SUBJECT: DATABASE MANAGEMENT SYSTEM (KCS-501)

UNIT-1

(Introduction to Database Management System)

To find out what database is, we have to start from data, which is the basic building block of any DBMS.

Data: Facts, figures, statistics etc. having no particular meaning (e.g. 1, ABC, 19 etc).
Record: Collection of related data items, e.g. in the above example the three data items had no meaning. But if we organize them in the following way, then they collectively represent meaningful information.

Roll	Name	Age
1	ABC	19

Table or **Relation**: Collection of related records.

Roll	Name	Age
1	ABC	19
2	DEF	22
3	XYZ	28

The columns of this relation are called **Fields**, **Attributes** or **Domains**. The rows are called **Tuples** or **Records**.
Database: Collection of related relations. Consider the following collection of tables:

T1

Roll	Name	Age
1	ABC	19
2	DEF	22
3	XYZ	28

T2

Roll	Address
1	KOL
2	DEL
3	MUM

T3

—

T4

—

Roll	Year
1	I
2	II
3	I

—

Year	Hostel
I	H1
II	H2

Age and ***Hostel*** attributes are in different tables.

A database in a DBMS could be viewed by lots of different people with different responsibilities.

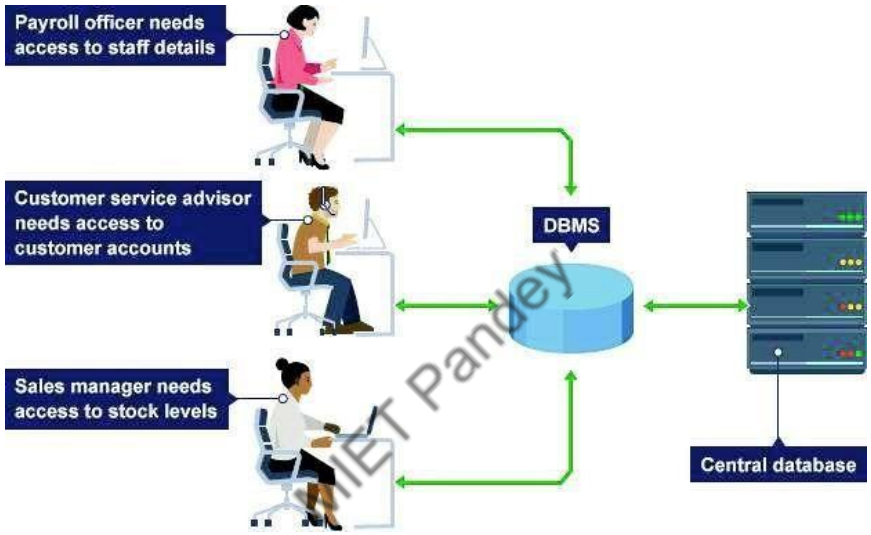


Figure 1.1: Employees are accessing Data through DBMS

For example, within a company there are different departments, as well as customers, who each need to see different kinds of data. Each employee in the company will have different levels of access to the database with their own customized **front-end** application.

In a database, data is organized strictly in row and column format. The rows are called **Tuple** or **Record**. The data items within one row may belong to different data types. On the other hand, the columns are often called **Domain** or **Attribute**. All the data items within a single attribute are of the same data type.

Database Management System

Database Management System is basically software that manages the collection of related data. It is used for storing data and retrieving the data effectively when it is needed. It also provides proper security measures for protecting the data from unauthorized access. In Database Management System the data can be fetched by SQL queries and relational algebra. It also provides mechanisms for data recovery and data backup.

A **database-management system** (DBMS) is a collection of interrelated data and a set of programs to access those data. This is a collection of related data with an implicit meaning and hence is a database. 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*. By **data,** we mean known facts that can be recorded and that have implicit meaning. Database systems are designed to manage large bodies of information. Management of data involves both defining structures for storage of information and providing mechanisms for the manipulation of information. In addition, the database system must ensure the safety of the Information stored, despite system crashes or attempts at unauthorized access. If data are to be shared among several users, the system must avoid possible anomalous results.

Database Management System (DBMS) and Its Applications:

A Database management system is a computerized record-keeping system. It is a repository or a container for collection of computerized data files. The overall purpose of DBMS is to allow he users to define, store, retrieve and update the information contained in the database on demand. Information can be anything that is of significance to an individual or organization.

Databases touch all aspects of our lives. Some of the major areas of application are as follows:

1. Banking
2. Airlines
3. Universities
4. Manufacturing and selling
5. Human resources

Database systems arose in response to early methods of computerized management of commercialdata. As an example of such methods, typical of the 1960s, consider part of a university organization that, among other data, keeps information about all instructors, students, departments, and course offerings. One way to keep the information on a computer is to store it in operating system files. To allow users to manipulate the information, the system has a number of application programs that manipulate the files, including programs to:

- Add new students, instructors, and courses
- Register students for courses and generate class rosters
- Assign grades to students, compute grade point averages (GPA), and generate transcripts

This 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. Before database management systems (DBMSs) were introduced, organizations usually stored information in such systems.

Advantages of DBMS:

Controlling of Redundancy: Data redundancy refers to the duplication of data (i.e. storing same data multiple times). In a database system, by having a centralized database and centralized control of data by the DBA the unnecessary duplication of data is avoided. It also eliminates the extra time for processing the large volume of data. It results in saving the storage space.

Improved Data Sharing: DBMS allows a user to share the data in any number of application programs.

