BCA IIIRD SEMESTER

DBMS NOTES UNIT WISE

UNIT 1

Objectives

At the end of this chapter the reader will be able to:

- Distinguish between data and information and Knowledge
- Distinguish between file processing system and DBMS
- Describe DBMS its advantages and disadvantages
- Describe Database users including data base administrator
- Describe data models, schemas and instances.
- Describe DBMS Architecture & Data Independence
- Describe Data Languages

Introduction

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.

Data Processing Vs. Data Management Systems

Although Data Processing and Data Management Systems both refer to functions that take raw data and transform it into usable information, the usage of the terms is very different. **Data Processing** is the term generally used to describe what was done by large mainframe computers from the late 1940's until the early 1980's (and which continues to be done in most large organizations to a greater or lesser extent even today): large volumes of raw transaction data fed into programs that update a master file, with fixed-format reports written to paper.

The term **Data Management Systems** refers to an expansion of this concept, where the raw data, previously copied manually from paper to punched cards, and later into data-entry terminals, is now fed into the system from a variety of sources, including ATMs, EFT, and direct customer entry through the Internet. The master file concept has been largely displaced by database management systems, and static reporting replaced or augmented by ad-hoc reporting and direct inquiry, including downloading of data by customers. The ubiquity of the Internet and the Personal Computer have been the driving force in the transformation of Data Processing to the more global concept of Data Management Systems.

File Oriented Approach

The earliest business computer systems were used to process business records and produce information. They were generally faster and more accurate than equivalent manual systems. These systems stored groups of records in separate files, and so they were called **file processing systems.** In a typical file processing systems, each department has its own files, designed specifically for those applications. The department itself working with the data processing staff, sets policies or standards for the format and maintenance of its files.

Programs are dependent on the files and vice-versa; that is, when the physical format of the file is changed, the program has also to be changed. Although the traditional file oriented approach to information processing is still widely used, it does have some very important disadvantages.

Characteristics

Traditionally data was organized in file formats. DBMS was all new concepts then and all the research was done to make it to overcome all the deficiencies in traditional style of data management. Modern DBMS has the following characteristics:

- Real-world entity: Modern DBMS are more realistic and uses real world entities to design its architecture. It uses the behavior and attributes too. For example, a school database may use student as entity and their age as their attribute.

- Relation-based tables: DBMS allows entities and relations among them to form as tables. This eases the concept of data saving. A user can understand the architecture of database just by looking at table names etc.

- Isolation of data and application: A database system is entirely different than its data. Where database is said to active entity, data is said to be passive one on which the database works and organizes. DBMS also stores metadata which is data about data, to ease its own process.

- Less redundancy: DBMS follows rules of normalization, which splits a relation when any of its attributes is having redundancy in values. Following normalization, which itself is a mathematically rich and scientific process, make the entire database to contain as less redundancy as possible.

- Consistency: DBMS always enjoy the state on consistency where the previous form of data storing applications like file processing does not guarantee this. Consistency is a state where every relation in database remains consistent. There exist methods and techniques, which can detect attempt of leaving database in inconsistent state.

- Query Language: DBMS is equipped with query language, which makes it more efficient to retrieve and manipulate data. A user can apply as many and different filtering options, as he or she wants. Traditionally it was not possible where file-processing system was used.

- ACID Properties: DBMS follows the concepts for ACID properties, which stands for Atomicity, Consistency, Isolation and Durability. These concepts are applied on transactions, which manipulate data in database. ACID properties maintains database in healthy state in multi-transactional environment and in case of failure.

- Multiuser and Concurrent Access: DBMS support multi-user environment and allows them to access and manipulate data in parallel. Though there are restrictions on transactions when they attempt to handle same data item, but users are always unaware of them.

- Multiple views: DBMS offers multiples views for different users. A user who is in sales department will have a different view of database than a person working in production department. This enables user to have a concentrate view of database according to their requirements.

- Security: Features like multiple views offers security at some extent where users are unable to access data of other users and departments. DBMS offers methods to impose constraints while entering data into database and retrieving data at later stage. DBMS offers many different levels of security features, which enables multiple users to have different view with different features.

Concurrent Use

A database system allows several users to access the database concurrently. Answering different questions from different users with the same (base) data is a central aspect of an information system. Such concurrent use of data increases the economy of a system.

