CHAPTER 14
Databases

Objectives

After studying this chapter, the student should be able to:

- Define a database and a database management system (DBMS) and describe the components of a DBMS.
- Describe the architecture of a DBMS based on the ANSI/SPARC definition.
- Define the three traditional database models: hierarchical, networking, and relational.
- Describe the relational model and relations.
- Understand operations on a relational database based on commands available in SQL.
- Describe the steps in database design.
- Define ERM and E-R diagrams and explain the entities and relationships in this model.
- Define the hierarchical levels of normalization and understand the rationale for normalizing the relations.
- List database types other than the relational model.

14-1 INTRODUCTION

Data storage traditionally used individual, unrelated files, sometimes called flat files. In the past, each application program in an organization used its own file. In a university, for example, each department might have its own set of files: the record office kept a file about the student information and their grades, the scheduling office kept the name of the professors and the courses they were teaching, the payroll department kept its own file about the whole staff and so on. Today, however, all of these flat files can be combined in a single entity, the database for the whole university.

Although it is difficult to give a universally agreed definition of a database, we use the following common definition:

A database is a collection of related, logically coherent, data used by the application programs in an organization.

Comparing the flat-file system, we can mention several advantages for a database system.

Less redundancy

In a flat-file system there is a lot of redundancy. For example, in the flat file system for a university, the names of professors and students are stored in more than one file.

Inconsistency avoidance

If the same piece of information is stored in more than one place, then any changes in the data need to occur in all places that data is stored.

Efficiency

A database is usually more efficient that a flat file system, because a piece of information is stored in fewer locations.

Data integrity

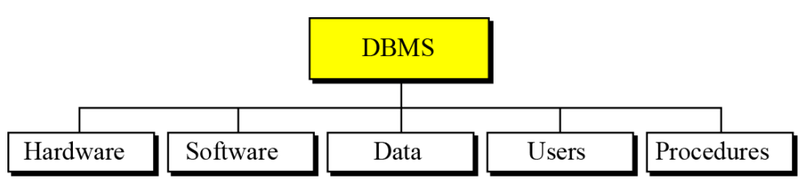
In a database system it is easier to maintain data integrity (see Chapter 16) because a piece of data is stored in fewer locations.

Confidentiality

It is easier to maintain the confidentiality of the information if the storage of data is centralized in one location.

A database management system (DBMS) defines, creates, and maintains a database. The DBMS also allows controlled access to data in the database. A DBMS is a combination of five components: hardware, software, data, users, and procedures (Figure 14.1).

Figure 14.1 DBMS components



Hardware

The hardware is the physical computer system that allows access to data.

Software

The software is the actual program that allows users to access, maintain, and update data. In addition, the software controls which user can access which parts of the data in the database.

Confidentiality

The data in a database is stored physically on the storage devices. In a database, data is a separate entity from the software that accesses it.

Users

The term users in a DBMS has a broad meaning. We can divide users into two categories: end users and application programs.

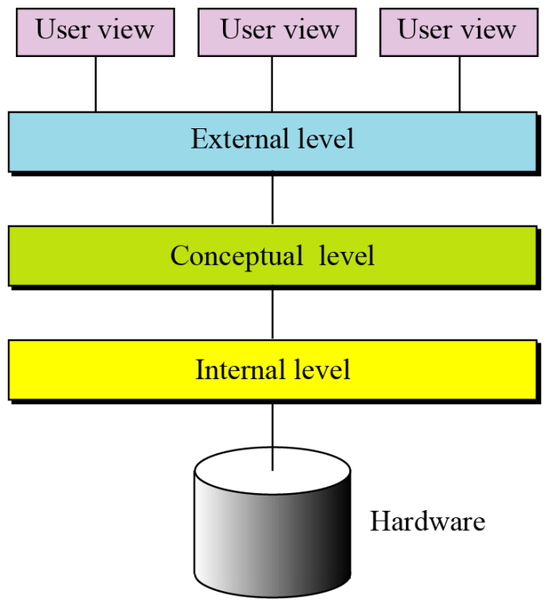
Procedures

The last component of a DBMS is a set of procedures or rules that should be clearly defined and followed by the users of the database.

14-3 DATABASE ARCHITECTURE

The American National Standards Institute/Standards Planning and Requirements Committee (ANSI/SPARC) has established a three-level architecture for a DBMS: internal, conceptual, and external (Figure 14.2).

