UNIT – I

UNIT – I: Introduction: Database System Applications, Purpose of Database Systems, View of Data, Database Languages, Relational Databases, Database Design, Object-based and Semi-structured Databases, Data storage and Querying, Transaction Management, Mining and Analysis, Database Architecture, Database Users and Administrators.

Database Design and the E-R Model: Overview of the Design Process, The Entity-Relationship Model, Constraints, Entity-Relationship Diagrams, Entity – Relationship Design Issues, Weak Entity Sets, Extended E-R Features, Database Design for Banking Enterprise, Reduction to Relational Schemas, Other Aspects of Database Design

INTRODUCTION

A database management system (DBMS) is a collection of interrelated and a set of application programs used to access, update and manage that data. DBMS is also simply called as database system (DBS). This interrelated data is usually referred to as the database (DB).

- ❖ The goal of DBMS is to provide an environment that is both convenient and efficient to use in Retrieving information from the database, Storing information into the database.

Databases are usually designed to manage large bodies of information. This involves definition of structures for information storage (data modeling), provision of mechanisms for the manipulation of information (systems structure, query processing), providing for the safety of information in the database (crash recovery and security) and concurrency control if the system is shared by users.

DATABASE SYSTEM APPLICATIONS

- ✓ Banking: All Transactions
- ✓ Universities: Registration, Grades
- ✓ Sales: Customers, Products, Purchases
- ✓ Online Retailers: Order Tracking, Customized Recommendations
- ✓ Manufacturing: Production, Inventory, Orders, Supply Chain
- ✓ Airlines: Reservations, Schedules
- ✓ Human Resources: Employee Records, Salaries, Tax Deductions

- ❖ Databases touch all aspects of our lives
- ❖

PURPOSE OF DATABASE SYSTEMS

The typical file processing system is supported by a conventional operating system. The system stores permanent records in various files, and it needs different application programs to extract records from, and add records to, the appropriate files.

A file processing system has a number of major disadvantages.

1. **Data Redundancy and Inconsistency:** In file processing, every user group maintains its own files for handling its data processing applications. For example, consider the UNIVERSITY database. Here, two groups of users might be the course registration personnel and the

accounting office. The accounting office also keeps data on registration and related billing information, whereas the registration office keeps track of student courses and grades. Storing the same data multiple times is called data redundancy. This redundancy leads to several problems:

- ✓ Need to perform a single logical update multiple times.
- ✓ Storage space is wasted.
- ✓ Files that represent the same data may become inconsistent.
- ✓ Data inconsistency is the various copies of the same data may no larger agree.
  Example: One user group may enter a student's birth date erroneously as JAN-19-1984, whereas the other user groups may enter the correct value of JAN-29-1984.

2. **Difficulty in Accessing Data:** File processing environments do not allow needed data to be retrieved in a convenient and efficient manner. Suppose that one of the bank officers needs to find out the names of all customers who live within a particular area. The bank officer has now two choices: either obtain the list of all customers and extract the needed information manually or ask a system programmer to write the necessary application program. Both alternatives are obviously unsatisfactory. Suppose that such a program is written, and that, several days later, the same officer needs to trim that list to include only those customers who have an account balance of $10,000 or more. A program to generate such a list does not exist. Again, the officer has the preceding two options, neither of which is satisfactory.
3. **Data Isolation:** Because data are scattered in various files, and files may be in different formats, writing new application programs to retrieve the appropriate data is difficult.
4. **Integrity Problems:** The data values stored in the database must satisfy certain types of consistency constraints. Example: The balance of certain types of bank accounts may never fall below a prescribed amount. Developers enforce these constraints in the system by addition appropriate code in the various application programs
5. **Atomicity Problems:** Atomic means the transaction must happen in its entirety or not at all. It is difficult to ensure atomicity in a conventional file processing system. Example: Consider a program to transfer $50 from account A to account B. If a system failure occurs during the execution of the program, it is possible that the $50 was removed from account A but was not credited to account B, resulting in an inconsistent database state.
6. **Concurrent Access Anomalies:** For the sake of overall performance of the system and faster response, many systems allow multiple users to update the data simultaneously. In such an environment, interaction of concurrent updates is possible and may result in inconsistent data. To guard against this possibility, the system must maintain some form of supervision. But supervision is difficult to provide because data may be accessed by many different application programs that have not been coordinated previously. Example: When several reservation clerks try to assign a seat on an airline flight, the system should ensure that each seat can be accessed by only one clerk at a time for assignment to a passenger.
7. **Security Problems:** Enforcing security constraints to the file processing system is difficult.

VIEW OF DATA

Data Abstraction: A major purpose of a database system is to provide users with an abstract view of the data. That is, the system hides certain details of how the data are stored and maintained. For the system to be usable, it must retrieve data efficiently. There are several levels of abstraction:

1. **Physical Level:** The lowest level of abstraction describes ***how*** the data are actually stored. The physical level describes complex low-level data structures in detail. Ex: index, B-tree, hashing.
2. **Logical Level:** The next-higher level of abstraction describes ***what*** data are stored in the database, and what relationships exist among those data. The logical level thus describes the entire database in terms of a small number of relatively simple structures. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.
3. **View Level:** Highest level of abstraction describes part of entire database for a particular group of users. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. View level of abstraction exists to simplify their interaction with the system. *The system may provide many views for same database.*

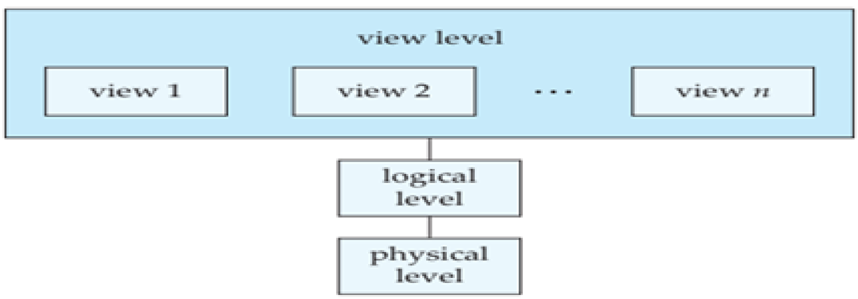


Figure The three levels of data abstraction.

Instances and Schemas: Similar to types and variables in programming languages.

Schema: The logical structure of the database (or) the overall design of the database is called the database schema. Schemas are changed infrequently, if at all. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. Types of Schema include:

Physical Schema: describes the database design at the physical level

Logical Schema: describes the database design at the logical level.

A database may also have several schemas at the view level, sometimes called ***subschemas*** that describe different views of the database.

- The logical schema is the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema.

Instance: Collection of information stored in the database at a particular moment is called an instance of the database. Values of variables in a program at a point in time relate to an instance of database schema.

Physical Data Independence: The ability to modify the physical schema without changing the logical schema is known as Physical data independence. Usually, applications depend on the logical schema.

Logical Data Independence: Ability to modify logical schema without changing physical schema. It is harder to achieve as application programs are usually heavily dependent on logical structure of data.

DATA MODELS

Underlying the structure of a database is the data model: "a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints". A data model provides a way to describe the design of a database at the physical, logical, and view levels.

The data models can be classified into four different categories:

1. **Relational Model:** The relational model uses a collection of tables to represent both data and the relationships among those data. Each table has multiple columns, and each column has a unique name. Tables are also known as ***relations***. The relational model is an example of a record-based model. Record-based models are so named because the database is structured in fixed-format records of several types. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. *The collection of attributes and records will create the table*. The relational data model is the most widely used data model, and a vast majority of current database systems are based on the relational model.
2. **Entity-Relationship Model:** The entity-relationship (E-R) data model uses a collection of basic objects, called ***entities***, and relationships among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects. The entity-relationship model is widely used in database design.
3. **Object-Based Data Model:** Object-oriented programming (especially in Java, C++, or C#) has become the dominant software-development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model. The object-relational data model combines features of the object-oriented data model and relational data model.
4. **Semi structured Data Model:** The semi structured data model permits the specification of data where individual data items of the same type may have different sets of attributes. The Extensible Markup Language (XML) is widely used to represent semi structured data.

- Historically, ***network data model*** and ***hierarchical data model*** preceded relational data model. These models were tied closely to the underlying implementation, and complicated task of modeling data. As a result they are used little now, except in old database code that is still in service in some places.

DATABASE LANGUAGES

A database system provides a ***Data-Definition Language*** to specify the database schema and a ***Data Manipulation Language*** to express database queries and updates. In practice, the data-definition and data manipulation languages are not two separate languages; instead they simply form parts of a single database language, such as the widely used SQL language.

Data-Definition Language: "We specify a database schema by a set of definitions expressed by a special language called a data-definition language (DDL)". DDL is also used to specify additional properties of data.

The DDL just like any other programming language gets some instructions (statements) as input and generates some output. ***The output of the DDL is placed in the data dictionary, which contains metadata—that is, data about data.*** The data dictionary is considered to be a special type of table that can only be accessed and updated by the database system itself (not a regular user). The database system consults the data dictionary before reading or modifying actual data.

Example: Create Table account (account_number char (10), balance integer)

The data values stored in the database must satisfy certain consistency constraints. For example, suppose the university requires that the account balance of a department must never be negative. DDL provides facilities to specify such constraints. Database systems implement ***integrity constraints*** that can be tested:

- ***Domain Constraints:*** A domain of possible values must be associated with every attribute (for example, integer types, character types, date/time types). Declaring an attribute to be of a particular domain acts as a constraint on the values that it can take. These are the most elementary form of integrity constraint.
- ***Referential Integrity:***. There are cases where we wish to ensure that a value that appears in one relation for a given set of attributes also appears in a certain set of attributes in another relation (referential integrity). For example, the dept_name value in a course record must appear in the dept_name attribute of some record of the department relation. Database modifications can cause violations of referential integrity. When a referential integrity constraint is violated, normal procedure is to reject the action that caused the violation.
- ***Assertions:*** An assertion is *any condition that the database must always satisfy*. Domain constraints and referential integrity constraints are special forms of assertions. However, there are many constraints that we cannot express by using only these special forms. For example, "Every department must have at least five courses offered every semester" must be expressed as an assertion.
- ***Authorization:*** We may want to differentiate among the users as far as the type of access they are permitted on various data values in the database. These differentiations are expressed in terms of authorization. (The most common authorization is: *read authorization*, which allows reading, but not modification of data; *insert authorization*, which allows insertion of new data, but not modification of existing data; *update authorization*, which allows modification, but not deletion, of data; and *delete authorization,* which allows deletion of data. We may assign the user all, none, or a combination of these types of authorization).

❖ We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a *data storage and definition language*. These statements define the implementation details of the database schemas, which are usually hidden from the users.

Data-Manipulation Language: A data-manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model. The types of access are:

- ✓ Retrieval of information stored in the database (Select)
- ✓ Insertion of new information into the database (Insert)
- ✓ Deletion of information from the database (Delete)
- ✓ Modification of information stored in the database (Update)

There are basically two types of DMLs:

1. ***Procedural DMLs*** require a user to specify what data are needed and how to get those data.
2. ***Declarative DMLs*** (also referred to as nonprocedural DMLs) require a user to specify what data are needed without specifying how to get those data.

Declarative DMLs are usually easier to learn and use than are procedural DMLs.

A query is a statement requesting the retrieval of information. The portion of a DML that involves information retrieval is called a query language. Although technically incorrect, it is common practice to use the terms query language and data-manipulation language synonymously. SQL is the most widely used query language. Levels of abstraction apply both to defining data, manipulating data.

RELATIONAL DATABASES

A relational database is based on the relational model and uses a collection of tables to represent both data and the relationships among those data. It also includes DML and DDL.

Tables: Each table has multiple columns and each column has a unique name. Each table contains records of a particular type. Each record type defines a fixed number of fields, or attributes. The columns of the table correspond to the attributes of the record type. It is also possible to create schemas in the relational model that have problems such as unnecessarily duplicated information. Figure below presents a sample relational database table of university instructors.

