**Introduction**

There are many topics that can be discussed in database. Among them, attributes is an important one. Before attributes are being discussed, a person should know about the database and database management system.

**Description**

**Database**

**Database**, also called **electronic database**, any collection of data, or [information](https://www.britannica.com/science/information-science), that is, specially organized for rapid search and retrieval by a [computer](https://www.britannica.com/technology/computer). Databases are structured to [facilitate](https://www.merriam-webster.com/dictionary/facilitate) the storage, retrieval, modification, and deletion of data in conjunction with various data-processing operations.  Database is stored as a [file](https://www.britannica.com/technology/file-computing) or a set of files on magnetic disk or tape, optical disk, or some other secondary storage device. The information in these files may be broken down into [records](https://www.britannica.com/technology/record-computing), each of which consists of one or more fields. Fields are the basic units of data storage, and each field typically contains information pertaining to one aspect or attribute of the entity described by the database. Records are also organized into tables that include information about relationships between its various fields. Although database is applied loosely to any collection of information in computer files, a database in the strict sense provides cross-referencing capabilities. Using keywords and various sorting commands, users can rapidly search, rearrange, group, and select the fields in many records to retrieve or create reports on particular [aggregates](https://www.merriam-webster.com/dictionary/aggregates) of data.

**Database Management System**

A database management system (or DBMS) is essentially nothing more than a computerized data-keeping system. Users of the system are given facilities to perform several kinds of operations on such a system for either manipulation of the data in the database or the management of the database structure itself. Database Management Systems (DBMSs) are categorized according to their data structures or types.

Mainframe sites tend to use a hierarchical model when the data **structure** (not data values) of the data needed for an application is relatively static. For example, a Bill of Material (BOM) database structure always has a high level assembly part number, and several levels of components with subcomponents. The structure usually has a component forecast, cost, and pricing data, and so on. The structure of the data for a BOM application rarely changes, and new data elements (not values) are rarely identified. An application normally starts at the top with the assembly part number, and goes down to the detail components.

Hierarchical and relational database systems have common benefits. RDBMS has the additional, significant advantage over the hierarchical DB of being non-navigational. By **navigational**, we mean that in a hierarchical database, the application programmer must know the structure of the database. The program must contain specific logic to navigate from the root segment to the desired child segments containing the desired attributes or elements. The program must still access the intervening segments, even though they are not needed.

**Attributes**

A [database](https://www.lifewire.com/databases-glossary-1019603) consists of tables. Each table has columns and rows.

Each row (called a tuple) is a data set that applies to a single item. Each column (attribute) contains describing characteristics of the rows. A database attribute is a column name and the content of the fields under it in a table in a database.

If we sell products and enter them into a table with columns for Product Name, Price, and Product ID, each of those headings is an attribute. In each field under those headings, we enter the product names, prices, and product IDs, respectively. Each one of the field entries is also an attribute.

This makes sense when we think of it, given that the nontechnical definition of an attribute is that it defines a characteristic or quality of something.

Attributes are the characteristics that describe entities and relationships. For example, a Student entity may be described by attributes including: student number, name, address, date of birth, degree major. An Invoice entity may be described by attributes including: invoice number, invoice date, invoice total. In an ERD using the Chen notation, we illustrate attributes using ovals as shown below.

****

Suppose we have entity types Student and Course that are associated via an enroll relationship. An attribute that helps to describe a student enrolled in a course is grade:



A common convention for naming attributes is to use singular nouns. A naming convention may require one of:

1. All characters are in upper case.

2. All characters are in lower case.

3. Only the first character is in upper case.

4. Each part of a multipart name has the first character capitalized.

A typical convention is for attribute names to have a prefix that indicates the entity the attribute describes. Subsequent characters are sufficiently descriptive to identify the attribute. Some examples of attribute names:

1. empLname = employee last name

2. stuGpa = student grade point average

3. prodCode = product code

4. invNum = invoice number

**Atomic Attributes**

An attribute is considered atomic (or simple) if it does not contain any meaningful smaller components.

For example, suppose "Gender" is an attribute in our design. The Gender attribute has a small set of possible values, for example M or F. It is not meaningful to decompose Gender into smaller units, and so we say Gender is a Simple attribute.

As another example consider an attribute for product price, prodPrice. A sample value for prodPrice is $21.03. Of course, one could decompose prodPrice into two attributes where one attribute represents the dollar component (21), and the other attribute represents the cents component (03), but our assumption here is that such a decomposition is not meaningful to the intended application or system that will make use of it. So we would consider prodPrice to be atomic because it cannot be usefully decomposed into meaningful components.

