on optical technology: CD-ROM, DVD, and Blu-ray. Most optical disks can reliably store records for at least 10 years under prescribed conditions of humidity and temperature. Actual storage life may be in the region of 30 to 50 years. Optical media are compact medium for the storage of permanent data. Measuring 12 cm (~ 4.75 inches) in diameter, the size consistency often means later generation optical readers can read earlier generation media.

Optical technology

| Medium | Introduced | Capacity |
| --- | --- | --- |
| Compact disc (CD) | 1982 | .65 Gbytes |
| Digital versatile disc (DVD) | 1995 | 1.5-17 Gbytes |
| Blu-ray disc (BD) | 2006 | 25-50 Gbytes |

Optical media come in three types. Read-only memory (ROM) is typically used for the large scale manufacturing of information to be distributed, such as a music album or movie, and it will also usually incorporate some form of digital rights management to prevent illegal copying. Recordable (R) for write-once media can be purchased for the storing of information that should not be alterable. Re-recordable media are re-usable, and are a general backup and distribution medium.

Optical media options

| Format | Description |
| --- | --- |
| ROM | Read-only |
| R | Recordable or write-once |
| RW | Read/write or re-recordable |

Optical media are commonly used to distribute various types of information. DVDs and Blu-ray have replaced CDs to distribute documents, music, and software, though the use of optical media for information distribution is declining because it is cheaper and faster to transmit via the Internet. Services such as Apple’s App store and Netflix are pushing the distribution of software and movies, respectively, to the Internet.

Data stored on read-only or write-once optical media are generally reckoned to have the highest legal standing of any of the forms discussed because, once written, the data cannot be altered. Thus, they are ideal for storage of documents that potentially may be used in court. Consequently, we are likely to see archival storage emerge as the major use for optical technology.

## Storage-area networks

A storage-area network (SAN) is a high-speed network for connecting and sharing different kinds of storage devices, such as tape libraries and disk arrays. In the typical LAN, the storage device (usually a disk) is closely coupled to a server and communicates through a bus connection. Communication among storage devices occurs through servers and over the LAN, which can lead to network congestion.

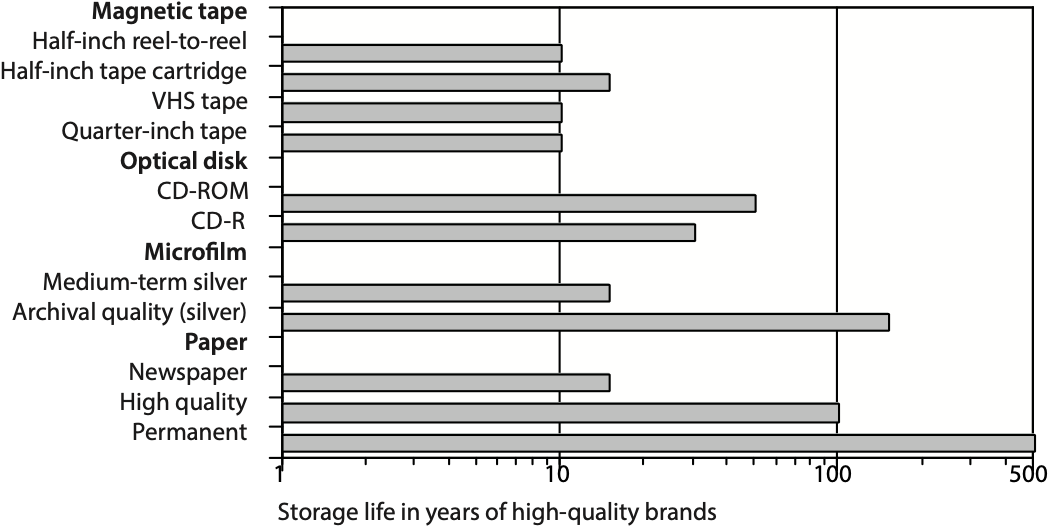
SANs support disk mirroring, backup and restore, archival and retrieval of archived data, data migration from one storage device to another, and the sharing of data among different servers in a network. Typically, a SAN is part of the overall network of computing resources for an enterprise. SANs are likely to be a critical element for supporting e-commerce and applications requiring large volumes of data. As Storage Area Networks (SANs) have become increasingly affordable, their use in enterprises and even smaller businesses has become widespread.

## Long-term storage

Long-term data storage has always been of concern to societies and organizations. Some data, such as the location of toxic-waste sites, must be stored for thousands of years. Increasingly, governments are converting their records to electronic format. Unlike paper, magnetic media do not show degradation until it is too late to recover the data. Magnetic tapes can become so brittle that the magnetic coating separates from the backing. In addition, computer hardware and software rapidly become obsolete. The medium may be readable, but there could be no hardware to read it and no software to decode it.

Paper, it seems, is still the best medium for long-term storage, and research is being conducted to create extra-long-life paper that can store information for hundreds of years. This paper, resistant to damage from heat, cold, and magnetism, will store data in a highly compact format but, obviously, nowhere near optical disk densities.

The life expectancy of various media at 20°C (68°F) and 40 percent relative humidity (source: National Media Lab)



# Data compression

Data compression is a method for encoding digital data so that they require less storage space and thus less communication bandwidth. There are two basic types of compression: lossless methods, in which no data are lost when the files are restored to their original format, and lossy methods, in which some data are lost when the files are decompressed.

## Lossless compression

During lossless compression, the data-compression software searches for redundant or repetitive data and encodes it. For example, a string of 100 asterisks (\*) can be stored more compactly by a compression system that records the repeated character (i.e., \*) and the length of the repetition (i.e., 100). The same principle can be applied to a photo that contains a string of horizontal pixels of the same color. Clearly, you want to use lossless compression with text files (e.g., a business report or spreadsheet).

## Lossy compression

Lossy compression is used for graphics, video, and audio files because humans are often unable to detect minor data losses in these formats. Audio files can often be compressed to 10 percent of their original size (e.g., an MP3 version of a CD recording).

Skill builder

An IS department in a major public university records the lectures for 10 of its classes for video streaming to its partner universities in India and China. Each twice-weekly lecture runs for 1.25 hours and a semester is 15 weeks long. A video streaming expert estimates that one minute of digital video requires 6.5 Mbytes using MPEG-4 and Apple’s QuickTime software. What is MPEG-4? Calculate how much storage space will be required and recommend a storage device for the department.

# Comparative analysis

Details of the various storage devices are summarized in the following table. A simple three-star rating system has been used for each device—the more stars, the better. In regard to access speed, RAID gets more stars than DVD-RAM because it retrieves a stored record more quickly. Similarly, optical rates three stars because it costs less per megabyte to store data on a magneto-optical disk than on a fixed disk. The scoring system is relative. The fact that removable disk gets two stars for reliability does not mean it is an unreliable storage medium; it simply means that it is not as reliable as some other media.

Relative merits of data storage devices

| Device | Access speed | Volume | Volatility | Cost per megabyte | Reliability | Legal standing |
| --- | --- | --- | --- | --- | --- | --- |
| Solid state | \*\*\* | \* | \*\*\* | \* | \*\*\* | \* |
| Fixed disk | \*\*\* | \*\*\* | \*\*\* | \*\*\* | \*\* | \* |
| RAID | \*\*\* | \*\*\* | \*\*\* | \*\*\* | \*\*\* | \* |
| Removable disk | \*\* | \*\* | \*\*\* | \*\* | \*\* | \* |
| Flash memory | \*\* | \* | \*\*\* | \* | \*\*\* | \* |
| Tape | \* | \*\* | \* | \*\*\* | \*\* | \* |
| Cartridge | \*\* | \*\*\* | \* | \*\*\* | \*\* | \* |
| Mass Storage | \*\* | \*\*\* | \* | \*\*\* | \*\* | \* |
| SAN | \*\*\* | \*\*\* | \*\*\* | \*\*\* | \*\*\* | \* |
| Optical-ROM | \* | \*\*\* | \* | \*\*\* | \*\*\* | \*\*\* |
| Optical-R | \* | \*\*\* | \* | \*\*\* | \*\*\* | \*\* |
| Optical-RW | \* | \*\*\* | \*\* | \*\*\* | \*\*\* | \* |

Legend

| Characteristic | More stars mean … |
| --- | --- |
| Access speed | Faster access to data |
| Volume | Device more suitable for large files |
| Volatility | Device more suitable for files that change frequently |
| Cost per megabyte | Less costly form of storage |
| Reliability | Device less susceptible to an unrecoverable read error |
| Legal standing | Media more acceptable as evidence in court |

## Conclusion

The internal and physical aspects of database design are a key determinant of system performance. Selection of appropriate data structures can substantially curtail disk access and reduce response times. In making data structure decisions, the database administrator needs to weigh the pros and cons of each choice. Similarly, in selecting data storage devices, the designer needs to be aware of the trade-offs. Various devices and media have both strengths and weaknesses, and these need to be considered.

## Summary

The data deluge is increasing the importance of data management for organizations. It takes considerably longer to retrieve data from a hard disk than from main memory. Appropriate selection of data structures and data access methods can considerably reduce delays by reducing disk accesses. The key characteristics of disk storage devices that affect database access are rotational speed and access arm speed. Access arm movement can be minimized by storing frequently used data on the same track on a single surface or on the same track on different surfaces. Records that are frequently used together should be clustered together. Intrafile clustering applies to the records within a single file. Interfile clustering applies to multiple files. The disk manager, the part of the operating system responsible for physical I/O, maintains a directory of pages. The file manager, a level above the disk manager, contains a directory of files.

Indexes are used to speed up retrieval by reducing disk accesses. An index is a file containing the value of the index field and the address of its full record. The use of indexes involves a trade-off between faster retrievals and slower updates. Indexes can be used for both sequential and direct access. A file can have multiple indexes. A sparse index does not contain an entry for every value of the indexed field. The B-tree, a particular form of index structure, consists of two parts: the sequence set and the index set. Hashing is a technique for reducing disk accesses that allows direct access to a file. There can be only one hashing field. A hashed file can no longer be processed sequentially because its physical sequence has lost any logical meaning. A linked list is a useful data structure for interfile clustering. It is a suitable structure for representing a 1:m relationship. Pointers between records are used to maintain a logical sequence. Lists can have forward, backward, and parent pointers.

Systems designers have to decide what data storage devices will be used for online data, backup files, and archival storage. In making this decision, they must consider the volume of data, volatility of data, required speed of access to data, cost of data storage, reliability of the data storage medium, and the legal standing of the stored data. Magnetic technology, the backbone of data storage for six decades, is based on magnetization and demagnetization of spots on a magnetic recording surface. Fixed disk, removable disk, magnetic tape, tape cartridge, and mass storage are examples of magnetic technology. RAID uses several cheaper drives whose total cost is less than one high-capacity drive. RAID uses a combination of mirroring or striping to provide greater data protection. RAID subsystems are divided into six levels labeled 0 through 6. A storage-area network (SAN) is a high-speed network for connecting and sharing different kinds of storage devices, such as tape libraries and disk arrays.

Optical technology offers high storage densities, low-cost medium, and direct access. CD, DVD, and Blu-ray are examples of optical technology. Optical disks can reliably store records for at least 10 years and possibly up to 30 years. Optical technology is not susceptible to head crashes.

Data compression techniques reduce the need for storage capacity and bandwidth. Lossless methods result in no data loss, whereas with lossy techniques, some data are lost during compression.

## Key terms and concepts

Access time

Archival file

ASCII

B-tree

Backup file

Bitmap index

Blue-ray disc (BD)

Compact disc (CD)

Clustering

Conceptual schema

Cylinder

Data compression

Data deluge

Data storage device

Database architecture

Digital versatile disc (DVD)

Disk manager

External schema

File manager

Hash address

Hash field

Hash function

Hashing

Index

Index set

Interfile clustering

Internal schema

Intrafile clustering

Join index

Linked list

Lossless compression

Lossy compression

Magnetic disk

Magnetic tape

Mass storage

Mirroring

Page

Parity

Pointer

Redundant arrays of inexpensive or independent drives (RAID)

Sequence set

Solid-state disk (SSD)

Sparse index

Storage-area network (SAN)

Striping

Track

Unicode

VSAM

## Exercises

1. Why is a disk drive considered a bottleneck?
2. What is the difference between a record and a page?
3. Describe the two types of delay that can occur prior to reading a record from a disk. What can be done to reduce these delays?
4. What is clustering? What is the difference between intrafile and interfile clustering?
5. Describe the differences between a file manager and a disk manager.
6. What is an index?
7. What are the advantages and disadvantages of indexing?
8. Write the SQL to create an index on the column natcode in the nation table.
9. A Paris insurance firm keeps paper records of all policies and claims made on it. The firm now has a vault containing 100 filing cabinets full of forms. Because Paris rental costs are so high, the CEO has asked you to recommend a more compact medium for long-term storage of these documents. Because some insurance claims are contested, she is very concerned with ensuring that documents, once stored, cannot be altered. What would you recommend and why?
10. A video producer has asked for your advice on a data storage device. She has specified that she must be able to record video at 5 to 7 Mbytes per second. What would you recommend and why?
11. A German consumer research company collects scanning data from supermarkets throughout central Europe. The scanned data include product code identifier, price, quantity purchased, time, date, supermarket location, and supermarket name, and in some cases where the supermarket has a frequent buyer plan, it collects a consumer identification code. It has also created a table containing details of the manufacturer of each product. The database contains over 500 Tbytes of data. The data are used by market researchers in consumer product companies. A researcher will typically request access to a slice of the database (e.g., sales of all detergents) and analyze these data for trends and patterns. The consumer research company promises rapid access to its data. Its goal is to give clients access to requested data within one to two minutes. Once clients have access to the data, they expect very rapid response to queries. What data storage and retrieval strategy would you recommend?
12. A magazine subscription service has a Web site for customers to place orders, inquire about existing orders, or check subscription rates. All customers are uniquely identified by an 11-digit numeric code. All magazines are identified by a 2- to 4-character code. The company has approximately 10 million customers who subscribe to an average of four magazines. Subscriptions are available to 126 magazines. Draw a data model for this situation. Decide what data structures you would recommend for the storage of the data. The management of the company prides itself on its customer service and strives to answer customer queries as rapidly as possible.
13. A firm offers a satellite-based digital radio service to the continental U.S. market. It broadcasts music, sports, and talk-radio programs from a library of 1.5 million digital audio files, which are sent to a satellite uplink and then beamed to car radios equipped to accept the service. Consumers pay $9.95 per month to access 100 channels.
    1. Assuming the average size of a digital audio file is 5 Mbytes (~4 minutes of music), how much storage space is required?
    2. What storage technology would you recommend?
14. Is MP3 a lossless or lossy compression standard?
15. What is the data deluge? What are the implications for data management?
16. According to Wikipedia, Apple has more than 28 million songs in the iTunes store.[[4]](#footnote-4) What storage technology might be a good choice for this library?

21. Data Processing Architectures

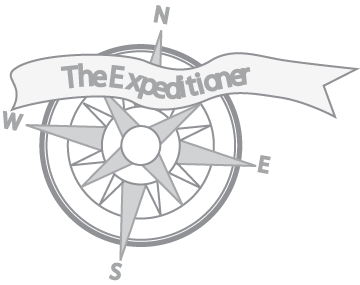
The difficulty in life is the choice.

George Moore, The Bending of the Bough, 1900

## Learning Objectives

Students completing this chapter will be able to

* recommend a data architecture for a given situation;
* understand multi-tier client/server architecture;
* discuss the fundamental principles that a hybrid architecture should satisfy;
* demonstrate the general principles of distributed database design.

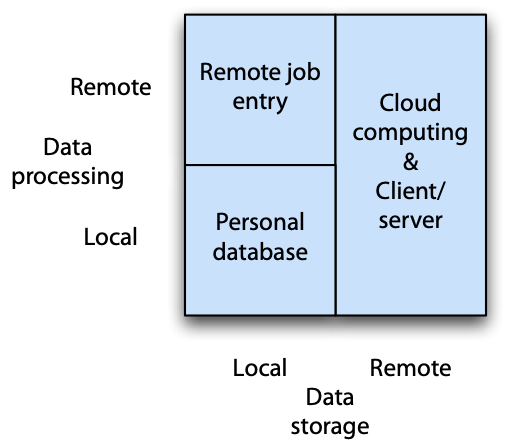
On a recent transatlantic flight, Sophie, the Personnel and PR manager, had been seated next to a very charming young man who had tried to impress her with his knowledge of computing. He worked for a large systems consulting firm, and his speech was sprinkled with words like “clouding computing,” “XML,” “server,” “client,” and “software as a service.” In return, Sophie responded with some “ums,” “aahs,” and an occasional “that’s interesting.” Of course, this was before the third glass of champagne. After that, he was more intent on finding out how long Sophie would be in New York and what restaurants and shows piqued her curiosity.

After an extremely pleasant week of wining, dining, and seeing the town—amid, of course, many hours of valuable work for The Expeditioner—Sophie returned to London. Now, she must ask Ned what all these weird terms meant. After all, it was difficult to carry on a conversation when she did not understand the language. That’s the problem with IS, she thought, new words and technologies were always being invented. It makes it very hard for the rest of us to make sense of what’s being said.

# Architectural choices

In general terms, data can be stored and processed locally or remotely. Combinations of these two options provide the basic information systems architectures.

Basic architectures



# Remote job entry

In remote job entry, data are stored locally and processed remotely. Data are sent over a network to a remote computer for processing, and the output is returned the same way. This once fairly common form of data processing is still used today because remote job entry can overcome speed or memory shortcomings of a personal computer. Scientists and engineers typically need occasional access to a supercomputer for applications, such as simulating global climate change, that are data or computational intensive. Supercomputers typically have the main memory and processing power to handle such problems.

Local storage is used for several reasons. First, it may be cheaper than storing on a supercomputer. Second, the analyst might be able to do some local processing and preparation of the data before sending them to the supercomputer. Supercomputing time can be expensive. Where possible, local processing is used for tasks such as data entry, validation, and reporting. Third, local data storage might be deemed more secure for particularly sensitive data.

# Personal database

People can store and process their data locally when they have their own computers. Many personal computer database systems (e.g., MS Access and FileMaker) permit people to develop their own applications, and many common desktop applications require local database facilities (e.g., a calendar).

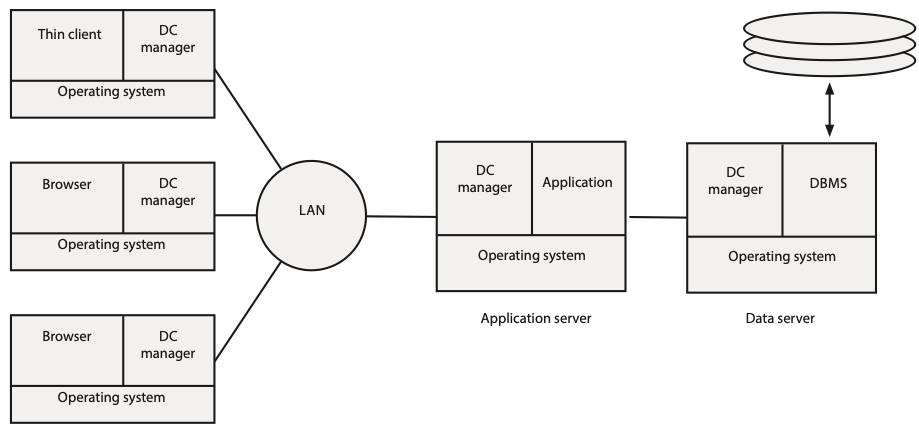
Of course, there is a downside to personal databases. First***,*** there is a great danger of repetition and redundancy. The same application might be developed in a slightly different way by many users. The same data get stored on many different systems. (It is not always the same, however, because data entry errors or maintenance inconsistencies result in discrepancies between personal databases.) Second, data are not readily shared because various users are not aware of what is available or find it inconvenient to share data. Personal databases are exactly that; but much of the data may be of corporate value and should be shared. Third, data integrity procedures are often quite lax for personal databases. People might not make regular backups, databases are often not secured from unauthorized access, and data validation procedures are often ignored. Fourth, often when the employee leaves the organization or moves to another role, the application and data are lost because they are not documented and the organization is unaware of their existence. Fifth, most personal databases are poorly designed and, in many cases, people use a spreadsheet as a poor substitute for a database. Personal databases are clearly very important for many organizations—when used appropriately. Data that are shared require a different processing architecture.

# Client/server

The client/server architecture, in which multiple computers interact in a superior and subordinate mode, is the dominant architecture these days. A client process initiates a request and a server responds. The client is the dominant partner because it initiates the request to which the server responds. Client and server processes can run on the same computer, but generally they run on separate, linked computers. In the three-tier client/server model, the application and database are on separate servers.

A generic client/server architecture consists of several key components. It usually includes a mix of operating systems, data communications managers (usually abbreviated to DC manager), applications, clients (e.g., browser), and database management systems. The DC manager handles the transfer of data between clients and servers.

Three-tier client/server computing



The three-tier model consists of three types of systems:

* Clients perform presentation functions, manage the graphical user interface (GUI), and execute communication software that provides network access. In most cases, the client is an Internet browser, though sometimes there might be a special program, a thin client, running on the client machine.
* Application servers are where the majority of the business and data logic are processed. They process commands sent by clients.
* Data servers run a DBMS. They respond to requests from an application, in which case the application is a client. They will also typically provide backup and recovery services and transaction management control.

Under the three-tier model, the client requests a service from an application server, and the application server requests data from a data server. The computing environment is a complex array of clients, application servers, and data servers. An organization can spread its processing load across many servers. This enables it to scale up the data processing workload more easily. For example, if a server running several applications is overloaded, another server can be purchased and some of the workload moved to the new server.

The client/server concept has evolved to describe a widely distributed, data-rich, cooperative environment. This is known as the n-tier client/server environment, which can consist of collections of servers dedicated to applications, data, transaction management, systems management, and other tasks. It extends the database side to incorporate nonrelational systems, such as multidimensional databases, multimedia databases, and legacy systems.

Evolution of client/server computing

| Architecture | Description |
| --- | --- |
| Two-tier | Processing is split between client and server, which also runs the DBMS. |
| Three-tier | Client does presentation, processing is done by the server, and the DBMS is on a separate server. |
| N-tier | Client does presentation. Processing and DBMS can be spread across multiple servers. This is a distributed resources environment. |

The rise of n-tier client/server can be attributed to the benefits of adopting a component-based architecture. The goal is to build quickly scalable systems by plugging together existing components. On the client side, the Web browser is a readily available component that makes deployment of new systems very simple. When everyone in the corporation has a browser installed on their personal computer, rolling out a new application is just a case of e-mailing the URL to the people concerned. Furthermore, the flexibility of the client/server model means that smartphones and tablets can easily be incorporated into the system. Thus, the client can be an app on an iPad.

On the data server side, many data management systems already exist, either in relational format or some other data model. Middle-tier server applications can make these data available to customers through a Web client. For example, UPS was able to make its parcel tracking system available via the Web, tablet, or smartphone because the database already existed. A middle-tier was written to connect customers using a range of devices to the database.

Skill builder

A European city plans to establish a fleet of two-person hybrid cars that can be rented for short periods (e.g., less than a day) for one-way trips. Potential renters first register online and then receive via the postal system a smart card that is used to gain entry to the car and pay automatically for its rental. The city also plans to have an online reservation system that renters can use to find the nearest car and reserve it. Reservations can be held for up to 30 minutes, after which time the car is released for use by other renters. Discuss the data processing architecture that you would recommend to support this venture. What technology would you need to install in each car? What technology would renters need? What features would the smart card require? Are there alternatives to a smart card?

# Cloud computing

With the development of client/server computing, many organizations created data centers to contain the host of servers they needed to meet their information processing requirements. These so-called server farms can run to tens of thousands of servers. Because of security concerns, many firms are unwilling to reveal the location and size of their server farms. Google is reputed to have hundreds of thousands of servers. Many corporations run much smaller, but still significantly costly, data centers. Do they really need to run these centers?

The goal of the corporate IS unit should be to create and deploy systems that meet the operational needs of the business or give it a competitive advantage. It can gain little advantage from managing a data center. As a result, some organizations have turned to cloud computing, which is the use of shared hardware and software, to meet their processing needs. Instead of storing data and running applications on the corporate data center, applications and data are shifted to a third party data center, the cloud, which is accessed via the Internet. Companies can select from among several cloud providers. For example, it might run office applications (word processing, spreadsheet, etc.) on the Google cloud, and customer relationship management on Amazon's cloud.

There are usually economies of scale from switching from an in-house data center to the massive shared resources of a cloud provider, which lowers the cost of information processing. As well, the IS unit can turn its attention to the systems needs of the organization, without the distraction of managing a data center.

Cloud computing vendors specialize in managing large-scale data centers. As a result, they can lower costs in some areas that might be infeasible for the typical corporation. For example, it is much cheaper to move photons through fiber optics than electrons across the electricity grid.[[5]](#footnote-5) This means that some cloud server farms are located where energy costs are low and minimal cooling is required. Iceland and parts of Canada are good locations for server farms because of inexpensive hydroelectric power and a cool climate. Information can be moved to and fro on a fiber optic network to be processed at a cloud site and then presented on the customer's computer.

Cloud computing can be more than a way to save some data center management costs. It has several features, or capabilities, that have strategic implications.[[6]](#footnote-6) We start by looking at the levels or layers of clouds.