Figure 14.2 Database architecture



14.2.1 Internal level

The internal level determines where data is actually stored on the storage devices. This level deals with low-level access methods and how bytes are transferred to and from storage devices. In other words, the internal level interacts directly with the hardware.

14.2.2 Conceptual level

The conceptual level defines the logical view of the data. The data model is defined on this level, and the main functions of the DBMS, such as queries, are also on this level. The DBMS changes the internal view of data to the external view that users need to see. The conceptual level is an intermediary and frees users from dealing with the internal level.

14.2.3 External level

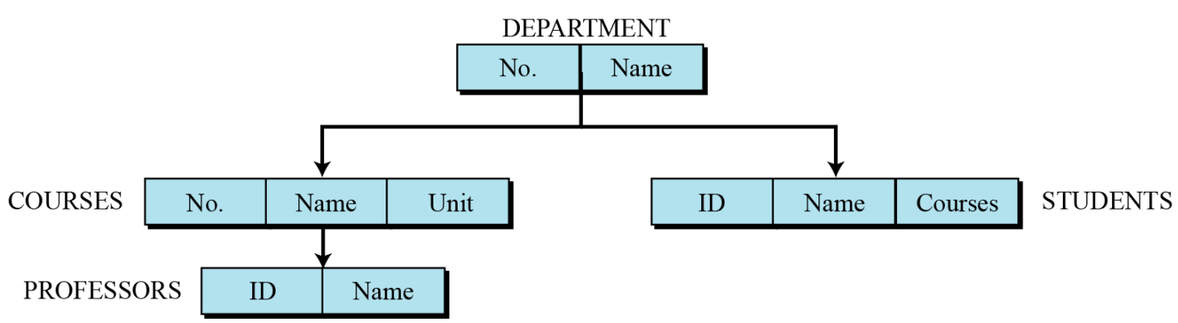
The external level interacts directly with the user (end users or application programs). It changes the data coming from the conceptual level to a format and view that is familiar to the users.

14-4 DATABASE MODELS

A database model defines the logical design of data. The model also describes the relationships between different parts of the data. In the history of database design, three models have been in use: the hierarchical model, the network model, and the relational model.

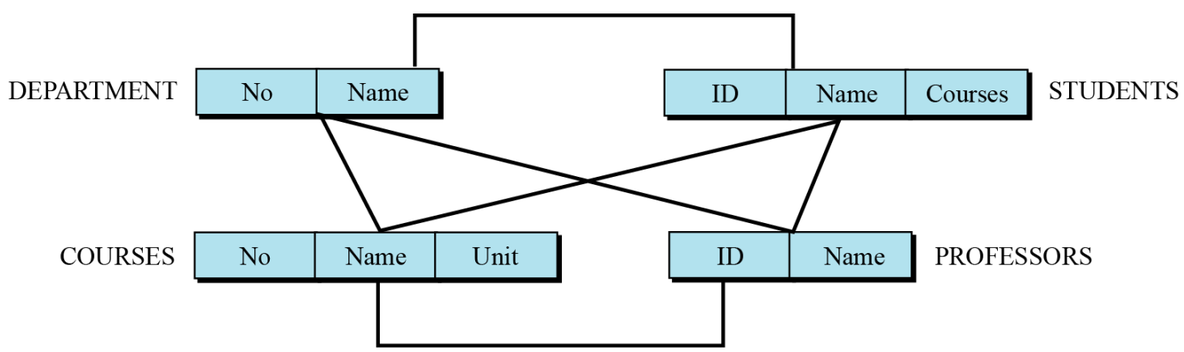
In the hierarchical model, data is organized as an inverted tree. Each entity has only one parent but can have several children. At the top of the hierarchy, there is one entity, which is called the root.

