combination of values of the shared attributes—Name, SSN, Department. The result relation, STUDENT_OR_INSTRUCTOR, will have the following attributes:

STUDENT_OR_INSTRUCTOR(Name, SSN, Department, Advisor, Rank)

All the tuples from both relations are included in the result, but tuples with the same (Name, SSN, Department) combination will appear only once in the result. Tuples appearing only in STUDENT will have a null for the Rank attribute, whereas tuples appearing only in INSTRUCTOR will have a null for the Advisor attribute. A tuple that exists in both relations, such as a student who is also an instructor, will have values for all its attributes.[10]

Notice that the same person may still appear twice in the result. For example, we could have a graduate student in the Mathematics department who is an instructor in the Computer Science department. Although the two tuples representing that person in STUDENT and INSTRUCTOR will have the same (Name, SSN) values, they will not agree on the Department value, and so will not be matched. This is because Department has two separate meanings in STUDENT (the department where the person studies) and INSTRUCTOR (the department where the person is employed as an instructor). If we wanted to union persons based on the same (Name, SSN) combination only, we should rename the Department attribute in each table to reflect that they have different meanings, and designate them as not being part of the union-compatible attributes.

Another capability that exists in most commercial languages (but not in the basic relational algebra) is that of specifying operations on values after they are extracted from the database. For example, arithmetic operations such as +, −, and * can be applied to numeric values that appear in the result of a query.

6.5 Examples of Queries in Relational Algebra

We now give additional examples to illustrate the use of the relational algebra operations. All examples refer to the database of Figure 5.6. In general, the same query can be stated in numerous ways using the various operations. We will state each query in one way and leave it to the reader to come up with equivalent formulations.

QUERY 1

Retrieve the name and address of all employees who work for the 'Research' department.

$\text{RESEARCH\_DEPT} \leftarrow \sigma_{\text{DNAME='Research'}}(\text{DEPARTMENT})$

$\text{RESEARCH\_EMPS} \leftarrow (\text{RESEARCH\_DEPT} \bowtie_{\text{DNUMBER=DNOEMPLOYEE}})$

$\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, ADDRESS}}(\text{RESEARCH\_EMPS})$

10. Notice that OUTER UNION is equivalent to a FULL OUTER JOIN if the join attributes are *all* the common attributes of the two relations.

This query could be specified in other ways; for example, the order of the JOIN and SELECT operations could be reversed, or the JOIN could be replaced by a NATURAL JOIN after renaming one of the join attributes.

QUERY 2

For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

$\text{STAFFORD\_PROJS} \leftarrow \sigma_{\text{PLOCATION='STAFFORD'}}(\text{PROJECT})$

$\text{CONTR\_DEPT} \leftarrow (\text{STAFFORD\_PROJS} \bowtie_{\text{DNUM=DNUMBER}} \text{DEPARTMENT})$

$\text{PROJ\_DEPT\_MGR} \leftarrow (\text{CONTR\_DEPT} \bowtie_{\text{MGRSSN=SSN}} \text{EMPLOYEE})$

$\text{RESULT} \leftarrow \pi_{\text{PNUMBER, DNUM, LNAME, ADDRESS, BDATE}}(\text{PROJ\_DEPT\_MGR})$

QUERY 3

Find the names of employees who work on *all* the projects controlled by department number 5.

$\text{DEPT5\_PROJS(PNO)} \leftarrow \pi_{\text{PNUMBER}}(\sigma_{\text{DNUM=5}}(\text{PROJECT}))$

$\text{EMP\_PROJ(SSN, PNO)} \leftarrow \pi_{\text{ESSN, PNO}}(\text{WORKS\_ON})$

$\text{RESULT\_EMP\_SSNS} \leftarrow \text{EMP\_PROJ} \div \text{DEPT5\_PROJS}$

$\text{RESULT} \leftarrow \pi_{\text{LNAME, FNAME}}(\text{RESULT\_EMP\_SSNS} * \text{EMPLOYEE})$

QUERY 4

Make a list of project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

$\text{SMITHS(ESSN)} \leftarrow \pi_{\text{SSN}}(\sigma_{\text{LNAME='SMITH'}}(\text{EMPLOYEE}))$