Cloud layers

| Layer | Description | Example |
| --- | --- | --- |
| Infrastructure | A virtual server over which the developer has complete control | Amazon Elastic Compute Cloud (EC2) |
| Platform as a service | A developer can build a new application with the provided tools and programming language | Salesforce.com |
| Application | Access to cloud applications | Google docs |
| Collaboration | A special case of an application cloud | Facebook |
| Service | Consulting and integration services | Appirio |

As the preceding table illustrates, there a several options for the cloud customer. Someone looking to replace an office package installed on each personal computer could select an offering in the Application layer. A CIO looking to have complete control over a new application being developed from scratch could look to the infrastructure cloud, which also has the potential for high scalability.

Ownership is another way of looking at cloud offerings.[[7]](#footnote-7)

Cloud ownership

| Type | Description |
| --- | --- |
| Public | A cloud provided to the general public by its owner |
| Private | A cloud restricted to an organization. It has many of the same capabilities as a public cloud but provides greater control. |
| Community | A cloud shared by several organizations to support a community project |
| Hybrid | Multiple distinct clouds sharing a standardized or proprietary technology that enables data and application portability |

## Cloud capabilities

Understanding the strategic implications of cloud computing is of more interest than the layers and types because a cloud’s seven capabilities offer opportunities to address the five strategic challenges facing all organizations.

### Interface control

Some cloud vendors provide their customers with an opportunity to shape the form of interaction with the cloud. Under open co-innovation, a cloud vendor has a very open and community-driven approach to services. Customers and the vendor work together to determine the roadmap for future services. Amazon follows this approach. The qualified retail model requires developers to acquire permission and be certified before releasing any new application on the vendor's platform. This is the model Apple uses with its iTunes application store. Qualified co-innovation involves the community in the creation and maintenance of application programming interfaces (APIs). Third parties using the APIs, however, must be certified before they can be listed in the vendor's application store. This is the model salesforce.com uses. Finally, we have the open retail model. Application developers have some influence on APIs and new features, and they are completely free to write any program on top of the system. This model is favored by Microsoft.

### Location independence

This capability means that data and applications can be accessed, with appropriate permission, without needing to know the location of the resources. This capability is particularly useful for serving customers and employees across a wide variety of regions. There are some challenges to location independence. Some countries restrict where data about their citizens can be stored and who can access it.

### Ubiquitous access

Ubiquity means that customers and employees can access any information service from any platform or device via a Web browser from anywhere. Some clouds are not accessible in all parts of the globe at this stage because of the lack of connectivity, sufficient bandwidth, or local restrictions on cloud computing services.

### Sourcing independence

One of the attractions of cloud computing is that computing processing power becomes a utility. As a result, a company could change cloud vendors easily at low cost. It is a goal rather than a reality at this stage of cloud development.

### Virtual business environments

Under cloud computing, some special needs systems can be built quickly and later abandoned. For example, in 2009 the U.S. Government had a Cash for Clunkers program that ran for a few months to encourage people to trade-in their old car for a more fuel-efficient new car. This short-lived system whose processing needs are hard to estimate is well suited to a cloud environment. No resources need to be acquired, and processing power can be obtained as required.

### Addressability and traceability

Traceability enables a company to track customers and employees who use an information service. Depending on the device accessing an information service, a company might be able to precisely identify the person and the location of the request. This is particularly the case with smartphones or tablets that have a GPS capability.

### Rapid elasticity

Organizations cannot always judge exactly their processing needs, especially for new systems, such as the previously mentioned Cash for Clunkers program. Ideally, an organization should be able to pull on a pool of computer processing resources that it can scale up and down as required. It is often easier to scale up than down, as this is in the interests of the cloud vendor.

## Strategic risk

Every firm faces five strategic risks: demand, innovation, inefficiency, scaling, and control risk.

| Risk | Description |
| --- | --- |
| Demand | Fluctuating demand or market collapse |
| Inefficiency | Inability to match competitors’ unit costs |
| Innovation | Not innovating as well as competitors |
| Scaling | Not scaling fast and efficiently enough to meet market growth |
| Control | Inadequate procedures for acquiring or managing resources |

Strategic risks

We now consider how the various capabilities of cloud computing are related to these risks which are summarized in a table following the discussion of each risk.

### Demand risk

Most companies are concerned about loss of demand, and make extensive use of marketing techniques (e.g., advertising) to maintain and grow demand. Cloud computing can boost demand by enabling customers to access a service wherever they might be. For example, Amazon combines ubiquitous access to its cloud and the Kindle, its book reader, to permit customers to download and start reading a book wherever they have access. Addressability and traceability enable a business to learn about customers and their needs. By mining the data collected by cloud applications, a firm can learn the connections between the when, where, and what of customers’ wants. In the case of excessive demand, the elasticity of cloud resources might help a company to serve higher than expected demand.

### Inefficiency risk

If a firm’s cost structure is higher than its competitors, its profit margin will be less, and it will have less money to invest in new products and services. Cloud computing offers several opportunities to reduce cost. First, location independence means it can use a cloud to provide many services to customers across a large geographic region through a single electronic service facility. Second, sourcing independence should result in a competitive market for processing power. Cloud providers compete in a way that a corporate data center does not, and competition typically drives down costs. Third, the ability to quickly create and test new applications with minimal investment in resources avoids many of the costs of procurement, installation, and the danger of obsolete hardware and software. Finally, ubiquitous access means employees can work from a wide variety of locations. For example, ubiquity enables some to work conveniently and productively from home or on the road, and thus the organization needs smaller and fewer buildings.

### Innovation risk

In most industries, enterprises need to continually introduce new products and services. Even in quite stable industries with simple products (e.g., coal mining), process innovation is often important for lowering costs and delivering high quality products. The ability to modify a cloud’s interface could be of relevance to innovation. Apple’s App store restrictions might push some to Android’s more open model. As mentioned in the previous section, the capability to put up and tear down a virtual business environment quickly favors innovation because the costs of experimenting are low. Ubiquitous access makes it easier to engage customers and employees in product improvement. Customers use a firm’s products every day, and some reinvent them or use them in ways the firm never anticipated. A firm that learns of these unintended uses might identify new markets and new features that will extend sales. Addressability and traceability also enhance a firm’s ability to learn about how, when, and where customers use electronic services and network connected devices. Learning is the first step of innovation.

### Scaling risk

Sometimes a new product will takeoff, and a company will struggle to meet unexpectedly high demand. If it can’t satisfy the market, competitors will move in and capture the sales that the innovator was unable to satisfy. In this case of digital products, a firm can use the cloud’s elasticity to quickly acquire new storage and processing resources. It might also take advantage of sourcing independence to use multiple clouds, another form of elasticity, to meet burgeoning demand.

### Control risk

The 2008 recession clearly demonstrated the risk of inadequate controls. In some cases, an organization acquired financial resources whose value was uncertain. In others, companies lost track of what they owned because of the complexity of financial instruments and the multiple transactions involving them. A system’s interface is a point of data capture, and a well-designed interface is a control mechanism. Although it is usually not effective at identifying falsified data, it can require that certain elements of data are entered. Addressability and traceability also means that the system can record who entered the data, from which device, and when.

| Risk/Capability | Demand | Inefficiency | Innovation | Scaling | Control |
| --- | --- | --- | --- | --- | --- |
| Interface control |  |  | ✔ |  | ✔ |
| Location independence |  | ✔ |  |  |  |
| Sourcing independence |  | ✔ |  | ✔ |  |
| Virtual business environment |  | ✔ | ✔ |  |  |
| Ubiquitous access | ✔ | ✔ | ✔ |  |  |
| Addressability and traceability | ✔ |  | ✔ |  | ✔ |
| Rapid elasticity | ✔ |  |  | ✔ |  |

Cloud capabilities and strategic risks

Most people think of cloud computing as an opportunity to lower costs by shifting processing from the corporate data center to a third party. More imaginative thinkers will see cloud computing as an opportunity to gain a competitive advantage.

# Distributed database

The client/server architecture is concerned with minimizing processing costs by distributing processing between multiple clients and servers. Another factor in the total processing cost equation is communication. The cost of transmitting data usually increases with distance, and there can be substantial savings by locating a database close to those most likely to use it. The trade-off for a distributed database is lowered communication costs versus increased complexity. While the Internet and fiber optics have lowered the costs of communication, for a globally distributed organization data transmission costs can still be an issue because many countries still lack the bandwidth found in advanced economies.

Distributed database architecture describes the situation where a database is in more than one location but still accessible as if it were centrally located. For example, a multinational organization might locate its Indonesian data in Jakarta, its Japanese data in Tokyo, and its Australian data in Sydney. If most queries deal with the local situation, communication costs are substantially lower than if the database were centrally located. Furthermore, since the database is still treated as one logical entity, queries that require access to different physical locations can be processed. For example, the query “Find total sales of red hats in Australia” is likely to originate in Australia and be processed using the Sydney part of the database. A query of the form “Find total sales of red hats” is more likely to come from a headquarters’ user and is resolved by accessing each of the local databases, though the user need not be aware of where the data are stored because the database appears as a single logical entity.

A distributed database management system (DDBMS) is a federation of individual DBMSs. Each site has a local DBMS and DC manager. In many respects, each site acts semi-independently. Each site also contains additional software that enables it to be part of the federation and act as a single database. It is this additional software that creates the DDBMS and enables the multiple databases to appear as one.

A DDBMS introduces a need for a data store containing details of the entire system. Information must be kept about the structure and location of every database, table, row, and column and their possible replicas. The system catalog is extended to include such information. The system catalog must also be distributed; otherwise, every request for information would have to be routed through some central point. For instance, if the systems catalog were stored in Tokyo, a query on Sydney data would first have to access the Tokyo-based catalog. This would create a bottleneck.

## A hybrid, distributed architecture

Any organization of a reasonable size is likely to have a mix of the data processing architectures. Databases will exist on stand-alone personal computers, multiple client/server networks, distributed mainframes, clouds, and so on. Architectures continue to evolve because information technology is dynamic. Today’s best solutions for data processing can become obsolete very quickly. Yet, organizations have invested large sums in existing systems that meet their current needs and do not warrant replacement. As a result, organizations evolve a hybrid architecture—a mix of the various forms. The concern of the IS unit is how to patch this hybrid together so that clients see it as a seamless system that readily provides needed information. In creating this ideal system, there are some underlying key concepts that should be observed. These fundamental principles were initially stated in terms of a distributed database. However, they can be considered to apply broadly to the evolving, hybrid architecture that organizations must continually fashion.

The fundamental principles of a hybrid architecture

| Principle |
| --- |
| Transparency |
| No reliance on a central site |
| Local autonomy |
| Continuous operation |
| Distributed query processing |
| Distributed transaction processing |
| Fragmentation independence |
| Replication independence |
| Hardware independence |
| Operating system independence |
| Network independence |
| DBMS independence |

## Transparency

The user should not have to know where data are stored nor how they are processed. The location of data, its storage format, and access method should be invisible to the client. The system should accept queries and resolve them expeditiously. Of course, the system should check that the person is authorized to access the requested data. Transparency is also known as location independence—the system can be used independently of the location of data.

## No reliance on a central site

Reliance on a central site for management of a hybrid architecture creates two major problems. First***,*** because all requests are routed through the central site, bottlenecks develop during peak periods. Second, if the central site fails, the entire system fails. A controlling central site is too vulnerable, and control should be distributed throughout the system.

## Local autonomy

A high degree of local autonomy avoids dependence on a central site. Data are locally owned and managed. The local site is responsible for the security, integrity, and storage of local data. There cannot be absolute local autonomy because the various sites must cooperate in order for transparency to be feasible. Cooperation always requires relinquishing some autonomy.

## Continuous operation

The system must be accessible when required. Since business is increasingly global and clients are geographically dispersed, the system must be continuously available. Many data centers now describe their operations as “24/7” (24 hours a day and 7 days a week). Clouds must operate continuously.

## Distributed query processing

The time taken to execute a query should be generally independent of the location from which it is submitted. Deciding the most efficient way to process the query is the system’s responsibility, not the client’s. For example, a Sydney analyst could submit the query, “Find sales of winter coats in Japan.” The system is responsible for deciding which messages and data to send between the various sites where tables are located.

## Distributed transaction processing

In a hybrid system, a single transaction can require updating of multiple files at multiple sites. The system must ensure that a transaction is successfully executed for all sites. Partial updating of files will cause inconsistencies.

## Fragmentation independence

Fragmentation independence means that any table can be broken into fragments and then stored in separate physical locations. For example, the sales table could be fragmented so that Indonesian data are stored in Jakarta, Japanese data in Tokyo, and so on. A fragment is any piece of a table that can be created by applying restriction and projection operations. Using join and union, fragments can be assembled to create the full table. Fragmentation is the key to a distributed database. Without fragmentation independence, data cannot be distributed.

## Replication independence

Fragmentation is good when local data are mainly processed locally, but there are some applications that also frequently process remote data. For example, the New York office of an international airline may need both American (local) and European (remote) data, and its London office may need American (remote) and European (local) data. In this case, fragmentation into American and European data may not substantially reduce communication costs.

Replication means that a fragment of data can be copied and physically stored at multiple sites; thus the European fragment could be replicated and stored in New York, and the American fragment replicated and stored in London. As a result, applications in both New York and London will reduce their communication costs. Of course, the trade-off is that when a replicated fragment is updated, all copies also must be updated. Reduced communication costs are exchanged for increased update complexity.

Replication independence implies that replication happens behind the scenes. The client is oblivious to replication and requires no knowledge of this activity.

There are two major approaches to replication: synchronous or asynchronous updates. Synchronous replication means all databases are updated at the same time. Although this is ideal, it is not a simple task and is resource intensive. Asynchronous replicatio**n** occurs when changes made to one database are relayed to other databases within a certain period established by the database administrator. It does not provide real-time updating, but it takes fewer IS resources. Asynchronous replication is a compromise strategy for distributed DBMS replication. When real-time updating is not absolutely necessary, asynchronous replication can save IS resources.

## Hardware independence

A hybrid architecture should support hardware from multiple suppliers without affecting the users’ capacity to query files. Hardware independence is a long-term goal of many IS managers. With virtualization, under which a computer can run another operating system, hardware independence has become less of an issue. For example, a Macintosh can run OS X (the native system), a variety of Windows operating systems, and many variations of Linux.

## Operating system independence

Operating system independence is another goal much sought after by IS executives. Ideally, the various DBMSs and applications of the hybrid system should work on a range of operating systems on a variety of hardware. Browser-based systems are a way of providing operating system independence.

## Network independence

Clearly, network independence is desired by organizations that wish to avoid the electronic shackles of being committed to any single hardware or software supplier.

## DBMS independence

The drive for independence is contagious and has been caught by the DBMS as well. Since SQL is a standard for relational databases, organizations may well be able to achieve DBMS independence. By settling on the relational model as the organizational standard, ensuring that all DBMSs installed conform to this model, and using only standard SQL, an organization may approach DBMS independence. Nevertheless, do not forget all those old systems from the pre-relational days—a legacy that must be supported in a hybrid architecture.

Organizations can gain considerable DBMS independence by using Open Database Connectivity (ODBC) technology, which was covered earlier in the chapter on SQL. An application that uses the ODBC interface can access any ODBC-compliant DBMS. In a distributed environment, such as an n-tier client/server, ODBC enables application servers to access a variety of vendors’ databases on different data servers.

## Conclusion—paradise postponed

For data managers, the 12 principles just outlined are ideal goals. In the hurly-burly of everyday business, incomplete information, and an uncertain future, data managers struggle valiantly to meet clients’ needs with a hybrid architecture that is an imperfect interpretation of IS paradise. It is unlikely that these principles will ever be totally achieved. They are guidelines and something to reflect on when making the inevitable trade-offs that occur in data management.

Now that you understand the general goals of a distributed database architecture, we will consider the major aspects of the enabling technology. First, we will look at distributed data access methods and then distributed database design. In keeping with our focus on the relational model, illustrative SQL examples are used.

# Distributed data access

When data are distributed across multiple locations, the data management logic must be adapted to recognize multiple locations. The various types of distributed data access methods are considered, and an example is used to illustrate the differences between the methods.

## Remote request

A remote request occurs when an application issues a single data request to a single remote site. Consider the case of a branch bank requesting data for a specific customer from the bank’s central server located in Atlanta. The SQL command specifies the name of the server (atlserver), the database (bankdb), and the name of the table (customer).

SELECT \* FROM atlserver.bankdb.customer

WHERE custcode = 12345;

A remote request can extract a table from the database for processing on the local database. For example, the Athens branch may download balance details of customers at the beginning of each day and handle queries locally rather than issuing a remote request. The SQL is

SELECT custcode, custbalance FROM atlserver.bankdb.customer

WHERE custbranch = 'Athens';

## Remote transaction

Multiple data requests are often necessary to execute a complete business transaction. For example, to add a new customer account might require inserting a row in two tables: one row for the account and another row in the table relating a customer to the new account. A remote transaction contains multiple data requests for a single remote location. The following example illustrates how a branch bank creates a new customer account on the central server:

BEGIN WORK;

INSERT INTO atlserver.bankdb.account

(accnum, acctype)

VALUES (789, 'C');

INSERT INTO atlserver.bankdb.cust\_acct

(custnum, accnum)

VALUES (123, 789);

COMMIT WORK;

The commands BEGIN WORK and COMMIT WORK surround the SQL commands necessary to complete the transaction. The transaction is successful only if both SQL statements are successfully executed. If one of the SQL statements fails, the entire transaction fails.

## Distributed transaction

A distributed transaction supports multiple data requests for data at multiple locations. Each request is for data on a single server. Support for distributed transactions permits a client to access tables on different servers.

Consider the case of a bank that operates in the United States and Norway and keeps details of employees on a server in the country in which they reside. The following example illustrates a revision of the database to record details of an employee who moves from the United States to Norway. The transaction copies the data for the employee from the Atlanta server to the Oslo server and then deletes the entry for that employee on the Atlanta server.

BEGIN WORK;

INSERT INTO osloserver.bankdb.employee

(empcode, emplname, …)

SELECT empcode, emplname, …

FROM atlserver.bankdb.employee

WHERE empcode = 123;

DELETE FROM atlserver.bankdb.employee

WHERE empcode = 123;

COMMIT WORK;

As in the case of the remote transaction, the transaction is successful only if both SQL statements are successfully executed.

## Distributed request

A distributed request is the most complicated form of distributed data access. It supports processing of multiple requests at multiple sites, and each request can access data on multiple sites. This means that a distributed request can handle data replicated or fragmented across multiple servers.

Let’s assume the bank has had a good year and decided to give all employees a 15 percent bonus based on their annual salary and add USD 1,000 or NOK 7,500 to their retirement account, depending on whether the employee is based in the United States or Norway.

BEGIN WORK;

CREATE VIEW temp

(empcode, empfname, emplname, empsalary)

AS

SELECT empcode, empfname, emplname, empsalary

FROM atlserver.bankdb.employee

UNION

SELECT empcode, empfname, emplname, empsalary

FROM osloserver.bankdb.employee;

SELECT empcode, empfname, emplname, empsalary\*.15 AS bonus

FROM temp;

UPDATE atlserver.bankdb.employee

SET empusdretfund = empusdretfund + 1000;

UPDATE osloserver.bankdb.employee

SET empkrnretfund = empkrnretfund + 7500;

COMMIT WORK;

The transaction first creates a view containing all employees by a union on the employee tables for both locations. This view is then used to calculate the bonus. Two SQL update commands then update the respective retirement fund records of the U.S. and Norwegian employees. Notice that retirement funds are recorded in U.S. dollars or Norwegian kroner.

Ideally, a distributed request should not require the application to know where data are physically located. A DDBMS should not require the application to specify the name of the server. So, for example, it should be possible to write the following SQL:

SELECT empcode, empfname, emplname, empsalary\*.15 AS bonus

FROM bankdb.employee;

It is the responsibility of the DDBMS to determine where data are stored. In other words, the DDBMS is responsible for ensuring data location and fragmentation transparency.

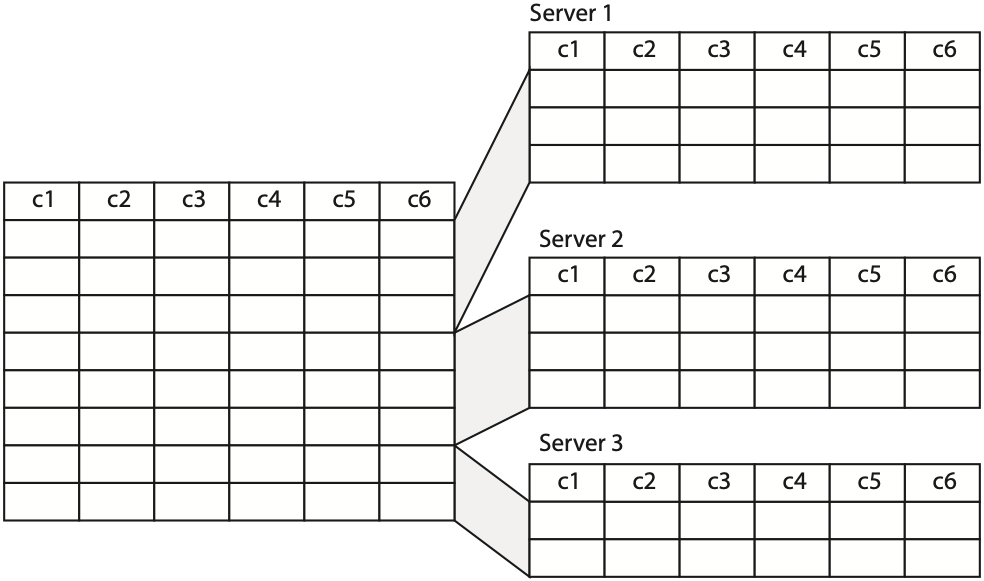
# Distributed database design

Designing a distributed database is a two-stage process. First***,*** develop a data model using the principles discussed in Section 2. Second, decide how data and processing will be distributed by applying the concepts of partitioning and replication. Partitioning is the fragmentation of tables across servers. Tables can be fragmented horizontally, vertically, or by some combination of both. Replication is the duplication of tables across servers.

## Horizontal fragmentation

A table is split into groups of rows when horizontally fragmented. For example, a firm may fragment its employee table into three because it has employees in Tokyo, Sydney, and Jakarta and store the fragment on the appropriate DBMS server for each city.

Horizontal fragmentation



Three separate employee tables would be defined. Each would have a different name (e.g., emp-syd) but exactly the same columns. To insert a new employee, the SQL code is

INSERT INTO TABLE emp\_syd

SELECT \* FROM new\_emp

WHERE emp\_nation = 'Australia';

INSERT INTO table emp\_tky

SELECT \* FROM new\_emp

WHERE emp\_nation = 'Japan';

INSERT INTO table emp\_jak

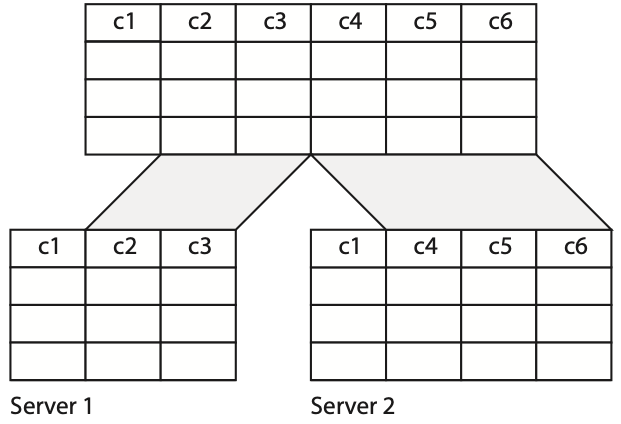
SELECT \* FROM new\_emp

WHERE emp\_nation = 'Indonesia';

## Vertical fragmentation

When vertically fragmented, a table is split into columns. For example, a firm may fragment its employee table vertically to spread the processing load across servers. There could be one server to handle address lookups and another to process payroll. In this case, the columns containing address information would be stored on one server and payroll columns on the other server. Notice that the primary key column (c1) must be stored on both servers; otherwise, the entity integrity rule is violated.

Vertical fragmentation



## Hybrid fragmentation

Hybrid fragmentation is a mixture of horizontal and vertical. For example, the employee table could be first horizontally fragmented to distribute the data to where employees are located. Then, some of the horizontal fragments could be vertically fragmented to split processing across servers. Thus, if Tokyo is the corporate headquarters with many employees, the Tokyo horizontal fragment of the employee database could be vertically fragmented so that separate servers could handle address and payroll processing.

Horizontal fragmentation distributes data and thus can be used to reduce communication costs by storing data where they are most likely to be needed. Vertical fragmentation is used to distribute data across servers so that the processing load can be distributed. Hybrid fragmentation can be used to distribute both data and applications across servers.

## Replication

Under replication, tables can be fully or partly replicated. Full replication means that tables are duplicated at each of the sites. The advantages of full replication are greater data integrity (because replication is essentially mirroring) and faster processing (because it can be done locally). However, replication is expensive because of the need to synchronize inserts, updates, and deletes across the replicated tables. When one table is altered, all the replicas must also be modified. A compromise is to usepartial replication by duplicating the indexes only. This will increase the speed of query processing. The index can be processed locally, and then the required rows retrieved from the remote database.

Skill builder

A company supplies pharmaceuticals to sheep and cattle stations in outback Australia. Its agents often visit remote areas and are usually out of reach of a mobile phone network. To advise station owners, what data management strategy would you recommend for the company?

## Conclusion

The two fundamental skills of data management, data modeling and data querying, are not changed by the development of a distributed data architecture such as client/server and the adoption of cloud computing. Data modeling remains unchanged. A high-fidelity data model is required regardless of where data are stored and whichever architecture is selected. SQL can be used to query local, remote, and distributed databases. Indeed, the adoption of client/server technology has seen a widespread increase in the demand for SQL skills.

