

Problema:

Trabaje con la tabla "Libreta".

2- Cree una tabla llamada "libreta".

Debe tener los siguientes campos:

nombre (cadena de 20),

domicilio (cadena de 30) y

teléfono (cadena de 11):

3- Visualice las tablas existentes para verificar la creación de "libreta".

4- Visualice la estructura de la tabla "libreta".

5- Ingrese los siguientes registros:

('Alberto Mores','Colon 123','4234567');

('Juan Torres','Avellaneda 135','4458787');

6- Seleccione y muestre todos los registros de la tabla:

7. Construya las sentencias que actualicen los datos que acaba de insertar.

8. Inserte 5 registros más.

9. Cuente cuantos registros se ingresaron

## Taller Introducción a BD NoSql

1. Que es una base de datos NOSql
2. que es un JSON?
3. Características de las bases de datos Nosql
4. tipos de baese de datos nosql
5. Diferencias entre sql y nosql
6. Ejemplos (minimo 2) de como pasar un base de datos relacional a nosql
- 7 Cuales son los principales motores de bd nosql?
8. como ejecutar consultas en motores de bases de datos nosql? (Ejemplos minimo 5)



Introducción a  
la base de datos



**MySQL**®

**FAVA** - Formación en Ambientes Virtuales de Aprendizaje

**SENA** - Servicio Nacional de Aprendizaje.

## ESTRUCTURA DE CONTENIDOS

|                                                         | Pág. |
|---------------------------------------------------------|------|
| Estructura de contenidos.....                           | 2    |
| Mapa de contenido .....                                 | 4    |
| Desarrollo de contenidos.....                           | 5    |
| 1. Generalidades de MySQL. ....                         | 5    |
| 1.1. Componentes de MySQL. ....                         | 5    |
| 1.2 Motores de almacenamiento de datos. ....            | 6    |
| 1.3. La consola de MySQL. ....                          | 7    |
| 1.4 Tipos de datos. ....                                | 8    |
| 3. Creación de la estructura de almacenamiento.....     | 14   |
| 3.1. Creación de la base de datos.....                  | 15   |
| 3.2. Creación de las tablas.....                        | 16   |
| 4. Modificación de la estructura de las tablas.....     | 21   |
| 5. Actualización e inserción de registros. ....         | 25   |
| 5.1. Inserción de registros. ....                       | 25   |
| 5.1.2. Inserción de registros con llaves foráneas. .... | 27   |
| 6. Consultas de registros.....                          | 29   |
| 6.1. Datos de prueba.....                               | 29   |
| 6.2. Consultas básicas. ....                            | 30   |
| 6.3. Aplicación de filtros a las consultas.....         | 33   |
| 6.4. Ordenamiento de las consultas.....                 | 34   |
| 6.5. Introducción a las consultas multi-tablas.....     | 35   |
| 6.6. Subconsultas.....                                  | 37   |
| Glosario .....                                          | 40   |
| Bibliografía.....                                       | 41   |
| Control del documento .....                             | 42   |



# INTRODUCCIÓN A LA BASE DE DATOS MySQL

## INTRODUCCIÓN

MySQL es un sistema manejador de bases de datos de libre uso y distribución bajo licencia GPL de los más utilizados y que está disponible para varios sistemas operativos (DUBOIS, 2009).

Su popularidad se debe principalmente a su licencia libre y a su facilidad de uso y administración. Por otra parte ha sido integrada con otras herramientas libres como son Linux, Apache, PHP, entre otras. Esta combinación e integración de tecnologías dio nombre a la plataforma de desarrollo conocida como LAMP (Linux, Apache, MySQL y PHP).

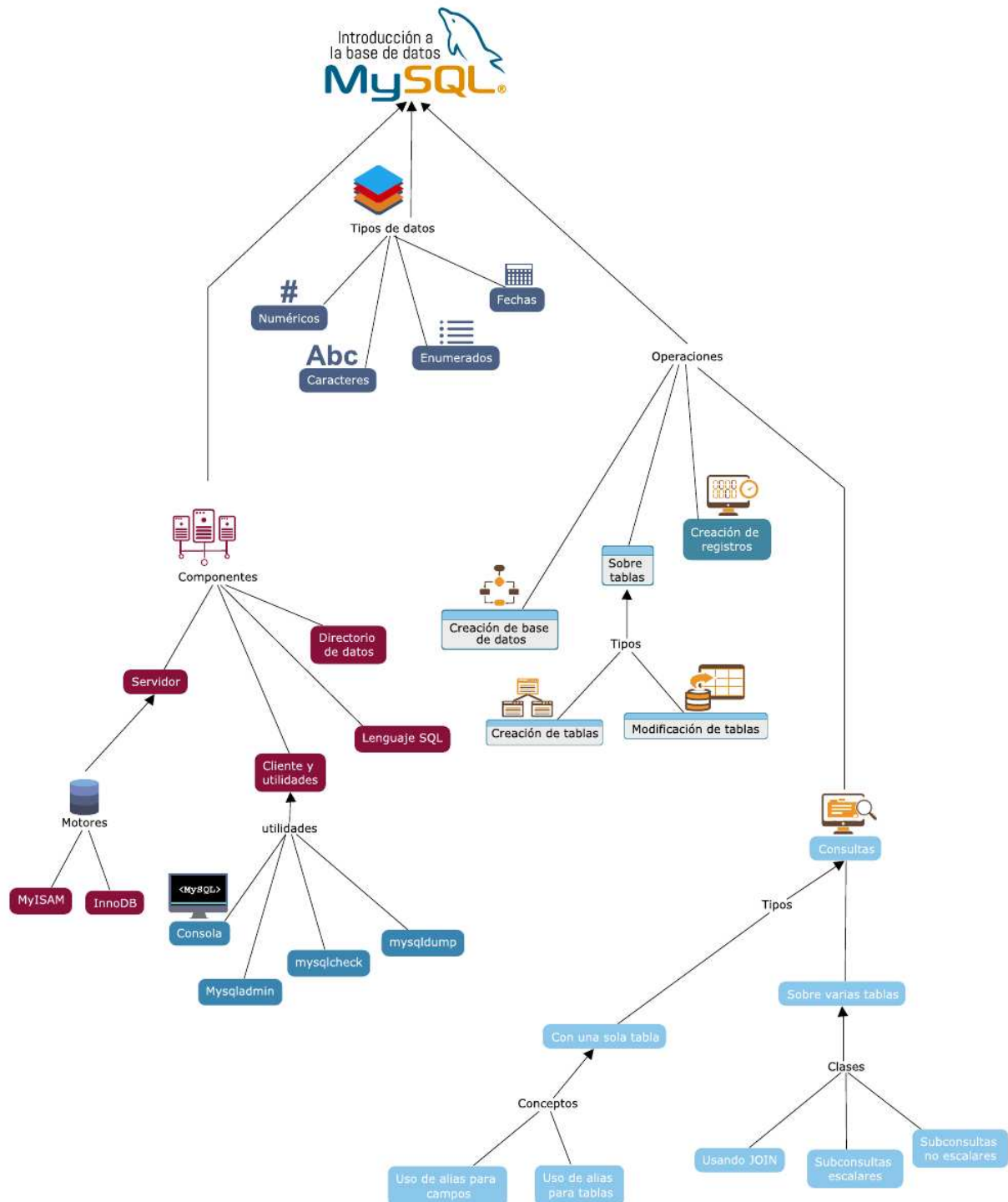
En años recientes MySQL fue adquirida por Sun Microsystems que luego fue adquirida por Oracle. Lo anterior no significa que MySQL deje de ser libre ya que su desarrollo está bajo la licencia GPL. Sin embargo, Oracle ofrece otras versiones de MySQL orientadas a empresas con modelo de licenciamiento de pago.

Para una mejor comprensión del recurso es necesario que el aprendiz haya estudiado los recursos de introducción al lenguaje SQL de la actividad de proyecto 6.

Para el desarrollo de este recurso se requiere que se tenga instalado el MySQL en el computador del aprendiz. Para apoyar la instalación existe un video tutorial en el área de materiales de la actividad de proyecto 6.



## MAPA DE CONTENIDO



## DESARROLLO DE CONTENIDOS

### 1. Generalidades de MySQL.

MySQL como manejador de bases de datos (SGBD) presenta las siguientes características (DUBOIS,2009):



#### Velocidad

MySQL es veloz comparado con la mayoría de las bases libres.



#### Portabilidad

MySQL corre en muchos sistemas operativos entre ellos windows, linux, unix.



#### Facilidad de uso

MySQL es de alto desempeño pero a la vez fácil de usar.



#### Conectividad

MySQL soporta distintos esquemas de conectividad y las bases de datos pueden ser accedidas desde cualquier sitio de Internet.



#### Soporta el lenguaje SQL

MySQL soporta el lenguaje estructurado de consultas (SQL).



#### Seguridad

MySQL maneja esquemas de seguridad que permiten asignar permisos a nivel de usuario.

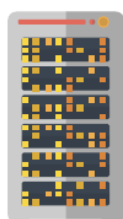


#### Robustez

El servidor MySQL es multi-hilo y puede atender varios usuarios de manera simultánea.

#### 1.1. Componentes de MySQL.

Los componentes de MySQL se pueden dividir en cuatro (DUBOIS, 2009):



##### *El servidor.*

El programa principal que corre como un servicio tanto en plataformas Linux como windows y es quien coordina todas las operaciones del manejador.

### *El cliente y las utilidades.*



MySQL suministra varias utilidades a saber:

- **MySQL:** Un programa interactivo que permite enviar consultas SQL al servidor. Requiere se puede llamar la consola.
- **mysqladmin:** Un programa administrativo para realizar tareas como subir y bajar el servidor, cambiar la configuración del servidor, monitorear el estado del mismo, entre otras.
- **mysqldump:** herramienta para hacer copias de seguridad o copiar datos a otros servidores.
- **mysqlcheck:** herramienta para realizar el chequeo, análisis y optimización de la base de datos.



### *Lenguaje SQL del servidor.*

Implementación del lenguaje estándar SQL dentro del servidor.



### *El directorio de datos.*

Es la ubicación donde se almacenan las bases de datos gestionadas por MySQL.

## **1.2 Motores de almacenamiento de datos.**

Muchas de las características de las bases de datos están provistas por el motor de almacenamiento. MySQL trae varios motores pero los más usados son MyISAM e InnoDB (DUBOIS,2009).

### **1.2.1. Motor MyISAM.**

Este es el motor por defecto usado por MySQL y representa cada tabla por medio de tres archivos en el sistema de archivos. Cada archivo tiene un nombre base que es el mismo que el de la tabla y una extensión que indica su función. Por ejemplo para una tabla llamada “clientes” existen los siguientes archivos:

- a. **clientes.frm:** es el archivo de formato que contiene la definición de la estructura de la tabla.
- b. **clientes.MYD:** es el archivo que contiene los datos de la tabla.
- c. **clientes.MYI:** es el archivo que contiene los índices de la tabla.

### 1.2.2. Motor InnoDB.

Este motor de almacenamiento está pensado para bases de datos transaccionales, es decir, que realizan operaciones que requieren ser tratadas como transacciones tipo CRUD (en inglés ACID).

Almacena las tablas en el sistema operativo de la siguiente manera:

- Un archivo .frm:** cada tabla de InnoDB es representada por un archivo .frm que contiene la definición de la estructura.
- Tablespace compartido:** consiste en uno o más archivos que contienen todos los datos de las tablas de manera contigua. Por defecto InnoDB almacena los datos de manera contigua.
- Tablespace individual:** se puede configurar InnoDB para que una tabla tenga un tablespace separado.

### 1.3. La consola de MySQL.

La consola es la utilidad que permite transmitir las instrucciones al servidor MySQL.

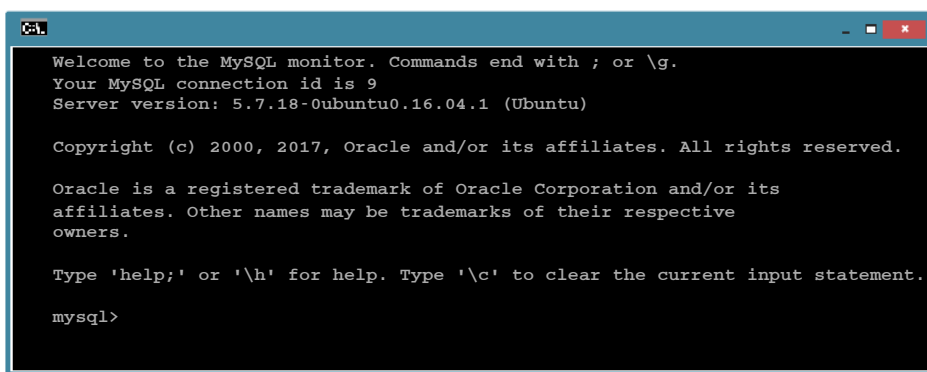
Para invocarla se usa la siguiente sintaxis:

```
$ mysql -h <nombre_del_host> -p -u <nombre_de_usuario>
```

Donde:

- h <nombre del host>: indica a cual servidor o host conectarse.
- u <nombre\_de\_usuario>: indica el nombre de usuario a conectar.
- p: indica al MySQL que pida el password del usuario.

Una vez ejecutado el comando aparece lo siguiente en pantalla:



```
mysql> Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.7.18-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Figura 1.1. Consola de MySQL.

## 1.4 Tipos de datos.

### 1.4.1. Tipos Numéricos.

MySQL soporta todos los tipos de datos SQL numéricos estándar: los tipos de datos enteros y los tipos de datos en coma flotante.

#### Tipos de datos enteros.

| TIPO                   | Bytes                                                 | Valor Mínimo<br>(Con signo/Sin signo) | Valor Máximo<br>(Con signo/Sin signo)       |
|------------------------|-------------------------------------------------------|---------------------------------------|---------------------------------------------|
| TINYINT                | 1                                                     | -128<br>0                             | 127<br>255                                  |
| BIT<br>(BOOL, BOOLEAN) | Número entero con valor 0 o 1. Sinónimo de TINYINT(1) |                                       |                                             |
| SMALLINT               | 2                                                     | -32768<br>0                           | 32767<br>65535                              |
| MEDIUMINT              | 3                                                     | -8388608<br>0                         | 8388607<br>16777215                         |
| INT                    | 4                                                     | -2147483648<br>0                      | 2147483647<br>4294967295                    |
| BIGINT                 | 8                                                     | -9223372036854775808<br>0             | 9223372036854775807<br>18446744073709551615 |

Tabla 1.1. Tipos de datos enteros.

#### Tipos de datos en coma flotante.

| Tipo             | Tamaño                                                                                                                                                                                                                                                 |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| FLOAT (m,d)      | Contiene un número en coma flotante de precisión sencilla. El valor M es la anchura a mostrar y D es el número de decimales.                                                                                                                           |
| DOUBLE (m,d)     | Contiene un número en coma flotante de precisión doble. Igual que FLOAT la diferencia es el rango de valores posibles.                                                                                                                                 |
| DECIMAL (m [,d]) | Se usan para guardar valores para los que es importante preservar una precisión exacta, por ejemplo con datos monetarios. Ejemplo: salario DECIMAL(5,2)<br>Si se omite D el valor por defecto es 0, los valores no tendrán punto decimal ni decimales. |

Tabla 1.2. Tipos de datos en coma flotante.

**Ejemplo:**

Si en la base de Datos se requiere almacenar las edades de las personas, cuyo valor máximo será 100, la opción más adecuada para el tipo de dato sería “TINYINT”.

Si se requiere sistematizar las notas de un colegio y la nota definitiva se debe dar en decimales y el valor máximo es 10.00, la opción más adecuada es “FLOAT”.

**1.4.2. Tipos de cadenas de caracteres.**

Listado de cada uno de los tipos de dato con formato string en MySQL, su ocupación en disco y valores.

| Tipo                     | Tamaño                                                                                                                                                                                                                                 | Sintaxis                      |
|--------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|
| CHAR (M)                 | Los valores válidos para M son de 0 a 255 caracteres.<br>Contiene una cadena de longitud constante.<br>Para mantener la longitud de la cadena, se rellena a la derecha con espacios. Estos espacios se eliminan al recuperar el valor. | PacIdentificacion<br>CHAR(10) |
| VARCHAR (M)              | Los valores válidos para M son de 0 a 255 caracteres.<br>Contiene una cadena de longitud variable.<br>Los espacios al guardar se eliminan.                                                                                             | PacNombres<br>VARCHAR(50)     |
| BLOB                     | Una longitud máxima de 65.535 caracteres.<br>Válido para objetos binarios como imágenes, documentos de texto, audio o video.                                                                                                           | PacImagenFoto BLOB            |
| TEXT                     | Una longitud máxima de 65.535 caracteres.<br>Sirve para almacenar texto plano sin formato.<br>Distingue entre minúsculas y mayúsculas.                                                                                                 | PacDescripcion TEXT           |
| TINYBLOB<br>TINYTEXT     | Longitud máxima de 255 caracteres.                                                                                                                                                                                                     |                               |
| MEDIUMBLOB<br>MEDIUMTEXT | Longitud máxima de 16777215 caracteres                                                                                                                                                                                                 |                               |
| LOB<br>LONGTEXT          | Longitud máxima de 4294967296 caracteres.                                                                                                                                                                                              |                               |

|      |                                                                                                                                                   |                             |
|------|---------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------|
| SET  | Contiene un conjunto.                                                                                                                             | SET('valor1','valor2',...)  |
|      | Un objeto de tipo cadena que puede tener cero o más valores, cada uno de los cuales debe estar entre una lista de valores 'valor1', 'valor2', ... |                             |
| ENUM | Contiene un enumerado.                                                                                                                            | ENUM('valor1','valor2',...) |
|      | Un objeto de tipo cadena que puede tener un único valor, entre una lista de valores 'valor1', 'valor2', ...,                                      |                             |

*Tabla 1.3. Tipos de datos de caracteres.*

En tabla se observa que el campo PacIdentificación se declaró como un “CHAR” de 10 caracteres porque los documentos de identidad tienen entre 1 y 10 caracteres.

Para almacenar nombres, direcciones e información con máximo de 100 caracteres se recomienda el tipo “VARCHAR”. En la tabla anterior el campo PacNombres se declaró varchar(50) porque los nombres tienen una cantidad de caracteres variable.

Para almacenar grandes cantidades de caracteres como descripciones, observaciones, comentarios en este caso se tomaría TEXT o BLOB.

### 1.4.3. Tipos de fecha y hora.

Los tipos de fecha y hora para representar valores temporales son:

| TIPO      | RANGO                                                                                                      | FORMATO             |
|-----------|------------------------------------------------------------------------------------------------------------|---------------------|
| DATE      | Válido para almacenar una fecha con año, mes y día.<br>Su rango oscila entre: '1000-01-01' y '9999-12-31'. | AAAA-MM-DD          |
| DATETIME  | Almacena una fecha y una hora.<br>Su rango oscila entre '1000-01-01 00:00:00' y '9999-12-31 23:59:59'.     | AAAA-MM-DD HH:MM:SS |
| TIME      | Una hora.<br>El rango está entre '-838:59:59' y '838:59:59'.                                               | HH:MM:SS            |
| TIMESTAMP | Almacena una fecha y hora UTC.<br>El rango está entre '1970-01-01 00:00:00' y algún momento del año 2037.  | AAAA-MM-DD HH:MM:SS |



YEAR (2|4) Almacena un año dado con 2 ó 4 dígitos de AA ó AAAA

longitud (por defecto son 4).  
El rango de valores oscila entre 1901 y 2155 con 4 dígitos. Mientras que con 2 dígitos el rango es desde 1970 a 2069 (70-69).

*Tabla 1.4. Tipos de datos fecha.*

### Ejemplo:

Para llevar el control de acceso (con horas, minutos y segundos) de los usuarios a un sistema de Información lo recomendable es tener un campo de tipo "DATETIME".

## 1.4. Modificadores.

Además de los tipos de datos requiere es necesario conocer algunos modificadores que se utilizan para el manejo de los campos. Estos modificadores se presentan a continuación:

| MODIFICADOR    | USO                                                                            | TIPO DE CAMPO QUE APLICA  |
|----------------|--------------------------------------------------------------------------------|---------------------------|
| AUTO_INCREMENT | El valor se va incrementando automáticamente en cada registro (1,2,3,etc).     | Enteros                   |
| DEFAULT        | Coloca un valor por defecto (el valor se coloca justo detrás de esta palabra). | Todos excepto TEXT y BLOB |
| NOT NULL       | Impide que un campo sea nulo.                                                  | Todos                     |
| PRIMARY KEY    | Hace que el campo se considere llave primaria.                                 | Todos                     |
| UNIQUE         | 65565 bytes. Evita la repetición de valores.                                   | Todos                     |

*Tabla 1.5. Modificadores.*

## 2. Base de datos didáctica.

En este recurso se presenta cómo hacer uso del motor de Base de Datos MySQL para crear una base de datos. Se utilizará para esto la base de datos "Citas" compuesta por cinco tablas: Pacientes, Medicos, Consultorios, Citas y Tratamientos, como se presenta en la figura 2.1.

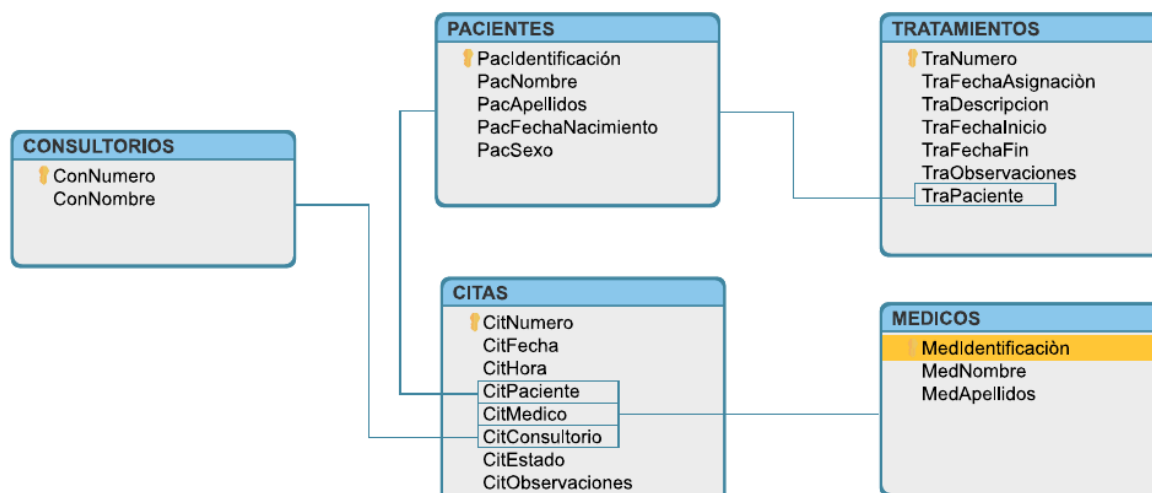


Figura 2.1. Diagrama relacional de la base de datos "Citas".

Se considera importante presentar la estructura de la base de datos detallando las tablas, tipos de datos de los campos y modificadores a utilizar. Con el fin de que pueda proceder a su creación usando SQL como Lenguaje de Definición de Datos (DDL).

## 2.1. Tabla "Pacientes".

| Tabla "Pacientes"  |                                              |       |                       |                       |
|--------------------|----------------------------------------------|-------|-----------------------|-----------------------|
| Atributo - Campo   | Tipo de Dato                                 | Long. | Modificador           | Tabla y Campo Foráneo |
| PacIdentificación  | Char                                         | 10    | Primary key, not null |                       |
| PacNombres         | Varchar                                      | 50    | not null              |                       |
| PacApellidos       | Varchar                                      | 50    | not null              |                       |
| PacFechaNacimiento | Date                                         |       | not null              |                       |
| PacSexo            | Por tener el Modificador enum, no se declara |       | (ENUM('M','F'))       |                       |

Figura 2.2. Definición de la tabla Pacientes.

## 2.2. Tabla “Medicos”.

| Tabla “Médicos”   |              |       |                       |               |
|-------------------|--------------|-------|-----------------------|---------------|
| Atributo - Campo  | Tipo de Dato | Long. | Modificador           | Tabla y Campo |
| MedIdentificacion | Char         | 10    | Primary key, not null |               |
| MedNombres        | Varchar      | 50    | not null              |               |
| MedApellidos      | Varchar      | 50    | not null              |               |

## 2.3. Tabla “Consultorios”.

| Tabla “Consultorios” |              |       |                       |                       |
|----------------------|--------------|-------|-----------------------|-----------------------|
| Atributo - Campo     | Tipo de Dato | Long. | Modificador           | Tabla y Campo Foráneo |
| ConNumero            | Int          | 3     | Primary key, not null |                       |
| ConNombre            | Varchar      | 50    | not null              |                       |

Figura 2.4. Definición de la tabla Consultorios.

## 2.4. Tabla “Tratamientos”.

| Tabla “Tratamientos” |              |       |                                      |                                   |
|----------------------|--------------|-------|--------------------------------------|-----------------------------------|
| Atributo - Campo     | Tipo de Dato | Long. | Modificador                          | Tabla y Campo Foráneo             |
| TraNumero            | Int          |       | Primary key, not null auto_increment |                                   |
| TraFechaAsignado     | Date         |       | not null                             |                                   |
| TraDescripcion       | Text         |       | not null                             |                                   |
| TraFechaInicio       | Date         |       | Primer key, not null                 |                                   |
| TraFechaFin          | Text         |       | not null                             |                                   |
| TraPacientes         | Char         | 10    | not null                             | Pacientes (PatientIdentificacion) |

Figura 2.5. Definición de la tabla Tratamientos.

## 2.5. Tabla “Citas”.

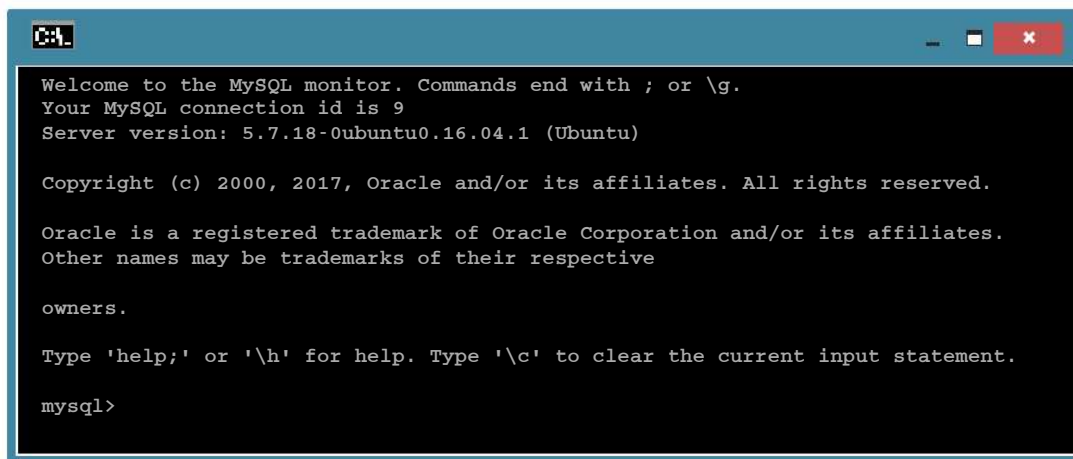
| Tabla “Citas”    |                                              |       |                                                     |                                   |
|------------------|----------------------------------------------|-------|-----------------------------------------------------|-----------------------------------|
| Atributo - Campo | Tipo de Dato                                 | Long. | Modificador                                         | Tabla y Campo Foráneo             |
| CitNumero        | Int                                          |       | Primary key, auto_increment                         |                                   |
| CitFecha         | Date                                         |       | not null                                            |                                   |
| CitHora          | Varchar                                      | 10    | not null                                            |                                   |
| CitPaciente      | Char                                         | 10    | not null                                            | Pacientes (PacientIdentificacion) |
| CitMedico        | Char                                         | 10    | not null                                            | Medicos(MedIdentificacion)        |
| CitConsultorio   | Int                                          |       | not null                                            | Consultorio (ConNumero)           |
| CitEstado        | Por tener el Modificador enum, no se declara |       | (ENUM("Asignada","Cumplida"))<br>DEFAULT "Asignada" |                                   |
| CitObservaciones | Text                                         |       | not null                                            |                                   |

Figura 2.6. Definición de la tabla Citas.

## 3. Creación de la estructura de almacenamiento.

Para iniciar el proceso de definición de datos en MySQL se debe realizar el siguiente procedimiento:

- Abrir el bloc de notas del equipo para digitar cada una de las instrucciones del Lenguaje de Definición de Datos, esto con el fin de ir construyendo el script completo de creación de la base de datos.
- Iniciar la consola de MySQL como se explica en videotutorial sobre instalación de MySQL.



```

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 9
Server version: 5.7.18-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates.
Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
  
```

Figura 3.1. Consola MySQL lista para recibir instrucciones.

c) Escribir o pegar las instrucciones en la consola. Las instrucciones deben terminar con punto y coma para que se ejecuten.

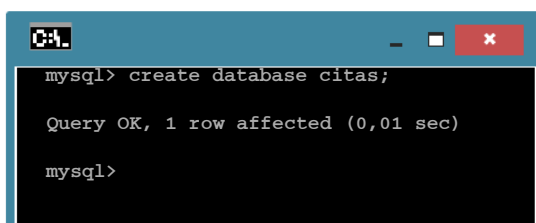
### 3.1 Creación de la base de datos.

Como el servicio ya está inicializado, se crea la Base de Datos, para nuestro ejemplo, CITAS.

La sintaxis de la instrucción es:

```
mysql> create database nombre_basedatos;
```

Para crear la base de datos “citas” se procede así:



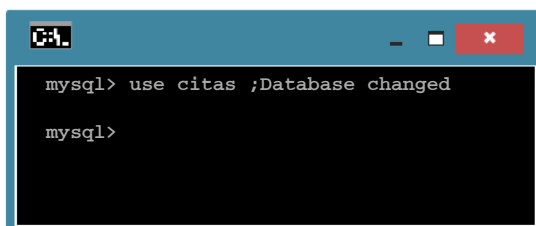
```
C:\>
mysql> create database citas;
Query OK, 1 row affected (0,01 sec)
mysql>
```

Figura 3.2. SQL para la creación de la base de datos.

Para crear las tablas debemos seleccionar “citas” como base de datos predeterminada. La instrucción es:

```
mysql> use nombre_base_de_datos ;
```

Para este ejemplo se tiene:

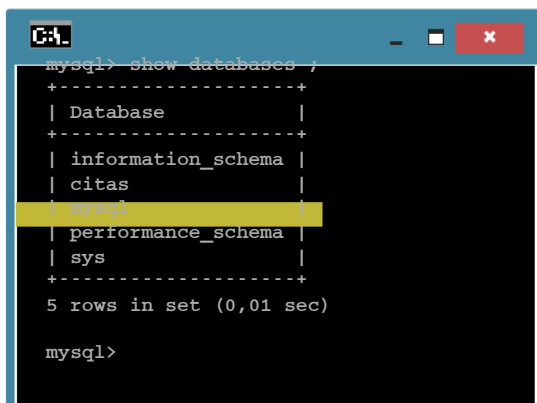


```
C:\>
mysql> use citas ;Database changed
mysql>
```

Figura 3.3. SQL para seleccionar una base de datos.

Para verificar que la base de datos fue creada se usa el siguiente comando:

```
mysql> show databases ;
```



```
mysql> show databases ;
+-----+
| Database |
+-----+
| information_schema |
| citas |
| mysql |
| performance_schema |
| sys |
+-----+
5 rows in set (0,01 sec)

mysql>
```

Figura 3.4. SQL para mostrar las bases de datos creadas.

En la pantalla anterior se puede observar que aparece la base de datos “citas”.

### 3.2 Creación de las tablas.

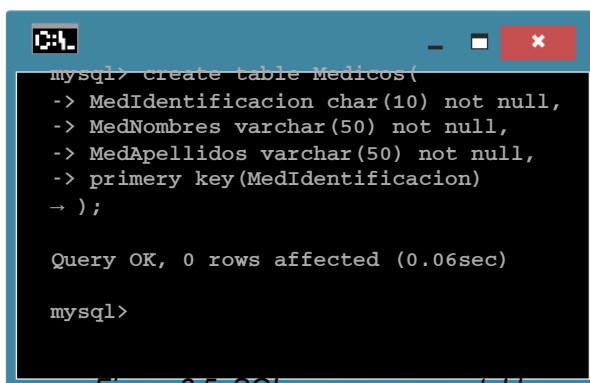
Una vez determinada “citas” como base de datos predeterminada, se procede a crear las respectivas tablas.

La sintaxis para la creación de tablas es la siguiente:

```
mysql> create table nombreTabla (
nombrecampo1 tipodatos(tamaño) modificador,
nombrecampo2 tipodatos(tamaño) modificador,
...
primary key (nombrecampo1)
);
```

#### 3.2.1 Creación de la tabla “Medicos”.

Para la creación de la tabla “Medicos” se procede con el siguiente comando:



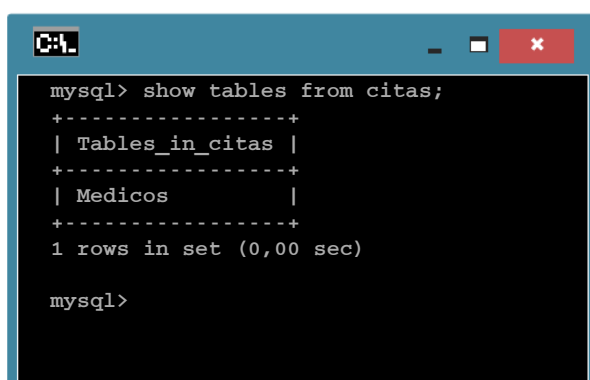
```
mysql> create table Medicos(  
-> MedIdentificacion char(10) not null,  
-> MedNombres varchar(50) not null,  
-> MedApellidos varchar(50) not null,  
-> primary key(MedIdentificacion)  
-> );  
  
Query OK, 0 rows affected (0.06sec)  
  
mysql>
```

Figura 3.5. SQL para crear una tabla.

Una vez creada se verifica que la tabla aparezca en la base de datos con el siguiente comando:

```
mysql> show tables from base_de_datos ;
```

Al ejecutar el comando en la consola de MySQL muestra los siguiente:



```
mysql> show tables from citas;  
+-----+  
| Tables_in_citas |  
+-----+  
| Medicos          |  
+-----+  
1 rows in set (0,00 sec)  
  
mysql>
```

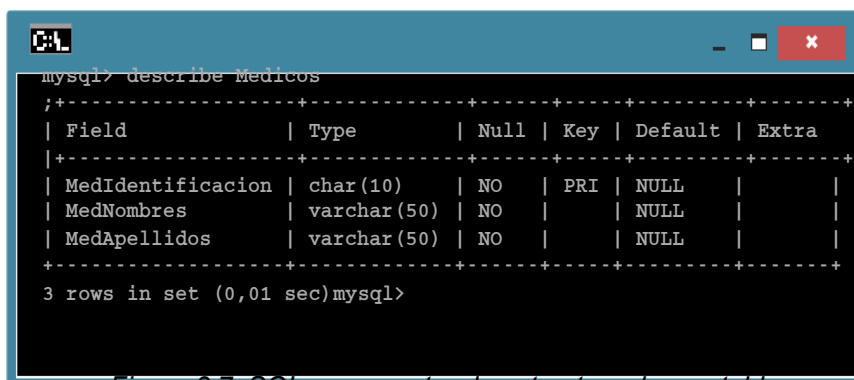
Figura 3.6. SQL para mostrar las tablas creadas en una base de datos.

Se puede observar que aparece la table “Medicos” en el listado.

Para verificar que la estructura de la tabla se usa el comando:

```
mysql> describe nombre_de_tabla ;
```

Para verificar la tabla “Medicos” se procede como se muestra a continuación:

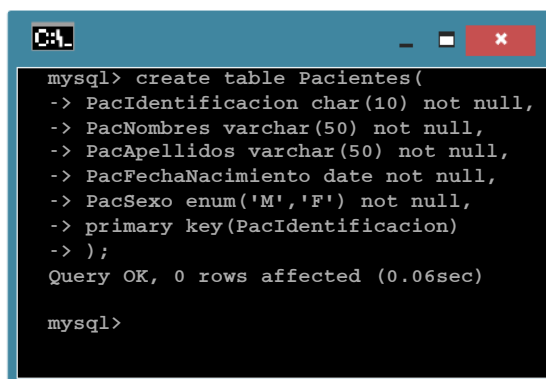


```
mysql> describe Medicos
+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+
| MedIdentificacion | char(10)      | NO   | PRI | NULL    |       |
| MedNombres       | varchar(50)   | NO   |     | NULL    |       |
| MedApellidos     | varchar(50)   | NO   |     | NULL    |       |
+-----+
3 rows in set (0,01 sec)mysql>
```

Figura 3.7. SQL para mostrar la estructura de una tabla.

### 3.2.2 Creación de la tabla “Pacientes”.

Para crear la tabla “Pacientes” se procede con el siguiente comando:

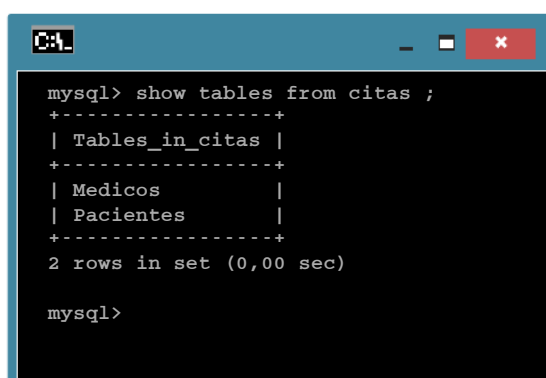