Data Integrity: Integrity means that the data in the database is accurate. Centralized control of the data helps in permitting the administrator to define integrity constraints to the data in the database. For example: in customer database we can enforce integrity that it must accept the customer only from Noida and Meerut city.

Security: Having complete authority over the operational data, enables the DBA in ensuring that the Only mean of access to the database is through proper channels. The DBA can define authorization checks to be carried out whenever access to sensitive data is attempted.

Data Consistency: By eliminating data redundancy, we greatly reduce the opportunities for inconsistency. For example: is a customer address is stored only once, we cannot have disagreement on the stored values. Also updating data values is greatly simplified when each value is stored in one place only. Finally, we avoid the wasted storage that results from redundant data storage.

Efficient Data Access: In a database system, the data is managed by the DBMS and all access to the data is through the DBMS providing a key to effective data processing

Enforcements of Standards: With the centralized of data, DBA can establish and enforce the data standards which may include the naming conventions, data quality standards etc.

Data Independence: Ina database system, the database management system provides the interface between the application programs and the data. When changes are made to the data representation, the Meta data obtained by the DBMS is changed but the DBMS is continues to provide the data to application program in the previously used way. The DBMs handles the task of transformation of data wherever necessary.

Reduced Application Development and Maintenance Time: DBMS supports many important functions that are common to many applications, accessing data stored in the DBMS, which facilitates the quick development of application.

Disadvantages of DBMS

1) It is bit complex. Since it supports multiple functionality to give the user the best, the underlying software has become complex. The designers and developers should have thorough knowledge about the software to get the most out of it.
2) Because of its complexity and functionality, it uses large amount of memory. It also needs large memory to run efficiently.
3) DBMS system works on the centralized system, i.e.; all the users from all over the world access this database. Hence any failure of the DBMS, will impact all the users.
4) DBMS is generalized software, i.e.; it is written work on the entire systems rather specific one. Hence some of the application will run slow.

File System:

The file system is basically a way of arranging the files in a storage medium like a hard disk. The file system organizes the files and helps in the retrieval of files when they are required. File systems consist of different files which are grouped into directories. The directories further contain other folders and files. The file system performs basic operations like management, file naming, giving access rules, etc.

Difference between File System and DBMS

FILE SYSTEM	DBMS
Used to manage and organize the files stored in the hard disk of the computer	A software to store and retrieve the user's data
Redundant data is present	No presence of redundant data
Query processing is not so efficient	Query processing is efficient
Data consistency is low	Due to the process of normalization, the data consistency is high
Less complex, does not support complicated transactions	More complexity in managing the data, easier to implement complicated transactions
Less security	Supports more security mechanisms
Less expensive in comparison to DBMS	Higher cost than the File system
Does not support crash recovery	Crash recovery mechanism is highly supported

View of Data

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

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-system users are not computer trained, developers hide the complexity from users through several levels of abstraction, to simplify users' interactions with the system:

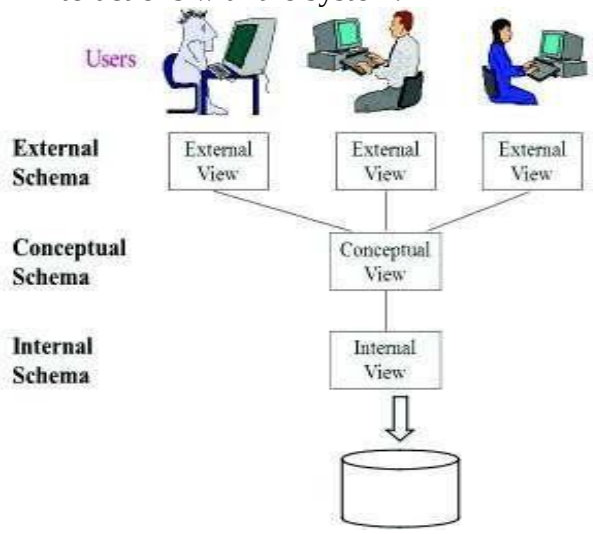


Figure 1.2: Levels of Abstraction in a DBMS

• **Physical level (or Internal View / Schema)**: 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 (or Conceptual View / Schema)**: 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. This is referred to as **physical data independence**.

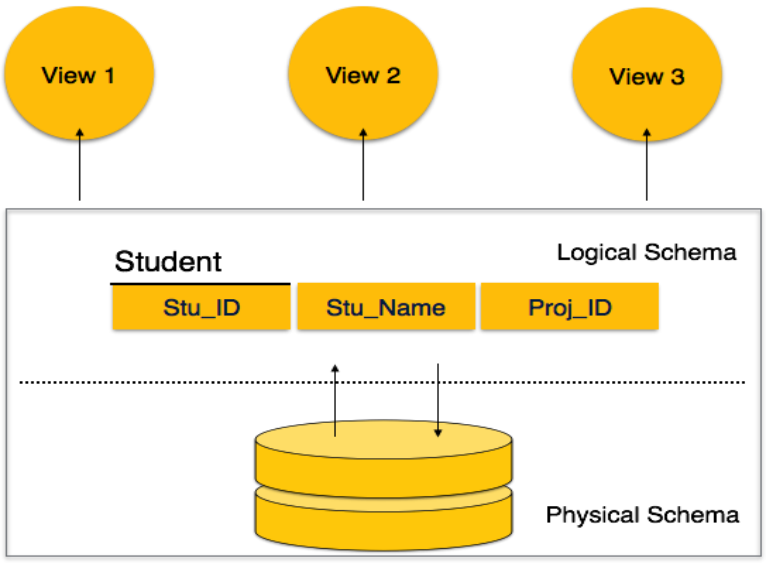
• **View level (or External View / Schema):** The highest level of abstraction describes only part of the entire database. Even though the logical level uses simpler structures, complexity remains because of the variety of information stored in a large database. Many users of the database system do not need all this information; instead, they need to access only a part of the database. The view level of abstraction exists to simplify their interaction with the system. The system may provide many views for the same database.

