Contents:

- **Recommended Books:**

1. A Silberschatz, H Korth, S Sudarshan, “Database System and Concepts”, fifth Edition McGraw- Hill
2. Database Systems ,Rob Coronel,Cengage Learning.
3. Programming with PL/SQL for Beginners, H. Dand, R. Patil and T. Sambare,X-Team.
4. Introduction to Database System,C.J.Date,Pearson

- **Prerequisites and Linking:**

Unit I	Pre-requisites	Sem. II	Sem. III	Sem. IV	Sem. V	Sem. VI
Introduction to Databases and Transactions	-	Web Programming (tables, database), OOPS (variable, function)	DBMS	Core java	Enterprise java, Advanced web programming, Project	Project

Notes:

Database Management System:

A database management system (DBMS) is a collection of interrelated data and a set of programs to access those data. The collection of data, usually referred to as the database, contains information relevant to an enterprise. The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

Database System Applications

Databases are widely used. Here are some representative applications:

- Banking: For customer information, accounts, and loans, and banking transactions.
- Airlines : For reservations and schedule information. Airlines were among the first to use databases in a geographically distributed manner – terminals situated around the world accessed the central database system through phone lines and other data networks.
- Universities : For student information, course registrations, and grades.
- Credit card transactions : For purchases on credit cards and generation of monthly statements.
- Telecommunication : For keeping records of calls made, generating monthly bills, maintaining balances on prepaid calling cards, and storing information about the communication networks.
- Finance : For storing information about holdings, sales, and purchases of financial instruments such as stocks and bonds.
- Sales : For customer, product, and purchase information.

- Manufacturing : for management of supply chain and for tracking production of items in factories, inventories of items in warehouses/stores, and orders for items.
- Human resources : For information about employees, salaries, payroll taxes and benefits, and for generation of paychecks.

Purpose and View of Data

View of Data

A database system is a collection of interrelated files and a set of programs that allow users to access and modify these files. 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.

Source: https://www.youtube.com/watch?v=PI8UrOAZs2U

Data Abstraction

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database–systems users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users interactions with the system :

- 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.
- 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.

Figure 1 shows the relationship among the three levels of abstraction.

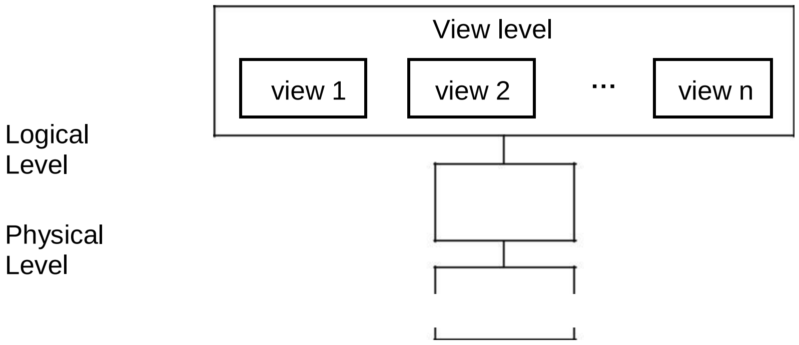


Fig. 1: The three levels of data abstraction.

RELATIONAL DATABASES

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

Tables

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 columns of the table correspond to the attributes of the record type.

It is not hard to see how tables may be stored in files. For instance, a special character (such as a comma) may be used to delimit the different attributes of a record, and another special character (such as a new-line character) may be used to delimit records. The relational model hides such low-level implementation details from database developers and users.

We also note that it is possible to create schemas in the relational model those problems such as unnecessarily duplicated information. For example, suppose we store the department budget as an attribute of the instructor record. Then, whenever the values of a particular budge (say that one for the Physics department).

ID	Name	dept_name	salary
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32143	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
3o456	Gold	Physic s	87000
76543	Singh	Finance	80000

(a) The instructor table.

dept_name	building	budget
Comp. Sci.	Taylor	100000
Biology	Watson	90000
Elec. Eng.	Taylor	35000
Music	Packard	80000
Finance	Painter	120000
History	Painter	130000
Phyics	Watson	70000

(b) The department table

Fig. 2 : A sample relational database.

