**10.1 INFORMAL DESIGN GUIDELINES FOR RELATION SCHEMAS**

We discuss four informal measures of quality for relation schema design in this section:

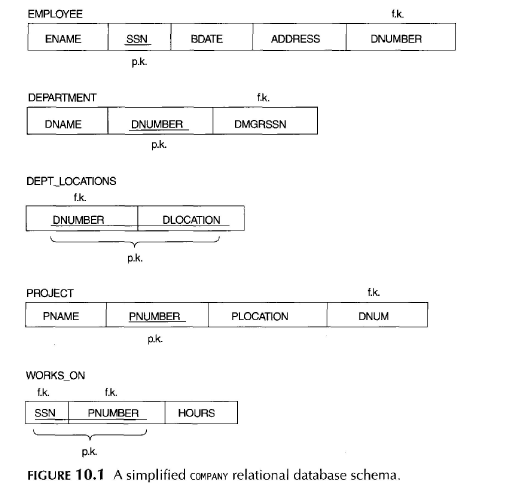
* Semantics of the attributes.
* Reducing the redundant values in tuples.
* Reducing the null values in tuples.
* Disallowing the possibility of generating spurious tuples.

These measures are not always independent of one another, as we shall see.

**10.1.1 Semantics of the Relation Attributes**

Whenever we group attributes to form a relation schema, we assume that attributes belonging to one relation have certain real-world meaning and a proper interpretation associated with them. In Chapter 5 we discussed how each relation can be interpretation as a set of facts or statements. This meaning, or sematics, specifies how to interpret the attribute values stored in a tuple of the relation – in other words, how the attribute values in a tuple relate to one another. If the conceptual design is done carefully, followed by a systematic mapping into relations, most of the semantics will have been accounted for and the resulting design should a clear meaning.

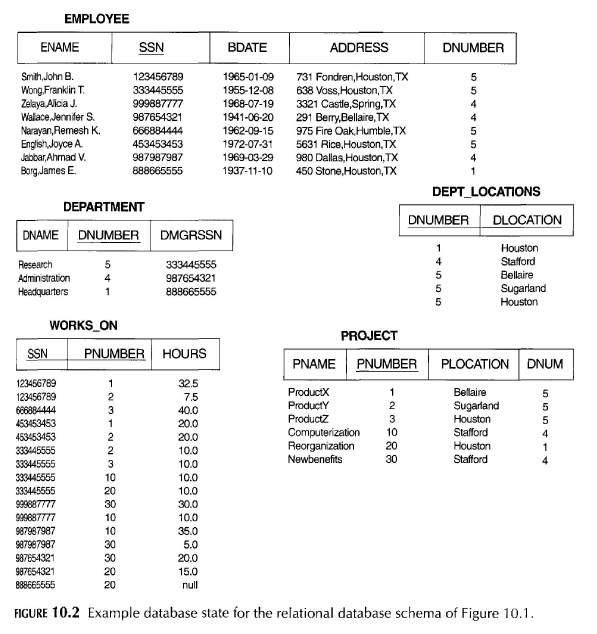
In general, the easier it is to explain the semantics of the relation, the better the relation schema design will be. To illustrate this, consider Figure 10.1, a simplified version of the COMPANY relational database schema of Figure 5.5, and Figure 10.2, which presents an example of populated relation states of this schema. The meaning of the EMPLOYEE relation scheme is quite simple: Each tuple represents an employee, with values for the employee’s name (ENAME), social security number (SSN), birthday (BDATE), and address (ADDRESS), and the number of the department that the employee work for (DNUMBER). The DNUMBER attribute is a foreign key that represents an implicit relationship between: EMPLOYEE and DEPARTMENT. The semantics of the DEPARTMENT and PROJECT schemas are also straightforward: Each DEPARTMENT tuple represents a department entity, and each PROJECT a project entity. The attribute DMGRSSN of DEPARTMENT relates a department to the employee who is its manager, while DNUM of PROJECT relates a project to its controlling department; both are foreign key attributes. The ease with which the meaning of a relation’s attributes can be explain is an informal measure of how well the relation is designed.