**Composite Attributes**

An attribute is considered composite if it comprises two or more other attributes.

Consider an attribute such as name that comprises first and last names. For example, suppose an employee's name is John McKenzie. The first name is John and the last name is McKenzie. It is easy to appreciate that one application may only want the last name, another may display the first name followed by the last name, and yet another application may display the last name, a comma, and then the first name.

Since it is meaningful to decompose empName into two attributes for first name, firstName, and last name, lastName, we consider the name attribute to be a composite attribute (firstName and lastName are non-composite; they are atomic attributes). A composite attribute is an attribute that is shown as comprising two or more simpler attributes; we show a composite attribute below.



**Derived Attributes**

If an attribute's value can be determined from the values of other attributes, then the attribute is derivable, and is said to be a derived attribute.

If an attribute's value can be derived from other attributes, you should consider dropping the attribute from the model. Perhaps you would keep it, if keeping it helped understandability (you can still decide to drop the attribute from the physical model). Derived attributes are shown with a dotted lined oval, see the figure below.



Sometimes an attribute of one entity is derived from attributes in other entities. Consider the attribute for the total of an Invoice, InvTotal. A value of InvTotal is derivable; it can be computed by knowing the value of InvLineAmount from each related invoice line. Keeping an attribute such as InvTotal in the model can easily promote communication with other interested parties (other business users). This example also indicates a situation where deriving a value (on-the-fly) could be an expensive operation to perform (to determine the value of an invoice would mean accessing all relevant invoice line data in the database). Such an attribute may find its way into the eventual physical model for that reason.



**Single-Valued Attributes**

An attribute is considered single-valued if there is at most one value associated with it at any one point in time.

For example, suppose "gender" is an attribute in our design. For most applications we would say that gender is single-valued; at any given point in time, there is just one value (male, female) recorded for gender for a person.

We characterize an attribute as being single-valued if, for any instance of the pertinent entity set, there is only one value at a given time for the attribute.

Single-valued attributes are shown with a simple oval, one with a single line for a border (as opposed to a double-lined border). In all of our examples so far, we have assumed that each attribute was single-valued.

**Multi-Valued Attributes**

An attribute is considered multi-valued if there can be many values associated with it at any one point in time.

Now, suppose someone proposes to track the university degrees earned by employee's with an attribute named empDegree. When an attribute could have none, one, or several values, we say the attribute is multi-valued. For a given employee and point in time, empDegree could have multiple values, and so we say it is multi-valued. If you need to keep track of all the degrees that each person has obtained, then there would be multiple values to store.

The following shows sample data for three employees. We assume that each employee has just a single employee number, and that we are keeping only one phone number per employee; both empNum and empPhone are considered to be single-valued.

|  |  |  |  |
| --- | --- | --- | --- |
| **empNum** | **empPhone** | | **empDegree** |
| 333 | 233-1231 | BA, BSc, PhD | |
| 679 | 233-1231 | BSc, MSc | |
| 123 | 233-9876 |  | |



**Stored Attributes**

If an attribute's value cannot be determined from the values of other attributes, then the attribute's value for an entity must be kept as part of the entity and we refer to such an attribute as a Stored attribute.

When an attribute is specified for an entity or relationship type, and when it is depicted with a solid line (as opposed to a dashed line), it is considered a stored attribute. The designers intent is that the attributes value will be part of an instance of the entity or relationship. This is in contrast to a derived attribute. In the example below where we have attributes for an employee, the birth date is shown as a stored attribute and current age is shown as derived.



**Key Attributes**

An attribute is a Key if values of the attribute uniquely identify instances of a corresponding entity set.

For example, suppose "Student Number" is an attribute for the Student entity type in our design for a University database. Suppose Student Number is just one of several attributes; others are: First Name, Last Name, and Gender. Suppose further that the University assigns each student a unique student number, and assuming that we intend to have just one entity instance per student, Student Number is a Key. In an ERD, keys are shown underlined:



For each entity type, we prefer to have an attribute, or a combination of attributes that identify individual entities.

We define a key to be an attribute, or a minimal set of attributes, that uniquely identify entities in an entity set. By minimal we mean that all of the attributes are required (none can be omitted). For instance, a typical key for an invoice line entity type would be the combination of invoice number and invoice line number. Both attributes are required to identify a particular invoice line. However, such a key is composite (see the section on composite attributes).

**Limitations**

This topic has no limitations so far.

**References**

**ELMASRI, R. AND NAVATHE, S.** (Book)

Fundamentals of Database Systems