Data-Manipulation Language

The SQL query language is nonprocedural. A query takes as input several tables (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 instructor.name
from instructor
where instructor.dept_name = 'History'
```

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
          (depLname          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. In addition, the DDL statement updates the data dictionary, which contains metadata. The schema of a table is an example of metadata.

Database Access from Application Programs

SQL is not as powerful as a universal Turing machine; that is, there are some computation that are possible using a general-purpose programming language but are not possible using SQL. SQL also does not support actions such as input users, output to displays, or communication over the network.

To access the database?

DMI statements need to be executed from the host language. 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 ODBC) standard provides corresponding features to the Java language.

- By extending the host language syntax to embed DML calls within the host language program. Usually a special character prefaces DML calls, and a preprocessor, called the DML precompiler, converts the DML statements to normal procedure calls in the host language.

DATABASE DESIGN

Database systems are designed to manage large bodies of information. These large bodies of information do not exist in isolation. They are part of the operation of some enterprise whose end product may be information from the database or may be some device or service for which the database plays only a supporting role.

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 frame-work 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, then, is to characterize fully the data needs of the prospective database users. The database designer needs to interact extensively with domain experts and users to carry out this task. The outcome of this phase is a specification of user requirements.

DATABASE ARCHITECTURE

Database System Structure

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.

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. A gigabyte is 1000 megabytes (1 billion bytes), and a terabyte is 1 million megabytes (1 trillion bytes). 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. Since the movement of data to and from disk is slow relative to the speed of the central processing unit, it is imperative that the database system structure the database so as to minimize the need to move data between disk and main memory.

Application Architectures

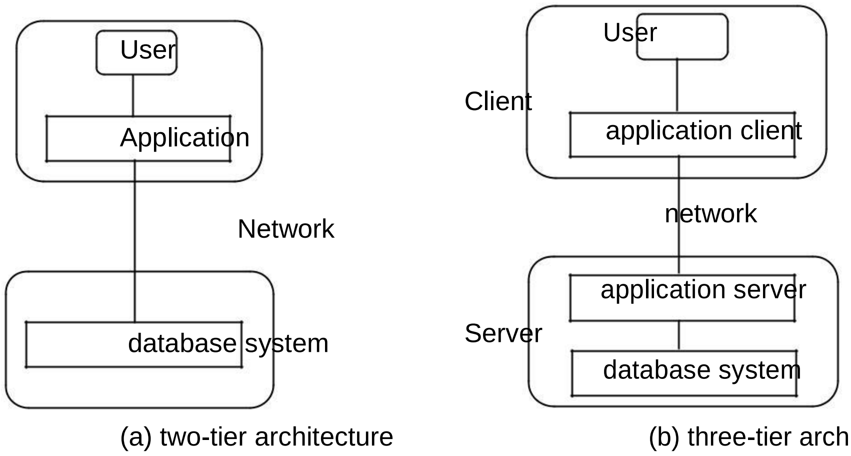


Fig. 2 : Two–tier and three-tier architectures.

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, in which remote database users work, and server machines, on which the database system runs.

Database applications are usually partitioned into two or three parts, as in figure 2. In a two–tier architecture, the application is partitioned into a component that resides at the client machine, which 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.

TRANSACTION MANAGEMENT

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. However, during the execution of a transaction, it may be necessary temporarily to allow inconsistency, since either the debit of A or the credit of B must be done before the other. This temporary inconsistency, although necessary, may lead to difficulty if a failure occurs.

It is the programmer's responsibility to define properly the various transactions, so that each preserves the consistency of the database. For example, the transaction to transfer funds from account A to account B could be defined to be composed of two separate programs: one that debits account A, and another that credits account B. The execution of these two programs one after the other will indeed preserve consistency. However, each program by itself does not transform the database from a consistent state to a new consistent state. Thus, those programs are not transactions.

Source: https://www.youtube.com/watch?v=r_RznCoUVIU

Data Models

IMPORTANCE OF DATA MODELS

•A data model is a set of concepts that can be used to describe the structure of data is a database.

•Data models are used to support the development of information system by prociding the definition & format of data to be involved in future system. Data model is acting like a guideline for development also gives idea about possible alternatives to achieve targeted solution.

EVOLUTION OF DATA MODELS

Source: https://www.youtube.com/watch?v=6uX_nXzeVDY

Early Data Models

As database management become popular during the 1970s and 1980s, a handful of popular data models emerged. Each of these early data models had advantages and disadvantages that played key roles in the development of the relational data model. In many ways, the relational data model represented an attempt to streamline and simplify the earlier data models. To understand the role and contribution of SQL and the relational model, it is useful to briefly examine some data models that preceded the development of SQL.

File Management Systems

Before the introduction of database management systems, all data permanently stored on a computer system, such as payroll and accounting records, was stored in individual files. A file management system, usually provided by the computer manufacturer as part of the computer's operating system, kept track of the names and locations of the files. The file management system basically had no data model; it knew nothing about the internal contents of files. To the file

Management system, a file containing a word processing document and a file containing payroll data appeared the same.

Knowledge about the contents of a file which data is contained and how the data was organized was embedded in the application programs that used the file, as shown in figure 1. If the structure of the data changed for example, if an additional item of data was to be stored for each employee every program that accessed the file had to be modified. As the number of files and programs grew over time, more and more of a data processing department's effort went into Maintaining existing applications rather than developing new ones.

The problems of maintaining large file based systems led in the late 1960s to the development of database management systems. The idea behind these systems was simple: take the definition of a file's content and structure out of the individual programs, and store it, together with the data, in a database. Using the information in the database, the DBMS that controlled it could take a much more active role in managing both the data and changes to the database structure.

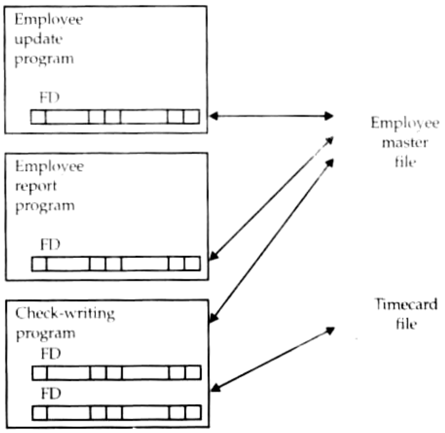


Fig. 1: A payroll application using a file management system.