Structured and Described Data

A fundamental feature of the database approach is that the database systems do not only contain the data but also the complete definition and description of these data. These descriptions are basically details about the extent, the structure, the type and the format of all data and, additionally, the relationship between the data. This kind of stored data is called metadata ("data about data").

Separation of Data and Applications

As described in the feature structured data the structure of a database is described through *metadata* which is also stored in the database. An application software does not need any knowledge about the physical data storage like encoding, format, storage place, etc. It only communicates with the management system f a database (DBMS) via a standardized interface with the help of a standardized language like SQL..

Data Integrity
Data integrity is a byword for the quality and the reliability of the data of a database system. In a broader sense data integrity includes also the protection of the database from unauthorized access (confidentiality) and un authorized changes..
Transactions
A transaction is a bundle of actions which are done within a database to bring it from one consistent state to a new consistent state..
Data Persistence
Data persistence means that in a DBMS all data is maintained as long as it is not deleted explicitly. The life span of data needs to be determined directly or indirectly be the user and must not be dependent on system features. Additionally data once stored in a database must not be lost. Changes of a database which are done by a transaction are persistent. When a transaction is finished even a system crash cannot put the data in danger.

Advantages and Disadvantages of a DBMS
Using a DBMS to manage data has many advantages:

Data independence: Application programs should be as independent as possible from details of data representation and storage. The DBMS can provide an abstract view of the data to insulate application code from such details.
Efficient data access: A DBMS utilizes a variety of sophisticated techniques to store and retrieve data efficiently. This feature is especially important if the data is stored on external storage devices.
Data integrity and security: If data is always accessed through the DBMS, the DBMS can enforce integrity constraints on the data. For example, before inserting salary information for an employee, the DBMS can check that the department budget is not exceeded. Also, the DBMS can enforce *access controls* that govern what data is visible to different classes of users.
Data administration: When several users share the data, centralizing the administration of data can offer significant improvements. Experienced professionals, who understand the nature of the data being managed, and how different groups of users use it, can be responsible for organizing the data representation to minimize redundancy and fine-tuning the storage of the data to make retrieval efficient.
Concurrent access and crash recovery: A DBMS schedules concurrent accesses to the data in such a manner that users can think of the data as being accessed by only one user at a time. Further, the DBMS protects users from the effects of system failures.
Reduced application development time: Clearly, the DBMS supports many important functions that are common to many applications accessing data stored in the DBMS. This, in conjunction with the high-level interface to the data, facilitates quick development of applications. Such applications are also likely to be more robust than applications developed

from scratch because many important tasks are handled by the DBMS instead of being implemented by the application.

Disadvantages of a DBMS

Danger of a Overkill: For small and simple applications for single users a database system is often not advisable.

Complexity: A database system creates additional complexity and requirements. The supply and operation of a database management system with several users and databases is quite costly and demanding.

Qualified Personnel: The professional operation of a database system requires appropriately trained staff. Without a qualified database administrator nothing will work for long.

Costs: Through the use of a database system new costs are generated for the system itselfs but also for additional hardware and the more complex handling of the system.

Lower Efficiency: A database system is a multi-use software which is often less efficient than specialized software which is produced and optimized exactly for one problem.

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. A database schema corresponds to the variable declarations (along with associated type definitions) in a program. 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. Therefore Database schema skeleton structure of and it represents the logical view of entire database. It tells about how the data is organized and how relation among them is associated. It formulates all database constraints that would be put on data in relations, which resides in database. A database schema defines its entities and the relationship among them. Database schema is a descriptive detail of the database, which can be depicted by means of schema diagrams. All these activities are done by database designer to help programmers in order to give some ease of understanding all aspect of database.

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 **sub schemas**, that 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. The physical schema is hidden beneath the logical schema, and can usually be changed easily without affecting application programs. 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.

.
Database schema skeleton structure of and it represents the logical view of entire database. It tells about how the data is organized and how relation among them is associated. It formulates all database constraints that would be put on data in relations, which resides in database.

DBMS 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.
To illustrate the concept of a data model, we outline two data models in this section: the entity-relationship model and the relational model. Both provide a way to describe the design of a database at the logical level. Data model tells how the logical structure of a database is modeled. Data Models are fundamental entities to introduce abstraction in DBMS. Data models define how data is connected to each other and how it will be processed and stored inside the system. The very first data model could be flat data-models where all the data used to be kept in same plane. Because earlier data models were not so scientific they were prone to introduce lots of duplication and update anomalies.