## Summary

Data can be stored and processed locally or remotely. Combinations of these two options provide the basic architecture options: remote job entry, personal database, and client/server. Many organizations are looking to cloud computing to reduce the processing costs and to allow them to focus on building systems. Cloud computing can also provide organizations with a competitive advantage if they exploit its seven capabilities to reduce one or more of the strategic risks.

Under distributed database architecture, a database is in more than one location but still accessible as if it were centrally located. The trade-off is lowered communication costs versus increased complexity. A hybrid architecture is a mix of data processing architectures. The concern of the IS department is to patch this hybrid together so that clients see a seamless system that readily provides needed information. The fundamental principles that a hybrid architecture should satisfy are transparency, no reliance on a central site, local autonomy, continuous operation, distributed query processing, distributed transaction processing, fragmentation independence, replication independence, hardware independence, operating system independence, network independence, and DBMS independence.

There are four types of distributed data access. In order of complexity, these are remote request, remote transaction, distributed transaction, and distributed request.

Distributed database design is based on the principles of fragmentation and replication. Horizontal fragmentation splits a table by rows and reduces communication costs by placing data where they are likely to be required. Vertical fragmentation splits a table by columns and spreads the processing load across servers. Hybrid fragmentation is a combination of horizontal and vertical fragmentation. Replication is the duplication of identical tables at different sites. Partial replication involves the replication of indexes. Replication speeds up local processing at the cost of maintaining the replicas.

## Key terms and concepts

Application server

Client/server

Cloud computing

Continuous operation

Data communications manager (DC)

Data processing

Data server

Data storage

Database architecture

Database management system (DBMS)

DBMS independence

DBMS server

Distributed data access

Distributed database

Distributed query processing

Distributed request

Distributed transaction

Distributed transaction processing

Fragmentation independence

Hardware independence

Horizontal fragmentation

Hybrid architecture

Hybrid fragmentation

Local autonomy

Mainframe

N-tier architecture

Network independence

Personal database

Remote job entry

Remote request

Remote transaction

Replication

Replication independence

Server

Software independence

Strategic risk

Supercomputer

Three-tier architecture

Transaction processing monitor

Transparency

Two-tier architecture

Vertical fragmentation

## References and additional readings

Child, J. (1987). Information technology, organizations, and the response to strategic challenges. *California Management Review, 30*(1), 33-50.

Iyer, B., & Henderson, J. C. (2010). Preparing for the future: understanding the seven capabilities of cloud computing. *MIS Executive, 9*(2), 117-131.

Watson, R. T., Wynn, D., & Boudreau, M.-C. (2005). JBoss: The evolution of professional open source software. *MISQ Executive, 4*(3), 329-341.

## Exercises

1. How does client/server differ from cloud computing?
2. In what situations are you likely to use remote job entry?
3. What are the disadvantages of personal databases?
4. What is a firm likely to gain when it moves from a centralized to a distributed database? What are the potential costs?
5. In terms of a hybrid architecture, what does transparency mean?
6. In terms of a hybrid architecture, what does fragmentation independence mean?
7. In terms of a hybrid architecture, what does DBMS independence mean?
8. How does ODBC support a hybrid architecture?
9. A university professor is about to develop a large simulation model for describing the global economy. The model uses data from 65 countries to simulate alternative economic policies and their possible outcomes. In terms of volume, the data requirements are quite modest, but the mathematical model is very complex, and there are many equations that must be solved for each quarter the model is run. What data processing/data storage architecture would you recommend?
10. A multinational company has operated relatively independent organizations in 15 countries. The new CEO wants greater coordination and believes that marketing, production, and purchasing should be globally managed. As a result, the corporate IS department must work with the separate national IS departments to integrate the various national applications and databases. What are the implications for the corporate data processing and database architecture? What are the key facts you would like to know before developing an integration plan? What problems do you anticipate? What is your intuitive feeling about the key features of the new architecture?
11. A university wants to teach a specialized data management topic to its students every semester. It will take about two weeks to cover the topic, and during this period students will need access to a small high performance computing cluster on which the necessary software is installed. The software is Linux-based. Investigate three cloud computing offerings and make a recommendation as to which one the university should use.

22. SQL and Java

The vision for Java is to be the concrete and nails that people use to build this incredible network system that is happening all around us.

James Gosling, 2000

## Learning objectives

Students completing this chapter will be able to

* write Java program to process a parameterized SQL query;
* read a CSV file and insert rows into tables;
* write a program using HMTL and JavaServer Pages (JSP) to insert data collected by a form data into a relational database;
* understand how SQL commands are used for transaction processing.

## Introduction

Java is a platform-independent application development language. Its object-oriented nature makes it easy to create and maintain software and prototypes. These features make Java an attractive development language for many applications, including database systems.

This chapter assumes that you have some knowledge of Java programming, can use an integrated development environment (IDE) (e.g., BlueJ, Eclipse, or NetBeans), and know how to use a Java library. It is also requires some HTML skills because JavaServer Pages (JSP) are created by embedding Java code within an HTML document. MySQL is used for all the examples, but the fundamental principles are the same for all relational databases. With a few changes, your program will work with another implementation of the relational model.

# JDBC

Java Database Connectivity (JDBC), a Java version of a portable SQL command line interface (CLI), is modeled on Open Database Connectivity (ODBC.) JDBC enables programmers to write Java software that is both operating system and DBMS independent. The JDBC core, which handles 90 percent of database programming, contains seven interfaces and two classes, which are part of the java.sql package. The purpose of each of these is summarized in the following table.

JDBC core interfaces and classes

| Interfaces | Description |
| --- | --- |
| Connection | Connects an application to a database |
| Statement | A container for an SQL statement |
| PreparedStatement | Precompiles an SQL statement and then uses it multiple times |
| CallableStatement | Executes a stored procedure |
| ResultSet | The rows returned when a query is executed |
| ResultSetMetaData | The number, types, and properties of the result set |
| Classes |  |
| DriverManager | Loads driver objects and creates database connections |
| DriverPropertyInfo | Used by specialized clients |

For each DBMS, implementations of these interfaces are required because they are specific to the DBMS and not part of the Java package. For example, specific drivers are needed for MySQL, DB2, Oracle, and MS SQL. Before using JDBC, you will need to install these interfaces for the DBMS you plan to access. The standard practice appears to be to refer to this set of interfaces as the driver, but the ‘driver’ also includes implementations of all the interfaces.

# Java EE

Java EE (Java Platform Enterprise Edition) is a platform for multi-tiered enterprise applications. In the typical three-tier model, a Java EE application server sits between the client’s browser and the database server. A Java EE compliant server, of which there are a variety, is needed to process JSP.

# Using SQL within Java

In this section, you will need a Java integrated development environment to execute the sample code. A good option is Eclipse,[[8]](#footnote-8) a widely used open source IDE.

We now examine each of the major steps in processing an SQL query and, in the process, create Java methods to query a database.

## Connect to the database

The getConnection method of the DriverManager specifies the URL of the database, the account identifier, and its password. You cannot assume that connecting to the database will be trouble-free. Thus, good coding practice requires that you detect and report any errors using Java’s try-catch structure.

You connect to a DBMS by supplying its url and the account name and password.

try {

conn = DriverManager.getConnection(url, account, password);

} catch (SQLException error) {

System.out.println("Error connecting to database: " + error.toString());

System.exit(1);

}

The format of the url parameter varies with the JDBC driver, and you will need to consult the documentation for the driver. In the case of MySQL, the possible formats are

jdbc:mysql:database

jdbc:mysql://host/database

jdbc:mysql://host:port/database

The default value for host is “localhost” and, for MySQL, the default port is “3306.”

For example:

jdbc:mysql://[www.richardtwatson.com:3306/Text](http://www.richardtwatson.com:3306/Text)

will enable connection to the database “Text” on the host “[www.richardtwatson.com](http://www.richardtwatson.com)” on port 3306.

## Create and execute an SQL statement

The prepareStatement method is invoked to produce a Statement object (see the following code). Note that the conn in conn.prepareStatement() refers to the connection created by the getConnection method. Parameters are used in prepareStatement to set execution time values, with each parameter indicated by a question mark (?). Parameters are identified by an integer for the order in which they appear in the SQL statement. In the following sample code, shrdiv is a parameter and is set to indiv by stmt.setInit(1,indiv), where 1 is its identifier (i.e., the first parameter in the SQL statement) and indiv is the parameter. The value of indiv is received as input at run time.

The SQL query is run by the executeQuery() method of the Statement object. The results are returned in a ResultSet object.

Create and execute an SQL statement

try {

stmt = conn.prepareStatement("SELECT shrfirm, shrdiv FROM shr WHERE shrdiv > ?");

// set the value for shrdiv to indiv

stmt.setInt(1,indiv);

rs = stmt.executeQuery();

}

catch (SQLException error)

{

System.out.println("Error reporting from database: "

+ error.toString());

System.exit(1);

}

## Report a SELECT

The rows in the table containing the results of the query are processed a row at a time using the next method of the ResultSet object (see the following code). Columns are retrieved one at a time using a get method within a loop, where the integer value specified in the method call refers to a column (e.g., rs.getString(1) returns the first column of the result set as a string). The get method used depends on the type of data returned by the query. For example, getString() gets a character string, getInt() gets an integer, and so on.

Reporting a SELECT

while (rs.next()) {

String firm = rs.getString(1);

int div = rs.getInt(2);

System.out.println(firm + " " + div);

}

## Inserting a row

The prepareStatement() is also used for inserting a row in a similar manner, as the following example shows. The executeUpdate() method performs the insert. Notice the use of try and catch to detect and report an error during an insert.

Inserting a row

try {

stmt = conn.prepareStatement("INSERT INTO alien (alnum, alname) VALUES (?,?)");

stmt.setInt(1,10);

stmt.setString(2, "ET");

stmt.executeUpdate();

}

catch (SQLException error)

{

System.out.println("Error inserting row into database: "

+ error.toString());

System.exit(1);

}

## Release the Statement object

The resources associated with the various objects created are freed as follows:

stmt.close();

rs.close();

conn.close();

To illustrate the use of Java, a pair of programs is available. The first, DataTest.java, creates a database connection and then calls a method to execute and report an SQL. The second program, DatabaseAccess.java, contains the methods called by the first. Both programs are available on the book’s web site for examination and use.

Skill builder

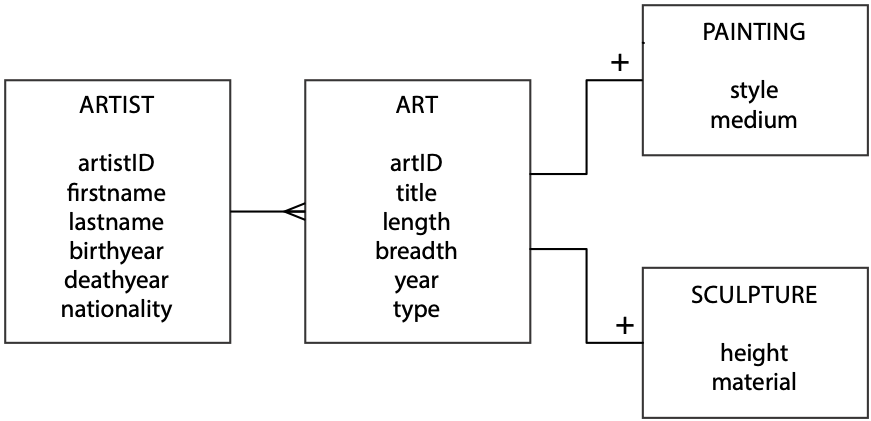
1. Get from this book’s supporting Web site the code of DatabaseTest.java and DatabaseAccess.java.
2. Inspect the code to learn how you use SQL from within a Java application.
3. Run DatabaseTest.java with a few different values for indiv.
4. Make modifications to query a different table.

## Loading a text file into a database

Java is useful when you have a dataset that you need to load into a database. For example, you might have set up a spreadsheet and later realized that it would be more efficient to use a database. In this case, you can export the spreadsheet and load the data into one or more tables.

We will illustrate loading data from a text file into the ArtCollection database with the following design. Notice that there different tables for different types of art, paintings, and sculptures in this case.[[9]](#footnote-9)

ArtCollection data model



### CSV file

A comma-separated values (CSV) file[[10]](#footnote-10) is a common form of text file because most spreadsheet programs offer it as an export option. There are a variety of options available in terms of the separator for fields (usually a comma or tab) and the use of quotation marks to denote text fields. Usually, each record has the same number of fields. Frequently, the first record contains names for each of the fields. Notice in the example file that follows that a record contains details of an artist and one piece of that person’s art. As the data model specifies, these two sets of data go into separate tables.

A CSV file

firstName,lastName,birthyear,deathyear,nationality,title,length,breadth,year,style,medium

Edvard,Munch,1863,1944,Norwegian,The Scream,36,29,1893,Cubist,Oil

Claude,Monet,1840,1926,French,Bridge over a Pond of Water Lilies,36,29,1899,Impressionist,Oil

Toulouse,Lautrec,1864,1901,French,Avril,55,33,1893,Impressionist,Oil

Vincent,Van Gogh,1853,1890,French,The Starry Night,29,36,1889,Impressionist,Oil

Johannes,Vermeer,1632,1675,Dutch,Girl with a Pearl Earring,17,15,1665,Impressionist,Oil

Salvador,Dali,1904,1989,Spanish,The Persistence of Memory,9.5,13,1931,Surreal,Oil

Andrew,Wyeth,1917,2009,American,Christina's World ,32,47,1948,Surreal,Oil

William,Turner,1789,1862,English,The Battle of Trafalgar,103,145,1822,Surreal,Oil

Tom,Roberts,1856,1931,Australian,Shearing the Rams,48,72,1890,Surreal,Oil

Paul,Gauguin,1848,1903,French,Spirit of the Dead Watching,28,36,1892,Surreal,Watercolor

Because CSV files are so widely used, there are Java libraries for processing them. We have opted for CsvReader.[[11]](#footnote-11) Here is some code for connecting to a CSV file defined by its URL. You can also connect to text files on your personal computer.

Connecting to a CSV file

try {

csvurl = new URL("[http://www.richardtwatson.com/data/painting.csv](http://people.terry.uga.edu/rwatson/data/painting.csv)");   
} catch (IOException error) {

System.out.println("Error accessing url: " + error.toString());

System.exit(1);

}

A CSV file is read a record at a time, and the required fields are extracted for further processing. The following code illustrates how a loop is used to read each record in a CSV file and extract some fields. Note the following:

* The first record, the header, contains the names of each field and is read before the loop starts.
* Field names can be used to extract each field with a get() method.
* You often need to convert the input string data to another format, such as integer.
* Methods, addArtist, and addArt are included in the loop to add details of an artist to the artist table and a piece of art to the art table.

Processing a CSV file

try {

input = new CsvReader(new InputStreamReader(csvurl.openStream()));

input.readHeaders();

while (input.readRecord())

{

// Artist

String firstName = input.get("firstName");

String lastName = input.get("lastName");

String nationality = input.get("nationality");

int birthYear = Integer.parseInt(input.get("birthyear"));

int deathYear = Integer.parseInt(input.get("deathyear"));

artistPK = addArtist(conn, firstName, lastName, nationality, birthYear, deathYear);

// Painting

String title = input.get("title");

double length = Double.parseDouble(input.get("length"));

double breadth = Double.parseDouble(input.get("breadth"));

int year = Integer.parseInt(input.get("year"));

addArt(conn, title, length, breadth, year,artistPK);

}

input.close();

} catch (IOException error) {

System.out.println("Error reading CSV file: " + error.toString());

System.exit(1);

}

We now need to consider the addArtist method. Its purpose is to add a row to the artist table. The primary key, artistID, is specified as AUTO\_INCREMENT, which means it is set by the DBMS. Because of the 1:m relationship between artist and art, we need to know the value of artistID, the primary key of art, to set the foreign key for the corresponding piece of art. We can use the following code to determine the primary key of the last insert.

Determining the value of the primary key of the last insert

rs = stmt.executeQuery("SELECT LAST\_INSERT\_ID()");

if (rs.next()) {

autoIncKey = rs.getInt(1);

}

The addArtist method returns the value of the primary key of the most recent insert so that we can then use this value as the foreign key for the related piece of art.

artistPK = addArtist(conn, firstName, lastName, nationality, birthYear, deathYear);

The call to the addArt method includes the returned value as a parameter. That is, it passes the primary key of artist to be used as foreign key of art.

addArt(conn, title, length, breadth, year,artistPK);

The Java code for the complete application is available on the book’s web site.[[12]](#footnote-12)

Skill builder

1. Add a few sculptors and a piece of sculpture to the CSV file. Also, decide how you will differentiate between a painting and a sculpture.
2. Add a method to ArtCollector.java to insert an artist and that person’s sculpture.
3. Run the revised program so that it adds both paintings and sculptures for various artists.

# JavaServer Pages (JSP)

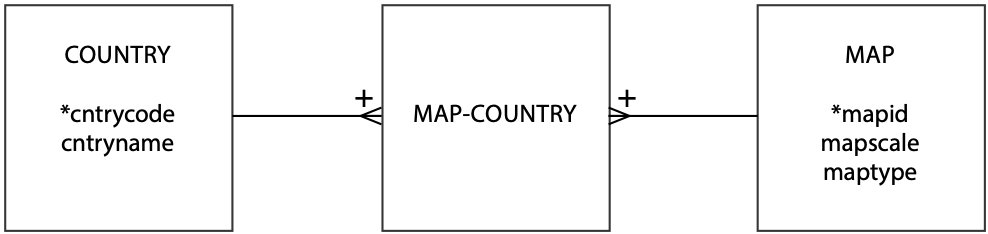
Standard HTML pages are static. They contain text and graphics that don’t change unless someone recodes the page. JSP and other technologies, such as PHP and ASP, enable you to create dynamic pages whose content can change based to suit the goals of the person accessing the page, the browser used, or the device on which the page is displayed.

A JSP contains standard HTML code, the static, and JSP elements, the dynamic. When someone requests a JSP, the server combines HTML and JSP elements to deliver a dynamic page. JSP is also useful for server side processing. For example, when someone submits a form, JSP elements can be used to process the form on the server.

***Map collection case***

The Expeditioner has added a new product line to meet the needs of its changing clientele. It now stocks a range of maps. The firm’s data modeler has created a data model describing the situation . A map has a scale; for example, a scale of 1:1 000 000 means that 1 unit on the map is 1,000,000 units on the ground (or 1 cm is 10 km, and 1 inch is ~16 miles). There are three types of maps: road, rail, and canal. Initially, The Expeditioner decided to stock maps for only a few European countries.

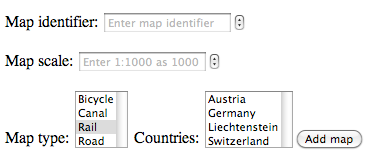
The Expeditioner’s map collection data model



### Data entry

Once the tables have been defined, we want to insert some records. This could be done using the insertSQL method of the Java code just created, but it would be very tedious, as we would type insert statements for each row (e.g., INSERT INTO map VALUES (1, 1000000, 'Rail');). A better approach is to use a HTML data entry form. An appropriate form and its associated code follow.

Map collection data entry form



Map collection data entry HTML code (index.html)

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "<http://www.w3.org/TR/html4/loose.dtd>">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Map collection</title>

</head>

</body>

<form name="MapInsert" action="mapinsert.jsp" method="post">

<p>

<label>Map identifier: <input type="number" required

pattern="[0-9]{3}" name="mapid" size="24" value=""

placeholder="Enter map identifier"></label>

<p>

<label>Map scale: <input type="number" required

pattern="[0-9]+" min="1000" max="100000" step="1000" name="mapscale"

size="24" value="" placeholder="Enter 1:1000 as 1000"></label>

<p>

<label>Map type: <select name="maptype" size="3"

placeholder="Select type of map"></label>

</p>

<option value="Bicycle">Bicycle</option>

<option value="Canal">Canal</option>

<option value="Rail" selected>Rail</option>

<option value="Road">Road</option>

</select> <label>Countries: <select name="countries" size="4" multiple

placeholder="Select countries"></label>

<option value="at">Austria</option>

<option value="de">Germany</option>

<option value="li">Liechtenstein</option>

<option value="ch">Switzerland</option>

<input type="submit" name="submitbutton" value="Add map">

</form>

</body>

</html>

The data entry form captures a map’s identifier, scale, type, and the number of countries covered by the map. A map identifier is of the form M001 to M999, and a regular expression is used to validate the input.[[13]](#footnote-13) A map scale must be numeric with a minimum of 1000 and maximum of 100000 in increments of 1000. The type of map is selected from the list, which is defined so that only one type can be selected. Multiple countries can be selected by holding down an appropriate key combination when selected from the displayed list of countries. When the “Add map” button is clicked, the page mapinsert.jsp is called, which is specified in the first line of the HTML body code.

### Passing data from a form to a JSP

In a form, the various entry fields are each identified by a name specification (e.g., name=”mapid” and name=”mapscale”). In the corresponding JSP, which is the one defined in the form’s action statement (i.e., action="mapinsert.jsp") these same fields are accessed using the getParameter method when there is a single value or getParameterValues when there are multiple values. The following chunks of code show corresponding form and JSP code for maptype.

<label>Map type: <select name="maptype" size="3"

placeholder="Select type of map"></label>

maptype = request.getParameter("maptype");

### Transaction processing

A transaction is a logical unit of work. In the case of adding a map, it means inserting a row in map for the map and inserting one row in mapCountry for each country on the map. All of these inserts must be executed without failure for the entire transaction to be processed. SQL has two commands for supporting transaction processing: COMMIT and ROLLBACK. If no errors are detected, then the various changes made by a transaction can be *committed*. In the event of any errors during the processing of a transaction, the database should be *rolled back* to the state prior to the transaction.

#### Autocommit

Before processing a transaction, you need to turn off autocommit to avoid committing each database change separately before the entire transaction is complete. It is a good idea to set the value for autocommit immediately after a successful database connection, which is what you will see when you inspect the code for mapinsert.jsp. The following code sets autocommit to false in the case where conn is the connection object.

conn.setAutoCommit(false);

#### Commit

The COMMIT command is executed when all parts of a transaction have been successfully executed. It makes the changes to the database permanent.

conn.commit(); // all inserts successful

#### Rollback

The ROLLBACK command is executed when any part of a transaction fails. All changes to the database since the beginning of the transaction are reversed, and the database is restored to its state before the transaction commenced.

conn.rollback(); // at least one insert failed

#### Completing the transaction

The final task, to commit or roll back the transaction depending on whether errors were detected during any of the inserts, is determined by examining transOK, which is a boolean variable set to false when any errors are detected during a transaction.

Completing the transaction

if (transOK) {

conn.commit(); // all inserts successful

} else {

conn.rollback(); // at least one insert failed

}

conn.close(); // close database

### Putting it all together

You have now seen some of the key pieces for the event handler that processes a transaction to add a map and the countries on that map. The code can be downloaded from the book’s Web site.[[14]](#footnote-14)

### mapinsert.jsp

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "<http://www.w3.org/TR/html4/loose.dtd>">

<%@ page import="java.util.\*"%>

<%@ page import="java.lang.\*"%>

<%@ page import="java.sql.\*"%>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Map insert page</title>

</head>

<body>

<%

String url;

String jdbc = "jdbc:mysql:";

String database = "//localhost:3306/MapCollection";

String username = "student", password = "student";

String mapid, maptype, countries;

String[] country;

int mapscale = 0;

boolean transOK = true;

PreparedStatement insertMap;

PreparedStatement insertMapCountry;

Connection conn = null;

// make connection

url = jdbc + database;

try {

conn = DriverManager.getConnection(url, username, password);

} catch (SQLException error) {

System.out.println("Error connecting to database: "

+ error.toString());

System.exit(1);

}

try {

conn.setAutoCommit(false);

} catch (SQLException error) {

System.out.println("Error turning off autocommit"

+ error.toString());

System.exit(2);

}

//form data

mapid = request.getParameter("mapid");

mapscale = Integer.parseInt(request.getParameter("mapscale"));

maptype = request.getParameter("maptype");

country = request.getParameterValues("countries");

transOK = true;

// insert the map

try {

insertMap = conn.prepareStatement("INSERT INTO map (mapid, mapscale, maptype) VALUES (?,?,?)");

insertMap.setString(1, mapid);

insertMap.setInt(2, mapscale);

insertMap.setString(3, maptype);

System.out.println(insertMap);

insertMap.executeUpdate();

// insert the countries

for (int loopInx = 0; loopInx < country.length; loopInx++) {

insertMapCountry = conn.prepareStatement("INSERT INTO mapCountry (mapid ,cntrycode ) VALUES (?,?)");

insertMapCountry.setString(1, mapid);

insertMapCountry.setString(2, country[loopInx]);

System.out.println(insertMapCountry);

insertMapCountry.executeUpdate();

}

} catch (SQLException error) {

System.out.println("Error inserting row: " + error.toString());

transOK = false;

}

if (transOK) {

conn.commit(); // all inserts successful

System.out.println("Transaction commit");

} else {

conn.rollback(); // at least one insert failed

System.out.println("Transaction rollback");

}

conn.close();

System.out.println("Database close"); // close database

%>

</body>

</html>

The code in mapinsert.jsp does the following:

1. Creates a connection to a database
2. Gets the data from the map entry form
3. Inserts a row for a new map
4. Uses a loop to insert a row for each country on the map
5. Shows how to handle a transaction failure.

