Example 2.7: In Example 2.6, the form of either Fig. 2.9 or Fig. 2.10 is acceptable, because the key is a single attribute. However, in a situation where the key has more than one attribute, we must use the style of Fig. 2.10. For instance, the relation `Movie`, whose key is the pair of attributes `title` and `year`, must be declared as in Fig. 2.11. However, as usual, `UNIQUE` is an option to replace `PRIMARY KEY`. □

```
CREATE TABLE Movies (
    title       CHAR(100),
    year        INT,
    length      INT,
    genre       CHAR(10),
    studioName  CHAR(30),
    producerC#  INT,
    PRIMARY KEY (title, year)
);
```

Figure 2.11: Making `title` and `year` be the key of `Movies`

2.3.7 Exercises for Section 2.3

Exercise 2.3.1: In this exercise we introduce one of our running examples of a relational database schema. The database schema consists of four relations, whose schemas are:

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

The `Product` relation gives the manufacturer, model number and type (PC, laptop, or printer) of various products. We assume for convenience that model numbers are unique over all manufacturers and product types; that assumption is not realistic, and a real database would include a code for the manufacturer as part of the model number. The `PC` relation gives for each model number that is a PC the speed (of the processor, in gigahertz), the amount of RAM (in megabytes), the size of the hard disk (in gigabytes), and the price. The `Laptop` relation is similar, except that the screen size (in inches) is also included. The `Printer` relation records for each printer model whether the printer produces color output (true, if so), the process type (laser or ink-jet, typically), and the price.

Write the following declarations:

a) A suitable schema for relation `Product`.

b) A suitable schema for relation `Laptop`.

c) A suitable schema for relation `Printer`.

d) A suitable schema for relation `PC`.

e) An alteration to your `Printer` schema from (d) to delete the attribute `color`.

f) An alteration to your `Laptop` schema from (c) to add the attribute `od` (optical-disk type, e.g., cd or dvd). Let the default value for this attribute be `'none'` if the laptop does not have an optical disk.

Exercise 2.3.2: This exercise introduces another running example, concerning World War II capital ships. It involves the following relations:

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

Ships are built in "classes" from the same design, and the class is usually named for the first ship of that class. The relation `Classes` records the name of the class, the type (`'bb'` for battleship or `'bc'` for battlecruiser), the country that built the ship, the number of main guns, the bore (diameter of the gun barrel, in inches) of the main guns, and the displacement (weight, in tons). Relation `Ships` records the name of the ship, the name of its class, and the year in which the ship was launched. Relation `Battles` gives the name and date of battles involving these ships, and relation `Outcomes` gives the result (sunk, damaged, or ok) for each ship in each battle.

Write the following declarations:

a) A suitable schema for relation `Ships`.

b) A suitable schema for relation `Outcomes`.

c) A suitable schema for relation `Classes`.

d) A suitable schema for relation `Battles`.

e) An alteration to your `Classes` relation from (a) to delete the attribute `bore`.

f) An alteration to your `Ships` relation from (b) to include the attribute `yard` giving the shipyard where the ship was built.

The first step computes the relation of the interior node labeled $\sigma_{length \geq 100}$ in Fig. 2.18, and the second step computes the node labeled $\sigma_{studioName='Fox'}$. Notice that we get renaming "for free," since we can use any attributes and relation name we wish for the left side of an assignment. The last two steps compute the intersection and the projection in the obvious way.

It is also permissible to combine some of the steps. For instance, we could combine the last two steps and write:

$$\texttt{R(t,y,l,i,s,p)} := \sigma_{length \geq 100}(\texttt{Movies})$$
$$\texttt{S(t,y,l,i,s,p)} := \sigma_{studioName='Fox'}(\texttt{Movies})$$
$$\texttt{Answer(title, year)} := \pi_{t,y}(\texttt{R} \cap \texttt{S})$$

We could even substitute for R and S in the last line and write the entire expression in one line. □

2.4.14 Exercises for Section 2.4

Exercise 2.4.1: This exercise builds upon the products schema of Exercise 2.3.1. Recall that the database schema consists of four relations, whose schemas are:

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

Some sample data for the relation `Product` is shown in Fig. 2.20. Sample data for the other three relations is shown in Fig. 2.21. Manufacturers and model numbers have been "sanitized," but the data is typical of products on sale at the beginning of 2007.