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

The *instructor* table

The instructor table shows, for example, that an instructor named Einstein with ID 22222 is a member of the Physics department and has an annual salary of $95,000. ***(Draw and Explain any table of your wish)***

Data-Definition Language: SQL provides a rich DDL that allows one to define tables, integrity constraints, assertions, etc. For instance, the following SQL DDL statement defines the department table:

Create Table department (dept_name char (20), building char (15), budget numeric (12,2));

Execution of the above DDL statement creates the department table with three columns: dept_name, building, and budget, each of which has a specific data type associated with it.

Data-Manipulation Language: The SQL language is nonprocedural. A query takes several tables as input (possibly only one) and always returns a single table. Here is an example of an SQL query that finds the names of all instructors in the History department:

```
Select name
From instructor
Where dept_name = ‘History’;
```

The query specifies that those rows from the table instructor where the dept_name is History must be retrieved, and the name attribute of these rows must be displayed. If the query is run on table in previous figure, result will consist of two rows, one with the name El Said and the other with the name Califieri.

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	85000
Music	Packard	80000
Finance	Painter	120000
History	Painter	50000
Physics	Watson	70000

The *department* table

Queries may involve information from more than one table. For instance, following query finds ID and department name of all instructors associated with a department with budget of greater than $95,000.

```
Select instructor.ID, department.dept_name
From instructor, department
Where instructor.dept_name= department.dept_name and department.budget > 95000;
```

If above query were run on the tables in above figure, the system would find that there are two departments with budget of greater than $95,000—Computer Science and Finance; there are five instructors in these departments. Thus, the result will consist of a table with two columns (ID, dept_name) and five rows: (12121, Finance), (45565, Computer Science), (10101, Computer Science), (83821, Computer Science), and (76543, Finance).

Database Access from Application Programs: SQL does not support actions such as input from users, output to displays, or communication over the network. Such computations and actions must be written in a host language, such as C, C++, or Java, with embedded SQL queries that access the data in the database. There are two ways to do this:

- By providing an application program interface (set of procedures) that can be used to send DML and DDL statements to the database and retrieve the results. The Open Database Connectivity (ODBC) standard for use with the C language is a commonly used application program interface standard. The Java Database Connectivity (JDBC) standard provides corresponding features to the Java language.
- By extending the host language syntax to embed DML calls within the host language program.

DATABASE DESIGN

Database systems are designed to manage large bodies of information. Database design mainly involves the design of the database schema. The design of a complete database application environment that meets the needs of the enterprise being modeled requires attention to a broader set of issues.

Design Process: A high-level data model provides the database designer with a conceptual framework in which to specify the data requirements of the database users, and how the database will be structured to fulfill these requirements.

- ✓ The initial phase of database design is to characterize fully the data needs of database users. The outcome of this phase is a specification of user requirements.
- ✓ Next, the designer chooses a data model, and by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database.
- ✓ The schema developed at this conceptual-design phase provides a detailed overview of the enterprise. The designer reviews the schema to confirm that all data requirements are indeed satisfied and are not in conflict with one another.

  The designer can also examine the design to remove any redundant features. The focus at this point is on describing the data and their relationships, rather than on specifying physical storage details.

- ❖ In terms of the relational model, the conceptual-design process involves decisions on what attributes we want to capture in the database and how to group these attributes to form the various tables. The "what" part is basically a business decision and the "how" part is mainly a computer-science problem. There are principally two ways to tackle the problem.
- ❖ First one is to use ***entity-relationship model***; the other is to employ a set of algorithms collectively known as ***normalization*** that takes as input the set of all attributes and generates a set of tables.
- ❖ The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases. In the **logical-design phase**, the designer maps the high-level conceptual schema onto the implementation data model of the database system that will be used. The designer uses the resulting system-specific database schema in the subsequent **physical-design phase**, in which the physical features of the database are specified.

Database Design for a University Organization: The description that arises from this design phase serves as the basis for specifying the conceptual structure of the database. Here are the major characteristics of the university.

- ✓ University is organized into departments. Each department is identified by a unique name (dept_name).
- ✓ Each department has a list of courses it offers. Each course has associated with it a course_id, title, dept_name, and credits, and may also have have associated prerequisites.
- ✓ Instructors are identified by their unique ID. Each instructor has name, associated department (dept_name), and salary.
- ✓ Students are identified by their unique ID. Each student has a name, an associated major department (dept_name), and tot_cred (total credit hours the student earned thus far).
- ✓ The university maintains a list of classrooms, specifying the name of the building, room number, and room capacity.
- ✓ The university maintains a list of all classes (sections) taught. Each section is identified by a course_id, sec_id, year, and semester, and has associated with it a semester, year, building, room number, and time_slot_id.
- ✓ The university has a list of all student course registrations, specifying, for each student, the courses and the associated sections that the student has taken (registered for).

DATABASE DESIGN FOR UNIVERSITY

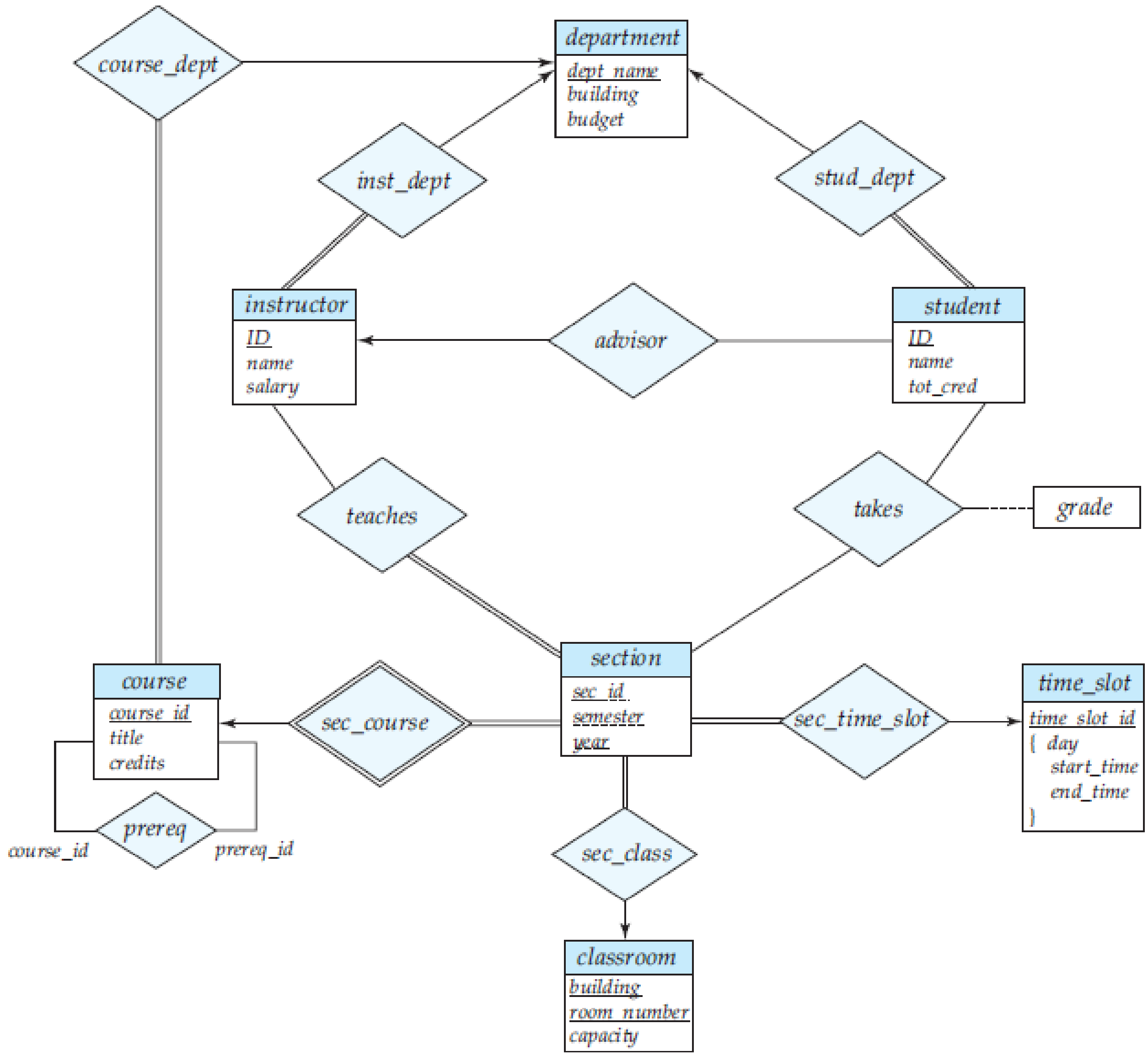


Figure E-R diagram for a university enterprise.

The Entity-Relationship Model: The entity-relationship (E-R) data model uses a collection of basic objects, called ***entities***, and ***relationships*** among these objects. An entity is a "thing" or "object" in the real world that is distinguishable from other objects. For example, each person is an entity, and bank accounts can be considered as entities. Entities are described in a database by a set of attributes. For example, the attributes ID, name, and salary may describe an instructor entity.

The extra attribute ID is used to identify an instructor uniquely (since it may be possible to have two instructors with the same name and the same salary). In the United States, many organizations use the social-security number of a person (a unique number the U.S. government assigns to every person in the United States) as a unique identifier.

- A relationship is an association among several entities. For example, a member relationship associates an instructor with her department. The set of all entities of the same type and the set of all relationships of the same type are termed an ***entity se***t and ***relationship set***, respectively.

The overall logical structure (schema) of a database can be expressed graphically by an entity-relationship (E-R) diagram. There are several ways in which to draw these diagrams. One of the most popular is to use the Unified Modeling Language (UML). In the notation we use, which is based on UML, an E-R diagram is represented as follows:

- ***Entity sets*** are represented by a partitioned rectangular box with the entity set name in the header and the attributes listed below it.
- ***Relationship sets*** are represented by a diamond connecting a pair of related entity sets. The name of the relationship is placed inside the diamond.

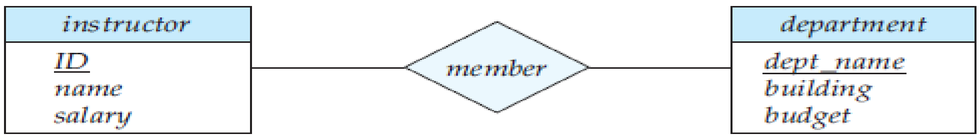


Fig: E-R Diagram

In addition to entities and relationships, the E-R model represents certain constraints to which the contents of a database must conform. One important constraint is mapping cardinalities, which express the number of entities to which another entity can be associated via a relationship set. For example, if each instructor must be associated with only a single department, the E-R model can express that constraint.

The entity-relationship model is widely used in database design.

Normalization: Another method for designing a relational database is to use normalization. To understand the need for normalization, let us look at what can go wrong in a bad database design. Among the undesirable properties that a bad design may have are:

- Repetition of information
- Inability to represent certain information

The goal of Normalization is to generate a set of relation schemas that allows us to store information without unnecessary redundancy, yet also allows us to retrieve information easily. The approach is to design schemas that are in an appropriate normal form. To determine whether a relation schema is in one of the desirable normal forms, we need additional information about the real-world enterprise that we are modeling with the database. The most common approach is to use functional dependencies.

DATABASE ARCHITECTURE

The architecture of a database system is greatly influenced by the underlying computer system on which the database system runs. Database systems can be centralized, or client-server, where one server machine executes work on behalf of multiple client machines. Database systems can also be designed to exploit parallel computer architectures. Distributed databases span multiple geographically separated machines.

Most users of a database system today are not present at the site of the database system, but connect to it through a network. We can therefore differentiate between client machines, on which remote database users work, and server machines, on which the database system runs. The figure given provides a single picture of the various components of a database system and the connections among them.