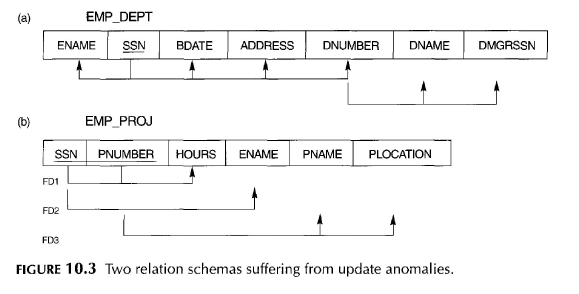


The semantics of the other two relation schemas in Figure 10.1 are slightly more complex. Each tuple in DEPT\_LOCATIONS gives a department number (DNUMBER) and one of the location of the department (DLOCATION). Each tuple in WORK\_ON gives an employee social security number (SSN), the project number of one of the projects that the employee works on (PNUMBER), and the number of hours per week that the employee works on that project (HOURS). However, both schemas have a well-defined and unambiguous interpretation. The schema DEPT\_LOCATIONS represents a multivalued attribute of DEPARTMENT, whereas WORK\_ON represents an M:N relationship between EMPLOYEE and PROJECT. Hence, all the relation schemas in Figure 10.1 may be considered as easy to explain and hence good from ther standpoint of having clear semantics. We can thus formulate the following informal design guideline.

**GUIDELINE 1.** Design a relation schema so that it is easy to explain its meaning. Do not combine attributes from multiple entity types and relationship tuples into a single relation. Intuitively, if a relation schema corresponds to one entity type or one relationship type, it is straigheforward to explain its meaning. Otherwise, if the relation corresponds to a mixture of multiple entities and relationships, semantic ambiguities will result and the relation cannot be easily explained.

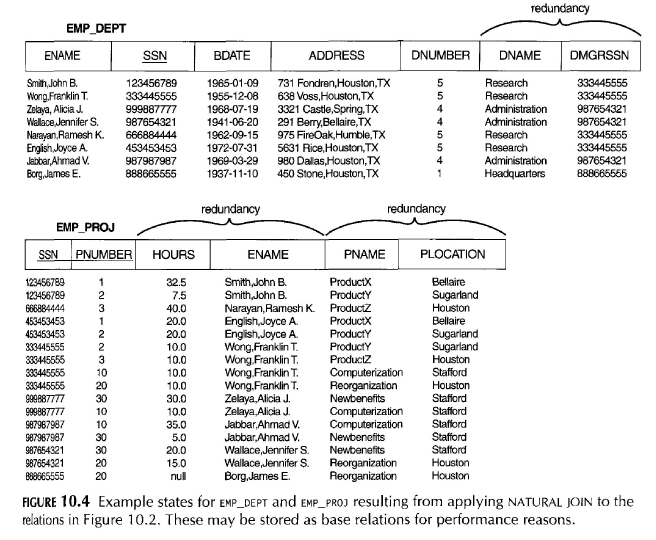
The relation schemas in Figures 10.3a and 10.3b also have clear semantics. (The reader should ignore the lines under relations for now; they are used to illustrate functional dependency notation, discussed in Section 10.2) A tuple in the EMP\_DEPT relation schema of Figure 10.3a represents a single employee bit includes additional information – namely, the name (DNAME) of the department for which the employee works and the social security number (DMGRSSN) of the department manager.For the EMP\_PROJ relation of Figure 10.3b, each tuple ralates an employee to a project but also includes the employee name (ENAME), project name (PNAME), and project location (PLOCATION). Although there it nothing wrong logically with these two relations, they are cosidered poor designs because they violate Guideline 1 by mixing attributes from distinct real-world entities EMP\_DEPT mixes attributes of employees and departments, and EMP\_PROJ mixes attribute of employee and projects. They may be used as view, but they cause problems when used as base relations, as we discuss in the following section.