Write expressions of relational algebra to answer the following queries. You may use the linear notation of Section 2.4.13 if you wish. For the data of Figs. 2.20 and 2.21, show the result of your query. However, your answer should work for arbitrary data, not just the data of these figures.

a) Find those manufacturers that sell printers, but not PC's.

b) What PC models have a speed of at least 2.50?

c) Which manufacturers make laptops with a hard disk of at least 120GB?

d) Find the model number and price of all products (of any type) made by manufacturer C.

e) Find the model numbers of all black-and-white laser printers.

! f) Find those hard-disk sizes that occur in two or more PC's.

maker	*model*	*type*
A	1001	pc
A	1002	pc
A	1003	pc
A	2004	laptop
A	2005	laptop
A	2006	laptop
B	1004	pc
B	1005	pc
B	1006	pc
B	2007	laptop
C	1007	pc
D	1008	pc
D	1009	pc
D	1010	pc
D	3004	printer
D	3005	printer
E	1011	pc
E	1012	pc
E	1013	pc
E	2001	laptop
E	2002	laptop
E	2003	laptop
E	3001	printer
E	3002	printer
E	3003	printer
F	2008	laptop
F	2009	laptop
G	2010	laptop
H	3006	printer
H	3007	printer

Figure 2.20: Sample data for Product

model	*speed*	*ram*	*hd*	*price*
1001	2.66	1024	250	2114
1002	2.10	512	250	995
1003	1.42	512	80	478
1004	2.80	1024	250	649
1005	3.20	512	250	630
1006	3.20	1024	320	1049
1007	2.20	1024	200	510
1008	2.20	2048	250	770
1009	2.00	1024	250	650
1010	2.80	2048	300	770
1011	1.86	2048	160	959
1012	2.80	1024	160	649
1013	3.06	512	80	529

(a) Sample data for relation PC

model	*speed*	*ram*	*hd*	*screen*	*price*
2001	2.00	2048	240	20.1	3673
2002	1.73	1024	80	17.0	949
2003	1.80	512	60	15.4	549
2004	2.00	512	60	13.3	1150
2005	2.16	1024	120	17.0	2500
2006	2.00	2048	80	15.4	1700
2007	1.83	1024	120	13.3	1429
2008	1.60	1024	100	15.4	900
2009	1.60	512	80	14.1	680
2010	2.00	2048	160	15.4	2300

(b) Sample data for relation Laptop

model	*color*	*type*	*price*
3001	true	ink-jet	99
3002	false	laser	239
3003	true	laser	899
3004	true	ink-jet	120
3005	false	laser	120
3006	true	ink-jet	100
3007	true	laser	200

(c) Sample data for relation Printer

Figure 2.21: Sample data for relations of Exercise 2.4.1

! g) Find those pairs of PC models that have both the same speed and RAM. A pair should be listed only once; e.g., list (i, j) but not (j, i).

!! h) Find those manufacturers of at least two different computers (PC's or laptops) with speeds of at least 2.20.

!! k) Find the manufacturers who sell exactly three different models of PC.

!! j) Find the manufacturer(s) of the computer (PC or laptop) with the highest available speed.

!! k) Find the manufacturers of PC's with at least three different speeds.

Exercise 2.4.2: Draw expression trees for each of your expressions of Exercise 2.4.1.

Exercise 2.4.3: This exercise builds upon Exercise 2.3.2 concerning World War II capital ships. Recall it involves the following relations:

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

Figures 2.22 and 2.23 give some sample data for these four relations.[4] Note that, unlike the data for Exercise 2.4.1, there are some "dangling tuples" in this data, e.g., ships mentioned in `Outcomes` that are not mentioned in `Ships`.

Write expressions of relational algebra to answer the following queries. You may use the linear notation of Section 2.4.13 if you wish. For the data of Figs. 2.22 and 2.23, show the result of your query. However, your answer should work for arbitrary data, not just the data of these figures.

a) Find the ships launched prior to 1917.

b) Find the ships sunk in the battle of Surigao Strait.

c) The treaty of Washington in 1921 prohibited capital ships heavier than 35,000 tons. List the ships that violated the treaty of Washington.

d) List the name, displacement, and number of guns of the ships engaged in the battle of North Cape.

e) List all the capital ships mentioned in the database. (Remember that all these ships may not appear in the `Ships` relation.)

f) Give the class names and countries of the classes that carried guns of at least 16-inch bore.