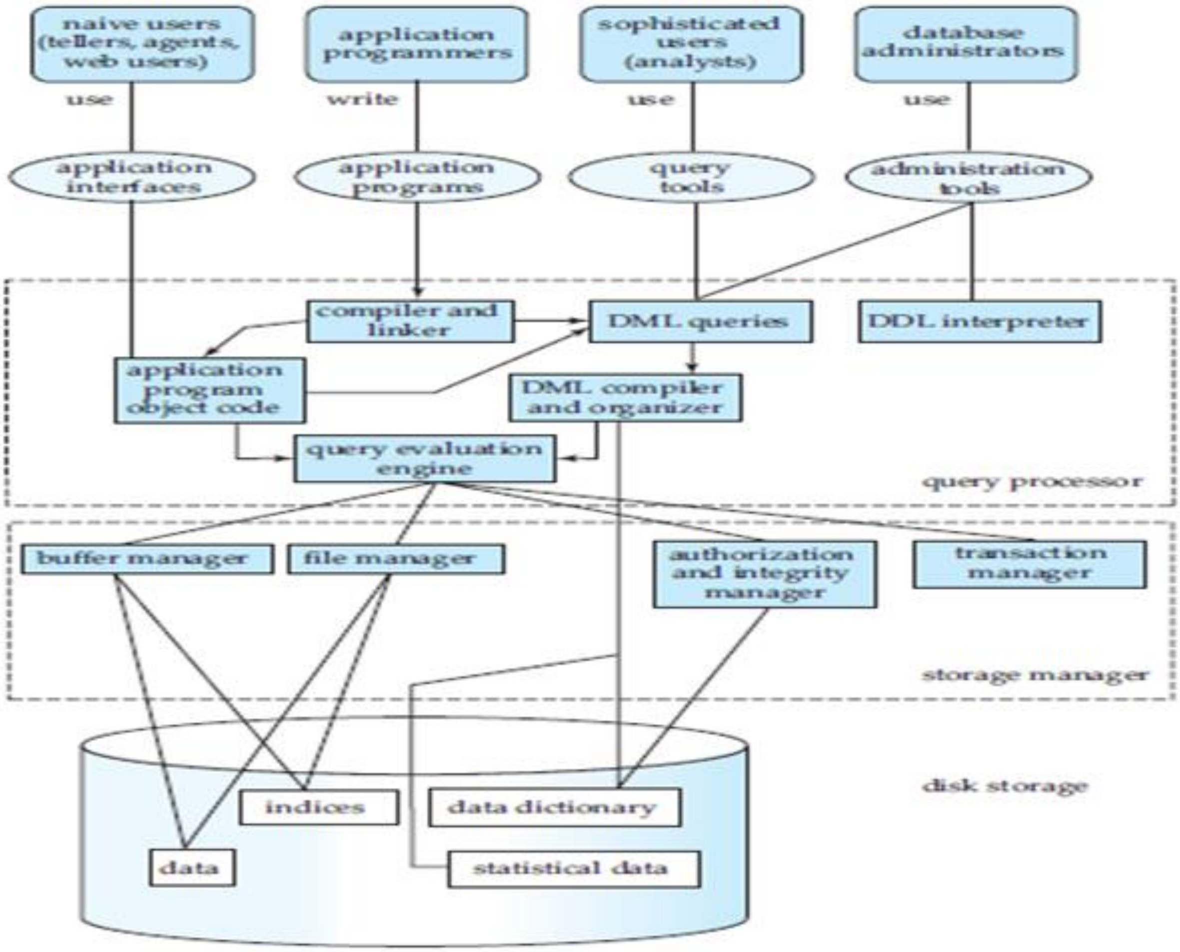


Figure System structure.

Database applications are usually partitioned into two or three parts. In two-tier architecture, the application resides at the client machine, where it invokes database system functionality at the server machine through query language statements. Application program interface standards like ODBC and JDBC are used for interaction between the client and the server.

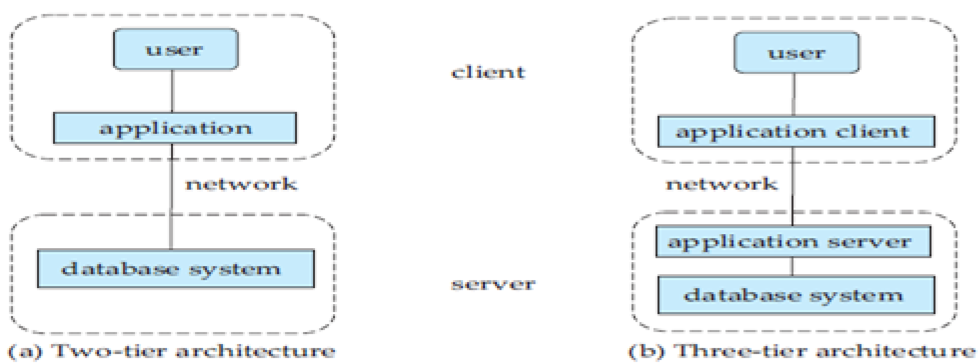


Figure Two-tier and three-tier architectures.

In contrast, in three-tier architecture, client end communicates with an application server, usually through a forms interface. The application server in turn communicates with a database system to access data. The business logic of the application, which says what actions to carry out under what conditions, is embedded in the application server. Three-tier applications are more appropriate for large applications, and for applications that run on the WWW.

DATABASE USERS AND ADMINISTRATORS

Primary goal of a database system is to retrieve information from and store new information into the database. People who work with a database can be categorized as database users or database administrators.

Database Users and User Interfaces: There are four different types of database-system users, differentiated by the way they expect to interact with the system. Different types of user interfaces have been designed for the different types of users.

1) **Naïve Users:** Naïve users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. The typical user interface for naïve users is a forms interface, where the user can fill in appropriate fields of the form. Naïve users may also simply read reports generated from the database.

As an example, consider a student, who during class registration period, wishes to register for a class by using a Web interface. Such a user connects to a Web application program that runs at a Web server. The application first verifies the identity of the user, and allows her to access a form where she enters the desired information. The form information is sent back to the Web application at the server, which then determines if there is room in the class and if so adds the student information to the class roster in the database.

2) **Application Programmers:** Application programmers are computer professionals who write application programs. Application programmers can choose from many tools to develop user interfaces. Rapid application development (RAD) tools are tools that enable an application programmer to construct forms and reports with minimal programming effort.
3) **Sophisticated Users:** Sophisticated users interact with the system without writing programs. Instead, they form their requests either using a database query language or by using tools such as data analysis software. Analysts who submit queries to explore data in the database fall in this category.
4) **Specialized Users:** Specialized users are sophisticated users who write specialized database applications that do not fit into the traditional data-processing framework. Among these applications are computer-aided design systems, systems that store data with complex data types (for example, graphics data and audio data), and environment-modeling systems.

Database Administrator: One of the main reasons for using DBMS is to have central control of both the data and the programs that access those data. ***A person who has such central control over the system is called a database administrator (DBA)***. The **functions/responsibilities** of a DBA include:

1. **Schema Definition:** The DBA creates the original database schema by executing a set of data definition statements in the DDL.
2. **Storage Structure and Access-Method Definition**
3. **Schema and Physical-Organization Modification:** The DBA carries out changes to the schema and physical organization to reflect the changing needs of the organization, or to alter the physical organization to improve performance.

4. **Granting of Authorization for Data Access:** By granting different types of authorization, the database administrator can regulate which parts of the database various users can access. The authorization information is kept in a special system structure that the database system consults whenever someone attempts to access the data in the system.
5. **Routine Maintenance:** Examples of the database administrator's routine maintenance activities are:
   - ✓ Periodically backing up the database, either onto tapes or onto remote servers, to prevent loss of data in case of disasters such as flooding.
   - ✓ Ensuring that enough free disk space is available for normal operations, and upgrading disk space as required.
   - ✓ Monitoring jobs running on the database and ensuring that performance is not degraded by very expensive tasks submitted by some users.

DATA STORAGE AND QUERYING

A database system is partitioned into modules that deal with each of the responsibilities of the overall system. The functional components of a database system can be broadly divided into the storage manager and the query processor components.

Storage Manager: Storage manager is the component of a database system that provides interface between low-level data stored in database, application programs and queries submitted to the system.

The storage manager is important because databases typically require a large amount of storage space. Corporate databases range in size from hundreds of gigabytes to, for the largest databases, terabytes of data. Since the main memory of computers cannot store this much information, the information is stored on disks. Data are moved between disk storage and main memory as needed.

- ❖ *The storage manager is responsible for the interaction with the file manager*. The raw data are stored on the disk using the file system provided by the operating system. The storage manager is responsible for storing, retrieving, and updating data in the database.

The storage manager components include:

1. **Authorization and Integrity Manager**, which tests for the satisfaction of integrity constraints and checks the authority of users to access data.
2. **Transaction Manager**, which ensures that the database remains in a consistent (correct) state despite system failures, and that concurrent transaction executions proceed without conflicting.
3. **File Manager**, which manages the allocation of space on disk storage and the data structures used to represent information stored on disk.
4. **Buffer Manager**, which is responsible for fetching data from disk storage into main memory, and deciding what data to cache in main memory. Buffer manager is a critical part of the database system, since it enables the database to handle data sizes that are much larger than the size of main memory.

The storage manager implements ***several data structures*** as part of the physical system implementation:

- ✓ Data files, which store the database itself.
- ✓ Data dictionary, which stores metadata about structure of database, in particular schema of database.
- ✓ Indices, which can provide fast access to data items. Like the index in a textbook, a database index provides pointers to those data items that hold a particular value.

The Query Processor: The query processor is important because it helps the database system to simplify and facilitate access to data. It allows database users to obtain good performance while being able to work at the view level. It is the job of the database system to translate updates and queries written in a nonprocedural language, at the logical level, into an efficient sequence of operations at the physical level. The query processor components include:

1. **DDL interpreter**, which interprets DDL statements and records the definitions in the data dictionary.
2. **DML compiler**, which translates DML statements in a query language into an evaluation plan consisting of low-level instructions that the query evaluation engine understands. The DML compiler also performs *query optimization*; that is, it picks lowest cost evaluation plan among the alternatives.
3. **Query evaluation engine**, which executes low-level instructions generated by the DML compiler.

SPECIALITY (OBJECT-BASED AND SEMI-STRUCTURED) DATABASES

Object-Based Data Models: Object-oriented programming has become the dominant software development methodology. This led to the development of an object-oriented data model that can be seen as extending the E-R model with notions of encapsulation, methods (functions), and object identity. Inheritance, and encapsulation (information hiding), with methods to provide an interface to objects, are among the key concepts of object-oriented programming that have found applications in data modeling. The object-oriented data model also supports a rich type system, including structured and collection types. In the 1980s, several database systems based on the object-oriented data model were developed. The major database vendors presently support the object-relational data model, a data model that combines features of the object-oriented data model and relational data model.

Semi Structured Data Models: Semi structured data models permit the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast with data models mentioned earlier, where every data item of a particular type must have the same set of attributes.

The XML language was initially designed as a way of adding markup information to text documents, but has become important because of its applications in data exchange. XML provides a way to represent data that have nested structure, and furthermore allows a great deal of flexibility in structuring of data, which is important for certain kinds of nontraditional data.

TRANSACTION MANAGEMENT

Often, several operations on the database form a single logical unit of work. An example is a funds transfer, in which one department account (say A) is debited and another department account (say B) is credited. Clearly, it is essential that either both the credit and debit occur, or that neither occur. That is, the funds transfer must happen in its entirety or not at all. This all-or-none requirement is called ***atomicity***. In addition, it is essential that the execution of the funds transfer preserve the consistency of the database. That is, the value of the sum of the balances of A and B must be preserved. This correctness requirement is called ***consistency***. Finally, after the successful execution of a funds transfer, the new values of the balances of accounts A and B must persist, despite the possibility of system failure. This persistence requirement is called ***durability***.

“A transaction is a collection of operations that performs a single logical function in a database application”. Each transaction is a unit of both atomicity and consistency. Thus, we require that transactions do not violate any database-consistency constraints. That is, if the database was consistent when a transaction started, the database must be consistent when the transaction successfully terminates. It is the responsibility of *programmer*.

Ensuring atomicity and durability properties is the responsibility of *recovery manager*. In the absence of failures, all transactions complete successfully, and atomicity is achieved easily. However, because of various types of failure, a transaction may not always complete its execution successfully. If we are to ensure the atomicity property, a failed transaction must have no effect on the state of the database. Thus, database must be restored to the state in which it was before the transaction in question started executing. The database system must therefore perform failure recovery, that is, detect system failures and restore the database to the state that existed prior to the occurrence of the failure.

