THREE SCHEMA ARCHETECTURE OF DBMS:

In this architecture, the overall database description can be defined at three levels, namely, internal, conceptual, and external levels (**three-level DBMS architecture)**. This architecture is proposed by ANSI/SPARC (American National Standards Institute/Standards Planning and Requirements Committee) and hence, is also known as **ANSI/SPARC architecture**. The three levels are discussed here.

- **Internal level**: It is the lowest level of data abstraction that deals with the physical representation of the database on the computer and thus, is also known as physical level. It describes how the data is physically stored and organized on the storage medium. At this level, various aspects are considered to achieve optimal runtime performance and storage space utilization. These aspects include storage space allocation techniques for data and indexes, access paths such as indexes, data compression and encryption techniques, and record placement.
- **Conceptual level**: This level of abstraction deals with the logical structure of the entire database and thus, is also known as logical level. It describes what data is stored in the database, the relationships among the data and complete view of the user's requirements without any concern for the physical implementation. That is, it hides the complexity of physical storage structures. The conceptual view is the overall view of the database and it includes all the information that is going to be represented in the database.
- **External level**: It is the highest level of abstraction that deals with the user's view of the database and thus, is also known as view level. In general, most of the users and application programs do not require the entire data stored in the database. The external level describes a part of the database for a particular group of users. It permits users to access data in a way that is customized according to their needs, so that the same data can be seen by different users in different ways, at the same time. In this way, it provides a powerful and flexible security mechanism by hiding the parts of the database from certain users, as the user is not aware of existence of any attributes that are missing from the view.

*These three levels are used to describe the schema of the database at various levels. Thus, the three-level architecture is also known as three-schema architecture. The internal level has an **Internal schema**, which describes the physical storage structure of the database. The conceptual level has a **Conceptual schema**, which describes the structure of entire database. The external level has **External schemas** or user views, which describe the part of the database according to a particular user's requirements, and hide the rest of the database from that user. The physical level is managed by the operating system under the direction of DBMS. The three-schema architecture is shown in Figure 1.1.*
http://...Fig. 1.1. Three-schema architecture

To understand the three-schema architecture, consider the three levels of the BOOK file in Online Book database as shown in Figure 1.2. In this figure, two views (view 1 and view 2) of the BOOK file have been defined at the external level. Different database users can see these views. The details of the data types are hidden from the users. At the conceptual level, the BOOK records are described by a type definition. The application programmers and the DBA generally work at this level of abstraction. At the internal level, the BOOK records are described as a block of consecutive storage locations such as words or bytes. The database users and the application programmers are not aware of these details; however, the DBA may be aware of certain details of the physical organization of the data.
http://...Fig. 1.2. Three levels of Online Book database (BOOK file)

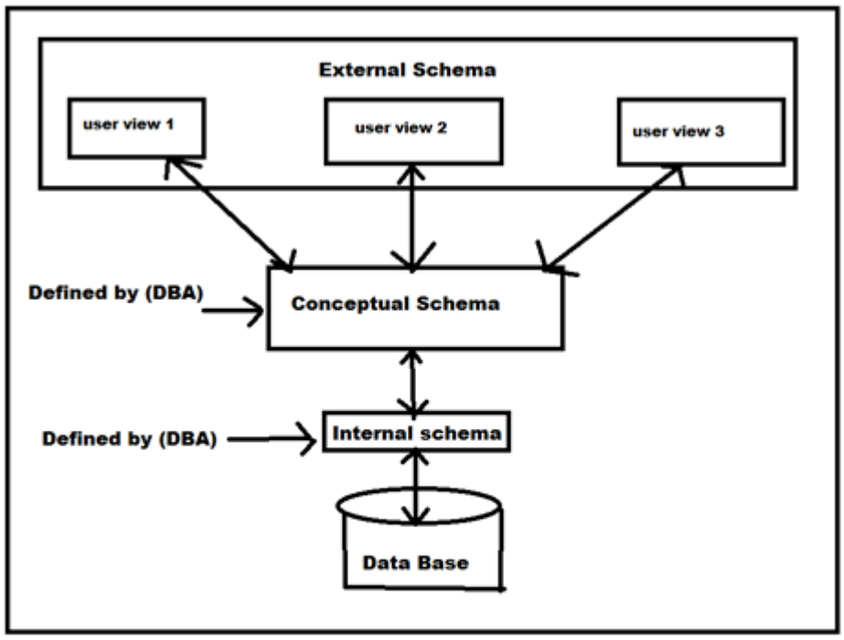
*http://...In three-schema architecture, each user group refers only to its own external view. Whenever a user specifies a request to generate a new external view, the DBMS must transform the request specified at external level into a request at conceptual level, and then into a request at physical level. If the user requests for data retrieval, the data extracted from the database must be presented according to the need of the user. This process of transforming the requests and results between various levels of DBMS architecture is known as **http://...mapping**.*

The main advantage of three-schema architecture is that it provides data independence.