Hierarchical Databases

One of the most important applications for the earliest database management systems was production planning for manufacturing companies. If an automobile manufacturer decided to produce 10,000 units of one car model and 5000 units of another model, it needed to know how many parts to order from its suppliers.

To answer the question, the product (a car) had to be decomposed into assemblies (engine, body, chassis), which were decomposed into subassemblies (valves, cylinders, spark plugs), and then into sub subassemblies, and so on. Handling this list of parts, known as a bill of materials, was a job tailor�made for computers.

The bill of materials for a product has a natural hierarchical structure. To store this data, the hierarchical data model, illustrated in figure 2, was developed. In this model, each record in the database represented a specific part. The records had parent/child relationships, linking each part to its subpart, and so on.

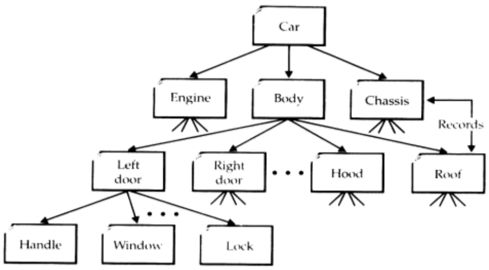


Fig. 2 : A hierarchical bill-of-materials database.

To access the data in the database, a program could perform the following tasks:

- Find a particular part by number (such as the left door)
- Move “down” to the first child (the door handle)
- Move “up” to its parent (the body)
- Move “sideways” to the next child (the right door)

Retrieving the data in a hierarchical database thus required navigating through the records, moving up, down, and sideways one record at a time.

One of the most popular hierarchical database management systems was IBM’s Information Management System (IMS), first introduced in 1968. The advantages of IMS and its hierarchical model follow.

- Simple structure: The organization of an IMS database was easy to understand. The database hierarchy paralleled that of a company organization chart or a family tree.
- Parent/child organization : An IMS database was excellent for representing parent/child relationships, such as “A is a part of B” or “A is owned by B”.

- Performance : IMS stored parent/child relationships as physical pointers from one data record to another, so that movement through the database was rapid. Because the structure was simple, IMS could place parent and child records close to one another on the disk, minimizing disk input/output.

Network Databases

The simple structure of a hierarchical database became a disadvantage when the data had a more complex structure. In an order�processing database, for example, a single order might participate in three different parent/child relationships, linking the order to the customer who placed it, the salesperson who took it, and the product ordered, as shown in figure 3. The structure of this type of data simply didn't fit the strict hierarchy of IMS.

To deal with applications such as order processing, a new network data model was developed. The network data model extended the hierarchical model by allowing a record to participate in multiple parent/child relationships, as shown in figure 4. These relationships were known as sets in the network model. In 1971, the Conference on Data Systems Languages published an official standard for network databases, which became known as the CODASYL model. IBM never developed a network DBMS of its own, choosing instead to extend IMS over the years. But during the 1970s, independent software companies rushed to embrace the network model, creating products such as Cullinet's IDMS, Cincom's Total, and the Adabas DBMS that became very popular.

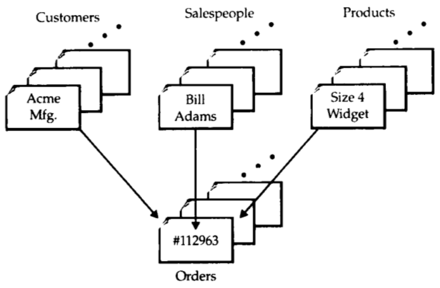


Fig. 3 : Multiple child/parent relationships

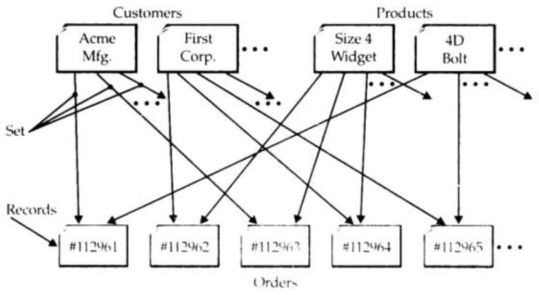


Fig. 4: A network (CODASYL) order-processing database.

For a programmer, accessing a network database was very similar to accessing a hierarchical database. An application program could do the following :

- Find a specific parent record by key (such as a customer number)
- Move down to the first child in a particular set (the first order placed by this customer)
- Move sideways from one child to the next in the set (the next order placed by the same customer)
- Move up from a child to its parent in another set (the salesperson who took the order)

Once again, the programmer had to navigate the database record by record, this time specifying which relationship to navigate s well as the direction.

Network databases had several advantages:

- Flexibility: Multiple parent/child relationships allowed a network database to represent data that did not have a simple hierarchical structure.
- Standardization : The CODASYL standard boosted the popularity of the network model, and minicomputer vendors such as Digital Equipment Corporation and Data General implemented network databases.
- Performance : Despite their greater complexity, network databases boasted performance approaching that of hierarchical databases. Sets were represented by pointers to physical data records, and on some systems, the database administrator could specify data clustering based on a set relationship.

Network databases had their disadvantages, too. Like hierarchical databases, they were very rigid. The set relationships and the structure of the records had to be specified in advance. Changing the database structure typically required rebuilding the entire database.

DATA ABSTRACTION

For the system to be usable, it must retrieve data efficiently. The need for efficiency has led designers to use complex data structures to represent data in the database. Since many database￾systems users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users interactions with the system :

- 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.
- 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. Although implementation of the simple structures at the logical level may involve complex physical￾level structures, the user of the logical level does not need to be aware of this complexity. Database administrators, who must decide what information to keep in the database, use the logical level of abstraction.
- View level : The highest level of abstraction describes only part of the entire database. 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. The view level of abstraction exists o simplify their interaction with the system. The system may provide many views for the same database.

Source: https://www.youtube.com/watch?v=6GZnGLwOVxM

Figure shows the relationship among the three levels of abstraction.

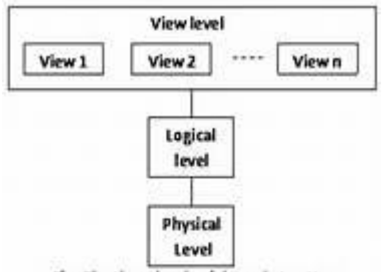


Fig: The three levels of data abstraction.

An analogy to the concept of data types in programming languages may clarify the distinction among levels of abstraction. Most high‐level programming languages support the notion of a record type. For example, in a Pascal‐like language, we may declare a record as follows :