```
mysql> create table Pacientes(
-> PacIdentificacion char(10) not null,
-> PacNombres varchar(50) not null,
-> PacApellidos varchar(50) not null,
-> PacFechaNacimiento date not null,
-> PacSexo enum('M','F') not null,
-> primary key(PacIdentificacion)
-> );
Query OK, 0 rows affected (0.06sec)

mysql>
```

Figura 3.8. SQL para crear una tabla.

Se puede verificar que la tabla quedó creada con el siguiente comando:



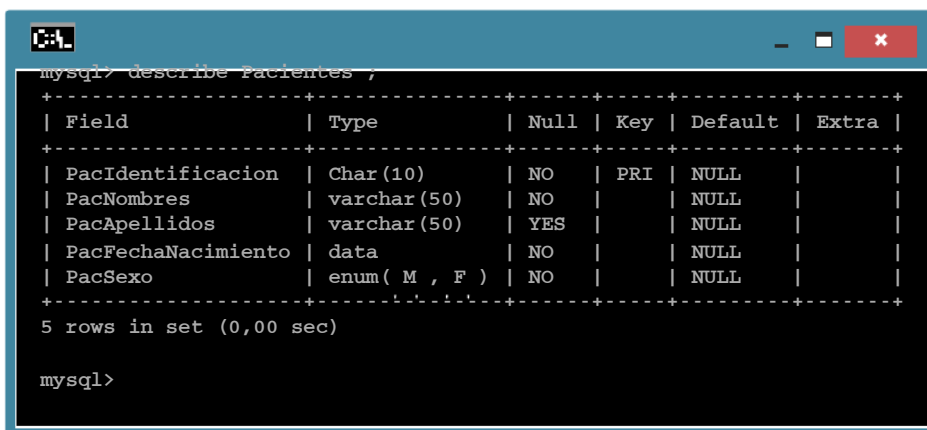
```
mysql> show tables from citas ;
+-----+
| Tables_in_citas |
+-----+
| Medicos          |
| Pacientes        |
+-----+
2 rows in set (0,00 sec)

mysql>
```

Figura 3.9. SQL para mostrar las tablas de una base de datos.



Para verificar la estructura de la tabla se procede con el siguiente comando:



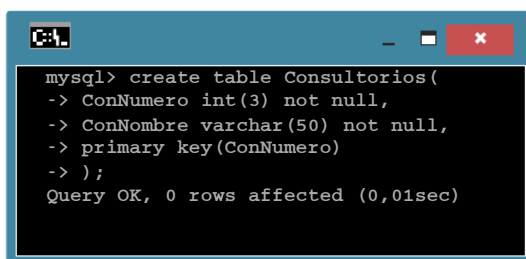
```
mysql> describe Pacientes ;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| PacIdentificacion | Char(10)      | NO   | PRI | NULL    |       |
| PacNombres       | varchar(50)   | NO   |     | NULL    |       |
| PacApellidos     | varchar(50)   | YES  |     | NULL    |       |
| PacFechaNacimiento | data          | NO   |     | NULL    |       |
| PacSexo          | enum( M , F ) | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
5 rows in set (0,00 sec)

mysql>
```

Figura 3.10. SQL para conocer la estructura de una tabla.

### 3.2.3. Creación tabla “Consultorios”.

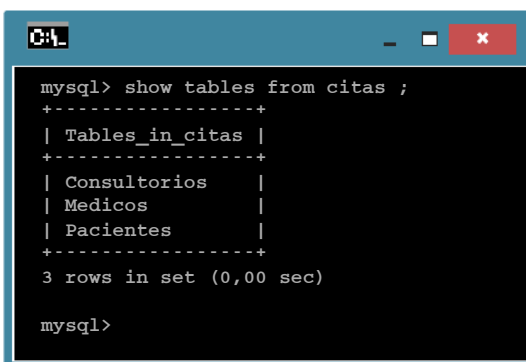
Para crear la tabla “Consultorios” se usa el siguiente comando:



```
mysql> create table Consultorios(
-> ConNumero int(3) not null,
-> ConNombre varchar(50) not null,
-> primary key(ConNumero)
-> );
Query OK, 0 rows affected (0,01sec)
```

Figura 3.11. SQL para crear una tabla.

Para verificar que la tabla haya sido creada se procede así:

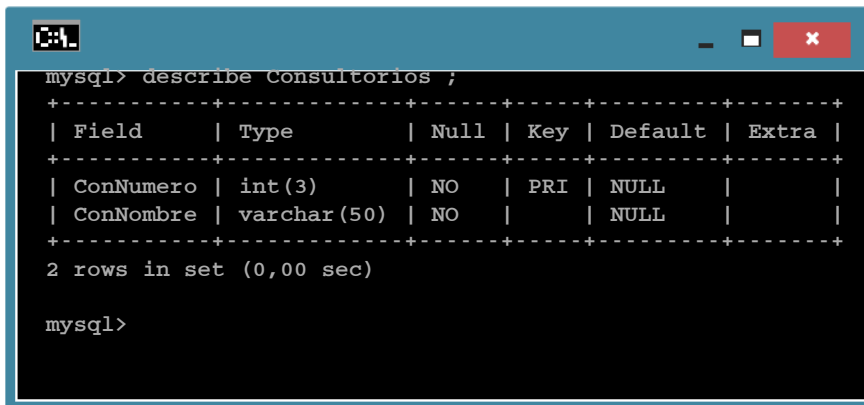


```
mysql> show tables from citas ;
+-----+
| Tables_in_citas |
+-----+
| Consultorios     |
| Medicos          |
| Pacientes        |
+-----+
3 rows in set (0,00 sec)

mysql>
```

Figura 3.12. SQL para mostrar las tablas de una base de datos.

Para verificar la estructura de la tabla se usa el siguiente comando:



```
mysql> describe Consultorios ;
+-----+-----+-----+-----+-----+-----+
| Field      | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| ConNumero  | int(3)        | NO   | PRI | NULL    |       |
| ConNombre  | varchar(50)   | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0,00 sec)

mysql>
```

Figura 3.13. SQL para mostrar la estructura de una tabla.

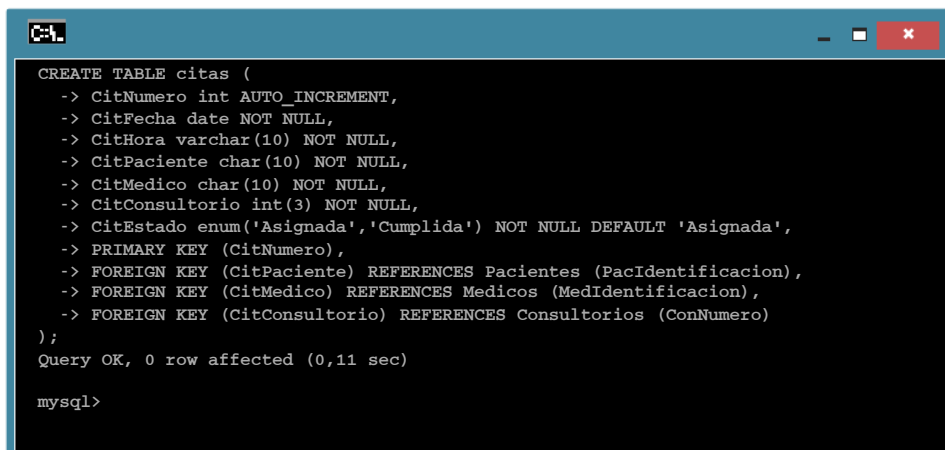
### 3.2.4. Creación tabla “Citas”.

Al revisar la estructura de la tabla “Citas” definida en un numeral anterior se puede observar que esta tabla tiene unos modificadores nuevos. Además esta tabla contiene llaves foráneas, es decir, está relacionada con otras tablas.

La sintaxis para la creación de una llave foránea es la siguiente:

```
foreign key(nombre_de_campo) references tabla(campo_tabla)
```

Al aplicar la sintaxis anterior a la creación de la tabla “Citas” se tiene lo siguiente:



```
CREATE TABLE citas (
  -> CitNumero int AUTO_INCREMENT,
  -> CitFecha date NOT NULL,
  -> CitHora varchar(10) NOT NULL,
  -> CitPaciente char(10) NOT NULL,
  -> CitMedico char(10) NOT NULL,
  -> CitConsultorio int(3) NOT NULL,
  -> CitEstado enum('Asignada','Cumplida') NOT NULL DEFAULT 'Asignada',
  -> PRIMARY KEY (CitNumero),
  -> FOREIGN KEY (CitPaciente) REFERENCES Pacientes (PacIdentificacion),
  -> FOREIGN KEY (CitMedico) REFERENCES Medicos (MedIdentificacion),
  -> FOREIGN KEY (CitConsultorio) REFERENCES Consultorios (ConNumero)
);
Query OK, 0 row affected (0,11 sec)

mysql>
```

Figura 3.14. SQL para crear una tabla.

## 4. Modificación de la estructura de las tablas.

Para modificar la estructura de una tabla se usa el siguiente comando:

```
mysql> alter table <nombre tabla> <atributo> <opciones> ;
```

La lista de los atributos o comandos es la siguiente:

| ATRIBUTOS | FUNCIÓN                                                                                                                                       | SINTAXIS                                                                                                                                                                              |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| MODIFY    | Modifica un campo, esto puede ser el tipo de dato y/o el ancho del campo.<br><br>También se utiliza para agregar y/o eliminar modificaciones. | alter table nombre_tabla MODIFY COLUMN nombre_columna modificador;<br><br>Para realizar este cambio se deben incluir los modificadores que tiene el campo y los que se van a agregar. |
| CHANGE    | Modifica el nombre de un campo.                                                                                                               | alter table nombre_tabla CHAGE COLUMN nombre_columna_actual nuevo_nombre_columna tipo_dato ancho_dato modificador(es);                                                                |
| RENAME    | Cambia el nombre de una tabla.                                                                                                                | alter table personas rename clientes;                                                                                                                                                 |
| DROP      | Borra Columnas.                                                                                                                               | alter table tabla DROP COLUMN ncolumnaABorrar;                                                                                                                                        |
| ADD       | Adiciona Columnas a una Tabla, es importante informar la ubicación de la nueva columna. Para ello se utiliza AFTER.                           | alter table nombre_tabla ADD nueva_Columna Tipo_dato Modificador AFTER Nombre_columna;                                                                                                |

Figura 4.1. Atributos para modificar la estructura de una tabla.

Se debe tener cuidado al momento de cambiar la estructura de una tabla ya que estos cambios pueden afectar toda la base de datos. El MySQL verifica cada cambio antes de aplicar. Si los cambios rompen las reglas de integridad el motor de base de datos no dejará aplicarlos y en su lugar emitirá un código y mensaje de error.

Para practicar esta instrucción se va a crear la tabla “Tratamientos” con una estructura inicial a la cual posteriormente se le introducirán unos cambios para llegar a una estructura deseada.

La estructura inicial de la tabla es la siguiente:

| Campo            | Tipo de campo | Longitud | Modificador                   |
|------------------|---------------|----------|-------------------------------|
| Tranumero        | Int           |          | Primary Key                   |
| TraFechaAsignado | Date          |          | Not null                      |
| Descripción      | Text          |          | Not null                      |
| TraFechaInicio   | Varchar       | 10       | Not null                      |
| TraObservaciones | Text          |          | Not null                      |
| TraTemporal      | Varchar       | 2        |                               |
| TraPaciente      | Char          | 10       | Pacientes (PacIdentificacion) |

Figura 4.2. Estructura de la tabla Tratamientos.

Para crearla se usa el siguiente comando:

```
mysql> create table Tratamientos(
-> Tranumero int,
-> TraFechaAsignado date not null,
-> Descripcion text not null,
-> TraFechaInicio varchar(10);
-> TraObservaciones text not null,
-> TraTemporal varchar(2);
-> TraPaciente char(10) not null,
-> primary key( Tranumero ),
-> foreign key ( TraPaciente ) references Pacientes ( PacIdentificacion )
-> );

Query OK, 0 rows affected (0,14 sec)

mysql>
```

Figura 4.3. SQL para crear una tabla.

Para verificar su estructura se usa el comando “describe” así:

```
mysql> describe Consultorios ;
+-----+-----+-----+-----+-----+-----+
| Field          | Type          | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| Tranumero      | int(11)       | NO   | PRI | NULL    |      |
| TraFechaAsignado | date          | NO   |     | NULL    |      |
| Descripcion     | text          | NO   |     | NULL    |      |
| TraFechaInicio | varchar(10)   | YES  |     | NULL    |      |
| TraObservaciones | text          | NO   |     | NULL    |      |
| TraTemporal     | varchar(2)    | NO   |     | NULL    |      |
| TraPaciente     | varchar(10)   | NO   | MUL | NULL    |      |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0,00 sec)

mysql>
```

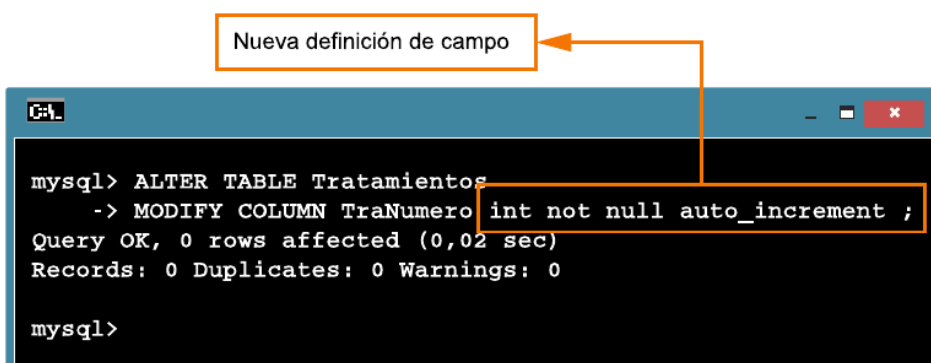
Figura 4.4. SQL para mostrar la estructura de una tabla.

Los cambios que se necesitan aplicar para llegar a la estructura requerida son los siguientes:

| No | Descripción                                                                                                            |
|----|------------------------------------------------------------------------------------------------------------------------|
| 1  | Se requiere que el campo "TraNumero" sea auto-incremental.                                                             |
| 2  | Se requiere que el campo "Descripcion" sea renombrado a "TraDescripcion" para seguir el estándar de nombres de campos. |
| 3  | Cambia el tipo de datos del campo "TraFechaInicio" al tipo "date".                                                     |
| 4  | Agregar el campo "TraFechaFin" de tipo "date" después de la fecha de inicio.                                           |

Figura 4.5. Lista de las modificaciones a realizar a la tabla *Tratamientos*.

1) Para aplicar el primer cambio: "Se requiere que el campo *TraNumero* sea autoincremental" se utiliza el siguiente comando:

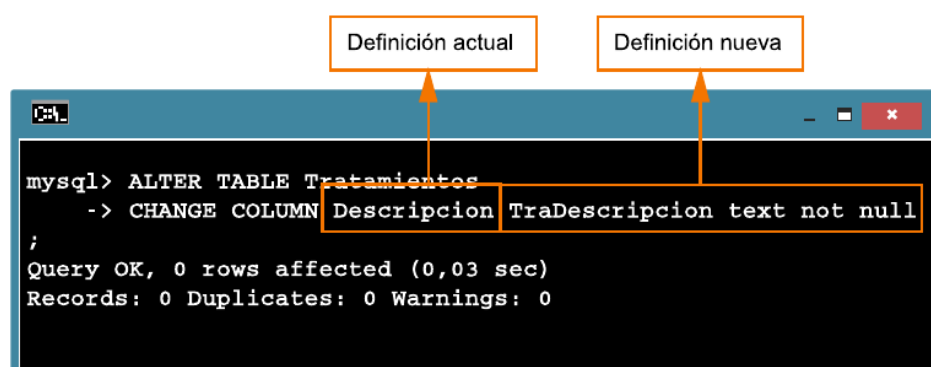


```
mysql> ALTER TABLE Tratamientos
-> MODIFY COLUMN TraNumero int not null auto_increment ;
Query OK, 0 rows affected (0,02 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
```

Figura 4.6. Descripción de la instrucción *Alter table modify column*.

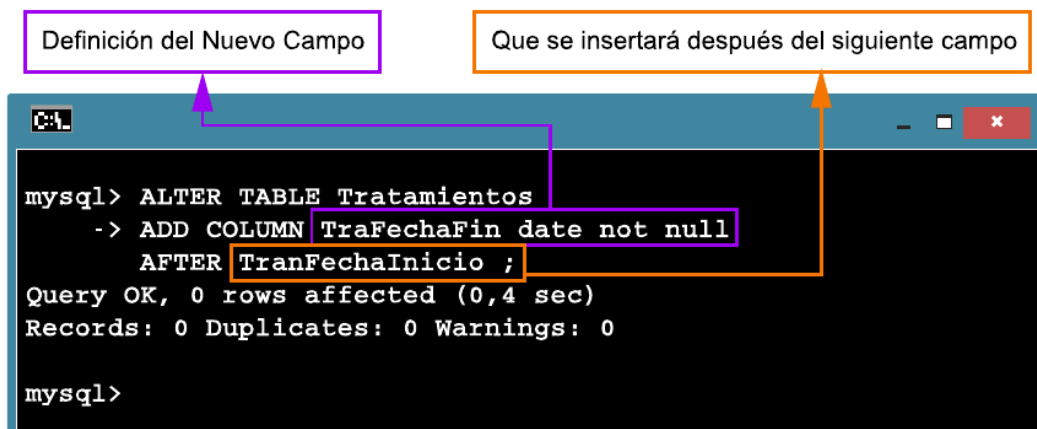
2) Para aplicar el segundo cambio: se requiere que el campo *Descripcion* sea renombrado a *TraDescripcion* se utiliza el comando "change" como se describe a continuación:



```
mysql> ALTER TABLE Tratamientos
-> CHANGE COLUMN Descripcion TraDescripcion text not null
;
Query OK, 0 rows affected (0,03 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Figura 4.7. Descripción de la instrucción *Alter table change column*.

3) Para aplicar el tercer cambio: “Cambiar el tipo de dato del campo “TraFechaInicio” al tipo date “ se utiliza el comando “modify” como se describe a continuación:

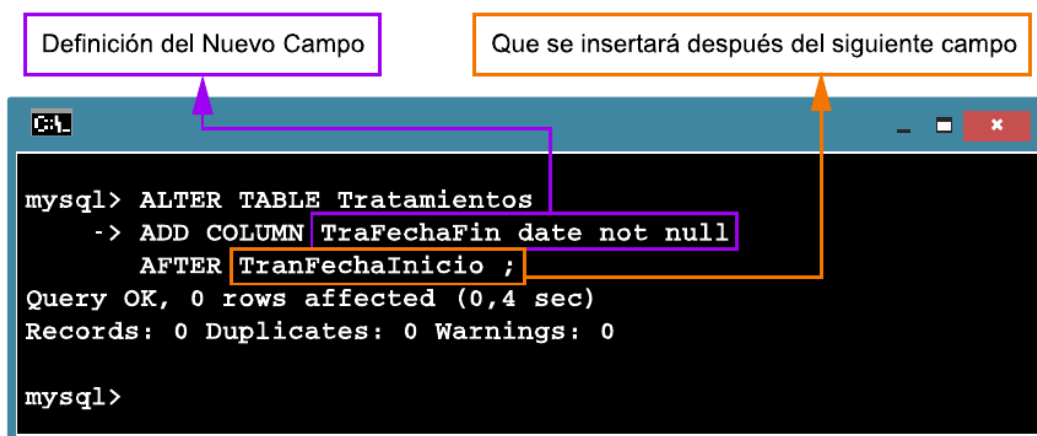


```
mysql> ALTER TABLE Tratamientos
-> ADD COLUMN TraFechaFin date not null
AFTER TranFechaInicio ;
Query OK, 0 rows affected (0,4 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
```

Figura 4.8. Descripción de la instrucción Alter table modify column.

4) Para aplicar el cuarto cambio: “Agregar el campo TraFechaFin de tipo date después de la fecha de inicio.” se aplica el siguiente comando:



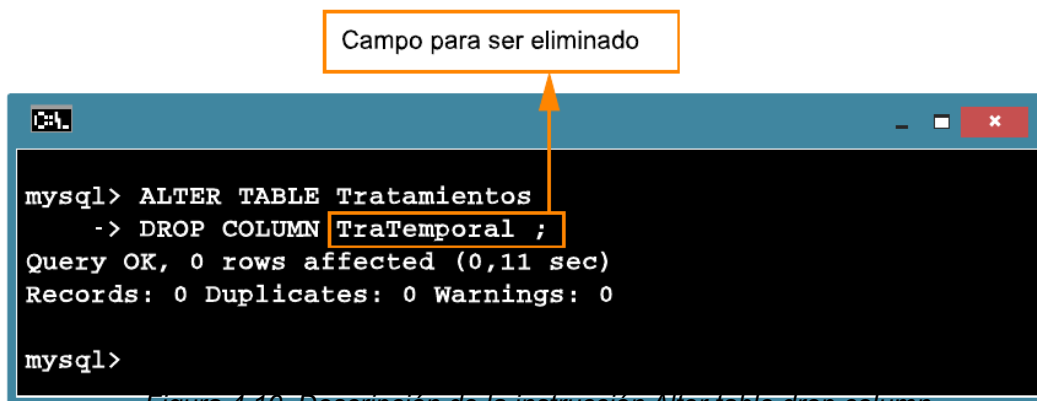
```
mysql> ALTER TABLE Tratamientos
-> ADD COLUMN TraFechaFin date not null
AFTER TranFechaInicio ;
Query OK, 0 rows affected (0,4 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
```

Figura 4.9. Descripción de la instrucción Alter table add column.

5) Para aplicar el quinto cambio: “Eliminar el campo TraTemporal” se procede con el comando “drop column” como se describe a continuación:

Campo para ser eliminado

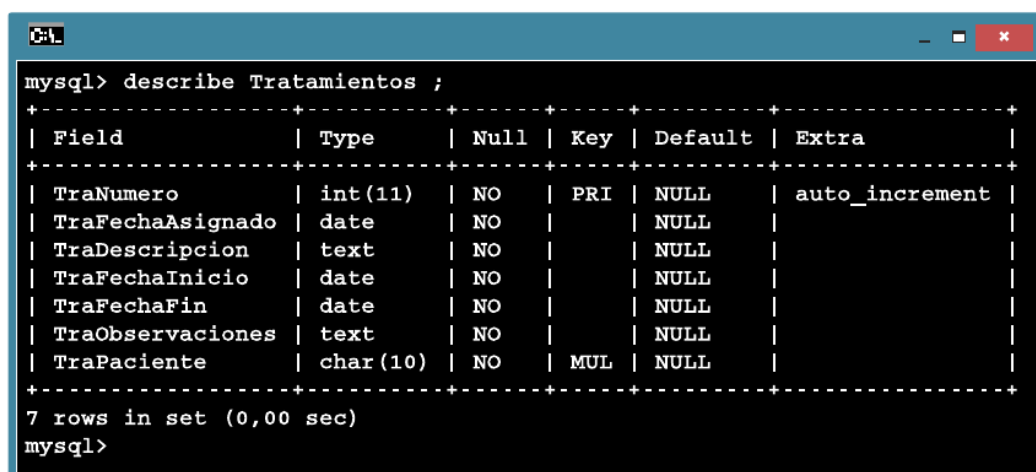


```
mysql> ALTER TABLE Tratamientos
-> DROP COLUMN TraTemporal ;
Query OK, 0 rows affected (0,11 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
```

Figura 4.10. Descripción de la instrucción Alter table drop column.

Por último se verifica la nueva estructura de la tabla usando el comando “describe” como se muestra a continuación:



```
mysql> describe Tratamientos ;
+-----+-----+-----+-----+-----+-----+
| Field          | Type   | Null | Key | Default | Extra           |
+-----+-----+-----+-----+-----+-----+
| TraNumero      | int(11)| NO   | PRI | NULL    | auto_increment |
| TraFechaAsignado | date   | NO   |     | NULL    |                 |
| TraDescripcion  | text   | NO   |     | NULL    |                 |
| TraFechaInicio  | date   | NO   |     | NULL    |                 |
| TraFechaFin     | date   | NO   |     | NULL    |                 |
| TraObservaciones | text   | NO   |     | NULL    |                 |
| TraPaciente     | char(10)| NO   | MUL | NULL    |                 |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0,00 sec)

mysql>
```

Figura 4.11. Descripción de la tabla Tratamientos después de los cambios.

## 5. Actualización e inserción de registros.

### 5.1 Inserción de registros.

Para realizar las operaciones de Inserción de Registros se utiliza la sentencia “INSERT INTO” del lenguaje de manipulación de datos (D.M.L.).

El comando “INSERT INTO” permite incluir los datos en cada uno de los campos que se tienen en las tablas de la base de datos creada. Su sintaxis es:

```
mysql> INSERT INTO <nombre tabla> (<campo1>, <campo2>, ..., <campoN>)
VALUES (<valor1>, <valor2>, ..., <valorN>);
```

Para los establecer lo valores que tomarán los campos se debe tener en cuenta lo siguiente:

- a) Los campos tipo texto, char o varchar deben estar encerrados entre comillas. Ejemplo: ‘Gomez Rodriguez’, ‘Carrera 42 45 -11 Apto 1102’.
- b) Los campos tipo entero debe ir sin ningún cambio.
- c) Los campos tipo decimal o flotante: la parte entera debe estar separada de la parte decimal por un punto “.”. Lo anterior ya que la coma es usada como separador de campos.
- d) Los campos tipo fecha: pueden tomar varios tipos de formatos. El formato predeterminado es ‘AAAA-MM-DD’ donde AAAA es el año , MM el mes y DD el día. El valor debe además ir encerrado entre comillas. Por ejemplo el 13 de Julio de 2017 quedaría expresado como ‘2017-07-13’.

### 5.1.1 Inserción de registros sin llaves foráneas.

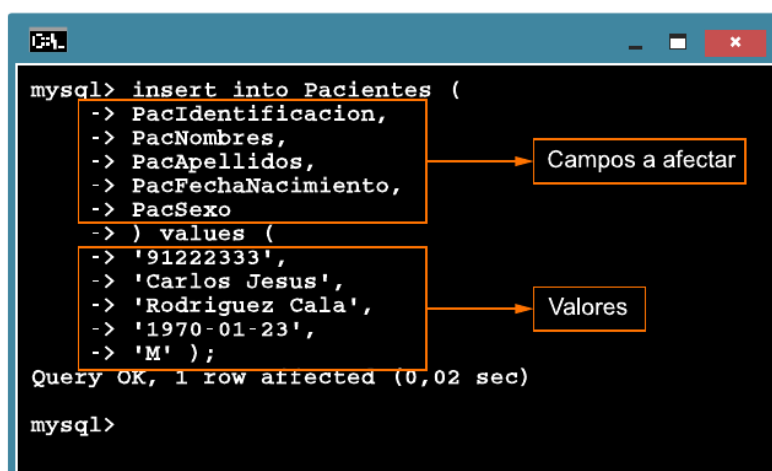
Para insertar el siguiente registro :

|                  |                     |
|------------------|---------------------|
| Identificación   | 91.222.333          |
| Nombres          | Carlos Jesús        |
| Apellidos        | Rodríguez Cala      |
| Fecha Nacimiento | 23 de Enero de 1970 |
| Sexo             | M                   |

Figura 5.1. Registro ejemplo a insertar.



Se utiliza el siguiente comando:



```
mysql> insert into Pacientes (
-> PacIdentificacion,
-> PacNombres,
-> PacApellidos,
-> PacFechaNacimiento,
-> PacSexo
-> ) values (
-> '91222333',
-> 'Carlos Jesus',
-> 'Rodriguez Cala',
-> '1970-01-23',
-> 'M' );
Query OK, 1 row affected (0,02 sec)
mysql>
```

Campos a afectar

Valores

Figura 5.2. Ejemplo del comando Insert into.

### 5.1.2 Inserción de registros con llaves foráneas.

Las llaves foráneas son reglas de integridad que se deben cumplir para que la base de datos sea consistente.

Para insertar registros en tablas que tienen llaves foráneas se debe identificar el valor de esas llaves para incluirlos en la sentencia de inserción.

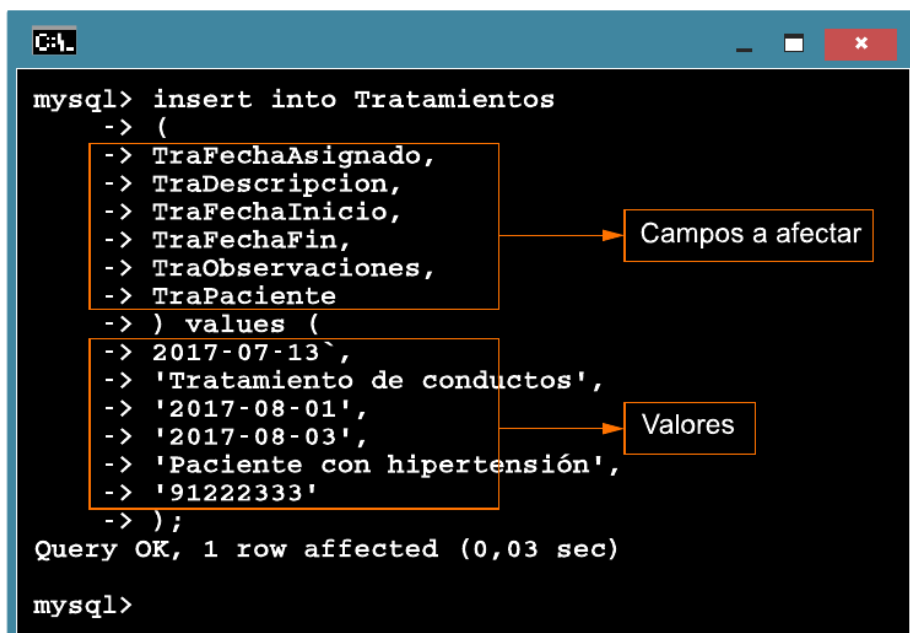
Por ejemplo, para insertar un registro en la tabla “Tratamientos” se debe determinar la llave primaria del paciente en la tabla “Pacientes”.

Para insertar el siguiente registro:

|                  |                           |
|------------------|---------------------------|
| TraFechaAsignado | “2017-07-13”              |
| TraDescripcion   | Tratamiento de Conductos  |
| TraFechaInicio   | “2017-08-01”              |
| TraFechaFin      | “2017-08-03”              |
| TraObservaciones | Paciente con hipertensión |
| TraPaciente      | 91.222.333                |

Figura 5.3. Ejemplo de registro con llaves foráneas a insertar.

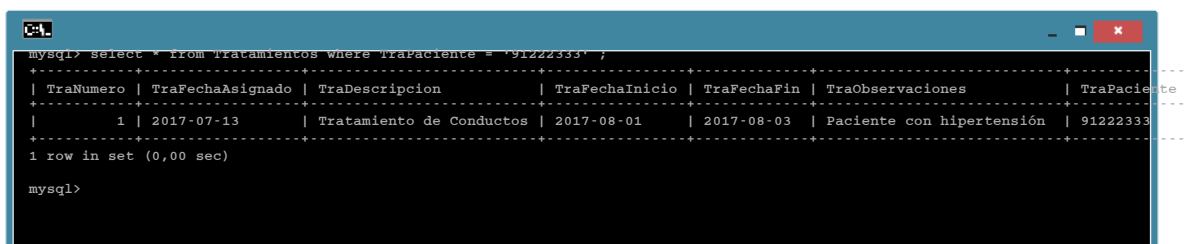
Se procede así:



```
mysql> insert into Tratamientos
-> (
-> TraFechaAsignado,
-> TraDescripcion,
-> TraFechaInicio,
-> TraFechaFin,
-> TraObservaciones,
-> TraPaciente
-> ) values (
-> '2017-07-13',
-> 'Tratamiento de conductos',
-> '2017-08-01',
-> '2017-08-03',
-> 'Paciente con hipertensión',
-> '91222333'
-> );
Query OK, 1 row affected (0,03 sec)
mysql>
```

Figura 5.4. Ejemplo de registro con llaves foráneas a insertar.

Para la inserción del anterior registro no se tuvo que definir el valor del campo “TraNumero” ya que fue definido como “auto\_increment”, es decir, que se auto incrementa con cada registro nuevo. Para verificar que el registro fue insertado se ejecuta una consulta como se muestra en la figura 5.5.



```
mysql> select * from Tratamientos where TraPaciente = '91222333';
+-----+-----+-----+-----+-----+-----+-----+
| TraNumero | TraFechaAsignado | TraDescripcion | TraFechaInicio | TraFechaFin | TraObservaciones | TraPaciente |
+-----+-----+-----+-----+-----+-----+-----+
| 1 | 2017-07-13 | Tratamiento de Conductos | 2017-08-01 | 2017-08-03 | Paciente con hipertensión | 91222333 |
+-----+-----+-----+-----+-----+-----+-----+
1 row in set (0,00 sec)
mysql>
```

Figura 5.5. Consulta para verificar registro creado.

## 6. Consultas de registros.

Para realizar las operaciones de Consulta de Registros se utiliza la sentencia SELECT del lenguaje de manipulación de datos (D.M.L.).

La sentencia SELECT permite visualizar la información de la Base de Datos, los datos que se presentan corresponden a una o más **columnas** de una tabla o requiere a una o más **columnas** de una o más tablas.

La sintaxis básica es:

```
SELECT <lista de campos >
FROM <tablas >
WHERE <condiciones>
GROUP BY <campos de agrupamiento>
HAVING <condiciones sobre los grupos >
ORDER BY <ordenamientos>
LIMIT <cantidad de registros a traer >
OFFSET <número de página >
```

### 6.1 Datos de prueba.

Para el ejemplo se procederá con los siguientes registros de prueba:

| IDENTIFICACIÓN | NOMBRES           | APELLIDOS      | FECHA NACIMIENTO | SEXO |
|----------------|-------------------|----------------|------------------|------|
| 29.234.333     | Alejandra Marcela | Díaz Granados  | 1980-03-24       | F    |
| 1098.343.678   | Juan Antonio      | Pérez Pereira  | 1978-08-09       | M    |
| 37.456.298     | Diana Marcela     | Estévez        | 1985-09-06       | F    |
| 51.890.654     | Adriana María     | Pataquiva      | 1990-12-24       | F    |
| 91.222.333     | Carlos Jesús      | Rodríguez Cala | 1970-01-23       | M    |

*Figura 6.1. Conjunto de registros de ejemplo.*

| TraFechaAsignado | TraDescripción           | TraFechaInicio | TraFechaFin | TraObservaciones          | TraPaciente |
|------------------|--------------------------|----------------|-------------|---------------------------|-------------|
| 2017-07-13       | Tratamiento de Conductos | 2017-08-01     | 2017-08-03  | Paciente con hipertensión | 91.222.333  |
| 2017-07-02       | Profilaxis               | 2017-07-02     | 2017-07-02  | Sin novedad               | 51.890.654  |
| 2017-06-05       | Resina                   | 2017-06-05     | 2017-06-05  | Paciente con sensibilidad | 37.456.298  |
| 2017-05-23       | Profilaxis               | 2017-05-23     | 2017-05-23  | Sin novedad               | 29.234.333  |

Figura 6.2. Conjunto de registros de ejemplo.

## 6.2 Consultas básicas.

Para hacer un ejemplo de consultas básicas se tomará la tabla pacientes. Esta consulta puede generarse de diferentes maneras dependiendo de los datos que se necesiten visualizar.

Si se quiere visualizar todos los registros y campos de la tabla se usa el asterisco (\*) el cual indica al intérprete de comandos que extraiga todos los campos.

La instrucción es:

```
foreign key(nombre_de_campo) references tabla(campo_tabla)
mysql> SELECT * FROM Pacientes ;
```

Al ejecutar el comando en la consola de MySQL aparece lo siguiente:  
Figura 6.3. SQL para hacer consultas a la base de datos.

