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.

```
SELECT name, SUM(length)
FROM MovieExec, Movies
WHERE producerC# = cert#
GROUP BY name
HAVING MIN(year) < 1930;
```

Figure 6.14: Computing the total length of film for early producers

6.4.8 Exercises for Section 6.4

Exercise 6.4.1: Write each of the queries in Exercise 2.4.3 in SQL, making sure that duplicates are eliminated.

Exercise 6.4.2: Write each of the queries in Exercise 2.4.1 in SQL, making sure that duplicates are eliminated.

! **Exercise 6.4.3:** For each of your answers to Exercise 6.3.1, determine whether or not the result of your query can have duplicates. If so, rewrite the query to eliminate duplicates. If not, write a query without subqueries that has the same, duplicate-free answer.

! **Exercise 6.4.4:** Repeat Exercise 6.4.3 for your answers to Exercise 6.3.2.

! **Exercise 6.4.5:** In Example 6.27, we mentioned that different versions of the query "find the producers of Harrison Ford's movies" can have different answers as bags, even though they yield the same set of answers. Consider the version of the query in Example 6.22, where we used a subquery in the `FROM` clause. Does this version produce duplicates, and if so, why?

Exercise 6.4.6: 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 the average hard-disk size of PC's.

b) Find the average price of laptops with speed at least 3.0.

c) Find the average price of PC's made by manufacturer "A."

! d) Find the average price of PC's and laptops made by manufacturer "D."

e) Find, for each different price, the average speed of a PC.

! f) Find the manufacturers that make at least three different models of PC.

! g) Find for each manufacturer who sells PC's the maximum speed of a PC.

! h) Find, for each speed of PC above 2.5, the average hard-disk size.

! i) Find for each manufacturer, the average speed of its laptops.

!! j) Find the average hard-disk size of a PC for all those manufacturers that make laptops.

Exercise 6.4.7: 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) Find the number of battle-cruiser classes.

b) Find the average gun bore of battleship classes.

! c) Find the average gun bore of battleships. Note the difference between parts (b) and (c); do we weight a class by the number of ships of that class or not?

! d) Find for each class the number of ships of that class sunk in battle.

! e) Find for each class the year in which the last ship of that class was launched.

!! f) Find for each class with at least two ships the number of ships of that class sunk in battle.

!! g) The weight (in pounds) of the shell fired from a naval gun is approximately one half the cube of the bore (in inches). Find the average weight of the shell for each country's ships.

Exercise 6.4.8: In Example 5.10 we gave an example of the query: "find, for each star who has appeared in at least three movies, the earliest year in which they appeared." We wrote this query as a γ operation. Write it in SQL.

! **Exercise 6.4.9:** The γ operator of extended relational algebra does not have a feature that corresponds to the `HAVING` clause of SQL. Is it possible to mimic a SQL query with a `HAVING` clause in relational algebra? If so, how would we do it in general?

6.5 Da

To this point
where staten
return a resu
shall focus o

1. Insert

2. Delete

3. Update

We refer to t

6.5.1 In

The basic fo

A tuple i
the list of at
tuple createc

Example 6
stars of *The*

```
1) IN
2) VA
```

The effect of
on line (2) is
are mention
values on lin
so 'The Mal
movieTitle

If, as in
then we may
we could jus

```
INSER
VALUE
```

However, if
the same as

6.5.4 Exercises for Section 6.5

Exercise 6.5.1: Write the following database modifications, 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. Describe the effect of the modifications on the data of that exercise.

a) Delete all PC's with less than 200 gigabytes of hard disk.

b) Using two `INSERT` statements, store in the database the fact that PC model 1500 is made by manufacturer A, has speed 3.1, RAM 1024, hard disk 300, and sells for $2499.

c) Delete all laptops made by a manufacturer that doesn't make PC's.

d) Manufacturer B buys manufacturer C. Change all products made by C so they are now made by B.

e) For each PC, double the amount of hard disk and add 1024 megabytes to the amount of RAM. (Remember that several attributes can be changed by one `UPDATE` statement.)

! f) For each laptop made by manufacturer D, add one inch to the screen size and subtract $200 from the price.

! g) Insert the facts that for every laptop there is a PC with the same manufacturer, speed, RAM, and hard disk, a model number 1100 less, and a price that is $500 less.

Exercise 6.5.2: Write the following database modifications, 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. Describe the effect of the modifications on the data of that exercise.

a) The two British battleships of the Nelson class — Nelson and Rodney — were both launched in 1927, had nine 16-inch guns, and a displacement of 34,000 tons. Insert these facts into the database.

b) Two of the three battleships of the Italian Vittorio Veneto class — Vittorio Veneto and Italia — were launched in 1940; the third ship of that class, Roma, was launched in 1942. Each had nine 15-inch guns and a displacement of 41,000 tons. Insert these facts into the database.

c) Delete all classes with fewer than three ships.

d) Delete from `Ships` all ships sunk in battle.

e) Modify the `Classes` relation so that gun bores are measured in centimeters (one inch = 2.5 centimeters) and displacements are measured in metric tons (one metric ton = 1.1 tons).

6.6 Transactions in SQL

To this point, our model of operations on the database has been that of one user querying or modifying the database. Thus, operations on the database are executed one at a time, and the database state left by one operation is the state upon which the next operation acts. Moreover, we imagine that operations are carried out in their entirety ("atomically"). That is, we assumed it is impossible for the hardware or software to fail in the middle of a modification, leaving the database in a state that cannot be explained as the result of the operations performed on it.

Real life is often considerably more complicated. We shall first consider what can happen to leave the database in a state that doesn't reflect the operations performed on it, and then we shall consider the tools SQL gives the user to assure that these problems do not occur.