Figure 14.3 An example of the hierarchical model



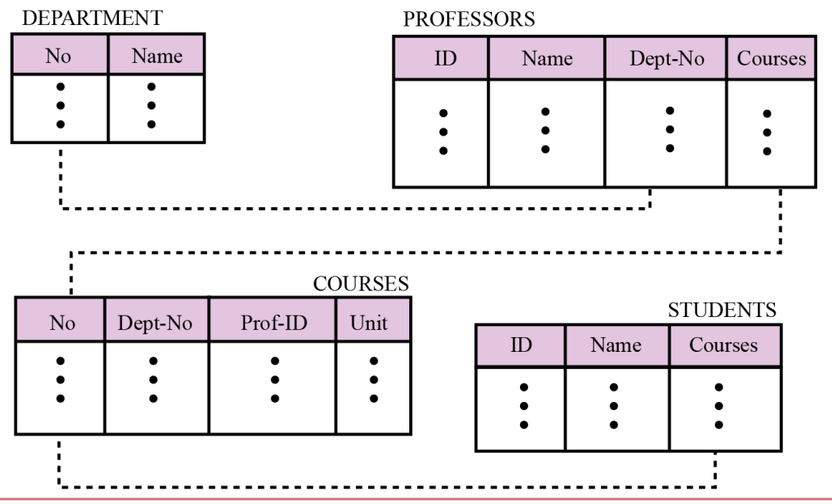
In the network model, the entities are organized in a graph, in which some entities can be accessed through several paths (Figure 14.4).

Figure 14.4 An example of the network model representing a university



In the relational model, data is organized in two-dimensional tables called relations. The tables or relations are, however, related to each other, as we will see shortly.

Figure 14.5 An example of the relational model representing a university

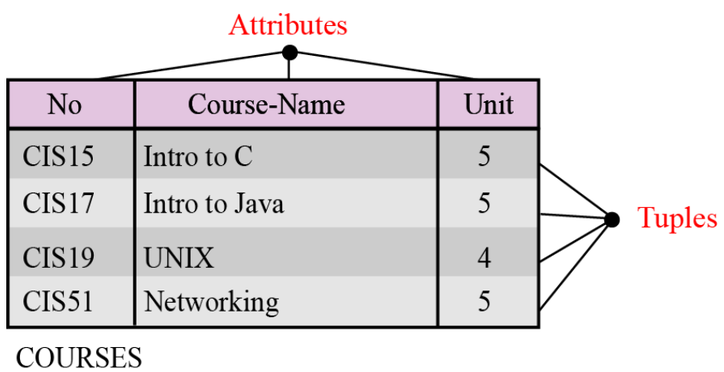


14.4 THE RELATIONAL DATABASE MODEL

In the relational database management system (RDBMS), the data is represented as a set of relations.

A relation, in appearance, is a two-dimensional table. The RDBMS organizes the data so that its external view is a set of relations or tables. This does not mean that data is stored as tables: the physical storage of the data is independent of the way in which the data is logically organized.

Figure 14.6 An example of a relation



A relation in an RDBMS has the following features:

Name. Each relation in a relational database should have a name that is unique among other relations.

Attributes. Each column in a relation is called an attribute. The attributes are the column headings in the table in Figure 14.6.

Tuples. Each row in a relation is called a tuple. A tuple defines a collection of attribute values. The total number of rows in a relation is called the cardinality of the relation. Note that the cardinality of a relation changes when tuples are added or deleted. This makes the database dynamic.

In a relational database we can define several operations to create new relations based on existing ones. We define nine operations in this section: insert, delete, update, select, project, join, union, intersection, and difference. Instead of discussing these operations in the abstract, we describe each operation as defined in the database query language SQL (Structured Query Language).

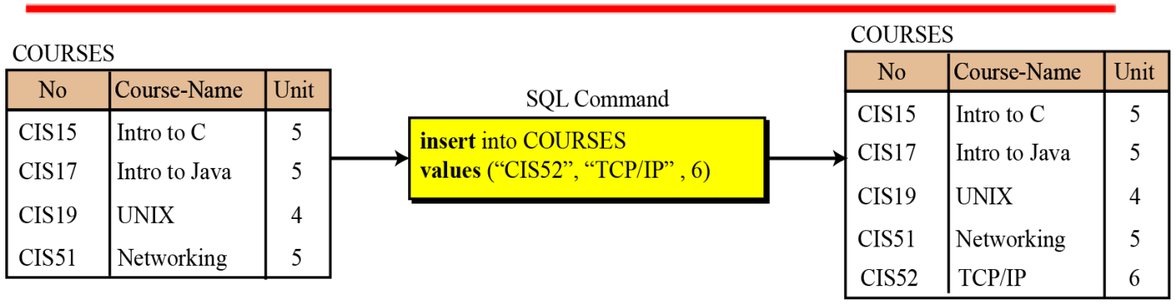
Structured Query Language (SQL) is the language standardized by the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) for use on relational databases. It is a *declarative* rather than *procedural* language, which means that users declare what they want without having to write a step-by-step procedure. The SQL language was first implemented by the Oracle Corporation in 1979, with various versions of SQL being released since then.