$\text{SMITH\_WORKER\_PROJ} \leftarrow \pi_{\text{PNO}}(\text{WORKS\_ON} * \text{SMITHS})$

$\text{MGRS} \leftarrow \pi_{\text{LNAME, DNUMBER}}(\text{EMPLOYEE} \bowtie_{\text{SSN=MGRSSN}} \text{DEPARTMENT})$

$\text{SMITH\_MANAGED\_DEPTS(DNUM)} \leftarrow \pi_{\text{DNUMBER}}(\sigma_{\text{LNAME='SMITH'}}(\text{MGRS}))$

$\text{SMITH\_MGR\_PROJS(PNO)} \leftarrow \pi_{\text{PNUMBER}}(\text{SMITH\_MANAGED\_DEPTS} * \text{PROJECT})$

$\text{RESULT} \leftarrow (\text{SMITH\_WORKER\_PROJS} \cup \text{SMITH\_MGR\_PROJS})$

QUERY 5

List the names of all employees with two or more dependents.

Strictly speaking, this query cannot be done in the *basic (original) relational algebra*. We have to use the AGGREGATE FUNCTION operation with the COUNT aggregate function. We assume that dependents of the *same* employee have *distinct* DEPENDENT_NAME values.

$T1\text{(SSN, NO\_OF\_DEPTS)} \leftarrow {}_{\text{ESSN}}\mathfrak{F}_{\text{COUNT DEPENDENT\_NAME}}(\text{DEPENDENT})$

$T2 \leftarrow \sigma_{\text{NO\_OF\_DEPS}\geq 2}(T1)$

$\text{RESULT} \leftarrow \pi_{\text{LNAME, FNAME}}(T2 * \text{EMPLOYEE})$

QUERY 6

Retrieve the names of employees who have no dependents.

This is an example of the type of query that uses the MINUS (SET DIFFERENCE) operation.

ALL_EMPS $\leftarrow \pi_{\text{SSN}}$(EMPLOYEE)

EMPS_WITH_DEPS(SSN) $\leftarrow \pi_{\text{ESSN}}$(DEPENDENT)

EMPS_WITHOUT_DEPS $\leftarrow$ (ALL_EMPS $-$ EMPS_WITH_DEPS)

RESULT $\leftarrow \pi_{\text{LNAME, FNAME}}$(EMPS_WITHOUT_DEPS * EMPLOYEE)

QUERY 7

List the names of managers who have at least one dependent.

MGRS(SSN) $\leftarrow \pi_{\text{MGRSSN}}$(DEPARTMENT)

EMPS_WITH_DEPS(SSN) $\leftarrow \pi_{\text{ESSN}}$(DEPENDENT)

MGRS_WITH_DEPS $\leftarrow$ (MGRS $\cap$ EMPS_WITH_DEPS)

RESULT $\leftarrow \pi_{\text{LNAME, FNAME}}$(MGRS_WITH_DEPS * EMPLOYEE)

As we mentioned earlier, the same query can in general be specified in many different ways. For example, the operations can often be applied in various orders. In addition, some operations can be used to replace others; for example, the INTERSECTION operation in Query 7 can be replaced by a NATURAL JOIN. As an exercise, try to do each of the above example queries using different operations.[11] In Chapter 8 and in Sections 6.6 and 6.7, we show how these queries are written in other relational languages.

6.6 The Tuple Relational Calculus

In this and the next section, we introduce another formal query language for the relational model called **relational calculus.** In relational calculus, we write one **declarative** expression to specify a retrieval request, and hence there is no description of how to evaluate a query. A calculus expression specifies *what* is to be retrieved rather than *how* to retrieve it. Therefore, the relational calculus is considered to be a **nonprocedural** language. This differs from relational algebra, where we must write a *sequence of operations* to specify a retrieval request; hence, it can be considered as a **procedural** way of stating a query. It is possible to nest algebra operations to form a single expression; however, a certain order among the operations is always explicitly specified in a relational algebra expression. This order also influences the strategy for evaluating the query. A calculus expression may be written in different ways, but the way it is written has no bearing on how a query should be evaluated.

11. When queries are optimized (see Chapter 15), the system will choose a particular sequence of operations that corresponds to an execution strategy that can be executed efficiently.