```
Type customer = record
                              customer-id: string;
                              customer-name: string;
                              customer-street: string;
                              customer-city: string;
                       end;
```

This code defines a new record type called customer with four fields. Each field has a name and a type associated with it. A banking enterprise may have several such record types, including

- account, with fields account_number and balance
- employee, with fields employee_name and salary

At the physical level, a customer, account, or employee record can be described as a block of consecutive storage locations (for example, words or bytes). The language compiler hides this level of detail from programmers. Similarly, the database system hides many of the lowest level storage details from database programmers. Database administrators, on the other hand, may be aware of certain details of the physical organization of the data.

At the logical level, each such record is described by a type definition, as in the previous code segment, and the interrelationship of these record types is defined as well. Programmers using a programming language work at this level of abstraction. Similarly, database administrators usually work at this level of abstraction.

Finally, at the view level, computer users see a set of application programs that hide details of the data types. Similarly, at the view level, several views of the database are defined, and database users see these views. In addition to hiding details of the logical level of the database, the views also provide a security mechanism to prevent users from accessing certain parts of the database. For example, tellers in a bank see only that part of the database that has information on customer accounts; they cannot access information about salaries of employees.

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 an enterprise in an entity. An entity has a set of properties, and the values for some set of properties may uniquely identity an entity. For instance, a person may have a person-id property whose value uniquely identifies that person. Thus, the value 677–89–9011 for person-id would uniquely identify one particular person in the enterprise. Similarly, loans can be thought of as entities, and loan number L-15 at the Perry ridge branch uniquely identifies a loan entity. An entity may be concrete, such as a person or a book, or it may be abstract, such as a loan, or a holiday, or a concept.

Specialization

Consider an entity set person, with attributes name, street, and city. A person may be further classified as one of he following:

- customer
- employee

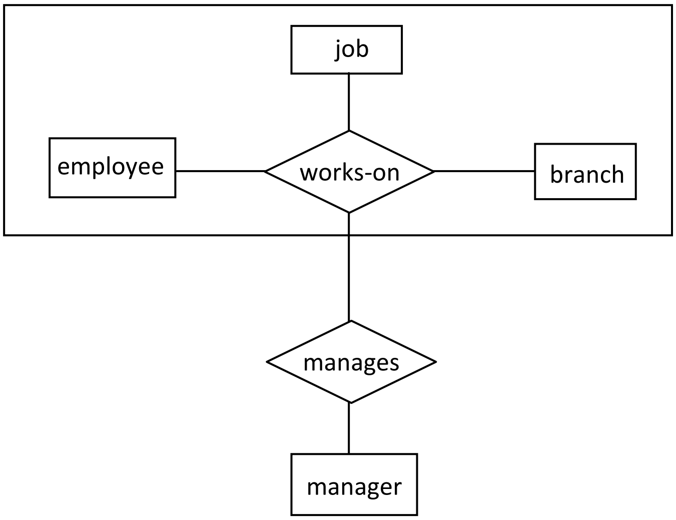


Fig. 1 : E-R diagram with aggregation.

Each of these person types is described by a set of attributes that includes all the attributes of entity set person plus possibly additional attributes. For example, customer entities may be described further by the attribute customer-id, whereas employee entities may be described further by the attributes employee-id and salary. The process of designating sub-groupings within an entity set is called specialization. The specialization of person allows us to distinguish among persons according to whether they are employees or customers.

ENTITY–RELATIONSHIP DIAGRAM

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. Such a diagram consists of the following major components :

- Rectangles, which represent entity sets
- Ellipses, which represent attributes
- Diamonds, which represent relationship sets
- Lines, which link attributes to entity sets and entity sets to relationship sets
- Double ellipses, which represent multi-valued attributes
- Dashed ellipses, which denote derived attributes
- Double lines, which indicate total participation of an entity in a relationship set.
- Double rectangles, which represent weak entity sets

Consider the entity-relationship diagram in figure 2, which consists of two entity sets, customer and loan, related through a binary relationship set borrower. The attributes associated with customer are customer-id, customer-name, customer-street, and customer-city. The attributes associated with loan are loan–number and amount. In figure 2, attributes of an entity set that are members of the primary key are underlined.

The relationship set borrower may be many-to-many, one-to-many, many-to-one, or one-to-one. 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.

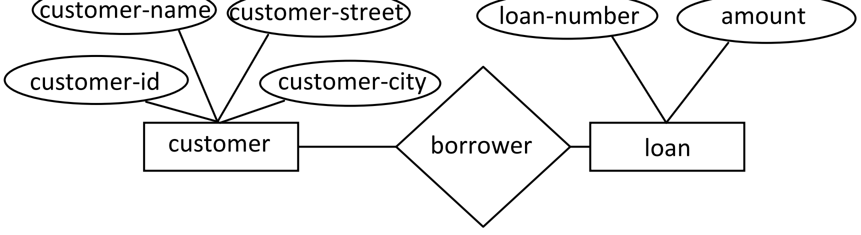


Fig. 2 : E-R diagram corresponding to customers and loans.

CONSTRAINTS

An E-R enterprise schema may define certain constraints to which the contents of a database must conform. In this section, we examine mapping cardinalities and participation constraints, which are two of the most important types of constraints.

1) Mapping Cardinalities

Mapping cardinalities, 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, although they can contribute to the description of relationship sets that involve more than two entity sets.

For a binary relationship set R between entity sets A and B, the mapping cardinality must be one of the following :

- 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. (see figure 3(a).)
- 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. (see figure 3(b))
- 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. (see figure 4(a)).
- any to may : 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. (see figure 4(b)).

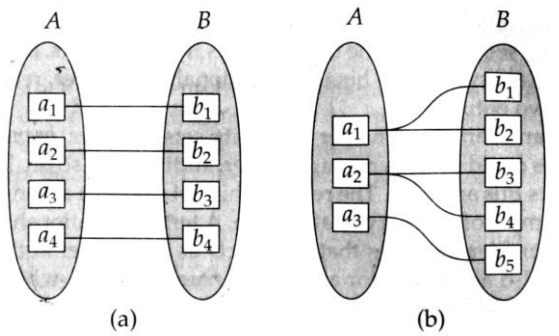