Insert

The *insert operation* is a unary operation—that is, it is applied to a single relation. The operation inserts a new tuple into the relation. The insert operation uses the following format:

```
insert into    RELATION-NAME
values    (…, …, …)
```

Figure 14.7 An example of an insert operation



Delete

The delete operation is also a unary operation. The operation deletes a tuple defined by a criterion from the relation. The delete operation uses the following format:

```
delete from RELATION-NAME
where criteria
```

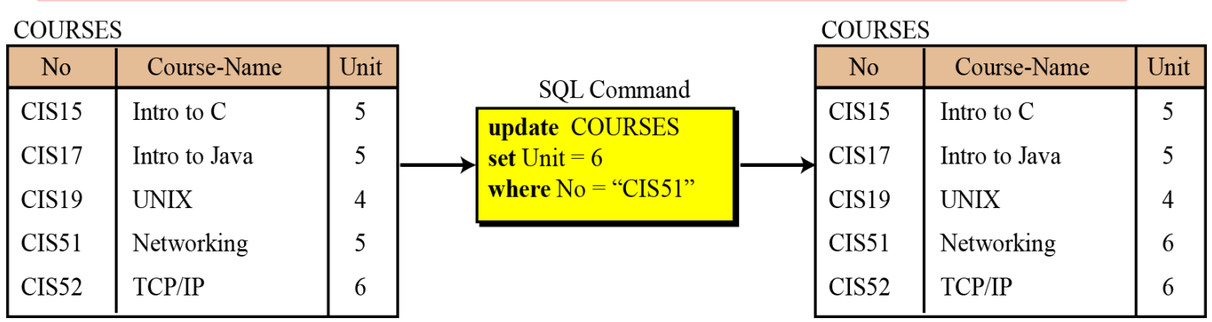
Figure 14.8 An example of a delete operation

Update

The update operation is also a unary operation that is applied to a single relation. The operation changes the value of some attributes of a tuple. The update operation uses the following format:

```
update   RELATION-NAME
set attribute1 = value1,   attribute2 = value2, ...
where    criteria
```

Figure 14.9 An example of an update operation

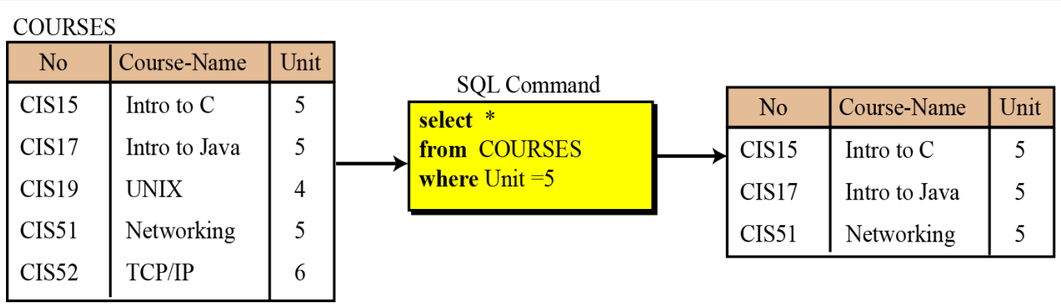


Select

The select operation is a unary operation. The tuples (rows) in the resulting relation are a subset of the tuples in the original relation.

```
select    *
from RELATION-NAME
where    criteria
```

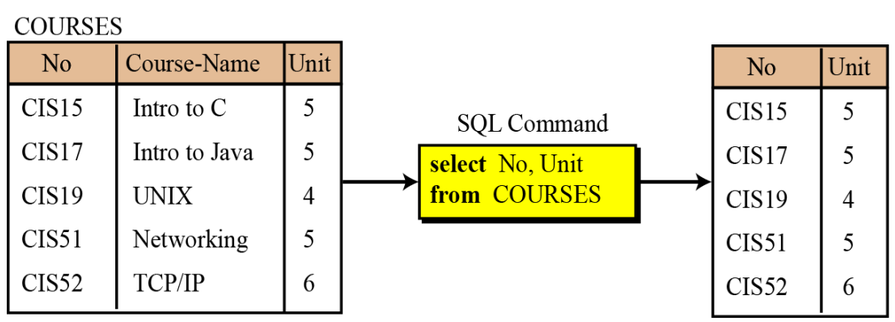
Figure 14.10 An example of a select operation



