CSCI ST 31206 Database Management Systems, Harvard Summer School 2003
Relational Algebra and SQL comparisons

QUERY 1

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

\[
\text{RESEARCH.DEPT} \leftarrow \sigma_{\text{DNAME='Research'}}(\text{DEPARTMENT})
\]

\[
\text{RESEARCH.EMPS} \leftarrow (\text{RESEARCH.DEPT} \times \text{DNUMBER=DNO EMPLOYEE})
\]

\[
\text{RESULT} \leftarrow \pi_{\text{FNAME, LNAME, ADDRESS}}(\text{RESEARCH.EMPS})
\]

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).

QUERY 1

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

Q1: SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNUMBER=DNO;

Query Q1 is similar to a SELECT–PROJECT–JOIN sequence of relational algebra operations. Such queries are often called select–project–join queries. In the WHERE-clause of Q1, the condition DNAME = ‘Research’ is a selection condition and corresponds to a SELECT operation in the relational algebra. The condition DNUMBER = DNO is a join condition, which corresponds to a JOIN condition in the relational algebra. The result of query Q1 is shown in Figure 8.2(b). In general, any number of select and join conditions may be specified in a single SQL query.

In SQL the same name can be used for two (or more) attributes as long as the attributes are in different relations. If this is the case, and a query refers to two or more attributes with the same name, we must qualify the attribute name with the relation name, to prevent ambiguity. This is done by prefixing the relation name to the attribute name and separating the two by a period. To illustrate this, suppose that in Figures 7.5 and 7.6 the DNO and LNAME attributes of the EMPLOYEE relation were called DNUMBER and NAME and the DNAME attribute of DEPARTMENT was also called NAME; then, to prevent ambiguity, query Q1 would be rephrased as shown in Q1A. We must prefix the attributes NAME and DNUMBER in Q1A to specify which ones we are referring to, because the attribute names are used in both relations:

Q1A: SELECT FNAME, EMPLOYEE.NAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DEPARTMENT.NAME='Research' AND DEPARTMENT.DNUMBER=EMPLOYEE.DNUMBER;
we could specify query Q1A as in Q1B just for convenience to shorten the relation names that prefix the attributes:

Q1B: SELECT E.FNAME, E.NAME, E.ADDRESS
    FROM EMPLOYEE E, DEPARTMENT D
    WHERE D.NAME='Research' AND D.DNUMBER=E.DNUMBER;

The concept of a joined table (or joined relation) was incorporated into SQL2 to permit users to specify a table resulting from a join operation in the FROM-clause of a query. This construct may be easier to comprehend than mixing together all the select and join conditions in the WHERE-clause. For example, consider query Q1, which retrieves the name and address of every employee who works for the ‘Research’ department. It may be easier first to specify the join of the EMPLOYEE and DEPARTMENT relations, and then to select the desired tuples and attributes. This can be written in SQL2 as in Q1A:

Q1A: SELECT FNAME, LNAME, ADDRESS
    FROM (EMPLOYEE JOIN DEPARTMENT ON DNO=DNUMBER)
    WHERE DNAME='Research';

The FROM-clause in Q1A contains a single joined table. The attributes of such a table are all the attributes of the first table, EMPLOYEE, followed by all the attributes of the second table, DEPARTMENT. The concept of a joined table also allows the user to specify different types of join, such as NATURAL JOIN and various types of OUTER JOIN. In a NATURAL JOIN on two relations R and S, no join condition is specified; an implicit equi-join condition for each pair of attributes with the same name from R and S is created. Each such pair of attributes is included only once in the resulting relation (see Section 7.4.5.)

If the names of the join attributes are not the same in the base relations, it is possible to rename the attributes so that they match, and then to apply NATURAL JOIN. In this case, the AS construct can be used to rename a relation and all its attributes in the FROM clause. This is illustrated in Q1B, where the DEPARTMENT relation is renamed AS DEPT and its attributes are renamed as DNAME, DNO (to match the name of the desired join attribute DNO in EMPLOYEE), MSSN, and MSDATE. The implied join condition for this NATURAL JOIN is EMPLOYEE.DNO = DEPT.DNO, because this is the only pair of attributes with the same name after renaming:

Q1B: SELECT FNAME, LNAME, ADDRESS
    FROM (EMPLOYEE NATURAL JOIN
           (DEPARTMENT AS DEPT (DNAME, DNO, MSSN, MSDATE)))
    WHERE DNAME='Research;
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 birthdate.

\[
\text{STAFFORD\_PROJS} \leftarrow \sigma_{\text{LOCATION}=\text{'Stafford'}}(\text{PROJECT})
\]

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

\[
\text{PROJ\_DEPT\_MGR} \leftarrow (\text{CONTR\_DEPT} \bowtie_{\text{MGR\_SSN} = \text{SSN}} \text{EMPLOYEE})
\]

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

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 birthdate.