Other Data Models

The **object-oriented data model** is another data model that has seen increasing attention.
The object-oriented model can be seen as extending the E-R model with notions object-oriented data model. The **object-relational data model** combines features of the object-oriented data model and relational data model. Semi structured data models permit the specification of data where individual data items of the same type may have different sets of attributes. This is in contrast with 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, two other data models, 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 little used now, except in old database code that is still in service in some places.

DBMS Architecture

Three important characteristics of the database approach are (1) insulation of programs and data (program-data and program-operation independence); (2) support of multiple user views; and (3) use of a catalog to store the database description (schema). In this section we specify architecture for database systems, called the **three-schema architecture**, which was proposed to help achieve and visualize these characteristics.

The Three-Schema Architecture

The goal of the three-schema architecture, illustrated in Figure is to separate the user applications and the physical database. In this architecture, schemas can be defined at the following three levels:

The **internal level** has an **internal schema,** which describes the physical storage structure of the database. The internal schema uses a physical data model and describes the complete details of data storage and access paths for the database.

The **conceptual level** has a **conceptual schema,** which describes the structure of the whole database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data types, relationships, user operations, and constraints. A high-level data model or an implementation data model can be used at this level.
The **external** or **view level** includes a number of **external schemas** or **user views.** Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high-level data model or an implementation data model can be used at this level.

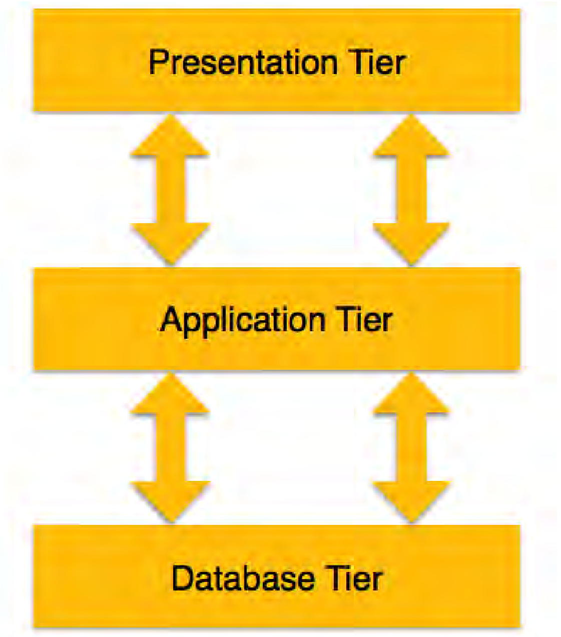
The three-schema architecture is a convenient tool for the user to visualize the schema levels in a database system. Most DBMSs do not separate the three levels completely, but support the three-schema architecture to some extent. Some DBMSs may include physical-level details in the conceptual schema. In most DBMSs that support user views,
external schemas are specified in the same data model that describes the conceptual-level information. Some DBMSs allow different data models to be used at the conceptual and external levels.
Notice that the three schemas are only *descriptions* of data; the only data that *actually* exists is at the physical level. In a DBMS based on the three-schema architecture, each user group refers only to its own external schema. Hence, the DBMS must transform a request specified on an external schema into a request against the conceptual schema, and then into a request on the internal schema for processing over the stored database. If the request is a database retrieval, the data extracted from the stored database must be reformatted to match the user‘s external view. The processes of transforming requests and results between levels are called **mappings.** These mappings may be time-consuming, so some DBMSs—especially those that are meant to support small databases—do not support external views. Even in such systems, however, a certain amount of mapping is necessary to transform requests between the conceptual and internal levels.

The design of a Database Management System highly depends on its architecture. It can be centralized or decentralized or hierarchical. DBMS architecture can be seen as single tier or

multi tier. n-tier architecture divides the whole system into related but independent n modules, which can be independently modified, altered, changed or replaced.
In 1-tier architecture, DBMS is the only entity where user directly sits on DBMS and uses it. Any changes done here will directly be done on DBMS itself. It does not provide handy tools for end users and preferably database designer and programmers use single tier architecture.
If the architecture of DBMS is 2-tier then must have some application, which uses the DBMS. Programmers use 2-tier architecture where they access DBMS by means of application. Here application tier is entirely independent of database in term of operation, design and programming.