The project operation is also a unary operation, and creates another relation. The attributes (columns) in the resulting relation are a subset of the attributes in the original relation.

```
select    attribute-list
from RELATION-NAME
```

Figure 14.11 An example of a project operation

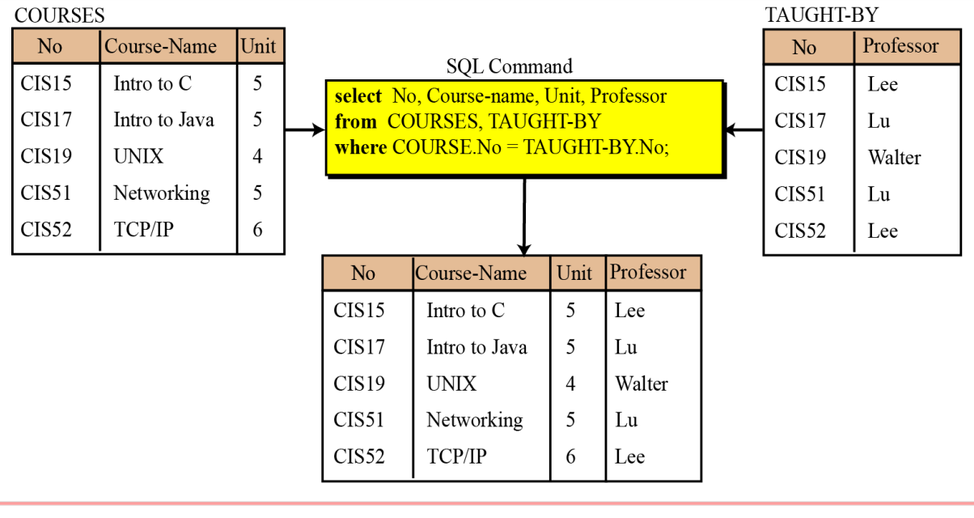


Join

The join operation is a binary operation that combines two relations on common attributes.

```
select    attribute-list
from RELATION1, RELATION2
where criteria
```

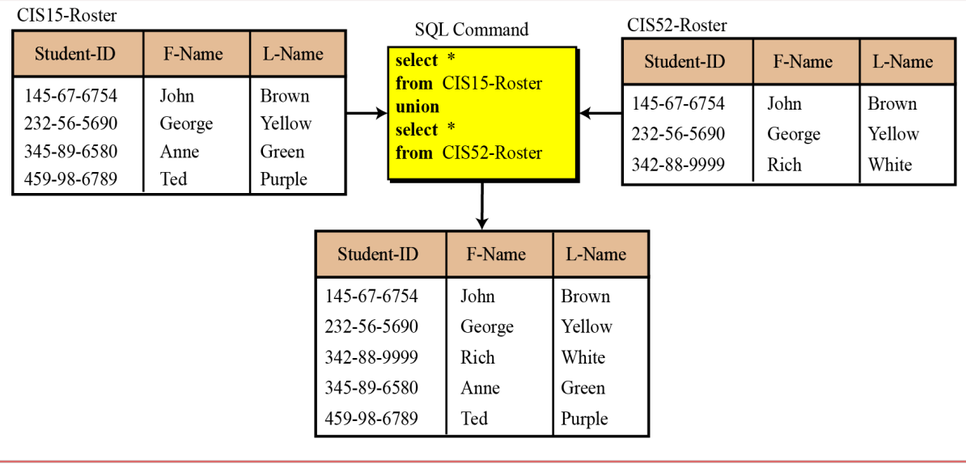
Figure 14.12 An example of a join operation



The union operation takes two relations with the same set of attributes.

```
select *
from RELATION1
union
select *
from RELATION2
```