6.6.1 Serializability

In applications like Web services, banking, or airline reservations, hundreds of operations per second may be performed on the database. The operations initiate at any of thousands or millions of sites, such as desktop computers or automatic teller machines. It is entirely possible that we could have two operations affecting the same bank account or flight, and for those operations to overlap in time. If so, they might interact in strange ways.

Here is an example of what could go wrong if the DBMS were completely unconstrained as to the order in which it operated upon the database. This example involves a database interacting with people, and it is intended to illustrate why it is important to control the sequences in which interacting events can occur. However, a DBMS would not control events that were so "large" that they involved waiting for a user to make a choice. The event sequences controlled by the DBMS involve only the execution of SQL statements.

Example 6.40: The typical airline gives customers a Web interface where they can choose a seat for their flight. This interface shows a map of available then a seat that is available on the first query at Step (1) will remain available at subsequent queries.

However, suppose some new tuples enter the relation `Flights`. For example, the airline may have switched the flight to a larger plane, creating some new tuples that weren't there before. Then under repeatable-read isolation, a subsequent query for available seats may also retrieve the new seats. □

Figure 6.17 summarizes the differences between the four SQL isolation levels.

Isolation Level	Dirty Reads	Nonrepeatable Reads	Phantoms
Read Uncommitted	Allowed	Allowed	Allowed
Read Committed	Not Allowed	Allowed	Allowed
Repeatable Read	Not Allowed	Not Allowed	Allowed
Serializable	Not Allowed	Not Allowed	Not Allowed

Figure 6.17: Properties of SQL isolation levels

6.6.7 Exercises for Section 6.6

Exercise 6.6.1: This and the next exercises involve certain programs that operate on the two relations

```
Product(maker, model, type)
PC(model, speed, ram, hd, price)
```

from our running PC exercise. Sketch the following programs, including SQL statements and work done in a conventional language. Do not forget to issue `BEGIN TRANSACTION`, `COMMIT`, and `ROLLBACK` statements at the proper times and to tell the system your transactions are read-only if they are.

a) Given a model number, delete the tuple for that model from both `PC` and `Product`.

b) Given a model number, decrease the price of that model PC by $50.

c) Given a maker, model number, processor speed, RAM size, hard-disk size, and price, check that there is no product with that model. If there is such a model, print an error message for the user. If no such model existed in the database, enter the information about that model into the `PC` and `Product` tables.

d) Given a speed and amount of hard disk (as arguments of the function), look up the PC's with that speed and hard-disk size, printing the model number and price of each.

! **Exercise 6.6.2:** Fo
icity problems, if an
of an execution of th

!! **Exercise 6.6.3:** Su
"forever," and at ea
3.5 or more and sells
and terminates. Du
one of the four prog
four isolation levels
uncommitted — tell

! **Exercise 6.6.4:** Su
grams of Exercise 6.6
or a different one of t
time. What behavio
run with isolation le
all ran with isolation
is any of the progran

6.7 Summa

✦ *SQL*: The lang
database syste
SQL3. Comme

✦ *Select-From-W*
the form selec
relations (the
(the `WHERE` cla

✦ *Subqueries*: S
within a `WHER`
`EXISTS`, `IN`, `A`
ditions about
clause.

✦ *Set Operations*
difference of re
defining the rel
respectively.

✦ *Join Expressio*
applied to rela
in a `FROM` claus

! **Exercise 6.6.2:** For each of the programs of Exercise 6.6.1, discuss the atomicity problems, if any, that could occur should the system crash in the middle of an execution of the program.

!! **Exercise 6.6.3:** Suppose we have a transaction T that is a function which runs "forever," and at each hour checks whether there is a PC that has a speed of 3.5 or more and sells for under \$1000. If it finds one, it prints the information and terminates. During this time, other transactions that are executions of one of the four programs described in Exercise 6.6.1 may run. For each of the four isolation levels — serializable, repeatable read, read committed, and read uncommitted — tell what the effect on T of running at this isolation level is.

! **Exercise 6.6.4:** Suppose we execute as a transaction T one of the four programs of Exercise 6.6.1, while other transactions that are executions of the same or a different one of the four programs may also be executing at about the same time. What behaviors of transaction T may be observed if all the transactions run with isolation level `READ UNCOMMITTED` that would not be possible if they all ran with isolation level `SERIALIZABLE`? Consider separately the case that T is any of the programs (a) through (d) of Exercise 6.6.1.

6.7 Summary of Chapter 6

✦ *SQL*: The language SQL is the principal query language for relational database systems. The most recent full standard is called SQL-99 or SQL3. Commercial systems generally vary from this standard.

✦ *Select-From-Where Queries*: The most common form of SQL query has the form select-from-where. It allows us to take the product of several relations (the `FROM` clause), apply a condition to the tuples of the result (the `WHERE` clause), and produce desired components (the `SELECT` clause).

✦ *Subqueries*: Select-from-where queries can also be used as subqueries within a `WHERE` clause or `FROM` clause of another query. The operators `EXISTS`, `IN`, `ALL`, and `ANY` may be used to express boolean-valued conditions about the relations that are the result of a subquery in a `WHERE` clause.

✦ *Set Operations on Relations*: We can take the union, intersection, or difference of relations by connecting the relations, or connecting queries defining the relations, with the keywords `UNION`, `INTERSECT`, and `EXCEPT`, respectively.

✦ *Join Expressions*: SQL has operators such as `NATURAL JOIN` that may be applied to relations, either as queries by themselves or to define relations in a `FROM` clause.