## Conclusion

Java is a widely used object-oriented programming language for developing distributed multi-tier applications. JDBC is a key technology in this environment because it enables a Java application to interact with various implementations of the relational database model (e.g., Oracle, SQL Server, MySQL). As this chapter has demonstrated, with the help of a few examples, JDBC can be readily understood and applied.

## Summary

Java is a platform-independent application development language. JDBC enables programs that are DBMS independent. SQL statements can be embedded within Java programs. JSP is used to support server side processing. COMMIT and ROLLBACK are used for transaction processing.

## Key terms and concepts

Autocommit

Comma-separated values

IDE

COMMIT

Java

Java Database Connectivity (JDBC)

Java Server Pages (JSP)

ROLLBACK

Transaction processing

## References and additional readings

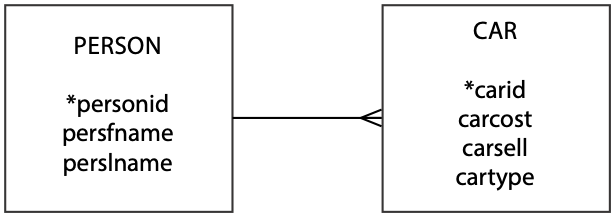
Barnes, D. J., and M. Kölling. 2005. Objects first with Java : a practical introduction using Blue J. 2nd ed. Upper Saddle River, NJ: Prentice Hall.

## Exercises

1. Write a Java program to run a parameterized query against the ClassicModels database. For example, you might report the sum of payments for a given customer’s name.

Extend ArtCollection.java so that it can handle inserting rows for multiple pieces of art for a single artist. You will have to rethink the structure of the CSV file and how to record multiple pieces of art from a single artist.

1. Write a JSP application to maintain the database defined by the following data model. The database keeps track of the cars sold by a salesperson in an automotive dealership. Your application should be able to add a person and the cars a person has sold. These should be separate transactions.



23. Data Integrity

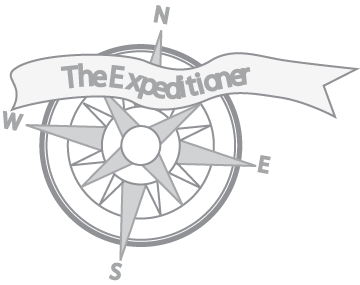
Integrity without knowledge is weak and useless, and knowledge without integrity is dangerous and dreadful.

Samuel Johnson, Rasselas, 1759

## Learning objectives

After completing this chapter, you will

* understand the three major data integrity outcomes;
* understand the strategies for achieving each of the data integrity outcomes;
* understand the possible threats to data integrity and how to deal with them;
* understand the principles of transaction management;
* realize that successful data management requires making data available and maintaining data integrity.

The Expeditioner has become very dependent on its databases. The day-to-day operations of the company would be adversely affected if the major operational databases were lost. Indeed, The Expeditioner may not be able to survive a major data loss. Recently, there have also been a few minor problems with the quality and confidentiality of some of the databases. A part-time salesperson was discovered making a query about staff salaries. A major order was nearly lost when it was shipped to the wrong address because the complete shipping address had not been entered when the order was taken. The sales database had been offline for 30 minutes last Monday morning because of a disk sector read error.

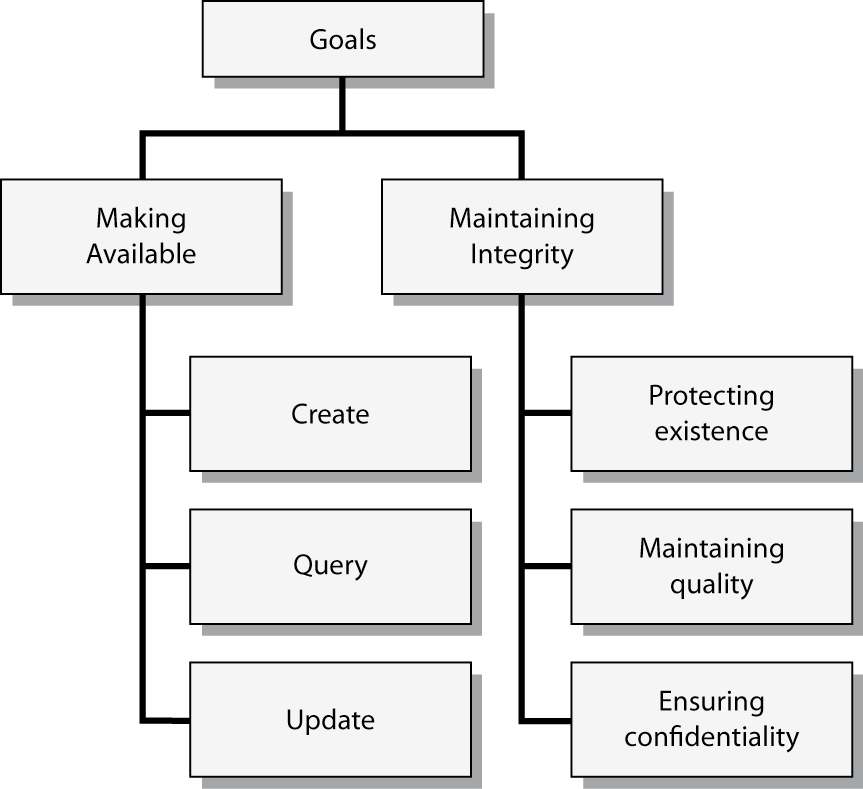
The Expeditioner had spent much time and money creating an extremely effective and efficient management system. It became clear, however, that more attention needed to be paid to maintaining the system and ensuring that high-quality data were continuously available to authorized users.

## Introduction

The management of data is driven by two goals: availability and integrity. Availability deals with making data available to whomever needs it, whenever and wherever he or she needs it, and in a meaningful form. As illustrated in following figure , availability concerns the creation, interrogation, and update of data stores. Although most of the book, thus far, has dealt with making data available, a database is of little use to anyone unless it has integrity. Maintaining data integrity implies three goals:[[15]](#footnote-15)

1. Protecting existence: Data are available when needed.
2. Maintaining quality: Data are accurate, complete, and current.
3. Ensuring confidentiality: Data are accessed only by those authorized to do so.

Goals of managing organizational memory



This chapter considers the three types of strategies for maintaining data integrity:

1. Legal strategies are externally imposed laws, rules, and regulations. Privacy laws are an example.
2. Administrative strategies are organizational policies and procedures. An example is a standard operating procedure of storing all backup files in a locked vault.
3. Technical strategies are those incorporated in the computer system (e.g., database management system [DBMS], application program, or operating system). An example is the inclusion of validation rules (e.g., NOT NULL) in a database definition that are used to ensure data quality when a database is updated.

A consistent database is one in which all data integrity constraints are satisfied.

Our focus is on data stored in multiuser computer databases and technical and administrative strategies for maintaining integrity in computer system environments., commonly referred to as database integrity. More and more, organizational memories are being captured in computerized databases. From an integrity perspective, this is a very positive development. Computers offer some excellent mechanisms for controlling data integrity, but this does not eliminate the need for administrative strategies. Both administrative and technical strategies are still needed.

Who is responsible for database integrity? Some would say the data users, others the database administrator. Both groups are right; database integrity is a shared responsibility, and the way it is managed may differ across organizations. Our focus is on the tools and strategies for maintaining data integrity, regardless of who is responsible.

The strategies for achieving the three integrity outcomes are summarized in the following table. We will cover procedures for protecting existence, followed by those for maintaining integrity, and finally those used to ensure confidentiality. Before considering each of these goals, we need to examine the general issue of transaction management.

Strategies for maintaining database integrity

| Database integrity outcome | Strategies for achieving the outcome |
| --- | --- |
| Protecting existence | Isolation (preventive)  Database backup and recovery (curative) |
| Maintaining quality | Update authorization  Integrity constraints/data validation  Concurrent update control |
| Ensuring confidentiality | Access control  Encryption |

# Transaction management

Transaction management focuses on ensuring that transactions are correctly recorded in the database. The transactionmanager is the element of a DBMS that processes transactions. A transaction is a series of actions to be taken on the database such that they must be entirely completed or entirely aborted. A transaction is a logical unit of work. All its elements must be processed; otherwise, the database will be incorrect. For example, with a sale of a product, the transaction consists of at least two parts: an update to the inventory on hand, and an update to the customer information for the items sold in order to bill the customer later. Updating only the inventory or only the customer information would create a database without integrity and an inconsistent database.

Transaction managers are designed to accomplish the ACID (atomicity, consistency, isolation, and durability) concept. These attributes are:

1. Atomicity**:** If a transaction has two or more discrete pieces of information, either all of the pieces are committed or none are.
2. Consistency**:** Either a transaction creates a valid new database state or, if any failure occurs, the transaction manager returns the database to its prior state.
3. Isolation**:** A transaction in process and not yet committed must remain isolated from any other transaction.
4. Durability**:** Committed data are saved by the DBMS so that, in the event of a failure and system recovery, these data are available in their correct state.

Transaction atomicity requires that all transactions are processed on an all-or-nothingbasis and that any collection of transactions is serializable. When a transaction is executed, either all its changes to the database are completed or none of the changes are performed. In other words, the entire unit of work must be completed. If a transaction is terminated before it is completed, the transaction manager must undo the executed actions to restore the database to its state before the transaction commenced (the consistency concept). Once a transaction is successfully completed, it does not need to be undone. For efficiency reasons, transactions should be no larger than necessary to ensure the integrity of the database. For example, in an accounting system, a debit and credit would be an appropriate transaction, because this is the minimum amount of work needed to keep the books in balance.

Serializability relates to the execution of a set of transactions. An interleaved execution schedule (i.e., the elements of different transactions are intermixed) is serializable if its outcome is equivalent to a non-interleaved (i.e., serial) schedule. Interleaved operations are often used to increase the efficiency of computing resources, so it is not unusual for the components of multiple transactions to be interleaved. Interleaved transactions cause problems when they interfere with each other and, as a result, compromise the correctness of the database.

The ACID concept is critical to concurrent update control and recovery after a transaction failure.

## Concurrent update control

When updating a database, most applications implicitly assume that their actions do not interfere with any other’s actions. If the DBMS is a single-user system, then a lack of interference is guaranteed. Most DBMSs, however, are multiuser systems, where many analysts and applications can be accessing a given database at the same time. When two or more transactions are allowed to update a database concurrently, the integrity of the database is threatened. For example, multiple agents selling airline tickets should not be able to sell the same seat twice. Similarly, inconsistent results can be obtained by a retrieval transaction when retrievals are being made simultaneously with updates. This gives the appearance of a loss of database integrity. We will first discuss the integrity problems caused by concurrent updates and then show how to control them to ensure database quality.

### Lost update

Uncontrolled concurrent updates can result in the lost-update or phantom-record problem. To illustrate the lost-update problem, suppose two concurrent update transactions simultaneously want to update the same record in an inventory file. Both want to update the quantity-on-hand field (quantity). Assume quantity has a current value of 40. One update transaction wants to add 80 units (a delivery) to quantity, while the other transaction wants to subtract 20 units (a sale).

Suppose the transactions have concurrent access to the record; that is, each transaction is able to read the record from the database before a previous transaction has been committed. This sequence is depicted in the following figure. Note that the first transaction, A, has not updated the database when the second transaction, B, reads the same record. Thus, both A and B read in a value of 40 for quantity. Both make their calculations, then A writes the value of 120 to disk, followed by B promptly overwriting the 120 with 20. The result is that the delivery of 80 units, transaction A, is lost during the update process.

Lost update when concurrent accessing is allowed

| Time | Action |  | Database record | |
| --- | --- | --- | --- | --- |
|  |  |  | Part# | Quantity |
|  |  |  | P10 | 40 |
| T1 | App A receives message for delivery of 80 units of P10 |  |  |  |
| T2 | App A reads the record for P10 | ← | P10 | 40 |
| T3 | App B receives message for sale of 20 units of P10 |  |  |  |
| T4 | App B reads the record for P10 | ← | P10 | 40 |
| T5 | App A processes delivery (40 + 80 = 120) |  |  |  |
| T6 | App A updates the record for P10 | → | P10 | 120 |
| T7 | App B processes the sale of (40 - 20 = 20) |  |  |  |
| T8 | App B updates the record P10 | → | P10 | 20 |

Inconsistent retrievals occur when a transaction calculates some aggregate function (e.g., sum) over a set of data while other transactions are updating the same data. The problem is that the retrieval may read some data before they are changed and other data after they are changed, thereby yielding inconsistent results.

### The solution: locking

To prevent lost updates and inconsistent retrieval results, the DBMS must incorporate a resource lock, a basic tool of transaction management to ensure correct transaction behavior. Any data retrieved by one application with the intent of updating must be locked out or denied access by other applications until the update is completed (or aborted).

There are two types of locks: Slocks (shared or read locks) and Xlocks (exclusive or write locks). Here are some key points to understand about these types of locks:

1. When a transaction has a Slock on a database item, other transactions can issue Slocks on the same item, but there can be no Xlocks on that item.
2. Before a transaction can read a database item, it must be granted a Slock or Xlock on that item.
3. When a transaction has an Xlock on a database item, no other transaction can issue either a Slock or Xlock on that item.
4. Before a transaction can write to a database item, it must be granted an Xlock on that item.

Consider the example used previously. When A accesses the record for update, the DBMS must refuse all further accesses to that record until transaction A is complete (i.e., an Xlock). As the following figure shows, B’s first attempt to access the record is denied until transaction A is finished. As a result, database integrity is maintained. Unless the DBMS controls concurrent access, a multiuser database environment can create both data and retrieval integrity problems.

Valid update when concurrent accessing is not allowed

| Time | Action |  | Database record | |
| --- | --- | --- | --- | --- |
|  |  |  | Part# | Quantity |
|  |  |  | P10 | 40 |
| T1 | App A receives message for delivery of 80 units of P10 |  |  |  |
| T2 | App A reads the record for P10 | ← | P10 | 40 |
| T3 | App B receives message for sale of 20 units of P10 |  |  |  |
| T4 | App B attempts to read the record for P10 | deny | P10 | 40 |
| T5 | App A process delivery (40 + 80 = 120) |  |  |  |
| T6 | App A updates the record for P10 | → | P10 | 120 |
| T7 | App B reads the record for P10 | ← | P10 | 120 |
| T8 | App B processes the sale of (120 - 20 = 100) |  |  |  |
| T9 | App B updates the record P10 | → | P10 | 100 |

To administer locking procedures, a DBMS requires two pieces of information:

1. Whether a particular transaction will update the database;
2. Where the transaction begins and ends.

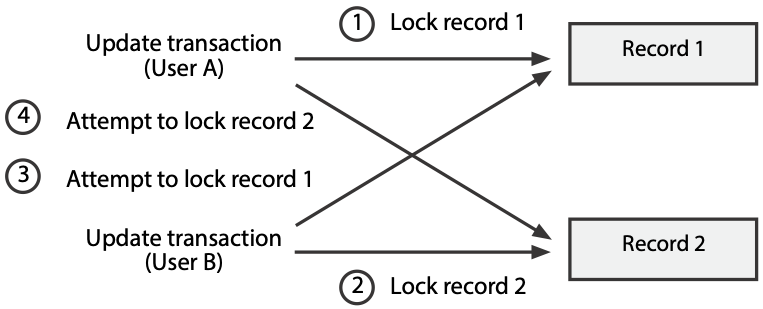
Usually, the data required by an update transaction are locked when the transaction begins and then released when the transaction is completed (i.e., committed to the database) or aborted. Locking mechanisms can operate at different levels of locking granularity: database, table, page, row, or data element. At the most precise level, a DBMS can lock individual data elements so that different update transactions can update different items in the same record concurrently. This approach increases processing overhead but provides the fewest resource conflicts. At the other end of the continuum, the DBMS can lock the entire database for each update. If there were many update transactions to process, this would be very unacceptable because of the long waiting times. Locking at the record level is the most common approach taken by DBMSs.

In most situations, applications are not concerned with locking, because it is handled entirely by the DBMS. But in some DBMSs, choices are provided to the programmer. These are primarily limited to programming language interfaces.

Resource locking solves some data and retrieval integrity problems, but it may lead to another problem, referred to as deadlock or the deadly embrace. Deadlock is an impasse that occurs because two applications lock certain resources, then request resources locked by each other. The following figure illustrates a deadlock situation. Both transactions require records 1 and 2. Transaction A first accesses record 1 and locks it. Then transaction B accesses record 2 and locks it. Next, B’s attempt to access record 1 is denied, so the application waits for the record to be released. Finally, A’s attempt to access record 2 is denied, so the application waits for the record to be released. Thus, application A’s update transaction is waiting for record 2 (locked by application B), and application B is waiting for record 1 (locked by application A). Unless the DBMS intervenes, both applications will wait indefinitely.

There are two ways to resolve deadlock: prevention and resolution. Deadlock prevention requires applications to lock in advance all records they will require. Application B would have to lock both records 1 and 2 before processing the transaction. If these records are locked, B would have to wait.

An example of deadlock



The two-phase locking protocol is a simple approach to preventing deadlocks. It operates on the notion that a transaction has a growing phase followed by a shrinking phase. During the growing phase, locks can be requested. The shrinking phase is initiated by a release statement, which means no additional locks can be requested. A release statement enables the program to signal to the DBMS the transition from requesting locks to releasing locks.

Another approach to deadlock prevention isdeadlock resolution**,** wherebythe DBMS detects and breaks deadlocks. The DBMS usually keeps a resource usage matrix, which instantly reflects which applications (e.g., update transactions) are using which resources (e.g., rows). By scanning the matrix, the DBMS can detect deadlocks as they occur. The DBMS then resolves the deadlock by backing out one of the deadlocked transactions. For example, the DBMS might release application A’s lock on record 1, thus allowing application B to proceed. Any changes made by application A up to that time (e.g., updates to record 1) would be rolled back. Application A’s transaction would be restarted when the required resources became available.

## Transaction failure and recovery

When a transaction fails, there is the danger of an inconsistent database. Transactions can fail for a variety of reasons, including

* Program error (e.g., a logic error in the code)
* Action by the transaction manager (e.g., resolution of a deadlock)
* Self-abort (e.g., an error in the transaction data means it cannot be completed)
* System failure (e.g., an operating-system bug)

If a transaction fails for any reason, then the DBMS must be able to restore the database to a correct state. To do this, two statements are required: an end of transaction (EOT) and **commit**. EOT indicates the physical end of the transaction, the last statement. Commit, an explicit statement, must occur in the transaction code before the EOT statement. The only statements that should occur between commit and EOT are database writes and lock releases.

When a transaction issues a commit, the transaction manager checks that all the necessary write-record locks for statements following the commit have been established. If these locks are not in place, the transaction is terminated; otherwise, the transaction is committed, and it proceeds to execute the database writes and release locks. Once a transaction is committed, a system problem is the only failure to which it is susceptible.

When a transaction fails, the transaction manager must take one of two corrective actions:

* If the transaction has not been committed, the transaction manager must return the database to its state prior to the transaction. That is, it must perform a rollback of the database to its most recent valid state.
* If the transaction has been committed, the transaction manager must ensure that the database is established at the correct post-transaction state. It must check that all write statements executed by the transaction and those appearing between commit and EOT have been applied. The DBMS may have to redo some writes.

# Protecting existence

One of the three database integrity outcomes is protecting the existence of the database—ensuring data are available when needed. Two strategies for protecting existence are isolation and database backup and recovery. Isolation is a preventive strategy that involves administrative procedures to insulate the physical database from destruction. Some mechanisms for doing this are keeping data in safe places, such as in vaults or underground; having multiple installations; or having security systems. For example, one organization keeps backup copies of important databases on removable magnetic disks. These are stored in a vault that is always guarded. To gain access to the vault, employees need a badge with an encoded personal voice print. Many companies have total-backup computer centers, which contain duplicate databases and documentation for system operation. If something should happen at the main center (e.g., a flood), they can be up and running at their backup center in a few hours, or even minutes in some highly critical situations. What isolation strategies do you use to protect the backup medium of your personal computer? Do you make backup files?

A study of 429 disaster declarations reported to a major international provider of disaster recovery services provides some insights as to the frequency and effects of different IT disasters. These data cover the period 1981–2000 and identify the most frequent disasters and statistics on the length of the disruption.[[16]](#footnote-16)

Most frequent IT disasters

| Category | Description |
| --- | --- |
| Disruptive act | Strikes and other intentional human acts, such as bombs or civil unrest, that are designed to interrupt normal organizational processes |
| Fire | Electrical or natural fires |
| IT failure | Hardware, software, or network problems |
| IT move/upgrade | Data center moves and CPU upgrades |
| Natural event | Earthquakes, hurricanes, severe weather |
| Power outage | Loss of power |
| Water leakage | Unintended escape of contained water (e.g., pipe leaks, main breaks) |

Days of disruption per year

| Category | Number | Minimum | Maximum | Mean |
| --- | --- | --- | --- | --- |
| Natural event | 122 | 0 | 85 | 6.38 |
| IT failure | 98 | 1 | 61 | 6.89 |
| Power outage | 67 | 1 | 20 | 4.94 |
| Disruptive act | 29 | 1 | 97 | 23.93 |
| Water leakage | 28 | 0 | 17 | 6.07 |
| Fire | 22 | 1 | 124 | 13.31 |
| IT move/upgrade | 14 | 1 | 204 | 20.93 |
| Environmental | 6 | 1 | 183 | 65.67 |
| Miscellaneous | 5 | 1 | 416 | 92.8 |
| IT capacity | 2 | 4 | 8 | 6 |
| Theft | 2 | 1 | 3 | 2 |
| Construction | 1 | 2 | 2 | 2 |
| Flood | 1 | 13 | 13 | 13 |
| IT user error | 1 | 1 | 1 | 1 |

## Backup and recovery

Database backup and recovery is a curative strategy to protect the existence of a physical database and to recreate or recover the data whenever loss or destruction occurs. The possibility of loss always exists. The use of, and choice among, backup and recovery procedures depends upon an assessment of the risk of loss and the cost of applying recovery procedures. The procedures in this case are carried out by the computer system, usually the DBMS. Data loss and damage should be anticipated. No matter how small the probability of such events, there should be a detailed plan for data recovery.

There are several possible causes for data loss or damage, which can be grouped into three categories.

### Storage-medium destruction

In this situation, a portion or all of the database is unreadable as a result of catastrophes such as power or air-conditioning failure, fire, flood, theft, sabotage, and the overwriting of disks or tapes by mistake. A more frequent cause is a disk failure. Some of the disk blocks may be unreadable as a consequence of a read or write malfunction, such as a head crash.

### Abnormal termination of an update transaction

In this case, a transaction fails part way through execution, leaving the database partially updated. The database will be inconsistent because it does not accurately reflect the current state of the business. The primary causes of an abnormal termination are a transaction error or system failure. Some operation within transaction processing, such as division by zero, may cause it to fail. A hardware or software failure will usually result in one or more active programs being aborted. If these programs were updating the database, integrity problems could result.

### Incorrect data discovered

In this situation, an update program or transaction incorrectly updated the database. This usually happens because a logic error was not detected during program testing.

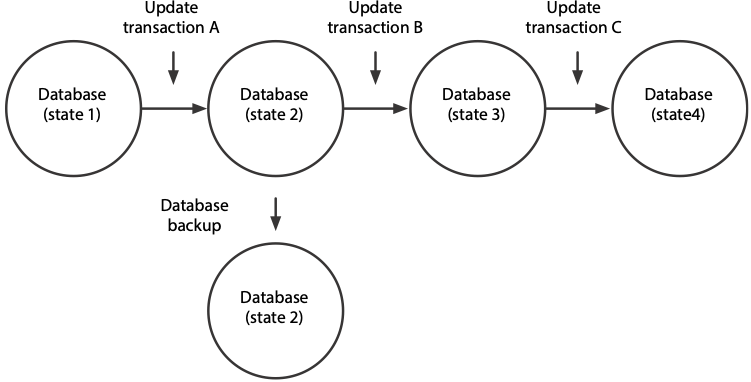
Because most organizations rely heavily on their databases, a DBMS must provide the following three mechanisms for restoring a database quickly and accurately after loss or damage:

1. Backup facilities, which create duplicate copies of the database
2. Journaling facilities, which provide backup copies or an audit trail of transactions or database changes
3. A recovery facility within the DBMS that restores the database to a consistent state and restarts the processing of transactions

Before discussing each of these mechanisms in more depth, let us review the steps involved in updating a database and how backup and journaling facilities might be integrated into this process.

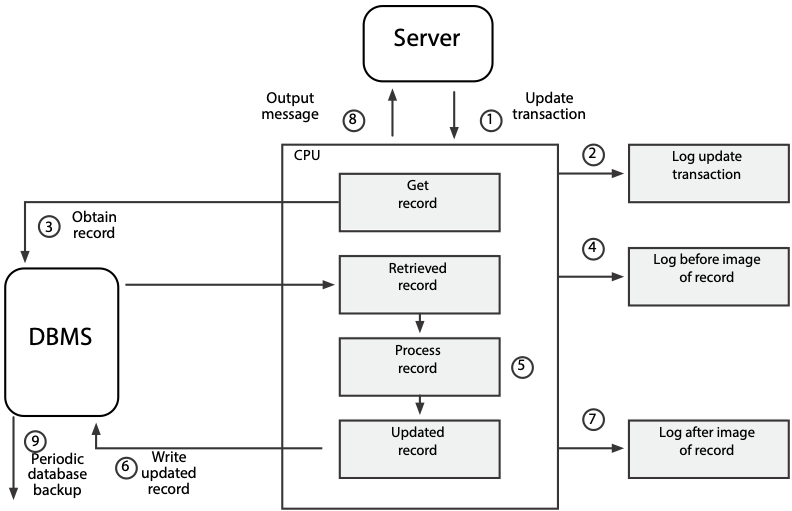
An overview of the database update process is captured in the following figure. The process can be viewed as a series of database state changes. The initial database, state 1, is modified by an update transaction, such as deleting customer Jones, creating a new state (state 2). State 1 reflects the state of the organization with Jones as a customer, while state 2 reflects the organization without this customer. Each update transaction changes the state of the database to reflect changes in organizational data. Periodically, the database is copied or backed up, possibly onto a different storage medium and stored in a secure location. The backup is made when the database is in state 2.