[4] Source: J. N. Westwood, *Fighting Ships of World War II*, Follett Publishing, Chicago, 1975 and R. C. Stern, *US Battleships in Action*, Squadron/Signal Publications, Carrollton, TX, 1980.

class	*type*	*country*	*numGuns*	*bore*	*displacement*
Bismarck	bb	Germany	8	15	42000
Iowa	bb	USA	9	16	46000
Kongo	bc	Japan	8	14	32000
South Dakota	bb	USA	9	16	37000
Renown	bc	Gt. Britain	6	15	32000
Revenge	bb	Gt. Britain	8	15	29000
Mississippi	bb	USA	12	14	33000
Yamato	bb	Japan	9	18	65000

(a) Sample data for relation Classes

name	*date*
Denmark Strait	5/24-27/41
Guadalcanal	11/15/42
North Cape	12/26/43
Surigao Strait	10/25/44

(b) Sample data for relation Battles

ship	*battle*	*result*
Arizona	Pearl Harbor	sunk
Bismarck	Denmark Strait	sunk
California	Surigao Strait	ok
Duke of York	North Cape	ok
Fuso	Surigao Strait	sunk
Hood	Denmark Strait	sunk
King George V	Denmark Strait	ok
Kirishima	Guadalcanal	sunk
Prince of Wales	Denmark Strait	damaged
Rodney	Denmark Strait	ok
Scharnhorst	North Cape	sunk
South Dakota	Guadalcanal	damaged
Tennessee	Surigao Strait	ok
Washington	Guadalcanal	ok
West Virginia	Surigao Strait	ok
Yamashiro	Surigao Strait	sunk

(c) Sample data for relation Outcomes

Figure 2.22: Data for Exercise 2.4.3

name	*class*	*launched*
Alabama	South Dakota	1942
Haruna	Kongo	1915
Hiei	Kongo	1914
Idaho	Mississippi	1919
Iowa	Iowa	1943
Kirishima	Kongo	1915
Kongo	Kongo	1913
Missouri	Iowa	1944
Musashi	Yamato	1942
New Jersey	Iowa	1943
New Mexico	Mississippi	1918
Ramillies	Revenge	1917
Renown	Renown	1916
Repulse	Renown	1916
Resolution	Revenge	1916
Revenge	Revenge	1916
Royal Oak	Revenge	1916
Royal Sovereign	Revenge	1916
South Dakota	South Dakota	1942
Wisconsin	Iowa	1944
Yamato	Yamato	1941

Figure 2.23: Sample data for relation Ships

! g) Find those countries that had both battleships and battlecruisers.

! h) Find those ships that "lived to fight another day"; they were damaged in one battle, but later fought in another.

! i) Find the classes that had only one ship as a member of that class.

Exercise 2.4.4: Draw expression trees for each of your expressions of Exercise 2.4.3.

Exercise 2.4.5: What is the difference between the natural join $R \bowtie S$ and the theta-join $R \bowtie_C S$ where the condition C is that $R.A = S.A$ for each attribute A appearing in the schemas of both R and S?

! **Exercise 2.4.6:** An operator on relations is said to be *monotone* if whenever we add a tuple to one of its arguments, the result contains all the tuples that it contained before adding the tuple, plus perhaps more tuples. Which of the operators described in this section are monotone? For each, either explain why it is monotone or give an example showing it is not.

! **Exercise 2.4.7:** Suppose relations R and S have n tuples and m tuples, respectively. Give the minimum and maximum numbers of tuples that the results of the following expressions can have.

a) $R \cup S$.

b) $R \bowtie S$.

c) $\sigma_C(R) \times S$, for some condition C.

d) $\pi_L(R) - S$, for some list of attributes L.

! **Exercise 2.4.8:** The *semijoin* of relations R and S, written $R \ltimes S$, is the set of tuples t in R such that there is at least one tuple in S that agrees with t in all attributes that R and S have in common. Give three different expressions of relational algebra that are equivalent to $R \ltimes S$.