Data independence is the ability to change the schema at one level of the database system without having to change the schema at the other levels. Data independence is of two types, namely, logical data independence and physical data independence.

- **Logical data independence**: It is the ability to change the conceptual schema without affecting the external schemas or application programs. The conceptual schema may be changed due to change in constraints or addition of new data item or removal of existing data item, etc., from the database. The separation of the external level from the conceptual level enables the users to make changes at the conceptual level without affecting the external level or the application programs. For example, if a new data item, say Edition is added to the BOOK file, the two views (view 1 and view 2 shown in Figure 1.2) are not affected.
- **Physical data independence**: It is the ability to change the internal schema without affecting the conceptual or external schema. An internal schema may be changed due to several reasons such as for creating additional access structure, changing the storage structure, etc. The separation of internal schema from the conceptual schema facilitates physical data independence.

Logical data independence is more difficult to achieve than the physical data independence because the application programs are always dependent on the logical structure of the database. Therefore, the change in the logical structure of the database may require change in the application programs.



The ER model defines the conceptual view of a database. It works around real-world entities and the associations among them. At view level, the ER model is considered a good option for designing databases.

Entity

An entity can be a real-world object, either animate or inanimate, that can be easily identifiable. For example, in a school database, students, teachers, classes, and courses offered can be considered as entities. All these entities have some attributes or properties that give them their identity.

An entity set is a collection of similar types of entities. An entity set may contain entities with attribute sharing similar values. For example, a Students set may contain all the students of a school; likewise a Teachers set may contain all the teachers of a school from all faculties. Entity sets need not be disjoint.

Attributes

Entities are represented by means of their properties, called **attributes**. All attributes have values. For example, a student entity may have name, class, and age as attributes.

There exists a domain or range of values that can be assigned to attributes. For example, a student's name cannot be a numeric value. It has to be alphabetic. A student's age cannot be negative, etc.

Types of Attributes

- **Simple attribute** − Simple attributes are atomic values, which cannot be divided further. For example, a student's phone number is an atomic value of 10 digits.
- **Composite attribute** − Composite attributes are made of more than one simple attribute. For example, a student's complete name may have first_name and last_name.
- **Derived attribute** − Derived attributes are the attributes that do not exist in the physical database, but their values are derived from other attributes present in the database. For example, average_salary in a department should not be saved directly in the database, instead it can be derived. For another example, age can be derived from data_of_birth.
- **Single-value attribute** − Single-value attributes contain single value. For example − Social_Security_Number.
- **Multi-value attribute** − Multi-value attributes may contain more than one values. For example, a person can have more than one phone number, email_address, etc.

These attribute types can come together in a way like −

- simple single-valued attributes
- simple multi-valued attributes
- composite single-valued attributes
- composite multi-valued attributes

Entity-Set and Keys

Key is an attribute or collection of attributes that uniquely identifies an entity among entity set.

For example, the roll_number of a student makes him/her identifiable among students.

- **Super Key** − A set of attributes (one or more) that collectively identifies an entity in an entity set.
- **Candidate Key** − A minimal super key is called a candidate key. An entity set may have more than one candidate key.
- **Primary Key** − A primary key is one of the candidate keys chosen by the database designer to uniquely identify the entity set.

Relationship

The association among entities is called a relationship. For example, an employee **works_at** a department, a student **enrolls** in a course. Here, Works_at and Enrolls are called relationships.

Relationship Set

A set of relationships of similar type is called a relationship set. Like entities, a relationship too can have attributes. These attributes are called **descriptive attributes**.