Figure 14.13 An example of a union operation

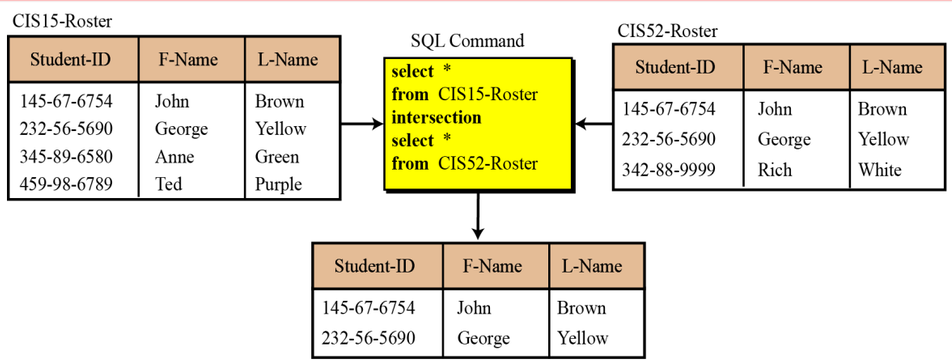


Intersection

The intersection operation takes two relations and creates a new relation, which is the intersection of the two.

```
select *
from RELATION1
intersection
select *
from RELATION2
```

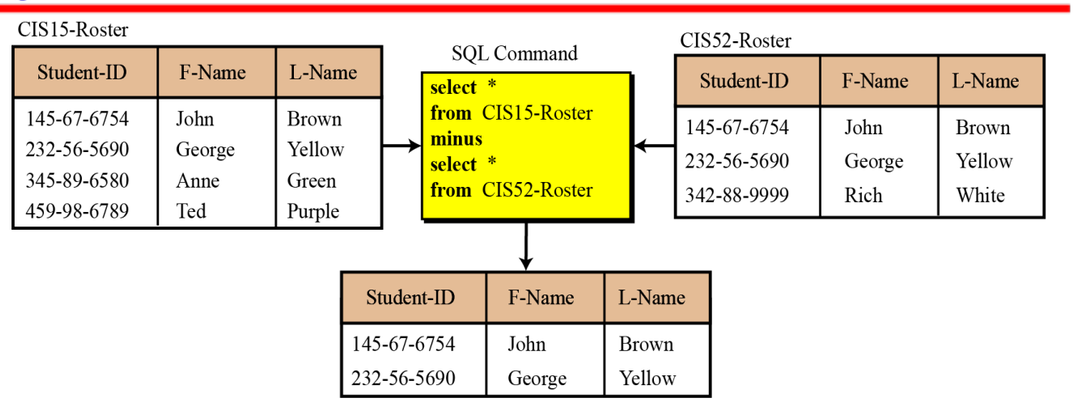
Figure 14.14 An example of an intersection operation



The difference operation is applied to two relations with the same attributes. The tuples in the resulting relation are those that are in the first relation but not the second.

```
select *
from RELATION1
minus
select *
from RELATION2
```

Figure 14.15 An example of a difference operation



14-5 DATABASE DESIGN

The design of any database is a lengthy and involved task that can only be done through a step-by-step process. The first step normally involves interviewing potential users of the database. The second step is to build an entity-relationship model (ERM) that defines the entities, the attributes of those entities, and the relationship between those entities.

In this step, the database designer creates an entity-relationship (E-R) diagram to show the entities for which information needs to be stored and the relationship between those entities. E-R diagrams uses several geometric shapes, but we use only a few of them here:

Rectangles represent entity sets.

Ellipses represent attributes.

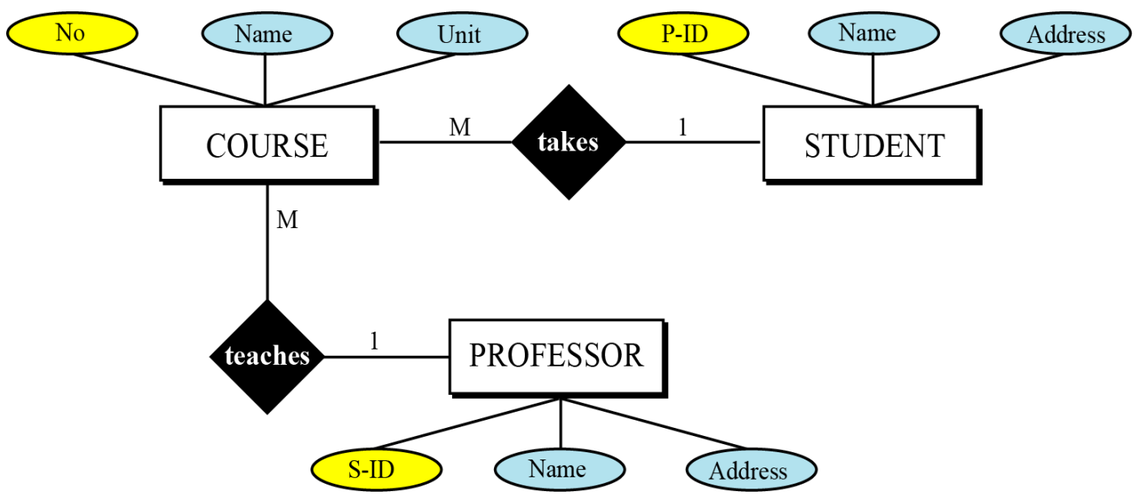
Diamonds represent relationship sets.