Fig.3 : Mapping cardinalities. (a) One to One (b) One to Many

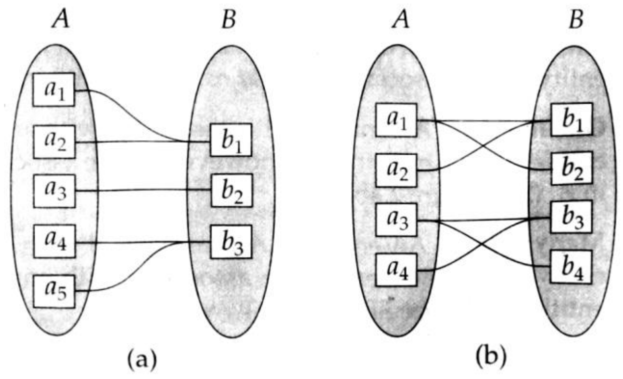


Fig. 4 : Mapping cardinalities. (a) Many to one. (b) Many to many

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 borrower relationship set. If, in a particular bank, a loan can belong to only one customer, and a customer can have several loans, then the relationship set from customer to loan is one to many. If a loan can belong to several customers (as can loans taken jointly by several business partners), the relationship set is many to many.

2) 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. For example, we expect every loan entity to be related to at least one customer through the borrower relationship. Therefore the participation of loan in the relationship set borrower is total. In contrast, an individual can be a bank customer whether or not she has a loan with the bank. Hence, it is possible that only some of the customer entities are related to the loan entity set through the borrower relationship, and the participation of customer in the borrower relationship set is therefore partial.

ERD ISSUES AND WEAK ENTITY SETS

An entity set may not have sufficient attributes to form a primary key. Such an entity set is termed a weak entity set. An entity set that has a primary key is termed a strong entity set.

As an illustration, consider the entity set payment, which has the three attributes : payment–number, payment–date, and payment–amount. Payment numbers are typically sequential numbers, starting from 1, generated separately for each loan.

Thus, although each payment entity is distinct, payments for different loans may share the same payment number. Thus, this entity set does not have a primary key; it is a weak entity set.

For a weak entity set to be meaningful, I must be associated with another entity set, called the identifying or owner entity set. Every weak entity must be associated with an identifying entity; that is, the weak entity set is said to be existence dependent on the identifying 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 payment is loan, and a relationship loan-payment that associates payment entities with their corresponding loan entities is the identifying relationship.

Although a weak entity set does not have a primary key, we nevertheless need a means of distinguishing among all those entities in the weak entity set that depend on one particular strong entity. The discriminator of a weak entity set is a set of attributes that allows this distinction to be made. For example, the discriminator of the weak entity set payment is the attribute payment–number, since, for each loan, a payment number uniquely identifies one single payment for that loan. The discriminator of a weak entity set is also called the partial key of the entity set.

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 payment, its primary key is {loan-number, payment-number}, where loan-number is the primary key of the identifying entity set, namely loan, and payment-number distinguishes payment entities within the same loan.
The identifying relationship set should have no descriptive attributes, since any required attributes can be associated with the weak entity set (see the discussion of moving relationship–set attributes to participating entity sets)

A weak entity set can participate in relationships other than the identifying relationship. For instance, the payment entity could participate in a relationship with the account entity set, identifying the account from which the payment was made. 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 form 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.

In E-R diagrams, a doubly outlined box indicates a weak entity set, and a doubly outlined diamond indicates the corresponding identifying relationship. In figure, the weak entity set payment depends on the strong entity set loan via the relationship set loan-payment.

The figure also illustrates the use of double lines to indicate total participation—the participation of the (weak) entity set payment in the relationship loan-payment is total, meaning that every payment must be related via loan-payment to some loan. Finally, the arrow from loan–payment to loan indicates that each payment is for a single loan. The discriminator of a weak entity set also is underlined, but with a dashed, rather than a solid, line.

In some cases, the database designer may choose to express a weak entity set as a multi-valued composite attribute of the owner entity set. In our example, this alternative would require that the entity set loan have a multi-valued, composite attribute payment, consisting of payment-number, payment-date, and payment-amount. A weak entity set may be more appropriately modeled as an attribute if it participates in only the identifying relationship, and if it has few attributes. Conversely, a weak-entity-set representation will more aptly model a situation where the set participates in relationships other than the identifying relationship, and where the weak entity set has several attributes.

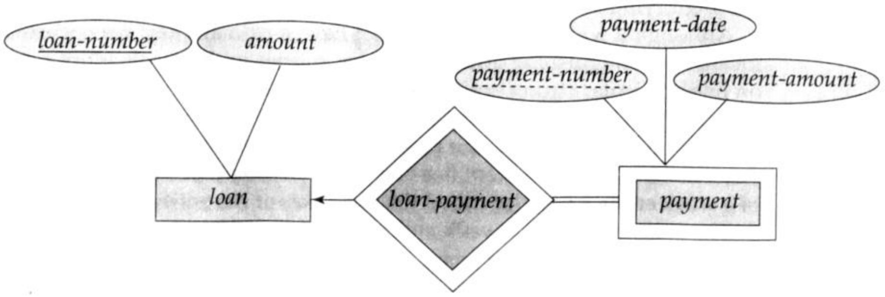


Fig. 5 : E-R diagram with a weak entity set.

As another example of an entity set that can be modeled as a weak entity set, consider offerings of a course at a university. The same course may be offered in different semesters, and within a semester there may be several sections for the same course. Thus we can create a weak entity set course-offering, existence dependent on course; different offerings of the same course are identified by a semester and a section number, which form a discriminator but not a primary key.