Para limitar el número de registros se usa el comando "LIMIT x". Ejemplo:

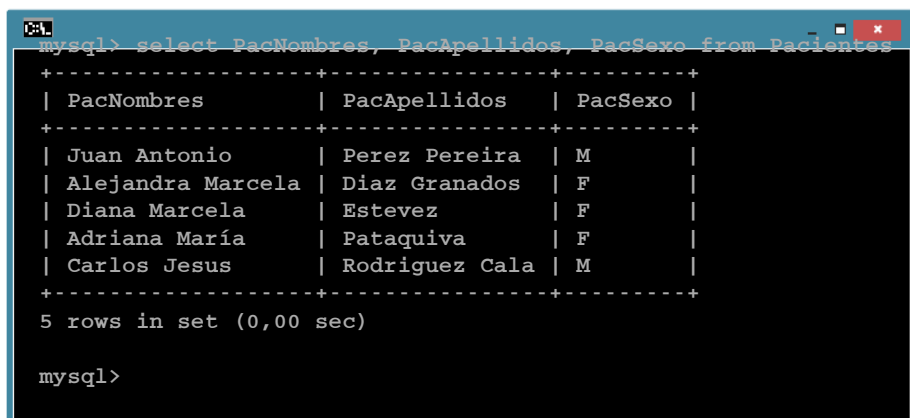


```
mysql> select * from Pacientes ;
+-----+-----+-----+-----+-----+
| PacIdentificacion | PacNombres      | PacApellidos  | PacFechaNacimiento | PacSexo |
+-----+-----+-----+-----+-----+
| 1098343678       | Juan Antonio    | Perez Pereira | 1978-08-09         | M       |
| 29234333         | Alejandra Marcela | Diaz Granados | 1980-03-24         | F       |
| 37456298         | Diana Marcela   | Estevez      | 1985-09-06         | F       |
| 51890654         | Adriana María   | Pataquiva    | 1990-12-24         | F       |
| 91222333         | Carlos Jesus    | Rodriguez Cala | 1970-01-23         | M       |
+-----+-----+-----+-----+-----+
5 rows in set (0,00 sec)mysql>
```

Figura 6.4. SQL para hacer consultas a la base de datos con límite.

La consulta anterior limita el resultado a máximo tres registros.

El asterisco indica al intérprete de comandos seleccionar todos los campos. Esto no es adecuado ya que puede generar consultas pesadas que consumen valiosos recursos del servidor. Para especificar los campos se procede así:



```
mysql> select PacNombres, PacApellidos, PacSexo from Pacientes;
+-----+-----+-----+
| PacNombres | PacApellidos | PacSexo |
+-----+-----+-----+
| Juan Antonio | Perez Pereira | M       |
| Alejandra Marcela | Diaz Granados | F       |
| Diana Marcela | Estevez       | F       |
| Adriana María | Pataquiva     | F       |
| Carlos Jesus  | Rodriguez Cala | M       |
+-----+-----+-----+
5 rows in set (0,00 sec)

mysql>
```

Figura 6.5. SQL para hacer consultas a la base de datos con límite de campos.

En la anterior consulta solo se trajeron los campos especificados.

Se pueden cambiar los nombres de los campos para que sean más entendibles para el usuario final a través de un “alias”.

La sintaxis para cuando se usan alias es la siguiente:

```
mysql> SELECT <nombre_campo> AS nombre_alterno_de_campo FROM <tabla>;
```

Para el ejemplo anterior se puede mejorar los nombres de los campos así:

```
mysql> select PacNombres as Nombres,
-> PacApellidos as Apellidos,
-> PacSexo as Sexo
-> from Pacientes ;

+-----+-----+-----+
| Nombres      | Apellidos      | Sexo |
+-----+-----+-----+
| Juan Antonio  | Perez Pereira  | M    |
| Alejandra Marcela | Diaz Granados | F    |
| Diana Marcela  | Estevez        | F    |
| Adriana María  | Pataquiva      | F    |
| Carlos Jesus   | Rodriguez Cala | M    |
+-----+-----+-----+

5 rows in set (0,00 sec)

mysql>
```

Figura 6.6. SQL para hacer consultas a la base de datos con alias de campos.

Las tablas también pueden tener un alias que será de gran utilidad cuando se realicen los JOIN o enlaces. La sintaxis es la siguiente:

```
mysql> SELECT <nombre_campo> FROM <tabla> [AS] nombre_alterno_de_tabla;
```

Para el ejemplo anterior se puede aplicar un alias a la tabla así:

```
mysql> select p.PacNombres, p.PacApellidos, p.PacSexo
-> from Pacientes as p ;

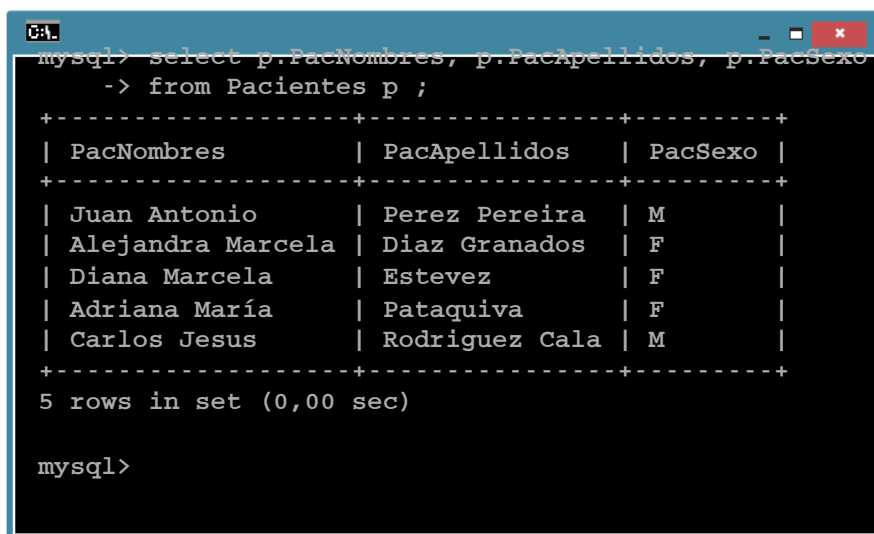
+-----+-----+-----+
| PacNombres      | PacApellidos      | PacSexo |
+-----+-----+-----+
| Juan Antonio    | Perez Pereira     | M       |
| Alejandra Marcela | Diaz Granados     | F       |
| Diana Marcela    | Estevez           | F       |
| Adriana María    | Pataquiva         | F       |
| Carlos Jesus     | Rodriguez Cala    | M       |
+-----+-----+-----+

5 rows in set (0,00 sec)

mysql>
```

Figura 6.7. SQL para hacer consultas a la base de datos con alias de campos y tablas.

El prefijo “AS” se puede omitir. Para el ejemplo requiere funcionar lo siguiente:



```
mysql> select p.PacNombres, p.PacApellidos, p.PacSexo
-> from Pacientes p ;

+-----+-----+-----+
| PacNombres      | PacApellidos  | PacSexo |
+-----+-----+-----+
| Juan Antonio    | Perez Pereira | M       |
| Alejandra Marcela | Diaz Granados | F       |
| Diana Marcela   | Estevez       | F       |
| Adriana María   | Pataquiva     | F       |
| Carlos Jesus    | Rodriguez Cala | M       |
+-----+-----+-----+
5 rows in set (0,00 sec)

mysql>
```

Figura 6.8. SQL para hacer consultas con alias de campos y tablas.

### 6.3 Aplicación de filtros a las consultas.

El filtro como su nombre lo indica limita el número de registros a mostrar a través de la incorporación de una o varias condiciones lógicas.

En el lenguaje SQL los filtros se implementan con la cláusula “WHERE” seguido de una o más preguntas lógicas. Estas preguntas se evalúan una a una y se aplican operaciones de matemática booleana para determinar el resultado lógico que siempre deberá ser “verdadero” o “falso”

La sintaxis es:

```
SELECT <campos>
FROM <tablas>
WHERE <bloque de condiciones>
```

Ejemplo, para seleccionar los pacientes de sexo femenino se procede con la siguiente consulta.

```
mysql> select PacNombres as Nombres,
PacApellidos as Apellidos, PacSexo as Sexo
-> from Pacientes
-> where PacSexo = 'F';
+-----+-----+-----+
| Nombres      | Apellidos      | Sexo |
+-----+-----+-----+
| Alejandra Marcela | Diaz Granados | F    |
| Diana Marcela    | Estevez        | F    |
| Adriana María    | Pataquiva      | F    |
+-----+-----+-----+
3 rows in set (0,00 sec)

mysql>
```

Figura 6.9. SQL para hacer consultas con alias de campos y tablas.

En el ejemplo anterior la pregunta “ PacSexo = ‘F’ “ se hace a cada registro de la tabla “Pacientes”. Aquellos registros en los cuales la pregunta es verdadera son extraídos.

Se pueden hacer múltiples preguntas apoyados en los operadores lógicos AND y OR.

Las preguntas se pueden agrupar con paréntesis. De la misma forma como se evalúan las expresiones aritméticas las expresiones booleanas encerradas en paréntesis se evalúan primero.

#### 6.4 Ordenamiento de las consultas.

Los resultados de las consultas se pueden ordenar en un orden específico con el comando “ORDER BY”.

La sintaxis es la siguiente:

```
SELECT <campos>
FROM <tablas>
WHERE <bloque de condiciones>
ORDER BY <lista de los campos separados por coma> ASC |
DESC
```

En el siguiente ejemplo la consulta es ordenada por dos campos: PacNombres y PacApellidos.



```
mysql> select PacNombres as Nombres,
PacApellidos as Apellidos, PacSexo as Sexo
-> from Pacientes
-> where PacSexo = 'F'
-> order by PacNombres, PacApellidos ;
+-----+-----+-----+
| Nombres      | Apellidos    | Sexo |
+-----+-----+-----+
| Adriana María | Pataquiva    | F    |
| Alejandra Marcela | Díaz Granados | F    |
| Diana Marcela | Estevez      | F    |
+-----+-----+-----+
3 rows in set (0,00 sec)

mysql>
```

Figura 6.10. SQL query con el filtro y ordenamiento.

Por defecto el ordenamiento es ascendente. Si se requiere en orden descendente se procede aplicando el modificador “DESC” así:

```
mysql> select PacNombres as Nombres, PacApellidos as Apellidos,
PacSexo as Sexo
-> from Pacientes
-> where PacSexo = 'F'
-> order by PacNombres DESC, PacApellidos DESC;
+-----+-----+-----+
| Nombres      | Apellidos    | Sexo |
+-----+-----+-----+
| Diana Marcela | Estevez      | F    |
| Alejandra Marcela | Díaz Granados | F    |
| Adriana María | Pataquiva    | F    |
+-----+-----+-----+
3 rows in set (0,00 sec)

mysql>
```

Figura 6.11. SQL query con el filtro y ordenamiento descendente.

## 6.5 Introducción a las consultas multi-tablas

En muchos casos se requiere que la consulta extraiga la información de varias tablas al

mismo tiempo.

En estos casos MySQL y en general el lenguaje SQL suministra la opción “JOIN”. La sintaxis es la siguiente:

```
SELECT <campos>
FROM <tabla1 alias t1 > JOIN <tabla2 alias t2 > ON <condición de unión>
WHERE <bloque de condiciones>
```

Ejemplo: se requiere listado con la cédula, nombres, apellidos y la descripción del tratamiento realizado por cada paciente.

| Identificación | Nombres      | Apellidos      | Tratamiento              | Observaciones             |
|----------------|--------------|----------------|--------------------------|---------------------------|
| 91.222..333    | Carlos Jesús | Rodríguez Cala | Tratamiento de conductos | Paciente con hipertensión |

Figura 6.12. Ejemplo para consultas multi-tablas.

Los campos nombres y apellidos están contenidos en la tabla “Pacientes”.

Los campos descripción y observaciones del tratamiento está contenido en la tabla “Tratamientos”.

La cédula es la llave primaria en la tabla “Pacientes” y es una llave foránea en la tabla “Tratamientos”.

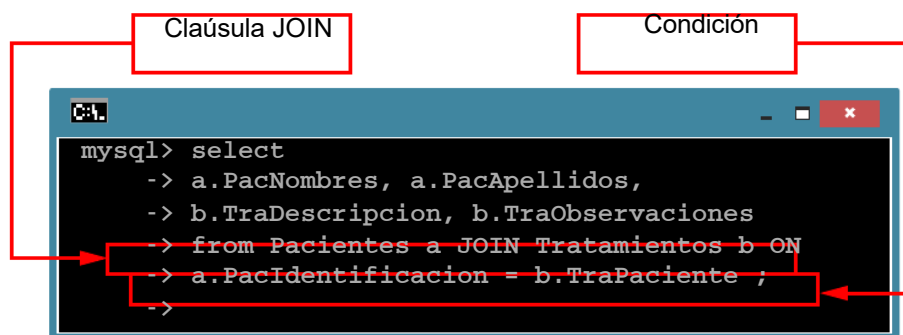


Figura 6.13. Ejemplo de una consulta con join.

Para hacer el enlace o “join” de las tablas se procede así:

El resultado es el siguiente:

```
mysql> select PacNombres, PacApellidos, TraDescripcion, TraObservaciones
from Pacientes a join Tratamientos b on
a.PacIdentificacion = b.TraPaciente;
+-----+-----+-----+-----+
| PacNombres | PacApellidos | TraDescripcion | TraObservaciones |
+-----+-----+-----+-----+
| Carlos Jesus | Rodriguez Cala | Tratamiento de Conductos | Paciente con hipertensión |
| Adriana María | Pataquiva | Profilaxis | Sin novedad |
| Diana Marcela | Estevez | Resina | Paciente con sensibilidad |
| Alejandra Marcela | Diaz Granados | Profilaxis | Sin novedad |
+-----+-----+-----+-----+
4 rows in set (0,01 sec)

mysql>
```

Figura 6.14. Resultado de la consulta con join.

El enlace o “join” genera una tabla a la cual se le pueden aplicar otros filtros. Para el ejemplo anterior se puede restringir la consulta sólo a los pacientes femeninos. En este caso el comando quedaría así:

```
mysql> select
-> a.PacNombres, a.PacApellidos,
-> b.TraDescripcion, b.TraObservaciones
-> from Pacientes a JOIN Tratamientos b ON
-> a.PacIdentificacion = b.TraPaciente
-> where a.PacSexo = 'F' ;
+-----+-----+-----+-----+
| PacNombres | PacApellidos | TraDescripcion | TraObservaciones |
+-----+-----+-----+-----+
| Alejandra Marcela | Diaz Granados | Profilaxis | Sin novedad |
| Diana Marcela | Estevez | Resina | Paciente con sensibilidad |
| Adriana María | Pataquiva | Profilaxis | Sin novedad |
+-----+-----+-----+-----+
3 rows in set (0,01 sec)

mysql>
```

Figura 6.15. Resultado de la consulta con join.

## 6.6. Subconsultas.

Las subconsultas son como su nombre lo indica una consulta que se lleva a cabo dentro de otra consulta. Esto se presenta cuando se requiere consultar un dato o varios datos que están en distintas tablas pero no se quiere o no se puede realizar un JOIN que sería la forma más organizada de realizar esta labor.

### 6.6.1. Subconsultas con resultados escalares.

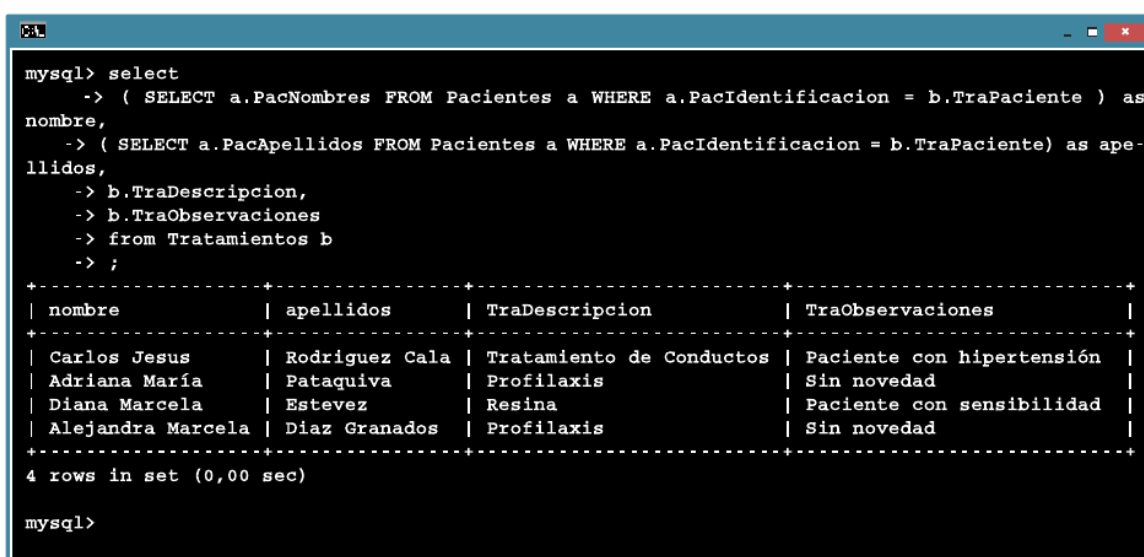
MySQL y en general el lenguaje SQL permite realizar subconsultas con la siguiente sintaxis:

```
SELECT * FROM tabla1 WHERE campo1 = (SELECT campo1 FROM t2);
```

Como se puede observar la subconsulta debe estar contenida entre un paréntesis. El lenguaje SQL primero resuelve la subconsulta y luego compara los resultados.

En este caso la subconsulta debe arrojar un único valor de lo contrario no se podrá realizar la comparación y el MySQL generará un error.

Ejemplo. Se puede recorrer la tabla Tratamientos y traer el nombre del paciente sin necesidad de hacer JOIN de la siguiente manera:



```
mysql> select
-> ( SELECT a.PacNombres FROM Pacientes a WHERE a.PacIdentificacion = b.TraPaciente ) as
nombre,
-> ( SELECT a.PacApellidos FROM Pacientes a WHERE a.PacIdentificacion = b.TraPaciente) as ape-
llidos,
-> b.TraDescripcion,
-> b.TraObservaciones
-> from Tratamientos b
-> ;
```

| nombre            | apellidos      | TraDescripcion           | TraObservaciones          |
|-------------------|----------------|--------------------------|---------------------------|
| Carlos Jesus      | Rodriguez Cala | Tratamiento de Conductos | Paciente con hipertensión |
| Adriana María     | Pataquiva      | Profilaxis               | Sin novedad               |
| Diana Marcela     | Estevez        | Resina                   | Paciente con sensibilidad |
| Alejandra Marcela | Diaz Granados  | Profilaxis               | Sin novedad               |

```
4 rows in set (0,00 sec)

mysql>
```

Figura 6.16. Ejemplo de una consulta con subconsultas.

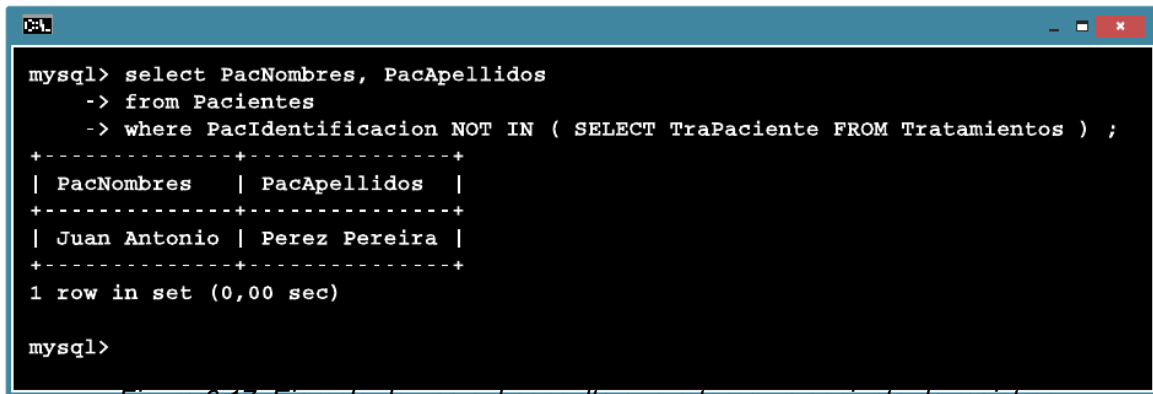
### 6.6.2. Subconsultas que retornan conjuntos de registros.

Las subconsultas pueden arrojar múltiples datos. En estos casos ya no se pueden comparar con los comparadores escalares como "=", ">", "<". Se deben comparar con operadores de conjuntos. La sintaxis es la siguiente:

```
SELECT * FROM tabla1 WHERE campo1 [NOT] IN (SELECT campo1 FROM t2);
```

Esta sintaxis indica que la cláusula WHERE es verdadera si el campo1 de la tabla1 está contenido en el resultado de la sub consulta.

Ejemplo. Se puede determinar cuales pacientes aún no tienen un tratamiento así:



```
mysql> select PacNombres, PacApellidos
-> from Pacientes
-> where PacIdentificacion NOT IN ( SELECT TraPaciente FROM Tratamientos ) ;
+-----+-----+
| PacNombres | PacApellidos |
+-----+-----+
| Juan Antonio | Perez Pereira |
+-----+-----+
1 row in set (0,00 sec)

mysql>
```

*Figura 6.17. Ejemplo de una subconsulta que retorna un conjunto de registros.*

Además del comparador IN existen otros comparadores como ANY, ALL, SOME.

## GLOSARIO

**Alias:** nombre alternativo que pueden tomar los campos de una tabla o las tablas mismas para efectos de simplificar o mejorar su descripción.

**Condición:** partes que en su conjunto forman un filtro.

**Consulta:** conjunto de instrucciones en lenguaje SQL que realizan una consulta a la base de datos.

**D.D.L.:** acrónimo de Data Definition Language. Subconjunto del lenguaje SQL utilizado para crear, modificar o borrar los componentes de las bases de datos.

**D.M.L.:** acrónimo de Data Manipulation Language. Subconjunto del lenguaje SQL que trata la forma con los registros son creados, actualizados, consultados o borrados.

**Filtro:** condición que limita la cantidad de registros en una consulta.

**GPL:** acrónimo de General Public License. Tipo de licenciamiento abierto para software.

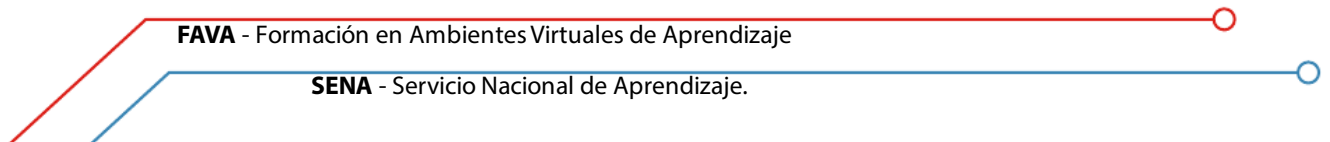
**Join:** unión entre una o más tablas.

**Llave primaria:** campo de una tabla cuyo valor identifica inequívocamente un registro.

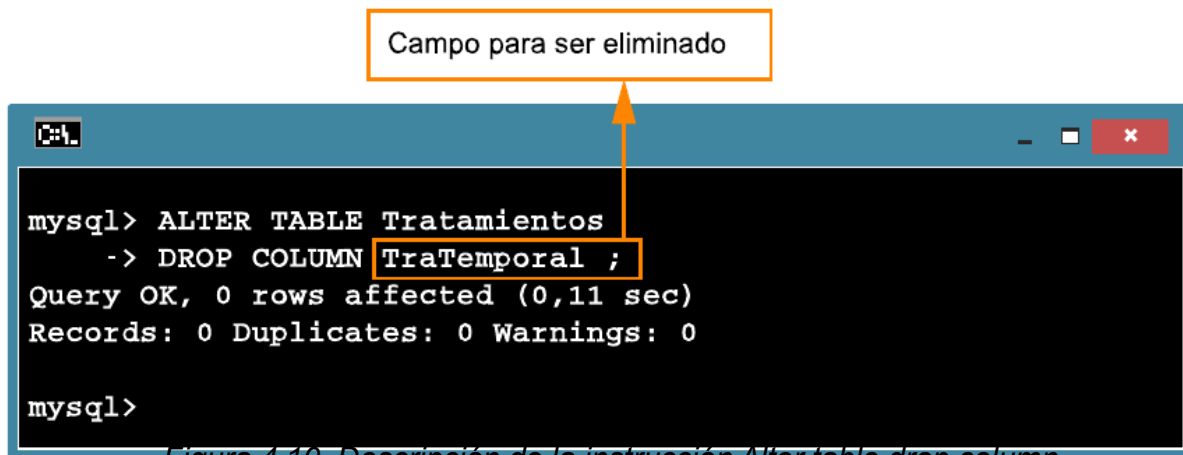
**Llave foránea:** campo de una tabla cuyo valor apunta a la llave primaria de otra tabla.

**SQL:** acrónimo de Structured Query Language. Lenguaje para el manejo de bases de datos.

5) Para aplicar el quinto cambio: “Eliminar el campo TraTemporal” se procede con el comando “drop column” como se describe a continuación:



Campo para ser eliminado

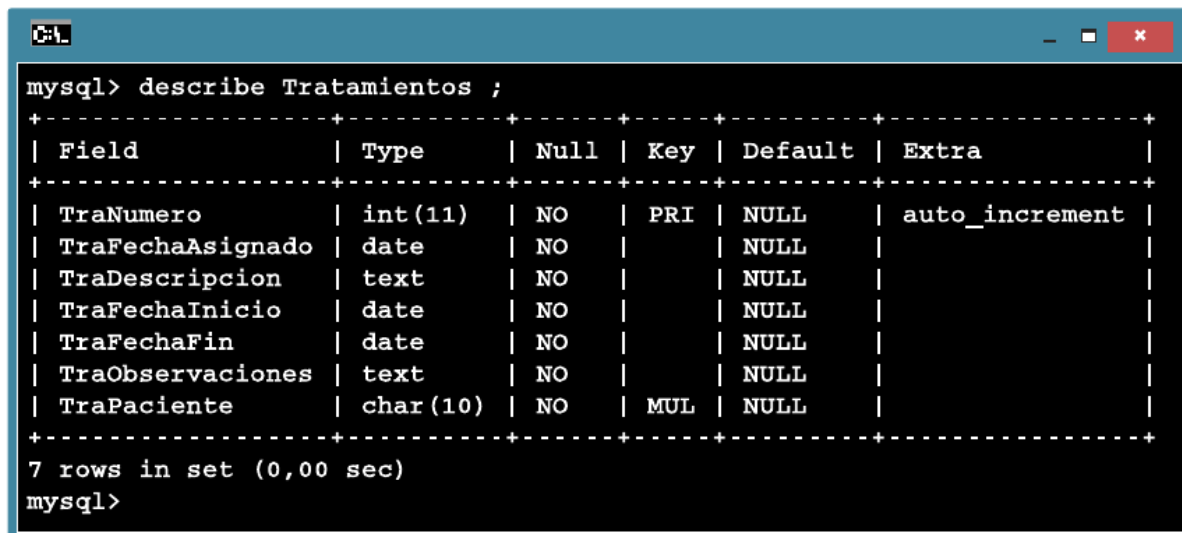


```
mysql> ALTER TABLE Tratamientos
-> DROP COLUMN TraTemporal ;
Query OK, 0 rows affected (0,11 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
```

Figura 4.10. Descripción de la instrucción Alter table drop column.

Por último se verifica la nueva estructura de la tabla usando el comando “describe” como se muestra a continuación:



```
mysql> describe Tratamientos ;
+-----+-----+-----+-----+-----+-----+
| Field          | Type      | Null | Key | Default | Extra          |
+-----+-----+-----+-----+-----+-----+
| TraNumero      | int(11)   | NO   | PRI | NULL    | auto_increment |
| TraFechaAsignado | date      | NO   |     | NULL    |                |
| TraDescripcion  | text      | NO   |     | NULL    |                |
| TraFechaInicio  | date      | NO   |     | NULL    |                |
| TraFechaFin     | date      | NO   |     | NULL    |                |
| TraObservaciones | text      | NO   |     | NULL    |                |
| TraPaciente     | char(10)  | NO   | MUL | NULL    |                |
+-----+-----+-----+-----+-----+-----+
7 rows in set (0,00 sec)

mysql>
```

Figura 4.11. Descripción de la tabla Tratamientos después de los cambios.

## 5. Actualización e inserción de registros.

### 5.1 Inserción de registros.

Para realizar las operaciones de Inserción de Registros se utiliza la sentencia “INSERT INTO” del lenguaje de manipulación de datos (D.M.L.).





● **videncia GA6-220501096-AA1-EV04 Elaboración de las bases de datos.**

Teniendo en cuenta el documento realizado en la evidencia 3 EV03 realizar la base de datos en mongoDb cumpliendo con las siguientes características.

**Elementos para tener en cuenta en el documento:**

- Cree una base de datos NoSQL.
- Cree una colección de datos llamada “parque”.
- Inserte 5 documentos con la estructura JSON Creada.
- Actualice los datos del primer y último registro.
- Liste la colección completa.
- Borre el tercer documento de la colección parque
- Liste la colección de datos completa.
- Creela sentencia que permita obtener un documento según el número de placa del documento.

**Lineamientos generales para la entrega de la evidencia:**

- **Productos para entregar:** Video la creación y manipulación de bases de datos NoSQL.
- **Extensión:** Mp4.

Para hacer el envío de la evidencia remítase al área de la actividad correspondiente y acceda al espacio:  
**Elaboración de las bases de datos- GA6-220501096-AA1-EV04**



#### FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO

|                       |                                  |
|-----------------------|----------------------------------|
| PROGRAMA DE FORMACIÓN | Implementación de bases de datos |
|-----------------------|----------------------------------|

|             |                                                                                           |                           |                                                                                                 |
|-------------|-------------------------------------------------------------------------------------------|---------------------------|-------------------------------------------------------------------------------------------------|
| COMPETENCIA | 2205011133 Administrar base de datos de acuerdo con los estándares y requisitos técnicos. | RESULTADOS DE APRENDIZAJE | 2205011133-01 Modelar la base de datos de acuerdo a los requerimientos del sistema informático. |
|-------------|-------------------------------------------------------------------------------------------|---------------------------|-------------------------------------------------------------------------------------------------|

|                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NÚMERO DEL COMPONENTE FORMATIVO | 4                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| NOMBRE DEL COMPONENTE FORMATIVO | Bases de Datos NoSQL                                                                                                                                                                                                                                                                                                                                                                                                                     |
| BREVE DESCRIPCIÓN               | En este componente se abordarán elementos claves del <b>Big Data</b> , administración masiva de datos, consulta de datos de las organizaciones y tipos de respuesta más cortos. El aprendiz conocerá cómo se plantean nuevas técnicas y tecnologías para atender esas nuevas necesidades: las <b>Bases de datos NoSQL</b> , orientadas a resolver problemas de escalabilidad y rendimiento, mediante nuevos entornos de manejo de datos. |
| PALABRAS CLAVE                  | Bases de datos, consulta interactiva, datos, documentos, escalabilidad                                                                                                                                                                                                                                                                                                                                                                   |

|                  |           |
|------------------|-----------|
| ÁREA OCUPACIONAL | No aplica |
| IDIOMA           | Español   |

#### A. TABLA DE CONTENIDOS:

##### Introducción

- 1. Introducción a NoSQL**
  - 1.1. Definición de NoSQL
  - 1.2. Como diferenciar NoSQL de SQL.
  - 1.3. Tipos de Bases de Datos NoSQL.
    - 1.3.1 Bases de datos Basadas en Clave Valor
    - 1.3.2 Bases de datos orientadas a documentos
    - 1.3.3 Bases de Datos Orientada a Grafos
  - 1.4. Sistemas de Gestión de Bases de Datos (SGDB) NoSQL.
- 2. NoSQL con MongoDB.**
  - 2.1. Conceptos Básicos.
  - 2.2. Instalación de MongoDB.
  - 2.3. Consola interactiva
  - 2.4. Crear Primer Documento.
- 3. Manipulación de datos con MogoDB**
  - 3.1. Crear Documentos
  - 3.2. Leer y Consultar colecciones.



3.3. Actualizar Documentos

3.4. Borrar Documentos.

## B. DESARROLLO DE CONTENIDOS:

### Introducción

El almacenamiento de información es importante en las empresas ya integradas en la era digital, debido a que, si se dispone de una cantidad significativa de datos. Los especialistas en análisis de datos elaboran estrategias que pueden ayudar a identificar mejor, las necesidades de clientes y prever riesgos o el estado de comportamiento del mercado. Es por esto que las empresas saben de la importancia de administrar tanta información de sus clientes y del mercado, como sea posible. Por tanto, aparecen nuevas necesidades para las cuales el Big Data que, hace referencia a la administración de enormes volúmenes de datos, enfoca sus esfuerzos en procesar información con características de gran volumen, dejando de lado otros aspectos de los datos que, en el caso de las bases de datos tradicionales se concentran más, como es la integridad o la no repetición de los datos.

### 1. Introducción a NoSQL

NoSQL ("no SQL") comprende una amplia clase de sistemas de gestión de bases de datos que difieren del modelo clásico de SGBDR (Sistema de Gestión de Bases de Datos Relacionales) en aspectos importantes, siendo el más destacado que no usan SQL, como lenguaje principal de consultas. (Wikimedia, 2021).



www.shutterstock.com - 1921771049

<https://image.shutterstock.com/image-photo/structured-query-language-database-technology-600w-1921771049.jpg>

Los datos almacenados no requieren estructuras fijas como tablas; normalmente no soportan operaciones JOIN ni garantizan completamente ACID (atomicidad, consistencia, aislamiento y durabilidad) y, habitualmente, escalan bien horizontalmente es decir puede ir aumentando la estructura de manera sencilla. Los sistemas NoSQL se denominan, a veces, "no solo SQL" para subrayar el hecho de que también pueden soportar lenguajes de consulta de tipo SQL. (Wikimedia, 2021).

#### 1.1. Definición del NoSQL

La web 2.0 (donde quienes suministran el contenido son los usuarios) trajo grandes necesidades de datos e infraestructura tecnológica, principalmente para empresas como Google, Amazon y Facebook. Ellos tuvieron que buscar soluciones propias a estos problemas, enfocados en la gran cantidad de almacenamiento y el crecimiento de sistemas web con acceso a información en tiempo real, donde surgió la necesidad de proporcionar información gestionada y procesada desde grandes volúmenes de datos. Debido a que las bases de datos relacionales no responden de forma adecuada a esta necesidad, para la exigencia en tiempos de demandada, nacen las bases de datos NoSQL.

Comentado [1]: Bloque de texto destacado.



<https://image.shutterstock.com/image-illustration/nosql-form-binary-code-3d-600w-454628641.jpg>

Se deben definir algunos conceptos técnicos de suma importancia:

#### Escalabilidad

Es la capacidad que tiene un sistema de ser configurado en software o hardware, para mejorar la respuesta a las exigencias o nuevas demandas de capacidad de procesamiento o almacenamiento (conservando las mismas funcionalidades). Actualmente se usa mucho el término escalabilidad Vertical, para referirse a este tipo de escalabilidad.

#### Escalabilidad Horizontal

Consiste en potenciar el rendimiento del sistema desde un aspecto de mejora global, a diferencia de aumentar la potencia de una única parte del mismo. Este tipo de escalabilidad se basa en la modularidad de su funcionalidad (Marco de Desarrollo de la Junta de Andalucía, 2021). Es decir que las funcionalidades se puedan extender con la misma facilidad con la que mejoran el rendimiento y capacidad.

#### Escalabilidad vertical y horizontal

Las bases de datos no relacionales están diseñadas para ofrecer sistemas con altas capacidades de escalabilidad vertical y horizontal.

### 1.2. Cómo diferenciar NoSQL de SQL

Al referirse con el término NoSQL no se hace referencia a una base de datos y ni a un tipo de base de datos, sino más bien es la definición de un conjunto de tipos de bases de datos, que son diferentes a las bases de datos convencionales (relacionales). La principal diferencia entre las bases de datos de NoSQL y las bases de datos relacionales tradicionales, es que las NoSQL emplean una forma de almacenamiento no estructurado o de estructura diferente a tablas. Simplificando la estructura de almacenamiento, es así como las NoSQL pueden concentrarse en procesar grandes volúmenes de datos.

A continuación, le mostraremos una comparación rápida de las funciones de NoSQL con las bases de datos relacionales.

| Característica | Base de Datos NoSQL | Bases de datos Relacionales |
|----------------|---------------------|-----------------------------|
| Desempeño      | Alto                | Bajo                        |
| Fiabilidad     | Pobre               | Buena                       |
| Disponibilidad | Buena               | Buena                       |

**Comentado [h2]:**  
Tarjetas botón sin imagen.

Escalabilidad  
Escalabilidad Horizontal  
Escalabilidad vertical y horizontal



|                         |                              |                         |
|-------------------------|------------------------------|-------------------------|
| Consistencia            | Pobre                        | Buena                   |
| Almacenamiento de Datos | Optimizado para muchos datos | Mediano o Grande        |
| Estabilidad             | Alta                         | Alta (pero más costosa) |

Importante

Entérese, entonces, de las ventajas y desventajas que tiene cada uno de estos tipos de bases de datos:

#### Ventajas

Contrario a las bases de datos relacionales, las NoSQL están basadas en *key-value pairs* (clave valor).

Las NoSQL tienen diferentes tipos de almacenamiento como, por ejemplo, el almacén de columnas, documentos, claves, valor de gráficos (capas), objetos, en formato de XML y otros modos de tipos de datos. Las bases de datos NoSQL de código abierto tienen una implementación rentable económicamente. Al no requerir licencias y ni hardware de precio alto.