Lines link attributes to entity sets and link entity sets to relationships sets.

Example 14.1

Figure 14.16 shows a very simple E-R diagram with three entity sets, their attributes, and the relationship between the entity sets.

Figure 14.16 Entities, attributes, and relationships in an E-R diagram



After the E-R diagram has been finalized, relations (tables) in the relational database can be created.

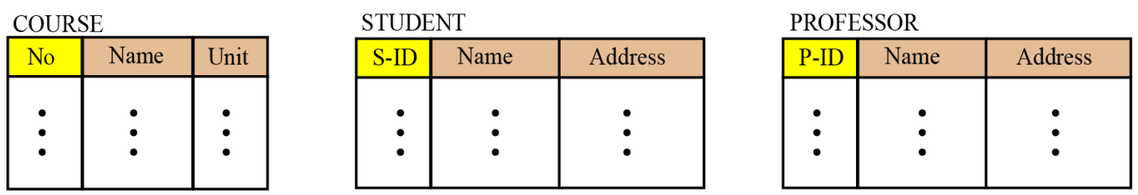
Relations for entity sets

For each entity set in the E-R diagram, we create a relation (table) in which there are *n* columns related to the *n* attributes defined for that set.

Example

We can have three relations (tables), one for each entity set defined in Figure 14.16, as shown in Figure 14.17.

Figure 14.17 Relations for entity set in Figure 14.16



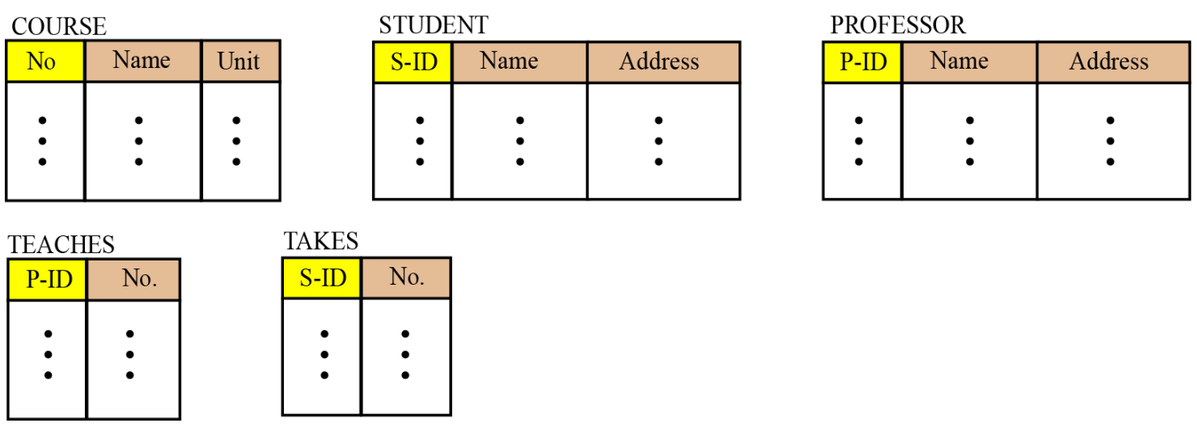
Relations for relationship sets

For each relationship set in the E-R diagram, we create a relation (table). This relation has one column for the key of each entity set involved in this relationship and also one column for each attribute of the relationship itself if the relationship has attributes (not in our case).

Example 14.3

There are two relationship sets in Figure 14.16, teaches and takes, each connected to two entity sets. The relations for these relationship sets are added to the previous relations for the entity set and shown in Figure 14.18.

