

or who live on Bayard Street, we would take the union of the two sets of indices:

```
>> union([1;3;5;6], [1;2])
ans: 1
      2
      3
      5
      6
```

Note that although index 1 is in both sets, it does not appear twice in the union.

When more than two sets are involved, the result can be computed by combining the sets in successive pairs. For example, the union of three lists, one, two, and three, can be computed by `union(one, union(two, three))`. The intersection works similarly.

There are other operations on pairs of sets that are also important. See the built-in operators `ismember`, `setdiff`, and `setxor`, as well as the single-list operator `unique`.

## 11.4 Databases

Tables are a wonderfully useful organizing scheme for data, as evidenced by the tremendous success of spreadsheet programs that store data as a rectangular array of cells. Common sense tells us that we need separate tables for separate purposes: It seems silly to integrate a phone book and a grocery list into a single table.

### Key Term

A *relational database* is a set of tables, all relating to a common purpose. One of the important skills in designing a database is to recognize when it's best to use multiple tables rather than a single table to represent data. The principal reason for doing this is to avoid whenever possible the redundancy of storing the same information in different places. A small problem with redundancy is that it wastes space. Much more significant is that redundancy introduces the possibility of error and inconsistency. The items intended to be the same may fail to be so, perhaps because one of the items has been updated but the others have not been.

Consider, for example, the census information described in Chapter 4, where we sought to keep track of several items for each household: the address, the income, and the names, ages, sex, and profession of the household members. In Chapter 4, we organized this information as a single table with one row for each household; the fields were address, income, first names, and so on. While all of the information could be stored in this manner, accessing the information was sometimes difficult.

In this section, we show how the census data can be stored as a relational database. This provides a much more flexible way of arranging the data, but to use the database, we must first master special table operators, such as “join,” that will be introduced in the next section.

The usual pattern in this book has been to use built-in MATLAB operators or to construct our own operators in MATLAB. We will deviate from that pattern here. Database operators are not included in MATLAB. There is not even a standard, general-purpose data structure for tables that fits in well with relational operators.

In principle, it would be possible to construct such data structures and operators using MATLAB primitives. There is little point in doing this, however, since commercial and freeware database systems are readily available, such as MySQL. These systems will give superior performance, reliability, portability, and flexibility compared to a home-brewed MATLAB-based system. Database systems are of such great economic importance that literally tens of thousands of man years of development have gone into them. There is even a standardized way of communicating with databases, called *Structured Query Language (SQL)*, that works with many commercial and freeware systems.

It would take much less time to learn to use a commercial or freeware relational database system than to program even a simple one in MATLAB. The skills gained will be portable. If one needs to access complicated data from MATLAB, the proper approach is to set up data communication between a database system and MATLAB. This can be done in many ways; there is even a MATLAB toolbox for database communications.

In this section, we do not attempt to teach the syntax of SQL or other skills of database use and management. (There are many texts that cover these matters, e.g., [16].) Instead, we want to introduce a few selected concepts of relational databases. Our main objective is to show how a relational database can provide flexible access to data. We hope to convince you that when it comes to complex forms of data, you should use appropriate database tools.

### Key Term

## Operators on Tables

### Key Term

The basic unit of a relational database is a table. The nomenclature that we have been using for tables has been informal: We’ve used “item” to denote one row of a table, and *field* to denote one column. While we will stay with this nomenclature, be aware that database management, like many specialized areas of technology, has its own terminology. In relational databases, a table is called a relation, an item is called a tuple, and a field is called an attribute.

When data in a relational database are accessed, the result is always returned in the form of a table. The transformation of database tables into new tables involves operators: Each operator takes one or more tables as input and returns a table as output.

**Key Term**

We have already encountered one such operator. When we pull items from a table that meet a specified criterion, we are performing a *selection*. For example, if we select from the telephone number table on page 255 using the criterion that the telephone number matches 433 4321, the returned table is

Last Name	First Name	Phone Number	Street Address
Webster	Miriam	433 4321	1812 Princeton

**Key Term**

It may seem grandiose to call such a thing a table, since there is only one item, but it is indeed in the format of a table with four fields and one item. We could even have an empty table; that is, a table with no items.

The *projection operator* creates a new table with only the fields specified from the input table. For example, projecting the telephone table onto the Last Name field would return

Last Name
Cummings
Wilson
Cummings
Webster
Cummings
Cummings

Such operators can be combined. For example, looking up Miriam Webster’s phone number involves a selection based on first and last name. The output of the selection is then projected onto telephone number, producing a simple table.

Phone Number
433 4321

**Key Term**

Another operator that takes a single table as input is *grouping*. When grouping, we divide all of the items in a table into sets; within each set, all of the items are identical in some defined way. Then a computation is performed on each of the sets. For example, grouping the telephone table over last name and then computing the count of items in each set, we get the following table:

Last Name	Count
Wilson	1
Webster	1
Cummings	4
Williams	1

In principle, any quantity can be computed on the sets; perhaps the most common quantities are simple ones such as the count, the mean, the minimum, and the maximum. In order to indicate which quantity is to be computed, we’ll use the notation *group/count*, *group/mean*, *group/min*, and so on.