Trabando con bases de datos NoSQL, la expansión de funcionalidades es más fácil y más económica que usando con bases de datos relacionales.

#### Desventajas

Casi todas las bases NoSQL no admiten funciones de fiabilidad o integridad de los datos, mientras que las bases de datos relacionales, sí. Las bases NoSQL no soportan las funcionalidades o restricciones de consistencia en los datos. Para garantizar la integridad, es responsabilidad de los desarrolladores implementar con código propio, lo que agrega más complejidad al sistema.

La falta de integridad referencial en los datos hace que no sean fiables para realizar transacciones seguras, como por ejemplo en los sistemas bancarios, donde la integridad de los datos es primordial. Otras formas de complejidad encontradas en la mayoría de las bases de datos NoSQL, incluyen la incompatibilidad con consultas SQL; se necesita un lenguaje de consulta manual, haciendo los procesos mucho más lentos y complejos.

### 1.3. Tipos de bases de datos NoSQL

Dentro de las bases de datos no relacionales, se encuentran distintos modelos o tipos de implementaciones. Este grupo de bases de datos no relaciones NoSQL, desempeñan funciones para diferentes ambientes en particular y con necesidades también diferentes.



www.shutterstock.com - 1024337071

**Comentado [h3]:** Ventana modal con la siguiente información:

Como muestra la comparación a nivel de tipos de base de datos, son diversos sistemas de gestión de bases de datos los que implementan ambos modelos.

Estos sistemas proporcionan sus propias técnicas patentadas para superar los problemas y deficiencias encontradas en el sistema; además de intentar mejorar significativamente el rendimiento y la fiabilidad. Bases de datos NoSQL: (Guía definitiva con las diferencias, ventajas y desventajas de cada una, 2021).

**Comentado [h4]:** Aplicar recurso Acordeón.

Ventajas  
Desventajas



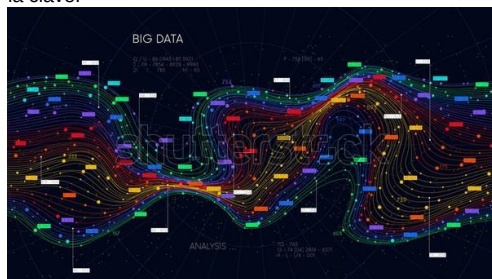
<https://image.shutterstock.com/image-illustration/server-room-3d-illustration-node-600w-1024337071.jpg>

Existen algunos tipos más comunes que otros ya que están orientadas a documentos, por permitir mejor escalabilidad horizontal (Tipos de bases de datos no relacionales, 2021).

Comentado [5]: Bloque de texto destacado.

### 1.3.1 Bases de datos Basadas en Clave Valor

Las Bases de datos basadas en clave de valor son un tipo de base de datos NoSQL, sencillo, que sirve para insertar y consultar datos. El concepto clave-valor significa que los datos se almacenan como una colección de pares clave-valor; la clave identifica de forma única la colección, y el valor contiene los datos, y se ordenan según la clave.



<https://image.shutterstock.com/image-vector/information-sorting-storage-business-technology-600w-1962818782.jpg>

#### Ejemplo

supóngase que se quieren almacenar datos de una persona como su cédula, nombres, apellidos, correo y varios números de teléfono y que, por ejemplo, necesita guardar los datos para las siguientes personas:

**Tabla 3.**

*Ejemplo 1 de base de datos*

- |                                                                  |                                                              |
|------------------------------------------------------------------|--------------------------------------------------------------|
| • Cédula: <b>1029283762</b>                                      | • Cédula: <b>98765322</b>                                    |
| • Nombre: <b>JUAN CARLOS</b>                                     | • Nombre: <b>DANIELA</b>                                     |
| • Apellidos: <b>MARTINEZ</b>                                     | • Apellidos: <b>VELEZ LEMOS</b>                              |
| • Correo: <a href="mailto:juancho@nerss.et">juancho@nerss.et</a> | • Correo: <a href="mailto:dnala@nerss.et">dnala@nerss.et</a> |
| • Teléfono: <b>3150817273</b>                                    |                                                              |
| • Teléfono: <b>3168272372</b>                                    |                                                              |

Considerando que se toma la cédula como una especie de llave para identificar a una persona, es decir la cédula es el identificador único para cada persona y no hay dos personas con la misma cédula.

#### Más posibilidades

En la misma base de datos se necesita guardar datos de carros, siendo la placa lo que permite identificar al vehículo; entonces los datos a guardar serían:



Placa: MSR-13  
Modelo: 2015  
Marca: MAZDA  
Tipo: SEDAN

La base de datos sería:

Tabla 4.

Ejemplo 2 de base de datos

| CLAVE                        | VALOR                                                |
|------------------------------|------------------------------------------------------|
| persona/cedula               | 1029283762                                           |
| persona/1029283762/nombres   | JUAN CARLOS                                          |
| persona/1029283762/apellidos | MARTINEZ                                             |
| persona/1029283762/correo    | <a href="mailto:juancho@nerss.e">juancho@nerss.e</a> |
| persona/1029283762/telefono  | 3150817273                                           |
| carro/placa                  | MSR-13                                               |
| carro/MSR-13/modelo          | 2015                                                 |
| carro/MSR-13/marca           | MAZDA                                                |
| carro/MSR-13/tipo            | SENAN                                                |
| persona/cedula               | 98765322                                             |
| persona/98765322/nombres     | DANIELA                                              |
| persona/98765322/apellidos   | VELEZ LEMOS                                          |
| persona/98765322/correo      | <a href="mailto:dnala@nerss.et">dnala@nerss.et</a>   |

Estas bases de datos son óptimas para consultas en ambientes de respuesta rápida, para aplicaciones de tiempo real, como sistemas de telefonía IP, sistemas de Chats

### 1.3.2 Bases de datos orientadas a documentos.

Los SGDB con estas características son denominadas como Bases de datos documentales o base de datos documental, funcionan en el marco de la definición de un "Documento"; la mayoría de estos SGDB coinciden en cómo se almacena la información, empleando un formato estándar. Los formatos estándar más comunes o populares son: JSON, BSON y XML. Se podría considerar este tipo de formatos como el más utilizado en la actualidad.

Comentado [6]: Aplicar recurso: Sliders tipo 4.

Ejemplo (incluye mini tabla)  
Más posibilidades (incluye imagen)  
La base de datos sería (incluye tabla)



<https://image.shutterstock.com/image-photo/document-management-concept-online-documentation-600w-1925677844.jpg>

### Formato XML

XML, sigla en inglés de eXtensible Markup Language, traducido como "Lenguaje de Marcado Extensible" o "Lenguaje de Marcas Extensible", es un metalenguaje que permite definir lenguajes de marcas, desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. Proviene del lenguaje SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML) para estructurar documentos grandes. A diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciones deben comunicarse entre sí o integrar información. (Wikimedia, 2021).

**Comentado [7]:** Aplicar recurso: Acordeón.

Formato XML  
Modelar información  
En base de datos relacional  
Formato JSON  
Bases de datos no relacionales documentales  
Ejemplo  
Representación en formato BSON

### Modelar Información

Suponga que se desea modelar la información contenida en un mensaje de correo electrónico.

#### Figura 1.

Modelación de información

```
<Edit_Mensaje>
  <Mensaje>
    <Remitente>
      <Nombre>Nombre del remitente</Nombre>
      <Mail>Correo del remitente</Mail>
    </Remitente>
    <Destinatario>
      <Nombre>Nombre del destinatario</Nombre>
      <Mail>Correo del destinatario</Mail>
    </Destinatario>
    <Texto>
      <Asunto>
        Este es mi documento con una estructura muy sencilla
        no contiene atributos ni entidades...
      </Asunto>
      <Parrafo>
        Este es mi documento con una estructura muy sencilla
        no contiene atributos ni entidades...
      </Parrafo>
    </Texto>
  </Mensaje>
</Edit_Mensaje>
```

Fuente: (Wikimedia, 2021).

### En Base de datos relacional





Si se quisiera modelar esto en una base de datos relacional, seguramente se definiría una tabla para personas (con atributos, nombre y el correo) y una tabla para los correos con dos relaciones a la tabla personal: una para el remitente y otra para el destino:

- Persona (nombres, correo)
- Correo (asunto y texto)

Sin embargo, en XML en lugar de tener filas en las tablas cada correo es un archivo con la estructura dada en el ejemplo.

#### Formato JSON.

JSON (acrónimo de JavaScript Object Notation, «notación de objeto de JavaScript») es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera (año 2019) un formato independiente del lenguaje (Wikimedia, 2021).

- **Cadenas de texto:** Al ser una secuencia de ninguno o más caracteres se ponen entre doble comilla y se cómo en leguajes JavaScript. Ejemplo: "Hola Mundo"
- **Números:** Se permiten números negativos y positivos que también pueden contener parte no entera separada por puntos. Ejemplo: 654.321
- **Arreglos:** Representa una lista ninguno o más valores los cuales pueden ser todos del mismo o diferente tipo. Los valores separados por comas y el vector entre llaves ([]). Ejemplo ["luis","Carlos","Julian",99]
- **Booleanos:** Representan valores booleanos y pueden tener dos valores: true y false
- **Valores Nulos:** Representan el valor nulo (null).
- **Objetos:** Son colecciones no ordenadas de pares de la forma <nombre>:<valor> donde el nombre debe ser separado por comas y puestas entre corchetes ({}).

#### Bases de datos no relacionales documentales

Las Bases de datos no relacionales documentales, su concepto de guardar la información con el modelo clave/valor, es por eso que los documentos más comunes son en formato JSON. La diferencia radica en que un documento se almacena en una clave; esta definición puede parecer abstracta. Esto se explica mejor con un ejemplo:

**Figura 2.** Representación de los datos de una persona en un documento

```
{
  "nombres": "JUAN CARLOS",
  "apellidos": "MARTINEZ",
  "fecha_nacimiento": "1995-05-15",
  "edad": 65,
  "saldo": 9876.134,
  "hobbies": ["música", "futbol", "leer"],
  "correo": "juancho@nerss.et",
  "pedidos": [],
  "acreedores": null
}
```



Note cómo “nombres” es la clave y “JUAN CARLOS” es el valor. De la misma forma como “edad” es la clave y 65 el valor.

Ejemplo

Suponga que necesita modelar un menú de una aplicación como se muestra a continuación:

Figura 3: ejemplo



Representación en formato BSON

La representación de JSON que se muestra en esta tabla, es una representación en formato texto, pero también se puede representar en formato binario (uno y ceros); a esta otra representación equivalente en binario se llama BSON. Es una cadena de unos y ceros sin nada que la caracterice de más, excepto que al convertirla en cadenas de texto se presenta como se observó en el ejemplo.

Tabla 5. ejemplo

| Su representación en XML                                                                                                                                                                                                                                                                                                                      | Su equivalente en JSON                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre>&lt;menu id="file" value="Archivo"&gt;   &lt;popup&gt;     &lt;menuitem value="Nuevo" onclick="CreateNewDoc()" /&gt;     &lt;menuitem value="Abrir" onclick="OpenDoc()" /&gt;     &lt;menuitem value="Cerrar" onclick="CloseDoc()" /&gt;     &lt;menuitem value="Guardar" onclick="SaveDoc()" /&gt;   &lt;/popup&gt; &lt;/menu&gt;</pre> | <pre>{   "menu": {     "id": "file",     "value": "Archivo",     "popup": {       "menuitems": [         {           "value": "Nuevo",           "onclick": "CreateNewDoc()"         },         {           "value": "Abrir",           "onclick": "OpenDoc()"         },         {           "value": "Cerrar",           "onclick": "CloseDoc()"         },         {           "value": "Guardar",           "onclick": "SaveDoc()"         }       ]     }   } }</pre> |

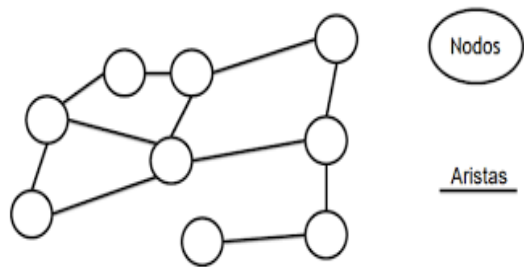


```
}
}
}
```

1.3.3 Bases de Datos Orientada a Grafos.

Un grafo es una colección de elementos ("nodos"), que aportan información, relacionados mediante "aristas", que son la conexión entre los nodos, y que pueden tener una dirección concreta (Tipos de bases de datos no relacionales, 2021).

Figura 4. Grafos



Fuente: (Tipos de bases de datos no relacionales 2021) <https://www.certia.net/wp-content/uploads/2021/03/Grafos.png>

Este es un tipo de base muy diferente a los anteriores; maneja la información de una manera particular empleando grafos y teoría de grafos. Cada nodo, sólo debe contener una sola columna, entonces se debe normalizar completamente las bases de datos. Y sólo aplica grafos de tipo binario, cada nodo sólo se puede relacionar con otros dos nodos como máximo). Las ventajas de este tipo de bases de datos van enfocadas a la integridad de los datos y la optimización de consultas.

Comentado [8]: Aplicar recurso: Bloque de texto destacado.

1.4. Sistemas de Gestión de bases de datos (SGDB) NoSQL

Como se sabe, la base de datos sólo sirve si existe un sistema que la gestione, un SGDB. En NoSQL hay varios SGDB, y la mayoría son de código libre o libre distribución. Le mostraremos algunas de ellas, las más populares, con sus respectivas características. Tenga en cuenta que, no obstante, pueden existir otros SGDB menos populares pero que pueden ser una buena opción en el momento de trabajar para BigData.

Tabla 6. Sistemas de gestión de bases de datos

| Tipo       |  | Nombre  | Lanzamiento | Licencia      |
|------------|--|---------|-------------|---------------|
| Documental |  | MongoDB | 2008        | GNU AGPL v3.0 |



|             |                                                                                                       |                     |      |                              |
|-------------|-------------------------------------------------------------------------------------------------------|---------------------|------|------------------------------|
|             | <br>CouchDB<br>relax | CouchDB             | 2005 | Apache License 2.0           |
|             | <br>RAVENDB          | Raven DB            | 2010 | GNU AGPL v3.0                |
| Clave/Valor | <br>cassandra        | Apache<br>Cassandra | 2008 | Apache License 2.0           |
|             | <br>riak             | Riak                | 2009 | Apache License 2.0           |
|             | <br>redis           | Redis               | 2009 | Licencia: BSD                |
| Grafos      | <br>neo4j          | Neo4j               | 2009 | GNU AGPL v3.0                |
|             | <br>dex            | Dex                 | 2008 | Comercial                    |
|             | <br>sones          | Sones GraphDB       | 2012 | GNU AGPL v3.0 y<br>comercial |

## 2. NoSQL con MongoDB.

Como se mencionó antes, MongoDB es un sistema de bases de datos no relacionales, multiplataforma, es decir, se puede instalar en cualquier sistema operativo para servidor (Windows, Linux, BSD, MAC), el tipo de base de datos empleada es el tipo de bases de datos documental, pero también se puede administrar bases de datos clave/valor.

El tipo de licencia de software libre, es específicamente GNU AGPL v3.0. MongoDB usa el formato BSON (JSON Compilado) para guardar la información.

**Comentado [9]:** Aplicar recurso: Bloque de texto destacado, incluir la imagen.

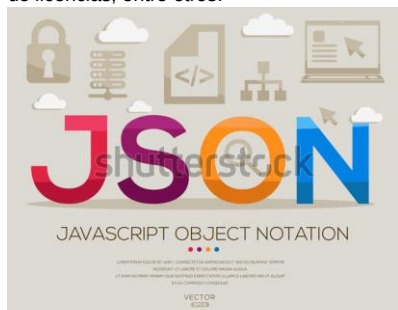


www.shutterstock.com - 467410661

<https://image.shutterstock.com/image-vector/nosql-non-relational-database-concept-600w-467410661.jpg>

## 2.1. Conceptos Básicos

Los siguientes conceptos son clave, para la asimilación de todos los elementos de este componente formativo. Para la implementación de bases de datos, conviene tener claridad acerca de lo que es un documento, una colección, a qué se hace referencia cuando se habla de creación y actualización de bases de datos, instalación de licencias, entre otros.



www.shutterstock.com - 1871534536

<https://image.shutterstock.com/image-vector/json-mean-javascript-object-notation-600w-1871534536.jpg>

### JSON – JavaScript Object Notation

Como la información gestionada usa el formato JSON para representar objetos según la especificación documento RFC 46274 y como JSON es independiente del lenguaje, se ha popularizado mucho el uso del lenguaje JavaScript. JSON es usado por muchos sistemas para intercambiar datos, por lo sencillo que es, si se compara con XML.

### Documento

Un documento, es un conjunto de datos estructurados, pero esta estructura no es estática, es decir puede variar de un documento a otro, la estructura básica que contiene pares clave/valor, y se usa BSON (JSON Binario) como formato para almacenar los documentos dentro del SGDB. Un documento es como el equivalente a una fila dentro de una tabla de un modelo relacional.

### Colección

En este contexto, cuando se habla de colección, se está haciendo referencia a un conjunto de documentos, es lo que se denomina una colección. Se puede asemejar a una tabla en las bases de datos relacionales.

**Comentado [10]:** Aplicar recurso: Tarjetas botón sin imagen.

JSON – JavaScript Object Notation  
Documento  
Colección





## 2.2. Instalación de MongoDB

La instalación de MongoDB en un sistema Windows 10, comenzando por su descarga, implica el seguimiento de una serie de pasos ordenados. En este punto del contenido formativo, le daremos a conocer el recorrido detallado para tal instalación.

Siga cuidadosamente las instrucciones que se encuentran en el pdf y luego aplíquelas en su computador visitando <https://www.mongodb.com/>, estando allí, haga clic en Software y Community Server para aplicar los pasos.

Descargar

**Comentado [h11]:** Crear pdf con la información contenida en el archivo:

Anexo\_1\_InstalacionMongoDB

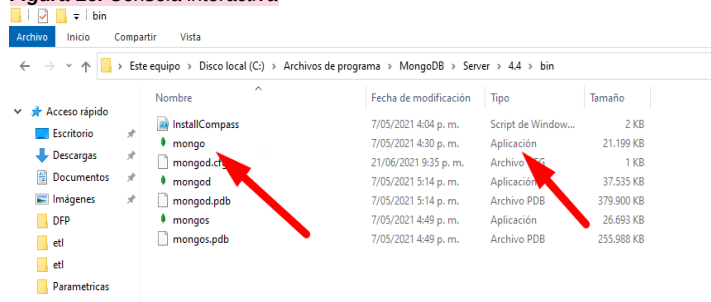
## 2.3. Consola interactiva

MongoDB tiene al igual que todas bases de datos una interface de línea de comando (CLI) desde la cual se puede acceder aun API de las funcionalidades. Podemos acceder a su consola interactiva y realizar nuestras primeras interacciones con MongoDB.

### Ingreso

Para ingresar a su consola interactiva, hay que dirigirse a la carpeta donde quedó instalado MongoDB. (C:\Program Files\MongoDB\Server\4.4\bin) y se debe dar clic en el ejecutable que se llama mongo.exe, como muestra aquí:

**Figura 16. Consola interactiva**



### Instalación exitosa

Si la vista es como se muestra a continuación es porque la instalación fue exitosa.

**Figura 17. Instalación exitosa**



```
Seleccionar C:\Program Files\MongoDB\Server\4.4\bin\mongo.exe
MongoDB shell version v4.4.6
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("e085b48a-4059-47e6-ae1f-89cb7bcb19c2") }
MongoDB server version: 4.4.6
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
  https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
  https://community.mongodb.com
---
The server generated these startup warnings when booting:
  2021-06-21T21:35:16.749-05:00: Access control is not enabled for the database. Read and write access to data and
  configuration is unrestricted
---
  Enable MongoDB's free cloud-based monitoring service, which will then receive and display
  metrics about your deployment (disk utilization, CPU, operation statistics, etc).

  The monitoring data will be available on a MongoDB website with a unique URL accessible to you
  and anyone you share the URL with. MongoDB may use this information to make product
  improvements and to suggest MongoDB products and deployment options to you.

  To enable free monitoring, run the following command: db.enableFreeMonitoring()
  To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
>
```

## Prueba

Para probar que está funcionando bien escribamos el comando help.

Figura 18. Comprobación de funcionamiento

```
> help
  db.help()                help on db methods
  db.mycoll.help()          help on collection methods
  sh.help()                 sharding helpers
  rs.help()                 replica set helpers
  help admin               administrative help
  help connect             connecting to a db help
  help keys                 key shortcuts
  help misc                 misc things to know
  help mr                  mapreduce

  show dbs                 show database names
  show collections          show collections in current database
  show users                show users in current database
  show profile              show most recent system.profile entries with time >=

lms
  show logs                 show the accessible logger names
  show log [name]           prints out the last segment of log in memory, 'global'

is default
  use <db_name>            set current database
  db.mycoll.find()          list objects in collection mycoll
  db.mycoll.find( { a : 1 } ) list objects in mycoll where a == 1
  it                        result of the last line evaluated; use to further

iterate
  DBQuery.shellBatchSize = x set default number of items to display on shell
  exit                      quit the mongo shell
```



```
>
```

### Seleccionar base de datos

Al igual que el base de datos relacionales se debe usar un comando para seleccionar una base de datos, pero antes de eso miremos que bases de datos existen con el siguiente comando:

**Figura 19.** Comando Show dbs

```
> show dbs;
admin    0.000GB
config  0.000GB
local    0.000GB
>
```

### Comando de selección

El comando para seleccionar una base de datos es `use <nombre base de datos>`.

**Figura 20.** Nombre base de datos

```
> use local;
switched to db local
>
```

### Crear base de datos

El comando `use` se usa también para crear una nueva base de datos prueba de lcomando seria: `use pruebas;`

**Figura 21.** Use pruebas

```
> use pruebas;
switched to db pruebas
>
```

El comando `show dbs` no mostrará esta base de datos hasta que exista el primer documento.

**Comentado [12]:** Aplicar recurso: Sliders tipo 1.

Ingreso  
Instalación exitosa  
Prueba  
Seleccionar base de datos  
Comando de selección  
Crear base de datos

## 2.4. Crear Primer Documento

Se creará una base de datos y una colección, y eso se hará almacenando un documento usando el objeto `db`, pero primero se debe ejecutar del comando `use`. Un documento puede tener, en teoría, un máximo de hasta 16MB de información.



<https://image.shutterstock.com/image-photo/keyboard-key-access-data-collection-600w-279088271.jpg>





Aprovechando el primer objeto visto, JSON, para almacenarlo en MongoDB y consultarlo, se debe tener en cuenta dentro de la consola interactiva, usar variables para crear o modificar documentos; de esta manera podemos evitar accidentes con una mala manipulación directa de la base de datos.

**Figura 22.** Creación del primer documento

Se define una variable documento

```
> documento = { "nombres" : "JUAN CARLOS",
... "apellidos" : "MARTINEZ",
... "fecha_nacimiento" : "1995-05-15",
... "edad" : 65,
... "saldo" : 987.432,
... "hobbies" : ["musica", "futbol", "leer"],
... "correo" : "juancho@nerss.et",
... "pedidos" : [],
... "acreedores" : null
... }
{
  "nombres" : "JUAN CARLOS",
  "apellidos" : "MARTINEZ",
  "fecha_nacimiento" : "1995-05-15",
  "edad" : 65,
  "saldo" : 987.432,
  "hobbies" : [
    "musica",
    "futbol",
    "leer"
  ],
  "correo" : "juancho@nerss.et",
  "pedidos" : [ ],
  "acreedores" : null
}
```

Comentado [13]: Aplicar recurso: Bloque de texto destacado (imagen).

Habría que adicionar que, de esta manera, para almacenar un documento debemos ejecutar el método. insert () del objeto db, especificando el nombre de la colección (la colección se crea de manera dinámica como la base de datos).

### Ejemplo

**Figura 23.** Almacenar un documento

```
> db.coleccion_personas.insert(documento)
WriteResult({ "nInserted" : 1 })
>
```

### Primera colección

Es así como se ha creado la primera colección de nombre `coleccion_personas` y nuestro primer documento, para confirmar esto, podemos ejecutar tanto el comando `show collections`



**Figura 24.** Creación de una colección

```
> show collections
coleccion_personas
>
```

### Método Find()

Para consultar los documentos se usa el método find() de la colocación como se muestra a continuación:

**Figura 25.** Consulta de documentos

```
> db.coleccion_personas.find()
{ "_id" : ObjectId("60d154cale0ed70729ced13d"), "nombres" : "JUAN CARLOS", "apellidos" :
"MARTINEZ", "fecha_nacimiento" : "1995-05-15", "edad" : 65, "saldo" : 987.432, "hobbies" : [
"musica", "futbol", "leer" ], "correo" : "juancho@nerss.et", "pedidos" : [ ], "acreedores" :
null }
>
```

### Llave primaria

Note que existe un atributo que nunca se definió, llamado `_id`, este es un índice que es lo equivalente a llave primaria.



<https://image.shutterstock.com/image-photo/hand-holding-glowing-light-bulb-600w-304959785.jpg>

## 3. Manipulación de datos con MongoDB

Las bases de datos deben poder manipular la información de varias maneras, ya que los procesos en los sistemas informáticos necesitan de esas operaciones básicas, tales como crear datos, consultarlos, actualizarlos y, en algún punto, eliminarlos. (Create, Read, Update and Delete - CRUD).

Por tanto, queda establecido que almacenarla o contenerla no es su mera y única función básica.



### Comentado [h14]:

Aplicar recurso: Bloque de texto destacado, incluir imagen.

<https://image.shutterstock.com/image-photo/update-software-computer-program-upgrade-600w-1129542374.jpg>

### 3.1. Crear Documentos



Es la primera de cuatro funciones elementales a la hora de utilizar bases de datos. Crear documentos nos permite insertar unidades de información; en el caso de MongoDB, a través de documentos en colecciones de datos.

#### Para crear un documento

- Para poder crear un documento, MongoDB posee el método `insert()`, `.save()` y de una manera especial `.update()` (Yohan D, 2014).

```
> documento = {  
... "titulo" : "Titulo Libro",  
... "Autores" : ["Peter", "Johnathan"]  
... }  
> db.libros.insert(documento)  
WriteResult({ "nInserted" : 1 })  
>
```

#### Insertar varios simultáneamente

- Se pueden insertar varios a la vez:

```
> documentos = [ {  
... "titulo" : "Otro Libro",  
... "Autores" : ["Peter", "Yolima"]  
... } ,  
... {  
... "titulo" : "Nuevo Libro",  
... "Autores" : ["Zulema"]  
... } ]  
> db.libros.insert(documentos)  
BulkWriteResult({  
  "writeErrors" : [ ],  
  "writeConcernErrors" : [ ],  
  "nInserted" : 2,  
  "nUpserted" : 0,  
  "nMatched" : 0,  
  "nModified" : 0,  
  "nRemoved" : 0,  
  "upserted" : [ ]  
})  
>
```



### Actualizar documento

- Tiene todas las funciones de `.insert()`, pero además permite actualizar un documento si ya existe el `_id` de dicho documento, en ese caso `.insert()` mostraría una excepción.



www.shutterstock.com - 1129540169

<https://image.shutterstock.com/image-photo/update-software-computer-program-upgrade-600w-1129540169.jpg>

**Comentado [15]:** Aplicar recurso: Slider tipo 1.

Para crear un documento  
Insertar varios simultáneamente  
Actualizar documentos

### 3.2. Leer y Consultar colecciones

Cuando de la función leer y consultar la información se trata, la forma básica consiste en usar el método `find()` de la colección, tal como ya se ha dicho a en punto anterior de este componente formativo.

#### • Leer y consultar colecciones

```
> db.libros.find()
{ "_id" : ObjectId("60d159461e0ed70729ced13e"), "titulo" : "Titulo Libro", "Autores" : [ "Peter", "Johnathan" ] }
{ "_id" : ObjectId("60d159df1e0ed70729ced13f"), "titulo" : "Otro Libro", "Autores" : [ "Peter", "Yolima" ] }
{ "_id" : ObjectId("60d159df1e0ed70729ced140"), "titulo" : "Nuevo Libro", "Autores" : [ "Zulema" ] }
>
```

#### • Función `pretty()`

Otra forma de mejor visualización es aplicar la función `pretty()`

```
> db.libros.find().pretty()
{
  "_id" : ObjectId("60d159461e0ed70729ced13e"),
  "titulo" : "Titulo Libro",
  "Autores" : [
    "Peter",
    "Johnathan"
  ]
}
{
  "_id" : ObjectId("60d159df1e0ed70729ced13f"),
  "titulo" : "Otro Libro",
  "Autores" : [
    "Peter",
    "Yolima"
  ]
}
{
  "_id" : ObjectId("60d159df1e0ed70729ced140"),
  "titulo" : "Nuevo Libro",
  "Autores" : [
    "Zulema"
  ]
}
>
```

**Comentado [16]:** Aplicar recurso: Pestañas o tabs tipo 1.

Leer y consultar colecciones  
Función `Pretty()`



### 3.3. Actualizar Documentos

Es momento de ver un ejemplo de cómo se actualiza la información en la base de datos; ello, conociendo el contenido del atributo `_id` que del documento a actualizar. Preste atención a la siguiente estructura:

```
db.collection.update (
<consulta - criterios>,
<documento_modificado >,
{upsert: true | false, multi : true | false }
)
```

La opción `upsert`, permite agregar un documento si no existe, siempre y cuando esta opción esté activada.

```
> db.libros.update({"_id": ObjectId("60d159df1e0ed70729ced13f") }, {"titulo": "Titulo
Modificado"})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> |
```

Comentado [17]: Asignar recurso: Bloque de texto destacado con imagen.

### 3.4. Borrar Documentos

El método `remove()`, elimina uno o más documentos de una colección. Recibe parámetros para realizar una eliminación selectiva; si no se le pasa ningún parámetro, elimina todos los documentos de la colección (Yohan D, 2014). Esto, nos conduce a concluir que las bases de datos No Relacionales son más sencillas de administrar al no tener que hacer uso de estructuras complejas e interrelacionadas.

```
> db.libros.remove({"_id": ObjectId("60d159df1e0ed70729ced13f") })
WriteResult({ "nRemoved" : 1 })
>
```

La función. `drop()` elimina toda una colección, y es la más recomendable a la hora de realizar esta tarea, ya que utiliza menos recursos que `.remove()`. (Yohan D, 2014).

```
> db.libros.drop() ;
true
> show collections
coleccion_personas
> |
```

Comentado [18]: Asignar recurso: Bloque de texto destacado con imagen.

### C. ACTIVIDADES DIDÁCTICAS (OPCIONALES SI SON SUGERIDAS)

| DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA |  |
|------------------------------------|--|
| Nombre de la Actividad             |  |



|                                                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Objetivo de la actividad</b>                                                     |                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| <b>Tipo de actividad sugerida</b>                                                   | <div>  Verdadero/Falso         </div> <div>  Opción múltiple         </div> <div>  Respuesta múltiple         </div> <div>  Completar los esp...         </div> <div>  Banco de palabras         </div> <div>  Arrastrar y soltar C...         </div> <div>  Elemento desple...         </div> <div>  Arrastrar y soltar e...         </div> <div>  Secuencia de elem...         </div> <div>  Numérico         </div> <div>  Punto clave         </div> |
| <b>Archivo de la actividad<br/>(Anexo donde se describe la actividad propuesta)</b> |                                                                                                                                                                                                                                                                                                                                                                                                                                                          |

#### D. MATERIAL COMPLEMENTARIO:

| Referencia APA del Material | Tipo de material<br>(Video, capítulo de libro, artículo, otro) | Enlace del Recurso o<br>Archivo del documento o material |
|-----------------------------|----------------------------------------------------------------|----------------------------------------------------------|
|                             |                                                                |                                                          |
|                             |                                                                |                                                          |
|                             |                                                                |                                                          |

#### E. GLOSARIO:

| TERMINO              | SIGNIFICADO                                                                                                                     |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------|
| BASE DATOS           | Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.                                        |
| COLECCIÓN            | Es un compendio de documentos que comparten una relación de estructura similar, no necesariamente igual para todos los objetos. |
| DICCIONARIO DE DATOS | Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.                     |



|               |                                                                                                                                                                                                                                       |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| DOCUMENTO     | En bases de datos no relaciones un documento es un objeto en formato JSON que es almacenado en binario en el motor NoSQL (BSON).                                                                                                      |
| ESCALABILIDAD | Es la capacidad que tiene un sistema de ser configurado en software o hardware, para mejorar la respuesta a las exigencias o nuevas demandas de capacidad de procesamiento o almacenamiento (conservando las mismas funcionalidades). |
| METADATOS     | Conjunto unitario de instrucciones que permite a una computadora realizar funciones diversas, como el tratamiento de textos, el diseño de gráficos, la resolución de problemas matemáticos, el manejo de bancos de datos              |
| NoSQL         | Base de daos que no usan el estándar SQL                                                                                                                                                                                              |

#### F. REFERENCIAS BIBLIOGRÁFICAS:

Altarade, M. Bases de datos NoSQL: Guía definitiva. (2021, Junio 10). [Web log post]. *Blogspot*.  
<https://pandorafms.com/blog/es/bases-de-datos-nosql>

Asociación de Academias de la Lengua Española (2021). *Diccionario de la lengua española*. RLE.RAE  
<https://dle.rae.es/base#CiiosqQ>

Gratero, Y. (2014) *Mongo DB en español*. GITHUB. <https://github.com/yograterol/ebook-mongoddb-basico>

Macarrón, P. (2021, Marzo 08). *Tipos de bases de datos no relacionales*. [Web log post] *Blogspot*.  
<https://www.certia.net/tipos-de-bases-de-datos-no-relacionales>

Marco de Desarrollo de la Junta de Andalucía (2021). *Conceptos sobre la escalabilidad*. JUNTADEANDALUCIA.  
<http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/220>

Office Solución de problemas de productos (2021). *Descripción de los conceptos básicos de normalización de la base de datos*. <https://docs.microsoft.com/es-es/office/troubleshoot/access/database-normalization-description>

Resnick, M. (16 de marzo de 2009). *Sembrando las semillas para una sociedad más creativa*. EDUTECA.  
<http://www.eduteka.org/ScratchResnickCreatividad.php>

Sudarshan, H. (2002). *Fundamentos de bases de datos*. McGraw-Hill.