Instances and Schemas

Databases change over time as information is inserted and deleted. The collection of information stored in the database at a particular moment is called an **instance** of the database. The overall design of the database is called the database **schema**. Schemas are changed infrequently, if at all. The concept of database schemas and instances can be understood by analogy to a program written in a programming language.

Each variable has a particular value at a given instant. The values of the variables in a program at a point in time correspond to an *instance* of a database schema. Database systems have several schemas, partitioned according to the levels of abstraction. The **physical schema** describes the database design at the physical level, while the **logical schema** describes the database design at the logical level. A database may also have several schemas at the view level, sometimes called **subschema**, which describe different views of the database. Of these, the logical schema is by far the most important, in terms of its effect on application programs, since programmers construct applications by using the logical schema. Application programs are said to exhibit **physical data independence** if they do not depend on the physical schema, and thus need not be rewritten if the physical schema changes.

A database schema is the structure that represents the logical view of the entire database. It defines how the data is organized and how the relations among them are associated. It formulates all the constraints that are to be applied on the data. A database schema defines its entities and the relationship among them. It contains a descriptive detail of the database, which can be depicted by means of schema diagrams. It's the database designers who design the schema to help programmers understand the database and make it useful.



A database schema can be divided broadly into two categories −

- **Physical Database Schema** − this schema pertains to the actual storage of data and its form of storage like files, indices, etc. It defines how the data will be stored in a secondary storage.
- **Logical Database Schema** − this schema defines all the logical constraints that need to be applied on the data stored. It defines tables, views, and integrity constraints.

Referential integrity constraints other than foreign key constraints are not shown explicitly in schemadiagrams. We will study a different diagrammatic representation called the entity-relationship diagram.

Data Models

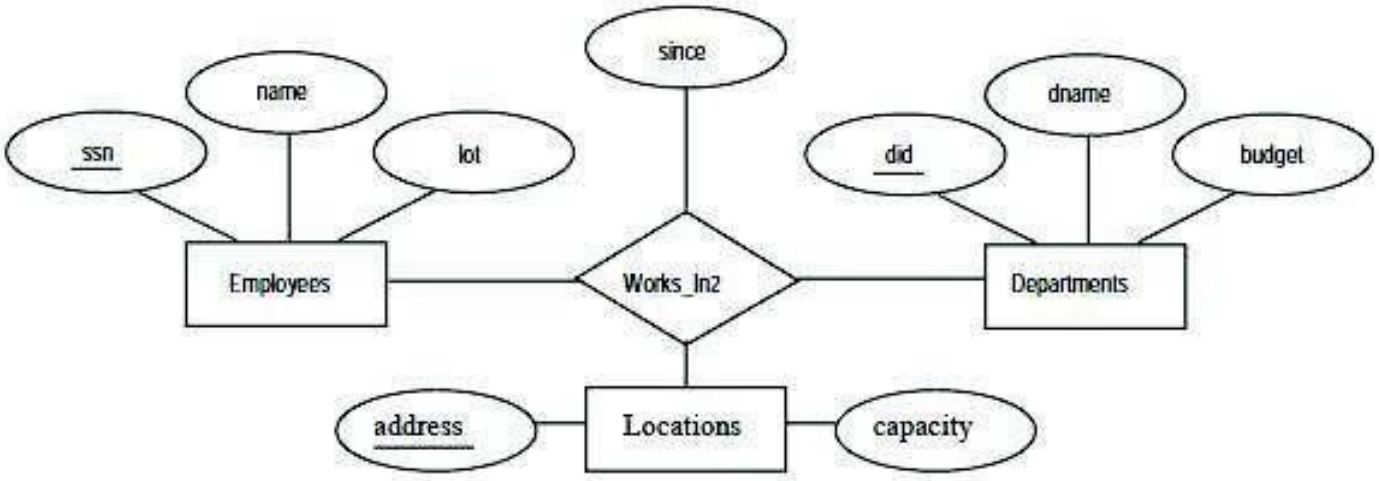
Underlying the structure of a database is the **data model**: a collection of conceptual tools for describing data, data relationships, data semantics, and consistency constraints.

The data models can be classified into four different categories:
• **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.

Entity-Relationship Model. The entity-relationship (E-R) data model uses a collection of basic objects, called *entities,* and *relationships* among these objects.

Suppose that each department has offices in several locations and we want to record the locations at which each employee works. The ER diagram for this variant of Works In, which we call Works In2

Example -



Conceptual Database Design - Entity Relationship(ER) Modeling:

Database Design Techniques
1. ER Modeling (Top down Approach)
2. Normalization (Bottom Up approach)

What is ER Modeling?

A graphical technique for understanding and organizing the data independent of the actual database implementation We need to be familiar with the following terms to go further.

Entity

Anything that has an independent existence and about which we collect data. It is also known as entity type.

In ER modeling, notation for entity is given below.

Entity instance

Entity instance is a particular member of the entity type.

Example for entity instance: A particular employee

Regular Entity

An entity which has its own key attribute is a regular entity.

Example for regular entity: Employee.