Finally, when several transactions update the database concurrently, the consistency of data may no longer be preserved, even though each individual transaction is correct. It is the responsibility of the *concurrency-control manager* to control interaction among concurrent transactions, to ensure consistency of database. ***Transaction manager consists of concurrency-control manager and recovery manager***.

MINING AND ANALYSIS

The term data mining refers to the process of semi automatically analyzing large databases to find useful patterns. Data mining deals with large volumes of data, stored primarily on disk. That is, data mining deals with “knowledge discovery in databases.” There are a variety of possible types of patterns that may be useful, and different techniques are used to find different types of patterns.

Usually there is a manual component to data mining, consisting of preprocessing data to a form acceptable to algorithms, post processing of discovered patterns to find novel ones that could be useful. There may also be more than one type of pattern that can be discovered from a given database, manual interaction may be needed to pick useful types of patterns. For this reason, data mining is really a semiautomatic process in real life.

Several techniques and tools are available to help with decision support. Several tools for data analysis allow analysts to view data in different ways. Other analysis tools pre compute summaries of very large amounts of data, to give fast responses to queries. The SQL standard contains additional constructs to support data analysis.

Large companies have diverse sources of data that they need to use for making business decisions. To execute queries efficiently on such diverse data, companies have built data warehouses. Data warehouses gather data from multiple sources under a unified schema, at a single site. Thus, they provide the user a single uniform interface to data.

Textual data is unstructured, unlike the rigidly structured data in relational databases. Querying of unstructured textual data is referred to as ***information retrieval***. Information retrieval systems have much in common with database systems—in particular, the storage and retrieval of data on secondary storage.

DATABASE DESIGN AND THE E-R MODEL

Overview of the Design Process: The task of creating a database application is a complex one, involving design of the database schema, design of the programs that access and update the data, and design of a security scheme to control access to data. ***Needs of users*** play a central role in design process.

Design Phases: For small applications, it may be feasible for a database designer who understands the application requirements to decide directly on the relations to be created, their attributes, and constraints on the relations. A high-level and complex data model provides database designer with a conceptual framework to specify data requirements of database users, and how database will be structured to fulfill these requirements.

- ✓ The initial phase of database design is to characterize fully the data needs of database users. The outcome of this phase is a specification of user requirements.
- ✓ Next, the designer chooses E-R data model, and by applying the concepts of the chosen data model, translates these requirements into a conceptual schema of the database.
- ✓ The schema developed at this conceptual-design phase provides a detailed overview of the enterprise. The entity-relationship model is typically used to represent the conceptual design. By specifying the entities, the attributes of the entities, the relationships among the entities, and constraints on the entities and relationships.

Typically, the conceptual-design phase results in the creation of an entity-relationship diagram that provides a graphic representation of the schema.

- ▪ The designer reviews the schema to confirm that all data requirements are indeed satisfied and are not in conflict with one another. Designer can also examine the design to remove any redundant features. The focus at this point is on describing the data and their relationships, rather than on specifying physical storage details.

The process of moving from an abstract data model to the implementation of the database proceeds in two final design phases.

- ❖ In the ***logical-design phase***, the designer maps the high-level conceptual schema onto the implementation data model of the database system that will be used.
- ❖ Finally, the designer uses the resulting system-specific database schema in the subsequent ***physical-design phase***, in which the physical features of the database are specified.

<u>Note:</u> The physical schema of a database can be changed relatively easily after an application has been built. However, changes to the logical schema are usually harder to carry out, since they may affect a number of queries and updates scattered across application code. It is therefore important to carry out the database design phase with care, before building the rest of the database application.

Design Alternatives: A major part of the database design process is deciding how to represent in the design the various types of "things" such as people, places, products, and the like. We use the term entity to refer to any such distinctly identifiable item. In a university database, examples of entities would include instructors, students, and departments. Various entities are related to each other in a variety of ways, all of which need to be captured in database design.

In designing a database schema, we must avoid two major pitfalls:

1. **Redundancy:** A bad design may repeat information. Redundancy can also occur in a relational schema. The biggest problem with redundant information is that the copies of a piece of information can become inconsistent if the information is updated without taking precautions to update all copies of the information. Ideally, information should appear in exactly one place.
2. **Incompleteness:** A bad design may make certain aspects of enterprise difficult/impossible to model. For example, suppose that, we only had entities corresponding to course offering, without having an entity corresponding to courses. Equivalently, in terms of relations, suppose we have a single relation where we repeat all of the course information once for each section that the course is offered. It would then be impossible to represent information about a new course, unless that course is offered.

THE ENTITY-RELATIONSHIP MODEL

The entity-relationship (E-R) data model was developed to facilitate database design by allowing specification of an enterprise schema that represents the overall logical structure of a database. The E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema. Because of this usefulness, many database-design tools draw on concepts from the E-R model. The E-R data model employs three basic concepts: entity sets, relationship sets, and attributes.

Entity Sets: An ***entity*** is a “thing” or “object” in the real world that is distinguishable from all other objects. For example, each person in a university is an entity. An entity has a set of properties, and the values for some set of properties may uniquely identify an entity. For instance, a person may have a person id property whose value uniquely identifies that person. An entity may be concrete, such as a person or a book, or it may be abstract, such as a course, a course offering, or a flight reservation.

An ***entity set*** is a set of entities of the same type that share the same properties, or attributes. The set of all people who are instructors at a given university, for example, can be defined as the entity set instructor. We use the term extension of the entity set to refer to the actual collection of entities belonging to the entity set. Thus, the set of actual instructors in the university forms the extension of the entity set instructor. The above distinction is similar to the difference between a relation and a relation instance.

An entity is represented by a set of ***attributes***. Attributes are descriptive properties possessed by each member of an entity set. Possible attributes of instructor entity set are ID, name, dept_name, and salary. Each entity has a ***value*** for each of its attributes. For instance, a particular instructor entity may have value 12121 for ID, the value Wu for name, the value Finance for dept_name, and the value 90000 for salary.

76766	Crick
45565	Katz
10101	Srinivasan
98345	Kim
76543	Singh
22222	Einstein

instructor

Fig: Instructor Entity Set

Relationship Sets: A ***relationship*** is an association among several entities. A ***relationship set*** is a set of relationships of same type. Formally, it is a mathematical relation on $n \geq 2$ (possibly non-distinct) entity sets.

If E1, E2,..., En are entity sets, then a relationship set R is a subset of

$\{(e1,e2,...,en) \mid e1 \in E1, e2 \in E2,...,en \in En\}$ where (e1,e2,...,en) is a relationship.

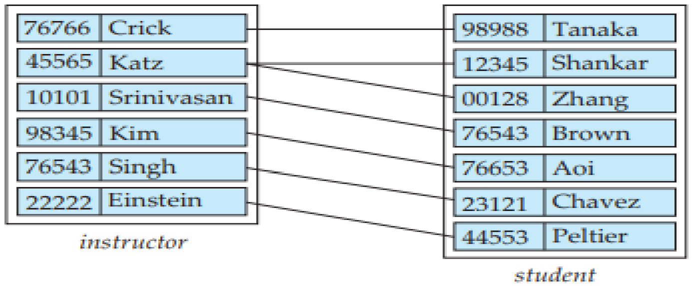


Figure Relationship set *advisor*.

Consider the two entity sets instructor and student in Figure. We define the relationship set advisor to denote the association between instructors and students. The association between entity sets is referred to as ***participation***; that is, the entity sets E1, E2,..., En participate in relationship set R. A relationship instance in an E-R schema represents an association between the named entities.

Ex: The individual instructor entity Katz, who has instructor ID 45565, and the student entity Shankar, who has student ID 12345, participate in a relationship instance of advisor. This relationship instance represents that in the university, the instructor Katz is advising student Shankar.

The function that an entity plays in a relationship is called that entity's ***role***. Since entity sets participating in a relationship set are generally distinct, roles are implicit and are not usually specified. However, they are useful when the meaning of a relationship needs clarification.

A relationship may also have attributes called ***descriptive attributes***. Consider a relationship set advisor with entity sets instructor and student. We could associate the attribute date with that relationship to specify the date when an instructor became the advisor of a student. Below figure shows the relationship set advisor with a descriptive attribute date.

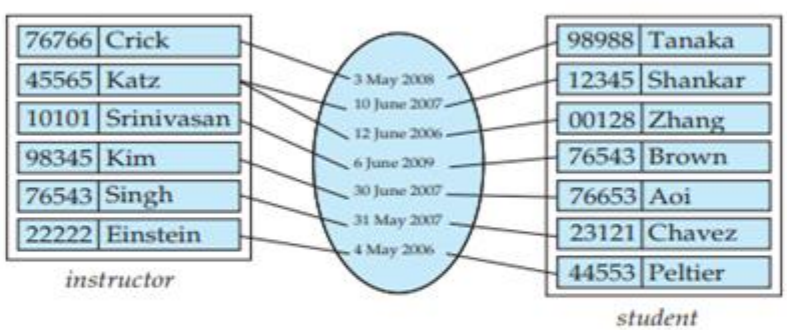


Figure *date* as attribute of the *advisor* relationship set.

The relationship set advisor provides an example of a binary relationship set— that is, one that involves two entity sets. Most of the relationship sets in a database system are binary. Occasionally, however, relationship sets involve more than two entity sets, often called as ternary relationships or non-binary or n-ary relationships. The number of entity sets that participate in a relationship set is the ***degree*** of the relationship set. A binary relationship set is of degree 2; a ternary relationship set is of degree 3.

Attributes: For each attribute, there is a set of permitted values, called the ***domain***, or ***value set***. The domain attribute semester might be strings from the set {Fall, Winter, Spring, Summer}. Since an entity set may have several attributes, each entity can be described by a set of (attribute, data value) pairs.

Ex: A particular instructor entity may be described by the set {(ID, 76766), (name, Crick), (dept_name, Biology), (salary, 72000)}, meaning that the entity describes a person named Crick whose instructor ID is 76766, who is a member of the Biology department with salary of $72,000.

An attribute, as used in the E-R model, can be characterized by the following attribute types.

1. **Simple and Composite Attributes:** In our examples thus far, the attributes have been simple; that is, they have not been divided into subparts. Composite attributes, on the other hand, can be divided into subparts (that is, other attributes). For example, an attribute name could be structured as a composite attribute consisting of first name, middle name, and last name. Composite attributes help us to group together related attributes, making the modeling cleaner.

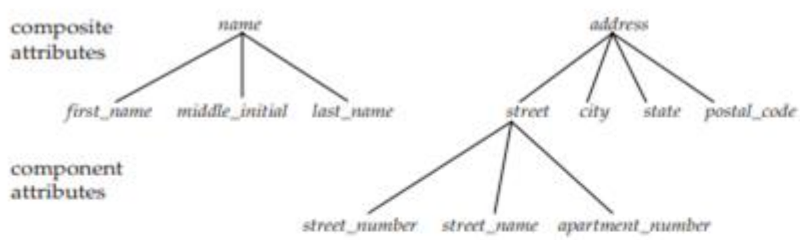


Figure Composite attributes instructor *name* and *address*.

Note: Using composite attributes in a design schema is a good choice if a user will wish to refer to an entire attribute on some occasions and to only a component of the attribute on other occasions.

2. **Single-valued and Multivalued Attributes:** There may be instances where an attribute has a set of values for a specific entity. Suppose an instructor may have one, or several phone numbers, and different instructors may have different numbers of phones. This type of attribute is said to be ***multivalued***. To denote that an attribute is multivalued, we enclose it in braces, for example {phone_number} or {dependent name}.

Note: Upper and lower bounds may be placed on number of values in a multivalued attribute, if needed.

3. **Derived Attributes:** Value for Derived attribute can be derived from values of other related attributes or entities. For instance, suppose that instructor entity set has an attribute age. If instructor entity set also has an attribute date_of_birth, we can calculate age from date_of_birth and current date. Thus, age is a derived attribute. In this case, date_of_birth may be referred to as a base attribute, or a stored attribute. Value of a derived attribute is not stored but is computed when required.