Q2: SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM = DNUMBER AND MGR\_SSN = SSN AND
PLOCATION = 'Stafford';

The join condition DNUM = DNUMBER relates a project to its controlling department, whereas the join condition MGR\_SSN = SSN relates the controlling department to the employee who manages that department. The result of query Q2 is shown in Figure 8.2(c).

The options available for specifying joined tables in SQL2 include INNER JOIN (same as JOIN), LEFT OUTER JOIN, RIGHT OUTER JOIN, and FULL OUTER JOIN. In the latter three, the keyword OUTER may be omitted. It is also possible to nest join specifications; that is, one of the tables in a join may itself be a joined table. This is illustrated by Q2A, which is a different way of specifying query Q2, using the concept of a joined table:

Q2A: SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM ((PROJECT JOIN DEPARTMENT ON DNUM = DNUMBER) JOIN EMPLOYEE ON MGR\_SSN = SSN)
WHERE PLOCATION = 'Stafford';
QUERY 3

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

```
DEPTS_PROJS(PNO) ← πPNUMBER(σDNUM=5(PROJECT))
EMP_PRJ0(SSID, PNO) ← πESSN, PNO(WORKS_ON)
RESULT_EMP_SNS ← EMP_PRJ0 ∪ DEPTS_PROJS
RESULT ← πLNAME, FNAME(RESULT_EMP_SNS * EMPLOYEE)
```

The second option is shown as Q3B below. Notice that we need two-level nesting in Q3B and that this formulation is quite a bit more complex than Q3, which used the CONTAINS comparison operator, and Q3A, which uses NOT EXISTS and EXCEPT. However, CONTAINS is not part of SQL, and not all relational systems have the EXCEPT operator even though it is part of SQL2:

```
Q3B: SELECT LNAME, FNAME FROM EMPLOYEE
WHERE NOT EXISTS (SELECT * FROM WORKS_ON B
WHERE (B.PNO IN (SELECT PNUMBER FROM PROJECT
WHERE DNUM=5))
AND NOT EXISTS (SELECT *
FROM WORKS_ON C
WHERE C.ESSN=SSID AND C.PNO=B.PNO));
```

In Q3B, the outer nested query selects any WORKS_ON (B) tuples whose PNO is of a project controlled by department 5, if there is not a WORKS_ON (C) tuple with the same PNO and the same SSN as that of the EMPLOYEE tuple under consideration in the outer query. If no such tuple exists, we select the EMPLOYEE tuple. The form of Q3B matches the following rephrasing of Query 3: select each employee such that there does not exist a project controlled by department 5 that the employee does not work on.

Notice that Query 3 is typically stated in relational algebra by using the DIVISION operation.
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.

\[
\begin{align*}
\text{SMITH}(:, \text{ESSN}) & \leftarrow \pi_{\text{SSN}}(\sigma_{\text{LNAME} = 'Smith'}(\text{EMPLOYEE})) \\
\text{SMITH}_{\text{WORKER}_\text{PROJ}} & \leftarrow \pi_{\text{PNO}}(\text{WORKS}_\text{ON} \times \text{SMITH}) \\
\text{MGRS} & \leftarrow \pi_{\text{LNAME}, \text{DNUMBER}}(\text{EMPLOYEE} \bowtie_{\text{SSN} = \text{MGRSSN}} \text{DEPARTMENT}) \\
\text{SMITH}_{\text{MANAGED}_\text{DEPTS}}(\text{DNUM}) & \leftarrow \pi_{\text{DNUMBER}}(\sigma_{\text{LNAME} = 'Smith'}(\text{MGRS})) \\
\text{SMITH}_{\text{MGR}_\text{PROJS}}(\text{PNO}) & \leftarrow \pi_{\text{PNUMBER}}(\text{SMITH}_{\text{MANAGED}_\text{DEPTS}} \times \text{PROJECT}) \\
\text{RESULT} & \leftarrow (\text{SMITH}_{\text{WORKER}_\text{PROJS}} \cup \text{SMITH}_{\text{MGR}_\text{PROJS}})
\end{align*}
\]

Q4: (SELECT DISTINCT PNUMBER FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME='Smith')

UNION

(SELECT DISTINCT PNUMBER FROM PROJECT, WORKS_ON, EMPLOYEE WHERE PNUMBER=PNO AND ESSN=SSN AND LNAME='Smith');

The first SELECT query retrieves the projects that involve a 'Smith' as manager of the department that controls the project, and the second retrieves the projects that involve a 'Smith' as a worker on the project. Notice that, if several employees have the last name 'Smith', the project names involving any of them will be retrieved. Applying the UNION operation to the two SELECT queries gives the desired result.