Database state changes as transactions are processed



A more detailed illustration of database update procedures and the incorporation of backup facilities is shown in the following figure.

Possible procedures for a database update



The updating of a single record is described below.

1. The client submits an update transaction from a workstation.
2. The transaction is edited and validated by an application program or the DBMS. If it is valid, it is logged or stored in the transaction log or journal. A journal or log is a special database or file that stores information for backup and recovery.
3. The DBMS obtains the record to be updated.
4. A copy of the retrieved record is logged to the journal file. This copy is referred to as the before image, because it is a copy of the database record before the transaction changes it.
5. The transaction is processed by changing the affected data items in the record.
6. The DBMS writes the updated record to the database.
7. A copy of the updated record is written to the journal. This copy is referred to as the after image, because it is a copy of the database record after the transaction has updated it.
8. An output message tells the application that the update has been successfully completed.
9. The database is copied periodically. This backup copy reflects all updates to the database up to the time when the copy was made. An alternative strategy to periodic copying is to maintain multiple (usually two) complete copies of the database online and update them simultaneously. This technique, known as mirroring, is discussed in Chapter 11.

In order to recover from data loss or damage, it is necessary to store backup data, such as a complete copy of the database or the necessary data to restore the accuracy of a database. Preferably, backup data should be stored on another medium and kept separate from the primary database. As the description of the update process indicates, there are several options for backup data, depending on the objective of the backup procedure.

Backup option

| Objective | Action |
| --- | --- |
| Complete copy of database | Dual recording of data (mirroring) |
| Past states of the database (also known as database dumps) | Database backup |
| Changes to the database | Before-image log or journal  After-image log or journal |
| Transactions that caused a change in the state of the database | Transaction log or journal |

Data stored for backup and recovery are generally some combination of periodic database backups, transaction logs, and before- and after-image logs. Different recovery strategies use different combinations of backup data to recover a database.

The recovery method is highly dependent on the backup strategy. The database administrator selects a strategy based on a trade-off between ease of recovery from data loss or damage and the cost of performing backup operations. For example, keeping a mirror database is more expensive then keeping periodic database backups, but a mirroring strategy is useful when recovery is needed very quickly, say in seconds or minutes. An airline reservations system could use mirroring to ensure fast and reliable recovery. In general, the cost of keeping backup data is measured in terms of interruption of database availability (e.g., time the system is out of operation when a database is being restored), storage of redundant data, and degradation of update efficiency (e.g., extra time taken in update processing to save before or after images).

## Recovery strategies

The type of recovery strategy or procedure used in a given situation depends on the nature of the data loss, the type of backup data available, and the sophistication of the DBMS’s recovery facilities. The following discussion outlines four major recovery strategies: switch to a duplicate database, backward recovery or rollback, forward recovery or roll forward, and reprocessing transactions.

The recovery procedure of switching to a duplicate database requires the maintenance of the mirror copy. The other three strategies assume a periodic dumping or backing up of the database. Periodic dumps may be made on a regular schedule, triggered automatically by the DBMS, or triggered externally by personnel. The schedule may be determined by time (hourly, daily, weekly) or by event (the number of transactions since the last backup).

### Switching to a duplicate database

This recovery procedure requires maintaining at least two copies of the database and updating both simultaneously. When there is a problem with one database, access is switched to the duplicate. This strategy is particularly useful when recovery must be accomplished in seconds or minutes. This procedure offers good protection against certain storage-medium destruction problems, such as disk failures, but none against events that damage or make both databases unavailable, such as a power failure or a faulty update program. This strategy entails additional costs in terms of doubling online storage capacity. It can also be implemented with dual computer processors, where each computer updates its copy of the database. This duplexed configuration offers greater backup protection at a greater cost.

### Backward recovery or rollback

Backward recovery (also called rollback or rolling back) is used to back out or undo unwanted changes to the database. For example, the database update procedures figure shows three updates (A, B, and C) to a database. Let’s say that update B terminated abnormally, leaving the database, now in state 3, inconsistent. What we need to do is return the database to state 2 by applying the before images to the database. Thus, we would perform a rollback by changing the database to state 2 with the before images of the records updated by transaction B.

Backward recovery reverses the changes made when a transaction abnormally terminates or produces erroneous results. To illustrate the need for rollback, consider the example of a budget transfer of $1,000 between two departments.

1. The program reads the account record for Department X and subtracts $1,000 from the account balance and updates the database.
2. The program then reads the record for Department Y and adds $1,000 to the account balance, but while attempting to update the database, the program encounters a disk error and cannot write the record.

Now the database is inconsistent. Department X has been updated, but Department Y has not. Thus, the transaction must be aborted and the database recovered. The DBMS would apply the before image to Department X to restore the account balance to its original value. The DBMS may then restart the transaction and make another attempt to update the database.

### Forward recovery or roll forward

Forward recovery (also called roll forward or bringing forward) involves recreating a database using a prior database state. Returning to the example in the database update procedures figure, suppose that state 4 of the database was destroyed and that we need to recover it. We would take the last database dump or backup (state 2) and then apply the after-image records created by update transactions B and C. This would return the database to state 4. Thus, roll forward starts with an earlier copy of the database, and by applying after images (the results of good transactions), the backup copy of the database is moved forward to a later state.

### Reprocessing transactions

Although similar to forward recovery, this procedure uses update transactions instead of after images. Taking the same example shown above, assume the database is destroyed in state 4. We would take the last database backup (state 2) and then reprocess update transactions B and C to return the database to state 4. The main advantage of using this method is its simplicity. The DBMS does not need to create an after-image journal, and there are no special restart procedures. The one major disadvantage, however, is the length of time to reprocess transactions. Depending on the frequency of database backups and the time needed to get transactions into the identical sequence as previous updates, several hours of reprocessing may be required. Processing new transactions must be delayed until the database recovery is complete.

The following table reviews the three types of data losses and the corresponding recovery strategies one could use. The major problem is to recreate a database using a backup copy, a previous state of organizational memory. Recovery is done through forward recovery, reprocessing, or switching to a duplicate database if one is available. With abnormal termination or incorrect data, the preferred strategy is backward recovery, but other procedures could be used.

What to do when data loss occurs

| Problem | Recovery procedures |
| --- | --- |
| Storage medium destruction  (database is unreadable) | \*Switch to a duplicate database—this can be transparent with RAID  Forward recovery  Reprocessing transactions |
| Abnormal  termination of an update transaction (transaction error or system failure) | \*Backward recovery  Forward recovery or reprocessing transactions—bring forward to the state just before termination of the transaction |
| Incorrect data detected (database has been incorrectly updated) | \*Backward recovery  Reprocessing transactions (excluding those from the update program that created the incorrect data) |
| \* Preferred strategy |  |

## Use of recovery procedures

Usually the person doing a query or an update is not concerned with backup and recovery. Database administration personnel often implement strategies that are automatically carried out by the DBMS. ANSI has defined standards governing SQL processing of database transactions that relate to recovery. Transaction support is provided through the use of the two SQL statements COMMIT and ROLLBACK. These commands are employed when a procedural programming language such as Java is used to update a database. They are illustrated in the chapter on SQL and Java.

Skill builder

An Internet bank with more than 10 million customers has asked for your advice on developing procedures for protecting the existence of its data. What would you recommend?

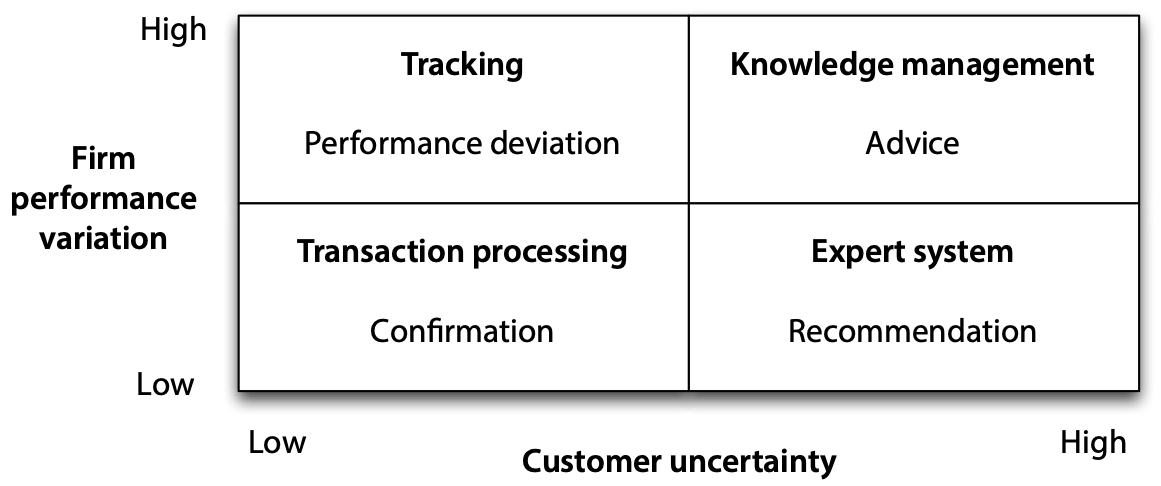
# Maintaining data quality

The second integrity goal is to maintain data quality, which typically means keeping data accurate, complete, and current. Data are high-quality if they fit their intended uses in operations, decision making, and planning. They are fit for use if they are free of defects and possess desired features.[[17]](#footnote-17) The preceding definition implicitly recognizes that data quality is determined by the customer. It also implies that data quality is relative to a task. Data could be high-quality for one task and low-quality for another. The data provided by a flight-tracking system are very useful when you are planning to meet someone at the airport, but not particularly helpful for selecting a vacation spot. Defect-free data are accessible, accurate, timely, complete, and consistent with other sources. Desirable features include relevance, comprehensiveness, appropriate level of detail, easy-to-navigate source, high readability, and absence of ambiguity.

Poor-quality data have several detrimental effects. Customer service decreases when there is dissatisfaction with poor and inaccurate information or a lack of appropriate information. For many customers, information is the heart of customer service, and they lose confidence in firms that can’t or don’t provide relevant information. Bad data interrupt information processing because they need to be corrected before processing can continue. Poor-quality data can lead to a wrong decision because inaccurate inferences and conclusions are made.

As we have stated previously, data quality varies with circumstances, and the model in the following figure will help you to understand this linkage. By considering variations in a customer’s uncertainty about a firm’s products and a firm’s ability to deliver consistently, we arrive at four fundamental strategies for customer-oriented data quality.

Customer-oriented data quality strategies



Transaction processing: When customers know what they want and firms can deliver consistently, customers simply want fast and accurate transactions and data confirming details of the transaction. Most banking services fall into this category. Customers know they can withdraw and deposit money, and banks can perform reliably.

Expert system: In some circumstances, customers are uncertain of their needs. For instance, Vanguard offers personal investors a choice from more than 150 mutual funds. Most prospective investors are confused by such a range of choices, and Vanguard, by asking a series of questions, helps prospective investors narrow their choices and recommends a small subset of its funds. A firm’s recommendation will vary little over time because the characteristics of a mutual fund (e.g., intermediate tax-exempt bond) do not change.

Tracking: Some firms operate in environments where they don’t have a lot of control over all the factors that affect performance. Handling more than 2,200 take-offs and landings and over 250,000 passengers per day,[[18]](#footnote-18) Atlanta’s Hartsfield Airport becomes congested when bad weather, such as a summer thunderstorm, slows down operations. Passengers clearly know what they want—data on flight delays and their consequences. They assess data quality in terms of how well data tracks delays and notifies them of alternative travel arrangements.

Knowledge management: When customers are uncertain about their needs for products delivered by firms that don’t perform consistently, they seek advice from knowledgeable people and organizations. Data quality is judged by the soundness of the advice received. Thus, a woman wanting a custom-built house would likely seek the advice of an architect to select the site, design the house, and supervise its construction, because architects are skilled in eliciting clients’ needs and knowledgeable about the building industry.

An organization’s first step toward improving data quality is to determine in which quadrant it operates so it can identify the critical information customers expect. Of course, data quality in many situations will be a mix of expectations. The mutual fund will be expected to confirm fund addition and deletion transactions. However, a firm must meet its dominant goal to attract and retain customers.

A firm will also need to consider its dominant data quality strategy for its internal customers, and the same general principles illustrated by the preceding figure can be applied. In the case of internal customers, there can be varying degrees of uncertainty as to what other organizational units can do for them, and these units will vary in their ability to perform consistently for internal customers. For example, consulting firms develop centers of excellence as repositories of knowledge on a particular topic to provide their employees with a source of expertise. These are the internal equivalent of knowledge centers for external customers.

Once they have settled on a dominant data quality strategy, organizations need a corporate-wide approach to data quality, just like product and service quality. There are three generations of data quality:

* First generation: Errors in existing data stores are found and corrected. This data cleansing is necessary when much of the data is captured by manual systems.
* Second generation: Errors are prevented at the source. Procedures are put in place to capture high-quality data so that there is no need to cleanse it later. As more data are born digital, this approach becomes more feasible. Thus, when customers enter their own data or barcodes are scanned, the data should be higher-quality than when entered by a data-processing operator.
* Third generation: Defects are highly unlikely. Data capture systems meet six-sigma standards (3.4 defects per million), and data quality is no longer an issue.

### Skill builder

1. A consumer electronics company with a well-respected brand offers a wide range of products. For example, it offers nine world-band radios and seven televisions. What data quality strategy would you recommend?
2. What data quality focus would you recommend for a regulated natural gas utility?
3. A telephone company has problems in estimating how long it takes to install DSL in homes. Sometimes it takes less than an hour and other times much longer. Customers are given a scheduled appointment and many have to make special arrangements so that they are home when the installation person arrives. What data might these customers expect from the telephone company, and how would they judge data quality?

## Dimensions

The many writers on quality all agree on one thing—quality is multidimensional. Data quality also has many facets, and these are presented in the following table. The list also provides data managers with a checklist for evaluating overall high-quality data. Organizations should aim for a balanced performance across all dimensions because failure in one area is likely to diminish overall data quality.

The dimensions of data quality

| Dimension | Conditions for high-quality data |
| --- | --- |
| Accuracy | Data values agree with known correct values. |
| Completeness | Values for all reasonably expected attributes are available. |
| Representation consistency | Values for a particular attribute have the same representation across all tables (e.g., dates). |
| Organizational consistency | There is one organization wide table for each entity and one organization wide domain for each attribute. |
| Row consistency | The values in a row are internally consistent (e.g., a home phone number’s area code is consistent with a city’s location). |
| Timeliness | A value’s recentness matches the needs of the most time-critical application requiring it. |
| Stewardship | Responsibility has been assigned for managing data. |
| Sharing | Data sharing is widespread across organizational units. |
| Fitness | The format and presentation of data fit each task for which they are required. |
| Interpretation | Clients correctly interpret the meaning of data elements. |
| Flexibility | The content and format of presentations can be readily altered to meet changing circumstances. |
| Precision | Data values can be conveniently formatted to the required degree of accuracy. |
| International | Data values can be displayed in the measurement unit of choice (e.g., kilometers or miles). |
| Accessibility | Authorized users can readily access data values through a variety of devices from a variety of locations. |
| Security and privacy | Data are appropriately protected from unauthorized access. |
| Continuity | The organization continues to operate in spite of major disruptive events. |
| Granularity | Data are represented at the lowest level necessary to support all uses (e.g., hourly sales). |
| Metadata | There is ready access to accurate data about data. |

Skill builder

1. What level of granularity of sales data would you recommend for an online retailer?
2. What level of data quality completeness might you expect for a university’s student DBMS?

## DBMS and data quality

To assist data quality, functions are needed within the DBMS to ensure that update and insert actions are performed by authorized persons in accordance with stated rules or integrity constraints, and that the results are properly recorded. These functions are accomplished by update authorization, data validation using integrity constraints, and concurrent update control. Each of these functions is discussed in turn.

### Update authorization

Without proper controls, update transactions can diminish the quality of a database. Unauthorized users could sabotage a database by entering erroneous values. The first step is to ensure that anyone who wants to update a database is authorized to do so. Some responsible person—usually the database owner or database administrator—must tell the DBMS who is permitted to initiate particular database operations. The DBMS must then check every transaction to ensure that it is authorized. Unauthorized access to a database exposes an organization to many risks, including fraud and sabotage.

Update authorization is accomplished through the same access mechanism used to protect confidentiality. We will discuss access control more thoroughly later in this chapter. In SQL, access control is implemented through GRANT, which gives a user a privilege, and REVOKE, which removes a privilege. (These commands are discussed in Chapter 10.) A control mechanism may lump all update actions into a single privilege or separate them for greater control. In SQL, they are separated as follows:

* UPDATE (privilege to change field values using UPDATE; this can be column specific)
* DELETE (privilege to delete records from a table)
* INSERT (privilege to insert records into a table)

Separate privileges for each of the update commands allow tighter controls on certain update actions, such as updating a salary field or deleting records.

### Data validation using integrity constraints

Once the update process has been authorized, the DBMS must ensure that a database is accurate and complete before any updates are applied. Consequently, the DBMS needs to be aware of any integrity constraints or rules that apply to the data. For example, the qdel table in the relational database described previously would have constraints such as

* Delivery number (delno) must be unique, numeric, and in the range 1–99999.
* Delivered quantity (delqty) must be nonzero.
* Item name (itemname) must appear in the qitem table.
* Supplier code (splno) must appear in the qspl table.

Once integrity constraints are specified, the DBMS must monitor or validate all insert and update operations to ensure that they do not violate any of the constraints or rules.

The key to updating data validation is a clear definition of valid and invalid data. Data validation cannot be performed without integrity constraints or rules. A person or the DBMS must know acceptable data format, valid values, and procedures to invoke to determine validity. All data validation is based on a prior expression of integrity constraints.

Data validation may not always produce error-free data, however. Sometimes integrity constraints are unknown or are not well defined. In other cases, the DBMS does provide a convenient means for expressing and performing validation checks.

Based on how integrity constraints have been defined, data validation can be performed outside the DBMS by people or within the DBMS itself. External validation is usually done by reviewing input documents before they are entered into the system and by checking system outputs to ensure that the database was updated correctly. Maintaining data quality is of paramount importance, and data validation preferably should be handled by the DBMS as much as possible rather than by the application, which should handle the exceptions and respond to any failed data validation checks.

Integrity constraints are usually specified as part of the database definition supplied to the DBMS. For example, the primary-key uniqueness and referential integrity constraints can be specified within the SQL CREATE statement. DBMSs generally permit some constraints to be stored as part of the database schema and are used by the DBMS to monitor all update operations and perform appropriate data validation checks. Any given database is likely to be subject to a very large number of constraints, but not all of these can be automatically enforced by the DBMS. Some will need to be handled by application programs.

The general types of constraints applied to a data item are outlined in the following table. Not all of these necessarily would be supported by a DBMS, and a particular database may not use all types.

Types of data items in integrity constraint

| Type of integrity constraint | Explanation | Example |
| --- | --- | --- |
| type | Validating a data item value against a specified data type | Supplier number is numeric. |
| size | Defining and validating the minimum and maximum size of a data item | Delivery number must be at least 3 digits, and at most 5. |
| values | Providing a list of acceptable values for a data item | Item colors must match the list provided. |
| range | Providing one or more ranges within which the data item must lie | Employee numbers must be in the range 1–100. |
| pattern | Providing a pattern of allowable characters that define permissible formats for data values | Department phone number must be of the form 542-nnnn (stands for exactly four decimal digits). |
| procedure | Providing a procedure to be invoked to validate data items | A delivery must have valid item name, department, and supplier values before it can be added to the database (tables are checked for valid entries). |
| Conditional | Providing one or more conditions to apply against data values | If item type is “Y,” then color is null. |
| Not null  (mandatory) | Indicating whether the data item value is mandatory (not null) or optional; the not-null option is required for primary keys | Employee number is mandatory. |
| Unique | Indicating whether stored values for this data item must be compared to other values of the item within the same table | Supplier number is unique. |

As mentioned, integrity constraints are usually specified as part of the database definition supplied to the DBMS. The following table contains some typical specifications of integrity constraints for a relational DBMS.

| Examples | Explanation |
| --- | --- |
| CREATE TABLE stock (  stkcode CHAR(3),  …,  natcode CHAR(3),  PRIMARY KEY(stkcode),  CONSTRAINT fk\_stock\_nation  FOREIGN KEY(natcode)  REFERENCES nation(natcode)  ON DELETE RESTRICT); | Column stkcode must always have 3 or fewer alphanumeric characters, and stkcode must be unique because it is a primary key.  Column natcode must be assigned a value of 3 or less alphanumeric characters and must exist as the primary key of the nation.  Do not allow the deletion of a row in nation while there still exist rows in stock containing the corresponding value of natcode. |

Examples of integrity constraints

Data quality control does not end with the application of integrity constraints. Whenever an error or unusual situation is detected by the DBMS, some form of response is required. Response rules need to be given to the DBMS along with the integrity constraints. The responses can take many different forms, such as abort the entire program, reject entire update transaction, display a message and continue processing, or let the DBMS attempt to correct the error. The response may vary depending on the type of integrity constraint violated. If the DBMS does not allow the specification of response rules, then it must take a default action when an error is detected. For example, if alphabetic data are entered in a numeric field, most DBMSs will have a default response and message (e.g., nonnumeric data entered in numeric field). In the case of application programs, an error code is passed to the program from the DBMS. The program would then use this error code to execute an error-handling procedure.

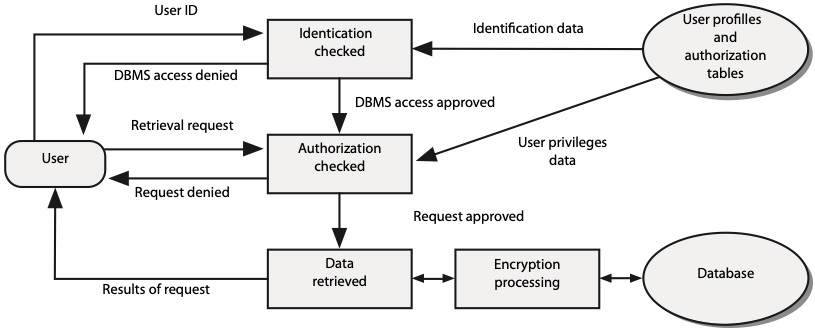
# Ensuring  confidentiality

Thus far, we have discussed how the first two goals of data integrity can be accomplished: Data are available when needed (protecting existence); data are accurate, complete, and current (maintaining quality). This section deals with the final goal: ensuring confidentiality or data security. Two DBMS functions—access control and encryption—are the primary means of ensuring that the data are accessed only by those authorized to do so. We begin by discussing an overall model of data security.

## General model of data security

The following figure depicts the two functions for ensuring data confidentiality: access control and encryption. Access control consists of two basic steps—identification and authorization. Once past access control, the user is permitted to access the database. Access control is applied only to the established avenues of entry to a database. Clever people, however, may be able to circumvent the controls and gain unauthorized access. To counteract this possibility, it is often desirable to hide the meaning of stored data by encrypting them, so that it is impossible to interpret their meaning. Encrypted data are stored in a transformed or coded format that can be decrypted and read only by those with the appropriate key.

A general model of data security



Now let us walk through the figure in detail.

1. A client must be identified and provide additional information required to authenticate this identification (e.g., an account name and password). Client profile information (e.g., a password or a voice print) is used to verify or authenticate a connection.
2. Having authenticated the client, the authorization step is initiated by a request (retrieve or update database). The previously stored client authorization rules (what data each client can access and the authorized actions on those data) are checked to determine whether the client has the right or privilege to access the requested data. (The previously stored client’s privileges are created and maintained by an authorized person, database owner, or administrator.) A decision is made to permit or deny the execution of the request. If access is permitted, the transaction is processed against the database.
3. Data are encrypted before storage, and retrieved data are decrypted before presentation.

## Data access control

Data access control begins with identification of an organizational entity that can access the database. Examples are individuals, departments, groups of people, transactions, terminals, and application programs. Valid combinations may be required, for example, a particular person entering a certain transaction at a particular terminal. A user identification (often called ***userid***) is the first piece of data the DBMS receives from the subject. It may be a name or number. The user identification enables the DBMS to locate the corresponding entry in the stored user profiles and authorization tables (see preceding figure).

Taking this information, the DBMS goes through the process of authentication. The system attempts to match additional information supplied by the client with the information previously stored in the client’s profile. The system may perform multiple matches to ensure the identity of the client (see the following table for the different types). If all tests are successful, the DBMS assumes that the subject is an authenticated client.

Authenticating mechanisms1

| Class | Examples |
| --- | --- |
| Something a person knows: remembered information | Name, account number, password |
| Something the person has: possessed object | Badge, plastic card, key |
| Something the person is: personal characteristic | Fingerprint, voiceprint, signature, hand size |

Many systems use remembered information to control access. The problem with such information is that it does not positively identify the client. Passwords have been the most widely used form of access control. If used correctly, they can be very effective. Unfortunately, people leave them around where others can pick them up, allowing unauthorized people to gain access to databases.