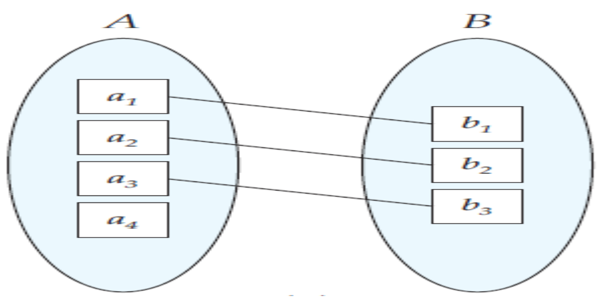
✓ An attribute takes a null value when an entity does not have a value for it. The null value may indicate "not applicable" (that is, that the value does not exist for the entity) or "unknown". An unknown value may be either missing (the value does exist, but we do not have that information) or not known (we do not know whether or not the value actually exists). A null value for the apartment number attribute could mean that the address does not include an apartment number (not applicable), that an apartment number exists but we do not know what it is (missing), or that we do not know whether or not an apartment number is part of the instructor’s address (unknown).

CONSTRAINTS

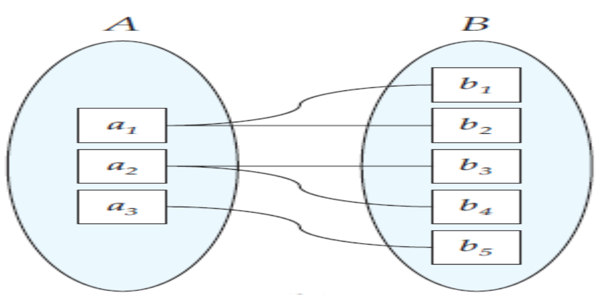
An E-R schema may define certain constraints to which the contents of a database must conform.

Mapping Cardinalities: Mapping cardinalities, or cardinality ratios, express the number of entities to which another entity can be associated via a relationship set. Mapping cardinalities are most useful in describing binary relationship sets. For a binary relationship set R between entity sets A and B, the mapping cardinality must be one of the following:

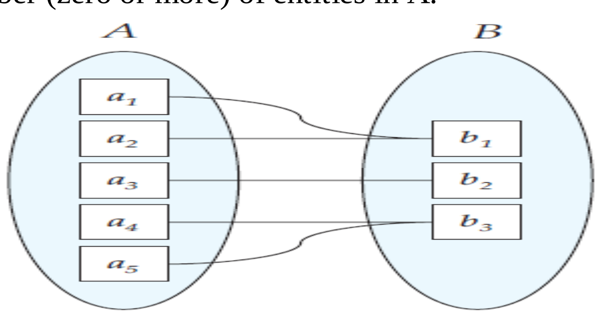
1) **One-to-One:** An entity in A is associated with at most one entity in B, and an entity in B is associated with at most one entity in A.



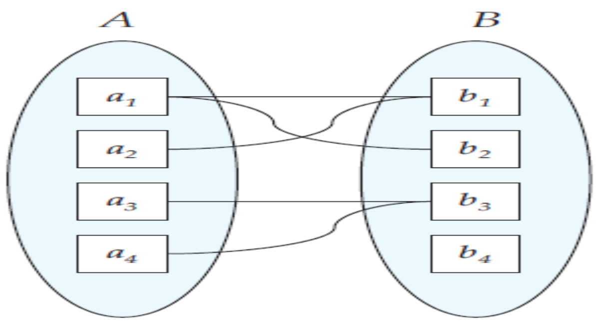
2) **One-to-Many:** An entity in A is associated with any number (zero or more) of entities in B. An entity in B, however, can be associated with at most one entity in A.



3) **Many-to-One:** An entity in A is associated with at most one entity in B. An entity in B, however, can be associated with any number (zero or more) of entities in A.



4) **Many-to-Many:** An entity in A is associated with any number (zero or more) of entities in B, and an entity in B is associated with any number (zero or more) of entities in A.



The appropriate mapping cardinality for a particular relationship set obviously depends on the real-world situation that the relationship set is modeling. As an illustration, consider the advisor relationship set. If, in a particular university, a student can be advised by only one instructor, and an instructor can advise several students, then the relationship set from instructor to student is one-to-many. If a student can be advised by several instructors (as in case of students advised jointly), relationship set is many-to-many.

Participation Constraints: The participation of an entity set E in a relationship set R is said to be total if every entity in E participates in at least one relationship in R. If only some entities in E participate in relationships in R, the participation of entity set E in relationship R is said to be partial.

Keys: We must have a way to specify how entities within a given entity set are distinguished. Conceptually, individual entities are distinct; from a database perspective, however, the differences among them must be expressed in terms of their attributes. Therefore, the values of the attribute values of an entity must be such that they can uniquely identify the entity. In other words, no two entities in an entity set are allowed to have exactly the same value for all attributes. ***A key for an entity is a set of attributes that suffice to distinguish entities from each other.*** Keys also help to identify relationships uniquely, and thus distinguish relationships from each other.

The **Primary Key** of an entity set allows us to distinguish among the various entities of the set. We use a similar mechanism to distinguish among the various relationships of a relationship set. The structure of the primary key for the relationship set depends on the mapping cardinality of the relationship set.

A ***super key*** is a set of columns (attributes) to uniquely identify rows in a table. If the attribute names of primary keys are not unique across entity sets, the attributes are renamed to distinguish them; the name of the entity set combined with the name of the attribute would form a unique name.

- ❖ If an entity set participates more than once in a relationship set, the role name is used instead of the name of the entity set, to form a unique attribute name.

REMOVING REDUNDANT ATTRIBUTES IN ENTITY SETS

When we design a database using the E-R model, we usually start by *identifying the entity sets* that should be included. For example, in the university organization, we include entity sets such as student, instructor, etc. Then we must *choose the appropriate attributes*. In the university organization, we decided that for the instructor entity set, we will include the attributes ID, name, dept_name, and salary. The choice of what attributes to include is up to the designer, who has a good understanding of the structure of the enterprise. Once the entities and their corresponding attributes are chosen, *the relationship sets among the various entities are formed*.

These relationship sets may result in a situation where attributes in the various entity sets are redundant and need to be removed from the original entity sets. To illustrate, consider the entity sets instructor and department:

- ❖ Entity set instructor includes attributes ID, name, dept_name, and salary, with ID forming primary key.
- ❖ Entity set department includes attributes dept_name, building, and budget, with dept_name forming primary key.

The attribute dept_name appears in both entity sets. Since it is the primary key for the entity set department, it is redundant in the entity set instructor and needs to be removed. When we create a relational schema from the E-R diagram, the attribute dept_name in fact gets added to the relation instructor, but only if each instructor has at most one associated department. If an instructor has more than one associated department, the relationship between instructors and departments is recorded in a separate relation inst_dept.

Note: Treating the connection between instructors and departments uniformly as a relationship, rather than as an attribute of instructor, makes the logical relationship explicit, and helps avoid a premature assumption that each instructor is associated with only one department.

A good entity-relationship design does not contain redundant attributes.

For our university example, we list entity sets and their attributes below, with primary keys underlined:

- ✓ classroom: with attributes (building, room_number, capacity).
- ✓ department: with attributes (dept_name, building, budget).
- ✓ course: with attributes (course_id, title, credits).
- ✓ instructor: with attributes (ID, name, salary).
- ✓ section: with attributes (course_id, sec_id, semester, year).
- ✓ student: with attributes (ID, name, tot_cred).

The relationship sets in our design are listed below:

- ✓ inst_dept: relating instructors with departments.
- ✓ stud_dept: relating students with departments.
- ✓ teaches: relating instructors with sections.
- ✓ takes: relating students with sections, with a descriptive attribute grade.
- ✓ course_dept: relating courses with departments.
- ✓ sec_course: relating sections with courses.
- ✓ sec_class: relating sections with classrooms.
- ✓ advisor: relating students with instructors.

You can verify that none of the entity sets has any attribute that is made redundant by one of the relationship sets.

ENTITY-RELATIONSHIP DIAGRAMS

An **E-R diagram** can express the overall logical structure of a database graphically. *E-R diagrams are simple and clear*—qualities that may well account in large part for the widespread use of the E-R model.

Basic Structure: An E-R diagram consists of the following major components:

- **Rectangles divided into two parts** represent entity sets. The first part, contains the name of the entity set. The second part contains the names of all the attributes of the entity set.
- **Diamonds** represent relationship sets.
- **Undivided rectangles** represent the attributes of a relationship set. Attributes that are part of the primary key are underlined.
- **Lines** link entity sets to relationship sets.
- **Dashed lines** link attributes of a relationship set to the relationship set.
- **Double lines** indicate total participation of an entity in a relationship set.
- **Double diamonds** represent identifying relationship sets linked to weak entity sets.

Figure E-R diagram corresponding to instructors and students.

Consider the E-R diagram in figure, which consists of two entity sets, *instructor* and *student* related through a binary relationship set *advisor*. The attributes associated with *instructor* are *ID*, *name*, and *salary*. The attributes associated with *student* are *ID*, *name*, and *tot_cred*. In figure, attributes of an entity set that are members of the primary key are underlined.

If a relationship set has some attributes associated with it, then we enclose the attributes in a rectangle and link the rectangle with a dashed line to the diamond representing that relationship set. For example, in below figure, we have the *date* descriptive attribute attached to the relationship set *advisor* to specify the date on which an instructor became the advisor.

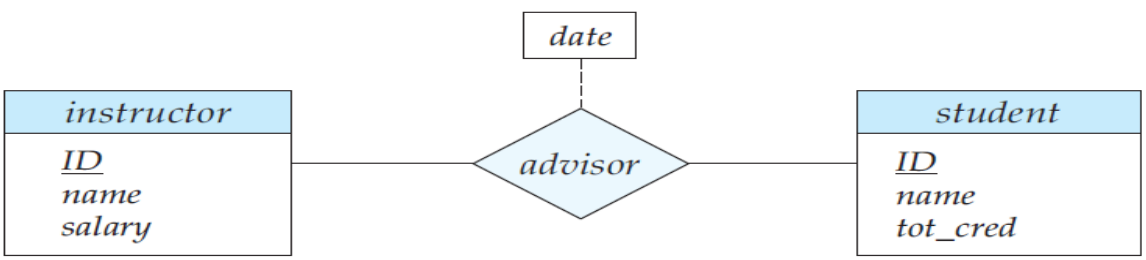


Figure E-R diagram with an attribute attached to a relationship set.

Mapping Cardinality: The relationship set *advisor*, between the *instructor* and *student* entity sets may be one-to-one, one-to-many, many-to-one, or many-to-many. To distinguish among these types, we draw either a directed line (→) or an undirected line (—) between the relationship set and the entity set in question, as follows:

1) **One-to-one:** We draw a directed line from the relationship set *advisor* to both entity sets *instructor* and *student*. This indicates that an instructor may advise at most one student, and a student may have at most one advisor.

2) **One-to-many:** We draw a directed line from the relationship set *advisor* to the entity set *instructor* and an undirected line to the entity set *student*. This indicates that an instructor may advise many students, but a student may have at most one advisor.

3) **Many-to-one:** We draw an undirected line from the relationship set *advisor* to the entity set *instructor* and a directed line to the entity set *student*. This indicates that an instructor may advise at most one student, but a student may have many advisors.

4) **Many-to-many:** We draw an undirected line from the relationship set *advisor* to both entity sets *instructor* and *student*. This indicates that an instructor may advise many students, and a student may have many advisors.

E-R diagrams also provide a way to indicate more complex constraints on the number of times each entity participates in relationships in a relationship set. ***A line may have an associated minimum and maximum cardinality, shown in the form l..h, where l is the minimum and h the maximum cardinality.*** A minimum value of 1 indicates total participation of the entity set in the relationship set; that is, each entity in the entity set occurs in at least one relationship in that relationship set. A maximum value of 1 indicates that the entity participates in at most one relationship, while a maximum value ∗ indicates no limit.