! **Exercise 2.4.9:** The *antisemijoin* $R \overline{\ltimes} S$ is the set of tuples t in R that do *not* agree with any tuple of S in the attributes common to R and S. Give an expression of relational algebra equivalent to $R \overline{\ltimes} S$.

!! **Exercise 2.4.10:** Let R be a relation with schema

$$(A_1, A_2, \ldots, A_n, B_1, B_2, \ldots, B_m)$$

and let S be a relation with schema $(B_1, B_2, \ldots, B_m)$; that is, the attributes of S are a subset of the attributes of R. The *quotient* of R and S, denoted $R \div S$, is the set of tuples t over attributes $A_1, A_2, \ldots, A_n$ (i.e., the attributes of R that are not attributes of S) such that for every tuple s in S, the tuple ts, consisting of the components of t for $A_1, A_2, \ldots, A_n$ and the components of s for $B_1, B_2, \ldots, B_m$, is a member of R. Give an expression of relational algebra, using the operators we have defined previously in this section, that is equivalent to $R \div S$.

2.5 Constraints on Relations

We now take up the third important aspect of a data model: the ability to restrict the data that may be stored in a database. So far, we have seen only one kind of constraint, the requirement that an attribute or attributes form a key (Section 2.3.6). These and many other kinds of constraints can be expressed in relational algebra. In this section, we show how to express both key constraints and "referential-integrity" constraints; the latter require that a value appearing in one column of one relation also appear in some other column of the same or a different relation. In Chapter 7, we see how SQL database systems can enforce the same sorts of constraints as we can express in relational algebra.