To deal with this problem, organizations are moving toward using personal characteristics and combinations of authenticating mechanisms to protect sensitive data. Collectively, these mechanisms can provide even greater security. For example, access to a large firm’s very valuable marketing database requires a smart card and a fingerprint, a combination of personal characteristic and a possessed object. The database can be accessed through only a few terminals in specific locations, an isolation strategy. Once the smart card test is passed, the DBMS requests entry of other remembered information—password and account number—before granting access.

Data access authorization is the process of permitting clients whose identity has been authenticated to perform certain operations on certain data objects in a shared database. The authorization process is driven by rules incorporated into the DBMS. Authorization rules are in a table that includes subjects, objects, actions, and constraints for a given database. An example of such a table is shown in the following table . Each row of the table indicates that a particular subject is authorized to take a certain action on a database object, perhaps depending on some constraint. For example, the last entry of the table indicates that Brier is authorized to delete supplier records with no restrictions.

Sample authorization table

| Subject/Client | Action | Object | Constraint |
| --- | --- | --- | --- |
| Accounting department | Insert | Supplier table | None |
| Purchase department clerk | Insert | Supplier table | If quantity < 200 |
| Purchase department supervisor | Insert | Delivery table | If quantity >= 200 |
| Production department | Read | Delivery table | None |
| Todd | Modify | Item table | Type and color only |
| Order-processing program | Modify | Sale table | None |
| Brier | Delete | Supplier table | None |

We have already discussed subjects, but not objects, actions, and constraints. Objects are database entities protected by the DBMS. Examples are databases, views, files, tables, and data items. In the preceding table, the objects are all tables. A view is another form of security. It restricts the client’s access to a database. Any data not included in a view are unknown to the client. Although views promote security, several persons may share a view or unauthorized persons may gain access. Thus, a view is another object to be included in the authorization process. Typical actions on objects are shown in the table: read, insert, modify, and delete. Constraints are particular rules that apply to a subject-action-object relationship.

### Implementing authorization rules

Most contemporary DBMSs do not implement the complete authorization table shown in the table. Usually, they implement a simplified version. The most common form is an authorization table for subjects with limited applications of the constraint column. Let us take the granting of table privileges, which are needed in order to authorize subjects to perform operations on both tables and views .

Authorization commands

| SQL Command | Result |
| --- | --- |
| SELECT | Permission to retrieve data |
| UPDATE | Permission to change data; can be column specific |
| DELETE | Permission to delete records or tables |
| INSERT | Permission to add records or tables |

The GRANT and REVOKE SQL commands discussed in Chapter 10 are used to define and delete authorization rules. Some examples:

GRANT SELECT ON qspl TO vikki;

GRANT SELECT, UPDATE (splname) ON qspl TO huang;

GRANT ALL PRIVILEGES ON qitem TO vikki;

GRANT SELECT ON qitem TO huang;

The GRANT commands have essentially created two authorization tables, one for Huang and the other for Vikki. These tables illustrate how most current systems create authorization tables for subjects using a limited set of objects (e.g., tables) and constraints.

A sample authorization table

| Client | Object (table) | Action | Constraint |
| --- | --- | --- | --- |
| vikki | qspl | SELECT | None |
| vikki | qitem | UPDATE | None |
| vikki | qitem | INSERT | None |
| vikki | qitem | DELETE | None |
| vikki | qitem | SELECT | None |
| huang | qspl | SELECT | None |
| huang | qspl | UPDATE | splname only |
| huang | qitem | SELECT | None |

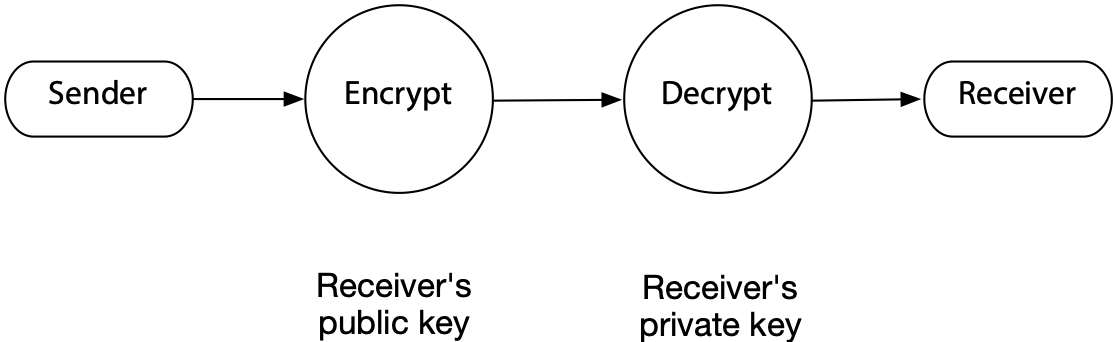
Because authorization tables contain highly sensitive data, they must be protected by stringent security rules and encryption. Normally, only selected persons in data administration have authority to access and modify them.

## Encryption

Encryption techniques complement access control. As the preceding figure illustrates, access control applies only to established avenues of access to a database. There is always the possibility that people will circumvent these controls and gain unauthorized access to a database. To counteract this possibility, encryption can be used to obscure or hide the meaning of data. Encrypted data cannot be read by an intruder unless that person knows the method of encryption and has the key. Encryption is any transformation applied to data that makes it difficult to extract meaning. Encryption transforms data into cipher text or disguised information, and decryption reconstructs the original data from cipher text.

Public-key encryption is based on a pair of private and public keys. A person’s public key can be freely distributed because it is quite separate from his or her private key. To send and receive messages, communicators first need to create private and public keys and then exchange their public keys. The sender encodes a message with the intended receiver’s public key, and upon receiving the message, the receiver applies her private key. The receiver’s private key, the only one that can decode the message, must be kept secret to provide secure message exchanging.

Public-key encryption

In a DBMS environment, encryption techniques can be applied to transmitted data sent over communication lines to and from devices, or between computers, and to all highly sensitive stored data in active databases or their backup versions. Some DBMS products include routines that automatically encrypt sensitive data when they are stored or transmitted over communication channels. Other DBMS products provide exits that allow users to code their own encryption routines. Encrypted data may also take less storage space because they are often compressed.

Skill builder

A university has decided that it will e-mail students their results at the end of each semester. What procedures would you establish to ensure that only the designated student opened and viewed the e-mail?

## Monitoring activity

Sometimes no single activity will be detected as an example of misuse of a system. However, examination of a pattern of behavior may reveal undesirable behavior (e.g., persistent attempts to log into a system with a variety of userids and passwords). Many systems now monitor all activity using audit trail analysis. A time- and date-stamped audit trail of all system actions (e.g., database accesses) is maintained. This audit log is dynamically analyzed to detect unusual behavior patterns and alert security personnel to possible misuse.

A form of misuse can occur when an authorized user violates privacy rules by using a series of authorized queries to gain access to private data. For example, some systems aim to protect individual privacy by restricting authorized queries to aggregate functions (e.g., AVG and COUNT). Since it is impossible to do non-aggregate queries, this approach should prevent access to individual-level data, which it does at the single-query level. However, multiple queries can be constructed to circumvent this restriction.

Assume we know that a professor in the IS department is aged 40 to 50, is single, and attended the University of Minnesota. Consider the results of the following set of queries.[[19]](#footnote-19)

SELECT COUNT(\*) from faculty

WHERE dept = 'MIS'

AND age >= 40 AND age <= 50;

|  |
| --- |
| 10 |

SELECT COUNT(\*) FROM faculty

WHERE dept = 'MIS'

AND age >= 40 AND age <= 50

AND degree\_from = 'Minnesota';

|  |
| --- |
| 2 |

SELECT COUNT(\*) FROM faculty

WHERE = 'MIS'

AND age >= 40 AND age <= 50;

AND degree\_from = 'Minnesota'

AND marital\_status = 'S';

|  |
| --- |
| 1 |

SELECT AVG(salary) FROM faculty

WHERE dept = 'MIS'

AND age >= 40 AND age <= 50

AND degree\_from = 'Minnesota'

AND marital\_status = 'S';

|  |
| --- |
| 85000 |

The preceding set of queries, while all at the aggregate level, enables one to deduce the salary of the professor. This is an invasion of privacy and counter to the spirit of the restriction queries to aggregate functions. An audit trail should detect such tracker **queries**,one or more authorized queries that collectively violate privacy. One approach to preventing tracker queries is to set a lower bound on the number of rows on which a query can report.

## Summary

The management of organizational data is driven by the joint goals of availability and integrity. Availability deals with making data available to whoever needs them, whenever and wherever they need them, and in a meaningful form. Maintaining integrity implies protecting existence, maintaining quality, and ensuring confidentiality. There are three strategies for maintaining data integrity: legal, administrative, and technical. A consistent database is one in which all data integrity constraints are satisfied.

A transaction must be entirely completed or aborted before there is any effect on the database. Transactions are processed as logical units of work to ensure data integrity. The transaction manager is responsible for ensuring that transactions are correctly recorded.

Concurrent update control focuses on making sure updated results are correctly recorded in a database. To prevent loss of updates and inconsistent retrieval results, a DBMS must incorporate a resource-locking mechanism. Deadlock is an impasse that occurs because two users lock certain resources, then request resources locked by each other. Deadlock prevention requires applications to lock all required records at the beginning of the transaction. Deadlock resolution uses the DBMS to detect and break deadlocks.

Isolation is a preventive strategy that involves administrative procedures to insulate the physical database from destruction. Database backup and recovery is a curative strategy that protects an existing database and recreates or recovers the data whenever loss or destruction occurs. A DBMS needs to provide backup, journaling, and recovery facilities to restore a database to a consistent state and restart the processing of transactions. A journal or log is a special database or file that stores information for backup and recovery. A before image is a copy of a database record before a transaction changes the record. An after image is a copy of a database record after a transaction has updated the record.

In order to recover from data loss or damage, it is necessary to store redundant, backup data. The recovery method is highly dependent on the backup strategy. The cost of keeping backup data is measured in terms of interruption of database availability, storage of redundant data, and degradation of update efficiency. The four major recovery strategies are switching to a duplicate database, backward recovery or rollback, forward recovery or roll forward, and reprocessing transactions. Database administration personnel often implement recovery strategies automatically carried out by the DBMS. The SQL statements COMMIT and ROLLBACK are used with a procedural programming language for implementing recovery procedures.

Maintaining quality implies keeping data accurate, complete, and current. The first step is to ensure that anyone wanting to update a database is required to have authorization. In SQL, access control is implemented through GRANT and REVOKE. Data validation cannot be performed without integrity constraints or rules. Data validation can be performed external to the DBMS by personnel or within the DBMS based on defined integrity constraints. Because maintaining data quality is of paramount importance, it is desirable that the DBMS handles data validation rather than the application. Error response rules need to be given to the DBMS along with the integrity constraints.

Two DBMS functions, access control and encryption, are the primary mechanisms for ensuring that the data are accessed only by authorized persons. Access control consists of identification and authorization. Data access authorization is the process of permitting users whose identities have been authenticated to perform certain operations on certain data objects. Encrypted data cannot be read by an intruder unless that person knows the method of encryption and has the key. In a DBMS environment, encryption can be applied to data sent over communication lines between computers and data storage devices.

Database activity is monitored to detect patterns of activity indicating misuse of the system. An audit trail is maintained of all system actions. A tracker query is a series of aggregate function queries designed to reveal individual-level data.

## Key terms and concepts

ACID

Administrative strategies

After image

All-or-nothing rule

Atomicity

Audit trail analysis

Authentication

Authorization

Backup

Before image

COMMIT

Concurrent update control

Consistency

Data access control

Data availability

Data quality

Data security

Database integrity

Deadlock prevention

Deadlock resolution

Deadly embrace

Decryption

Durability

Encryption

Ensuring confidentiality

GRANT

Integrity constraint

Isolation

Journal

Legal strategies

Locking

Maintaining quality

Private key

Protecting existence

Public-key encryption

Recovery

Reprocessing

REVOKE

Roll forward

ROLLBACK

Rollback

Serializability

Slock

Technical strategies

Tracker query

Transaction

Transaction atomicity

Transaction manager

Two-phase locking protocol

Validation

Xlock

## Exercises

1. What are the three goals of maintaining organizational memory integrity?
2. What strategies are available for maintaining data integrity?
3. A large corporation needs to operate its computer system continuously to remain viable. It currently has data centers in Miami and San Francisco. Do you have any advice for the CIO?
4. An investment company operates out of a single office in Boston. Its business is based on many years of high-quality service, honesty, and reliability. The CEO is concerned that the firm has become too dependent on its computer system. If some disaster should occur and the firm’s databases were lost, its reputation for reliability would disappear overnight and so would many of its customers in this highly competitive business. What should the firm do?
5. What mechanisms should a DBMS provide to support backup and recovery?
6. What is the difference between a before imageandanafter image?
7. A large organization has asked you to advise on backup and recovery procedures for its weekly, batch payroll system. They want reliable recovery at the lowest cost. What would you recommend?
8. An online information service operates globally and prides itself on its uptime of 99.98 percent. What sort of backup and recovery scheme is this firm likely to use? Describe some of the levels of redundancy you would expect to find.
9. The information systems manager of a small manufacturing company is considering the backup strategy for a new production planning database. The database is used every evening to create a plan for the next day’s production. As long as the production plan is prepared before 6 a.m. the next day, there is no impact upon plant efficiency. The database is currently 200 Mbytes and growing about 2 percent per year. What backup strategy would you recommend and why?
10. How do backward recovery and forward recovery differ?
11. What are the advantages and disadvantages of reprocessing transactions?
12. When would you use ROLLBACK in an application program?
13. When would you use COMMIT in an application program?
14. Give three examples of data integrity constraints.
15. What is the purpose of locking?
16. What is the likely effect on performance between locking at a row compared to locking at a page?
17. What is a deadly embrace? How can it be avoided?
18. What are three types of authenticating mechanisms?
19. Assume that you want to discover the grade point average of a fellow student. You know the following details of this person. She is a Norwegian citizen who is majoring in IS and minoring in philosophy. Write one or more aggregate queries that should enable you to determine her GPA.
20. What is encryption?
21. What are the disadvantages of the data encryption standard (DES)?
22. What are the advantages of public-key encryption?
23. A national stock exchange requires listed companies to transmit quarterly reports to its computer center electronically. Recently, a hacker intercepted some of the transmissions and made several hundred thousand dollars because of advance knowledge of one firm’s unexpectedly high quarterly profits. How could the stock exchange reduce the likelihood of this event?

24. Data Administration

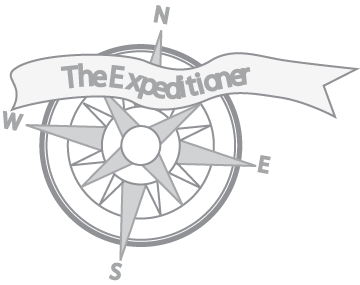
Bad administration, to be sure, can destroy good policy; but good administration can never save bad policy.

Adlai Stevenson, speech given in Los Angeles, September 11, 1952

## Learning objectives

Students completing this chapter will

* understand the role of the Chief Data Officer (CDO);
* understand the importance and role of data administration;
* understand how system-level data administration functions are used to manage a database environment successfully;
* understand how project-level data administration activities support the development of a database system;
* understand what skills data administration requires and why it needs a balance of people, technology, and business skills to carry out its roles effectively;
* understand how computer-based tools can be used to support data administration activities;
* understand the management issues involved in initiating, staffing, and locating data administration organizationally.

The Tahiti tourist resort had been an outstanding success for The Expeditioner. Located 40 minutes from Papeete, the capital city, on a stretch of tropical forest and golden sand, the resort had soon become the favorite meeting place for The Expeditioner’s board. No one objected to the long flight to French Polynesia. The destination was well worth the journey, and The Expeditioner had historic ties to that part of the South Pacific. Early visitors to Tahiti, James Cook and William Bligh, had been famous customers of The Expeditioner in the eighteenth century. Before Paul Gauguin embarked on his journey to paint scenes of Tahiti, he had purchased supplies from L’Explorateur, now the French division of The Expeditioner.

Although The Expeditioner is very successful, there are always problems for the board to address. A number of board members are very concerned by the seeming lack of control over the various database systems that are vital to the firm’s profitability. Recently, there had been a number of incidents that had underscored the problem. Purchasing had made several poor decisions. For example, it had ordered too many parkas for the North American stores and had to discount them heavily to sell all the stock. The problem was traced to poor data standards and policies within Sales. In another case, Personnel and Marketing had been squabbling for some time over access to the personnel database. Personnel claimed ownership of the data and was reluctant to share data with Marketing, which wanted access to some of the data to support its new incentive program. In yet another incident, a new database project for the Travel Division had been seriously delayed when it was discovered that the Travel Division’s development team was planning to implement a system incompatible with The Expeditioner’s existing hardware and software.

After the usual exchange of greetings and a presentation of the monthly financial report, Alice forthrightly raised the database problem. “We all know that we depend on information technology to manage The Expeditioner,” she began as she glanced at her notes on her tablet. “The Information Systems department does a great job running the computers, building new systems, and providing us with excellent service, but,” she stressed, “we seem to be focusing on managing the wrong things. We should be managing what really matters: the data we need to run the business. Data errors, internecine fighting over data, and project delays are costly. Our present system for managing data is fragmented. We don’t have anyone or any group who manages data centrally. It is critical that we develop an action plan for the organizational management of data.” Pointing to Bob, she continued, “I have invited Bob to brief us on data administration and present his proposal for solving our data management problem. It’s all yours, Bob.”

# Introduction

In the information age, data are the lifeblood of every organization and need to be properly managed to retain their value to the organization. The importance of data as a key organizational resource has been emphasized throughout this book. Data administration is the management of organizational data stores.

Information technology permits organizations to capture, organize, and maintain a greater variety of data. These data can be hard (e.g., financial or production figures) or soft (e.g., management reports, correspondence, voice conversations, and video). If these data are to be used in the organization, they must be managed just as diligently as accounting information. Data administration is the common term applied to the task of managing organizational memory. Although common, basic management principles apply to most kinds of organizational data stores, the discussion in this chapter refers primarily to databases.

## Why manage data?

Data are constantly generated in every act and utterance of every stakeholder (employee, shareholder, customer, or supplier) in relation to the organization. Some of these data are formal and structured, such as invoices, grade sheets, or bank withdrawals. A large amount of relatively unstructured data is generated too, such as tweets, blogs, and Facebook comments from customers. Much of the unstructured data generated within and outside the organizations are frequently captured but rarely analyzed deeply.

Organization units typically begin maintaining systematic records for data most likely to impinge on their performance. Often, different departments or individuals maintain records for the same data. As a result, the same data may be used in different ways by each department, and so each of them may adopt a different system of organizing its data. Over time, an organization accumulates a great deal of redundant data which demands considerable, needless administrative overhead for its maintenance. Inconsistencies may begin to emerge between the various forms of the same data. A department may incorrectly enter some data, which could result in embarrassment at best or a serious financial loss for the organization at worst.

When data are fragmented across several departments or individuals, and especially when there is personnel turnover, data may not be accessible when most needed. This is nearly as serious a problem as not having any data. Yet another motivation is that effective data management can greatly simplify and assist in the identification of new information system application development opportunities. Also, poor data management can result in breaches of security. Valuable information may be revealed to competitors or antagonists.

In summary, the problems arising from poor data management are:

* The same data may be represented through multiple, inconsistent definitions.
* There may be inconsistencies among different representations.
* Essential data may be missing from the database.
* Data may be inaccurate or incomplete.
* Some data may never be captured for the database and thus are effectively lost to the organization.
* There may be no way of knowing how to locate data when they are needed.

The overall goal of data administration is to prevent the occurrence of these problems by enabling users to access the data they need in the format most suitable for achieving organizational goals and by ensuring the integrity of organizational databases.

# The Chief Data Officer

Firms are increasingly recognizing the importance of data to organizational performance, particularly with the attention given to big data in the years following 2010. As a result, some have created the new C-level position of Chief Data Officer (CDO), who is responsible for the strategic management of data systems and ensuring that the organization fully seizes data-driven opportunities to create new business, reduce costs, and increase revenues. The CDO assists the top management team in gaining full value from data, a key strategic asset.[[20]](#footnote-20) In 2003, Capital One was perhaps to first firm to appoint a CDO. Other early creators of this new executive role were Yahoo! and Microsoft Germany. The US Federal Communications Commission (FCC) has appointed a CDO for each of its 11 major divisions. Many firms report plans to create data stewards and CDOs. Given the strategic nature of data for business, it is not surprising that one study reports that 30 percent of CDOs report to the CEO, with another 20% reporting to the COO.

## CDO role dimensions

Three dimensions of the CDO role have been identified and described (see the following figure), namely, collaboration direction (inward or outward), data management focus (traditional transaction or big data, and value orientation (service or strategy). We now discuss each of these dimensions.

# 

The three dimensions of the CDO role

### Inward vs. outward collaboration

A CDO can focus collaborative efforts inward or outward. An inward emphasis might mean working with production to improve the processes for capturing manufacturing data. An outward oriented CDO, in contrast, might work with customers to improve the data flow between the firm and the customer.

Inward oriented initiatives might include developing data quality assessment methods, establishing data products standards, creating procedures for managing metadata, and establishing data governance. The goal is to ensure consistent data delivery and quality inside the organization.

An outwardly-focused CDO will strive to cooperate with an organization’s external stakeholders. For example, one CDO led a program for “global unique product identification” to improve collaboration with external global partners. Another might pay attention to improving the quality of data supplied to external partners.

### Traditional vs. big data management focus

A CDO can stress managing traditional transactional data, which is typically managed with a relational databases, or shift direction towards handling expanding data volumes with new files structures, such as Hadoop data file structure (HDFS).

Traditional data are still the foundation of many organization’s operations, and there remains in many firms a need for a CDO with a transactional data orientation. Big data promises opportunities for improving operations or developing new business strategies based on analyses and insights not available from traditional data. A CDO attending to big data can provide leadership in helping a business gain deeper knowledge of its customers, suppliers, and so forth based on mining large volumes of data.

### Service vs. strategy orientation

A CDO can stress improving services or exploring an organization’s strategic opportunities. This dimension should reflect the organization’s goals for the CDO position. If the top management team is mainly concerned with oversight and accountability, then the CDO should pay attention to improving existing data-related processes. Alternatively, if the senior team actively seeks new data-driven strategic value, then the CDO needs to be similarly aligned and might look at how to exploit digit data streams, for example. One strategy-directed CDO, for instance, led an initiative to identify new information products for advancing the firm’s position in the financial industry.

## CDO archetypes

Based on the three dimensions just discussed, eight different CDO roles can be identified, as shown in the following table.

Dimensions of CDO archetypes

|  | Inward | Outward | Traditional data | Big data | Strategy | Service |
| --- | --- | --- | --- | --- | --- | --- |
| Coordinator |  |  |  |  |  |  |
| Reporter |  |  |  |  |  |  |
| Architect |  |  |  |  |  |  |
| Ambassador |  |  |  |  |  |  |
| Analyst |  |  |  |  |  |  |
| Marketer |  |  |  |  |  |  |
| Developer |  |  |  |  |  |  |
| Experimenter |  |  |  |  |  |  |

While eight different roles are possible, it is important to note that a CDO might take on several of these roles as the situation changes and the position evolves. Also, treat an archetype as a broad indicator of a role rather than a confining specification.

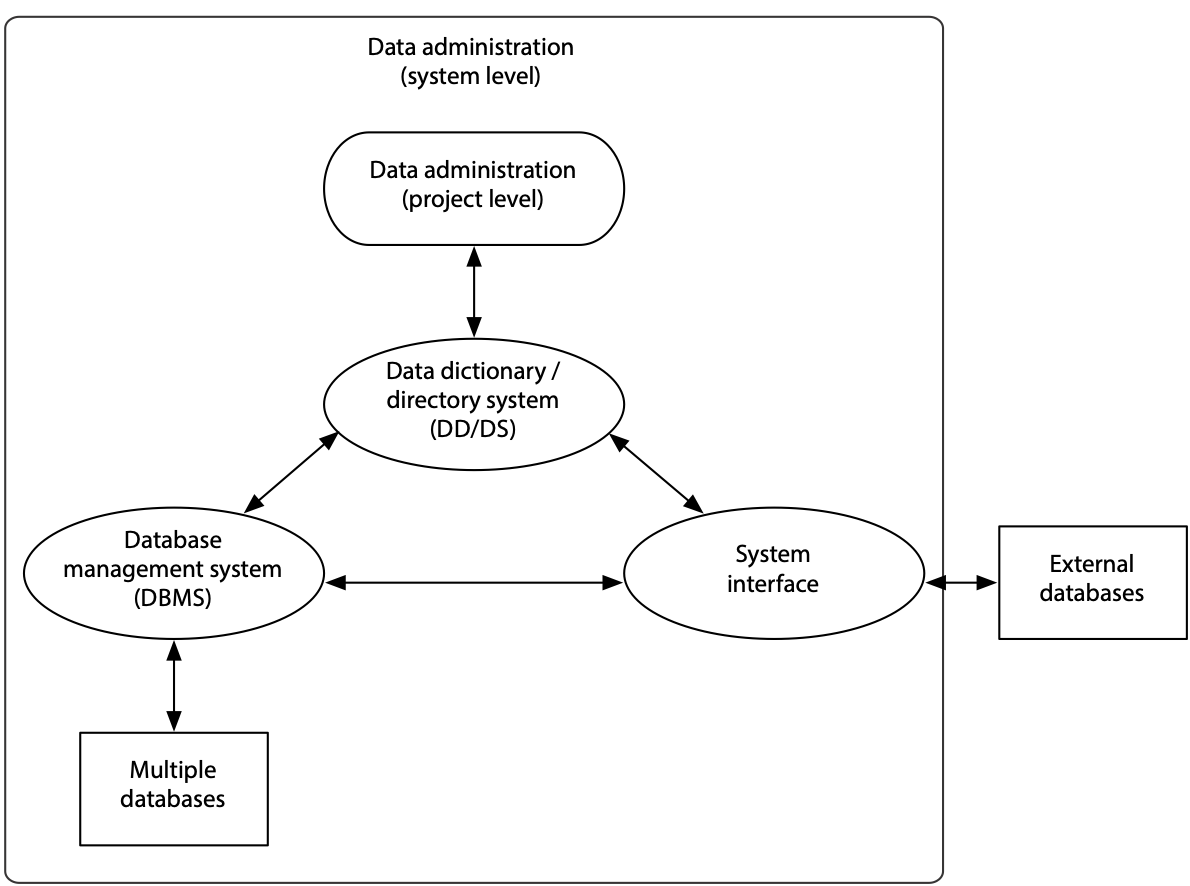
CDO archetypes

| Archetype | Definition |
| --- | --- |
| Coordinator | Fosters internal collaboration using transactional data to support business services. |
| Reporter | Provides high quality enterprise data delivery services for external reporting. |
| Architect | Designs databases and internal business processes to create new opportunities for the organization. |
| Ambassador | Develops internal data policies to support business strategy and external collaboration using traditional data sources. |
| Analyst | Improves internal business performance by exploiting big data to provide new services. |
| Marketer | Develops relationships with external data partners and stakeholders to improve externally provided data services using big data. |
| Developer | Navigates and negotiates with internal enterprise divisions in order to create new services by exploiting big data. |
| Experimenter | Engages with external parties, such as suppliers and industry peers, to explore new, unidentified markets and products based on insights derived from big data. |

# Management of the database environment

In many large organizations, there is a formal data administration function to manage corporate data. For those companies with a CDO, data administration is subsumed within this function. For those without, data administration is typically the responsibility of the CIO. The relationship among the various components of data administration is shown in the following figure.

