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

b) The distributive law of intersection over union:

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

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

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.

5. The *sorting operator* τ turns a relation into a list of tuples, sorted according to one or more attributes. This operator should be used judiciously, because some relational-algebra operators do not make sense on lists. We can, however, apply selections or projections to lists and expect the order of elements on the list to be preserved in the output.

6. The *outerjoin* operator is a variant of the join that avoids losing dangling tuples. In the result of the outerjoin, dangling tuples are "padded" with the null value, so the dangling tuples can be represented in the output.

5.2.1 Duplicate Elimination

Sometimes, we need an operator that converts a bag to a set. For that purpose, we use $\delta(R)$ to return the set consisting of one copy of every tuple that appears one or more times in relation R.

Example 5.8: If R is the relation

A	B
1	2
3	4
1	2
1	2

from Fig. 5.1, then $\delta(R)$ is

A	B
1	2
3	4

Note that the tuple $(1,2)$, which appeared three times in R, appears only once in $\delta(R)$. □

5.2.2 Aggregation Operators

There are several operators that apply to sets or bags of numbers or strings. These operators are used to summarize or "aggregate" the values in one column of a relation, and thus are referred to as *aggregation* operators. The standard operators of this type are:

1. `SUM` produces the sum of a column with numerical values.

2. `AVG` produces the average of a column with numerical values.

3. `MIN` and `MAX`, applied to a column with numerical values, produces the smallest or largest value, respectively. When applied to a column with character-string values, they produce the lexicographically (alphabetically) first or last value, respectively.

4. COUNT produces the number of (not necessarily distinct) values in a column. Equivalently, COUNT applied to any attribute of a relation produces the number of tuples of that relation, including duplicates.

Example 5.9: Consider the relation

A	*B*
1	2
3	4
1	2
1	2

Some examples of aggregations on the attributes of this relation are:

1. SUM(B) $= 2 + 4 + 2 + 2 = 10$.
2. AVG(A) $= (1 + 3 + 1 + 1)/4 = 1.5$.
3. MIN(A) $= 1$.
4. MAX(B) $= 4$.
5. COUNT(A) $= 4$.

□

5.2.3 Grouping

Often we do not want simply the average or some other aggregation of an entire column. Rather, we need to consider the tuples of a relation in groups, corresponding to the value of one or more other columns, and we aggregate only within each group. As an example, suppose we wanted to compute the total number of minutes of movies produced by each studio, i.e., a relation such as:

studioName	*sumOfLengths*
Disney	12345
MGM	54321
...	...

Starting with the relation

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

from our example database schema of Section 2.2.8, we must group the tuples according to their value for attribute studioName. We must then sum the length column within each group. That is, we imagine that the tuples of Movies are grouped as suggested in Fig. 5.4, and we apply the aggregation SUM(length) to each group independently.

	studioName	
	Disney	
	Disney	
	Disney	
	MGM	
	MGM	
	○	
	○	
	○	

Figure 5.4: A relation with imaginary division into groups

5.2.4 The Grouping Operator

We shall now introduce an operator that allows us to group a relation and/or aggregate some columns. If there is grouping, then the aggregation is within groups.

The subscript used with the γ operator is a list L of elements, each of which is either:

a) An attribute of the relation R to which the γ is applied; this attribute is one of the attributes by which R will be grouped. This element is said to be a *grouping attribute*.

b) An aggregation operator applied to an attribute of the relation. To provide a name for the attribute corresponding to this aggregation in the result, an arrow and new name are appended to the aggregation. The underlying attribute is said to be an *aggregated attribute*.

The relation returned by the expression $\gamma_L(R)$ is constructed as follows:

1. Partition the tuples of R into *groups*. Each group consists of all tuples having one particular assignment of values to the grouping attributes in the list L. If there are no grouping attributes, the entire relation R is one group.

2. For each group, produce one tuple consisting of:

   i. The grouping attributes' values for that group and

   ii. The aggregations, over all tuples of that group, for the aggregated attributes on list L.

Example 5.10: Suppose we have the relation

```
StarsIn(title, year, starName)
```

δ is a Special Case of γ

Technically, the δ operator is redundant. If $R(A_1, A_2, \ldots, A_n)$ is a relation, then $\delta(R)$ is equivalent to $\gamma_{A_1,A_2,\ldots,A_n}(R)$. That is, to eliminate duplicates, we group on all the attributes of the relation and do no aggregation. Then each group corresponds to a tuple that is found one or more times in R. Since the result of γ contains exactly one tuple from each group, the effect of this "grouping" is to eliminate duplicates. However, because δ is such a common and important operator, we shall continue to consider it separately when we study algebraic laws and algorithms for implementing the operators.