Example 2.25: Suppose we wish to require that one must have a net worth of at least \$10,000,000 to be the president of a movie studio. We can express this constraint algebraically as follows. First, we need to theta-join the two relations

```
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

using the condition that presC# from Studio and cert# from MovieExec are equal. That join combines pairs of tuples consisting of a studio and an executive, such that the executive is the president of the studio. If we select from this relation those tuples where the net worth is less than ten million, we have a set that, according to our constraint, must be empty. Thus, we may express the constraint as:

$$\sigma_{netWorth<10000000}(\texttt{Studio} \bowtie_{presC\#=cert\#} \texttt{MovieExec}) = \emptyset$$

An alternative way to express the same constraint is to compare the set of certificates that represent studio presidents with the set of certificates that represent executives with a net worth of at least \$10,000,000; the former must be a subset of the latter. The containment

$$\pi_{presC\#}(\texttt{Studio}) \subseteq \pi_{cert\#}\Big(\sigma_{netWorth\geq 10000000}(\texttt{MovieExec})\Big)$$

expresses the above idea. □

2.5.5 Exercises for Section 2.5

Exercise 2.5.1: Express the following constraints about the relations of Exercise 2.3.1, reproduced here:

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

You may write your constraints either as containments or by equating an expression to the empty set. For the data of Exercise 2.4.1, indicate any violations to your constraints.

a) A PC with a processor speed less than 3.00 must not sell for more than \$800.

b) A laptop with a screen size less than 15.4 inches must have at least a 120 gigabyte hard disk or sell for less than \$1000.

! c) No manufacturer of PC's may also make printers.

! d) If a laptop has a larger main memory than a PC, then the laptop must also have a higher price than the PC.

!! e) A manufacturer of a PC must also make a laptop with at least as great a processor speed.

Exercise 2.5.2: Express the following constraints in relational algebra. The constraints are based on the relations of Exercise 2.3.2:

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

You may write your constraints either as containments or by equating an expression to the empty set. For the data of Exercise 2.4.3, indicate any violations to your constraints.

a) No class of ships may have guns with larger than 18-inch bore.

b) If a class of ships has more than 10 guns, then their bore must be no larger than 15 inches.

! c) No class may have more than 3 ships.

! d) No country may have both battleships and battlecruisers.

!! e) No ship with more than 10 guns may be in a battle with a ship having fewer than 9 guns that was sunk.

! Exercise 2.5.3: Suppose R and S are two relations. Let C be the referential integrity constraint that says: whenever R has a tuple with some values $v_1, v_2, \ldots, v_n$ in particular attributes $A_1, A_2, \ldots, A_n$, there must be a tuple of S that has the same values $v_1, v_2, \ldots, v_n$ in particular attributes $B_1, B_2, \ldots, B_n$. Show how to express constraint C in relational algebra.

! Exercise 2.5.4: Another algebraic way to express a constraint is $E_1 = E_2$, where both E_1 and E_2 are relational-algebra expressions. Can this form of constraint express more than the two forms we discussed in this section?

2.6 Summary of Chapter 2

✦ *Data Models*: A data model is a notation for describing the structure of the data in a database, along with the constraints on that data. The data model also normally provides a notation for describing operations on that data: queries and data modifications.

The computation of the join is as follows. Tuple $(1,2)$ from R and $(4,5)$ from S meet the join condition. Since each appears twice in its relation, the number of times the joined tuple appears in the result is 2×2 or 4. The other possible join of tuples — $(1,2)$ from R with $(2,3)$ from S — fails to meet the join condition, so this combination does not appear in the result. □

5.1.7 Exercises for Section 5.1

Exercise 5.1.1: Let PC be the relation of Fig. 2.21(a), and suppose we compute the projection $\pi_{hd}(\text{PC})$. What is the value of this expression as a set? As a bag? What is the average value of tuples in this projection, when treated as a set? As a bag?

Exercise 5.1.2: Repeat Exercise 5.1.1 for the projection $\pi_{speed}(\text{PC})$.

Exercise 5.1.3: This exercise refers to the "battleship" relations of Exercise 2.4.3.

a) The expression $\pi_{numGuns}(\texttt{Classes})$ yields a single-column relation with the numbers of guns of the various classes. For the data of Exercise 2.4.3, what is this relation as a set? As a bag?

! b) Write an expression of relational algebra to give the numbers of guns of the ships (not the classes). Your expression must make sense for bags; that is, the number of times a value g appears must be the number of ships that have g guns.

! **Exercise 5.1.4:** Certain algebraic laws for relations as sets also hold for relations as bags. Explain why each of the laws below hold for bags as well as sets.

a) The commutative law for union: $(R \cup S) = (S \cup R)$.

b) The commutative law for intersection: $(R \cap S) = (S \cap R)$.

c) The commutative law for natural join: $(R \bowtie S) = (S \bowtie R)$.

d) The associative law for union: $(R \cup S) \cup T = R \cup (S \cup T)$.

e) The associative law for intersection: $(R \cap S) \cap T = R \cap (S \cap T)$.

f) The associative law for natural join: $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$.

g) $\pi_L(R \cup S) = \pi_L(R) \cup \pi_L(S)$. Here, L is an arbitrary list of attributes.

h) The distributive law of union over intersection:

$$R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$$

i) $\sigma_{C \text{ AND } D}(R) = \sigma_C(R) \cap \sigma_D(R)$. Here, C and D are arbitrary conditions about the tuples of R.

!! **Exercise 5.1.5:** The following algebraic laws hold for sets but not for bags. Explain why they hold for sets and give counterexamples to show that they do not hold for bags.

a) The distributive law of intersection over union:

$$T \cap (R \cup S) = (T \cap R) \cup (T \cap S)$$

b) $\sigma_{C \text{ OR } D}(R) = \sigma_C(R) \cup \sigma_D(R)$. Here, C and D are arbitrary conditions about the tuples of R.

c) $(S \cap T) - R = S \cap (T - R)$.

5.2 Extended Operators of Relational Algebra

Section 2.4 presented the classical relational algebra, and Section 5.1 introduced the modifications necessary to treat relations as bags of tuples rather than sets. The ideas of these two sections serve as a foundation for most of modern query languages. However, languages such as SQL have several other operations that have proved quite important in applications. Thus, a full treatment of relational operations must include a number of other operators, which we introduce in this section. The additions:

1. The *duplicate-elimination operator* δ turns a bag into a set by eliminating all but one copy of each tuple.

2. *Aggregation operators*, such as sums or averages, are not operations of relational algebra, but are used by the grouping operator (described next). Aggregation operators apply to attributes (columns) of a relation; e.g., the sum of a column produces the one number that is the sum of all the values in that column.

3. *Grouping* of tuples according to their value in one or more attributes has the effect of partitioning the tuples of a relation into "groups." Aggregation can then be applied to columns within each group, giving us the ability to express a number of queries that are impossible to express in the classical relational algebra. The *grouping operator* γ is an operator that combines the effect of grouping and aggregation.

4. *Extended projection* gives additional power to the operator π. In addition to projecting out some columns, in its generalized form π can perform computations involving the columns of its argument relation to produce new columns.

The ordering is performed on the result of the FROM, WHERE, and other clauses, just before we apply the SELECT clause. The tuples of this result are then sorted by the attributes in the list of the ORDER BY clause, and then passed to the SELECT clause for processing in the normal manner.

Example 6.11: The following is a rewrite of our original query of Example 6.1, asking for the Disney movies of 1990 from the relation