CODD'S 12 RULES

In his 1985 Computerworld article, Ted Codd presented 12 rules that a database must obey if it is to be considered truly relational. Codd's 12 rules, shown in the following list, have since become a semiofficial definition of a relational database. The rules come out of Codd's theoretical work on the relational model and actually represent more of an ideal goal than a definition of a relational database.

1) Information rule. All information in a relational database is represented explicitly at the logical level and in exactly one way —by values in tables.
2) Guaranteed access rule. Each and every datum (atomic value) in a relational database is guaranteed to be logically accessible by resorting to a combination of table name, primary key value, and column name.
3) Systematic treatment of NULL values. NULL values (distinct from an expty character string or a string of blank characters and distinct from zero or any other number) are supported in a fully relational DBMS for representing missing information and inapplicable information in a systematic way, independent of the data type.
4) Dynamic online catalog based on the relational mode.The database description is represented at the logical level in the same way as ordinary data, so that authorized users can apply the same relational language to its interrogation as the apply to the regular data.
5) Comprehensive data sublanguage rule.A relational system may support several languages and various modes of terminal use (for example, the fill–in–the–blanks mode). However, there must be at least one language whose statements are expressible, per some well–defined syntax, as character strings, and that is comprehensive in supporting all of the following items:
    - Data definition
    - View definition
    - Data manipulation (interactive and by program)
    - Integrity constraints
    - Authorization
    - Transaction boundaries (begin, commit, and rollback)
6) View updating rule. All views that are theoretically updateable are also updateable by the system.
7) High-level insert, update, and delete. The capability of handling a base relation or a derived relation as a single operand applies not only to the retrieval of data, but also to the insertion, update, and deletion of data.
8) Physical data independence. Application programs and terminal activities remain logically unimpaired whenever any changes are made in either storage representations or access methods.

9) Logical data independence. Application programs and terminal activities remain logically unimpaired when information-preserving changes of any kind that theoretically permit un-impairment are made to the base tables.
10) Integrity independence. Integrity constraints specific to a particular relational database must be definable in the relational data sublanguage and storable in the catalog, not in the application programs.
11) Distribution independence. A relational DBMS has distribution independence.
12) Non-subversion rule. If a relational system has a low-level (single record at a time) language, that low level cannot be used to subvert or bypass the integrity rules and constraints expressed in the higher–level relational language (multiple records at a time).

During the early 1990s, it became popular practice to compile “scorecards” for commercial DBMS products, showing how well they satisfy each of the rules. Unfortunately, the rules are subjective, so the scorecards were usually full of footnotes and qualifications, and they didn’t reveal a great deal about the products. Today, the basis of competition for database vendors tends to revolve around performance, features, the availability of development tools, the quality of vendor support, the availability of application programs that support the particular database system, and other issues, rather than conformance to Codd’s rules. Nonetheless, they are an important part of the history of the relational mode.

Rule 1 is basically the informal definition of a relational database presented at the beginning of this section.

Rule 2 stresses the importance of primary keys for locating data in the database. The table name locates the correct table, the column name finds the correct column, and the primary key value finds the row containing an individual data item of interest.

Rule 3 requires support for mission data through NULL values.

Rule 4 requires that a relational database be self-describing. In other words, the database must contain certain system tables whose columns describe the structure of the database itself.

Rule 5 mandates using a relational database language, such as SQL, although SQL is not specifically required. The language must be able to support all the central functions of a DBMS–creating a database, retrieving and entering data, implementing database security, and so on.

Rule 6 deals with views, which are virtual tables used to give various users of a database different views of its structure. It is one of the most challenging rules to implement in practice, and no commercial product fully satisfies it today.

Rule 7 stresses the set–oriented nature of a relational database. It requires that rows be treated as sets in insert, delete, and update operations. The rule is designed to prohibit implementations that support only row-at-a-time, navigational modification of the database.

Rule 8 and rule 9 insulate the user or application program from the low-level implementation of the database. They specify that specific access or storages techniques used by the DBMS, and even changes to the structure of the tables in the database, should not affect the user's ability to work with the data.