Wikipedia la enciclopedia libre (10 de junio de 2021). *Extensible Markup Language*.  
[https://es.wikipedia.org/w/index.php?title=Extensible\\_Markup\\_Language&oldid=136233446](https://es.wikipedia.org/w/index.php?title=Extensible_Markup_Language&oldid=136233446)

Wikipedia la enciclopedia libre (16 de enero de 2021). *NoSQL*. WIKIPEDIA.  
<https://es.wikipedia.org/w/index.php?title=NoSQL&oldid=132486305>



#### G. CONTROL DEL DOCUMENTO

| Autor (es) | Nombre                        | Cargo                   | Dependencia<br>(Para el SENA indicar Regional y Centro de Formación) | Fecha            |
|------------|-------------------------------|-------------------------|----------------------------------------------------------------------|------------------|
|            | Henry Eduardo Bastidas Paruma | Instructor              | Regional Cauca, Centro de teleinformática y producción industrial    | Marzo 25 de 2021 |
|            | Peter Emerson Pinchao Solis   | Instructor              | Regional Cauca, Centro de teleinformática y producción industrial    | Marzo 25 de 2021 |
|            | Fabián Leonardo Correa Díaz   | Diseñador Instruccional | Centro agropecuario La Granja, Regional Tolima                       | Junio de 2021    |

#### H. CONTROL DE CAMBIOS

(Diligenciar únicamente si realiza ajustes a la Unidad Temática)

| Autor (es) | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
|------------|--------|-------|-------------|-------|------------------|
|------------|--------|-------|-------------|-------|------------------|





# P.O.O modelado y codificación con java



@SENAComunica

[www.sena.edu.co](http://www.sena.edu.co)



# Diagrama de clases e implementación en java

Instructor: GUSTAVO  
SALAZAR ESCOBAR

# TABLA DE **CONTENIDO**



**01**

## Ejercicio Modelado UBER:

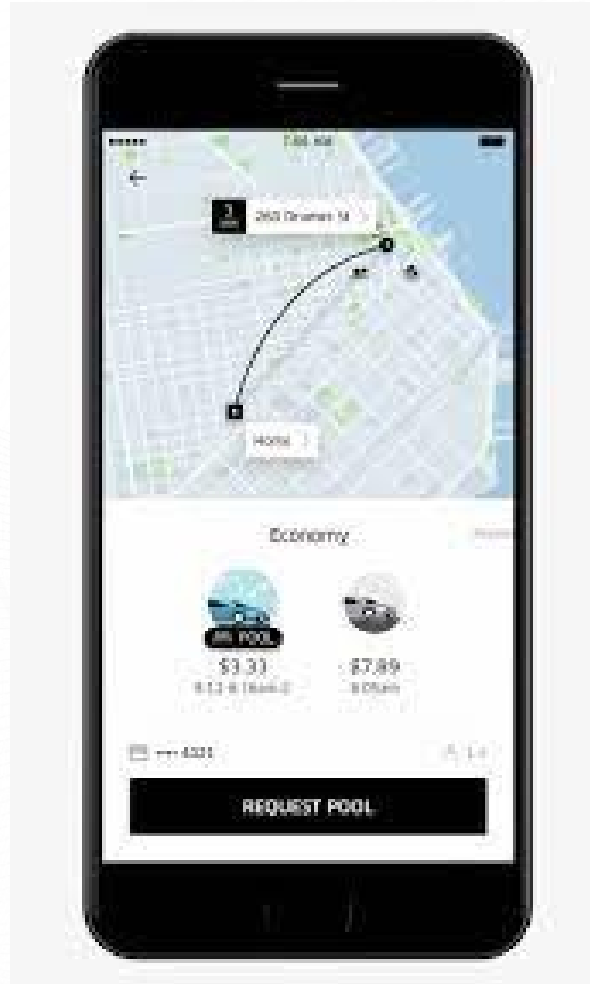
- Analizando Uber en Objetos
- Clases en UML y su sintaxis en código
- Modelando nuestros objetos Uber
- Aplicando Herencia a nuestro proyecto Uber

**02**

## Codificación ejercicio UBER en JAVA:

- Definiendo clases en Java
- Objetos, método constructor y su sintaxis en código
- Objetos. Dando vida a nuestras clases en Java
- Declarando un Método Constructor en Java
- Aplicando herencia en lenguaje Java

# 01 Ejercicio Modelado Objetos de Uber



# Analizando Uber en Objetos



Para resolver un problema es importante dividirlos en pequeños modulos y de aqui parte el problema a analizar de Uber

1. Todo parte de una necesidad, es necesario contar con un celular y querer trasladarse de un punto A a un B
  2. Se solicita un automovil y en la app se selecciona de que punto a que punto se va a trasladar
  3. La app muestra un catalogo de servicios el cual cada uno de ellos tiene un costo por ser servicios diferentes
- Uber x
  - Uber Pool
  - Uber Black
  - Uber Van

# Analizando Uber en Objetos



4. Cuando se elige uno de los servicios, el usuario espera que lo recoja el automovil que decidio escoger para prestar el servicio y trasladar del punto A escogido por el usuario al punto B



# Analizando Uber en Objetos



## Objetos

- 1.User -> Quien solicita al automovil
- 2.Route -> Necesidad de trasladar de un punto A a un punto B
- 3.4 objetos que aparecen. Uber x, Uber Pool, Uber Black, Uber Van
- 4.Driver -> El cual representa al conductor
- 5.Card, Paypal, cash -> Las cuales representan la forma en que se va a pagar el servicio al estar completo y finalizado
- 6.Trip -> Es de tipo conceptual y va a captar quien esta ejecutando el viaje, el lugar de recogida y entrega, ruta del mapa, etc

# Reto 1: identificando objetos



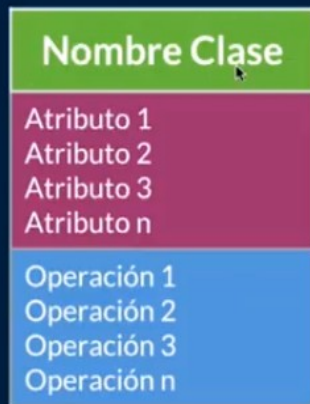
- 1.Existe la necesidad de adoptar un perro, para ello una persona se debe trasladar a las instalaciones físicas por ejemplo de una fundación de perros que permite la adopción de perros
- 2.Debe existir una persona disponible la cual atienda y muestre de manera física los perros que están disponibles para adopción
- 3.Los perros están disponibles para que la persona los vea y escoja cada uno de ellos



# Clases en UML y su sintaxis en código



## UML



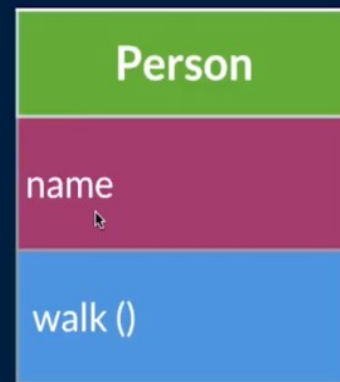
Identidad

Estado

Comportamiento



q|>



# Clases en UML y su sintaxis en código



```
class Person {  
  constructor() {  
    this.name = "";  
  }  
  walk() {...}  
}
```

Java

```
class Person {  
  String name = "";  
  void walk() { }  
}
```

Python

```
class Person:  
  name = "";  
  def walk():
```



JavaScript

```
Person.prototype.walk = function () {  
  ...  
}
```

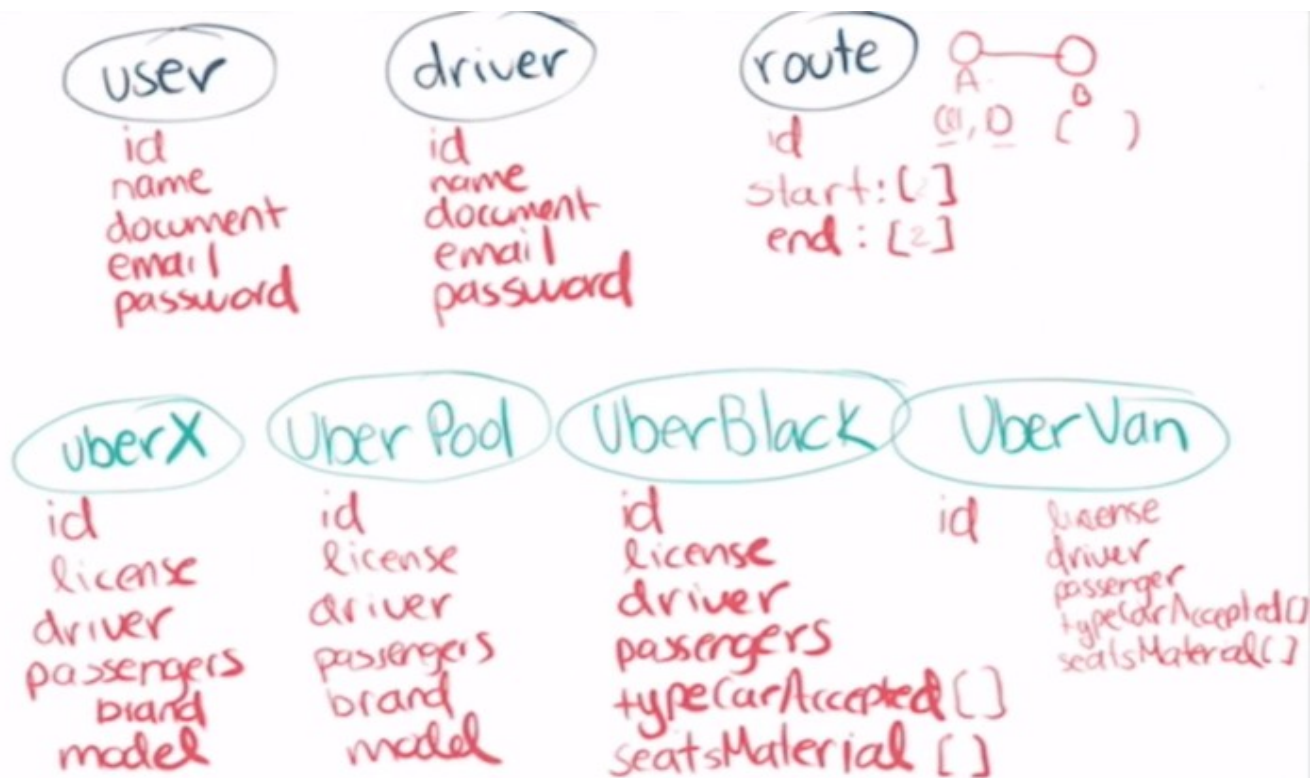
PHP

```
class Person {  
  $name = "";  
  function walk() { }  
}
```

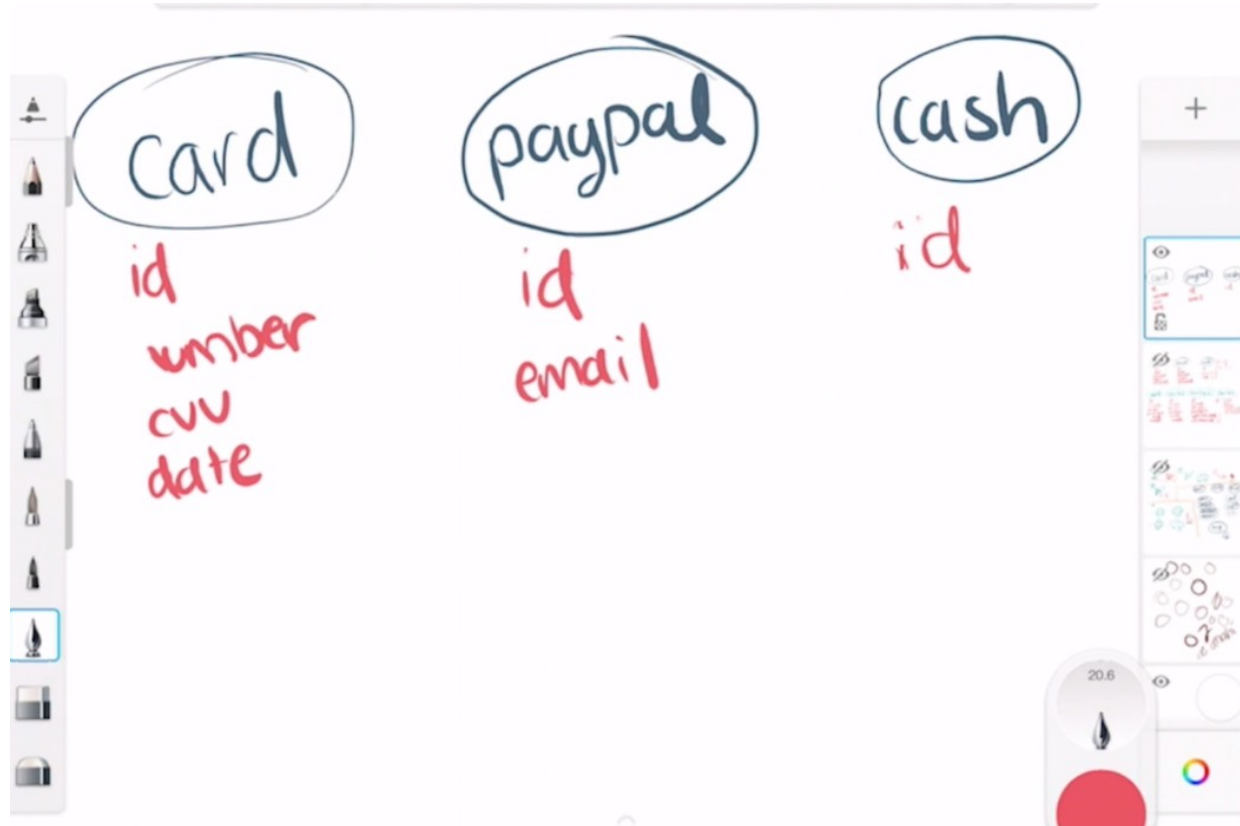
# Modelando nuestros objetos Uber



Los objetos analizados en la clase 12 donde se identificaron los objetos se establecio que eran los siguientes, cada uno de estos va estar compuesto por lo que esta a continuacion de la flecha -> y el ejemplo representa a la Ciudad de Mexico porque los vehiculos Uber black y Uber Van tienen requisitos mayores a los de la Ciudad de Bogotá



# Modelando nuestros objetos Uber



# Aplicando Herencia a nuestro proyecto Uber



| User                                        |
|---------------------------------------------|
| id<br>name<br>document<br>email<br>password |
|                                             |

| Driver                                      |
|---------------------------------------------|
| id<br>name<br>document<br>email<br>password |
|                                             |

| Route                   |
|-------------------------|
| id<br>star:[]<br>end:[] |
|                         |

| Trip                                                                                                                         |
|------------------------------------------------------------------------------------------------------------------------------|
| id<br>user: User<br>car: Car<br>route: Route<br>amount: Double<br>payment: Payment<br>progress: Double<br>completed: Boolean |
|                                                                                                                              |

| UberX                                                                                            |
|--------------------------------------------------------------------------------------------------|
| id<br>license: String<br>driver: Driver<br>passengers: integer<br>brand: String<br>model: String |
|                                                                                                  |

| UberPool                                                                                         |
|--------------------------------------------------------------------------------------------------|
| id<br>license: String<br>driver: Driver<br>passengers: integer<br>brand: String<br>model: String |
|                                                                                                  |

| UberBlack                                                                                                                                                                         |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| id<br>license: String<br>driver: Driver<br>passengers: integer<br>typeCarAccepted: Map<String, ArrayList<String, Integer>><br>seatsMaterial: ArrayList<String> = leather on vinyl |
|                                                                                                                                                                                   |

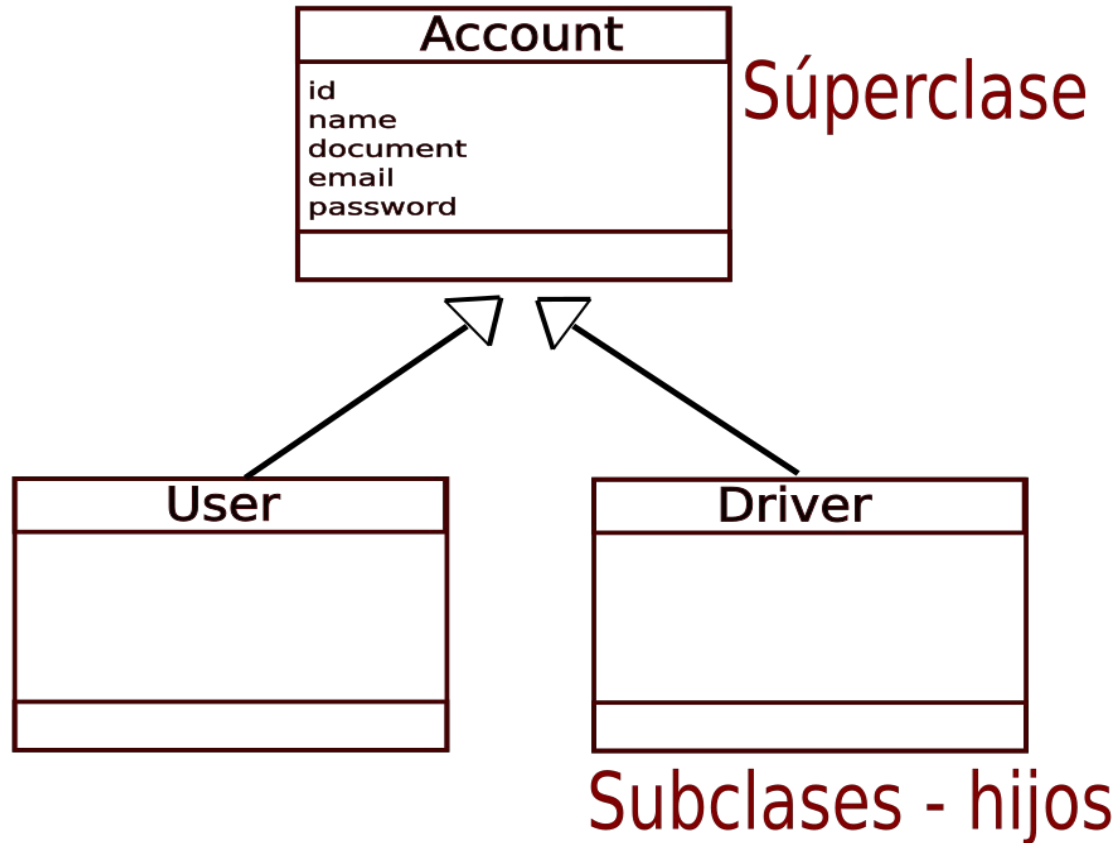
| UberVan                                                                                                                                                                           |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| id<br>license: String<br>driver: Driver<br>passengers: integer<br>typeCarAccepted: Map<String, ArrayList<String, Integer>><br>seatsMaterial: ArrayList<String> = leather on vinyl |
|                                                                                                                                                                                   |

| Card                                                  |
|-------------------------------------------------------|
| id<br>number: integer<br>cvv: integer<br>date: String |
|                                                       |

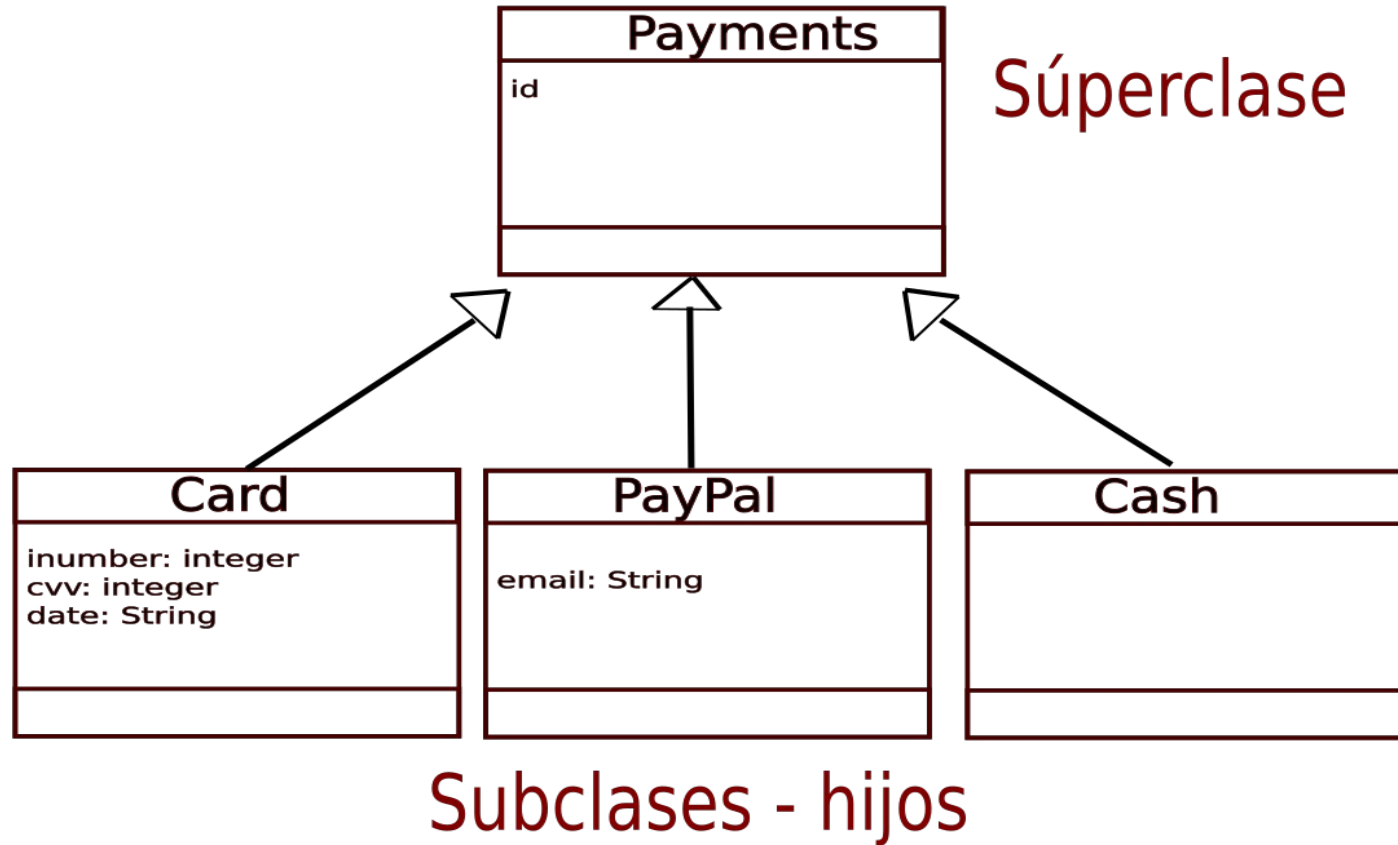
| PayPal              |
|---------------------|
| id<br>email: String |
|                     |

| Cash |
|------|
| id   |
|      |

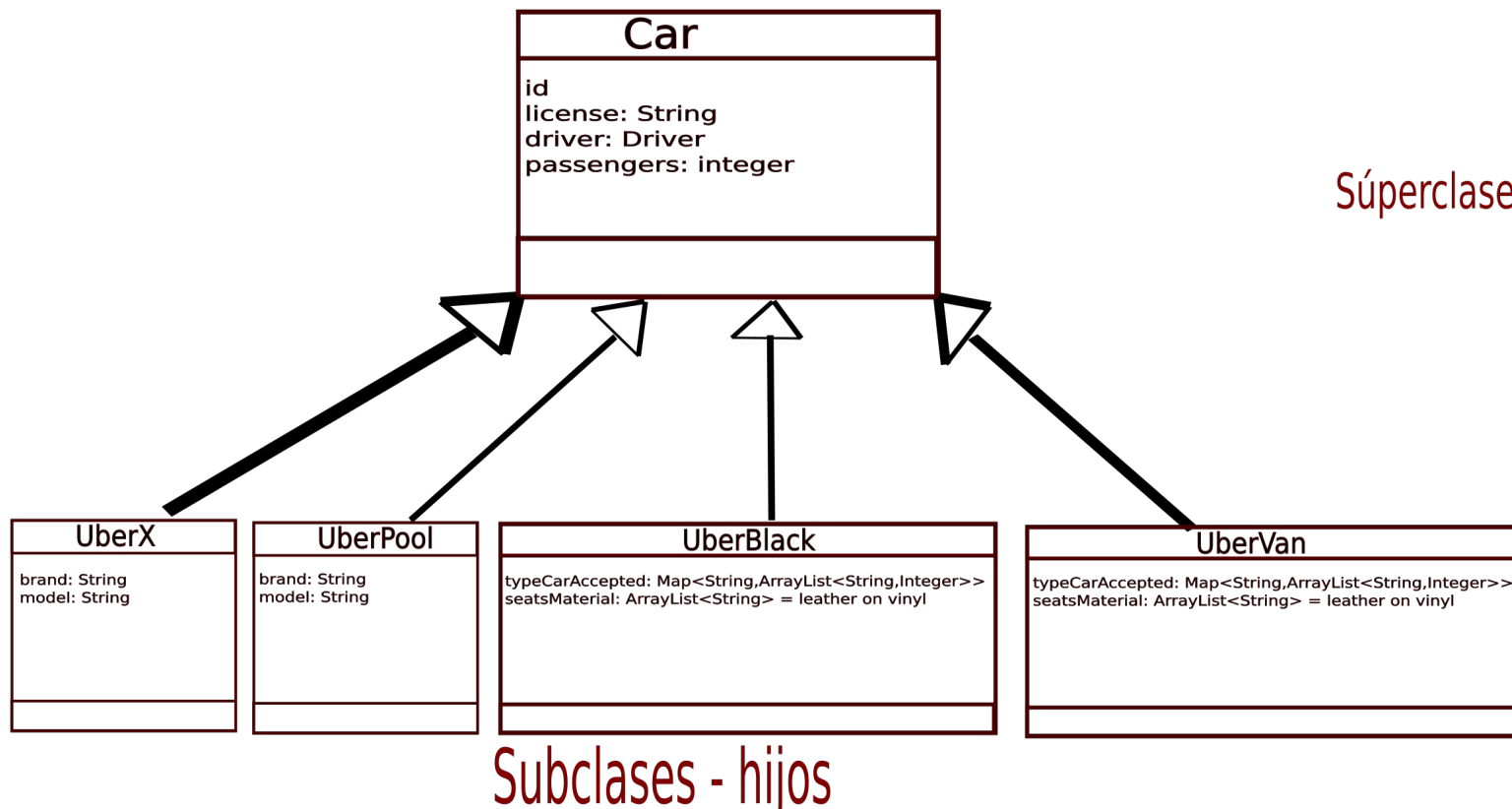
# Aplicando Herencia a nuestro proyecto Uber



# Aplicando Herencia a nuestro proyecto Uber

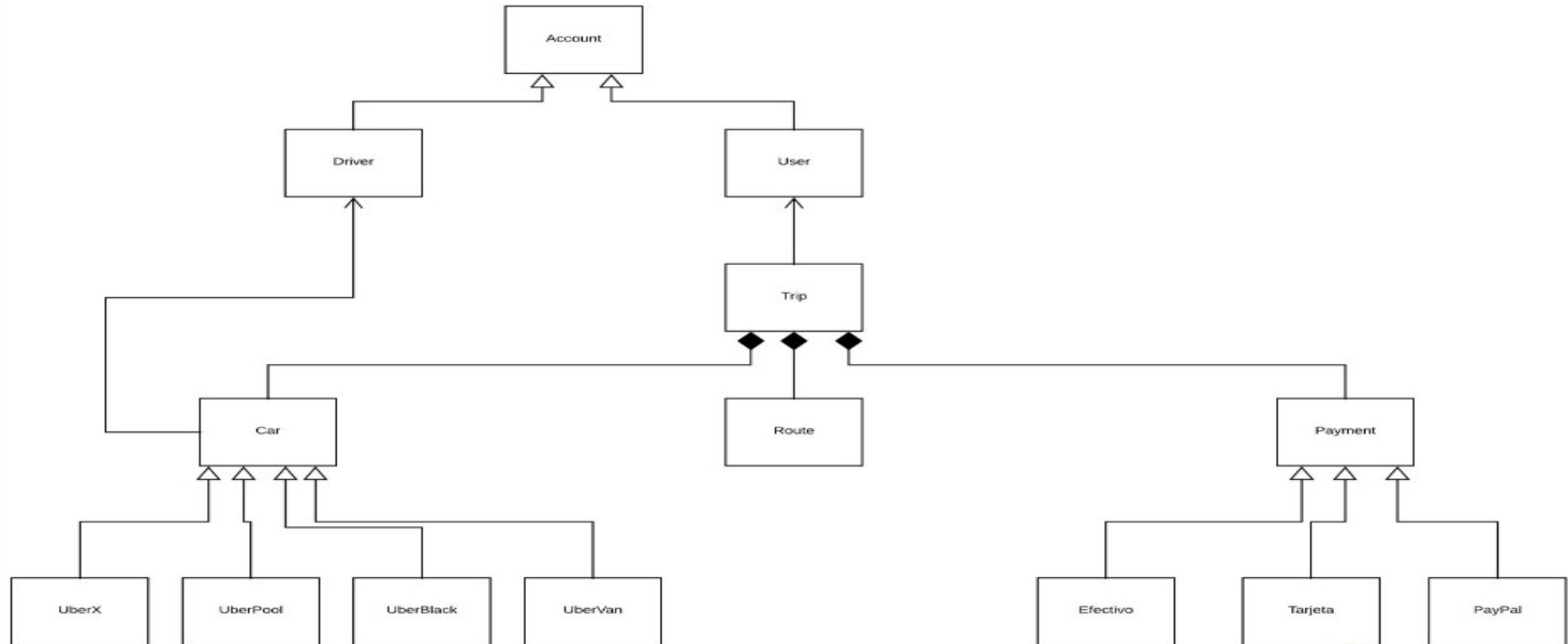


# Aplicando Herencia a nuestro proyecto Uber





# Aplicando Herencia a nuestro proyecto Uber

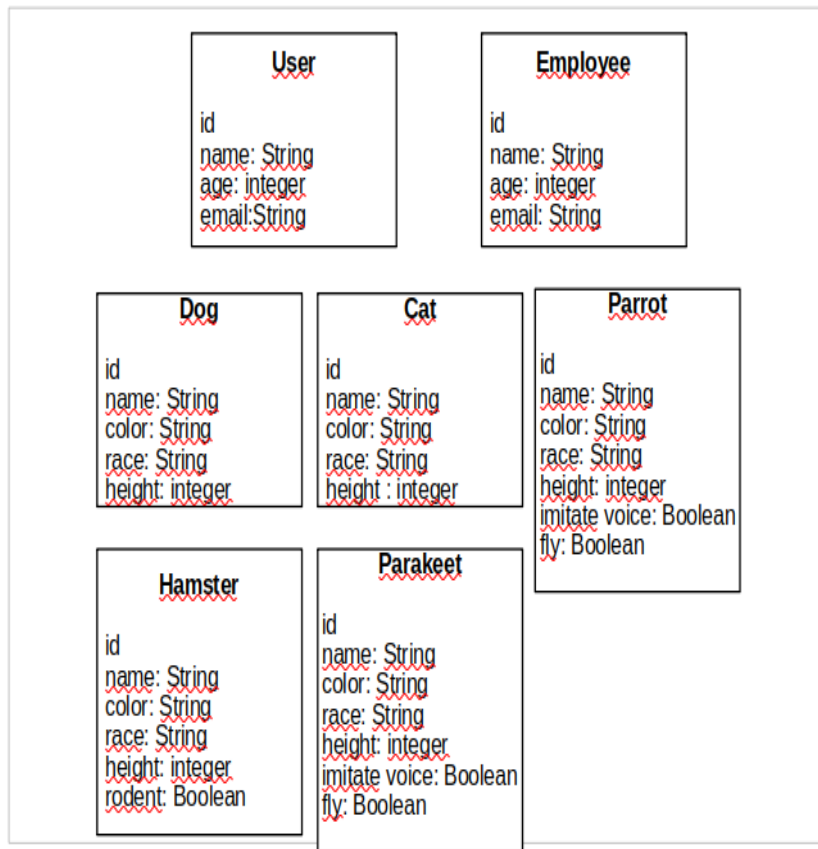


# Reto 2: analicemos un problema

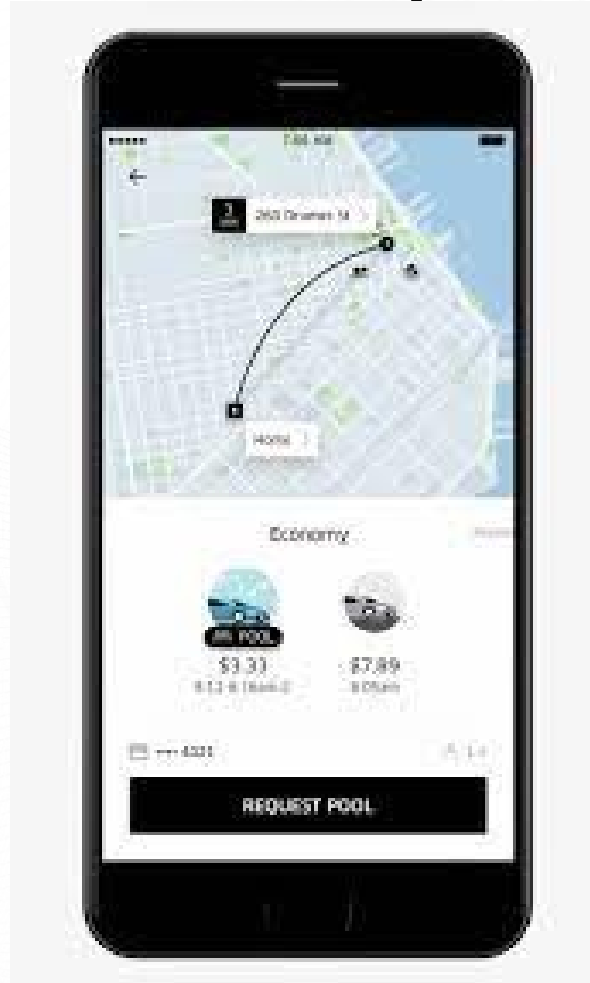


Imagina que nuestro sistema de adopciones creció y ahora ofrece adoptar pericos, loros, gatos y hamsters. Genera un nuevo análisis, aplica herencia para abstraer mejor el problema y lograr modularidad en el software.

- Lo primero que se hace es definir las clases u objetos que van a existir



# 02 Codificación ejercici Uber en java



# Definiendo clases en Java



A continuación sigue la etapa de programación del proyecto Uber:

1. instalar y configurar vs code para trabajar con JAVA
2. Crear proyecto Java en vs code.
3. Ejecutar el archivo App.java
4. Crear un paquete llamado Modelo.
- 5, Crear cada una de las clases que se modelaron en el diagrama de clases.

# Objetos, método constructor y su sintaxis en código



Los objetos nos ayudan a crear instancia de una clase, el objeto es el resultado de lo que modelamos, de los parámetros declarados y usaremos los objetos para que nuestras clases cobren vida.

Para el caso de Java crear una instancia de una clase se hace de la siguiente forma

```
Person person = new Person();
```

Person hace referencia a la Clase creada

person es el nombre o instancia del objeto proveniente de la clase, es el objeto que adquiere todos los atributos de la clase Person

new Person() hace referencia al metodo constructor de la clase, su representacion en todos los casos es finalizando con un parentesis () los metodos van escritos con la primer letra mayuscula y tienen el mismo nombre de la clase que es Person()

# Objetos, método constructor y su sintaxis en código



El Metodo constructor tiene la siguiente funcionalidad:

- Dar un estado inicial al objeto
- Tiene el mismo nombre de la clase
- Son los parametros minimos que necesita el objeto para que pueda vivir

# Objetos, método constructor y su sintaxis en código



Los métodos constructores dan un estado inicial al objeto y podemos añadirle algunos datos al objeto mediante estos métodos. Los atributos o elementos que pasemos a través del constructor serán los datos mínimos que necesita el objeto para que pueda vivir.

En el caso de Java lo primero que va es el modificador de acceso public, seguido del nombre del método que es el mismo nombre de la clase el cual es Person, después vienen los atributos, en Java debe ser declarado el tipo de variable o atributo y para tener acceso al objeto se hace mediante la sentencia `this.atributo`

Java

```
public Person(String name){  
    this.name = name;  
}
```

# Objetos. Dando vida a nuestras clases en Java



Los métodos constructores dan un estado inicial al objeto y podemos añadirle algunos datos al objeto mediante estos métodos. Los atributos o elementos que pasemos a través del constructor serán los datos mínimos que necesita el objeto para que pueda vivir.