```
Movies(title, year, length, genre, studioName, producerC#)
```

To get the movies listed by length, shortest first, and among movies of equal length, alphabetically, we can say:

```
SELECT *
FROM Movies
WHERE studioName = 'Disney' AND year = 1990
ORDER BY length, title;
```

A subtlety of ordering is that all the attributes of Movies are available at the time of sorting, even if they are not part of the SELECT clause. Thus, we could replace SELECT * by SELECT producerC#, and the query would still be legal. □

An additional option in ordering is that the list following ORDER BY can include expressions, just as the SELECT clause can. For instance, we can order the tuples of a relation $R(A, B)$ by the sum of the two components of the tuples, highest first, with:

```
SELECT *
FROM R
ORDER BY A+B DESC;
```

6.1.9 Exercises for Section 6.1

Exercise 6.1.1: If a query has a SELECT clause

```
SELECT X Y
```

how do we know whether X and Y are two different attributes or Y is an alias of X?

Exercise 6.1.2: Write the following queries, based on our running movie database example

```
Movies(title, year, length, genre, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

in SQL.

a) Find Aishwara Rai's birthdate.

b) Find the address of Film City.

c) Find all the stars that appeared either in a movie made in 2000 or a movie with "Story" in the title.

d) Find all the stars who either are female or live in Mumbai (have string `Mumbai` as a part of their address).

e) Find all executives worth at least $20,000,000.

Exercise 6.1.3: Write the following queries in SQL. They refer to the database schema of Exercise 2.4.1:

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

Show the result of your queries using the data from Exercise 2.4.1.

a) Find the model number, memory size, and screen size for laptops costing more than $1200.

b) Find all the tuples in the `Printer` relation for color printers. Remember that `color` is a boolean-valued attribute.

c) Find the model number and hard-disk size for those PC's that have a speed of 3.0 and a price less than $1000.

d) Find the model number, speed, and hard-disk size for all PC's whose price is under $800.

e) Do the same as (a), but rename the `speed` column `gigahertz` and the `ram` column `gigabytes`.

f) Find the manufacturers of laptops.

Exercise 6.1.4: Write the following queries based on the database schema of Exercise 2.4.3:

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

and show the result of your query on the data of Exercise 2.4.3.

a) Find the class name and country for all classes with at least 12 guns.

b) Find the names of all ships launched prior to 1915, but call the resulting column `shipName`.

c) Find the names of all ships that begin with the letter "M."

d) Find the names of ships sunk in battle and the name of the battle in which they were sunk.

e) Find all ships that have the same name as their class.

! f) Find the names of all ships whose name consists of three or more words (e.g., King George V).

Exercise 6.1.5: Let a and b be integer-valued attributes that may be `NULL` in some tuples. For each of the following conditions (as may appear in a `WHERE` clause), describe exactly the set of (a, b) tuples that satisfy the condition, including the case where a and/or b is `NULL`.

a) `a < 50 OR a >= 50`

b) `a = 0 OR b = 10`

c) `a = 20 AND b = 10`

! d) `a = b`

! e) `a > b`

! **Exercise 6.1.6:** In Example 6.10 we discussed the query

```
SELECT *
FROM Movies
WHERE length <= 120 OR length > 120;
```

which behaves unintuitively when the length of a movie is `NULL`. Find a simpler, equivalent query, one with a single condition in the `WHERE` clause (no `AND` or `OR` of conditions).