Management of the database environment



## Databases

A database management system (DBMS) can manage multiple databases covering different aspects of an organization’s activities. When a database has multiple clients, it may be designed to meet all their requirements, even though a specific person may need only a portion of the data contained within the database. For instance, a finished-goods database may be accessed by Production and Finance, as well as Marketing. It may contain cost information that is accessible by Production and Finance but not by Marketing.

## System interface

The interface consists of windows, menus, icons, and commands that enable clients to direct the system to manipulate data. Clients may range from casual novices, who need to be insulated from the underlying complexity of the data, to expert application developers, who manipulate the data using programming languages or other data-handling tools.

## Data dictionary

A data dictionary is a reference repository containing metadata (i.e., data about data) that is stored in the database. Among other things, the data dictionary contains a list of all the databases; their component parts and detailed descriptions such as field sizes, data types, and data validation information for data capture purposes; authorized clients and their access privileges; and ownership details.

A data dictionary is a map of the data in organizational data stores. It permits the data administration staff and users to document the database, design new applications, and redesign the database if necessary. The data dictionary/directory system (DD/DS), itself a DBMS, is software for managing the data dictionary.

## External databases

For organizations to remain competitive in a rapidly changing marketplace, access to data from external sources is becoming increasingly critical. Research and development groups need access to the latest developments in their technical fields and need to track information such as patents filed, research reports, and new product releases. Marketing departments need access to data on market conditions and competitive situations as reported in various surveys and the media in general. Financial data regarding competitors and customers are important to senior executives. Monitoring political situations may be critical to many business decisions, especially in the international business arena.

Extensive external data are available electronically through information services such as Bloomberg (financial data), Reuters (news), and LexisNexis (legal data), various government sites,such as data.gov, or from various Web sites. Tools are available to download external data into internal databases, from which they may be accessed through the same interface as for internal data.

# Data administration

Data administration is responsible for the management of data-related activities. There are two levels of data administration activities: system and project.

System-level administration is concerned with establishing overall policies and procedures for the management and use of data in the organization. Formulating a data strategy and specifying an information architecture for the organization are also system-level data administration functions.

Project-level administration deals more with the specifics, such as optimizing specific databases for operational efficiency, establishing and implementing database access rights, creating new databases, and monitoring database use.

In general, the system-level function takes a broader perspective of the role played by data in achieving business objectives, while the project-level function is more concerned with the actual mechanics of database implementation and operation. We use the term data administration to refer to both functional levels.

## Data administration functions and roles

The functions of data administration may be accomplished through multiple roles or job titles such as database administrator, database developer, database consultant, and database analyst, collectively referred to as the data administration staff. A single role could be responsible for both system and project levels of data administration, or responsibility may be distributed among several persons, depending on the size of the organization, the number of database applications, and the number of clients.

In addition, data administration could be carried out entirely in a client department. For instance, a client could be the data steward responsible for managing all corporate data for some critical business-related entity or activity (e.g., a customer, a production facility, a supplier, a division, a project, or a product) regardless of the purpose for which the data are used.

Data stewards coordinate planning of the data for which they are responsible. Tasks include data definition, quality control and improvement, security, and access authorization. The data steward’s role is especially important today because of the emphasis on customer satisfaction and cross-functional teams. Data stewardship seeks to align data management with organizational strategy.

## Database levels

Databases may be maintained at several levels of use: personal, workgroup (e.g., project team or department), and organizational. More clients usually results in greater complexity of both the database and its management.

Personal databases in the form of calendars, planners, and name and address books have existed for a long time. The availability of laptops, tablets, and smartphones have made it convenient to maintain synchronized electronic personal databases across multiple devices. Behind the interface of many of these apps is a lightweight relational database, such as SQLite.[[21]](#footnote-21)

Workgroup databases cannot be as idiosyncratic because they are shared by many people. Managing them requires more planning and coordination to ensure that all the various clients’ needs are addressed and data integrity is maintained. Organizational databases are the most complex in terms of both structure and need for administration. All databases, regardless of scope or level, require administration.

Managing a personal database is relatively simple. Typically, the owner of the database is also its developer and administrator. Issues such as access rights and security are settled quite easily, perhaps by locking the computer when away from the desk. Managing workgroup databases is more complex. Controls almost certainly will be needed to restrict access to certain data. On the other hand, some data will need to be available to many group members. Also, responsibility for backup and recovery must be established. Small workgroups may jointly perform both system- and project-level data administration activities. Meetings may be a way to coordinate system-level data administration activities, and project-level activities may be distributed among different workgroup members. Larger groups may have a designated data administrator, who is also a group member.

Managing organizational databases is typically a full-time job requiring special skills to work with complex database environments. In large organizations, several persons may handle data administration, each carrying out different data administration activities. System-level data administration activities may be carried out by a committee led by a senior IS executive (who may be a full- or part-time data administrator), while project-level data administration activities may be delegated to individual data administration staff members.

## System-level data administration functions

System-level data administration functions, which may be performed by one or more persons, are summarized in the following table.

| Function |
| --- |
| Planning |
| Developing data standards and policies |
| Defining XML data schemas |
| Maintaining data integrity |
| Resolving data conflict |
| Managing the DBMS |
| Establishing and maintaining the data dictionary |
| Selecting hardware and software |
| Managing external databases |
| Benchmarking |
| Internal marketing |

### Planning

Because data are a strategic corporate resource, planning is perhaps the most critical data administration function. A key planning activity is creating an organization’s information architecture, which includes all the major data entities and the relationships between them. It indicates which business functions and applications access which entities. An information architecture also may address issues such as how data will be transmitted and where they will be stored. Since an information architecture is an organization’s overall strategy for data and applications, it should dovetail with the organization’s long-term plans and objectives.[[22]](#footnote-22)

### Developing data standards and policies

Whenever data are used by more than one person, there must be standards to govern their use. Data standards become especially critical in organizations using heterogeneous hardware and software environments. Why could this become a problem? For historical reasons, different departments may use different names and field sizes for the same data item. These differences can cause confusion and misunderstanding. For example, “sale date” may have different meanings for the legal department (e.g., the date the contract was signed) and the sales department (e.g., the date of the sales call). Furthermore, the legal department may store data in the form yyyy-mm-dd and the sales department as dd-mm-yy. Data administration’s task is to develop and publish data standards so that field names are clearly defined, and a field’s size and format are consistent across the enterprise.

Furthermore, some data items may be more important to certain departments or divisions. For instance, customer data are often critical to the marketing department. It is useful in such cases to appoint a data steward from the appropriate functional area as custodian for these data items.

Policies need to be established regarding who can access and manipulate which data, when, and from where. For instance, should employees using their home computers be allowed to access corporate data? If such access is permitted, then data security and risk exposure must be considered and adequate data safeguards implemented.

### Defining XML data schemas

Data administration is also often responsible for defining data schemas for data exchange within the organization and among business partners. Data administration is also responsible for keeping abreast of industry schema standards so that the organization is in conformance with common practice. Some data administrators may even work on defining a common data schema for an industry.

### Maintaining data integrity

Data must be made available when needed, but only to authorized users. The data management aspects of data integrity are discussed at length in the Data Integrity chapter.

### Resolving data conflict

Data administration involves the custodianship of data owned or originating in various organizational departments or functions, and conflicts are bound to arise at some point. For instance, one department may be concerned about a loss of security when another department is allowed access to its data. In another instance, one group may feel that another is contaminating a commonly used data pool because of inadequate data validation practices. Incidents like these, and many others, require management intervention and settlement through a formal or informal process of discussion and negotiation in which all parties are assured of an outcome that best meets the needs of the organization. Data administration facilitates negotiation and mediates dispute resolution.

### Managing the DBMS

While project-level data administration is concerned more directly with the DBMS, the performance and characteristics of the DBMS ultimately impinge on the effectiveness of the system-level data administration function. It is, therefore, important to monitor characteristics of the DBMS. Over a period, benchmark statistics for different projects or applications will need to be compiled. These statistics are especially useful for addressing complaints regarding the performance of the DBMS, which may then lead to design changes, tuning of the DBMS, or additional hardware.

Database technology is rapidly advancing. For example, relational DBMSs are continually being extended, and Hadoop offers a new approach to handle large data processing tasks. Keeping track of developments, evaluating their benefits, and deciding on converting to new database environments are critical system-level data administration functions that can have strategic implications for the corporation.

### Establishing and maintaining the data dictionary

A data dictionary is a key data administration tool that provides details of data in the organizational database and how they are used (e.g., by various application programs). If modifications are planned for the database (e.g., changing the size of a column in a table), the data dictionary helps to determine which applications will be affected by the proposed changes.

More sophisticated data dictionary systems are closely integrated with specific database products. They are updated automatically whenever the structure of the underlying database is changed.

### Selecting hardware and software

Evaluating and selecting the appropriate hardware and software for an organizational database are critical responsibilities with strategic organizational implications. These are not easy tasks because of the dynamic nature of the database industry, the continually changing variety of available hardware and software products, and the rapid pace of change within many organizations. Today’s excellent choice might become tomorrow’s nightmare if, for instance, the vendor of a key database component goes out of business or ceases product development.

Extensive experience and knowledge of the database software business and technological progress in the field are essential to making effective database hardware and software decisions. The current and future needs of the organization need to be assessed in terms of capacity as well as features. Relevant questions include the following:

* How many people and apps will simultaneously access the database?
* Will the database need to be geographically distributed? If so, what is the degree to which the database will be replicated, and what is the nature of database replication that is supported?
* What is the maximum size of the database?
* How many transactions per second can the DBMS handle?
* What kind of support for online transaction processing is available?
* What are the initial and ongoing costs of using the product?
* Can the database be extended to include new data types?
* What is the extent of training required, who can provide it, and what are the associated costs?

DBMS selection should cover technical, operational, and financial considerations. An organization’s selection criteria are often specified in a request for proposal (RFP). This document is sent to a short list of potential vendors, who are invited to respond with a software or hardware/software proposal outlining how their product or service meets each criterion in the RFP. Discussions with current customers are usually desirable to gain confirming evidence of a vendor’s claims. The final decision should be based on the manner and degree to which each vendor’s proposal satisfies these criteria.

Skill builder

A small nonprofit organization has asked for your help in selecting a relational database management system (RDBMS) for general-purpose management tasks. Because of budget limitations, it is very keen to adopt an open source RDBMS. Search the Web to find at least two open source RDBMSs, compare the two systems, and make a recommendation to the organization.

### Benchmarking

Benchmarking, the comparison of alternative hardware and software combinations, is an important step in the selection phase. Because benchmarking is an activity performed by many IS units, the IS community gains if there is one group that specializes in rigorous benchmarking of a wide range of systems. The Transaction Processing Council[[23]](#footnote-23)(TPC) is the IS profession’s Consumer Union. TPC has established benchmarks for a variety of business situations.

### Managing external databases

Providing access to external databases has increased the level of complexity of data administration, which now has the additional responsibility of identifying information services that meet existing or potential managerial needs. Data administration must determine the quality of such data and the means by which they can be channeled into the organization’s existing information delivery system. Costs of data may vary among vendors. Some may charge a flat monthly or annual fee, while others may have a usage charge. Data may arrive in a variety of formats, and data administration may need to make them adhere to corporate standards. Data from different vendors and sources may need to be integrated and presented in a unified format and on common screens.

Monitoring external data sources is critical because data quality may vary over time. Data administration must determine whether organizational needs are continuing to be met and data quality is being maintained. If they are not, a subscription may be canceled and an alternative vendor sought. Security is another critical problem. When corporate databases are connected to external communication links, there is a threat of hackers breaking into the system and gaining unauthorized access to confidential internal data. Also, corporate data may be contaminated by spurious data or even by viruses entering from external sources. Data administration must be cautious when incorporating external data into the database.

### Internal marketing

Because IS applications can have a major impact on organizational performance, the IS function is becoming more proactive in initiating the development of new applications. Many clients are not aware of what is possible with newly emergent technologies, and hence do not see opportunities to exploit these developments. Also, as custodian of organizational data, data administration needs to communicate its goals and responsibilities throughout the organization. People and departments need to be persuaded to share data that may be of value to other parts of the organization. There may be resistance to change when people are asked to switch to newly set data standards. In all these instances, data administration must be presented in a positive light to lessen resistance to change. Data administration needs to market internally its products and services to its customers.

## Project-level data administration

At the project level, data administration focuses on the detailed needs of individual clients and applications. It supports the development and use of a specific database system.

## Systems development life cycle (SDLC)

Database development follows a fairly predictable sequence of steps or phases similar to the systems development life cycle (SDLC) for applications. This sequence is called the database development life cycle (DDLC). The database and application development life cycles together constitute the systems development life cycle (SDLC).

Systems development life cycles

| Application development life cycle (ADLC) | Database development life cycle (DDLC) |
| --- | --- |
| Project planning | Project planning |
| Requirements definition | Requirements definition |
| Application design | Database design |
| Application construction |  |
| Application testing | Database testing |
| Application implementation | Database implementation |
| Operations | Database usage |
| Maintenance | Database evolution |

Application development involves the eight phases shown in the preceding table. It commences with project planning which, among other things, involves determining project feasibility and allocating the necessary personnel and material resources for the project. This is followed by requirements definition, which involves considerable interaction with clients to clearly specify the system. These specifications become the basis for a conceptual application design, which is then constructed through program coding and tested. Once the system is thoroughly tested, it is installed and user operations begin. Over time, changes may be needed to upgrade or repair the system, and this is called system maintenance.

The database development phases parallel application development. Data administration is responsible for the DDLC. Data are the focus of database development, rather than procedures or processes. Database construction is folded into the testing phase because database testing typically involves minimal effort. In systems with integrated data dictionaries, the process of constructing the data dictionary also creates the database shell (i.e., tables without data). While the sequence of phases in the cycle as presented is generally followed, there is often a number of iterations within and between steps. Data modeling is iterative, and the final database design evolves from many data modeling sessions. A previously unforeseen requirement may surface during the database design phase, and this may prompt a revision of the specifications completed in the earlier phase.

System development may proceed in three different ways:

1. The database may be developed independently of applications, following only the DDLC steps.
2. Applications may be developed for existing databases, following only the ADLC steps.
3. Application and database development may proceed in parallel, with both simultaneously stepping through the ADLC and DDLC.

Consider each of these possibilities.

Database development may proceed independently of application development for a number of reasons. The database may be created and later used by an application or for ad hoc queries using SQL. In another case, an existing database may undergo changes to meet changed business requirements. In such situations, the developer goes through the appropriate stages of the DDLC.

Application development may proceed based on an existing database. For instance, a personnel database may already exist to serve a set of applications, such as payroll. This database could be used as the basis for a new personnel benefits application, which must go through all the phases of the ADLC.

A new system requires both application and database development. Frequently, a new system will require creation of both a new database and applications. For instance, a computer manufacturer may start a new Web-based order sales division and wish to monitor its performance. The vice-president in charge of the division may be interested in receiving daily sales reports by product and by customer as well as a weekly moving sales trend analysis for the prior 10 weeks. This requires both the development of a new sales database as well as a new application for sales reporting. Here, the ADLC and DDLC are both used to manage development of the new system.

### Database development roles

Database development involves several roles, chiefly those of developer, client, and data administrator. The roles and their responsibilities are outlined in the following table.

Database development roles

| Database development phase | Database developer | Data administrator | Client |
| --- | --- | --- | --- |
| Project planning | Does | Consults | Provides information |
| Requirements definition | Does | Consults | Provides requirements |
| Database design | Does | Consults Data integrity | Validates data models |
| Database testing | System and client testing | Consults Data integrity | Testing |
| Database implementation | System-related activities | Consults Data integrity | Client activities |
| Database usage | Consults | Data integrity monitoring | Uses |
| Database evolution | Does | Change control | Provides additional requirements |

The database developer shoulders the bulk of the responsibility for developing data models and implementing the database. This can be seen in the table, where most of the cells in the database developer column are labeled “Does.” The database developer does project planning, requirements definition, database design, database testing, and database implementation, and in addition, is responsible for database evolution.

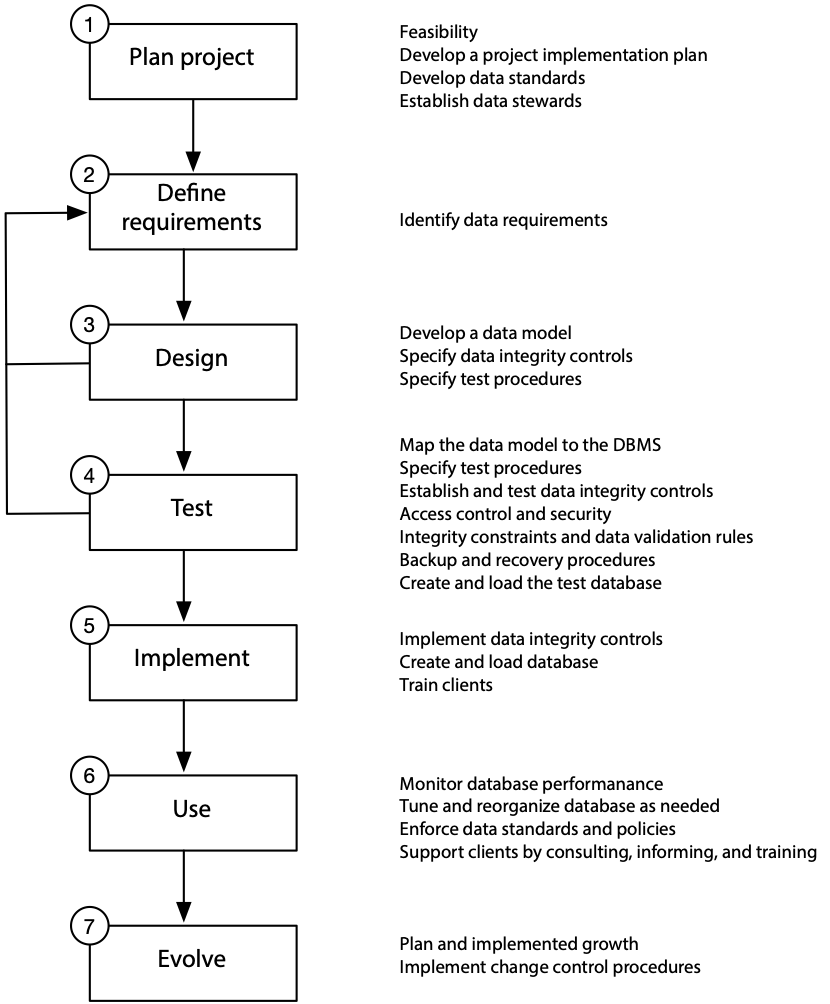
The client’s role is to establish the goals of a specific database project, provide the database developers with access to all information needed for project development, and review and regularly scrutinize the developer’s work.

The data administrator’s prime responsibilities are implementing and controlling, but the person also may be required to perform activities and consult. In some situations, the database developer is not part of the data administration staff and may be located in a client department, or may be an analyst from an IS project team. In these cases, the data administrator advises the developer on organizational standards and policies as well as provides specific technical guidelines for successful construction of a database. When the database developer is part of the data administration staff, developer and data administration activities may be carried out by the same person, or by the person(s) occupying the data administration role. In all cases, the data administrator should understand the larger business context in which the database will be used and should be able to relate business needs to specific technical capabilities and requirements.

### Database development life cycle (DDLC)

Previously, we discussed the various roles involved in database development and how they may be assigned to different persons. In this section, we will assume that administration and development are carried out by the data administration staff, since this is the typical situation encountered in many organizations. The activities of developer and administrator, shown in the first two columns of the table, are assumed to be performed by data administration staff. Now, let us consider data administration project-level support activities in detail . These activities are discussed in terms of the DDLC phase they support.

Database development life cycle



#### Database project planning

Database project planning includes establishing project goals, determining project feasibility (financial, technical, and operational), creating an implementation plan and schedule, assigning project responsibilities (including data stewards), and establishing standards. All project stakeholders, including clients, senior management, and developers, are involved in planning. They are included for their knowledge as well as to gain their commitment to the project.

#### Requirements definition

During requirements definition, clients and developers establish requirements and develop a mutual understanding of what the new system will deliver. Data are defined and the resulting definitions stored in the data dictionary. Requirements definition generates documentation that should serve as an unambiguous reference for database development. Although in theory the clients are expected to sign-off on the specifications and accept the developed database as is, their needs may actually change in practice. Clients may gain a greater understanding of their requirements and business conditions may change. Consequently, the original specifications may require revision. In the preceding figure, the arrows connecting phase 4 (testing) and phase 3 (design) to phase 2 (requirements definition) indicate that modeling and testing may identify revisions to the database specification, and these amendments are then incorporated into the design.

#### Database design

Conceptual and internal models of the database are developed during database design. Conceptual design, or data modeling, is discussed extensively in Section 2 of this book. Database design should also include specification of procedures for testing the database. Any additional controls for ensuring data integrity are also specified. The external model should be checked and validated by the user.

#### Database testing

Database testing requires previously developed specifications and models to be tested using the intended DBMS. Clients are often asked to provide operational data to support testing the database with realistic transactions. Testing should address a number of key questions.

* Does the DBMS support all the operational and security requirements?
* Is the system able to handle the expected number of transactions per second?
* How long does it take to process a realistic mix of queries?

Testing assists in making early decisions regarding the suitability of the selected DBMS. Another critical aspect of database testing is verifying data integrity controls. Testing may include checking backup and recovery procedures, access control, and data validation rules.

#### Database implementation

Testing is complete when the clients and developers are extremely confident the system meets specified needs. Data integrity controls are implemented, operational data are loaded (including historical data, if necessary), and database documentation is finalized. Clients are then trained to operate the system.

#### Database use

Clients may need considerable support as they learn and adapt to the system. Monitoring database performance is critical to keeping them satisfied; enables the data administrator to anticipate problems even before the clients begin to notice and complain about them, and tune the system to meet organizational needs; and also helps to enforce data standards and policies during the initial stages of database implementation.