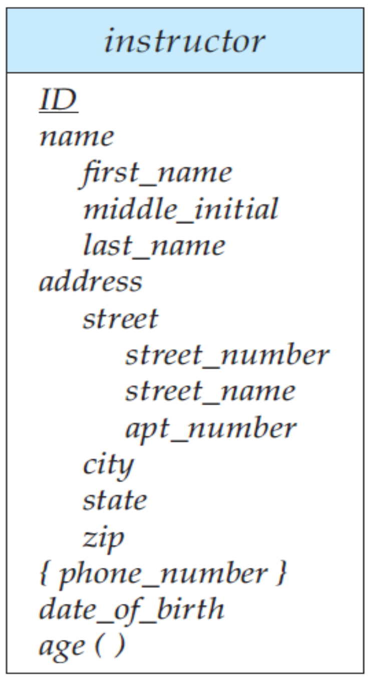
Figure Cardinality limits on relationship sets.

For example, consider the figure, the line between *advisor* and *student* has a cardinality constraint of 1..1, meaning the minimum and the maximum cardinality are both 1. That is, each student must have exactly one advisor. The limit 0..∗ on the line between *advisor* and *instructor* indicates that an instructor can have zero or more students. Thus, the relationship *advisor* is one-to-many from *instructor* to *student*, and further the participation of *student* in *advisor* is total, implying that a student must have an advisor.

If both edges have a maximum value of 1, relationship is one-to-one. If we had specified a cardinality limit of 1..∗ on left edge, we would be saying that each instructor must advise at least one student.

Note: The E-R diagram in the figure could alternatively have been drawn with a double line from *student* to *advisor* and an arrow on the line from *advisor* to *instructor*, in place of cardinality constraints shown. This alternative diagram would enforce exactly the same constraints as the constraints shown in figure.

Complex Attributes: Following figure shows how composite attributes can be represented in the E-R notation. As an example, suppose we were to add an address to the *instructor* entity-set. The address can be defined as the composite attribute *address* with the attributes: *street*, *city*, *state*, and *zip code*. The attribute *street* is itself a composite attribute whose component attributes are *street number*, *street name*, and *apartment number*.
The figure also illustrates a multivalued attribute *phone_number*, denoted by "*{phone_number}*", and a derived attribute *age*, depicted by a "*age* ( )".



Roles: We indicate roles in E-R diagrams by labeling the lines that connect diamonds to rectangles. Figure 7.12 shows the role indicators *course_id* and *prereq id* between the *course* entity set and the *prereq* relationship set.

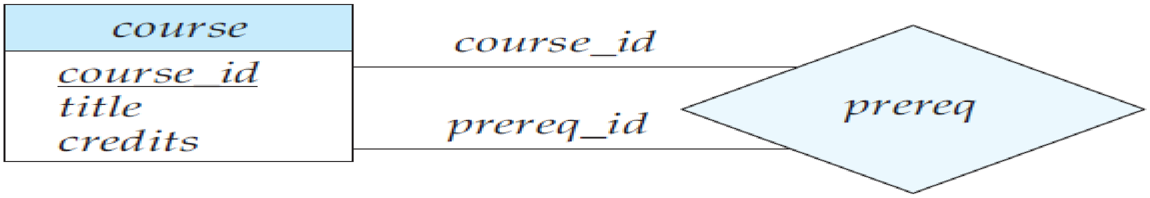


Figure E-R diagram with role indicators.

Non-binary/n-ary/Ternary Relationship Sets: Non-binary/n-ary/Ternary relationship sets can be specified easily in an E-R diagram. Figure 7.13 consists of the three entity sets *instructor*, *student*, and *project*, related through the relationship set *proj_guide*.

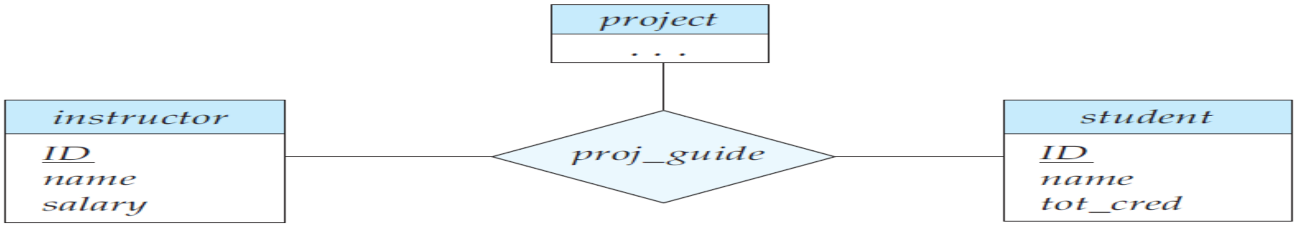


Figure E-R diagram with a ternary relationship.

We can specify some types of many-to-one relationships in the case of non-binary relationship sets. Suppose a *student* can have at most one instructor as a guide on a project. This constraint can be specified by an arrow pointing to *instructor* on the edge from *proj_guide*.

We permit at most one arrow out of a relationship set, since an E-R diagram with two or more arrows out of a non-binary relationship set can be interpreted in two ways.

Suppose there is a relationship set R between entity sets $A_1, A_2, \ldots, A_n$, and the only arrows are on the edges to entity sets $A_{i+1}, A_{i+2}, \ldots, A_n$. Then, the two possible interpretations are:

1. A particular combination of entities from $A_1, A_2, \ldots, A_i$ can be associated with at most one combination of entities from $A_{i+1}, A_{i+2}, \ldots, A_n$. Thus, the primary key for the relationship R can be constructed by the union of the primary keys of $A_1, A_2, \ldots, A_i$.

2. For each entity set A_k , $i < k \leq n$, each combination of the entities from other entity sets can be associated with at most one entity from A_k. Each set $\{A_1, A_2, \ldots, A_{k-1}, A_{k+1}, \ldots, A_n\}$, for $i < k \leq n$, then forms a candidate key.

To avoid confusion, we permit only one arrow out of a relationship set, in which case the two interpretations are equivalent.

Weak Entity Sets: An entity set that is dependent on other entity set (or) an entity set that does not have a primary key is referred to as a **weak entity set**. An entity set that has a primary key is termed a **strong entity set**.

For a weak entity set to be meaningful, it must be associated with another entity set, called the **identifying** or **owner entity set**. The identifying entity set is said to **own** the weak entity set that it identifies. The relationship associating the weak entity set with the identifying entity set is called the **identifying relationship**. The identifying relationship is many-to-one from the weak entity set to the identifying entity set, and the participation of the weak entity set in the relationship is total.

In our example, the identifying entity set for *section* is *course*, and the relationship *sec_course*, which associates *section* entities with their corresponding *course* entities, is the identifying relationship.

The **discriminator (***partial key***)** of a weak entity set is a set of attributes that allows the distinction to be made as weak entity set does not have Primary Key. For example, the discriminator of the weak entity set *section* consists of the attributes *sec_id*, *year*, and *semester*, since, for each course, this set of attributes uniquely identifies one single section for that course.

The primary key of a weak entity set is formed by the primary key of the identifying entity set, plus the weak entity set's discriminator. In the case of the entity set *section*, its primary key is *{course_id*, *sec_id*, *year*, *semester}*, where *course_id* is the primary key of the identifying entity set, namely *course*, and *{sec_id*, *year*, *semester}* distinguishes *section* entities for the same course.

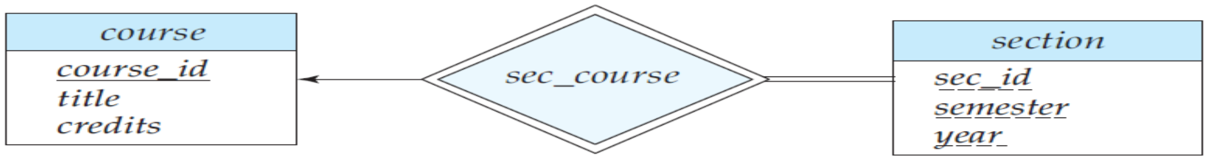


Figure E-R diagram with a weak entity set.

- In Figure, the weak entity set *section* depends on the strong entity set *course* via the relationship set *sec_course*.

In E-R diagrams, a weak entity set is depicted with a rectangle, like a strong entity set, but there are two main differences:

- ✓ The discriminator of a weak entity is underlined with a dashed, rather than a solid, line.
- ✓ Relationship set connecting weak entity set to identifying strong entity set is depicted by a double diamond.

The above figure also illustrates the use of double lines to indicate *total participation*; finally, the arrow from *sec_course* to *course* indicates that each section is related to a single course.

Note: A weak entity set can also participate in relationships other than the identifying relationship. A weak entity set may participate as owner in an identifying relationship with another weak entity set. It is also possible to have a weak entity set with more than one identifying entity set. A particular weak entity would then be identified by a combination of entities, one from each identifying entity set. The primary key of the weak entity set would consist of the union of the primary keys of the identifying entity sets, plus the discriminator of the weak entity set. A weak entity set representation more aptly models a situation where set participates in relationships than identifying relationship, and where weak entity set has several attributes.

REDUCTION TO RELATIONAL SCHEMAS

We can represent a database that conforms to an E-R database schema by a collection of relation schemas. For each entity set and for each relationship set in the database design, there is a unique relation schema to which we assign the name of the corresponding entity set or relationship set.

Both the E-R model and the relational database model are abstract, logical representations of real-world enterprises. Because the two models employ similar design principles, we can convert an E-R design into a relational design.

1. Representation of Strong Entity Sets with Simple Attributes: Let E be a strong entity set with only simple descriptive attributes $a_1, a_2, \ldots, a_n$. We represent this entity by a schema called E with n distinct attributes. Each tuple in a relation on this schema corresponds to one entity of the entity set E.

For schemas derived from strong entity sets, the primary key of the entity set serves as the primary key of the resulting schema. This follows directly from the fact that each tuple corresponds to a specific entity in the entity set.

As an example, consider the entity set *student* of above E-R diagram. This entity set has three attributes: *ID*, *name*, *tot_cred*. We represent this entity set by a schema called *student* with three attributes: *student* (*ID*, *name*, *tot_cred*)

Note that since student ID is primary key of entity set, it is also the primary key of the relation schema.

2. Representation of Strong Entity Sets with Complex Attributes: Composite attributes are handled by creating a separate attribute for each of the component attributes; we do not create a separate attribute for the composite attribute itself.

To illustrate, consider the *instructor* entity set. For the composite attribute *name*, the schema generated for *instructor* contains the attributes *first_name*, *middle_name*, and *last_name*; there is no separate attribute for *name*. Similarly, for the composite attribute *address*, the schema generated contains the attributes *street*, *city*, *state*, and *zip code*. Since *street* is a composite attribute it is replaced by *street number*, *street name*, and *apt number*. The relational schema derived from the version of entity set *instructor* with complex attributes, without including the multivalued attribute, is thus:

instructor (*ID, first_name*, *middle_name*, *last_name*, *street_number*, *street_name, apt_number*, *city*, *state*, *zip_code*, *date_of_birth*)

- ***Multivalued attributes are treated differently from other attributes.*** We have seen that attributes in an E-R diagram generally map directly into attributes for appropriate relation schemas. Multivalued attributes, however, are an exception; new relation schemas are created for these attributes. For a multivalued attribute *M*, we create a relation schema *R* with an attribute *A* that corresponds to *M* and attributes corresponding to primary key of the entity set or relationship set of which *M* is an attribute.

As an example, consider entity set *instructor*, which includes multivalued attribute *phone_number*. Primary key of *instructor* is *ID*. For this multivalued attribute, we create a relation schema *instructor_phone* (*ID*, *phone_number*). Each phone_number of an instructor is represented as a unique tuple in relation on this schema. Thus, if we had an instructor with *ID* 22222, and phone_numbers 9849012345 and 9848012345, the relation *instructor phone* would have two tuples (22222, 9849012345) and (22222, 9848012345).

We create a primary key of the relation schema consisting of all attributes of the schema. We also create a foreign-key on the relation schema created from multivalued attribute, with the primary key of the entity set referencing the relation generated from the entity set. In the above example, the foreign-key constraint on the *instructor_phone* relation would be that attribute *ID* references the *instructor* relation.