6.2 Queries Involving More Than One Relation

Much of the power of relational algebra comes from its ability to combine two or more relations through joins, products, unions, intersections, and differences. We get all of these operations in SQL. The set-theoretic operations — union, intersection, and difference — appear directly in SQL, as we shall learn in Section 6.2.5. First, we shall learn how the select-from-where statement of SQL allows us to perform products and joins.

6.2.6 Exercises for Section 6.2

Exercise 6.2.1: Using the database schema of our running movie example

```
Movies(title, year, length, genre, studioName, producerC#)
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

write the following queries in SQL.

a) Who is the president of Film City?

b) Who were the male stars in *Monsoon Wedding*?

c) Which stars appeared in movies produced by Sony in 2005?

! d) Which executives are worth more than Subhash Ghai?

! e) Which movies are longer than *Bride and Prejudice*?

Exercise 6.2.2: Write the following queries, based on the database schema

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

of Exercise 2.4.1, and evaluate your queries using the data of that exercise.

a) Find those manufacturers that sell PC's but not Laptops.

b) Give the manufacturer and speed of laptops with a hard disk of at least 100 gigabytes.

c) Find the model number and price of all products (of any type) made by manufacturer C.

! d) Find those pairs of PC models that have both the same RAM and hard disk. A pair should be listed only once; e.g., list (i, j) but not (j, i).

! e) Find those processor speeds that occur in two or more PC's.

!! f) Find those manufacturers of at least two different computers (PC's or laptops) with speeds of at least 2.0.

Exercise 6.2.3: Write the following queries, based on the database schema

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

of Exercise 2.4.3, and evaluate your queries using the data of that exercise.

a) List the name, displacement, and number of guns of the ships engaged in the battle of North Cape.

b) Find the ships heavier than 37,000 tons.

c) List all the ships mentioned in the database. (Remember that all these ships may not appear in the `Ships` relation.)

! d) Find those battles with at least three ships of the same country.

! e) Find those countries that have both battleships and battlecruisers.

! f) Find those ships that were damaged in one battle, but later fought in another.

! **Exercise 6.2.4:** A general form of relational-algebra query is

$$\pi_L\Big(\sigma_C(R_1 \times R_2 \times \cdots \times R_k)\Big)$$

Here, L is an arbitrary list of attributes, and C is an arbitrary condition. The list of relations $R_1, R_2, \ldots, R_k$ may include the same relation repeated several times, in which case appropriate renaming may be assumed applied to the R_i's. Show how to express any query of this form in SQL.

! **Exercise 6.2.5:** Another general form of relational-algebra query is

$$\pi_L\Big(\sigma_C(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_k)\Big)$$

The same assumptions as in Exercise 6.2.4 apply here; the only difference is that the natural join is used instead of the product. Show how to express any query of this form in SQL.

6.3 Subqueries

In SQL, one query can be used in various ways to help in the evaluation of another. A query that is part of another is called a *subquery*. Subqueries can have subqueries, and so on, down as many levels as we wish. We already saw one example of the use of subqueries; in Section 6.2.5 we built a union, intersection, or difference query by connecting two subqueries to form the whole query. There are a number of other ways that subqueries can be used:

6.3.9 Exercises for Section 6.3

Exercise 6.3.1: Write the following queries, based on the database schema

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

of Exercise 2.4.1. You should use at least one subquery in each of your answers and write each query in two significantly different ways (e.g., using different sets of the operators `EXISTS`, `IN`, `ALL`, and `ANY`).

a) Find the makers of laptops with a speed of at least 2.0.

b) Find the printers with the highest price.

! c) Find the laptops whose speed is slower than that of the fastest PC.

! d) Find the model number of the item (PC, laptop, or printer) with the lowest price.

! e) Find the maker of the color printer with the highest price.

!! f) Find the maker(s) of the PC(s) with the fastest processor among all those PC's that have the greatest amount of RAM.

Exercise 6.3.2: Write the following queries, based on the database schema

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
Battles(name, date)
Outcomes(ship, battle, result)
```

of Exercise 2.4.3. You should use at least one subquery in each of your answers and write each query in two significantly different ways (e.g., using different sets of the operators `EXISTS`, `IN`, `ALL`, and `ANY`).

a) Find the countries whose ships had guns of the largest bore.

b) Find the names of the ships with 9 guns.

c) Find the battles in which ships of the South Dakota class participated.