3-tier architecture

Most widely used architecture is 3-tier architecture. 3-tier architecture separates it tier from each other on basis of users. It is described as follows:



Database Languages

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

Data-Definition Language

We specify a database schema by a set of definitions expressed by a special language called a data-definition language (DDL). For instance, the following statement in the SQL language defines the account table:

create table account (account-number char(10), balance integer)

Execution of the above DDL statement creates the account table. In addition, it updates a

special set of tables called the data dictionary or data directory. A data dictionary contains metadata—that is, data about data. The schema of a table is an example of metadata. A database system consults the data dictionary before reading or modifying actual data. We specify the storage structure and access methods used by the database system by a set of statements in a special type of DDL called a data storage and definition language.

These statements define the implementation details of the database schemas, which are

usually hidden from the users. The data values stored in the database must satisfy certain consistency constraints. For example, suppose the balance on an account should not fall below $100. The DDL provides facilities to specify such constraints. The database systems check these

Constraints every time the database is updated.

Data-Manipulation Language

Data manipulation is The retrieval of information stored in the database The insertion of new information into the database The deletion of information from the database The modification of information stored in the database A data-manipulation language (DML) is a language that enables users to access or manipulate data as organized by the appropriate data model. 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. Declarative DMLs are usually easier to learn and use than are procedural DMLs. However, since a user does not have to specify how to get the data, the database system has to figure out an efficient means of accessing data. The DML component of the SQL language is nonprocedural. 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. This query in the SQL language finds the name of the customer whose customer-id is 192-83-7465:

Select customer. Customer-name from customer where customer. Customer-id = 192-83-7465

The query specifies that those rows from the table customer where the customer-id is 192-83-7465 must be retrieved, and the customer-name attribute of these rows must be displayed. Queries may involve information from more than one table. For instance, the following query finds the balance of all accounts owned by the customer with customer id 192-83-7465.

select account.balance from depositor, account where depositor.customer-id = 192-83-7465 and depositor.account-number = account.account-number

There are a number of database query languages in use, either commercially or experimentally. The levels of abstraction apply not only to defining or structuring data, but also to manipulating data. At the physical level, we must define algorithms that allow efficient access to data. At higher levels of abstraction, we emphasize ease of use. The goal is to allow humans to interact efficiently with the system. The query processor component of the database system translates DML queries into sequences of actions at the physical level of the database system.

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. The two main types of data dictionary exist, integrated and stand alone. An integrated data dictionary is included with the DBMS. For example, all relational DBMSs include a built in data dictionary or system catalog that is frequently accessed and updated by the RDBMS. Other DBMSs especially older types, do not have a built in data dictionary instead the DBA may use third party stand alone data dictionary systems. Data dictionaries can also be classified as active or passive. An active data dictionary is automatically updated by the DBMS with every database access, thereby keeping its 15access information up-to-date. A passive data dictionary is not updated automatically and usually requires a batch process to be run. Data dictionary access information is normally used by the DBMS for query optimization purpose. The data dictionary‗s main function is to store the description of all objects that interact with the database. Integrated data dictionaries tend to limit their metadata to the data managed by the DBMS. Stand alone data dictionary systems are more usually more flexible and allow the DBA to describe and manage all the organization‗s data, whether or not they are computerized. Whatever the data dictionary‗s format, its existence provides database designers and end users with a much improved ability to communicate. In addition, the data dictionary is the tool that helps the DBA to resolve data conflicts. Although, there is no standard format for the information stored in the data dictionary several features are common. 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, datatypes, 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.

If the data dictionary can be organized to include data external to the DBMS itself, it becomes an specially flexible to for more general corporate resource management. The management of such an extensive data dictionary, thus, makes it possible to manage the use and allocation of all of the organization information regardless whether it has its roots in the database data. This is why some managers consider the data dictionary to be the key element of the information resource management function. And this is also why the data dictionary might be described as the information resource dictionary. The metadata stored in the data dictionary is often the bases for monitoring the database use and assignment of access rights to the database users. The information stored in the database is usually based on the relational table format, thus , enabling the DBA to query the database with SQL command. For example, SQL command can be used to extract information about the users of the specific table or about the access rights of a particular users.

UNIT 2

Objectives

At the end of this chapter the reader will be able to:

- Describe Data modeling, Entity Relation Model
- Distinguish between Entity set , weak entity strong entity
- Describe Relational model and relational Constraints
- Describe Relational model Concepts

Introduction

A data model is a conceptual representation of the data structures that are required by a database. The data structures include the data objects, the associations between data objects, and the rules which govern operations on the objects. As the name implies, the data model focuses on what data is required and how it should be organized rather than what operations will be performed on the data. To use a common analogy, the data model is equivalent to an architect's building plans. A data model is independent of hardware or software constraints. Rather than try to represent the data as a database would see it, the data model focuses on representing the data as the user sees it in the "real world". It serves as a bridge between the concepts that make up real-world events and processes and the physical representation of those concepts in a database.

