**Binary Relational Operations: JOIN and DIVISION**

**The JOIN Operation**The JOIN operation, denoted by (bowtie symbol), is used to combine related tuples from two relations into single “longer” tuples. This operation is very important for any relational database with more than a single relation because it allows us to process relationships among relations.

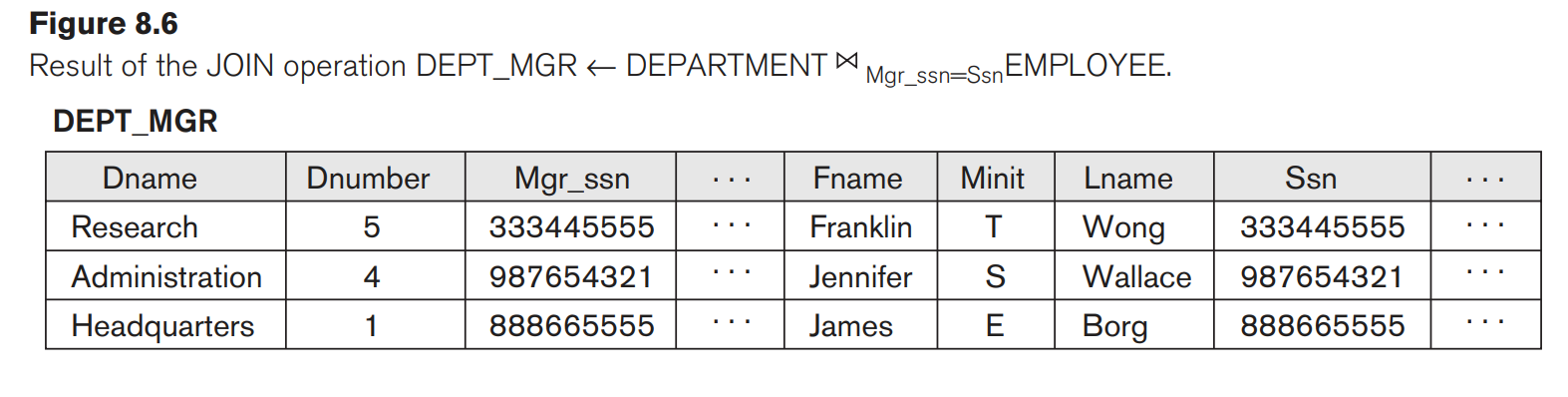
To illustrate JOIN, suppose that we want to retrieve the name of the manager of each department. To get the manager’s name, we need to combine each department tuple with the employee tuple whose Ssn value matches the Mgr\_ssn value in the department tuple.

We do this by using the JOIN operation and then projecting the result over the necessary attributes, as follows:

DEPT\_MGR ← DEPARTMENT Mgr\_ssn=Ssn EMPLOYEE

RESULT ← π Dname, Lname, Fname(DEPT\_MGR)

The first operation is illustrated in Figure 8.6.



Note that Mgr\_ssn is a foreign key of the DEPARTMENT relation that references Ssn, the primary key of the EMPLOYEE relation.

The general form of a JOIN operation on two relations R(A1, A2, … , An) and S(B1, B2, … , Bm) is

R <join condition>S

The result of the JOIN is a relation Q with n + m attributes Q(A1, A2, … , An, B1, B2,… , Bm) in that order; Q has one tuple for each combination of tuples—one from R and one from S—whenever the combination satisfies the join condition. This is the main difference between CARTESIAN PRODUCT and JOIN. In JOIN, only combinations of tuples satisfying the join condition appear in the result, whereas in the CARTESIAN PRODUCT all combinations of tuples are included in the result.

A general join condition is of the form <condition> AND <condition> AND … AND <condition> where each <condition> is of the form Ai θ Bj , Ai is an attribute of R, Bj is an attribute of S, Ai and Bj have the same domain, and θ (theta) is one of the comparison operators {=, <, ≤, >, ≥, ≠}. A JOIN operation with such a general join condition is called a THETA JOIN.

Tuples whose join attributes are NULL or for which the join condition is FALSE do not appear in the result.

**The EQUIJOIN**

The most common use of JOIN involves join conditions with equality comparisons only. Such a JOIN, where the only comparison operator used is =, is called an EQUIJOIN.

Both previous examples were EQUIJOINs.

Notice that in the result of an EQUIJOIN we always have one or more pairs of attributes that have identical values in every tuple. For example, in Figure 8.6, the values of the attributes Mgr\_ssn and Ssn are identical in every tuple of DEPT\_MGR (the EQUIJOIN result) because the equality join condition specified on these two attributes requires the values to be identical in every tuple in the result.

**Natural Join**

one of each pair of attributes with identical values is superfluous(meaning unnecessary, especially through being more than enough), a new operation called NATURAL JOIN—denoted by \*—was created to get rid of the second (superfluous) attribute in an EQUIJOIN condition.

NATURAL JOIN is basically an EQUIJOIN followed by the removal of the superfluous attributes.

The standard definition of NATURAL JOIN requires that the two join attributes (or each pair of join attributes) have the same name in both relations. If this is not the case, a renaming operation is applied first.

Suppose we want to combine each PROJECT tuple with the DEPARTMENT tuple that controls the project.

In the following example, first we rename the Dnumber attribute of DEPARTMENT to Dnum—so that it has the same name as the Dnum attribute in PROJECT—and then we apply NATURAL JOIN.

PROJ\_DEPT ← PROJECT \* ρ(Dname, Dnum, Mgr\_ssn, Mgr\_start\_date)(DEPARTMENT)

The same query can be done in two steps by creating an intermediate table DEPT

as follows:

DEPT ← ρ(Dname, Dnum, Mgr\_ssn, Mgr\_start\_date)(DEPARTMENT)

PROJ\_DEPT ← PROJECT \* DEPT

The attribute Dnum is called the join attribute for the NATURAL JOIN operation, because it is the only attribute with the same name in both relations.

The resulting relation is illustrated in Figure 8.7(a).

In the PROJ\_DEPT relation, each tuple combines a PROJECT tuple with the DEPARTMENT tuple for the department that controls the project, but only one join attribute value is kept.

If the attributes on which the natural join is specified already have the same names in both relations, renaming is unnecessary.

For example, to apply a natural join on the Dnumber attributes of DEPARTMENT and DEPT\_LOCATIONS, it is sufficient to write

DEPT\_LOCS ← DEPARTMENT \* DEPT\_LOCATIONS

The resulting relation is shown in Figure 8.7(b), which combines each department with its locations and has one tuple for each location.

In general, the join condition for NATURAL JOIN is constructed by equating each pair of join attributes that have the same name in the two relations and combining these conditions with AND.

