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 $\pi_{hd}(\texttt{PC})$. 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 $\pi_{speed}(\texttt{PC})$.

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.

A	$U.B$	$U.C$	$V.B$	$V.C$	D
4	5	6	2	3	10
4	5	6	2	3	11
7	8	9	2	3	10
7	8	9	2	3	11
1	2	3	$\perp$	$\perp$	$\perp$
$\perp$	$\perp$	$\perp$	6	7	12

Figure 5.7: Result of a theta-outerjoin

5.2.8 Exercises for Section 5.2

Exercise 5.2.1: Here are two relations:

$R(A,B)$: $\{(1,2),\ (3,4),\ (1,2),\ (3,5),\ (4,5)\}$

$S(B,C)$: $\{(1,2),\ (3,5),\ (3,6),\ (4,5),\ (1,3),\ (4,5)\}$

Compute the following: a) $\pi_{A^2,B^2,A+B}(R)$; b) $\pi_{B-1,C+1}(S)$; c) $\tau_{A,B}(R)$; d) $\tau_{C,B}(S)$; e) $\delta(R)$; f) $\delta(S)$; g) $\gamma_{A,\ \mathtt{AVG}(B)}(R)$; h) $\gamma_{B,\mathtt{SUM}(C)}(S)$; ! i) $\gamma_A(R)$; ! j) $\gamma_{A,\mathtt{MAX}(C)}(R \bowtie S)$; k) $R \overset{\circ}{\bowtie}_R S$; l) $R \overset{\circ}{\bowtie}_L S$; m) $R \overset{\circ}{\bowtie} S$; n) $R \overset{\circ}{\bowtie}_{R.B<S.B} S$.

! **Exercise 5.2.2:** A unary operator f is said to be *idempotent* if for all relations R, $f(f(R)) = f(R)$. That is, applying f more than once is the same as applying it once. Which of the following operators are idempotent? Either explain why or give a counterexample.

a) π_L; b) σ_C; c) γ_L; d) τ; e) δ.

! **Exercise 5.2.3:** One thing that can be done with an extended projection, but not with the original version of projection that we defined in Section 2.4.5, is to duplicate columns. For example, if $R(A,B)$ is a relation, then $\pi_{A,A}(R)$ produces the tuple (a,a) for every tuple (a,b) in R. Can this operation be done using only the classical operations of relation algebra from Section 2.4? Explain your reasoning.

5.3 A Logic for Relations

As an alternative to abstract query languages based on algebra, one can use a form of logic to express queries. The logical query language *Datalog* ("database logic") consists of if-then rules. Each of these rules expresses the idea that from certain combinations of tuples in certain relations, we may infer that some other tuple must be in some other relation, or in the answer to a query.

is the head relation defined by this rule. More generally, had tuple $(1,2)$ appeared n times in R and tuple $(2,3)$ appeared m times in S, then tuple $(1,3)$ would appear nm times in H. □

If a relation is defined by several rules, then the result is the bag-union of whatever tuples are produced by each rule.

Example 5.23: Consider a relation H defined by the two rules

```
H(x,y) ← S(x,y) AND x>1
H(x,y) ← S(x,y) AND y<5
```

where relation $S(B,C)$ is as in Example 5.22; that is, $S = \{(2,3), (4,5), (4,5)\}$. The first rule puts each of the three tuples of S into H, since they each have a first component greater than 1. The second rule puts only the tuple $(2,3)$ into H, since $(4,5)$ does not satisfy the condition $y < 5$. Thus, the resulting relation H has two copies of the tuple $(2,3)$ and two copies of the tuple $(4,5)$. □

5.3.7 Exercises for Section 5.3

Exercise 5.3.1: Write each of the queries of Exercise 2.4.3 in Datalog. You should use only safe rules, but you may wish to use several IDB predicates corresponding to subexpressions of complicated relational-algebra expressions.

Exercise 5.3.2: Write each of the queries of Exercise 2.4.1 in Datalog. Again, use only safe rules, but you may use several IDB predicates if you like.

!! **Exercise 5.3.3:** The requirement we gave for safety of Datalog rules is sufficient to guarantee that the head predicate has a finite relation if the predicates of the relational subgoals have finite relations. However, this requirement is too strong. Give an example of a Datalog rule that violates the condition, yet whatever finite relations we assign to the relational predicates, the head relation will be finite.

5.4 Relational Algebra and Datalog

Each of the relational-algebra operators of Section 2.4 can be mimicked by one or several Datalog rules. In this section we shall consider each operator in turn. We shall then consider how to combine Datalog rules to mimic complex algebraic expressions. It is also true that any single safe Datalog rule can be expressed in relational algebra, although we shall not prove that fact here. However, Datalog queries are more powerful than relational algebra when several rules are allowed to interact; they can express recursions that are not expressable in the algebra (see Example 5.35).