Some operators involve more than one table as input. Two such operators that we have already seen are “intersection” and “union.” The intersection operator takes two tables as input and produces an output with all of the items that are in both input tables. The union operator is similar, but the output has all of the items that are in either table. Union and intersection cannot operate on any two tables: Both input tables must have the same fields. Such same-field tables are called *union compatible*. For example, the two inputs might be the result of two selections on the same table.

**Key Term**

**Key Term**

A related operator is *minus*. If A and B are two union-compatible tables, then A minus B is a table with all of the items in A except for those that are found in B.

**Key Term**

Another important operator with two tables as input is *natural join*. When joining two tables, the items in one table are associated in some way with items in the other table. The output table has all of the fields in either table (that is the union of the fields, but this is different from the union of the items). To illustrate, consider the grocery “list,” really a table that describes what we need to buy. We’ll use the name “Grocery Item” to emphasize the fact that each row in the table is a single item; the set of rows is the grocery list.

Grocery Item			
Name	Needed	Gotten	Unit
cheese	1	1	lb.
milk	1	1	gal.
oranges	5	0	
kiwi	3	0	
baguette	1	1	
bananas	1	0	bunch
apples	3	3	lb.

An uncommonly organized shopper might also have a Store table that describes where to buy each item. Here is a short version:

<b>Store</b>			
<b>Commodity</b>	<b>Store</b>	<b>Price</b>	<b>Aisle</b>
apples	Fruiteria	1.25	1
apples	Shop & Drop	1.49	18
baguette	Boul’s Bakery	1.99	
baguette	Shop & Drop	2.49	9
milk	Shop & Drop	3.57	23
oranges	Fruiteria	0.40	1

We can join the two tables, setting as the association between items a match on the name field in the Grocery table and the commodity field in the Store table:

<b>Natural Join: Grocery Item and Its Store Data</b>							
<b>Name</b>	<b>Needed</b>	<b>Gotten</b>	<b>Unit</b>	<b>Commodity</b>	<b>Store</b>	<b>Price</b>	<b>Aisle</b>
apples	3	0	lb.	apples	Fruiteria	1.25	1
apples	3	0	lb.	apples	Shop & Drop	1.49	18
baguette	1	0		baguette	Boul’s Bakery	1.99	counter
baguette	1	0		baguette	Shop & Drop	2.49	9
milk	1	0	gal.	milk	Shop & Drop	3.57	23
oranges	5	0		oranges	Fruiteria	0.40	1

The joined table contains much redundant information. For example, the fact that we need three apples but have not yet gotten any appears in two different rows in the table. But by selecting and projecting, we can create another table with information in a useful format. For instance, selecting on Store equals Fruiteria and projecting onto Name, Needed, Gotten, Unit, Price and Aisle, we get a table that tells us what we should pick up while in the Fruiteria:

<b>Name</b>	<b>Needed</b>	<b>Gotten</b>	<b>Unit</b>	<b>Price</b>	<b>Aisle</b>
apples	3	0	lb.	1.25	1
oranges	5	0		0.40	1

**Key Term**

In a natural join of two tables, only those items that have a matching item in the other table are present in the output. In an *outer join* each item in either table appears in the output regardless of whether there is a match in the other table; this involves leaving blanks in the undefined fields of the unmatched items. For example, the outer join of the Grocery Item and Store tables will have rows for kiwi and cheese but blanks in the columns relating to the Store data.



**Example: A Census Database**

One of the aspects of the census data in Chapter 4 that makes them hard to work with is that there are two different units of analysis. Census data are fundamentally about individual people: names, ages, genders, and so on. This suggests arranging the data as a table where each item is an individual person and each field is one of the variables of interest.

**Person**

PID	First Name	Middle Name	Last Name	YOB	Sex	Profession	HID
34342	Emily	Lisa	Smith	1965	F	Mathematician	76
74523	George	Robert	Smith	1976	M	Primary School Teacher	76
92734	Felicia	Grace	Smith	1993	F	Student	76
98342	Nancy	Emily Grace	Smith	1998	F	Student	76
24973	Wilbur	Chase	Bucket	1956	M	Postal Worker	83
22534	Monique	Joelle	Floquet	1955	F	Painter	83
67342	Sandy		Leopard	1978	M	Systems Manager	14
66583	Carmela	Ghere	Ghimenti	1976	F	Chef	61
96743	Jeremy	Ralph	Prosit	1995	M	Student	61
86739	Phillip		Leopard	1993	M	Student	14

We have added two new fields to the original census data. The PID is a unique identifier for each person. HID is a unique identifier for each household. No PID value is repeated in the table, but HID will be the same for all the people in one household.