Components of a Data Model

The data model gets its inputs from the planning and analysis stage. Here the modeler, along with analysts, collects information about the requirements of the database by reviewing existing documentation and interviewing end-users. The data model has two outputs. The first is an entity-relationship diagram which represents the data structures in a pictorial form. Because the diagram is easily learned, it is valuable tool to communicate the model to the end-user. The second component is a data document. This a document that describes in detail the data objects, relationships, and rules required by the database. The dictionary provides the detail required by the database developer to construct the physical database.

Why is Data Modeling Important?

Data modeling is probably the most labor intensive and d time consuming part of the development process. Why bother especially if you are pressed for time? A common response by practitioners who write on the subject is that you should no more build a database without a model than you should build a house without blueprints. The goal of the data model is to make sure that the all data objects required by the database are completely and accurately represented. Because the data model uses easily understood notations and natural language, it can be reviewed and verified as correct by the end-users. The data model is also detailed enough to be used by the database developers to use as a "blueprint" for building the physical database. The information contained in the data model will be used to define the relational tables, primary and foreign keys, stored procedures, and triggers.

Entity-Relationship Model

The entity-relationship (E-R) data model is based on a perception of a real world that consists of a collection of basic objects, called *entities*, and of *relationships* among these objects. An entity is a ―thing” or ―object” in the real world that is distinguishable from other objects. **Entity-Relationship model is based on the notion of real world entities and relationship among them. While formulating real-world scenario into database model, ER Model creates entity set, relationship set, general attributes and constraints.** For example, each person is an entity, and bank accounts can be considered as entities. Entities are described in a database by a set of **attributes**. For example, the attributes *account-number* and *balance* may describe one particular account in a bank, and they form attributes of the *account* entity set. Similarly, attributes *customer-name*, *customer-street* address and *customer-city* may describe a *customer* entity.

An extra attribute *customer-id* is used to uniquely identify customers (since it may be possible to have two customers with the same name, street address, and city).
A unique customer identifier must be assigned to each customer. In the United States, many enterprises use the social-security number of a person (a unique number the U.S. government assigns to every person in the United States) as a customer identifier.

A **relationship** is an association among several entities. For example, a *depositor* relationship associates a customer with each account that she has. The set of all entities of the same type and the set of all relationships of the same type are termed an **entity set** and **relationship set**, respectively.

Therefore it can be summarized as;
ER Model is best used for the conceptual design of database.

ER Model is based on:

- Entities and their attributes
- Relationships among entities

These concepts are explained below**.**

Entity

An entity in ER Model is real world entity, which has some properties called attributes. Every attribute is defined by its set of values, called domain.

For example, in a school database, a student is considered as an entity. Student has various attributes like name, age and class etc.

ER Notation

There is no standard for representing data objects in ER diagrams. Each modeling methodology uses its own notation. All notational styles represent entities as rectangular boxes and relationships as lines connecting boxes. Each style uses a special set of symbols to represent the cardinality of a connection. The notation used in this document is from Martin. The symbols used for the basic ER constructs are:

• Entities are represented by labeled rectangles. The label is the name of the entity. Entity names should be singular nouns.

• Relationships are represented by a solid line connecting two entities. The name of the relationship is written above the line. Relationship names should be verbs.

• Attributes, when included, are listed inside the entity rectangle. Attributes which are identifiers are underlined. Attribute names should be singular nouns.

• Cardinality of many is represented by a line ending in a crow's foot. If the crow's foot is omitted, the cardinality is one.

• Existence is represented by placing a circle or a perpendicular bar on the line.

Mandatory existence is shown by the bar (looks like a 1) next to the entity for an instance is required. Optional existence is shown by placing a circle next to the entity that is optional.