- Derived attributes are not explicitly represented in the relational data model. However, they can be represented as “methods” in other data models such as the object-relational data model

3. Representation of Weak Entity Sets: Let *A* be a weak entity set with attributes $a_1, a_2, \ldots, a_m$. Let *B* be the strong entity set with attributes $b_1, b_2, \ldots, b_n$ on which *A* depends. Let the primary key of *B* is b_1. We represent the entity set *A* by a relation schema called *A* as: $\{a_1, a_2, \ldots, a_m\} \cup \{b_1\}$

For schemas derived from a weak entity set, the combination of the primary key of the strong entity set and the discriminator of the weak entity set serves as the primary key of the schema. We also create a foreign-key constraint on the relation *A*, specifying that the attribute b_1 reference the primary key of the relation *B*. The foreign key constraint ensures that for each tuple representing a weak entity, there is a corresponding tuple representing the corresponding strong entity.

Ex: *section* (*course_id*, *sec_id*, *semester*, *year*)

✓ Because of the "on delete cascade" specification on the foreign key constraint, if a *course* entity is deleted, then so are all the associated *section* entities.

4. Representation of Relationship Sets: Let *R* be a relationship set, let $a_1, a_2, \ldots, a_m$ be the set of attributes formed by the union of the primary keys of each of the entity sets participating in *R*, and let the descriptive attributes (if any) of *R* be $b_1, b_2, \ldots, b_n$. We represent this relationship set by a relation schema called *R* with one attribute for each member of the set: $\{a_1, a_2, \ldots, a_m\} \cup \{b_1, b_2, \ldots, b_n\}$

The primary key is chosen as follows:

- ❖ For a binary many-to-many relationship, the union of the primary-key attributes from the participating entity sets becomes the primary key.
- ❖ For a binary one-to-one relationship set, the primary key of either entity set can be chosen as the primary key.
- ❖ For a binary many-to-one or one-to-many relationship set, the primary key of the entity set on the "many" side of the relationship set serves as the primary key.
- ❖ For an *n*-ary relationship set without any arrows on its edges, the union of the primary key-attributes from the participating entity sets becomes the primary key.
- ❖ For an *n*-ary relationship set with an arrow on one of its edges, the primary keys of the entity sets not on the "arrow" side of the relationship set serve as the primary key for the schema. Recall that we allowed only one arrow out of a relationship set.

We can also create foreign-key constraints on the relation schema *R*. As an illustration, consider the relationship set *advisor* in the E-R diagram. This relationship set involves the following two entity sets:

- *instructor* with the primary key *ID*.
- *student* with the primary key *ID*.

Since the relationship set has no attributes, the *advisor* schema has two attributes, the primary keys of *instructor* and *student*. Since both attributes have the same name, we rename them *i_ID* and s*_ID*. Since the *advisor* relationship set is many-to- one from *student* to *instructor* the primary key for the *advisor* relation schema is *s ID*.

Note: The schema for the relationship set linking a weak entity set to its corresponding strong entity set is redundant and does not need to be present in a relational database design based upon an E-R diagram.

ENTITY-RELATIONSHIP DESIGN ISSUES

1. Use of Entity Sets versus Attributes: Two natural questions arise in a database design are: What constitutes an attribute, and what constitutes an entity set? Unfortunately, there are no simple answers.

The distinctions mainly depend on the structure of the real-world enterprise being modeled, and on the semantics associated with the attribute in question.

A common mistake is to use the primary key of an entity set as an attribute of another entity set, instead of using a relationship. Another related mistake that people sometimes make is to designate the primary-key attributes of the related entity sets as attributes of the relationship set. This should not be done since the primary-key attributes are already implicit in the relationship set.

2. Use of Entity Sets versus Relationship Sets: It is not always clear whether an object is best expressed by an entity set or a relationship set. One possible guideline in determining whether to use an entity set or a relationship set is to designate a relationship set to describe an action that occurs between entities. This approach can also be useful in deciding whether certain attributes may be more appropriately expressed as relationships.

3. Binary versus *n*-ary Relationship Sets: Relationships in databases are often binary. Some relationships that appear to be non-binary could be better represented by several binary relationships. It is always possible to replace a non-binary (n-ary, for $n > 2$) relationship set by a number of distinct binary relationship sets. For simplicity, consider the abstract ternary ($n = 3$) relationship set R, relating entity sets A, B, and C. We replace the relationship set R by an entity set E, and create three relationship sets as shown in figure.

RA, relating E and A.
RB, relating E and B.
RC, relating E and C.

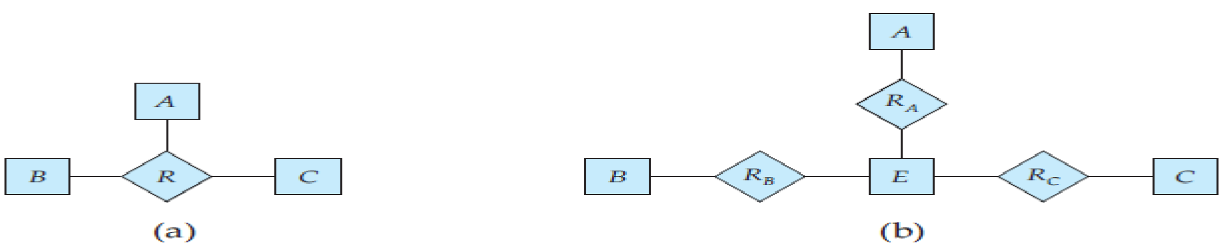


Figure Ternary relationship versus three binary relationships.

If the relationship set R had any attributes, these are assigned to entity set E; further, a special identifying attribute is created for E. For each relationship (a_i , b_i , c_i) in the relationship set R, we create a new entity e_i in the entity set E. Then, in each of the three new relationship sets, we insert a relationship as follows:

(e_i , a_i) in RA.
(e_i , b_i) in RB.
(e_i , c_i) in RC.

We can generalize this process in a straightforward manner to n-ary relationship sets. Conceptually, we can restrict E-R model to include only binary relationship sets. But this restriction is not always desirable.

- ✓ An identifying attribute may have to be created for the entity set created to represent the relationship set to increase the complexity of the design and overall storage requirements.
- ✓ An n-ary relationship set shows more clearly that several entities participate in a single relationship.
- ✓ There may not be a way to translate constraints on the ternary relationship into constraints on the binary relationships.

4. Placement of Relationship Attributes: The cardinality ratio of a relationship can affect the placement of relationship attributes. Thus, attributes of one-to-one or one-to-many relationship sets can be associated with one of the participating entity sets, rather than with the relationship set.

- ✓ For one-to-one relationship sets, relationship attribute can be associated with either of the participating entities.
- ✓ Attributes of a one-to-many relationship set can be repositioned to only the entity set on the "many" side of the relationship.
  - ■ The design decision of where to place descriptive attributes in such cases, as a relationship or entity attribute, should reflect the characteristics of the enterprise being modeled.
- ✓ The choice of attribute placement is more clear-cut for many-to-many relationship sets. When an attribute is determined by the combination of participating entity sets, rather than by either entity separately, that attribute must be associated with the many-to-many relationship set.

EXTENDED E-R FEATURES

Although the basic E-R concepts can model most database features, some aspects of a database may be more aptly expressed by certain extensions to the basic E-R model. The extended E-R features include: specialization, generalization, attribute inheritance, and aggregation.

- ❖ **Specialization:** "The process of designating sub groupings within an entity set is called **specialization**". An entity set may include sub groupings of entities that are distinct in some way from other entities in the set. The E-R model provides a means for representing these distinctive entity groupings. As an example, the entity set *person* may be further classified as one of the following:
  - *employee*.
  - *student*.

Each of these person types is described by a set of attributes that includes all attributes of entity set *person* plus possibly additional attributes. For example, *employee* entities may be described further by attribute *salary*, whereas *student* entities may be described further by attribute *tot_cred*. The specialization of *person* allows us to distinguish among person entities according to whether they correspond to employees or students: in general, a person could be an employee, a student, both, or neither.

We can apply specialization repeatedly to refine a design. For instance, university employees may be further classified as one of the following:

- *instructor*.
- *secretary*.

Each of these employee types is described by a set of attributes that includes all the attributes of entity set *employee* plus additional attributes. For example, *instructor* entities may be described further by the attribute *rank* while *secretary* entities are described by the attribute *hours per week*. An entity set may be specialized by more than one distinguishing feature. In our example, the distinguishing feature among employee entities is the job the employee performs.

We refer to this relationship as the ***ISA relationship***, which stands for "is a" and represents, for example, that an instructor "is a" employee. In terms of an E-R diagram, specialization is depicted by a hollow arrow-head pointing from the specialized entity to the other entity.

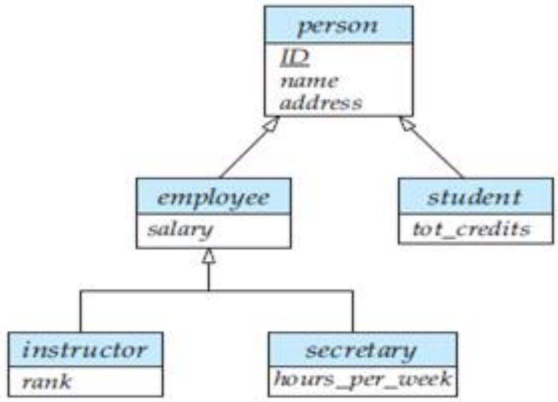


Figure Specialization and generalization.

The way we depict specialization in an E-R diagram depends on whether an entity may belong to multiple specialized entity sets or if it must belong to at most one specialized entity set. The former case (multiple sets permitted) is called **overlapping specialization**, while the latter case (at most one permitted) is called **disjoint specialization**. For an overlapping specialization (as is the case for *student* and *employee* as specializations of *person*), two separate arrows are used. For a disjoint specialization (as is the case for *instructor* and *secretary* as specializations of *employee*), a single arrow is used.

The specialization relationship may also be referred to as a **superclass-subclass** relationship. Higher- and lower-level entity sets also may be designated by the terms **superclass** and **subclass**, respectively. The refinement from an initial entity set into successive levels of entity sub groupings represents a **top-down** design process in which distinctions are made explicit.

❖ **Generalization:** "The design process in which multiple entity sets are synthesized into a higher-level entity set on the basis of common features is known as **generalization**". This is a **bottom-up** design process.

The database designer may have first identified:

• *instructor* entity set with attributes *instructor_id*, *instructor_name*, *instructor_salary*, and *rank*.

• *secretary* entity set with attributes *secretary_id*, *secretary_name*, *secretary_salary*, and *hours per week*.

There are similarities between the *instructor* entity set and the *secretary* entity set in the sense that they have several attributes that are conceptually the same across the two entity sets: namely, the id, name, and salary attributes. To create a generalization, the attributes must be given a common name and represented with the higher-level entity *employee*. In our example, *employee* is the higher-level entity set and *instructor* and *secretary* are lower-level entity sets. The *employee* entity set is the superclass of the *instructor* and *secretary* subclasses.

For all practical purposes, generalization is a simple inversion of specialization. We apply both processes, in combination, in the course of designing the E-R schema for an enterprise. In terms of the E-R diagram itself, we do not distinguish between specialization and generalization. New levels of entity representation are distinguished (specialization) or synthesized (generalization) as the design schema comes to express fully the database application and the user requirements of the database. Differences in the two approaches may be characterized by their starting point and overall goal.

Constraints on Generalizations: One type of constraint involves determining which entities can be members of a given lower-level entity set. Such membership may be one of the following:

1. **Condition-defined:** In condition-defined lower-level entity sets, membership is evaluated on the basis of whether or not an entity satisfies an explicit condition or predicate. Since all lower-level entities are evaluated on the basis of the same attribute this type of generalization is said to be **attribute-defined**.
2. **User-defined:** User-defined lower-level entity sets are not constrained by a membership condition; rather, the database user assigns entities to a given entity set. The assignment is implemented by an operation that adds an entity to an entity set.

❖ A second type of constraint relates to whether or not entities may belong to more than one lower-level entity set within a single generalization. The lower level entity sets may be one of the following:

  1. **Disjoint:** A *disjoint constraint* requires an entity belong to no more than one lower-level entity set.
  2. **Overlapping:** In *overlapping generalizations*, the same entity may belong to more than one lower-level entity set within a single generalization.