Q4A: SELECT DISTINCT PNUMBER FROM PROJECT WHERE PNUMBER IN (SELECT PNUMBER FROM PROJECT, DEPARTMENT, EMPLOYEE WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME='Smith')

OR

SELECT PNO FROM WORKS_ON, EMPLOYEE WHERE ESSN=SSN AND LNAME='Smith');

The first nested query selects the project numbers of projects that have a 'Smith' involved as manager, while the second selects the project numbers of projects that have a 'Smith' involved as worker. In the outer query, we select a PROJECT tuple if the PNUMBER value of that tuple is in the result of either nested query. The comparison operator IN compares a value v with a set (or multiset) of values V and evaluates to TRUE if v is one of the elements in V.
QUERY 5

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

Strictly speaking, this query cannot be done in the basic 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.

\[
T_1(\text{SSN, NO}_\text{OF}_\text{DEPTS}) \leftarrow \text{ESSN} \bowtie \text{COUNT DEPENDENT_NAME(Dependent)} \\
T_2 \leftarrow \sigma_{\text{NO}_\text{OF}_\text{DEPTS} \geq 2}(T_1) \\
\text{RESULT} \leftarrow \pi_{\text{LNAME, FNAME}}(T_2 \text{ * EMPLOYEE})
\]

In some cases we may need to use functions to select particular tuples. In such cases we specify a correlated nested query with the desired function, and we use that nested query in the WHERE-clause of an outer query. For example, to retrieve the names of all employees who have two or more dependents (Query 5), we can write:

Q5: \begin{align*}
\text{SELECT} & \quad \text{LNAME, FNAME} \\
\text{FROM} & \quad \text{EMPLOYEE} \\
\text{WHERE} & \quad (\text{SELECT COUNT (*)} \\
& \quad \text{FROM} \quad \text{DEPENDENT} \\
& \quad \text{WHERE} \quad \text{SSN=ESSN}) \geq 2;
\end{align*}

The correlated nested query counts the number of dependents that each employee has; if this is greater than or equal to 2, the employee tuple is selected.

QUERY 6

Retrieve the names of employees who have no dependents.

\[
\begin{align*}
\text{ALL}_\text{EMPS} & \leftarrow \pi_{\text{SSN(EMPLOYEE)}} \\
\text{EMPS}_\text{WITH}_\text{DEPS}(\text{SSN}) & \leftarrow \pi_{\text{ESSN(DEPENDENT)}} \\
\text{EMPS}_\text{WITHOUT}_\text{DEPS} & \leftarrow (\text{ALL}_\text{EMPS} - \text{EMPS}_\text{WITH}_\text{DEPS}) \\
\text{RESULT} & \leftarrow \pi_{\text{LNAME, FNAME}}(\text{EMPS}_\text{WITHOUT}_\text{DEPS} \text{ * EMPLOYEE})
\end{align*}
\]

QUERY 6

Retrieve the names of employees who have no dependents.

Q6: \begin{align*}
\text{SELECT} & \quad \text{FNAME, LNAME} \\
& \quad (8.3.2 \text{ excluded reading}) \\
\text{FROM} & \quad \text{EMPLOYEE} \\
\text{WHERE} & \quad \text{NOT EXISTS} \quad (\text{SELECT} \quad * \\
& \quad \text{FROM} \quad \text{DEPENDENT} \\
& \quad \text{WHERE} \quad \text{SSN=ESSN});
\end{align*}

In Q6, the correlated nested query retrieves all DEPENDENT tuples related to an EMPLOYEE tuple. If none exist, the EMPLOYEE tuple is selected. We can explain Q6 as follows: for each EMPLOYEE tuple, the correlated nested query selects all DEPENDENT tuples whose ESSN value matches the EMPLOYEE SSN; if the result is empty, no dependents are related to the employee, so we select that EMPLOYEE tuple and retrieve its FNAME and LNAME. There is another SQL function UNIQUE(Q) that returns TRUE if there are no duplicate tuples in the result of query Q; otherwise, it returns FALSE.
QUERY 7

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

\[ \text{MGRS}(\text{SSN}) \leftarrow \pi_{\text{MGRSSN}}(\text{DEPARTMENT}) \]
\[ \text{EMPS}_{\text{WITH}}_{\text{DEPS}}(\text{SSN}) \leftarrow \pi_{\text{ESSN}}(\text{DEPENDENT}) \]
\[ \text{MGRS}_{\text{WITH}}_{\text{DEPS}} \leftarrow (\text{MGRS} \cap \text{EMPS}_{\text{WITH}}_{\text{DEPS}}) \]
\[ \text{RESULT} \leftarrow \pi_{\text{LNAME}, \text{FNAME}}(\text{MGRS}_{\text{WITH}}_{\text{DEPS}} \ast \text{EMPLOYEE}) \]

QUERY 7

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

Q7: SELECT FNAME, LNAME FROM EMPLOYEE WHERE EXISTS (SELECT * FROM DEPENDENT WHERE SSN=ESSN) AND EXISTS (SELECT * FROM DEPARTMENT WHERE SSN=MGRSSN);

One way to write this query is shown in Q7, where we specify two nested correlated queries; the first selects all DEPENDENT tuples related to an EMPLOYEE, and the second selects all DEPARTMENT tuples managed by the EMPLOYEE. If at least one of the first and at least one of the second exist, we select the EMPLOYEE tuple. Can you rewrite this query using only a single nested query or no nested queries?