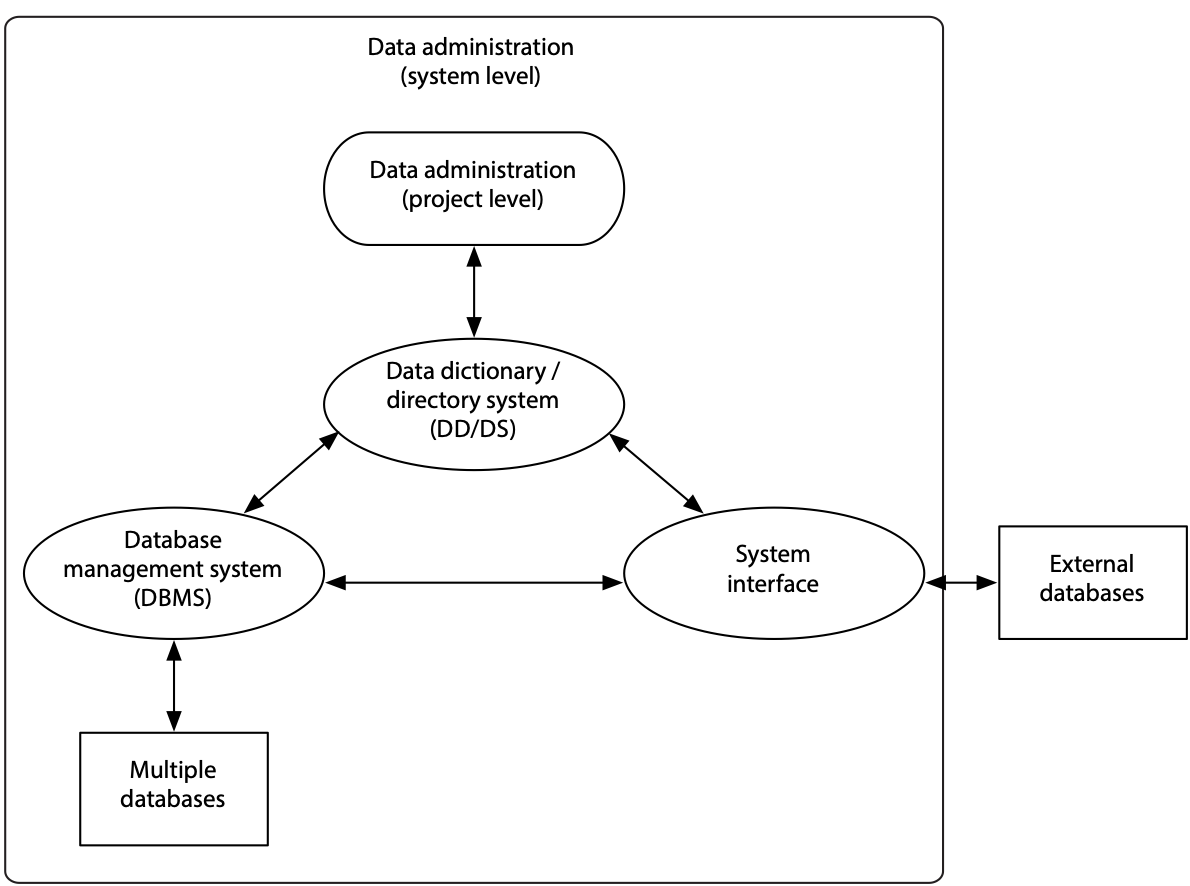
#### Database evolution

Since organizations cannot afford to stand still in today’s dynamic business environment, business needs are bound to change over time, perhaps even after a few months. Data administration should be prepared to meet the challenge of change. Minor changes, such as changes in display formats, or performance improvements, may be continually requested. These have to be attended to on an ongoing basis. Other evolutionary changes may emerge from constant monitoring of database use by the data administration staff. Implementing these evolutionary changes involves repeating phases 3 to 6. Significant business changes may merit a radical redesign of the database. Major redesign may require repeating all phases of the DDLC.

## Data administration interfaces

Data administration is increasingly a key corporate function, and it requires the existence of established channels of communication with various organizational groups. The key data administration interfaces are with clients, management, development staff, and computer operations. The central position of the data administration staff, shown in the following figure, reflects the liaison role that it plays in managing databases. Each of the groups has a different focus and different terminology and jargon. Data administration should be able to communicate effectively with all participants. For instance, operations staff will tend to focus on technical, day-to-day issues to which management is likely to pay less attention. These different focuses can, and frequently do, lead to conflicting views and expectations among the different groups. Good interpersonal skills are a must for data administration staff in order to deal with a variety of conflict-laden situations. Data administration, therefore, needs a balance of people, technical, and business skills for effective execution of its tasks.

Major data administration interfaces



Data administration probably will communicate most frequently with computer operations and development staff, somewhat less frequently with clients, and least frequently with management. These differences, however, have little to do with the relative importance of communicating with each group. The interactions between the data administration staff and each of the four groups are discussed next.

### Management

Management sets the overall business agenda for data administration, which must ensure that its actions directly contribute to the achievement of organizational goals. In particular, management establishes overall policy guidelines, approves data administration budgets, evaluates proposals, and champions major changes. For instance, if the data administration staff is interested in introducing new technology, management may request a formal report on the anticipated benefits of the proposed expenditure.

Interactions between the data administration staff and management may focus on establishing and evolving the information architecture for the organization. In some instances, these changes may involve the introduction of a new technology that could fundamentally transform roles or the organization.

### Clients

On an ongoing basis, most clients will be less concerned with architectural issues and will focus on their personal needs. Data administration must determine what data should be collected and stored, how they should be validated to ensure integrity, and in what form and frequency they should be made available.

Typically, the data administration staff is responsible for managing the database, while the data supplier is responsible for ensuring accuracy. This may cause conflict, however, if there are multiple clients from separate departments. If conflict does arise, data administration has to arbitrate.

### Development staff

Having received strategic directions from management, and after determining clients’ needs, data administration works with application and database developers, in order to fulfill the organization’s goals for the new system. On an ongoing basis, this may consist of developing specifications for implementation. Data administration works on an advisory basis with systems development, providing inputs on the database aspects. Data administration is typically responsible for establishing standards for program and database interfaces and making developers aware of these standards. Developers may need to be told which commands may be used in their programs and which databases they can access.

In many organizations, database development is part of data administration, and it has a very direct role in database design and implementation. In such instances, communication between data administration and database development is within the group. In other organizations, database development is not part of data administration, and communication is between groups.

### Computer operations

The focus of computer operations is on the physical hardware, procedures, schedules and shifts, staff assignments, physical security of data, and execution of programs. Data administration responsibilities include establishing and monitoring procedures for operating the database. The data administration staff needs to establish and communicate database backup, recovery, and archiving procedures to computer operations. Also, the scheduling of new database and application installations needs to be coordinated with computer operations personnel.

Computer operations provide data administration with operational statistics and exception reports. These data are used by data administration to ensure that corporate database objectives are being fulfilled.

### Communication

The diverse parties with which data administration communicates often see things differently. This can lead to misunderstandings and results in systems that fail to meet requirements. Part of the problem arises from a difference in perspective and approaches to viewing database technology.

Management is interested in understanding how implementing database technology will contribute to strategic goals. In contrast, clients are interested in how the proposed database and accompanying applications will affect their daily work. Developers are concerned with translating management and client needs into conceptual models and converting these into tables and applications. Operations staff are concerned primarily with efficient daily management of DBMS, computer hardware, and software.

Data models can serve as a common language for bridging the varying goals of clients, developers, management, and operational staff. A data model can reduce the ambiguity inherent in verbal communications and thereby ensure that overall needs are more closely met and all parties are satisfied with the results. A data model provides a common meeting point and language for understanding the needs of each group.

Data administration does not work in isolation. It must communicate successfully with all its constituents in order to be successful. The capacity to understand and correctly translate the needs of each stakeholder group is the key to competent data administration.

### Data administration tools

Several computer-based tools have emerged to support data administration. There are five major classes of tools: data dictionary, DBMS, performance monitoring, computer-aided software engineering (CASE), and groupware tools. Each of these tools is now examined and its role in supporting data administration considered. We focus on how these tools support the DDLC. Note, however, that groupware is not shown in the following table because it is useful in all phases of the life cycle.

Data administration tool use during the DDLC

|  | Database development phase | Data | DBMS | Performance monitoring | Case tools |
| --- | --- | --- | --- | --- | --- |
| 1 | Project planning tools | Dictionary (DD) |  |  | Estimation |
| 2 | Requirements definition | Document  Design aid |  |  | Document  Design aid |
| 3 | Database design | Document  Data map  Design aid  Schema generator |  |  | Document  Design aid  Data map |
| 4 | Database testing | Data map  Design aid  Schema generator | Define, create, test, data integrity | Impact analysis | Data generator  Design aid |
| 5 | Database implementation | Document  Change control | Data integrity  Implement  Design | Monitor  Tune |  |
| 6 | Database usage | Document  Data map  Schema generator  Change control | Provide tools for retrieval and update  Enforce integrity controls and procedures | Monitor  Tune |  |
| 7 | Database evolution | Document  Data map  Change control | Redefine | Impact analysis |  |

Data administration staff, clients, and computer operations all require information about organizational databases. Ideally, such information should be stored in one central repository. This is the role of the DD/DS, perhaps the main data administration tool. Thus, we start our discussion with this tool.

### Data dictionary/directory system (DD/DS)

The DD/DS, a database application that manages the data dictionary, is an essential tool for data administration. The DD/DS is the repository for organizational metadata, such as data definitions, relationships, and privileges. The DBMS manages data, and the DD/DS manages data about data. The DD/DS also uses the data dictionary to generate table definitions or schema required by the DBMS and application programs to access databases.

Clients and data administration can utilize the DD/DS to ask questions about characteristics of data stored in organizational databases such as

* What are the names of all tables for which the user Todd has delete privileges?
* Where does the data item customer number appear or where is it used?

The report for the second query could include the names and tables, transactions, reports, display screens, account names, and application programs.

In some systems, such as a relational DBMS, the catalog, discussed in the SQL chapter, performs some of the functions of a DD/DS, although the catalog does not contain the same level of detail. The catalog essentially contains data about tables, columns, and owners of tables, whereas the DD/DS can include data about applications, forms, transactions, and many other aspects of the system. Consequently, a DD/DS is of greater value to data administration.

Although there is no standard format for data stored in a data dictionary, several features are common across systems. For example, a data dictionary for a typical relational database environment would contain descriptions of the following:

* All columns that are defined in all tables of all databases. The data dictionary stores specific data characteristics such as name, data type, display format, internal storage format, validation rules, and integrity constraints. It indicates to which table a column belongs.
* All relationships among data elements, what elements are involved, and characteristics of relationships, such as cardinality and degree.
* All defined databases, including who created each database, the date of creation, and where the database is located.
* All tables defined in all databases. The data dictionary is likely to store details of who created the table, the date of creation, primary key, and the number of columns.
* All indexes defined for each of the database tables. For each of the indexes, the DBMS stores data such as the index name, location, specific index characteristics, and creation date.
* All clients and their access authorizations for various databases.
* All programs that access the database, including screen formats, report formats, application programs, and SQL queries.

A data dictionary can be useful for both systems and project level data administration activities. The five major uses of a data dictionary are the following:

1. Documentation support: recording, classifying, and reporting metadata.
2. Data maps: a map of available data for data administration staff and users. A data map allows users to discover what data exist, what they mean, where they are stored, and how they are accessed.
3. Design aid: documenting the relationships between data entities and performing impact analysis.
4. Schema generation: automatic generation of data definition statements needed by software systems such as the DBMS and application programs.
5. Change control: setting and enforcing standards, evaluating the impact of proposed changes, and implementing amendments, such as adding new data items.

# Database management systems (DBMSs)

The DBMS is the primary tool for maintaining database integrity and making data available to users. Availability means making data accessible to whoever needs them, when and where they need them, and in a meaningful form. Maintaining database integrity implies the implementation of control procedures to achieve the three goals discussed in a prior chapter: protecting existence, maintaining quality, and ensuring confidentiality. In terms of the DDLC life cycle (see the following figure), data administration uses, or helps others to use, the DBMS to create and test new databases, define data integrity controls, modify existing database definitions, and provide tools for clients to retrieve and update databases. Since much of this book has been devoted to DBMS functions, we limit our discussion to reviewing its role as a data administration tool.

## Performance monitoring tools

Monitoring the performance of DBMS and database operations by gathering usage statistics is essential to improving performance, enhancing availability, and promoting database evolution. Monitoring tools are used to collect statistics and improve database performance during the implementation and use stages of the DDLC. Monitoring tools can also be used to collect data to evaluate design choices during testing.

Many database factors can be monitored and a variety of statistics gathered. Monitoring growth in the number of rows in each table can reveal trends that are helpful in projecting future needs for physical storage space. Database access patterns can be scrutinized to record data such as:

* Type of function requested: query, insert, update, or delete
* Response time (elapsed time from query to response)
* Number of disk accesses
* Identification of client
* Identification of error conditions

The observed patterns help to determine performance enhancements. For example, these statistics could be used to determine which tables or files should be indexed. Since gathering statistics can result in some degradation of overall performance, it should be possible to turn the monitoring function on or off with regard to selected statistics.

## CASE tools

A CASE tool, as broadly defined, provides automated assistance for systems development, maintenance, and project management activities. Data administration may use project management tools to coordinate all phases within the DDLC. The dictionary provided by CASE systems can be used to supplement the DD/DS, especially where the database development effort is part of a systems development project.

One of the most important components of a CASE tool is an extensive dictionary that tracks all objects created by systems designers. Database and application developers can use the CASE dictionary to store descriptions of data elements, application processes, screens, reports, and other relevant information. Thus, during the first three phases of the life cycle, the CASE dictionary performs functions similar to a DD/DS. During stages 4 and 5, data from the CASE dictionary would be transferred, usually automatically, to the DD/DS.

# Groupware

Groupware can be applied by data administration to support any of the DDLC phases shown in the figure. Groupware supports communication among people and thus enhances access to organizational memory residing within humans. As pointed out, data administration interfaces with four major groups during the DDLC: management, clients, developers, and computer operations. Groupware supports interactions with all of these groups.

Data administration is a complex task involving a variety of technologies and the need to interact with, and satisfy the needs of, a diverse range of clients. Managing such a complex environment demands the use of computer-based tools, which make data administration more manageable and effective. Software tools, such as CASE and groupware, can improve data administration.

# Data integration

A common problem for many organizations is a lack of data integration, which can manifest in a number of ways:

* Different identifiers for the same instance of an entity (e.g., the same product with different codes in different divisions)
* The same, or what should be the same, data stored in multiple systems (e.g., a customer’s name and address)
* Data for, or related to, a key entity stored in different databases (e.g., a customer’s transaction history and profile stored in different databases)
* Different rules for computing the same business indicator (e.g., the Australian office computes net profit differently from the U.S. office)

In the following table, we see an example of a firm that practices data integration. Different divisions use the same numbers for parts, the same identifiers for customers, and have a common definition of sales date. In contrast, the table after that shows the case of a firm where there is a lack of data integration. The different divisions have different identifiers for the same part and different codes for the same customer, as well as different definitions for the sales date. Imagine the problems this nonintegrated firm would have in trying to determine how much it sold to each of its customers in the last six months.

Firm with data integration

|  | Red division | Blue division |
| --- | --- | --- |
| partnumber  (code for green widget) | 27 | 27 |
| customerid  (code for UPS) | 53 | 53 |
| Definition of salesdate | The date the customer signs the order | The date the customer signs the order |

Firm without data integration

|  | Red division | Blue division |
| --- | --- | --- |
| partnumber  (code for green widget) | 27 | 10056 |
| customerid  (code for UPS) | 53 | 613 |
| Definition of salesdate | The date the customer signs the order | The date the customer receives the order |

Skill builder

Complete the data integration lab exercise described on the book’s Web site (Lab exercises > Data integration).

Not surprisingly, many organizations seek to increase their degree of data integration so that they can improve the accuracy of managerial reporting, reduce the cost of managing data, and improve customer service by having a single view of the customer.

There are several goals of data integration:

1. A standard meaning for all data elements within the organization (e.g., customer acquisition date is the date on which the customer first purchased a product)
2. A standard format for each and every data element (e.g., all dates are stored in the format yyyymmdd and reported in the format yyyy-mm-dd)
3. A standard coding system (e.g., female is coded “f” and male is coded “m”)
4. A standard measurement system (e.g., all measurements are stored in metric format and reported in the client’s preferred system)
5. A single corporate data model, or a least a single data model for each major business entity

These are challenging goals for many organizations and sometimes take years to achieve. Many organizations are still striving to achieve the fifth, and most difficult, goal of a single corporate data model. Sometimes, however, data integration might not be a goal worth pursuing if the costs outweigh the benefits.

The two major factors that determine the desirable degree of data integration between organizational units are unit interdependence and environmental turbulence. There is a high level of interdependence between organizational units when they affect each other’s success (for example, the output of one unit is used by the other). As a result of the commonality of some of their goals, these units will gain from sharing standardized information. Data integration will make it easier for them to coordinate their activities and manage their operations. Essentially, data integration means that they will speak a common information language. When there is low interdependence between two organizational units, the gains from data integration are usually outweighed by the bureaucratic costs and delays of trying to enforce standards. If two units have different approaches to marketing and manufacturing, then a high level of data integration is unlikely to be beneficial. They gain little from sharing data because they have so little in common.

When organizational units operate in highly turbulent environments, they need flexibility to be able to handle rapid change. They will often need to change their information systems quickly to respond to new competitive challenges. Forcing such units to comply with organizational data integration standards will slow down their ability to create new systems and thus threaten their ability to respond in a timely fashion.

Firms have three basic data integration strategies based on the level of organizational unit interdependence and environmental turbulence. When unit interdependence is low and environmental turbulence high, a unit should settle for a low level of data integration, such as common financial reporting and human resources systems. Moderate data integration might mean going beyond the standard financial reporting and human resources system to include a common customer database. If unit independence is high and turbulence high, then moderate data integration would further extend to those areas where the units overlap (e.g., if they share a manufacturing system, this would be a target for data integration). A high level of data integration, a desirable target when unit interdependence is high and environmental turbulence low, means targeting common systems for both units.

Target level of data integration between organizational units

|  |  |  |  |
| --- | --- | --- | --- |
|  |  | Unit interdependence | |
|  |  | Low | High |
| Environmental  turbulence | High | Low | Moderate |
| Low | Moderate | High |

Skill builder

1. Global Electronics has nine factories in the United States and Asia producing components for the computer industry, and each has its own information systems. Although there is some specialization, production of any item can be moved to another factory, if required. What level of data integration should the company seek, and what systems should be targeted for integration?
2. European Radio, the owner of 15 FM radio stations throughout Europe, has just purchased an electronics retailing chain of 50 stores in Brazil. What level of data integration should the company seek, and what systems should be targeted for integration?
3. Australian Leather operates several tanneries in Thailand, two leather goods manufacturing plants in Vietnam, and a chain of leather retailers in Australia and New Zealand. Recently, it purchased an entertainment park in Singapore. The various units have been assembled over the last five years, and many still operate their original information systems. What level of data integration should the company seek, and what systems should be targeted for integration?

# Conclusion

Data administration has become increasingly important for most organizations, particularly for those for whom data-driven decision making can significantly enhance performance. The emergence of the Chief Data Officer is a strong indicator of the growing impact of data on the ability of an organization to fulfill its mission. For many organizations, it is no longer sufficient to just focus on administering data. Rather, it is now critical for many to insert to insert data management and exploitation into the strategic thinking of the top management team.

## Summary

Companies are finding that data management is so critical to their future that they need a Chief Data Officer (CDO). Eight different types of CDO roles have been identified: coordinator, reporter, architect, ambassador, analyst, marketer, developer, and experimenter. In practice, a CDO is likely to shift among these roles depending on the enterprise’s needs.

Data administration is the task of managing that part of organizational memory that involves electronically available data. Managing electronic data stores is important because key organizational decisions are based on data drawn from them, and it is necessary to ensure that reliable data are available when needed. Data administration is carried out at both the system level, which involves overall policies and alignment with organizational goals, and the project level, where the specific details of each database are handled. Key modules in data administration are the DBMS, the DD/DS, user interfaces, and external databases.

Data administration is a function performed by those with assigned organizational roles. Data administration may be carried out by a variety of persons either within the IS department or in user departments. Also, this function may occur at the personal, workgroup, or organizational level.

Data administration involves communication with management, clients, developers, and computer operations staff. It needs the cooperation of all four groups to perform its functions effectively. Since each group may hold very different perspectives, which could lead to conflicts and misunderstandings, it is important for data administration staff to possess superior communication skills. Successful data administration requires a combination of interpersonal, technical, and business skills.

Data administration is complex, and its success partly depends on a range of computer-based tools. Available tools include DD/DS, DBMS, performance monitoring tools, CASE tools, and groupware.

## Key terms and concepts

Application development life cycle (ADLC)

Benchmark

Change agent

Computer-aided software engineering (CASE)

Data administration

Data dictionary

Data dictionary/directory system (DD/DS)

Data integrity

Data steward

Database administrator

Database developer

Database development life cycle (DDLC)

Database management system (DBMS)

External database

Groupware

Matrix organization

Performance monitoring

Project-level data administration

Request for proposal (RFP)

System-level data administration

Systems development life cycle (SDLC)

Transaction Processing Council (TPC)

User interface

## References and additional readings

Bostrom, R. P. 1989. Successful application of communication techniques to improve the systems development process. Information & Management 16:279–295.

Goodhue, D. L., J. A. Quillard, and J. F. Rockart. 1988. Managing the data resource: A contingency perspective. MIS Quarterly 12 (3):373–391.

Goodhue, D. L., M. D. Wybo, and L. J. Kirsch. 1992. The impact of data integration on the costs and benefits of information systems. *MIS Quarterly* 16 (3):293–311.

Redman, T. C. 2001. Data quality: The field guide. Boston: Digital Press.

## Exercises

1. Why do organizations need to manage data?
2. What problems can arise because of poor data administration?
3. What is the purpose of a data dictionary?
4. Do you think a data dictionary should be part of a DBMS or a separate package?
5. How does the management of external databases differ from internal databases?
6. What is the difference between system- and project-level data administration?
7. What is a data steward? What is the purpose of this role?
8. What is the difference between workgroups and organizational databases? What are the implications for data administration?
9. What is an information architecture?
10. Why do organizations need data standards? Give some examples of typical data items that may require standardization.
11. You have been asked to advise a firm on the capacity of its database system. Describe the procedures you would use to estimate the size of the database and the number of transactions per second it will have to handle.
12. Why would a company issue an RFP?
13. How do the roles of database developer and data administrator differ?
14. What do you think is the most critical task for the user during database development?
15. A medium-sized manufacturing company is about to establish a data administration group within the IS department. What software tools would you recommend that group acquire?
16. What is a stakeholder? Why should stakeholders be involved in database project planning?
17. What support do CASE tools provide for the DDLC?
18. How can groupware support the DDLC?
19. A large international corporation has typically operated in a very decentralized manner with regional managers having considerable autonomy. How would you recommend the corporation establish its data administration function?
20. Describe the personality of a successful data administration manager. Compare your assessment to the personality appropriate for a database technical adviser.
21. Write a job advertisement for a data administrator for your university.
22. What types of organizations are likely to have data administration reporting directly to the CIO?
23. What do you think are the most critical phases of the DDLC? Justify your decision.
24. When might application development and database development proceed independently?
25. Why is database monitoring important? What data would you ask for in a database monitoring report?

1. [http://www.gartner.com/it/page.jsp?id=2149415](http://www.terry.uga.edu/people/rwatson/) [↑](#footnote-ref-1)
2. https://www.theregister.co.uk/2017/05/22/ssd\_price\_premium\_over\_disk\_falling/ [↑](#footnote-ref-2)
3. [www.pcworld.com/businesscenter/article/217883/seagate\_solidstate\_disks\_are\_doomed\_at\_least\_for\_now.html](http://www.pcworld.com/businesscenter/article/217883/seagate_solidstate_disks_are_doomed_at_least_for_now.html) [↑](#footnote-ref-3)
4. <http://en.wikipedia.org/wiki/ITunes_Store> [↑](#footnote-ref-4)
5. The further electrons are moved, the more energy is lost. As much as a third of the initial energy generated can be lost during transmission across the grid before it gets to the final customer. [↑](#footnote-ref-5)
6. This section is based on Iyer, B., & Henderson, J. C. (2010). Preparing for the future: understanding the seven capabilities of cloud computing *MIS Executive*, 9(2), 117-131. [↑](#footnote-ref-6)
7. Mell, P., & Grance, T. (2009). The NIST definition of cloud computing: National Institute of Standards and Technology. [↑](#footnote-ref-7)
8. Install Eclipse IDE for Java EE Developers. [↑](#footnote-ref-8)
9. The code for creating the database is online at [http://www.richardtwatson.com/dm6e/Reader/java.html](http://richardtwatson.com/dm6e/Reader/java.html). [↑](#footnote-ref-9)
10. <http://en.wikipedia.org/wiki/Comma-separated_values> [↑](#footnote-ref-10)
11. Get the Java archive from <http://sourceforge.net/projects/javacsv/> [↑](#footnote-ref-11)
12. [http://www.richardtwatson.com/dm6e/Reader/java.html](http://richardtwatson.com/dm6e/Reader/java.html) [↑](#footnote-ref-12)
13. This is a feature of HTML5 and is not supported by older browsers. [↑](#footnote-ref-13)
14. [http://www.richardtwatson.com/dm6e/Reader/java.html](http://richardtwatson.com/dm6e/Reader/java.html) [↑](#footnote-ref-14)
15. To the author’s knowledge, Gordon C. Everest was the first to define data integrity in terms of these three goals. See Everest, G. (1986). Database management: objectives, systems functions, and administration. New York, NY: McGraw-Hill. [↑](#footnote-ref-15)
16. Source: Lewis Jr, W., Watson, R. T., & Pickren, A. (2003). An empirical assessment of IT disaster probabilities. Communications of the ACM, 46(9), 201-206. [↑](#footnote-ref-16)
17. Redman, T. C. (2001). Data quality : the field guide. Boston: Digital Press. [↑](#footnote-ref-17)
18. November 2012. <http://www.atlanta-airport.com/docs/Traffic/201211.pdf> [↑](#footnote-ref-18)
19. Adapted from Helman, P. (1994). The science of database management. Burr Ridge, IL: Richard D. Irwin, Inc. p. 434 [↑](#footnote-ref-19)
20. This section is based on Lee, Y., Madnick, S., Wang, R., Forea, W., & Zhang, H. (2014). A cubic framework for the Chief Data Officer (CDO): Succeeding in a world of Big Data emergence of Chief Data Officers. *MISQ Executive*. [↑](#footnote-ref-20)
21. <http://www.sqlite.org> [↑](#footnote-ref-21)
22. For more on information architecture, see Smith, H. A., Watson, R. T., & Sullivan, P. (2012). Delivering Effective Enterprise Architecture at Chubb Insurance. MISQ Executive. 11(2) [↑](#footnote-ref-22)
23. <http://www.tpc.org> [↑](#footnote-ref-23)