Weak entity

An entity which depends on other entity for its existence and doesn't have any key attribute of its own is a weak entity.

Example for a weak entity: In a parent/child relationship, a parent is considered as a strong entity and the child is a weak entity.

In ER modeling, notation for weak entity is given below.

Attributes

Properties/characteristics which describe entities are called attributes.

In ER modeling, notation for attribute is given below.

Domain of Attributes

The set of possible values that an attribute can take is called the domain of the attribute. For example, the attribute day may take any value from the set {Monday, Tuesday ... Friday}. Hence this set can be termed as the domain of the attribute day.

Key attribute

The attribute (or combination of attributes) which is unique for every entity instance is called key attribute. e.g. the employee_id of an employee, pan_card_number of a person etc. If the key attributeconsists of two or more attributes in combination; it is called a composite key.

In ER modeling, notation for key attribute is given below.

Simple attribute

If an attribute cannot be divided into simpler components, it is a simple attribute.

Example for simple attribute: employee_id of an employee.

Composite attribute

If an attribute can be split into components, it is called a composite attribute.

Example for composite attribute: Name of the employee which can be split into First_name, Middle_name, and Last_name.

Single valued Attributes

If an attribute can take only a single value for each entity instance, it is a single valued attribute.

Example for single valued attribute: age of a student. It can take only one value for a particular student.

Multi-valued Attributes

If an attribute can take more than one value for each entity instance, it is a multi-valued attribute. Multi-valued.

Example for multi valued attribute: telephone number of an employee, a particular employee may have multiple telephone numbers.

In ER modeling, notation for multi-valued attribute is given below.

Stored Attribute

An attribute which need to be stored permanently is a stored attribute

Example for stored attribute: name of a student

Derived Attribute

An attribute which can be calculated or derived based on other attributes is a derived attribute.

Example for derived attribute: age of employee which can be calculated from date of birth and current date.

In ER modeling, notation for derived attribute is given below.

Relationships

Associations between entities are called relationships

Example: An employee works for an organization. Here "works for" is a relation between theentities employee and organization.

In ER modeling, notation for relationship is given below.

However in ER Modeling, to connect a weak Entity with others, you should use a weak relationship notation as given below

Degree of a Relationship

Degree of a relationship is the number of entity types involved. The n-ary relationship is the general form for degree n. Special cases are unary, binary, and ternary, where the degree is 1, 2, and 3, respectively.

Example for unary relationship: An employee ia a manager of another employee Example for binary relationship: An employee works-for department. Example for ternary relationship: customer purchase item from a shop keeper **Cardinality of a Relationship**

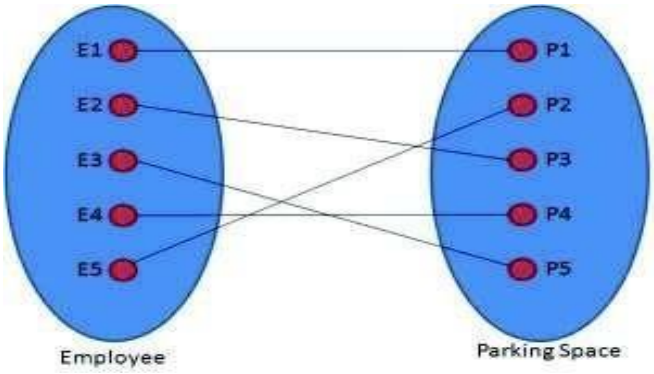
Relationship cardinalities specify how many of each entity type is allowed. Relationships can have four possible connectivity's as given below.

1. One to one (1:1) relationship
2. One to many (1:N) relationship
3. Many to one (M:1) relationship
4. Many to many (M:N) relationship

The minimum and maximum values of this connectivity is called the cardinality of the relationship

Example for Cardinality – One-to-One (1:1)

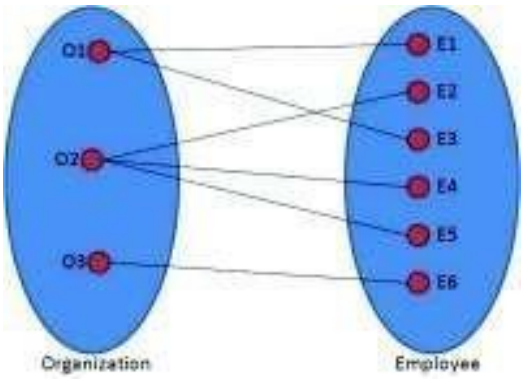
Employee is assigned with a parking space.



One employee is assigned with only one parking space and one parking space is assigned to only one employee. Hence it is a 1:1 relationship and cardinality is One-To-One (1:1)
In ER modeling, this can be mentioned using notations as given below

Example for Cardinality – One-to-Many (1: N)

Organization has employees

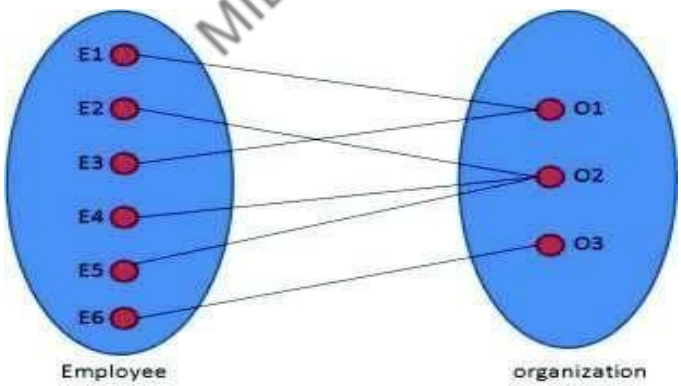


One organization can have many employees, but one employee works in only one organization. Hence it is a 1: N relationship and cardinality is One-To-Many (1: N)