En el caso de Java lo primero que va es el modificador de acceso public, seguido del nombre del método que es el mismo nombre de la clase el cual es Person, después vienen los atributos, en Java debe ser declarado el tipo de variable o atributo y para tener acceso al objeto se hace mediante la sentencia `this.atributo`

Java

```
public Person(String name){  
    this.name = name;  
}
```

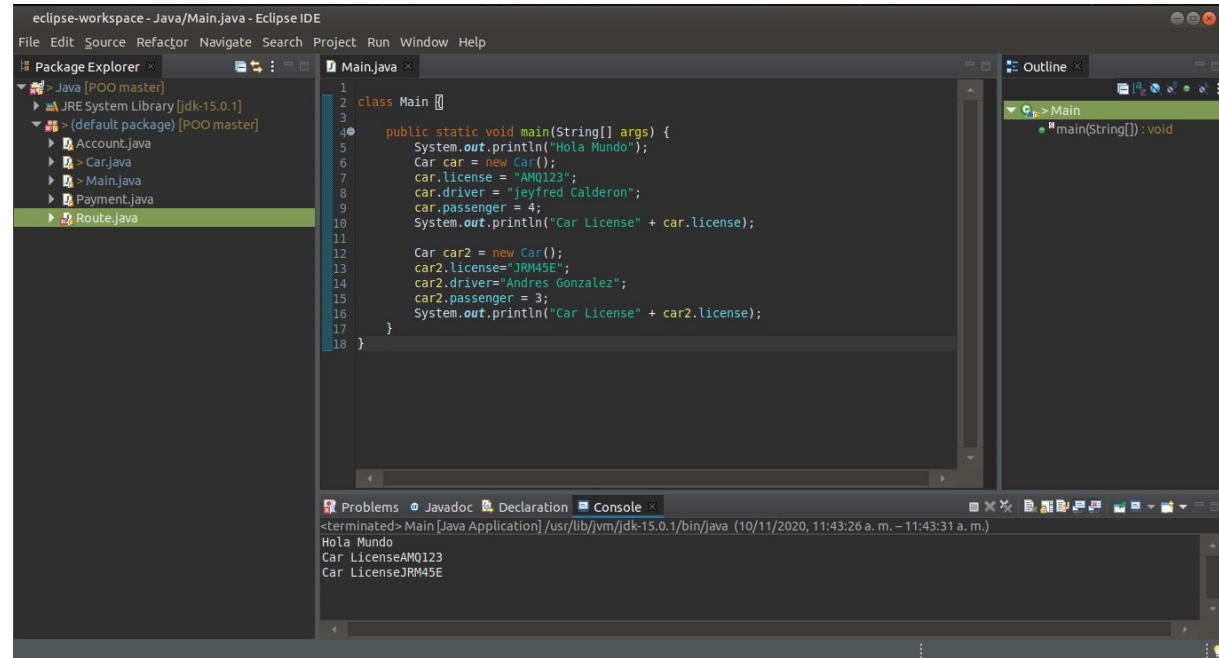


# Objetos, método constructor y su sintaxis en código



Abrir el archivos App.java para empezar a instanciar cada una de las clases creadas en clase anteriores

Aqui se hace instancia de la clase Car, para acceder a cada uno de sus atributos se usa el nombre del objeto seguido de un punto car. y el nombre del metodo, estos pueden ser accedidos porque el modificador de acceso en la clase es publico

A screenshot of the Eclipse IDE interface. The Package Explorer on the left shows a project named 'Java' with a sub-package 'default package' containing several classes, including 'Main.java'. The Main.java editor in the center shows the following code:

```
1 class Main {  
2  
3  
4 public static void main(String[] args) {  
5     System.out.println("Hola Mundo");  
6     Car car = new Car();  
7     car.license = "AMQ123";  
8     car.driver = "Jeyfred Calderon";  
9     car.passenger = 4;  
10    System.out.println("Car License" + car.license);  
11  
12    Car car2 = new Car();  
13    car2.license="JRM45E";  
14    car2.driver="Andres Gonzalez";  
15    car2.passenger = 3;  
16    System.out.println("Car License" + car2.license);  
17 }  
18 }
```

The Outline view on the right shows the 'main(String[]) : void' method. The Console at the bottom shows the output of the program:

```
<terminated> Main [Java Application] /usr/lib/jvm/jdk-15.0.1/bin/java (10/11/2020, 11:43:26 a. m. - 11:43:31 a. m.)  
Hola Mundo  
Car LicenseAMQ123  
Car LicenseJRM45E
```

# Objetos, método constructor y su sintaxis en código



Abrir el archivo Car.java y añadir el metodo

```
void printDataCar(){  
    System.out.println("License: "  
+ license + " Driver: " + driver);  
}
```

A screenshot of an IDE window showing the 'Car.java' file. The code defines a 'Car' class with attributes 'id', 'license', 'driver', and 'passenger'. A method 'printDataCar()' is implemented, which prints the license and driver information. The line numbers 1 through 12 are visible on the left side of the editor.

```
1 class Car {  
2     Integer id;  
3     String license;  
4     String driver;  
5     Integer passenger;  
6  
7     void printDataCar(){  
8         System.out.println("License: " + license + " Driver: " + driver);  
9     }  
10  
11 }  
12
```

# Objetos, método constructor y su sintaxis en código



y nuevamente pasar al archivo Main.java para llamar al metodo en este caso se edita el archivo de la siguiente forma

A screenshot of the Eclipse IDE interface. The Package Explorer on the left shows a project named 'Java [POO master]' with several files, including 'Car.java' and 'Main.java'. The main editor window displays the code for 'Main.java'. The code defines a 'Main' class with a 'main' method that creates two 'Car' objects, initializes their attributes (license, driver, passenger), and calls the 'printDataCar()' method for each. The Console at the bottom shows the output of the program, which prints 'Hola Mundo' followed by the details of the two cars.

```
1 class Main {
2
3
4     public static void main(String[] args) {
5         System.out.println("Hola Mundo");
6         Car car = new Car();
7         car.license = "AMQ123";
8         car.driver = "jeffred Calderon";
9         car.passenger = 4;
10        car.printDataCar();
11
12        Car car2 = new Car();
13        car2.license="JRM45E";
14        car2.driver="Andres Gonzalez";
15        car2.passenger = 3;
16        car2.printDataCar();
17    }
18 }
```

Console Output:

```
<terminated> Main [Java Application] /usr/lib/jvm/jdk-15.0.1/bin/java (10/11/2020, 12:07:15 p. m. – 12:07:20 p. m.)
Hola Mundo
License: AMQ123 Driver: jeffred Calderon
License: JRM45E Driver: Andres Gonzalez
```

# Declarando un Método Constructor en Java



Ahora con el lenguaje Java se van a empezar a contruir en cada una de las clases el metodo constructor, para esto abrir Car.java y colocar el metodo `public Car()`, lo que va dentro de los parentesis son los parametros los cuales van a obligar que para que sea creado un objeto Car debe contener los mismos parametros del metodo constructor, estos parametros obligatorios seran la licencia y el conductor, como estaba el diagrama se habia indicado que Car heredaba de la clase Account por tanto se cambia la palabra String por Account

# Aplicando herencia en lenguaje Java



Cada uno de los lenguajes tiene una forma de heredar de una clase padre, en su mayoría la forma en la que heredan es similar y lo que quiere decir en los ejemplos a continuación es que la clase estudiante hereda de una clase superior que sería la clase persona y estudiante puede adquirir o no atributos de la clase Persona

Java

```
class Student extends Person
```

# Aplicando herencia en lenguaje Java



Dentro de la carpeta modelo crear los objetos que van a heredar de la clase Car los cuales serán:

- UberX.java

Los atributos de la clase como ya se habia definido en el diagrama UML que va a llevar la clase UberX era brand y model, así que lo primero que se hace es crear los atributos, seguido de esto va el metodo constructor pero como esta extendiendo o heredado de la clase Car debe tener los mismos parametros del metodo constructor de la clase Car que eran license y driver, seguido de esto los atributos que se quieran colocar de la clase UberX, lo importante es que el metodo constructor obtenga los mismos parametros de la clase padre y para hacer referencia a que esta heredando de la clase padre se agrega debajo la sintaxis `super(license, driver)`

# Aplicando herencia en lenguaje Java



```
class UberX extends Car {  
  
    String brand;  
  
    String model;  
  
    public UberX(String license, Account driver, String brand, String model){  
  
        super(license, driver);  
  
        this.brand = brand;  
  
        this.model = model;  
  
    }  
  
}
```

# Aplicando herencia en lenguaje Java



UberPool.java

```
class UberPool extends Car {  
  
    String brand;  
  
    String model;  
  
    public UberPool(String license, Account driver, String brand, String model){  
  
        super(license, driver);  
  
        this.brand = brand;  
  
        this.model = model;  
  
    }  
  
}
```



# Aplicando herencia en lenguaje Java



## UberBlack.java

Para el UberBlack es necesario utilizar dos funciones de la libreria java.util para poder importar el ArrayList y el Map y de esta forma pasar los atributos a la clase

```
import java.util.ArrayList;

import java.util.Map;

class UberBlack extends Car{

    Map<String, Map<String,Integer>> typeCarAccepted;

    ArrayList<String> seatsMaterial;

    public UberBlack(String license, Account driver, Map<String, Map<String,Integer>> typeCarAccepted,ArrayList<String> seatsMaterial){

        super(license, driver);

        this.typeCarAccepted = typeCarAccepted;

        this.seatsMaterial = seatsMaterial;

    }
```

# Aplicando herencia en lenguaje Java



UberVan.java

```
import java.util.ArrayList;
```

```
import java.util.Map;
```

```
class UberVan extends Car{
```

```
    Map<String, Map<String,Integer>> typeCarAccepted;
```

```
    ArrayList<String> seatsMaterial;
```

```
    public UberVan(String license, Account driver, Map<String, Map<String,Integer>> typeCarAccepted,ArrayList<String>  
seatsMaterial){
```

```
        super(license, driver);
```

```
        this.typeCarAccepted = typeCarAccepted;
```

```
        this.seatsMaterial = seatsMaterial;
```

```
    }
```

# Aplicando herencia en lenguaje Java



## UberBlack.java

Para el UberBlack es necesario utilizar dos funciones de la libreria java.util para poder importar el ArrayList y el Map y de esta forma pasar los atributos a la clase

```
import java.util.ArrayList;

import java.util.Map;

class UberBlack extends Car{

    Map<String, Map<String,Integer>> typeCarAccepted;

    ArrayList<String> seatsMaterial;

    public UberBlack(String license, Account driver, Map<String, Map<String,Integer>> typeCarAccepted,ArrayList<String> seatsMaterial){

        super(license, driver);

        this.typeCarAccepted = typeCarAccepted;

        this.seatsMaterial = seatsMaterial;

    }
```

- <https://github.com/jeyfredc/Programacion-orientada-a-objetos-POO-/blob/master/README.md#Clase-11-Analizando-Uber-en-Objetos>

# Gracias



## CAPITULO 3

---

# CONCEPTOS FUNDAMENTALES DE LA ORIENTACION A OBJETOS

### 3.1. ¿QUE ES LA PROGRAMACIÓN ORIENTADA A OBJETOS?

La POO no es un lenguaje de programación. La POO es una nueva manera de "atacar" los problemas de programación, especialmente los relativos a grandes y medianos proyectos, una manera más fácil de implementar la programación estructurada. En efecto, la POO se ha basado en la programación estructurada para implementar conceptos innovadores que simplifican la creación de programas; permite dividir un problema en pequeñas unidades lógicas de código, independientes del resto del programa, que interactúan entre sí. A estas pequeñas unidades lógicas de código se les ha denominado objetos para establecer una analogía entre las mismas y los objetos materiales del mundo real. Para comprender el código de aplicaciones complejas creadas mediante POO los desarrolladores del mismo sólo necesitan entender los mensajes que los objetos se envían entre sí para entender cómo funciona el programa. Para enviarle una orden a un objeto sólo es necesario proporcionarle la información de lo que queremos que haga, sin preocuparnos por entender exactamente cómo se va a ejecutar la tarea.

Finalmente podemos decir que la POO es un técnica de programación que aumenta la velocidad de desarrollo de los programas y hace que su mantenimiento sea más fácil al volver a utilizar "objetos" que tienen comportamientos, características, y relaciones asociadas con el programa. Los objetos son organizados en grupos que están disponibles para la creación y mantenimiento de aplicaciones.

Según Grady Booch, autor del método de diseño orientado a objetos, define la programación orientada a objetos(POO) como:

***“Un método de implementación en el que los programas se organizan como colecciones cooperativas de objetos, cada uno de los cuales representan una instancia de alguna clase, y cuyas clases son todas***

***miembros de un jerarquía de clases unida mediante relaciones de herencia”.***

Los conceptos fundamentales de programación son: objetos, clases, herencia, mensajes y polimorfismo.

### 3.1.1. El Objeto

Un objeto es la pieza básica de la POO, es una representación o modelización de un objeto real perteneciente a nuestro mundo, por ejemplo; podemos tener un objeto perro que represente a un perro dentro de nuestra realidad, todos los objetos en la vida real tienen dos características: **estado y comportamiento**. El estado de un objeto esta dado por una serie de parámetros que lo definen y lo diferencian de objetos del mismo tipo.

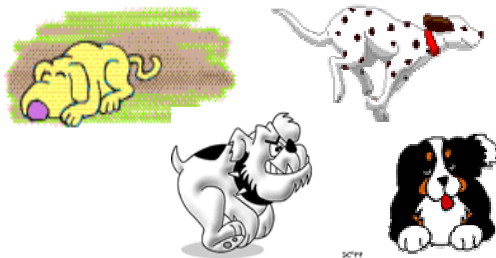


Figura 5. Representación de objetos

En el caso de tener un objeto perro, su estado estaría definido por su raza, color de pelo, tamaño, etc. Y el comportamiento viene definido por las acciones que pueden realizar los objetos, por ejemplo, en el caso del perro su comportamiento sería: saltar, correr, ladrar, etc.

Si tomamos un ejemplo que tiene que ver más con el mundo de la informática se pueden ver más claros estos dos conceptos. Si tenemos un objeto *pantalla* que representa la pantalla de nuestro ordenador, el *estado* de la misma estaría definido por los siguientes parámetros: encendida o apagada, tamaño, resolución, número de colores, etc.; y su comportamiento podría ser: imprimir, encender, apagar, etc.



Figura 6. Clase Pantalla

Los parámetros o variables que definen el estado de un objeto se denominan **atributos o variables miembro** y las acciones que pueden realizar los objetos se denominan **métodos o funciones miembro**; para indicar variables miembro y funciones miembro se utiliza el término general **miembro**.

Si lo comparamos con la programación estructurada los atributos o variables miembro serían variables y los métodos o funciones miembro procedimientos y funciones.

Los atributos de un objeto deben encontrarse ocultos al resto de los objetos, es decir, no se puede acceder directamente a ellos para modificar su estado o consultarlo.

Para acceder a los atributos de un objeto se deben utilizar **métodos**. Es decir, los métodos exponen toda la funcionalidad del objeto, mientras que los detalles del estado interno del objeto permanecen ocultos, incluso algunos métodos también pueden permanecer ocultos.

El hecho de ocultar la implementación interna de un objeto, es decir, como está construido y de que se compone se denomina **encapsulación**. La encapsulación es uno de los beneficios del paradigma de la Programación Orientada a Objetos.

Normalmente un objeto ofrece una **parte pública** que será utilizada por otros objetos para interactuar entre sí, pero también permanece una **parte oculta o privada** para encapsular los detalles de la implementación del objeto.

Ya se ha dicho que un objeto está compuesto de atributos y métodos. Como la caja negra de un avión, el objeto recubre la información que almacena y solamente podemos obtener la información e indicarle que realice acciones por medio de lo que comúnmente se denomina **interfaz del objeto**.



Los datos y la implementación queda oculta a los demás objetos que interaccionan en el programa, lo que favorece enormemente la protección de los datos y las estructuras internas contra las modificaciones externas al objeto, de este modo es mucho más sencillo localizar errores en los programas puesto que cada objeto está altamente especializado, y sólo se encarga de su tarea; como se puede observar, esto consigue una mayor modularidad, que facilita además el diseño en equipo de programas y la reutilización de clases creadas por otros desarrolladores.

Un objeto lo podemos representar como una rosquilla, donde la parte interna permanece oculta al mundo exterior conteniendo todos los detalles de la implementación del objeto, y la parte externa, que representa lo que el objeto muestra al mundo exterior y le permite utilizar para interactuar con él.

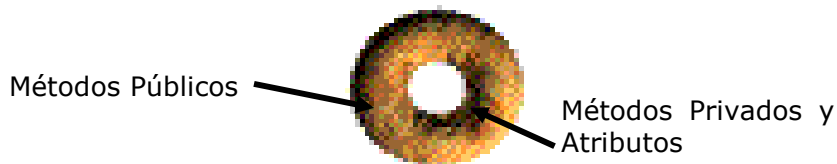


Figura 7. Composición del objeto

### 3.1.2. Métodos y Mensajes

Los objetos como entidades físicas (cafetera, persona, automóvil) o conceptuales (cuenta bancaria, mercado del dólar) responden a un conjunto determinado de mensajes.

La forma de comunicarse entre entidades es a través de **mensajes**. Un mensaje es un “mandato que el objeto debe obedecer”. Los objetos están entrenados para llevar a cabo las acciones de un conjunto determinado de mandatos.

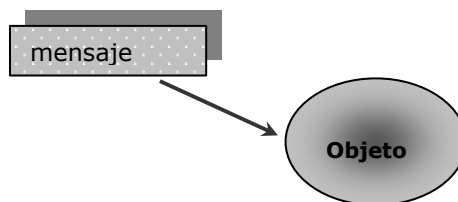


Figura 8. Paso de mensajes

Por ejemplo, podríamos indicarle a un objeto “**CUENTA BANCARIA**” que lleve a cabo la operación de depositar a la cuenta 100 pesos , entonces depositar es

el mensaje. El objeto incrementara su capital en 100 unidades y se puede confiar en que la operación se llevará a cabo correctamente.

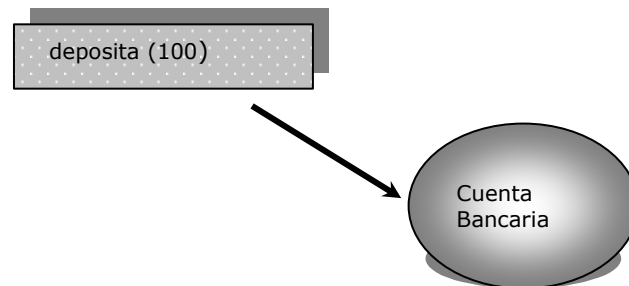


Figura 9. Paso de mensajes

Otros mensajes solicitan del objeto cierta información, por lo que el objeto debe responder dando esa información.

Por ejemplo, si a un objeto "**PERSONA**" le pedimos que nos dé su nombre, el objeto regresará la información.

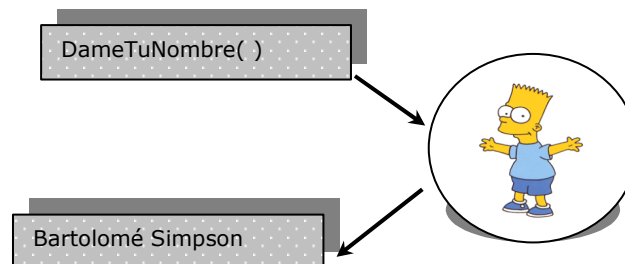


Figura 10. Paso de mensajes con retorno de información

En el caso del objeto "**CUENTA BANCARIA**", si le solicitamos que nos dé el saldo actual de la cuenta, el objeto responderá con el valor de su capital.

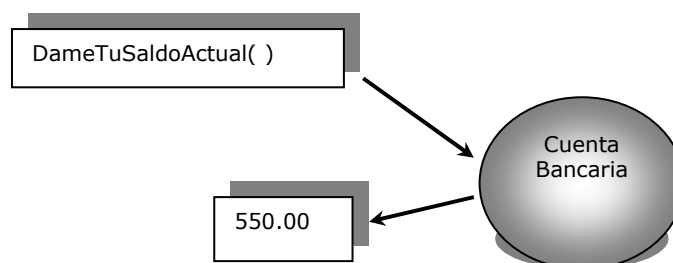


Figura 11. Paso de mensajes con retorno de información

### 3.1.3. Clases

En una aplicación vamos a encontrar que los objetos no están aislados y no son únicos. Vamos a observar incluso que hay objetos que se comportan de la misma forma.

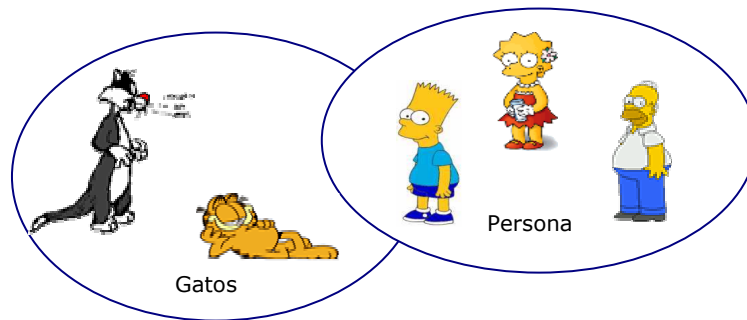


Figura 12. Representación de clases

Una **clase** define a un grupo de objetos que tienen el mismo comportamiento, es decir, responden al mismo conjunto de funciones. Además los objetos son instancias de las clases.

De esta forma los objetos; “silvestre”, “garfield”, “micifú”, pertenecen a la clase “**GATO**”, y los objetos “lisa”, “payaso”, “maggie”, pertenecen a la clase “**PERSONA**”.

Una clase se describe normalmente como la plantilla a partir del cual se hace realmente el objeto.

La forma normal de imaginarse las clases es pensando en ellas como si fuese un molde para galletas, mientras que el objeto, propiamente dicho, sería la galleta fabricada con dicho molde.

#### **Declaración Clases en C++**

Una clase en C++ debe ser declarada antes de utilizarla. El formato de declaración es el siguiente:

```
class Nombre
{
    //cuerpo de la clase
};
```

El cuerpo de la clase contiene:

- a. **Datos miembro:** Son declarados como variables pero no pueden ser inicializados.
- b. **Funciones miembro:** Son declaradas con prototipos de funciones y su definición puede hacerse interna o externa al cuerpo de la clase.

Los miembros de una clase pueden ser de :

- a. **Privados:** Accesibles solo por las funciones miembro de la clase, ocultan las estructuras de datos y los procedimientos internos de un objeto.
- b. **Públicos:** Accesibles desde cualquier parte del programa.
- c. **Protegidos:** Accesibles por las funciones miembro de la clase y por las funciones miembro de las clases derivadas.

```
class Nombre
{
    private:
        //miembros privados
    public:
        //miembros públicos
    protected:
        //miembros protegidos
};
```

### **Definición de clases**

La implementación de las funciones miembro constituyen la **definición** de la clase. En C++ se utiliza el siguiente formato:

```
tipo_retorno nombre_clase::nombre_funcion(lista de argumentos)
{
    //implementación...
}
```

#### **3.1.4. Herencia**

La herencia provee el mecanismo mas simple para especificar una forma alternativa de acceso a una clase existente, o bien para definir una nueva clase

que añada características nuevas a una clase existente. Esta nueva clase se denomina “clase derivada” y la clase existente “clase base”.

A partir de esta estructura jerárquica podemos definir los conceptos de Subclases y Superclases.

- “A” es SubClase de “B” si los objetos definidos por “A” también pertenecen a “B”.
- “B” es SuperClase de “A”, si “A” es SubClase de “B”.

Por ejemplo; la clase Niño es un subclase de la clase Persona que a su vez es subclase de la clase Mamífero(Elefante).

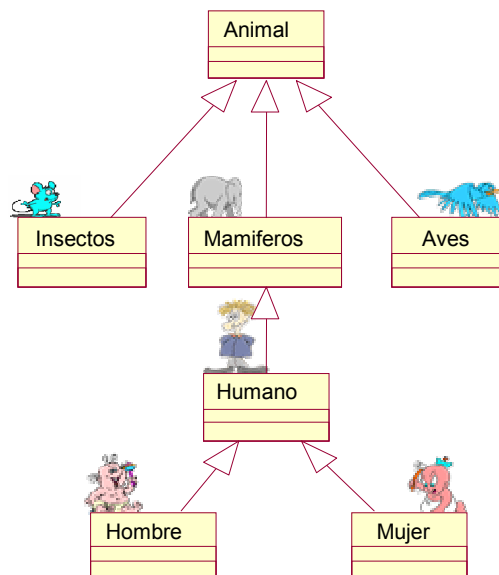


Figura 13. Jerarquía de clases

### **Clase Base**

Cuando necesitamos ampliar, en cualquier sentido, una clase existente o simplemente utilizarla a nuestro modo, la mejor solución es definir una clase derivada, ya que gracias al mecanismo de **herencia**, no tendremos que rescribir de nuevo todo el código que aporta la “clase base”, para seguir disponiendo de la funcionalidad que ella proporciona.

Esto quiere decir que la herencia es una forma de **reutilización de código** proporcionado por otras clases existentes.

### **Clase Derivada**

Es un tipo de dato definido por el usuario que tiene la propiedad de heredar los datos y funciones miembro de uno o más clases, definidas previamente, denominadas “clases base”.

Las clases derivadas(subclases) pueden heredar el código y los datos de su clase base(superclases), añadiendo su propio código especial y datos a ellas, incluso cambiar aquellos elementos de la clase base que necesita sean diferentes.

Con herencia si “A” es SubClase de “B”, entonces todas las propiedades de “B” también son propiedades de “A”. Se dice que la clase “A” hereda estas propiedades de la clase “B”.

Por ejemplo supongamos que definimos atributos para la siguiente estructura jerárquica:

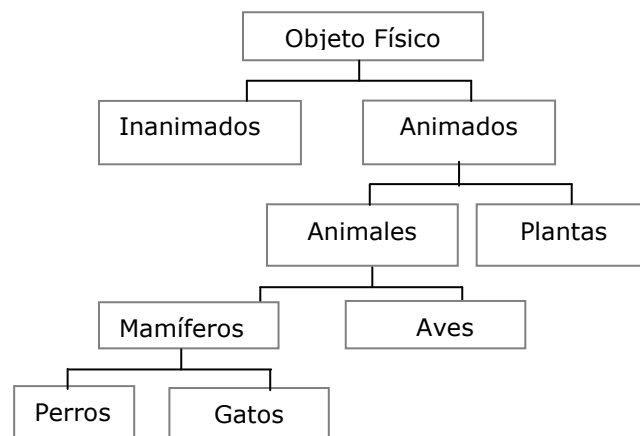


Figura 14. Estructura jerarquía Objeto Físico

- Los objetos físicos tienen un peso y dimensiones.
- Los objetos animados o vivos tienen una fecha de nacimiento.
- Los animales tienen un habitat y padres.
- Los mamíferos pertenecen a una raza.
- Los perros tienen a su amo.
- Los gatos cuentan con un número de canarios que han cazado.

Los atributos que se definen no son los únicos con los que cuenta dicha clase, sino que heredan también los atributos de todas las clases superiores a él.

En el ejemplo, los atributos con los que cuenta cada clase se definen en la siguiente tabla.

| Si un objeto se define de la clase: | Tiene las propiedades                                          |
|-------------------------------------|----------------------------------------------------------------|
| Objeto Físico                       | Peso,dimensiones                                               |
| Animados                            | Peso,dimensiones,FechaNacimiento                               |
| Animales                            | Peso,dimensiones,FechaNacimiento,habitat,padres                |
| Mamíferos                           | Peso,dimensiones,FechaNacimiento,habitat,padres,raza           |
| Perros                              | Peso,dimensiones,FechaNacimiento,habitat,padres,raza,amo       |
| Gatos                               | Peso,dimensiones,FechaNacimiento,habitat,padres,raza,#canarios |

Figura 15. Atributos de Objeto Físico

Tipos de Herencia

Herencia Simple

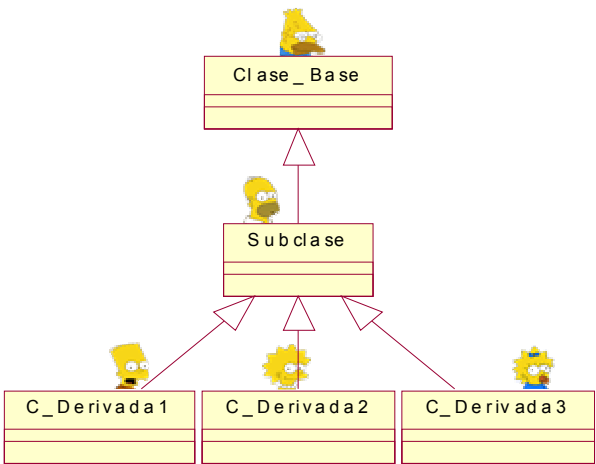


Figura 16. Herencia Simple

En este caso una clase solo se deriva de otra clase; las subclases heredan características de sus superclases. Las características pueden ser variables de instancias y/o métodos.

Las definiciones de clases para esta jerarquía pueden tomar la siguiente estructura:

```
clase abuelo
  atributos
    nombre: string;
    cédula: int;
    fecha_nacimiento: int;
  operaciones
    calcular_edad( );
    imprimir_edad( );
    ...
end abuelo.

clase padre hereda abuelo
```

```
atributos  
    sueldo: real;  
operaciones  
    calcular_sueldo( );  
    ...  
end padre.
```

```
clase hijo1 hereda padre  
atributos  
    estatura: real;  
operaciones  
    ...  
end hijo1.
```

### Herencia Múltiple

Es el mecanismo que permite a una clase tener mas de una clase base directa, es decir más de un ascendiente inmediato. Se puede decir entonces que una clase es una extensión de dos o mas clases.

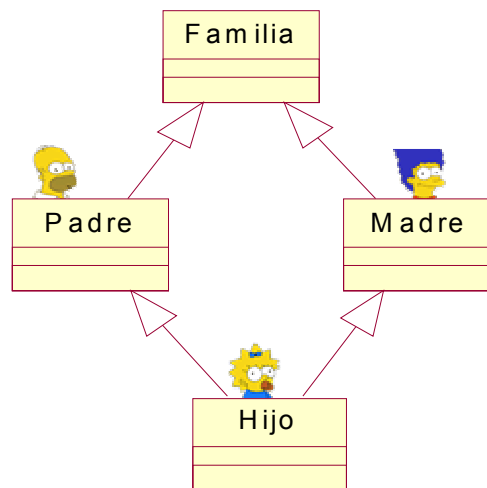


Figura 17. Herencia Múltiple

Con herencia múltiple se pueden combinar diferentes clases existentes para producir clases que utilizan cada una de sus múltiples superclases.

En la estructura jerárquica anterior **Maggie Simpsons** hereda características de la clase **padre Homero Simpsons** y a su vez de la clase **madre Marge Simpsons**; añadiendo características propias.

#### 3.1.5. Clases Abstractas

Al momento de derivar clases es muy probable que lleguemos a formar nuevas clases que no tengan otro fin que el de agrupar subclases, indicando solamente qué funciones deben compartir entre ellas.



Por ejemplo, supongamos que estamos realizando un sistema para calcular la nómina de una empresa y nos encontramos que existen tres diferentes tipos de empleados a los cuales se les paga de distinta forma:

- Empleado de planta, recibe un sueldo fijo.
- Empleado ocasional, se le paga por horas trabajadas.
- Vendedor, se le paga un porcentaje por comisiones.

Ahora bien, dado que todos ellos manejan información común (nombre, NIT, etc) se decide definir una clase superior “**EMPLEADO**”, de la cual las demás clases se deriven.

Esta clase Empleado tendrá la información y las funciones comunes, pero ¿Tiene caso definir un objeto de la clase Empleado? Por la definición del problema, el objeto debe ser definido como Planta, Ocasional o Vendedor. La clase Empleado sólo nos ayuda a agrupar a las subclases y a definir el conjunto de funciones comunes.

Por ejemplo, todas las subclases deben definir e implementar su propia función sueldo( ) que devuelva la cantidad de dinero que le corresponde a cada uno, la

cual será diferente, pero dado que esta es una función que todos utilizan, cualquier objeto de esta clase deberá contar con una función sueldo( ); también es necesario definirla en la clase Empleado.

Lo único que interesa en la clase Empleado es indicar que la función sueldo( ) existe y cuáles son los valores que recibe y devuelve la función. El cómo debe calcular la función se definirá en sus subclases, ya que dicha función se calculará diferente para cada tipo de Empleado.

De esta forma a las clases como Empleado se les conoce como **Clases Abstractas**.

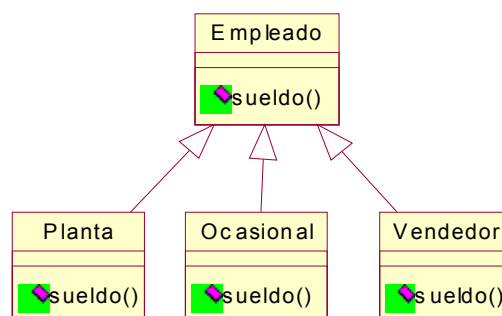


Figura 18. Clase Abstracta Empleado

Una **Clase Abstracta** es una clase que puede utilizarse solamente como clase base de otras clases para agrupar y capturar información que es común al grupo. No se pueden crear objetos a partir de ella, se crearan objetos a partir de sus clases derivadas las cuales son denominadas **clases concretas**.

En la jerarquía de clases se muestra la clasificación, en ella encontramos la clase abstracta Empleado y tres clases concretas, Planta, Ocasional y Vendedor. En las clases concretas se ha redefinido el método sueldo con la misma interfaz a fin de utilizarlo de igual forma independiente de la clase a la que pertenece al objeto, pero el resultado dependerá de la definición que se haya hecho del método en su respectiva clase.

### 3.1.6. Sobrecarga

La sobrecarga es uno de los mecanismos más utilizados de C++ pues reporta una gran cantidad de beneficios a la hora de diseñar las funciones miembro. Existen dos tipos fundamentales de sobrecarga: la sobrecarga de funciones y la sobrecarga de operadores.

#### **Sobrecarga de funciones**

La sobrecarga de funciones consiste, básicamente, en crear funciones con el mismo nombre dentro de una clase pero con distinto tipo de argumentos de tal forma que, al llamar a la función el compilador se encargará de escoger la adecuada mediante la comparación de la lista de argumentos pasados en la invocación. Un sencillo ejemplo sería pensar en dos funciones llamadas **"ImprimirMensaje"** Podemos definir las así:

```
void ImprimirMensaje(void);  
void ImprimirMensaje (char *msg);
```

Como podemos observar, se llaman igual pero tienen argumentos distintos, mientras que la primera no está definida para recibir ningún tipo de datos, la segunda esperará recibir una cadena de caracteres.

```
void NombreClase::ImprimirMensaje(void)  
{  
    cout << "\n Función sin argumentos";  
}
```

```
void NombreClase :: ImprimirMensaje  
                        (char *msg)  
{  
    cout << "\n Función con argumentos y ha  
            recibido el mensaje " << msg;  
}
```

Incluyendo estas dos funciones en una clase y creando una instancia de la misma, se puede llamar a la función ***ImprimirMensaje*** obteniendo los siguientes resultados:

**ImprimirMensaje( );**

Obtendríamos por pantalla:

*Esta es la función sin argumentos*

**ImprimirMensaje("Enlace Informático");**

Obtendríamos por pantalla:

*Esta es la función con argumentos y ha recibido el mensaje Enlace Informático*

### ***Sobrecarga de Operadores***

Permite redefinir ciertos operadores, como "+" y "-", para usarlos con las clases que hemos definido. Se llama sobrecarga de operadores porque estamos reutilizando el mismo operador con un número de usos diferentes, y el compilador decide cómo usar ese operador dependiendo sobre qué opera.

La sobrecarga de operadores sólo se puede utilizar con clases, no se pueden redefinir los operadores para los tipos simples predefinidos.

La sobrecarga de operadores en C++ tiene una serie de restricciones que es necesario tener presente a la hora de manejar operadores sobrecargados:

- Se pueden sobrecargar sólo los operadores definidos en C++
- La sobrecarga de operadores funciona sólo cuando se aplica a objetos de una clase
- No se puede cambiar la preferencia o asociatividad de los operadores en C++
- No se puede cambiar un operador binario para funcionar con un único objeto
- No se puede cambiar un operador unitario para funcionar con dos objetos
- No se puede sobrecargar un operador que funcione exclusivamente con punteros

#### **3.1.7. Polimorfismo**

El polimorfismo se da cuando un método funciona de varias maneras según el objeto que invoque a dicho método. Hay tres formas de implementar

polimorfismo: por sobreescritura (ligadura dinámica), por sobrecarga de funciones (ligadura estática).

El Polimorfismo es una característica de la Programación Orientada a Objetos que permite construir métodos para cada una de las clases derivadas que parten de una misma clase base, con el fin de que adopten comportamientos totalmente distintos.

El polimorfismo se asocia con funciones u objetos, donde una **función polimórfica** es aquella que tiene el mismo nombre para clases diferentes de la misma familia, pero diferentes implementaciones para las diversas clases, y; un **objeto polimórfico** es aquel que tiene el mismo nombre que los objetos de otras clases en un jerarquía de clases tales que cada objeto puede tener un comportamiento diferente.

Se puede afirmar que el **Polimorfismo** es un concepto realmente potente y que se lleva a cabo mediante la utilización de **funciones sobrecargadas o funciones virtuales**.

Las funciones sobrecargadas se llaman utilizando ligadura estática y las funciones virtuales se llaman utilizando ligadura dinámica.

### **Ligadura Dinámica**

Es un concepto que en programación estructurada se refiere a la asociación que hay entre la llamada a una función y el cuerpo de la función que finalmente se ejecuta. En tiempo de compilación se sabe que cuando llama a una función, va a ejecutar ese cuerpo.

En programación orientada a objetos como puede haber mas de una función con el mismo nombre (sobrecarga), se produce lo que se llama la **ligadura dinámica**; este tipo de ligadura hace que el compilador elija en tiempo de ejecución que función va a ejecutar finalmente. El compilador sabe a que función debe llamar viendo a que puntero esta referenciada.

La ligadura dinámica en C++ es implementada través de las **funciones virtuales**, no genera un código de un salto incondicional a una función concreta del programa, sino que deja esa decisión para cuando se ejecute el programa.

Será en tiempo de ejecución cuando decida a que función debe llamar (debido a que puede haber la sobrecarga de funciones). Esto lo sabe mediante el puntero **this**, si apunta a un objeto de clase A, llamará a la función de la clase A.

Como ejemplo tenemos la siguiente jerarquía de clases:

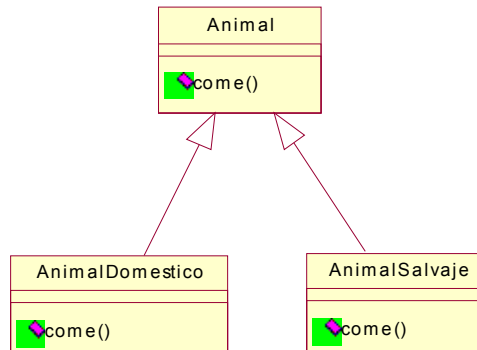


Figura 19. Clase Animal

Definimos la clase **Animal**, con el método **come( )** que imprime *"estoy comiendo"*.

Las clases hijas son:

- **AnimalDoméstico**, con el método **come( )** que *"busca a su dueño y se alimenta de su mano"*;
- **AnimalSalvaje**, con el método **come( )** que *"busca a su presa, la mata y se alimenta"*.

Ambas tienen el mismo método pero tendrán distintos comportamientos cuando una persona se acerque a alimentar a los animales.

### 3.1.8. Funciones Virtuales

Una función virtual es un mecanismo que permite a clases derivadas redefinir las funciones de las clases base. Dicho método, pese a ser común para los objetos derivados, es tratado de forma distinta y, por tanto, dando resultados también diferentes dependiendo de con qué clase lo invoquemos. Las funciones virtuales se definen en la clase base y son las que serán redefinidas, luego, en las clases derivadas.