One can also see γ as an extension of the projection operator on sets. That is, $\gamma_{A_1,A_2,\ldots,A_n}(R)$ is also the same as $\pi_{A_1,A_2,\ldots,A_n}(R)$, if R is a set. However, if R is a bag, then γ eliminates duplicates while π does not.

and we wish to find, for each star who has appeared in at least three movies, the earliest year in which they appeared. The first step is to group, using `starName` as a grouping attribute. We clearly must compute for each group the `MIN(year)` aggregate. However, in order to decide which groups satisfy the condition that the star appears in at least three movies, we must also compute the `COUNT(title)` aggregate for each group.

We begin with the grouping expression

$$\gamma_{starName,\ \mathtt{MIN}(year)\rightarrow minYear,\ \mathtt{COUNT}(title)\rightarrow ctTitle}(\mathtt{StarsIn})$$

The first two columns of the result of this expression are needed for the query result. The third column is an auxiliary attribute, which we have named `ctTitle`; it is needed to determine whether a star has appeared in at least three movies. That is, we continue the algebraic expression for the query by selecting for `ctTitle >= 3` and then projecting onto the first two columns. An expression tree for the query is shown in Fig. 5.5. □

5.2.5 Extending the Projection Operator

Let us reconsider the projection operator $\pi_L(R)$ introduced in Section 2.4.5. In the classical relational algebra, L is a list of (some of the) attributes of R. We extend the projection operator to allow it to compute with components of tuples as well as choose components. In *extended projection*, also denoted $\pi_L(R)$, projection lists can have the following kinds of elements:

1. A single attribute of R.

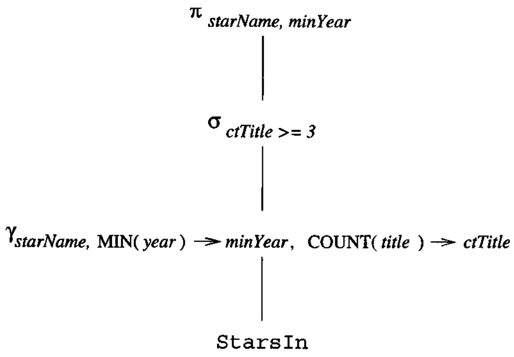

Figure 5.5: Algebraic expression tree for the query of Example 5.10

2. An expression $x \rightarrow y$, where x and y are names for attributes. The element $x \rightarrow y$ in the list L asks that we take the attribute x of R and *rename* it y; i.e., the name of this attribute in the schema of the result relation is y.

3. An expression $E \rightarrow z$, where E is an expression involving attributes of R, constants, arithmetic operators, and string operators, and z is a new name for the attribute that results from the calculation implied by E. For example, $a+b \rightarrow x$ as a list element represents the sum of the attributes a and b, renamed x. Element $c \,||\, d \rightarrow e$ means concatenate the presumably string-valued attributes c and d and call the result e.

The result of the projection is computed by considering each tuple of R in turn. We evaluate the list L by substituting the tuple's components for the corresponding attributes mentioned in L and applying any operators indicated by L to these values. The result is a relation whose schema is the names of the attributes on list L, with whatever renaming the list specifies. Each tuple of R yields one tuple of the result. Duplicate tuples in R surely yield duplicate tuples in the result, but the result can have duplicates even if R does not.

Example 5.11: Let R be the relation

A	B	C
0	1	2
0	1	2
3	4	5

Then the result of $\pi_{A,B+C\rightarrow X}(R)$ is

A	X
0	3
0	3
3	9

The result's schema has two attributes. One is A, the first attribute of R, not renamed. The second is the sum of the second and third attributes of R, with the name X.

For another example, $\pi_{B-A\rightarrow X, C-B\rightarrow Y}(R)$ is

X	Y
1	1
1	1
1	1

Notice that the calculation required by this projection list happens to turn different tuples $(0,1,2)$ and $(3,4,5)$ into the same tuple $(1,1)$. Thus, the latter tuple appears three times in the result. □

5.2.6 The Sorting Operator

There are several contexts in which we want to sort the tuples of a relation by one or more of its attributes. Often, when querying data, one wants the result relation to be sorted. For instance, in a query about all the movies in which Sean Connery appeared, we might wish to have the list sorted by title, so we could more easily find whether a certain movie was on the list. We shall also see when we study query optimization how execution of queries by the DBMS is often made more efficient if we sort the relations first.

The expression $\tau_L(R)$, where R is a relation and L a list of some of R's attributes, is the relation R, but with the tuples of R sorted in the order indicated by L. If L is the list $A_1, A_2, \ldots, A_n$, then the tuples of R are sorted first by their value of attribute A_1. Ties are broken according to the value of A_2; tuples that agree on both A_1 and A_2 are ordered according to their value of A_3, and so on. Ties that remain after attribute A_n is considered may be ordered arbitrarily.