Degree of Relationship

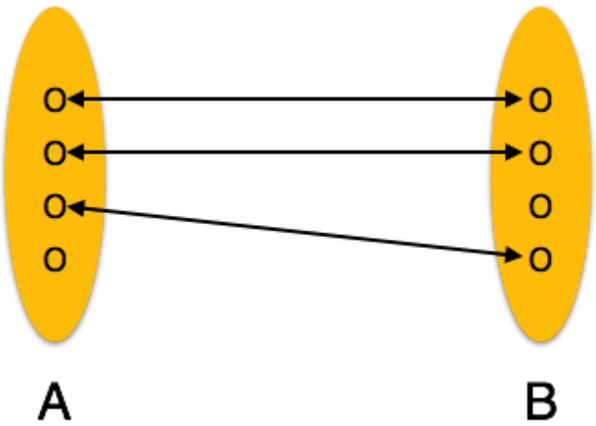
The number of participating entities in a relationship defines the degree of the relationship.

- Binary = degree 2
- Ternary = degree 3
- n-ary = degree

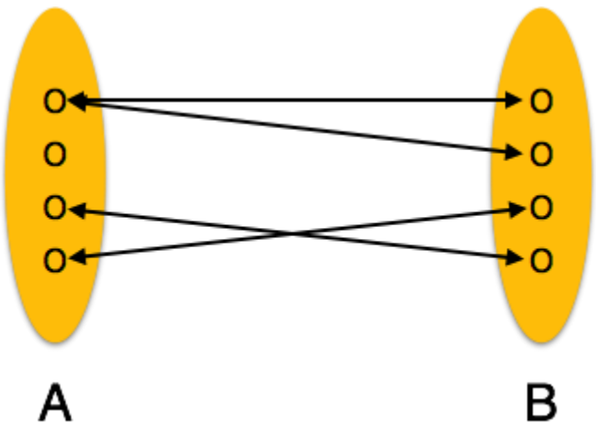
Mapping Cardinalities

Cardinality defines the number of entities in one entity set, which can be associated with the number of entities of other set via relationship set.

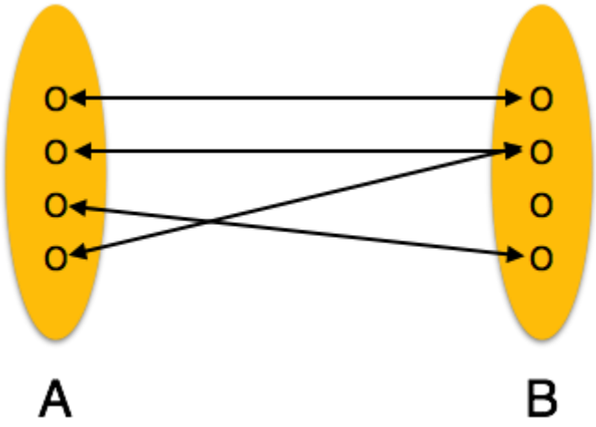
- **One-to-one** − One entity from entity set A can be associated with at most one entity of entity set B and vice versa.



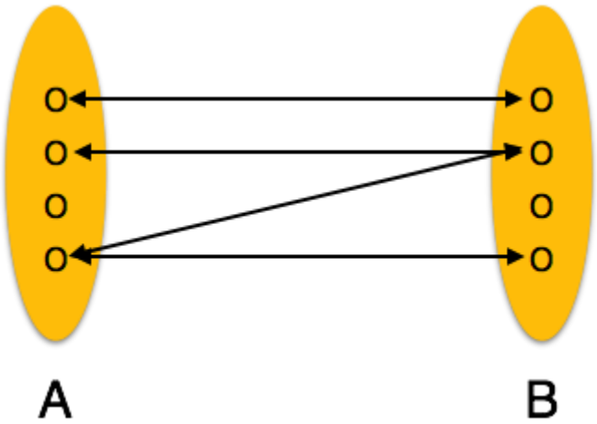
- **One-to-many** − One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



- **Many-to-one** − More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.



- **Many-to-many** − One entity from A can be associated with more than one entity from B and vice versa.



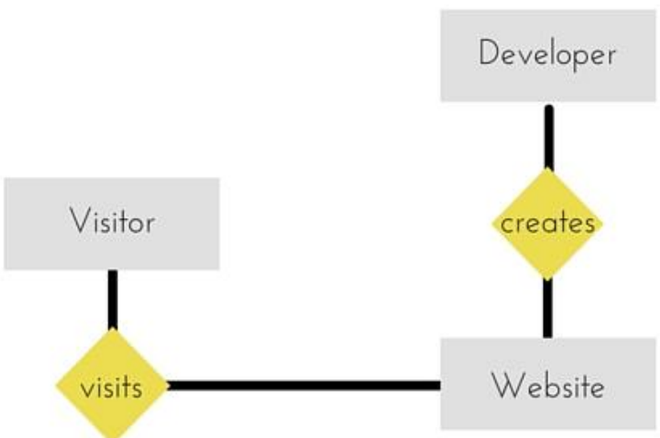
E-R Model:

The entity-relationship model (ER model) is a way of graphically representing the logical relationships of entities in order to create a database. In ER modelling, the structure for a database is represented as a diagram, called an entity-relationship diagram (or ER diagram), that resembles the graphical breakdown. Entities are rendered as points, polygons, circles, or ovals. Relationships are portrayed as lines connecting the points, polygons, circles, or ovals. Any ER diagram has an equivalent relational table, and any relational table has an equivalent ER diagram. ER diagramming is a tool to engineers in the design, optimization, and debugging of database programs.

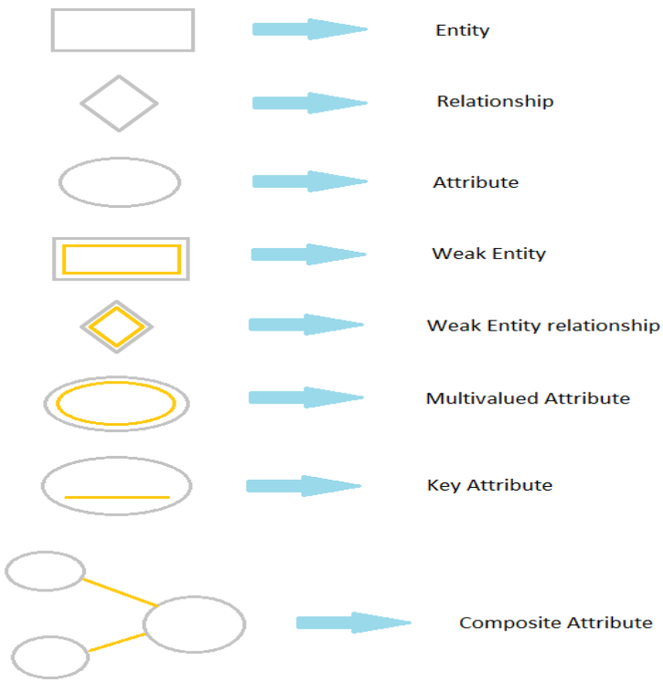
E-R Diagram

An *entity -relationship (ER) diagram*, is a graphical representation of entities and their relationships to each other, typically used in computing in regard to the organization of data within databases or information systems. An entity is a piece of data-an object or concept about which data is stored.

ER-Diagram is a visual representation of data that describes how data is related to each other.



Symbols and Notations



Components of E-R Diagram

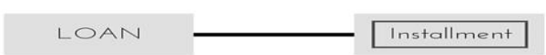
The E-R diagram has three main components.

1) Entity

An **Entity** can be any object, place, person or class. In E-R Diagram, an **entity** is represented using rectangles. Consider an example of an Organisation. Employee, Manager, Department, Product and many more can be taken as entities from an Organisation.

Weak Entity

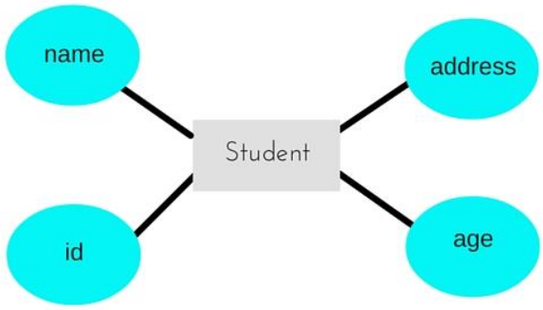
Weak entity is an entity that depends on another entity. Weak entity doen't have key attribute of their own. Double rectangle represents weak entity.



---

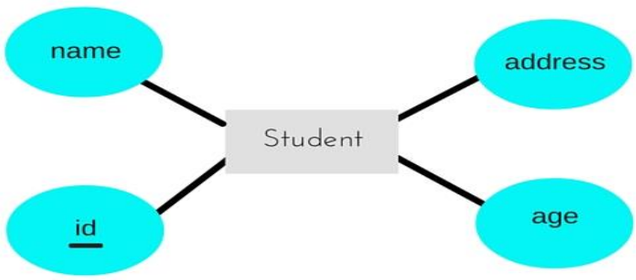
2) Attribute

An **Attribute** describes a property or characterstic of an entity. For example, Name, Age, Address etc can be attributes of a Student. An attribute is represented using eclipse.