In ER modeling, this can be mentioned using notations as given below

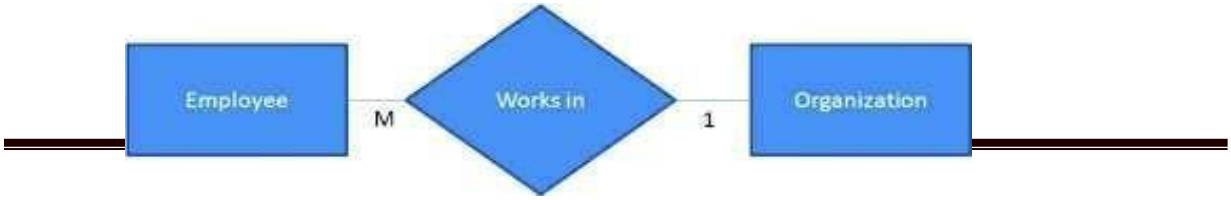
Example for Cardinality – Many-to-One (M: 1)

It is the reverse of the One to much relationship. Employee works in organization



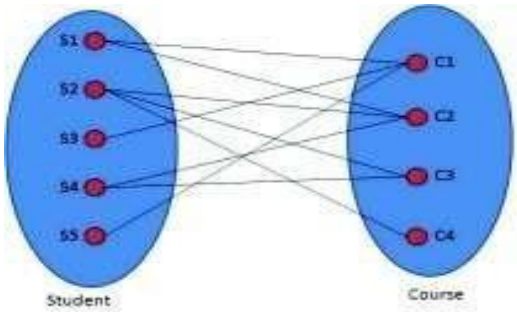
One employee works in only one organization But one organization can have many employees. Hence it is a M: 1 relationship and cardinality is Many-to-One (M :1)

In ER modeling, this can be mentioned using notations as given below.



Cardinality – Many-to-Many (M: N)

Students enrolls for courses



One student can enroll for many courses and one course can be enrolled by many students. Hence it is an M: N relationship and cardinality is Many-to-Many (M: N)

In ER modeling, this can be mentioned using notations as given below

Relationship Participation

1. Total

In total participation, every entity instance will be connected through the relationship to another instance of the other participating entity types

2. Partial

Example for relationship participation

Consider the relationship - Employee is head of the department.

Here all employees will not be the head of the department. Only one employee will be the head of the department. In other words, only few instances of employee entity participate in the above relationship. So employee entity's participation is partial in the said relationship.

Advantages of ER Modeling (Merits and Demerits of ER Modeling)Advantages

1. ER Modeling is simple and easily understandable. It is represented in business user's language andit can be understood by non-technical specialist.
2. Intuitive and helps in Physical Database creation.
3. Can be generalized and specialized based on needs.
4. Can help in database design.
5. Gives a higher level description of the system.