**10.1.2 Redundant Information in Tuples and Update Anomalies**

One goal of schema desgin is to minimize the storage space used by the base relations (and hence the corressponding files). Grouping attributes into relation schemas has a signficant effect on storage space. For example, compare the space used by the two base relations EMPLOYEE and DEPARTMENT in Figure 10.2 with that for an EMP\_DEPT base relation in Figure 10.4, which is the result of applying the NATURAL JOIN operation to EMPLOYEE and DEPARTMENT, In EMP\_DEPT, the attribute values pertaining to a particular deparment. In contrast, each department’s information appears only once in the DEPARTMENT relation in Figure 10.2. Only the department number (DNUMBER) is repeated in the EMPLOYEE relation for each employee who works in that department. Similar comments apply to the EMP\_PLOJ relation (Figure 10.4), which augments the WORK\_ON relation with additional attributes from EMPLOYEE and PROJECT.



Another serious problem with using the relations in Figure 10.4 as base raltion is the problem of update anomalies. These can be classified into insertion anomalies, deletion anomalies, and modification anomalies.

Insertion Anomalies. Insertion anomalies can be differentiated into two types, illustrated by the following examples based on the EMP\_DEPT relation:

To insert a new employee tuple into EMP\_DEPT, we must include either the attribute values for the department that the employee works for, or nulls (if the employee does not work for a department as yet). For example, to insert a new tuple for an employee who works in department number 5, we must enter the attribute values of department 5 correctly so that they are consitent with values for department 5 in the other tuples in EMP\_DEPT. In the design of Figure 10.2, we do not have worry about this consistency problem because we enter only the department number in the employee tuple, all other attribute values of department 5 are recorded only once in the datebases, as a single tuple in the DEPARTMENT relation.

It is difficult to insert a new department that has no employees as yet in the EMP\_DEPT relation. The only way to do this is to place null values in the attributes for employee. This causes a problem because SSN is the primary key of EMP\_DEPT, and each tuple is supposed to represent an employee entity – not a department entity. Moreover, when the first employee is assigned to department, we do not need this tuple with null value any more. This problem dose not occur in the design of Figure 10.2, because a department is entered in the DEPARTMENT relation whether or not any employee work for it, and whenever an employee is assigned to that department, a corresponding tuple is inserted in EMPLOYEE.

Deletion Anomalies. The problem of detection anomalies is related to the second insertion anomaly situation discussed earlier. If we delete from EMP\_DEPT an employee tuple that happens to represent the last employee working for a particular department, the information concerning that department is lost from the database. This problem does not occur in the database of Figure 10.2 because DEPARTMENT tuples are stored separately.

Modification Anomalies. In EMP\_DEPT, if we change the value of one of the attributes of a particular department – say, the manager of department 5 – we must update the tuples of all employees who work in that department; otherwise, the database will become inconsistent. If we fail to update some tuples, the same department will be shown to have two different values for manager in different employee tuples, which would be wrong.

Based on the preceding three anomalies, we can state the guideline follows.

GUIDELINE 2. Design the base relation schemas so that no insertion, deletion, or modification anomalies are present in the relations. If any anomalies are present, note them clearly and make sure that the programs that update the database will operate correcly:

The second guideline is consistent with and, in a way, a restatement of the first guideline. We can also see the need for a more formal approach to evaluating whether a design meets these guidelines. Sections 10.2 through 10.4 provide these needed formal concepts. It is important to note that these guidelines may sometimes have to be violated in order to improve the performance of certain queries. For example, if an importatn query retrieves informatin concerning the department of an employee along with employee attributes, the EMP\_DEPT schema may be used as a base relation. However, the anomalies in EMP\_DEPT must be noted and accounted for (for example, by using triggers or stored procedures that would make automatic updates) so that, whenever the base relation is updated, we do not end up with inconsistencies. In general, it is advisable to use anomaly free base relations and to specify views that include the joins for placing together the attributes frequently referenced in important queries. This reduces the number of JOIN terms specified in the query, making it simpler to write the query correctly, and in many cases it improves the performance.