Steps In Building the Data Model

While ER model lists and defines the constructs required to build a data model, there is no standard process for doing so. Some methodologies, such as IDEFIX, specify a bottom-up development process were the model is built in stages. Typically, the entities and relationships are modeled first, followed by key attributes, and then the model is finished by adding non-key attributes. Other experts argue that in practice, using a phased approach is impractical because it requires too many meetings with the end-users. The sequence used for this document are:

1. Identification of data objects and relationships
2. Drafting the initial ER diagram with entities and relationships
3. Refining the ER diagram
4. Add key attributes to the diagram
5. Adding non-key attributes
6. Diagramming Generalization Hierarchies
7. Validating the model through normalization

8. Adding business and integrity rules to the Model

In practice, model building is not a strict linear process. As noted above, the requirements analysis and the draft of the initial ER diagram often occur simultaneously. Refining and validating the diagram may uncover problems or missing information which require more information gathering and analysis

Identifying Data Objects and Relationships

In order to begin constructing the basic model, the modeler must analyze the information

gathered during the requirements analysis for the purpose of:

- Classifying data objects as either entities or attributes
- Identifying and defining relationships between entities
- Naming and defining identified entities, attributes, and relationships
- Documenting this information in the data document

To accomplish these goals the modeler must analyze narratives from users, notes from meeting, policy and procedure documents, and, if lucky, design documents from the current information system. Although it is easy to define the basic constructs of the ER model, it is not an easy task to distinguish their roles in building the data model. What makes an object an entity or attribute? For example, given the statement "employees work on projects". Should employees be classified as an entity or attribute? Very often, the correct answer depends upon the requirements of the database. In some cases, employee would be an entity, in some it would be an attribute.

Attributes

Attributes are data objects that either identify or describe entities. Attributes that identify entities are called key attributes. Attributes that describe an entity are called non-key attributes. Key attributes will be discussed in detail in a latter section. The process for identifying attributes is similar except now you want to look for and extract those names that appear to be descriptive noun phrases.

Relationships

Relationships are associations between entities. Typically, a relationship is indicated by a verb connecting two or more entities. For example: employees are assigned to projects As relationships are identified they should be classified in terms of cardinality, optionality, direction, and dependence. As a result of defining the relationships, some relationships may be dropped and new relationships added. Cardinality quantifies the relationships between entities by measuring how many instances of one entity are related to a single instance of another. To determine the cardinality, assume the existence of an instance of one of the entities. The logical association among entities is called relationship.

Relationships are mapped with entities in various ways. Mapping cardinalities define the number of association between two entities.

Mapping cardinalities:

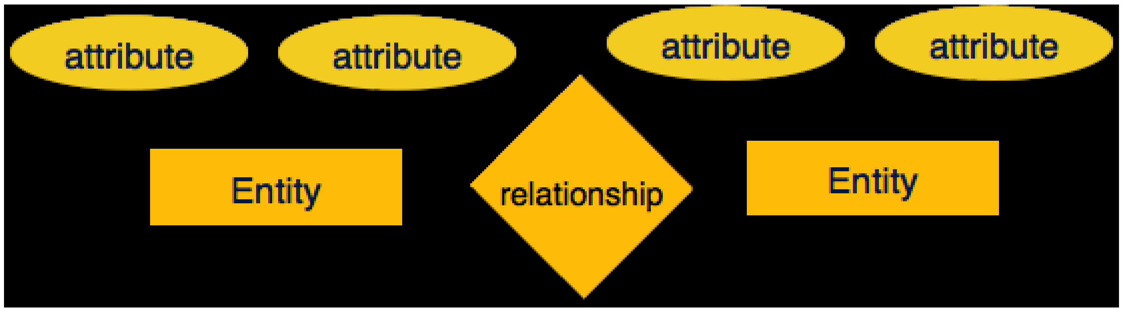
one to one

one to many

many to one

many to many

The overall logical structure (schema) of a database can be expressed graphically by an *E-R diagram.as*