! d) Find the classes of ships, at least one of which was sunk in a battle.

!! e) Find the names of the ships whose bore was the largest for those ships with the same number of guns.

! **Exercise 6.3.3:** Write the query of Fig. 6.10 without any subqueries.

! **Exercise 6.3.4:** Write the following queries without using the intersection or difference operators:

a) The intersection query of Fig. 6.5.

b) The difference query of Example 6.17.

!! **Exercise 6.3.5:** We have noticed that certain operators of SQL are redundant, in the sense that they always can be replaced by other operators. For example, we saw that s `IN` R can be replaced by s `= ANY` R. Show that `EXISTS` and `NOT EXISTS` are redundant by explaining how to replace any expression of the form `EXISTS` R or `NOT EXISTS` R by an expression that does not involve `EXISTS` (except perhaps in the expression R itself). *Hint*: Remember that it is permissible to have a constant in the `SELECT` clause.

Exercise 6.3.6: For these relations from our running movie database schema

```
StarsIn(movieTitle, movieYear, starName)
MovieStar(name, address, gender, birthdate)
MovieExec(name, address, cert#, netWorth)
Studio(name, address, presC#)
```

describe the tuples that would appear in the following SQL expressions:

a) `StarsIn CROSS JOIN MovieStar;`

b) `Studio NATURAL FULL OUTER JOIN MovieExec;`

c) `StarsIn FULL OUTER JOIN MovieStar ON name = starName;`

! **Exercise 6.3.7:** Consider expression $\pi_L(R_1 \bowtie R_2 \bowtie \cdots \bowtie R_k)$ of relational algebra, where L is a list of attributes all of which belong to R_1. Show that this expression can be written in SQL using subqueries only. More precisely, write an equivalent SQL expression where no `FROM` clause has more than one relation in its list.

! **Exercise 6.3.8:** Using the database schema

```
Product(maker, model, type)
PC(model, speed, ram, hd, rd, price)
Laptop(model, speed, ram, hd, screen, price)
Printer(model, color, type, price)
```

write a SQL query that will produce information about all products — PC's, laptops, and printers — including their manufacturer if available, and whatever information about that product is relevant (i.e., found in the relation for that type of product).

Exercise 6.3.9: Using the two relations

```
Classes(class, type, country, numGuns, bore, displacement)
Ships(name, class, launched)
```

from our database schema of Exercise 2.4.3, write a SQL query that will produce all available information about ships, including that information available in the `Classes` relation. You need not produce information about classes if there are no ships of that class mentioned in `Ships`.

! **Exercise 6.3.10:** Repeat Exercise 6.3.9, but also include in the result, for any class C that is not mentioned in `Ships`, information about the ship that has the same name C as its class. You may assume that there is a ship with the class name, even if it doesn't appear in `Ships`.

! **Exercise 6.3.11:** The join operators (other than outerjoin) we learned in this section are redundant, in the sense that they can always be replaced by select-from-where expressions. Explain how to write expressions of the following forms using select-from-where:

a) `R CROSS JOIN S;`

b) `R NATURAL JOIN S;`

c) `R JOIN S ON` C`;`, where C is a SQL condition.

6.4 Full-Relation Operations

In this section we shall study some operations that act on relations as a whole, rather than on tuples individually or in small numbers (as do joins of several relations, for instance). First, we deal with the fact that SQL uses relations that are bags rather than sets, and a tuple can appear more than once in a relation. We shall see how to force the result of an operation to be a set in Section 6.4.1, and in Section 6.4.2 we shall see that it is also possible to prevent the elimination of duplicates in circumstances where SQL systems would normally eliminate them.

Then, we discuss how SQL supports the grouping and aggregation operator γ that we introduced in Section 5.2.4. SQL has aggregation operators and a `GROUP-BY` clause. There is also a "`HAVING`" clause that allows selection of certain groups in a way that depends on the group as a whole, rather than on individual tuples.

6.4.1 Eliminating Duplicates

As mentioned in Section 6.3.4, SQL's notion of relations differs from the abstract notion of relations presented in Section 2.2. A relation, being a set, cannot have more than one copy of any given tuple. When a SQL query creates a new relation, the SQL system does not ordinarily eliminate duplicates. Thus, the SQL response to a query may list the same tuple several times.