Rule 10 says that the database language should support integrity constraints that restrict the data that can be entered into the database and the database modifications that can be made. This is another rule that is not supported in most commercial DBMS products.

Rule 11 says that the database language must be able to manipulate distributed data located on other computer systems.

Finally, Rule 12 prevents "other paths" into the database that might subvert its relational structure and integrity.

Introduction of Unified Modeling Language (UML)

Entity-relationship diagrams help diagrams help model the data representation component of a software system. Data representation, however, forms only one part of an overall system design. Other components include models of user interactions with the systems, specification of functional modules of the system and their interaction, etc. The Unified Modeling Language (UML), is a proposed standard for creating specifications of various components of a software system. Some of the parts of UML are:

- Class diagram: A class diagram in similar to an E-R diagram. Later in this section we illustrate a few features of class diagrams and how they relate to E-R diagrams.

- Use case diagram: Use case diagrams show the interaction between users and the system, in particular the steps of tasks that users perform (such as withdrawing money or registering for a course).
- Activity diagram: Activity diagrams depict the flow of tasks between various components of a system.
- Implementation diagram: Implementation diagrams show the system components and their interconnections, both at the software component level and the hardware component level.

We do not attempt to provide detailed coverage of the different parts of UML here. See the bibliographic notes for references on UML. Instead we illustrate some features of UML through examples.

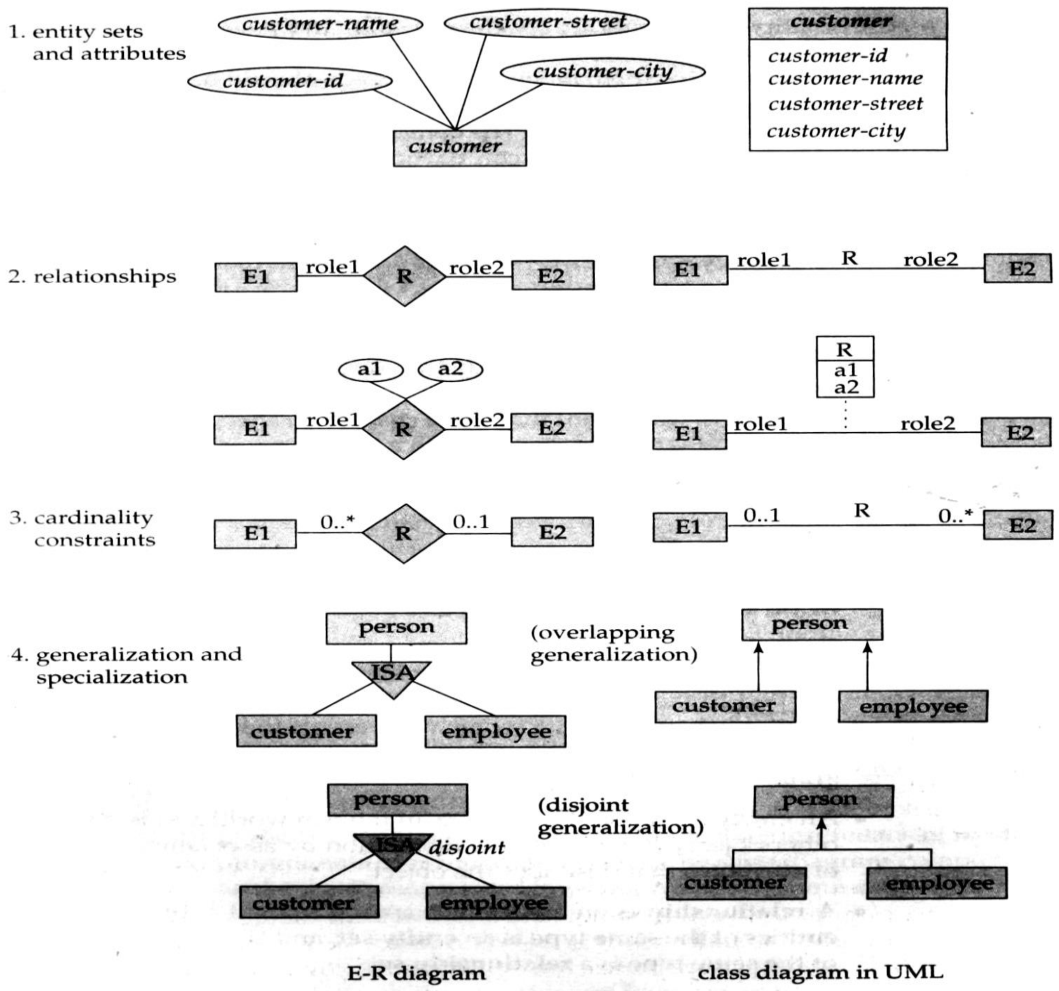
Figure 6 shows several E-R diagram constructs and their equivalent UML class diagram constructs. We describe these constructs below. UML shows entity sets as boxes and, unlike E-R, shows attributes within the box rather than as separate ellipses. UML actually models objects, whereas E-R models entities. Objects are like entities, and have attributes, but additionally provide a set of functions (called methods) that can be invoked to compute values on the basis of attributes of the objects, or to update the object itself. Class diagrams can depict methods in addition to attributes.

We represent binary relationship sets in UML by just drawing a line connecting the entity sets. We write the relationship set name adjacent to the line. We may also specify the role played by an entity set in a relationship set by writing the role name

on the line, adjacent to the entity set. Alternatively, we may write the relationship set name in a box, along with attributes of the relationship set, and connect the box by a dotted line to the line depicting the relationship set. This box can then be treated as an entity set, in the same way as an aggregation in E-R diagrams and can participate in relationships with other entity sets.

Non-binary relationships cannot be directly represented in UML – they have to be converted to binary relationships by the technique.

Cardinality constraints are specified in UML in the same way as in E-R diagrams, in



their attributes. **Fig. 6 :** Symbols used in the UML class diagram notation.

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 allows us to identify a set of attributes that suffice to distinguish entities from each other. Keys also help uniquely identify relationships, and thus distinguish relationships from each other.

Questions

1. What are the advantages of DBMS over file System?
2. Explain Database Architecture
3. Explain the purpose of Database System?
4. Write a short note on Transaction Properties?
5. What is DDL?
6. What is DML?
7. What is database system structure?
8. Explain the data abstraction in DBMS?
9. Explain importance of data model?
10. Explain hierarchical data model with its advantages & disadvantages.
11. Explain network database model with its advantages & disadvantages.
12. Explain relational database model with its advantages & disadvantages.
13. Explain Entity Relationship model with its advantages & disadvantages.
14. Define Normalization. Explain different types of Normalization with example. (Table).
15. Explain the 5 rules given by Codd.
16. Draw ER diagram for an Car insurance company that has a set of customer each of whose having one or more car. Each car associated with it zero or any number of accidents.
17. Draw ER diagram for University Database consisting of four entities Student, Department, Class, and Faculty. Student has a unique id, the student can enrol for multiple classes and has a most one major. Faculty must belong to department and faculty can teach multiple classes. Each class is taught by only faculty. Every student will get grade for the class he/she has enrolled.
18. What is UML? Explain class and activity diagram.
19. Explain advantages and disadvantages of UML Diagram.
20. Assume we have the following application that models soccer teams, the games they play, and the players in each team. In the design, we want to capture the following:

    •We have a set of teams, each team has an ID (unique identifier), name, main stadium, and to which city this team belongs.