Figure 14.18 Relations for E-R diagram in Figure 14.16



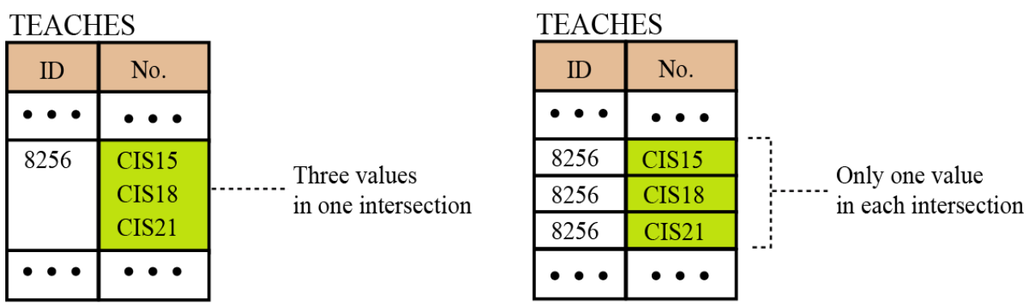
Normalization is the process by which a given set of relations are transformed to a new set of relations with a more solid structure. Normalization is needed to allow any relation in the database to be represented, to allow a languages like SQL to use powerful retrieval operations composed of atomic operations, to remove anomalies in insertion, deletion, and updating, and reduce the need for restructuring the database as new data type are added.

The normalization process defines a set of hierarchical normal forms (NFs). Several normal forms have been proposed, including 1NF, 2NF, 3NF, BCNF (Boyce-Codd Normal Form), 4NF, PJNF (Projection/Joint Normal Form), 5NF, and so on. structure.

First normal form (1NF)

When we transform entities or relationships into tabular relations, there may be some relations in which there are more values in the intersection of a row or column.

Figure 14.19 An example of 1NF



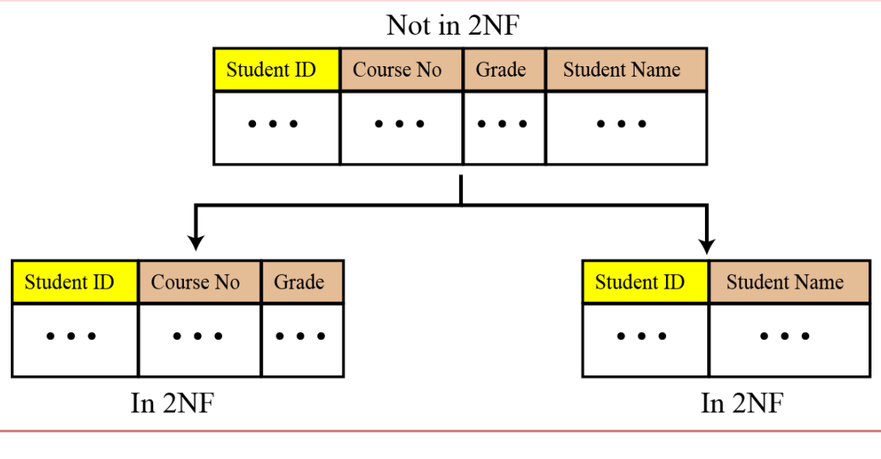
a. Not in 1NF

b. In 1NF

Second normal form (2NF)

In each relation we need to have a key (called a primary key) on which all other attributes (column values) needs to depend. For example, if the ID of a student is given, it should be possible to find the student's name.

Figure 14.20 An example of 2NF



Other normal forms

Other normal forms use more complicated dependencies among attributes. We leave these dependencies to books dedicated to the discussion of database topics.

14-6 OTHER DATABASE MODELS

The relational database is not the only database model in use today. Two other common models are distributed databases and object-oriented databases. We briefly discuss these here.

The distributed database model is not a new model, but is based on the relational model. However, the data is stored on several computers that communicate through the Internet or a private wide area network. Each computer (or site) maintains either part of the database or the whole database.

Fragmented distributed databases

In a fragmented distributed database, data is localized—locally used data is stored at the corresponding site. However, this does not mean that a site cannot access data stored at another site, but access is mostly local, but occasionally global.

Replicated distributed databases

In a replicated distributed database, each site holds an exact replica of another site. Any modification to data stored in one site is repeated exactly at every site. The reason for having such a database is security. If the system at one site fails, users at the site can access data at another site.

An object-oriented database tries to keep the advantages of the relational model and at the same time allows applications to access structured data. In an object-oriented database, objects and their relations are defined. In addition, each object can have attributes that can be expressed as fields.

XML

The query language normally used for objected-oriented databases is XML (Extensible Markup Language). As we discussed in Chapter 6, XML was originally designed to add markup information to text documents, but it has also found its application as a query language in databases. XML can represent data with nested structure.