La declaración de una función virtual se consigue mediante la palabra clave virtual precediendo a la declaración de la función. Por ejemplo:

```
virtual void PonInformacion(void);
```

En este caso, hemos definido una función virtual llamada **"PonInformacion"** que pertenece a una clase base y que podrá ser redefinida, completamente, en

una clase derivada para que actúe de forma totalmente distinta a cómo lo hace en la clase base. ***Es muy importante que la función de la clase base que vamos a redefinir lleve el identificador virtual*** pues de lo contrario no funcionará correctamente.

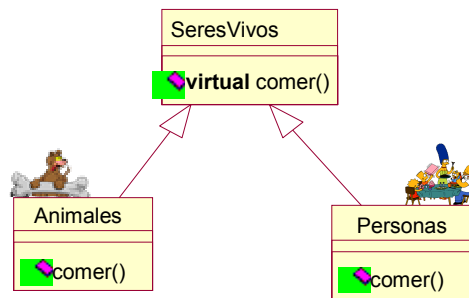


Figura 20. Clase seres vivos

En este caso, el método virtual se define en la clase base “**SeresVivos**”, y el código procedimental real utilizado se define en cada una de las subclases “**Animal**”, “**Personas**”.

La capacidad de utilizar funciones virtuales y ejecutar sobrecarga es lo que nos conduce a una de las características más importantes de los Sistemas Orientados a Objetos como es el ***Polimorfismo***.

# PARADIGMA ORIENTADO A OBJETOS



**Ing. Carolina Moreno**

Departamento de Sistemas  
Facultad de Electrónica y Telecomunicaciones  
Universidad del Cauca

## Contenido

1. Que es Orientación a Objetos?
2. Abstracción
3. Que es una Clase
4. Que es un Objeto
5. Mensajes
6. Métodos
7. Modularidad
8. Encapsulamiento
9. Jerarquías
10. Polimorfismo

# 1. QUE ES ORIENTACION A OBJETOS?

- No es un lenguaje de programación.
- Conjunto de disciplinas (ingeniería) que desarrollan y modelizan software que facilita la construcción de sistemas complejos a partir de componentes.
- Idea general: **Permite dividir el problema en pequeñas unidades de código independientes del resto del programa, que interactúan entre si.**
- Según Booch deben existir los siguientes elementos para ser un modelo O.O.:
  - Abstracción.
  - Modularidad.
  - Encapsulamiento.
  - Jerarquía.
  - Polimorfismo.

3

# 2. ABSTRACCIÓN

- Propiedad que permite representar las características esenciales de un objeto sin preocuparse de las demás.
- Describir una entidad del mundo real y utilizarla en un programa.
- Sirve para separar el comportamiento esencial de un objeto de su implementación.

## ABSTRACCIÓN DE CONTROL:

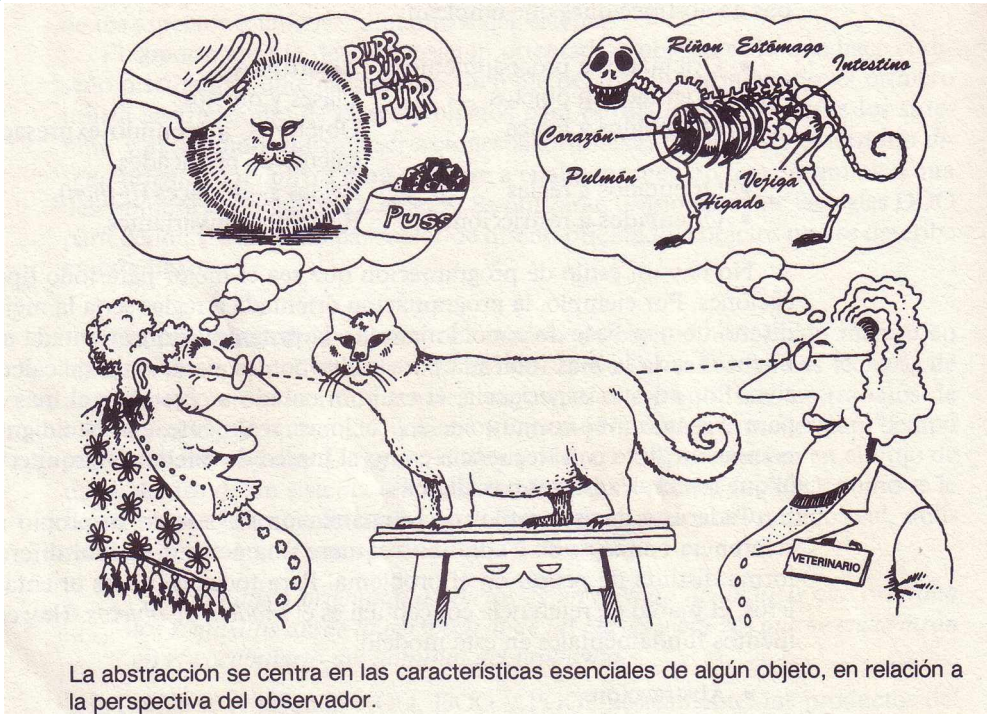
- Mecanismo para controlar el flujo y ejecución de las instrucciones.
- Describen el orden en que se ejecutan las instrucciones.

## ABSTRACCIÓN DE DATOS:

- Nuevos tipos de datos a los que ya existen (TAD's).
- Describir los datos que conforman el TAD y los procedimientos para manejar estos datos.

4



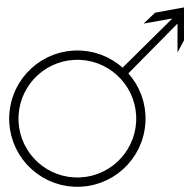


5



Objeto real : CASA

La elección de los detalles esenciales es subjetiva:



6

### 3. QUE ES UNA CLASE?

- Es un TAD que representa un objeto del mundo real.
- Cada clase es especializada y solo se encarga de su tarea.



- Una clase es una abstracción o visión generalizada de un conjunto de objetos del mundo real que tienen características (variables) y métodos (operaciones) comunes o similares.

7

#### Como identificar clases?

- Incidentes (vuelo, sucesos, accidentes, ..).
- Lugares (muelle, carretera, ...).
- Especificaciones (frío, caliente, alto,...).
- etc.

Pueden ser entidades:  
Físicas y Conceptuales.



Roles o Papeles  
(gerente, cliente, vendedor, ...)



Cosas tangibles  
(avión, auto, televisor,...).

Interacciones  
(transacciones,  
contratar, ...).

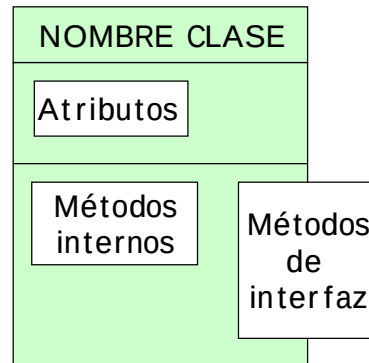


Organizaciones  
(empresa, departamento, ...)

8

## ESTRUCTURA DE UNA CLASE:

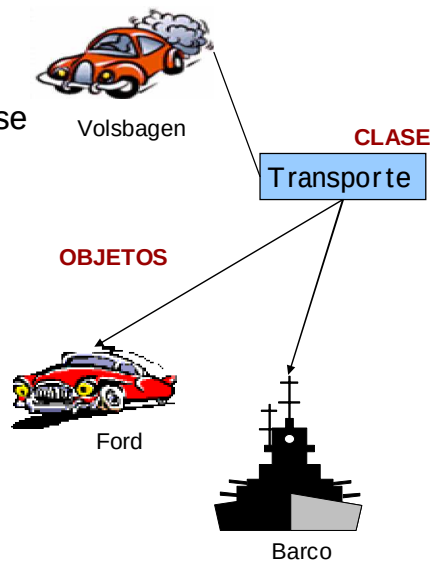
- **Nombre:** Permite distinguir una clase de otra.
- **Atributos y Propiedades:**
  - Características del objeto.
  - Parámetros que lo definen y lo diferencian de objetos del mismo tipo (Variables miembro y valores).
- **Identificador:** Permite distinguir un objeto de otro.
- **Estado:** Valores que toman los atributos en un tiempo determinado.
- **Métodos o Funciones miembro:**
  - Define el comportamiento de los objetos de una clase.
  - Acciones que manejan los datos.
  - Son respuesta a los mensajes.
- **Interfaz:** Servicios que prestan los objetos o formas de comunicarse con ellos.



9

## 4. QUE ES UN OBJETO?

- Se comunican con otros objetos pasándose mensajes.
- Un objeto es una instancia de una clase totalmente independiente de otros objetos de su misma clase.
- Instancia de una clase: ocupa espacio en memoria del tamaño de la clase y dura un tiempo determinado en memoria.
- Se pueden definir muchos objetos (instancias) de una misma clase.
- Los objetos tienen la posibilidad de realizar operaciones

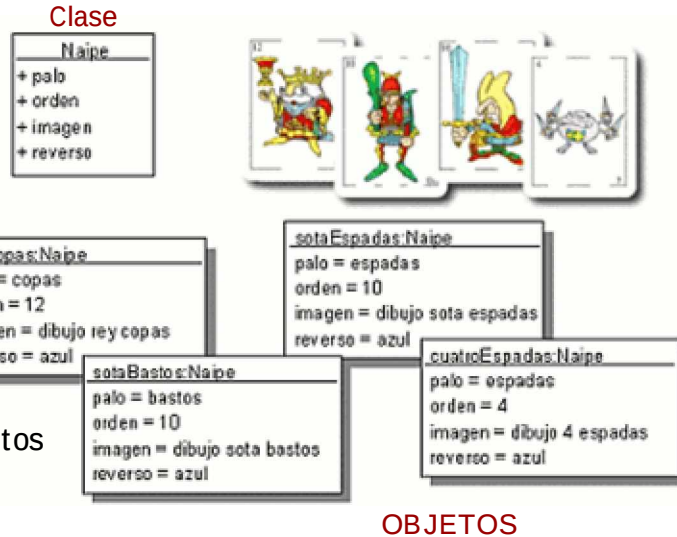


10

- Los objetos de una clase se diferencian entre ellos por su nombre, estado y principalmente por su identificador.

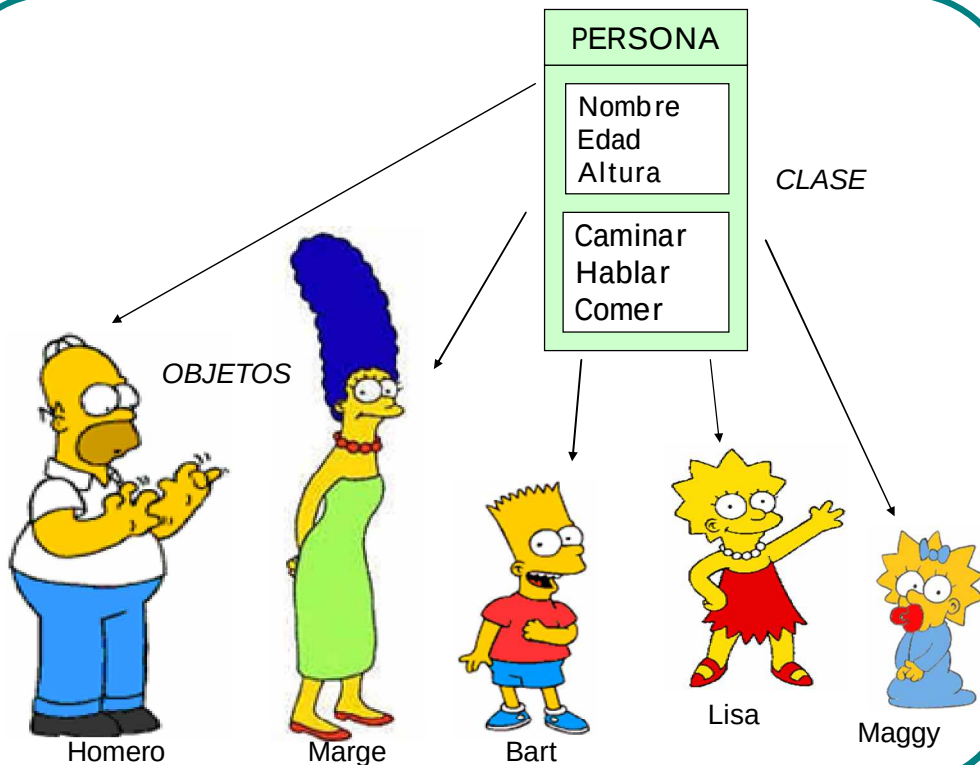
- Todos los objetos de una clase usan el mismo método en respuesta a mensajes similares.

- Un objeto se crea cuando un mensaje de petición de creación se recibe por la clase base.

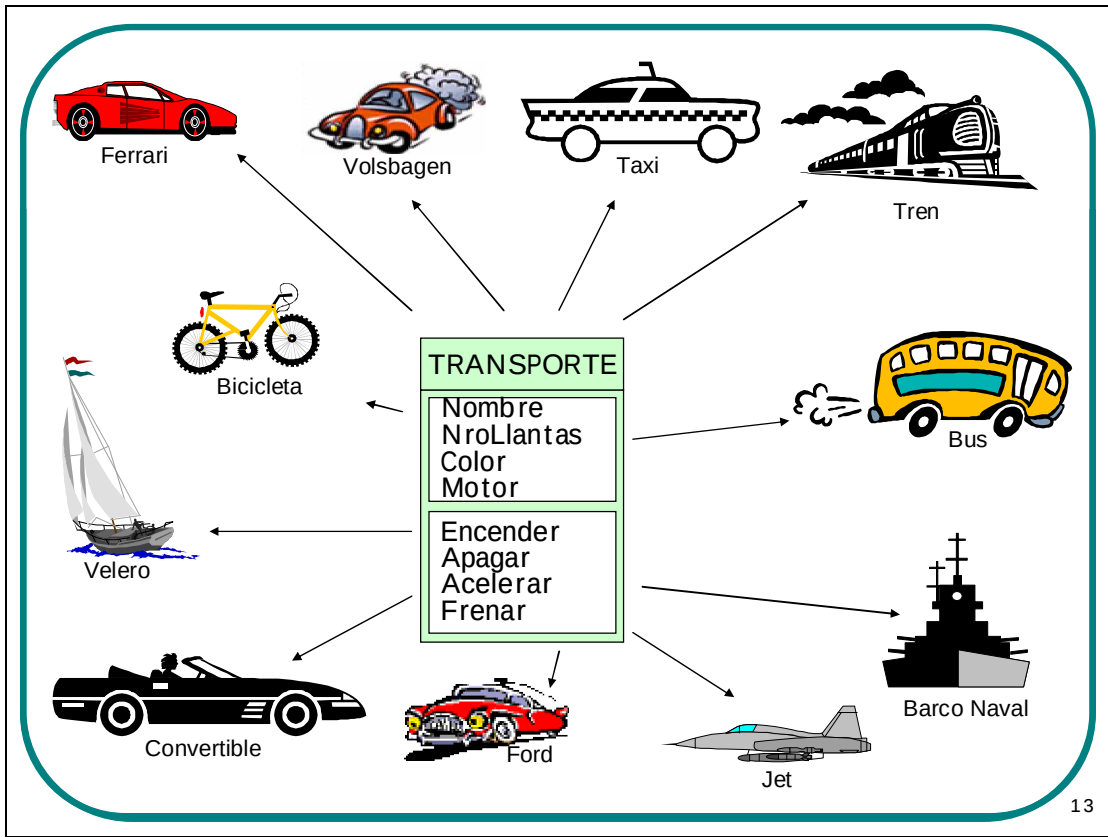
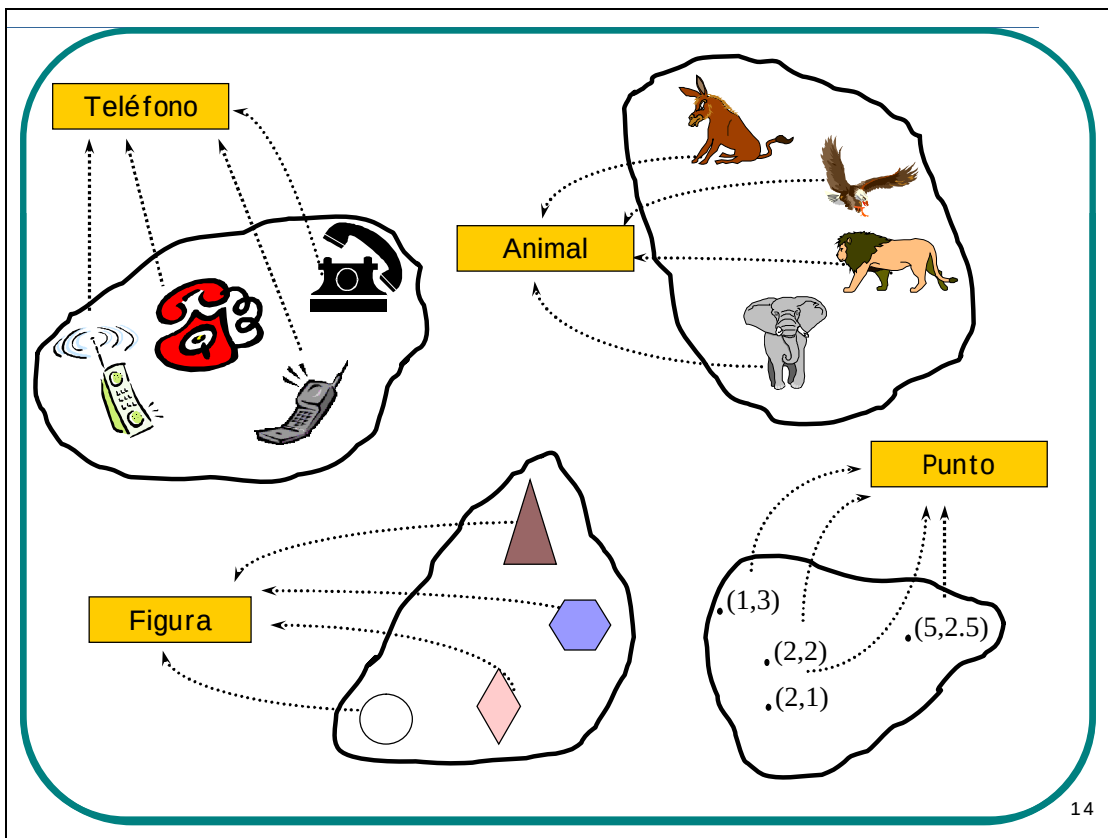


Clases ~ Tipos de datos  
Objetos ~ Variables

11

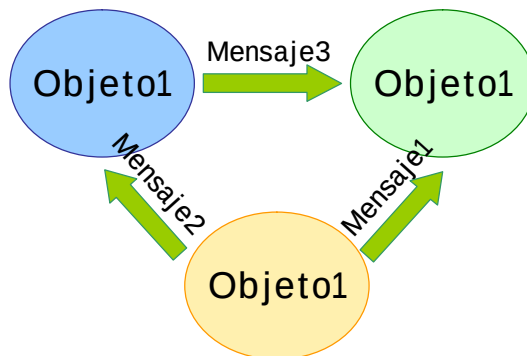


12



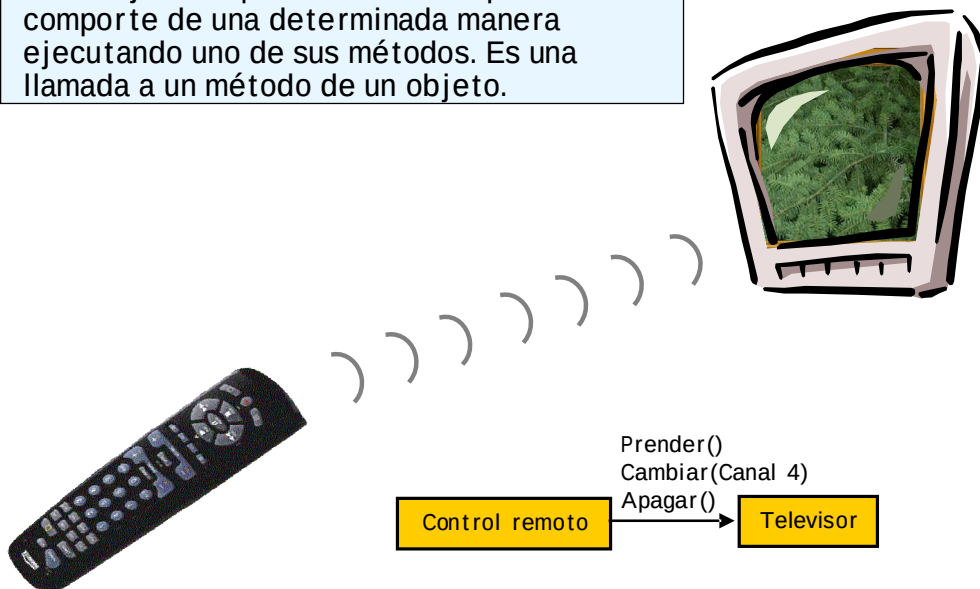
## 5. MENSAJES

- La forma como se comunican e interactúan los objetos es mediante el envío de mensajes.
- Un POO consiste de una serie de objetos que se comunican unos con otros llamando a métodos a través de mensajes. A través de la interacción de esos objetos, los programadores alcanzan un alto nivel de funcionalidad y comportamientos mas complejos.



15

Definición: Es una petición de un objeto a otro objeto al que se le solicita que se comporte de una determinada manera ejecutando uno de sus métodos. Es una llamada a un método de un objeto.



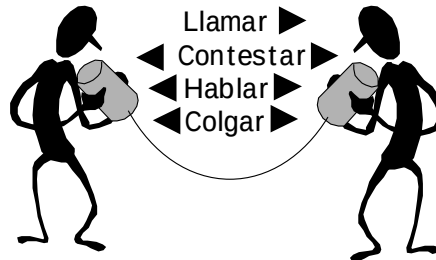
16



## Componentes de un mensaje

- El objeto que va a enviar el mensaje (Emisor)
- El objeto al cual se le envía el mensaje (Receptor)
- Nombre del mensaje o método que se va a ejecutar.
- Cualquier otra información que necesite el método para poder actuar (parámetros)

```
Juan.Llamar ( Nokia )  
Ana.Contestar(Nokia, Juan.GetNumCel())  
Ana.Hablar()  
Juan.Hablar()  
Juan.Colgar()  
Ana.Colgar()
```

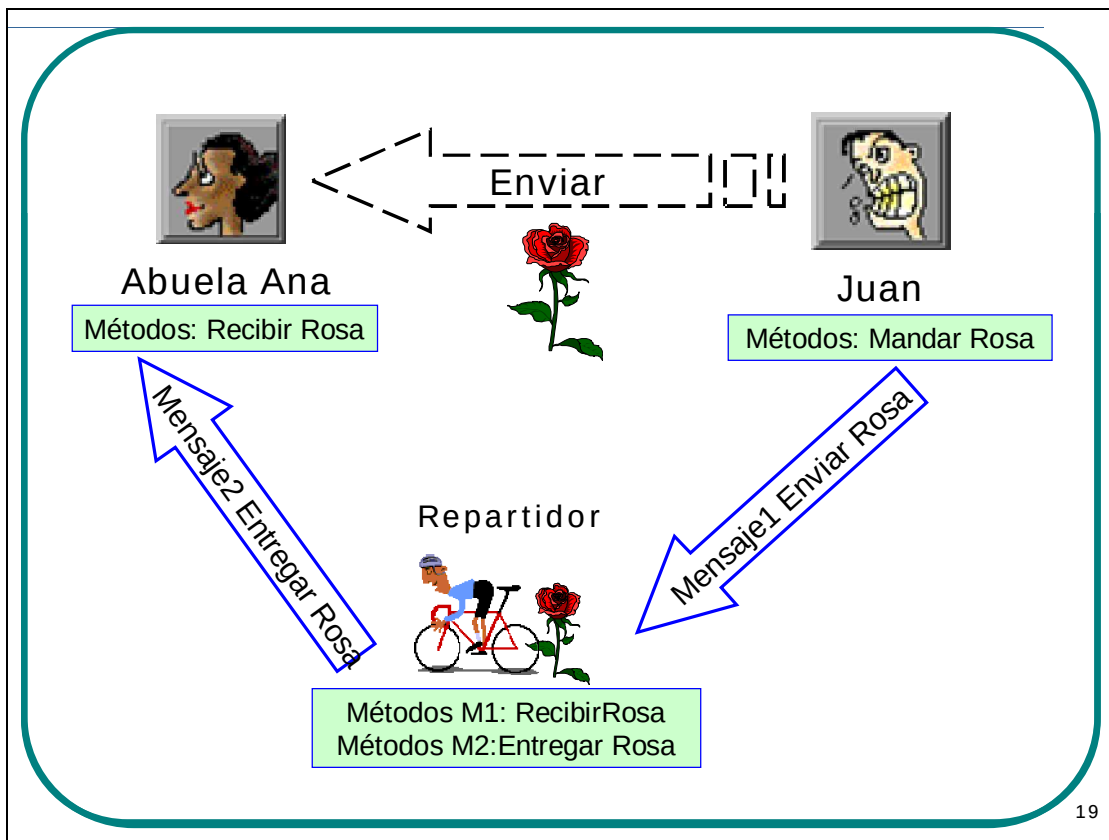
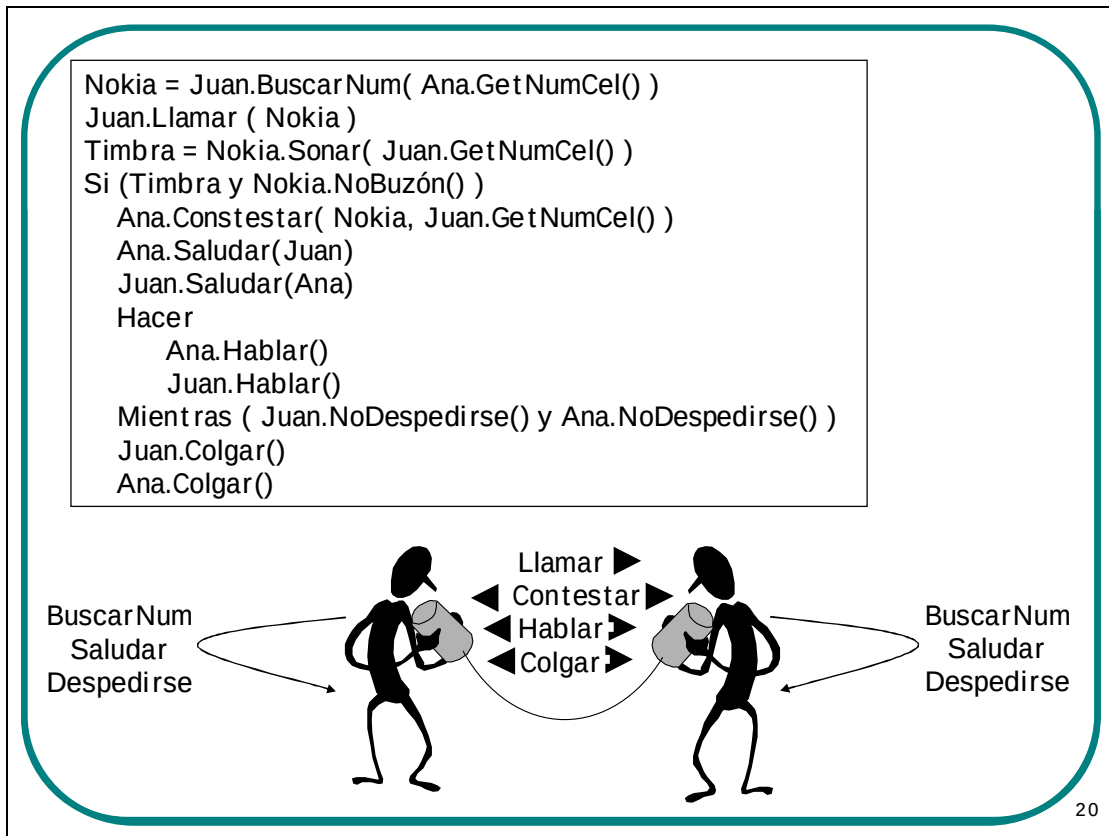


17

## 6. MÉTODOS

- Definen el comportamiento de una clase. Denominados también Funciones Miembro.
- Se implementan dentro de la clase.
- Un método especifica como se ejecuta (acción que hace un objeto) cuando se le envía un mensaje.
- El método es el procedimiento o función que se invoca para actuar sobre un objeto.
- Los métodos determinan como actúan los objetos cuando reciben un mensaje.
- El conjunto de mensajes a los cuales puede responder un objeto se llama "Protocolo del Objeto".

18





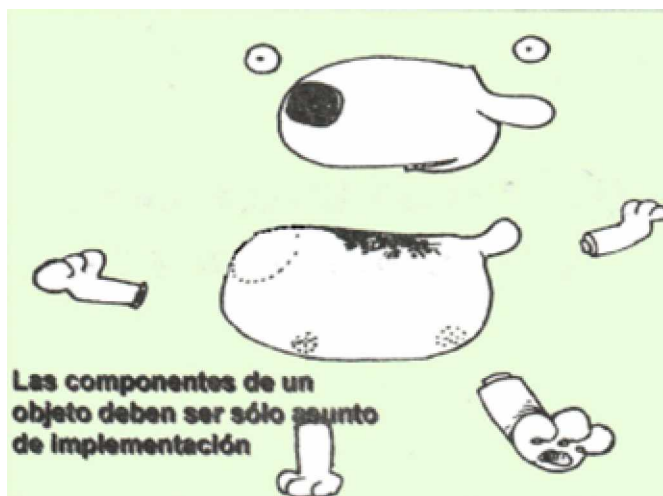
## 7. MODULARIDAD

- Propiedad que permite dividir un programa en partes mas pequeñas e independientes, que reducen en alguna manera su complejidad.
- Cada módulo se puede compilar separadamente, aunque tengan conexiones con otros módulos.
- Facilita la sustitución de un modulo por otro sin afectar su funcionamiento, el seguimiento de errores y el aislamiento de los módulos defectuosos.
- Propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos coherentes y débilmente acoplados.

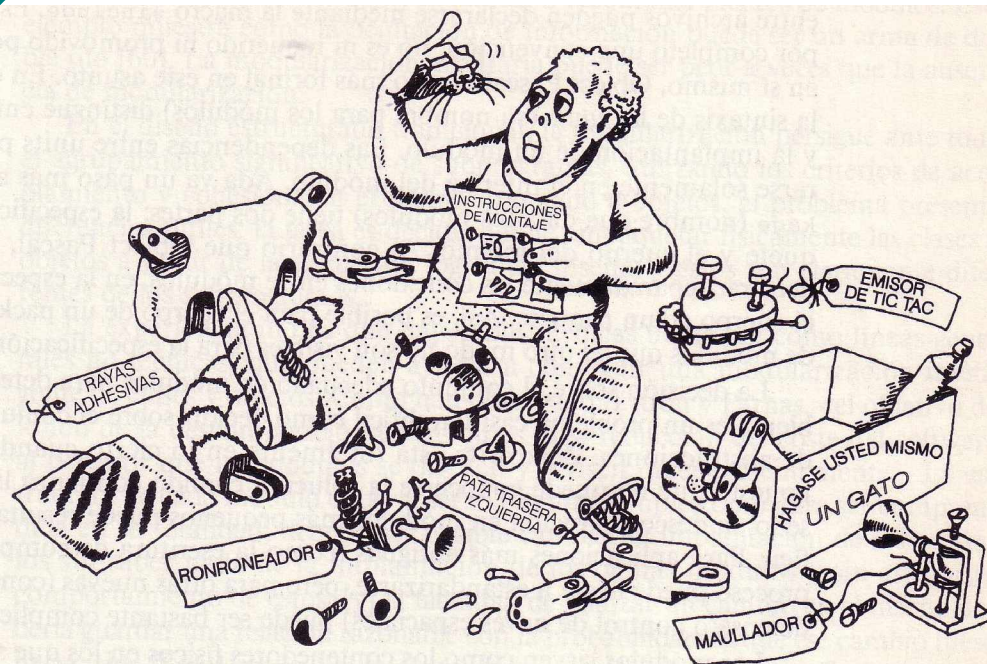
21

En un diseño estructural -> Agrupamiento de subprogramas.

En diseño O.O -> Agrupamiento lógico de clases.



22

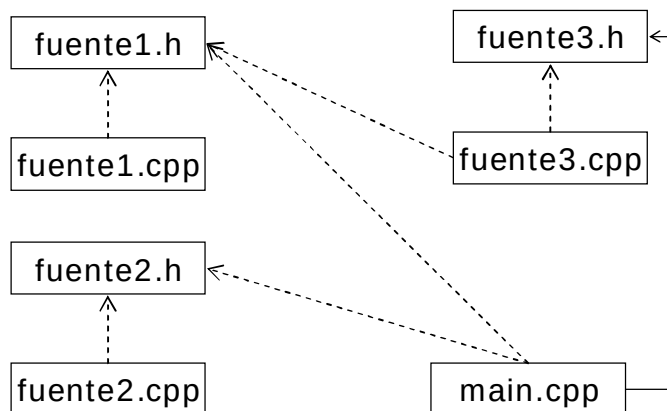


La modularidad empaquetada las abstracciones en unidades discretas.

23

En C son archivos. La dependencia entre ficheros se realiza a través de la macro `#include`.

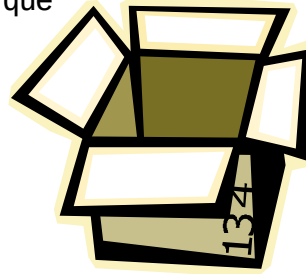
- Módulos Interfase: ficheros con extensión `.h`  
(Header Files = Archivos Cabecera).
- Módulos con implementación: ficheros con extensión `.c` o `.cpp`



24

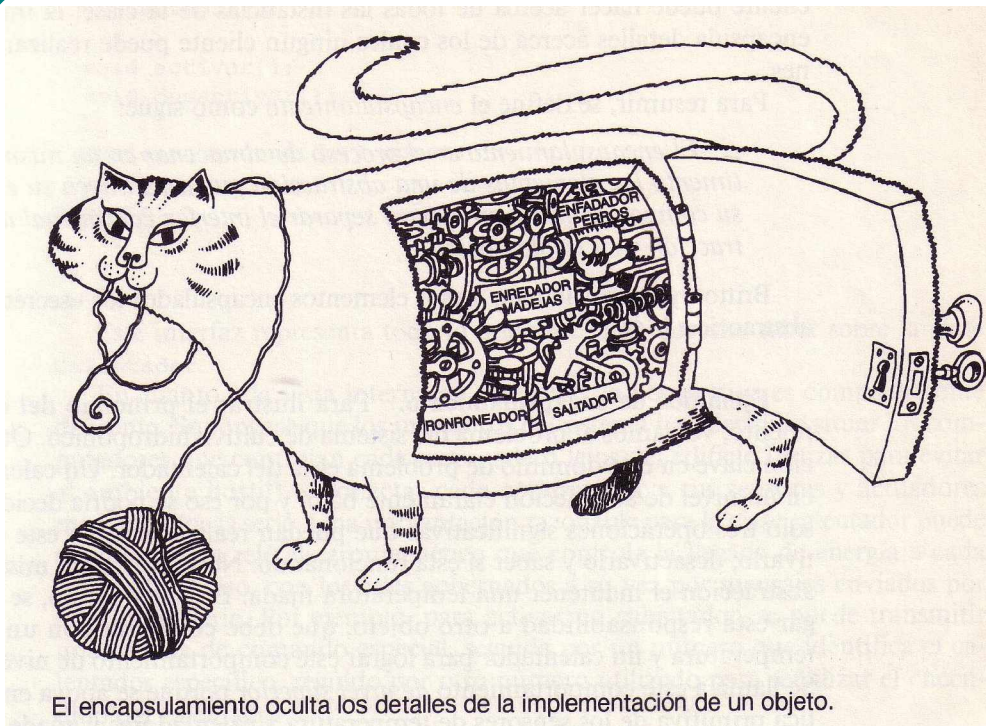
## 8. ENCAPSULAMIENTO

- Separa aquellos atributos del objeto de los que deben ser conocidos por otros y los que pertenecen a su funcionamiento propio.



- Permite asegurar que el contenido de la información de un objeto esta oculto al mundo exterior (es como una Caja Negra).

25



El encapsulamiento oculta los detalles de la implementación de un objeto.

26

## Usos:

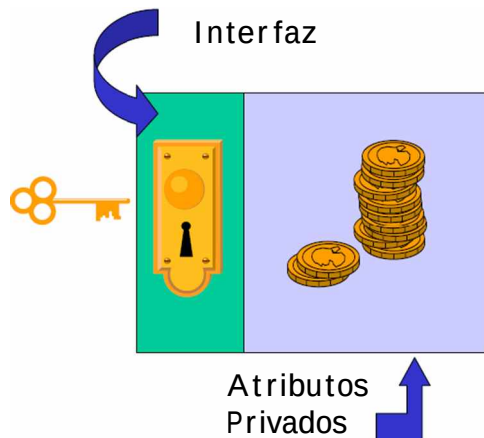
- Las variables temporales NO ocupan memoria y no interfieren en el funcionamiento de otros objetos.
- Es una característica de la modularidad, si el modulo funciona bien no afecta el programa.
- Mantenibilidad: Permite adaptarse al cambio con modificaciones pequeñas.
- Reusabilidad: Una parte de un programa (definición de clases e implementación) puede ser usado en otro programa sin tener que realizarle modificaciones.