5.4.7 Comparison Between Datalog and Relational Algebra

We see from Section 5.4.6 that every expression in the basic relational algebra of Section 2.4 can be expressed as a Datalog query. There are operations in the extended relational algebra, such as grouping and aggregation from Section 5.2, that have no corresponding features in the Datalog version we have presented here. Likewise, Datalog does not support bag operations such as duplicate elimination.

It is also true that any single Datalog rule can be expressed in relational algebra. That is, we can write a query in the basic relational algebra that produces the same set of tuples as the head of that rule produces.

However, when we consider collections of Datalog rules, the situation changes. Datalog rules can express recursion, which relational algebra can not. The reason is that IDB predicates can also be used in the bodies of rules, and the tuples we discover for the heads of rules can thus feed back to rule bodies and help produce more tuples for the heads. We shall not discuss here any of the complexities that arise, especially when the rules have negated subgoals. However, the following example will illustrate recursive Datalog.

Example 5.35: Suppose we have a relation `Edge(X,Y)` that says there is a directed edge (arc) from node X to node Y. We can express the transitive closure of the edge relation, that is, the relation `Path(X,Y)` meaning that there is a path of length 1 or more from node X to node Y, as follows:

1. `Path(X,Y)` $\leftarrow$ `Edge(X,Y)`
2. `Path(X,Y)` $\leftarrow$ `Edge(X,Z) AND Path(Z,Y)`

Rule (1) says that every edge is a path. Rule (2) says that if there is an edge from node X to some node Z and a path from Z to Y, then there is also a path from X to Y. If we apply Rule (1) and then Rule (2), we get the paths of length 2. If we take the *Path* facts we get from this application and use them in another application of Rule (2), we get paths of length 3. Feeding those *Path* facts back again gives us paths of length 4, and so on. Eventually, we discover all possible path facts, and on one round we get no new facts. At that point, we can stop. If we haven't discovered the fact $Path(a, b)$, then there really is no path in the graph from node a to node b. □

5.4.8 Exercises for Section 5.4

Exercise 5.4.1: Let $R(a, b, c)$, $S(a, b, c)$, and $T(a, b, c)$ be three relations. Write one or more Datalog rules that define the result of each of the following expressions of relational algebra:

a) $R \cap S$.

b) $R \cup S$.

c) $\pi_{a,b}(R)$.

d) $R - S$.

e) $(R \cup S) - T$.

! f) $\pi_{a,b}(R) \cap \rho_{U(a,b)}\big(\pi_{b,c}(S)\big)$.

! g) $(R - S) \cap (R - T)$.

Exercise 5.4.2: Let $R(x, y, z)$ be a relation. Write one or more Datalog rules that define $\sigma_C(R)$, where C stands for each of the following conditions:

a) $y = z$.

b) $x > y$ `AND` $y > z$.

c) $x > y$ `OR` $y > z$.

d) `NOT` $(x > y$ `OR` $x < y)$.

! e) `NOT` $\big((x > y$ `OR` $x < y)$ `AND` $y > z\big)$.

! f) `NOT` $\big((x > y$ `OR` $x > z)$ `AND` $y > z\big)$.

Exercise 5.4.3: Let $R(a, b, c)$, $S(b, c, d)$, and $T(d, e)$ be three relations. Write single Datalog rules for each of the natural joins:

a) $S \bowtie T$.

b) $R \bowtie S$.

c) $(S \bowtie T) \bowtie R$. (*Note*: since the natural join is associative and commutative, the order of the join of these three relations is irrelevant.)

Exercise 5.4.4: Let $R(x, y, z)$ and $S(x, y, z)$ be two relations. Write one or more Datalog rules to define each of the theta-joins $R \bowtie_C S$, where C is one of the conditions of Exercise 5.4.2. For each of these conditions, interpret each arithmetic comparison as comparing an attribute of R on the left with an attribute of S on the right. For instance, $x < y$ stands for $R.x < S.y$.

! **Exercise 5.4.5:** It is also possible to convert Datalog rules into equivalent relational-algebra expressions. While we have not discussed the method of doing so in general, it is possible to work out many simple examples. For each of the Datalog rules below, write an expression of relational algebra that defines the same relation as the head of the rule.

a) `T(x,y) ← S(x,z) AND S(z,y)`

b) `T(x,y) ← S(x,z) AND R(z,y)`

c) `T(x,y) ← S(x,z) AND R(z,y) AND x < y`