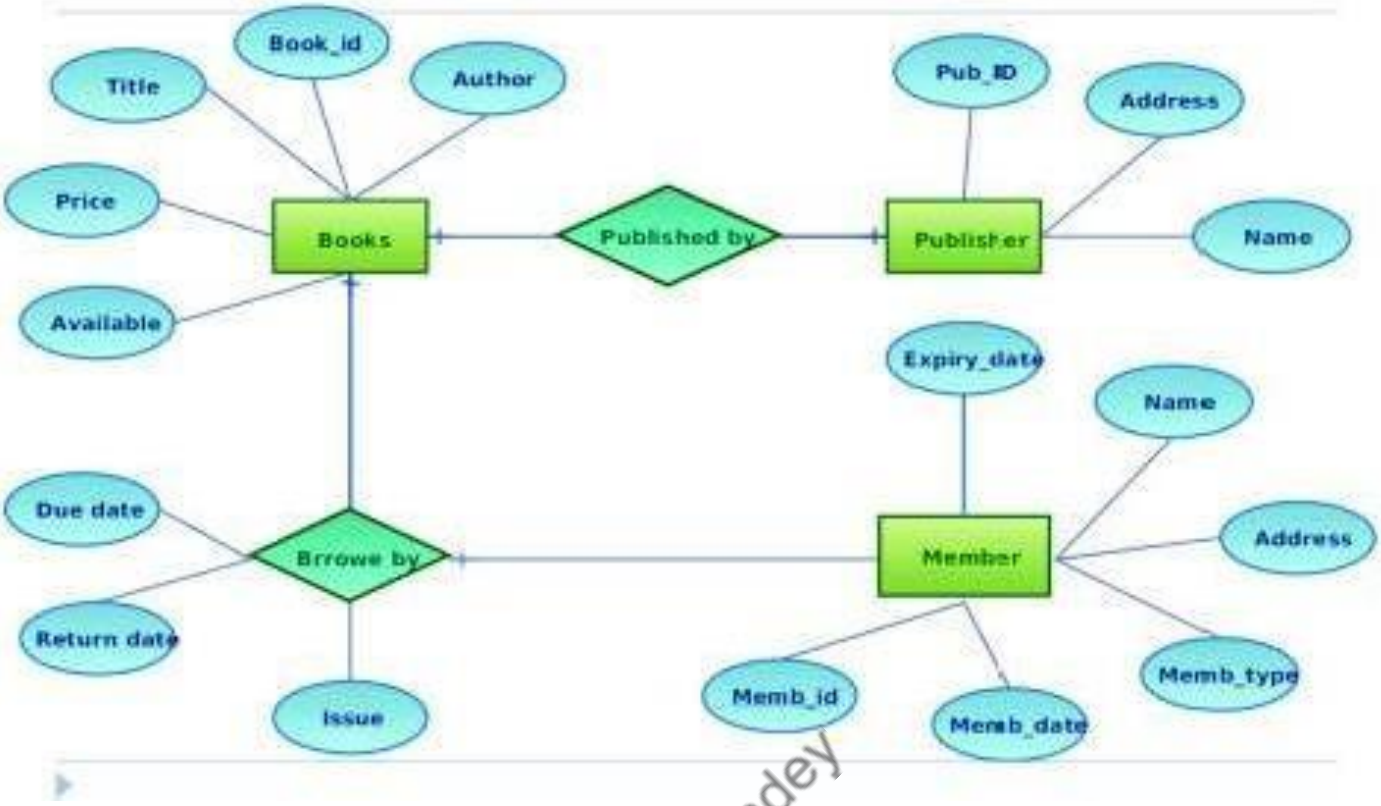
Relational Model

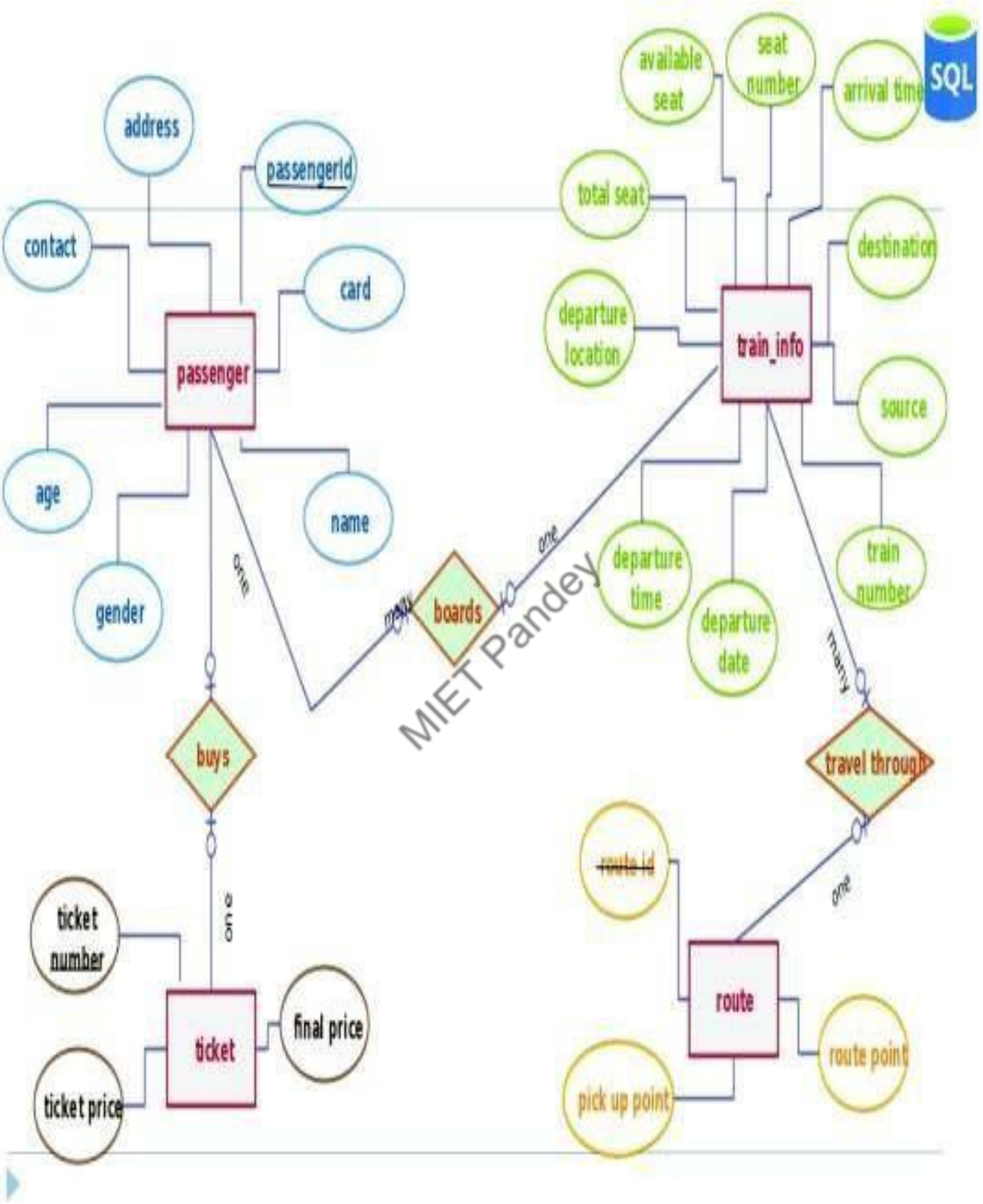
The relational model is today the primary data model for commercial data processing applications. It attained its primary position because of its simplicity, which eases the job of the programmer, compared to earlier data models such as the network model or the hierarchical model.

Structure of Relational Databases:

A relational database consists of a collection of **tables**, each of which is assigned a unique name. Which stores information about instructors? The table has four column headers: *ID*, *name*, *dept name*, and *salary*. Each row of this table records information about an instructor, consisting of the instructor's *ID*, *name*, *dept name*, and *salary*.