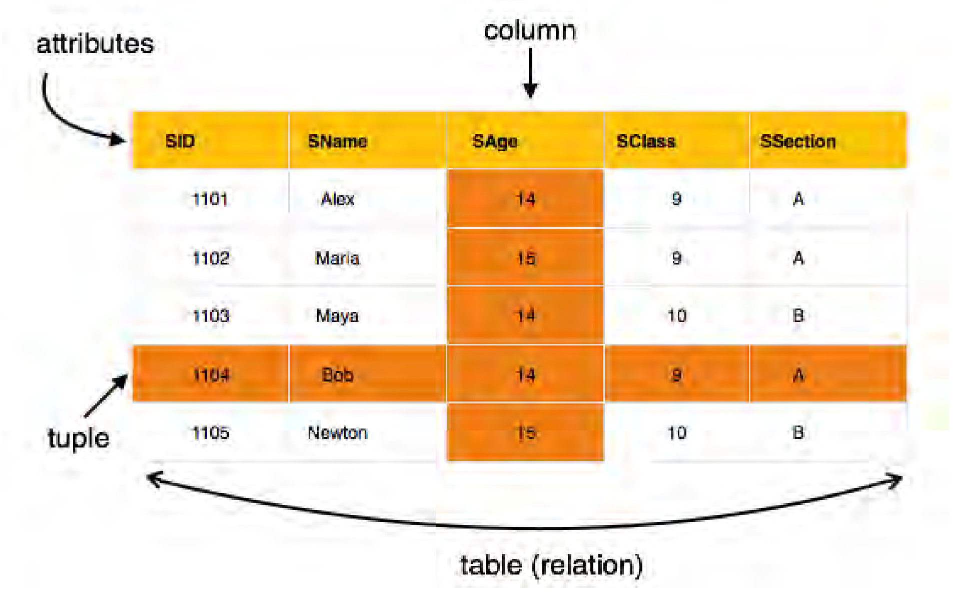


Relational Model

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.
The data is arranged in a relation which is visually represented in a two dimensional table. The data is inserted into the table in the form of tuples (which are nothing but rows). A tuple is formed by one or more than one attributes, which are used as basic building blocks in the formation of various expressions that are used to derive meaningful information. There can be any number of tuples in the table, but all the tuple contain fixed and same attributes with varying values. The relational model is implemented in database where a relation is represented by a table, a tuple is represented by a row, an attribute is represented by a column of the table, attribute name is the name of the column such as _identifier', _name', _city' etc., attribute value contains the value for column in the row. Constraints are applied to the table and form the logical schema. In order to facilitate the selection of a particular row/tuple from the table, the attributes i.e. column names are used, and to expedite the selection of the rows some fields are defined uniquely to use them as indexes, this helps in searching the required data as fast as possible. All

the relational algebra operations, such as Select, Intersection, Product, Union, Difference, Project, Join, Division, Merge etc. can also be performed on the Relational Database Model. Operations on the Relational Database Model are facilitated with the help of different conditional expressions, various key attributes, pre-defined constraints etc. Hence in nutshell The most popular data model in DBMS is Relational Model. It is more scientific model then others. This model is based on first-order predicate logic and defines table as an n-ary relation.
The main highlights of this model are:

- Data is stored in tables called relations.
- Relations can be normalized.
- In normalized relations, values saved are atomic values.
- Each row in relation contains unique value
- Each column in relation contains values from a same domain



SID	SName	SAge	SClass	SSection
1101	Alex	14	9	A
1102	Maria	15	9	A
1103	Maya	14	10	B
1104	Bob	14	9	A
1105	Newton	15	10	B

Relational Model Concepts
We shall represent a relation as a table with columns and rows. Each column of the **table** has a name, or ***attribute***. Each row is called a ***tuple***.
• **Domain**: a set of atomic values that an attribute can take
• **Attribute**: name of a column in a particular table (all data is stored in tables). Each attribute Ai must have a *domain*, dom(A_i).

• **Relational Schema**: The design of one table, containing the name of the table (i.e. the name of the relation), and the names of all the columns, or attributes.

Example: STUDENT(Name, SID, Age, GPA)
• **Degree of a Relation**: the number of attributes in the relation's schema.
• **Tuple, t, of R(A1, A2, A3, …, An):** an ORDERED set of values, < v1, v2, v3, …, vn>, where each v_i is a value from dom(A_i).

Properties of relations

- ❖ Properties of database relations are:
- ❖ Relation name is distinct from all other relations
- ❖ Each cell of relation contains exactly one atomic (single) value
- ❖ Each attribute has a distinct name
- ❖ Values of an attribute are all from the same domain
- ❖ Order of attributes has no significance
- ❖ Each tuple is distinct; there are no duplicate tuples
- ❖ Order of tuples has no significance, theoretically.

Relational keys :

There are two kinds of keys in relations. The first are identifying keys: the **primary key** is the main concept, while two other keys – **super key** and **candidate key** – are related concepts. The second kind is the foreign key.

Identity Keys

Super Keys

A super key is a set of attributes whose values can be used to uniquely identify a tuple within a relation. relation may have more than one super key, but it always has at least one: the set of all attributes that make up the relation.

Candidate Keys

A candidate key is a super key that is minimal; that is, there is no proper subset that is itself a super key. A relation may have more than one candidate key, and the different candidate keys may have a different number of attributes. In other words, you should not interpret 'minimal' to mean the super key with the fewest attributes.