**10.1.3 Null values in Tuples**

In some schema designs we may group many attribute together into a “fat” relation. If many of the attributes do not apply to all tuples in the relation, we end up with many nulls in those tuples. This can waste spce ate the storage level and may also lead to problems with understanding the meaning of the atttributes and with specifying JOIN operations at the logical level. Another problem with nulls is how to account for them when aggregate operations such as COUNT or SUM are applied. Moreover, nulls can have multiple interpretations, such as the follwing:

The attribute does not apply to this tuple.

The attribute value for this tuple is unknown.

The value is known but absent; this is, it has not been recorded yet.

Having the same representation for all nulls compromises the different meanings they may have. Therefore, we may state another guideline.

**GUIDELINE 3.** As far as possilble, avoid placing attributes in a base relation whose values may frequently be null. If null are unavoidable, make sure that they apply in exceptional cases only and do not apply to majority of tuples in the relation.

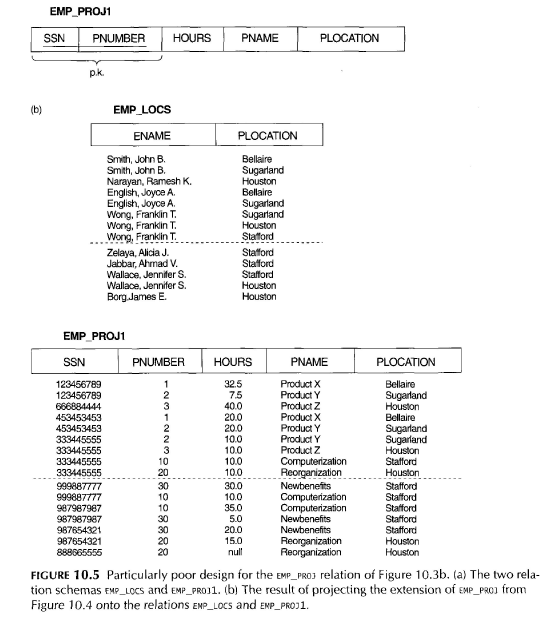
Using space efficiently and avoiding joins are the two overriding criteria that determine whether to include the columns that may have nulls in a relation or to have a separate relation for those columns (with the appropriate key columns). For example, if only 10 percent of employees have individual officers there is litte justification, for including an attribute OFFICE\_NUMBER in the EMPLOYEE relation; rather; a relation EMP\_OFFICES(ESSN, OFFICE\_NUMBER) can be created to include tuples for only the employees with individual offices.

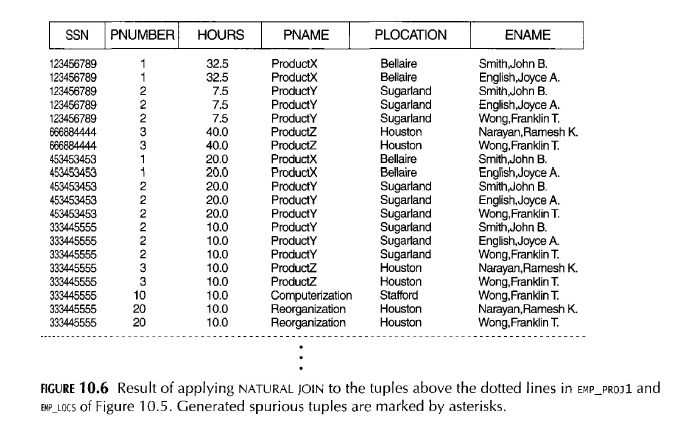
**10.1.4 Generation of Spurious Tuples**