E-R Diagram for Library Management System





E R Diagram-(Railway Booking System)

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 with notions of encapsulation, methods (functions), and object identity.

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. This is in contrast to the data models mentioned earlier, where every data item of a particular type must have the same set of attributes. The **Extensible Markup Language (XML)** is widely used to represent semi-structured data.

Historically, the **network data model** and the **hierarchical data model** preceded the relational data model.
These models were tied closely to the underlying implementation, and complicated the 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-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
- Insertion of new information into the database
- Deletion of information from the database
- Modification of information stored in the database

There are basically two types:
• **Procedural DMLs** require a user to specify *what* data are needed and *how* to get those data.
• **Declarative DMLs** (also referred to as **nonprocedural DMLs**) require a user to specify *what* data are needed *without* specifying how to get those data.

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.

Data-Definition Language (DDL)

We specify a database schema by a set of definitions expressed by a special language called a **data-definition language** (**DDL**). The DDL is also used to specify additional properties of the data.

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. Domain constraints are the most elementary form of integrity constraint. They are tested easily by the system whenever a new data item is entered into the database.

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 department listed for each course must be one that actually exists. More precisely, the *dept name* value in a *course* record must appear in the *dept name* attribute of some record of the *department* relation.

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 being: **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.
The DDL, just like any other programming language, gets as input some instructions (statements) and generates some output. The output of the DDL is placed in the **data dictionary**, which contains **metadata**—that is, data about data.

Data Dictionary

We can define a data dictionary as a DBMS component that stores the definition of data characteristics and relationships. You may recall that such "data about data" were labeled metadata. The DBMS data dictionary provides the DBMS with its self describing characteristic. In effect, the data dictionary resembles and X-ray of the company's entire data set, and is a crucial element in the data administration function.
For example, the data dictionary typically stores descriptions of all:

- Data elements that are define in all tables of all databases. Specifically the data dictionary stores the name, data types, display formats, internal storage formats, and validation rules. The data dictionary tells where an element is used, by whom it is used and so on.
- Tables define in all databases. For example, the data dictionary is likely to store the name of the table creator, the date of creation access authorizations, the number of columns, and so on.
- Indexes define for each database tables. For each index the DBMS stores at least the index name the attributes used, the location, specific index characteristics and the creation date.
- Define databases: who created each database, the date of creation where the database is located, who the DBA is and so on.
- End users and The Administrators of the data base
- Programs that access the database including screen formats, report formats application formats, SQL queries and so on.
- Access authorization for all users of all databases.
- Relationships among data elements which elements are involved: whether the relationship is mandatory or optional, the connectivity and cardinality and so on.

Database Administrators (DBA) and Database Users
A primary goal of a database system is to retrieve information from and store new information in the database.

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.

Naive users are unsophisticated users who interact with the system by invoking one of the application programs that have been written previously. For example, a bank teller who needs to transfer $50 from account *A* to account *B* invokes a program called *transfer*.

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 without writing a program.

Sophisticated users interact with the system without writing programs. Instead, they form their requests in a database query language. They submit each such query to a **query processor**, whose function is to break down DML statements into instructions that the storage manager understands. Analysts who submit queries to explore data in the database fall in this category.

Online analytical processing (OLAP) tools simplify analysts' tasks by letting them view summaries of data in different ways. For instance, an analyst can see total sales by region (for example, North, South, East, and West), or by product, or by a combination of region and product (that is, total sales of each product in each region).

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.

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. The query processor is important because it helps the database system simplify and facilitate access to data.

2-Tier Architecture

A 2 Tier Architecture in DBMS is a Database architecture where the presentation layer runs on a client (PC, Mobile, Tablet, etc.), and data is stored on a server called the second tier. Two tier architecture provides added security to the DBMS as it is not exposed to the end-user directly. It also provides direct and faster communication.

3-Tier Architecture

A 3 Tier Architecture in DBMS is the most popular client server architecture in DBMS in which the development and maintenance of functional processes, logic, and data access, data storage, and user interface is done independently as separate modules. Three Tier architecture contains a presentation layer, an application layer, and a database server.

3-Tier database Architecture design is an extension of the 2-tier client-server architecture. 3-tier architecture has the following layers:

1. Presentation layer (your PC, Tablet, Mobile, etc.)
2. Application layer (server)
3. Database Server

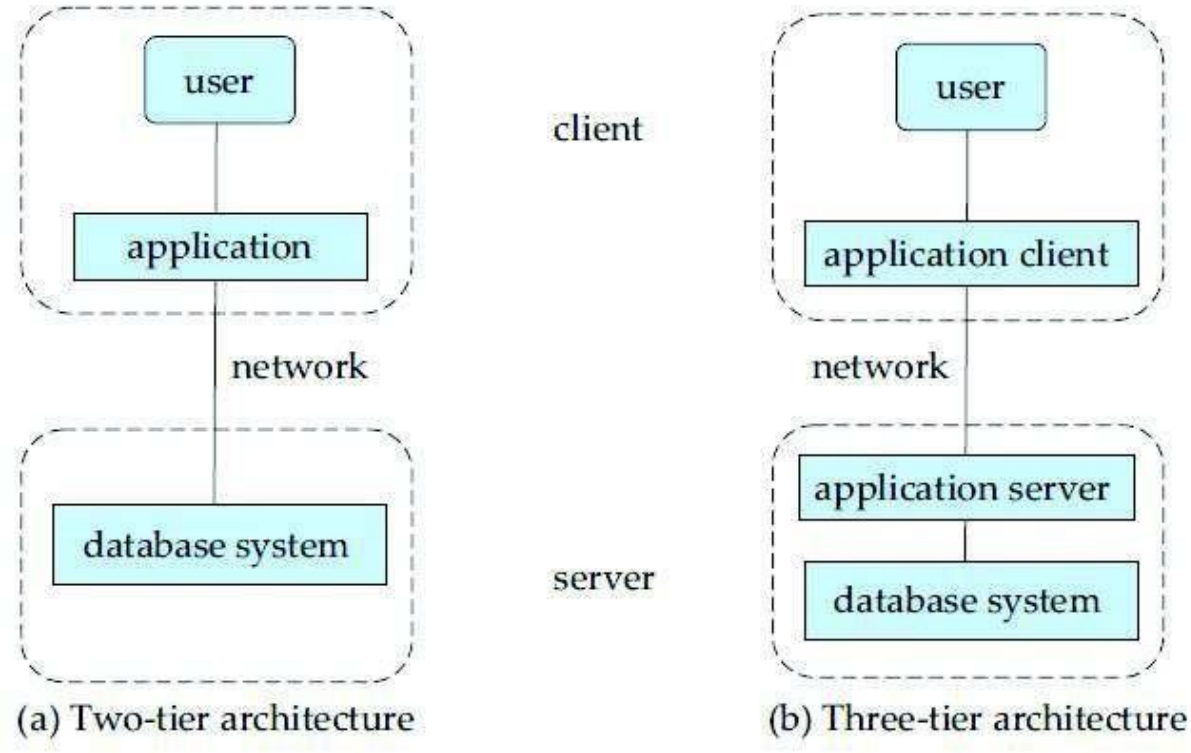


Figure 1.4: Two-tier and three-tier architectures.

Keys

Super key

A **super key** is a set of one or more attributes that, taken collectively, allow us to identify uniquely a tuple in the relation. For example, the *ID* attribute of the relation *instructor* is sufficient to distinguish one instructor tuple from another. Thus, *ID* is a super key. The *name* attribute of *instructor*, on the other hand, is not a super key, because several instructors might have the same name.
A super key may contain extraneous attributes. For example, the combination of *ID* and *name* is a super key for the relation *instructor*. If *K* is a super key, then so is any superset of *K*. We are often interested in super keys for which no proper subset is a super key. Such minimal super keys are called

Candidate keys

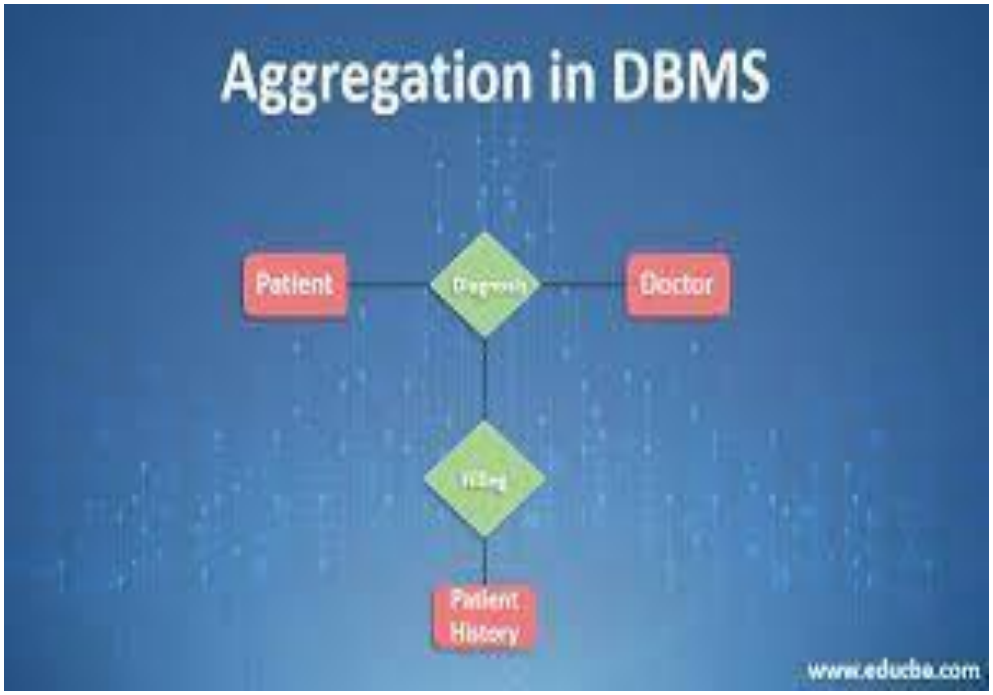
A **primary key** is the column or columns that contain values that uniquely identify each row in a table. A database table must have a primary key for Optimize to insert, update, restore, or delete data from a database table.

Foreign Key

A **Foreign Key** is a field (or collection of fields) in one table that refers to the PRIMARY KEY in another table. The table with the foreign key is called the child table, and the table with the primary key is called the referenced or parent table.

AGGREGATION

Aggregation refers to the process by which entities are combined to form a single meaningful entity. The specific entities are combined because they do not make sense on their own. To establish a single entity, aggregation creates a relationship that combines these entities.



GENERALIZATION

Generalization is a bottom-up approach in which two lower level entities combine to form a higher level entity. In generalization, the higher level entity can also combine with other lower level entities to make further higher level entity.