A candidate key has two properties:
(i) in each tuple of R, the values of K uniquely identify that tuple (uniqueness)
(ii) no proper subset of K has the uniqueness property (irreducibility).

Primary Key

The primary key of a relation is a candidate key especially selected to be the key for the relation. In other words, it is a choice, and there can be only one candidate key designated to be the primary key.

Relationship between identity keys

The relationship between keys:

Super key ⊇ Candidate Key ⊇ Primary Key

Foreign keys

The attribute(s) within one relation that matches a candidate key of another relation. A relation may have several foreign keys, associated with different target relations.

Foreign keys allow users to link information in one relation to information in another relation. Without FKs, a database would be a collection of unrelated tables.

Relational Model Constraints

Integrity Constraints

Each relational schema must satisfy the following four types of constraints.

A. Domain constraints

Each attribute **Ai** must be an atomic value from dom(**Ai**) for that attribute.

The attribute, Name in the example is a BAD DESIGN (because sometimes we may want to search person by only using their last name.

B. Key Constraints

Super key of : A set of attributes, SK, of R such that no two tuples in any valid relational instance, r(R), will have the same value for SK. Therefore, for any two distinct tuples, t1 and t2 in r(R), t1[SK] != t2[SK].

Key of R: A minimal superkey. That is, a superkey, K, of R such that the removal of ANY attribute from K will result in a set of attributes that are not a superkey.

Example CAR(State, LicensePlateNo, VehicleID, Model, Year, Manufacturer)

This schema has two keys:

K1 = { State, LicensePlateNo}

K2 = { VehicleID }

Both K1 and K2 are superkeys.

K3 = { VehicleID, Manufacturer} is a superkey, but not a key (Why?).

If a relation has more than one keys, we can select any one (arbitrarily) to be the primary key. Primary Key attributes are underlined in the schema:

CAR(<u>State, LicensePlateNo</u>, VehicleID, Model, Year, Manufacturer)

C. Entity Integrity Constraints

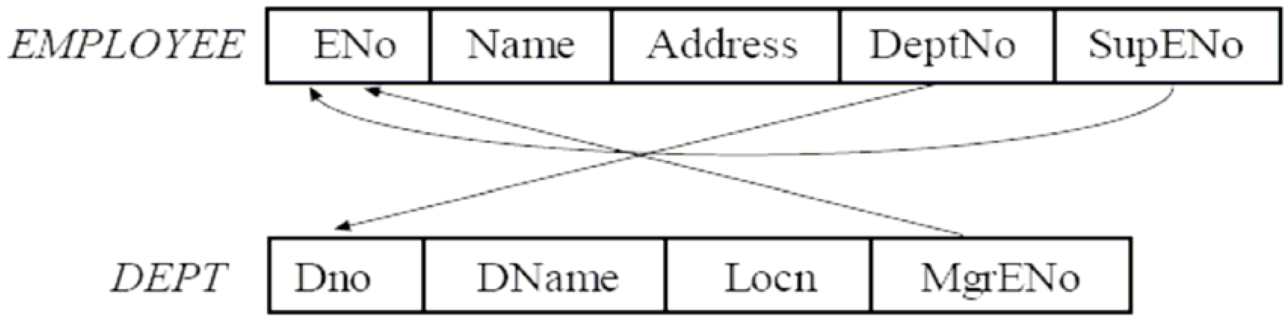
The primary key attribute, PK, of any relational schema R in a database cannot have null values in any tuple. In other words, for each table in a DB, there must be a key; for each key, every row in the table must have non-null values. This is because PK is used to identify the individual tuples.

Mathematically, t[PK] != NULL for any tuple t € r(R).

D. Referential Integrity Constraints

Referential integrity constraints are used to specify the relationships between two relations in a database.

Consider a referencing relation, R1, and a referenced relation, R2. Tuples in the referencing relation, R1, have attributed FK (called **foreign key** attributes) that reference,the primary key attributes of the referenced relation, R2. A tuple, t1, in R1 is said to reference a tuple, t2, in R2 if t1[FK] = t2[PK].
A referential integrity constraint can be displayed in a relational database schema as a directed arc from the referencing (foreign) key to the referenced (primary) key. Examples are shown in the figure below:


EMPLOYEE
ENo
Name
Address
DeptNo
SupENo
DEPT
Dno
DName
Locn
MgrENo