Consider the two relation schemas EMP\_LOCS and EMP\_PROJ1 in Figure 10.5a, which can be used instead of the single EMP\_PROJ relation of Figure 10.3b. A tuple in EMP\_LOCS means that the employee whose name is ENAME works on some project whose location is PLOCATION. A tuple in EMP\_PROJ1 means that employee whose social security number is SSN works HOURS per week on the project whose name, number and location are PNAME, PNUMBER, and PLOCATION. Figure 10.5b shows relation states of EMP\_LOCS and EMP\_PROJ1 corresponding to the EMP\_PROJ relation of Figure 10.4, which are obtained by applying the approprite PROJECT (bi) operations to EMP\_PROJ (ignore the dotted line in Figure 10.5b for now).

Suppose that we used EMP\_PROJ and EMP\_LOCS as the base relations instead of EMP\_PROJ. This produces a particularly bad schema design, because we cannot recover the information that was originally in EMP\_PROJ from EMP\_PROJ1 and EMP\_LOCS. If we attempt a NATURAL JOIN operationon EMP\_PROJ1 and EMP\_LOCS, the result produces many more tuples than the original set of tuples in EMP\_PROJ. In Figure 10.5b is shown (to reduce the size of the resulting relation). Additional tuples that were not in EMP\_PROJ are called spurious tuples because they represent spurious or wrong information that is not valid. The spurious tuples are marked by asterisks in Figure 10.6.

Decomposing EMP\_PROJ into EMP\_LOCS and EMP\_PROJ1 is undesirable because, when we JOIN them back using NATURAL JOIN, we do not get the correct original information. This is because in the case PLOCATION is the attribute that relates EMP\_LOCS and EMP\_PROJ1, and PLOCATION is a primary key nor a foreign key in either EMP\_LOC or EMP\_PROJ1. We can now informally state another design guideline.





**GUIDELINE 4.** Design relation schemas so that they can be joined with equality conditions on attributes that are either primary keys or foreign keys in a way that guarantees that no spurious tuples are genarated. Avoid relations that contain matching attributes that are not (foreign key, primary key) combinations, because joining on such attributes may produce spurious stuples.

This informal guide obviously needs to be stated more formally. In Chapter 11 we discuss a formal condition, called the nonadditive (or losseless) join property, that guarantees that certain joins do not produce spurious tuples.

10.1.5 Summary and Discussion of Design Guidelines.

In section 10.1.1.1 through 10.1.4, we informally discussed situations that lead to problematic relation schemas, and we proposed informal guidelines for a good relational design. The problems we pointed out, which can be detected without additional tools of analysis, are as follows:

Anomalies that cause redundant work to be down during insertion into and modification of a relation, an thay may cause accidental loss of information during a deletion from a relation.

Waste of storage space due to nulls and the difficulty of performing aggregation opearation and joins due to null values.

Generation of invalid and spurios data during joins on improperly related base relations.

In the rest of this chapter we present formal concepts and thery that may be used to define the “goodness” and “badness” of individual relation schemas more precisely. We first discuss functional dependency as a tool for analysis. Then we specify the three normal additional normal froms and Boyce-Codd normal form (BCNF) for ration schemas. In Chaper 11, we define additional normail form that which are based on additional types of data dependencis called multivalued dependencees and jion dependencies.

**10.2 FUNCTIONAL DEPENDENCIES.**

The simple most important concepts in relation schema design theory is that of functional, dependency). It this section we formallly define the concept, and in Section 10.3 we see how it can be used to define normal forms normal for relation IT.

**10.2.1 Defition of Functional,**

A functional dependency is a constraint bettween two sets of attribute from the database. Suppose that our relational databases. Database schema has n attribule A1, A2, A3, A4, A1, …A­n, let us think of the whole database as being by a single universal relation schema R = {Ai}