Example 5.12: If R is a relation with schema $R(A,B,C)$, then $\tau_{C,B}(R)$ orders the tuples of R by their value of C, and tuples with the same C-value are ordered by their B value. Tuples that agree on both B and C may be ordered arbitrarily. □

If we apply another operator such as join to the sorted result of a τ, the sorted order usually becomes meaningless, and the elements on the list should be treated as a bag, not a list. However, bag projections can be made to preserve the order. Also, a selection on a list drops out the tuples that do not satisfy the condition of the selection, but the remaining tuples can be made to appear in their original sorted order.

5.2.7 Outerjoins

A property of the join operator is that it is possible for certain tuples to be "dangling"; that is, they fail to match any tuple of the other relation in the common attributes. Dangling tuples do not have any trace in the result of the join, so the join may not represent the data of the original relations completely. In cases where this behavior is undesirable, a variation on the join, called "outerjoin," has been proposed and appears in various commercial systems.

We shall consider the "natural" case first, where the join is on equated values of all attributes in common to the two relations. The *outerjoin* $R \overset{\circ}{\bowtie} S$ is formed by starting with $R \bowtie S$, and adding any dangling tuples from R or S. The added tuples must be padded with a special *null* symbol, $\perp$, in all the attributes that they do not possess but that appear in the join result. Note that $\perp$ is written `NULL` in SQL (recall Section 2.3.4).

A	B	C
1	2	3
4	5	6
7	8	9

(a) Relation U

B	C	D
2	3	10
2	3	11
6	7	12

(b) Relation V

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

(c) Result $U \overset{\circ}{\bowtie} V$

Figure 5.6: Outerjoin of relations

Example 5.13: In Fig. 5.6(a) and (b) we see two relations U and V. Tuple $(1,2,3)$ of U joins with both $(2,3,10)$ and $(2,3,11)$ of V, so these three tuples are not dangling. However, the other three tuples — $(4,5,6)$ and $(7,8,9)$ of U and $(6,7,12)$ of V — are dangling. That is, for none of these three tuples is there a tuple of the other relation that agrees with it on both the B and C components. Thus, in $U \overset{\circ}{\bowtie} V$, seen in Fig. 5.6(c), the three dangling tuples are padded with $\perp$ in the attributes that they do not have: attribute D for the tuples of U and attribute A for the tuple of V. □

There are many variants of the basic (natural) outerjoin idea. The *left outerjoin* $R \overset{\circ}{\bowtie}_L S$ is like the outerjoin, but only dangling tuples of the left argument R are padded with $\perp$ and added to the result. The *right outerjoin* $R \overset{\circ}{\bowtie}_R S$ is like the outerjoin, but only the dangling tuples of the right argument S are padded with $\perp$ and added to the result.

Example 5.14: If U and V are as in Fig. 5.6, then $U \overset{\circ}{\bowtie}_L V$ is:

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

and $U \overset{\circ}{\bowtie}_R V$ is:

A	B	C	D
1	2	3	10
1	2	3	11
$\perp$	6	7	12

□

In addition, all three natural outerjoin operators have theta-join analogs, where first a theta-join is taken and then those tuples that failed to join with any tuple of the other relation, when the condition of the theta-join was applied, are padded with $\perp$ and added to the result. We use $\overset{\circ}{\bowtie}_C$ to denote a theta-outerjoin with condition C. This operator can also be modified with L or R to indicate left- or right-outerjoin.

Example 5.15: Let U and V be the relations of Fig. 5.6, and consider

$$U \overset{\circ}{\bowtie}_{A>V.C} V$$

Tuples $(4,5,6)$ and $(7,8,9)$ of U each satisfy the condition with both of the tuples $(2,3,10)$ and $(2,3,11)$ of V. Thus, none of these four tuples are dangling in this theta-join. However, the two other tuples — $(1,2,3)$ of U and $(6,7,12)$ of V — are dangling. They thus appear, padded, in the result shown in Fig. 5.7. □

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:

$R(A,B)$: $\{(0,1),\ (2,3),\ (0,1),\ (2,4),\ (3,4)\}$

$S(B,C)$: $\{(0,1),\ (2,4),\ (2,5),\ (3,4),\ (0,2),\ (3,4)\}$

Compute the following: a) $\pi_{A+B,A^2,B^2}(R)$; b) $\pi_{B+1,C-1}(S)$; c) $\tau_{B,A}(R)$; d) $\tau_{B,C}(S)$; e) $\delta(R)$; f) $\delta(S)$; g) $\gamma_{A,\ \text{SUM}(B)}(R)$; h) $\gamma_{B,\text{AVG}(C)}(S)$; ! i) $\gamma_A(R)$; ! j) $\gamma_{A,\text{MAX}(C)}(R \bowtie S)$; k) $R \overset{\circ}{\bowtie}_L S$; l) $R \overset{\circ}{\bowtie}_R S$; 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, $f\big(f(R)\big) = f(R)$. 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) δ; b) π_L; c) σ_C; d) γ_L; e) τ.

! **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.