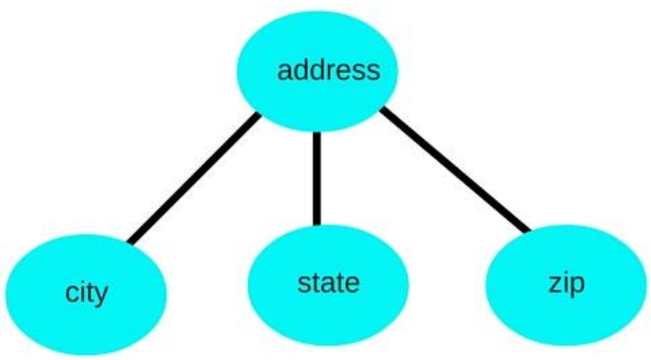
Key Attribute

Key attribute represents the main characterstic of an Entity. It is used to represent Primary key. Ellipse with underlying lines represent Key Attribute.



Composite Attribute

An attribute can also have their own attributes. These attributes are known as **Composite** attribute.



---

3) Relationship

A Relationship describes relations between **entities**. Relationship is represented using diamonds.

There are three types of relationship that exist between Entities.

- Binary Relationship
- Recursive Relationship
- Ternary Relationship

Different types of relationship:

There are four relationships in database.

One to One: One entity is associated with another entity.
For Ex: Each employee is associated with one department

- One to Many: One entity is associated with many other entities.
For Ex: A company is associated with all working employees in one branch/office/country.

- Many to One: Many entities are associated with only one entity.
For Ex: Many employees are associated with one project.

- Many to Many: Many entities are associated with many other entities.
For Ex: In a company many employees are associated with multiple projects (completed/existing), and at the same time, projects are associated with multiple employees.

[SEE BOOK PAGE NO. 22/23]

Relational Model:

A relational database is based on the relational model developed by E.F. Codd. A relational database allows the definition of data structures, storage and retrieval operations and integrity constraints. In such a database the data and relations between them are organized into tables. A table is a collection of records and each record in a table contains the same fields. The contents of a table can be permanently saved for future use.

The relational model used the basic concept of a relation or table. The columns or fields in the table identify the attributes such as name, age, and so. In the relational model, every tuple must have a unique identification or key based on the data. Often, keys are used to join data from two or more relations based on matching identification. The relational model also includes concepts such as foreign keys, which are primary keys in one relation that re kept in another relation to allow for the joining of data.

Properties of the relational database model

1. Data is presented as a collection of relations (tables).
2. Each relation is depicted as a table.
3. Columns are attributes that belong to the entity modelled by the table (ex. In a student table, we may have name, address, student ID, major, etc.).
4. Each row ("tuple") represents a single entity
5. Every table has a set of attributes that taken together as a "key" (technically, a "superkey") uniquely identifies each entity. (Ex. In the student table, “student ID” would uniquely identify each student – no two students would have the same student ID).

Integrity constraints:

Integrity Constraints are sets of rules that can help maintain the quality of information that is inserted in a database. Integrity constraints are mostly used when trying to promote accuracy and consistency of data that is found in a relational database. This is very important to companies because information can be considered as an asset to certain organizations and it must be protected.
Relational database model includes two general integrity rules-

- **Entity Integrity**
  Entity integrity is a mechanism that allows uniquely identified rows in a table. According to tis rule if an attribute of a table is prime attribute then it cannot accept null values. That means we can apply Entity Integrity to a table in RDBMS by specifying PRIMARY KEY constraint.

- **Referential Integrity**
  This rule ensures that the relationship between tables remain preserved when we insert, update or delete data from a database. It mainly concern with the concept of Foreign Key and primary key. Referential integrity ensures that relationships between tables remain consistent. When one table has a foreign key to another table, the concept of referential integrity states that we may not add a record to the table that contains the foreign key unless there is a corresponding record in the linked table.

Example:

TABLE-1

EMP_ID	**DEP_ID**	**EMP_NAME**
1001	PSC	JOHN
1002	ENG	ROHIT
1003	DIT	SMITA

TABLE-2

DEP_ID	**HOD**
PSC	PRANJIT
ENG	KOUSHIK
ECO	DALIMI
MIL	LEENA
EDU	RANJU

Here “DEP_ID” DIT in the table TABLE-1 is not allowed as because we don’t have a PRIMARY KEY match in the related table TABLE-2. That means since DEP_ID is primary key in TABLE-2, so every entry of DEP_ID must first present in TABLE-2 before we use it in some other table. This is called referential integrity.