    •Each team has many players, and each player belongs to one team. Each player has a number (unique identifier), name, DoB, start year, and shirt number that he uses.

•Teams play matches, in each match there is a host team and a guest team. The match takes place in the stadium of the host team.
•For each match we need to keep track of the following:
O The date on which the game is played
O The final result of the match
O The players participated in the match. For each player, how many goals he scored,
whether or not he took yellow card, and whether or not he took red card.
O During the match, one player may substitute another player. We want to capture this substitution and the time at which it took place.

- **Multiple Choice Questions**

1. A ------------ is a collection of related data items stored at one place.
    a. Data
    b. Database
    c. File
    d. None of these

2. ------------ is a software system that helps in the process of defining, constructing, manipulating the database.
    a. File
    b. DBMS
    c. Database
    d. None of these
3. ------- is also known as computerized record-keeping system.
    a. File
    b. DBMS
    c. Database
    d. None of these
4. --------- an integral part of the information systems of many organizations.
    a. Data
    b. Database
    c. File
    d. None of these
5. A description of data in terms of a data model is called a ________
    a. Schema
    b. Data
    c. Database
    d. File System
6. The description of a database is called -------------
    a. Database schema
    b. Data design
    c. DBMS
    d. None of these
7. The collection of information stored in the database at a particular moment is called as ________
    a. Instance
    b. Data
    c. DBMS
    d. None of these

8. ------------ is a capacity to change the conceptual schema without having any changes to external schema.
   a. Logical data independence
   b. Physical data independence
   c. Data abstraction
   d. None of these
9. ------------ is a capacity to change the internal schema without having any changes to conceptual schema.
   a. Physical data independence
   b. Logical data independence
   c. Data abstraction
   d. None of these
10. A -------------- is composed of many relations in the form of two-dimensional tables of rows and columns containing related tuples known as logical view.
   a. Relational database
   b. Database
   c. File system
   d. All of these
11. A --------- is a series of small database operations that together form a single large operation
   a. Transaction
   b. Data
   c. File
   d. Registers
12. A transaction is started by issuing a -------------- command
   a. BEGIN TRANSACTION
   b. END TRANSACTION
   c. START TRANSACTION
   d. START
13. ACID properties are Atomicity, Consistency, Durability, --------
   a. Isolation
   b. Abstraction
   c. Null
   d. None of these
14. Ability of the DBMS to recover the committed transaction updates against any kind of system failure is called as -------
   a. Atomicity,
   b. Consistency

c. Durability
d. Isolation

15. Through --------, one can ensure only accurate data is stored within the database.
a. Integrity
b. Durability
c. Isolation
d. None of these

16 . A ________ is a set of concepts that can be used to describe the structure of data in a database
a. Data model
b. DBMS
c. Data in a File
d. None of these

17. A data model can sometimes be referred to as _______
a. data structure
b. Data abstraction
c. Data Management
d. All of these

18. Data repetition and data type compatibility can be checked and removed with help of ________
a. Data Model
b. Data management
c. Data Deletion
d. None of these

19. A fundamental component of a model which has its own independent existence in real world.
a. Entity
b. Attributes
c. Relation
d. Business Rule

20. Each entity has its own properties which describes that entity, such properties are called as _______
a. Entity
b. Attributes
c. Relation
d. Business Rule

21. _______ is an association among several entities.
a. Entity
b. Attributes
c. Relation
d. Business Rule
22. One entity is associated with at most one other entity is called as ______ type of relationship.
a. One-to-one
b. One-to-many
c. Many-to-many
d. None of these
23. One entity is associated with any number of entities in other entity is called as ______ type of relationship.
a. One-to-one
b. One-to-many
c. Many-to-many
d. None of these
24. One entity is associated with any number of entities in other entity entity is called as ______ type of relationship.
a. One-to-one
b. One-to-many
c. Many-to-many
d. None of these
25. _______ may define actors and prescribe how they should behave by setting constraints and help to manage business change in the system.
a. Entity
b. Attributes
c. Relation
d. Business Rule
26. In _______ data model, data sorted hierarchically in top down or bottom up approach.
a. Hierarchical
b. Network
c. Relational
d. None of these
27. ______ model is an attempt to simplify database structure by making use of tables and columns.
a. Hierarchical

b. Network

c. Relational

d. None of these

28. _____ model uses pointers but without the need of parent child relationships.

a. Hierarchical

b. Network

c. Relational

d. None of these

29. _____ model defines data elements and relationships among various data elements for a specified system

a. Hierarchical

b. Network

c. Relational

d. Entity Relationship

30. In ________ model, data is stored in the form of objects which are structures called classes that display the data within it.

a. Hierarchical

b. Network

c. Relational

d. Object oriented database