❖ A final constraint, the **completeness constraint** on a generalization or specialization, specifies whether or not an entity in the higher-level entity set must belong to at least one of the lower-level entity sets within the generalization/specialization. This constraint may be one of the following:

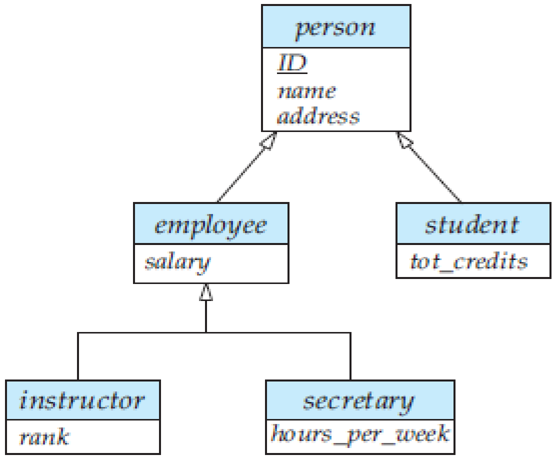
  1. **Total Generalization** or **Specialization:** Each higher-level entity must belong to a lower-level entity set.
  2. **Partial Generalization** or **Specialization:** Some higher-level entities may not belong to any lower-level entity set.

<u>**Note:**</u> Partial generalization is the default. We can specify total generalization in an E-R diagram by adding the keyword "total" in the diagram and drawing a dashed line from the keyword to the corresponding hollow arrow-head to which it applies (for a total generalization), or to the set of hollow arrow-heads to which it applies (for an overlapping generalization).

Attribute Inheritance: A crucial property of the higher- and lower-level entities created by specialization and generalization is ***attribute inheritance***. The attributes of the higher-level entity sets are said to be inherited by the lower-level entity sets.

For example, *student* and *employee* inherit the attributes of *person*. Thus, *student* is described by its *ID*, *name*, and *address* attributes, and additionally a *tot_cred* attribute; *employee* is described by its *ID*, *name*, and *address* attributes, and additionally a *salary* attribute. Attribute inheritance applies through all tiers of lower-level entity sets; thus, *instructor* and *secretary*, which are subclasses of *employee*, inherit the attributes *ID*, *name*, and *address* from *person*, in addition to inheriting *salary* from *employee*.

A lower-level entity set (or subclass) also inherits participation in the relationship sets in which its higher-level entity (or superclass) participates. Like attribute inheritance, participation inheritance applies through all tiers of lower-level entity sets. For example, suppose the *person* entity set participates in a relationship *person_dept* with *department*. Then, the *student*, *employee*, *instructor* and *secretary* entity sets, which are subclasses of the *person* entity set, also implicitly participate in the *person_dept* relationship with *department*.



The above figure depicts a **hierarchy** of entity sets. In the figure, *employee* is a lower-level entity set of *person* and a higher-level entity set of the *instructor* and *secretary* entity sets. In a hierarchy, a given entity set may be involved as a lower level entity set in only one ISA relationship; that is, entity sets in this diagram have only **single inheritance**. If an entity set is a lower-level entity set in more than one ISA relationship, then the entity set has **multiple inheritance**, and the resulting structure is said to be a ***lattice***.

❖ **Aggregation:** One limitation of the E-R model is that it cannot express relationships among relationships. To illustrate the need for such a construct, consider the ternary relationship proj_guide, between an instructor, student and project. Now suppose that each instructor guiding a student on a project is required to file a monthly evaluation report. We model the evaluation report as an entity evaluation, with a primary key evaluation id. One alternative for recording the (student, project, instructor) combination to which an evaluation corresponds is to create a quaternary (4-way) relationship set eval_for between instructor, student, project, and evaluation.

Using the basic E-R modeling constructs, we obtain the following E-R diagram.

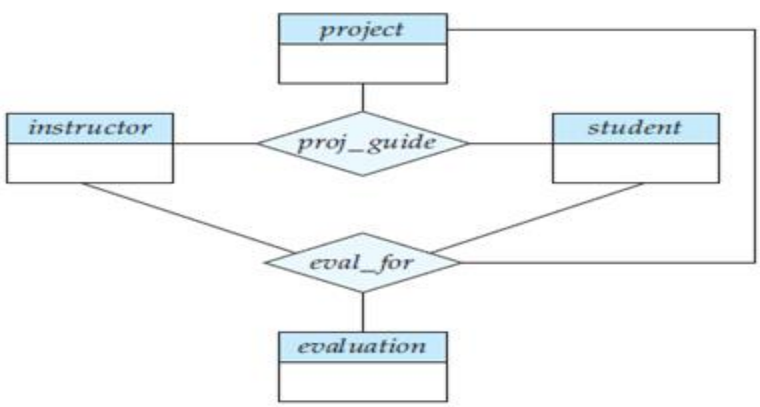


Figure E-R diagram with redundant relationships.

It appears that the relationship sets proj_guide and eval_for can be combined into one single relationship set. Nevertheless, we should not combine them into a single relationship, since some instructor, student, project combinations may not have an associated evaluation.

There is ***redundant information*** in the resultant figure, however, since every instructor, student, project combination in eval_for must also be in proj_guide. The best way to model a situation such as the one just described is to use aggregation. **Aggregation** is an abstraction through which relationships are treated as higher-level entities. Thus, for our example, we regard the relationship set proj_guide (relating the entity sets instructor, student, and project) as a higher-level entity set called proj_guide. Such an entity set is treated in the same manner as is any other entity set.

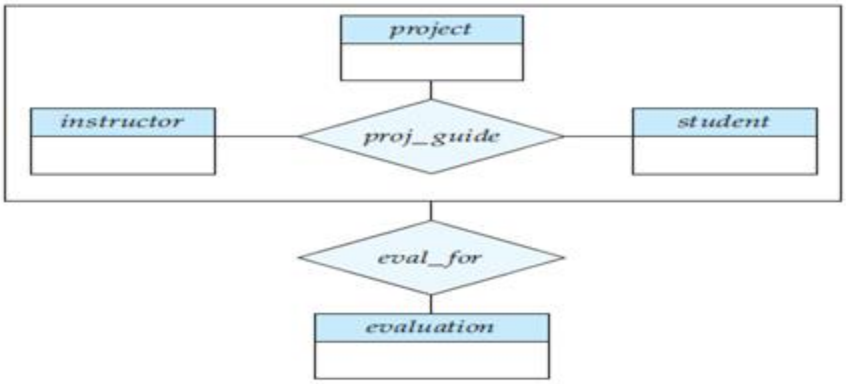


Figure E-R diagram with aggregation.

We can then create a binary relationship eval_for between proj_*guide* and *evaluation* to represent which (*student*, *project*, *instructor*) combination an *evaluation* is for, as shown in the figure.

REDUCTION TO RELATION SCHEMAS

Representation of Generalization: There are two different methods of designing relation schemas for an E-R diagram that includes generalization. We refer to Generalization figure and simplify it by including only the first tier of lower-level entity sets—that is, *employee* and *student*. We assume that *ID* is the primary key of *person*.

1) Create a schema for the higher-level entity set. For each lower-level entity set, create a schema that includes an attribute for each of the attributes of that entity set plus one for each attribute of the primary key of the higher-level entity set. Thus, for the E-R diagram of figure we have three schemas: *person* (*ID*, *name*, *street*, *city*), *employee* (*ID*, *salary*), *student* (*ID*, *tot_cred*)

❖ The primary-key attributes of the higher-level entity set become primary key attributes of all lower-level entity sets also which can be underlined. In addition, we create foreign-key constraints on the lower-level entity sets. In the above example, the *ID* attribute of *employee* would reference the primary key of *person*, and similarly for *student*.

2) An alternative representation is possible, if the generalization is disjoint and complete. Here, we do not create a schema for the higher-level entity set. Instead, for each lower level entity set, we create a schema that includes an attribute for each of the attributes of that entity set plus one for *each* attribute of the higher-level entity set. Then, for the E-R diagram of figure, we have two schemas: *employee* (*ID*, *name*, *street*, *city*, *salary*), *student* (*ID*, *name*, *street*, *city*, *tot_cred*)

Both these schemas have *ID*, which is the primary-key attribute of the higher level entity set *person*, as their primary key.

One drawback of second method lies in defining foreign-key constraints. To avoid this problem, we need to create a relation schema containing at least the primary-key attributes of same entity. If the second method were used for an overlapping generalization, some values would be stored multiple times.

Representation of Aggregation: Designing schemas for an E-R diagram containing aggregation is straightforward. Consider the diagram of Aggregation. Schema for the relationship set *eval_for* between the aggregation of *proj_guide* and the entity set *evaluation* includes an attribute for each attribute in the primary keys of the entity set *evaluation*, and the relationship set *proj_guide*. It also includes an attribute for any descriptive attributes, if they exist, of the relationship set *eval_for*. We then transform relationship sets and entity sets within the aggregated entity set following the rules we have already defined.

The rules for creating primary-key and foreign-key constraints on relationship sets can be applied to relationship sets involving aggregations as well, with the aggregation treated like any other entity set. The primary key of the aggregation is the primary key of its defining relationship set. No separate relation is required to represent the aggregation; the relation created from the defining relationship is used instead.

OTHER ASPECTS OF DATABASE DESIGN

Data Constraints and Relational Database Design: Constraints serve several purposes. The most obvious one is the automation of consistency preservation. By expressing constraints in the SQL DDL, the designer is able to ensure that the database system itself enforces the constraints. This is more reliable than relying on each application program individually to enforce constraints. It also provides a central location for the update of constraints and the addition of new ones.

A further advantage of stating constraints explicitly is that certain constraints are particularly useful in designing relational database schemas. Constraint enforcement comes at a potentially high price in performance each time the database is updated.

Data constraints are also useful in determining the physical structure of data. It may be useful to store data that are closely related to each other in physical proximity on disk so as to gain efficiencies in disk access. Certain index structures work better when the index is on a primary key.

Usage Requirements: Queries, Performance: Database system performance is a critical aspect of most enterprise information systems. There are two main metrics for performance:

1. ***Response time***—the amount of time a *single* transaction takes from start to finish in either the average case or the worst case.
2. ***Throughput***—the number of queries or updates (often referred to as *transactions*) that can be processed on average per unit of time.

Systems that process large numbers of transactions in a batch style focus on having high throughput. Systems that interact with people or time-critical systems often focus on response time. Most commercial database systems historically have focused on throughput; however, a variety of applications including Web-based applications require good response time.

Queries that involve joins require more resources to evaluate than those that do not. In cases where a join is required, the database administrator may choose to create an index that facilitates evaluation of that join. For queries—whether a join is involved or not—indices can be created to speed evaluation of selection predicates (SQL where clause) that are likely to appear.

Another aspect of queries that affects choice of indices is relative mix of update and read operations. While an index may speed queries, it also slows updates.

Authorization Requirements: Authorization constraints affect design of the database as well because SQL allows access to be granted to users on the basis of components of the logical design of the database. A relation schema may need to be decomposed into two or more schemas to facilitate the granting of access rights in SQL.

Data Flow, Workflow: The term *workflow* refers to the combination of data and tasks involved in processes like a CAD system. Workflows interact with the database system as they move among users and users perform their tasks on the workflow.

In addition to the data on which workflows operate, the database may store data about the workflow itself, including the tasks making up a workflow and how they are to be routed among users.

Other Issues in Database Design: Database design is usually not a one-time activity. The needs of an organization evolve continually, and the data that it needs to store also evolve correspondingly. During the initial database-design phases, the database designer may realize that changes are required at the conceptual, logical, or physical schema levels. Changes in the schema can affect all aspects of the database application. A good database design anticipates future needs of an organization, and ensures that the schema requires minimal changes as the needs evolve.

A good design should account not only for current policies, but should also avoid or minimize changes due to changes that are anticipated, or have a reasonable chance of happening.

Furthermore, the enterprise that the database is serving likely interacts with other enterprises and, therefore, multiple databases may need to interact. Conversion of data between different schemas is an important problem in real-world applications. The XML data model is widely used for representing data when it is exchanged between different applications.

Finally, it is worth noting that database design is a human-oriented activity in two senses: the end users of the system are people (even if an application sits between the database and the end users); and the database designer needs to interact extensively with experts in the application domain to understand the data requirements of the application. All the people involved with the data should be taken into account for a database design and deployment to succeed within the enterprise.

DATABASE DESIGN FOR BANKING ENTERPRISE