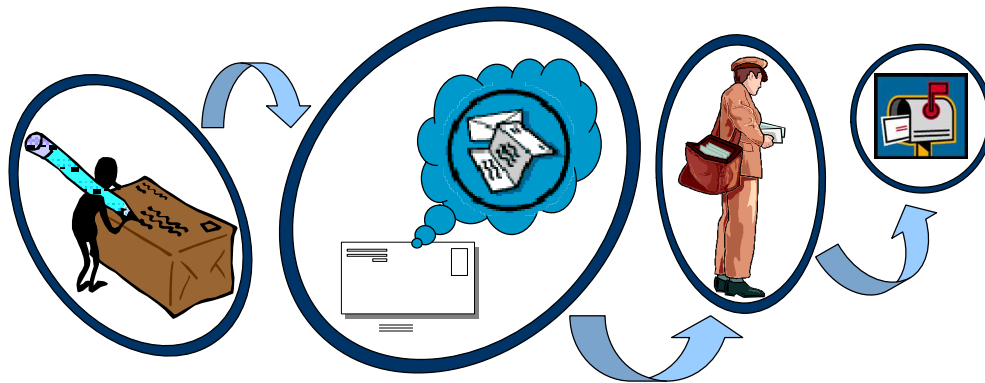
27

La encapsulación da lugar a que las clases se dividan en dos partes:

- Interface (Externa-Publica): Lo que necesitan los otros objetos para interactuar con el objeto que definimos. (Visión externa de una clase).
- Implementación (Interna-Privada): Información que necesita el objeto para funcionar (Mecanismos que conducen al comportamiento deseado).



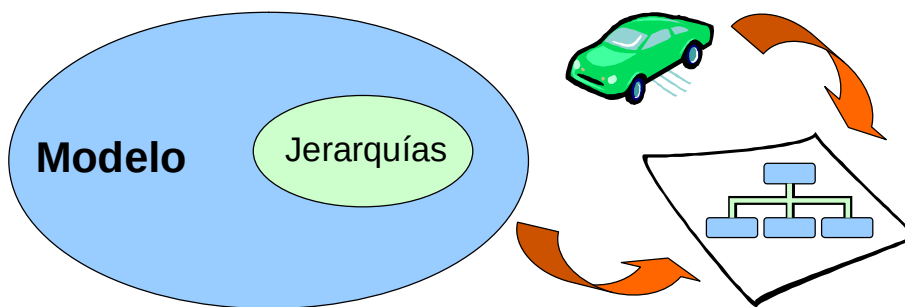
28



29

## 9. JERARQUIAS

- Es una clasificación u ordenación de las abstracciones.
- Describe el orden de relación que se produce entre abstracciones diferentes.
- Son utilizadas para representar simplificadaamente un problema del mundo real, imaginario o hipotético.



30



## TIPOS DE JERARQUÍAS:

### Estructuras de Objetos:

Asociaciones ➡ "A utiliza a B"

Agregación ➡ "A es Parte de B", "A Tiene un B"

### Estructuras de Clase (Generalización, Especialización):

Herencia ➡ "A Es un B"

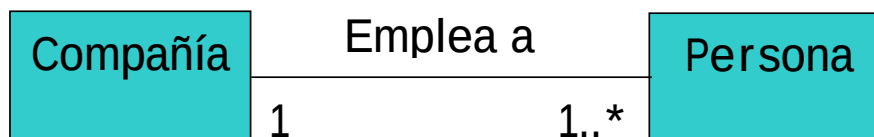
Objetos construidos a partir de otros.

Tipos: Sencilla y Múltiple (Selectiva y Múltiple)

31

## 9.1. Estructuras de Objetos: ASOCIACIONES

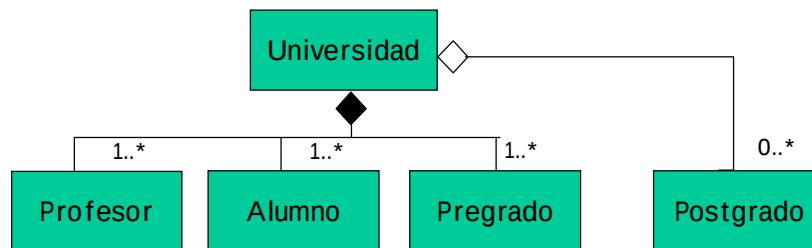
- Es la conexión física o conceptual entre dos o más objetos.
- Esta conexión puede ser binarias o N-arias.
- Aparecen como verbos en la descripción del problema.
- Se indican las restricciones de cardinalidad o Multiplicidad (Número de instancias de una clase; cuantos objetos de esa clase pueden participar en la relación dada).



32

## 9.2. Estructuras de Objetos: AGREGACIÓN

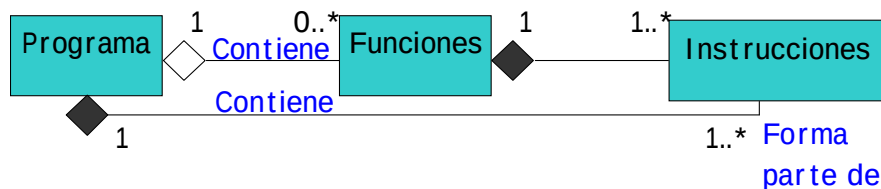
- Una o más clases (Clases Componentes) forman parte o son componentes de otra clase (Clase Agregada).
- Una clase agregada puede potencialmente existir sin sus partes constituyentes.
- En cualquier momento dado, cualquier objeto constituyente puede formar parte de más de un objeto agregado.



33

• Dependencia existencial o Fuerte (Composición): Un objeto componente depende del objeto agregado del cuál forma parte. El elemento dependiente desaparece al destruirse el que lo contiene y, si es de cardinalidad 1, es creado al mismo tiempo. (Rombo Lleno)

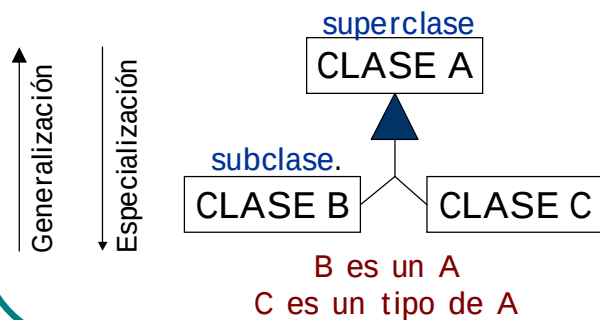
• Independencia existencial o Dependencia Débil: se trata de una relación entre las dos clases no muy fuerte. El objeto componente no desaparece al destruirse el que lo contiene. (Rombo Vacío)



34

## 9.3. Estructuras de Objetos: GENERALIDAD - Herencia

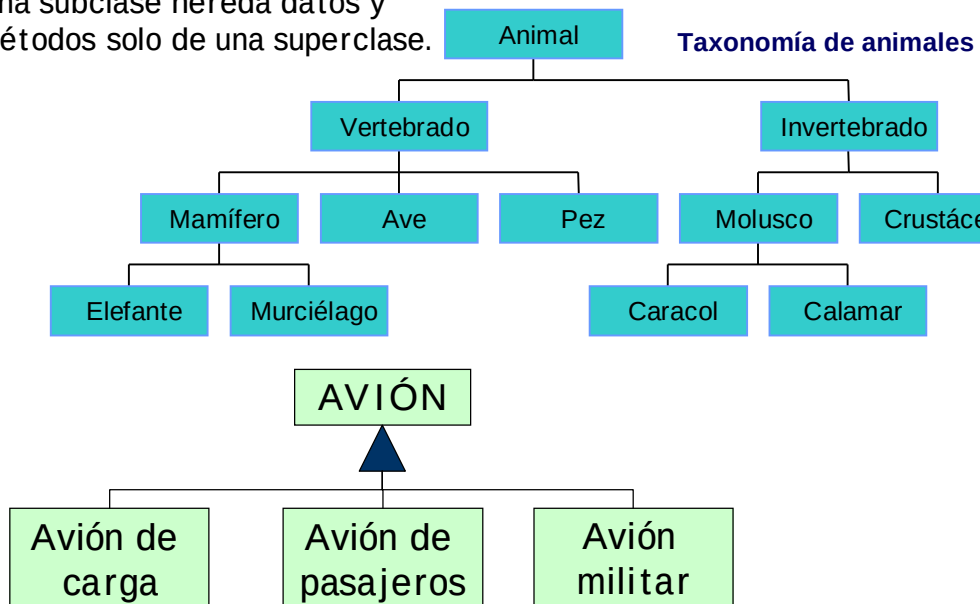
- Una clase comparte la estructura definida en otra clase.
- Característica que permite que un objeto sea construido a partir de otros.
- Las Subclases heredan las características de la Superclase (Padre-Hijo, Base-Derivada).
- Ventaja principal: Reutilizar código existente



35

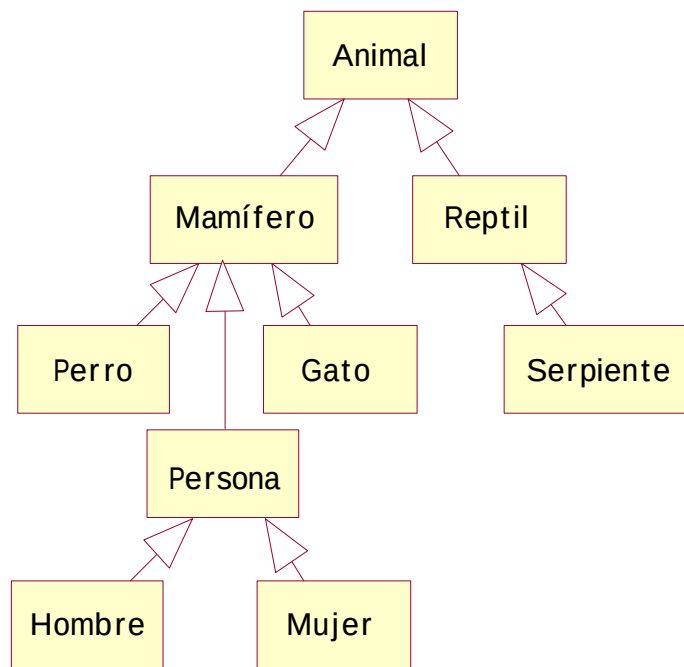
### HERENCIA SIMPLE:

Una subclase hereda datos y métodos solo de una superclase.



36

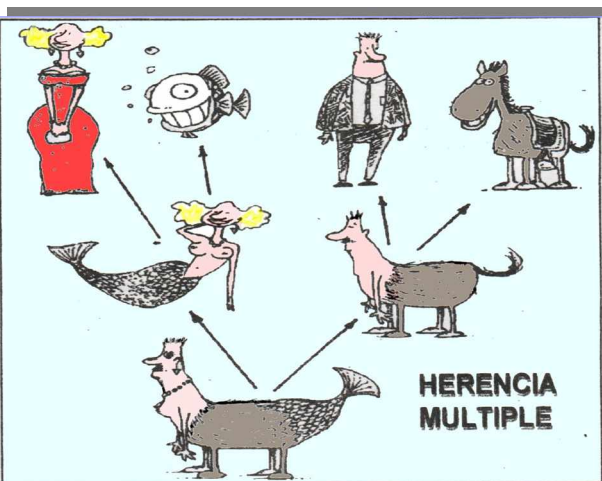
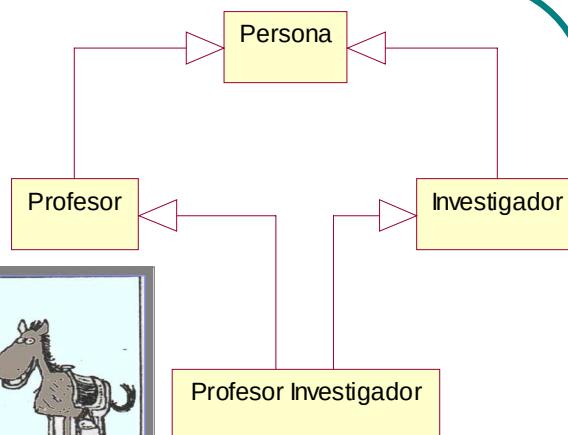




37

### HERENCIA MULTIPLE:

Una subclase puede heredar datos y métodos de mas de una clase.



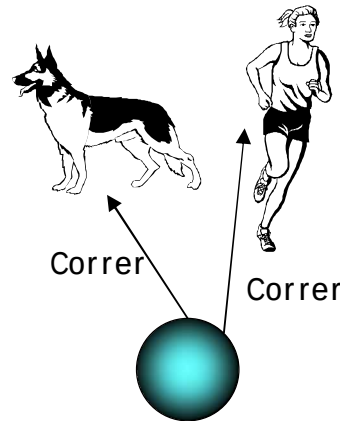
### Desventajas:

Al hereda todo pueden existir conflictos de nombre, valores y restricciones.

38

# 10. POLIMORFISMO

- Es la característica de la POO que permite que una misma acción se comporte de diferentes formas según sea la clase sobre la que se aplica. Dicho de otra manera, puede haber implementaciones diferentes (con la misma semántica) para una misma operación.
- Es el uso de un mismo nombre o símbolo para representar más de una acción
- “Un solo mensaje puede producir respuestas diferentes dependiendo del objeto que lo reciba. La interpretación depende del receptor”



39

## POLIMORFISMO: Sobrecarga

- Es el uso de o de un mismo nombre (Sobrecarga de funciones) o símbolo (Sobrecarga de Operadores) para representar más de una acción.
- Describe la característica de utilizar el mismo nombre de operación y/o el mismo operador para representar operaciones similares que se comportan de manera diferente cuando se aplican a clases diferentes.
  - Usándola como se hace con nuestro lenguaje natural, como por ejemplo las expresiones: tomar agua, tomar del pelo, tomar medidas, tomar por hecho, tomar el poder, ...
  - O usándola en métodos con el mismo nombre y diferente signatura (tipos y/o número de parámetros de entrada), por ejemplo: `int sumar(Obj1, Obj2)`  $\Rightarrow$  **Obj1 + Obj2**

40



# Universidad **Nebrija**

## **Escuela Politécnica Superior de Ingeniería Departamento de Ingeniería Industrial**

**Fundamentos de la informática**

### **4. Programación orientada a objetos con Java**

# Contenido

---

- n** Introducción
- n** La programación orientada a objetos
- n** Programación Java
- n** Introspección
- n** Utilidades de Java



# Introducción

---

## Análisis y diseño orientado a objetos

- n** La programación orientada a objetos es una técnica de análisis y diseño que se enfoca en los elementos de un sistema, sus atributos y responsabilidades
- n** El modelo abstracto está formado de clases. Una clase describe a un conjunto de objetos que comparte los mismos atributos, comportamiento y semántica
- n** Un objeto es una instancia de una clase

# Programación orientada a objetos

---

## Análisis y diseño orientado a objetos

- n Las clases representan un esquema simplificado de la casuística de un problema determinado**
- n Para diseñar un sistema orientado a objetos es necesario responder las siguientes preguntas**
  - n ¿Cuáles son los elementos tangibles de un sistema?**
  - n ¿Cuáles son sus atributos?**
  - n ¿Cuáles son sus responsabilidades?**
  - n ¿Cómo se relacionan los elementos del sistema?**
  - n ¿Qué objeto debe “saber”...?**
  - n ¿Qué objeto debe “hacer”...?**

# Programación orientada a objetos

---

## Conceptos básicos

**n Encapsulación**

**n Herencia**

**n Polimorfismo**

# Programación orientada a objetos

---

## Encapsulación

- n La encapsulación consiste en formar un “paquete” con los atributos (variables) y el comportamiento (métodos) de un objeto**
- n Los métodos forman la membrana exterior de un objeto y “esconden” los detalles de implementación al usuario**

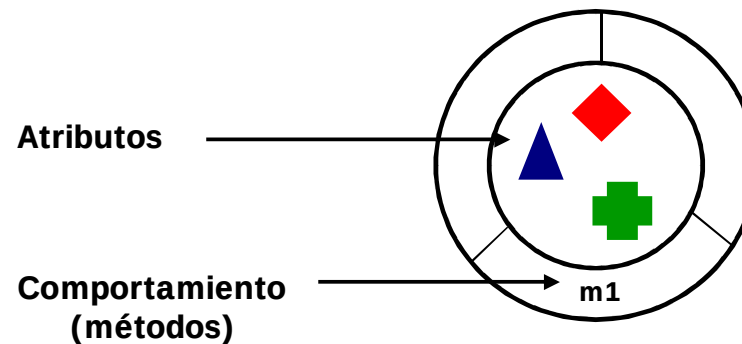


# Programación orientada a objetos

---

## Encapsulación

- n La encapsulación hace que un sistema sea más fácil de comprender y facilita el mantenimiento de una aplicación**

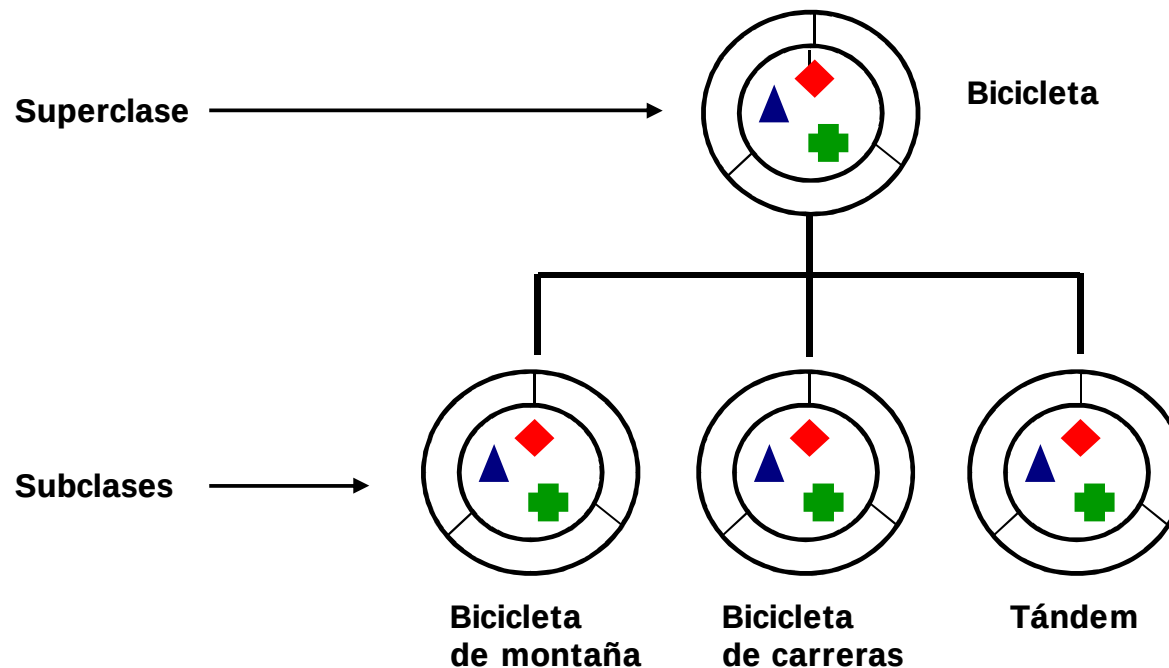


# Programación orientada a objetos

---

## Herencia

- n** La herencia es la capacidad de una clase para definirse en términos de otra clase y “heredar” atributos y responsabilidades de la clase de orden superior



# Programación orientada a objetos

---

## Polimorfismo

- n El polimorfismo permite que distintos objetos pertenecientes a una misma clase “respondan” de diferentes formas a un mismo mensaje**
- n El polimorfismo permite modificar el comportamiento de un método en cada subclase. En este ejemplo, la superclase mascota tiene las subclases gato, pato y perro. El método “saludar” cada subclase es diferente**



En este ejemplo cada tipo de mascota “saluda” de forma distinta

# Programación orientada a objetos

---

## Encapsulación, herencia y polimorfismo

- n La reutilización de código es una de las grandes ventajas de la programación orientada a objetos**
- n Reduce el tiempo de desarrollo de aplicaciones e incrementa la productividad de los ingenieros de software**
  - n Reutilización de clases**
  - n Diseño de una nueva clase a partir de otra (herencia)**

# Programación orientada a objetos

---

## Diagramas de clases

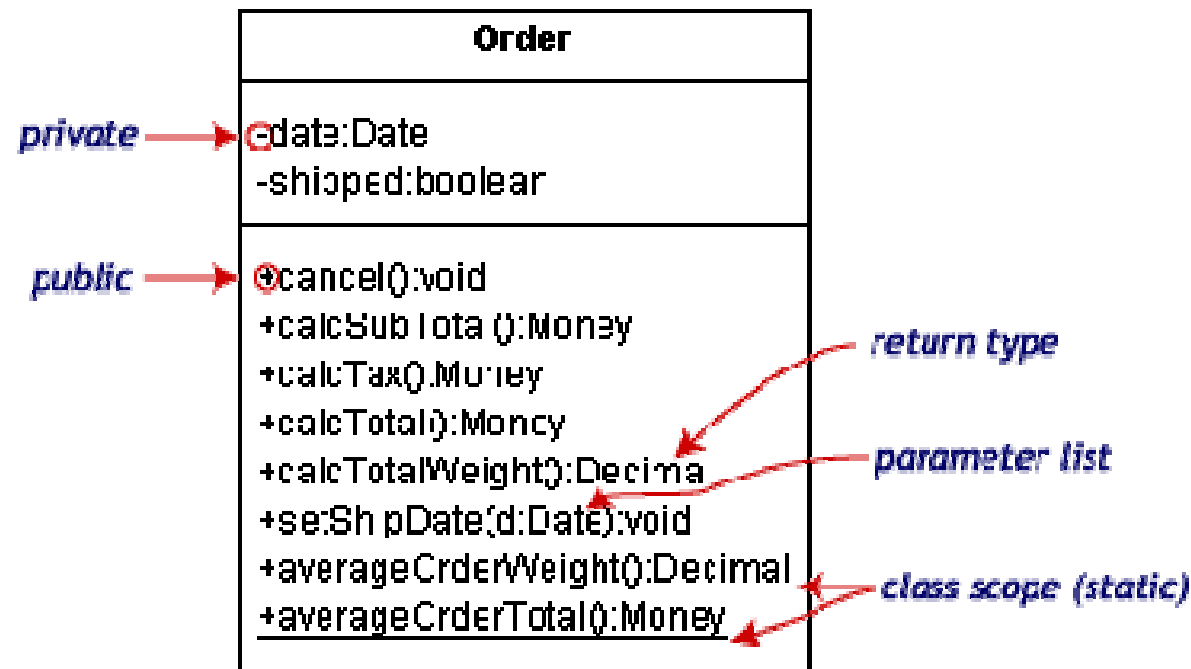
- n En UML una clase se representa como un rectángulo dividido en tres partes: el nombre de la clase, sus atributos y métodos. Las clases abstractas se identifican por la letra cursiva**
- n Los métodos de una clase se especifican indicando el tipo de acceso (+, -, #), el nombre, la lista de parámetros y el tipo que devuelve**
  - n + Acceso público**
  - n - Acceso privado**
  - n # Acceso protegido**

# Programación orientada a objetos

---

## Diagramas de clases

- n** El diagrama de una clase muestra el nombre de la clase, sus atributos y métodos



# Programación orientada a objetos

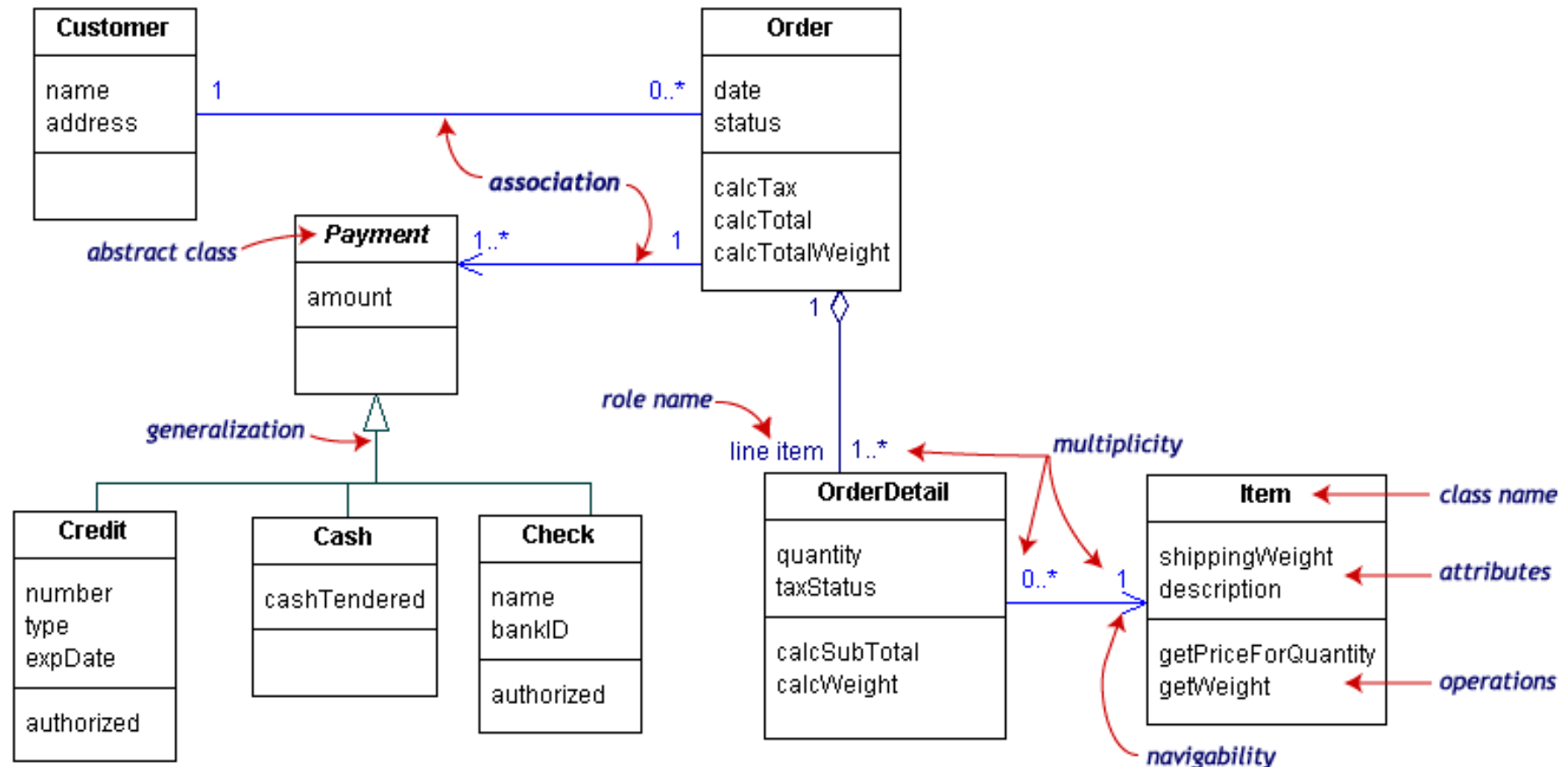
---

## Diagramas de clases

- n El diagrama de clases ofrece una vista general de un sistema, muestra las clases y las relaciones existentes entre ellas**
- n El diagrama de clases es estático, muestra las interacciones entre clases pero no lo que sucede como resultado de esa interacción**
- n Cada diagrama se compone de clases y asociaciones. Cada asociación afecta a un número determinado de instancias de una clase, denominado “multiplicidad”**

# Programación orientada a objetos

## Diagramas de clases





# Programación orientada a objetos

---

## Relaciones entre clases

- n Asociación.** Relaciona instancias de dos clases. Existe una asociación entre dos clases cuando una instancia de una clase debe saber sobre otra instancia para llevar a cabo sus funciones
- n Agregación.** Es un tipo de asociación donde una clase pertenece a una colección. Se utiliza un diamante en el extremo de la asociación de la clase que representa el todo
- n Generalización.** Indica una relación de herencia entre dos clases. Se utiliza un triángulo en el extremo de la relación para señalar a la superclase

# Programación orientada a objetos

---

## Asociaciones entre clases

- n Rol.** Se utiliza para aclarar la naturaleza de la asociación
- n Navegabilidad.** La flecha de la asociación indica la dirección de navegación
- n Multiplicidad.** Determina el número posible de instancias

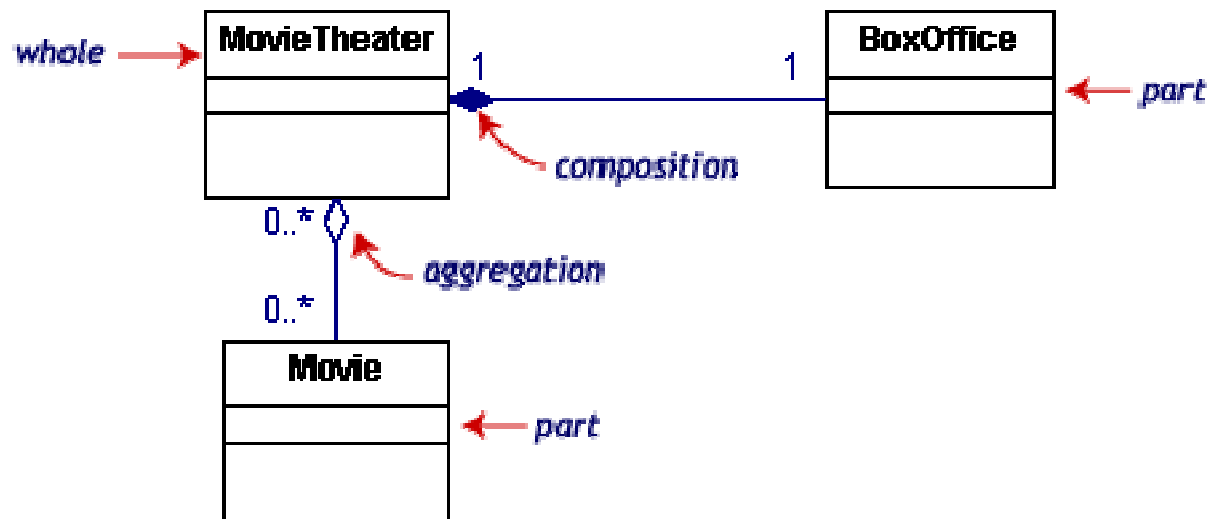
| Multiplicidad | Significado                                                             |
|---------------|-------------------------------------------------------------------------|
| 0..1          | Cero o una instancia. La notación $n..m$ indica de $n$ a $m$ instancias |
| 0..*          | No existe límite de instancias, desde cero hasta un número ilimitado    |
| 1             | Exactamente una instancia                                               |
| 1..*          | Al menos una instancia                                                  |

# Programación orientada a objetos

---

## Asociaciones entre clases

- n Agregación:** Asociación que relaciona a un objeto que es parte de otro
- n Composición:** Agregación fuerte, la parte no puede existir sin el todo



# Programación orientada a objetos

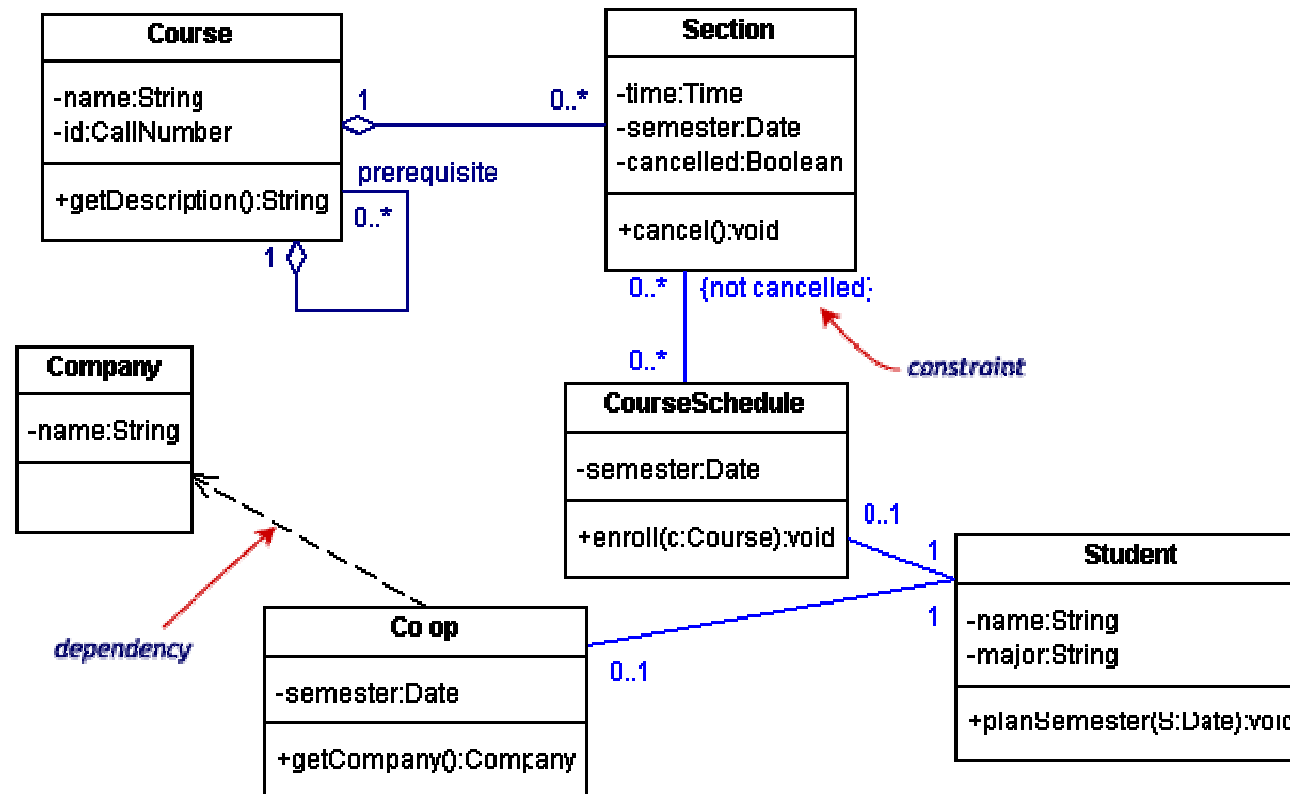
---

## Dependencias y limitaciones

- n** Una dependencia es una relación entre dos clases. Si se produce un cambio en una de ellas puede producir cambios en la otra. Las dependencias se representan con líneas de puntos
- n** Una limitación es una condición que se aplica al diseño. Las limitaciones se expresan utilizando los símbolos { }

# Programación orientada a objetos

## Dependencias y limitaciones

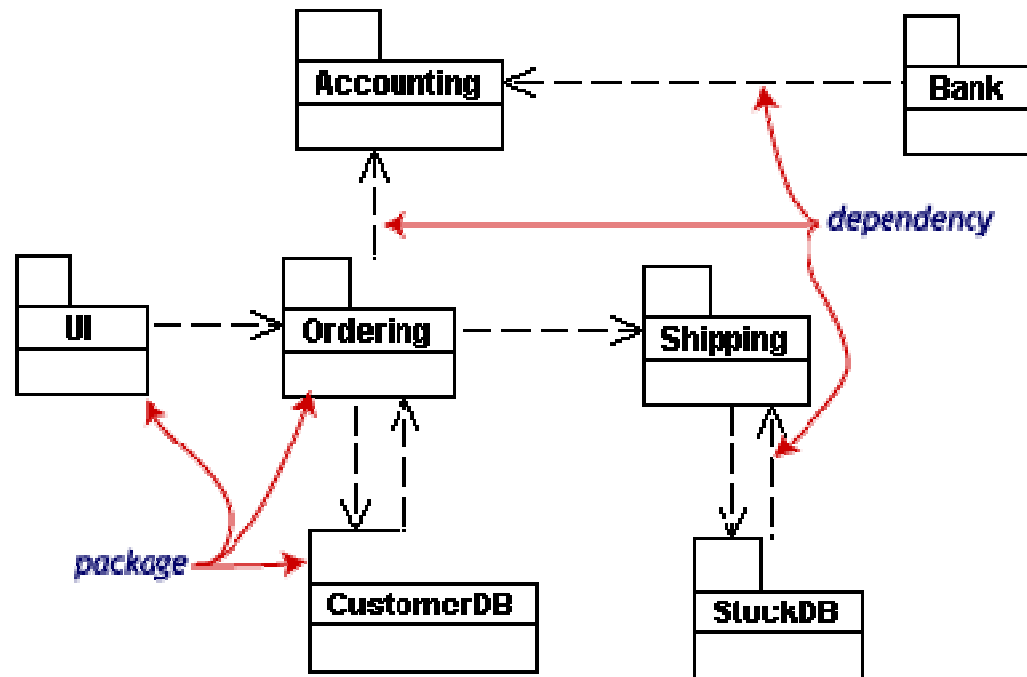


# Programación orientada a objetos

---

## Diagramas de clases

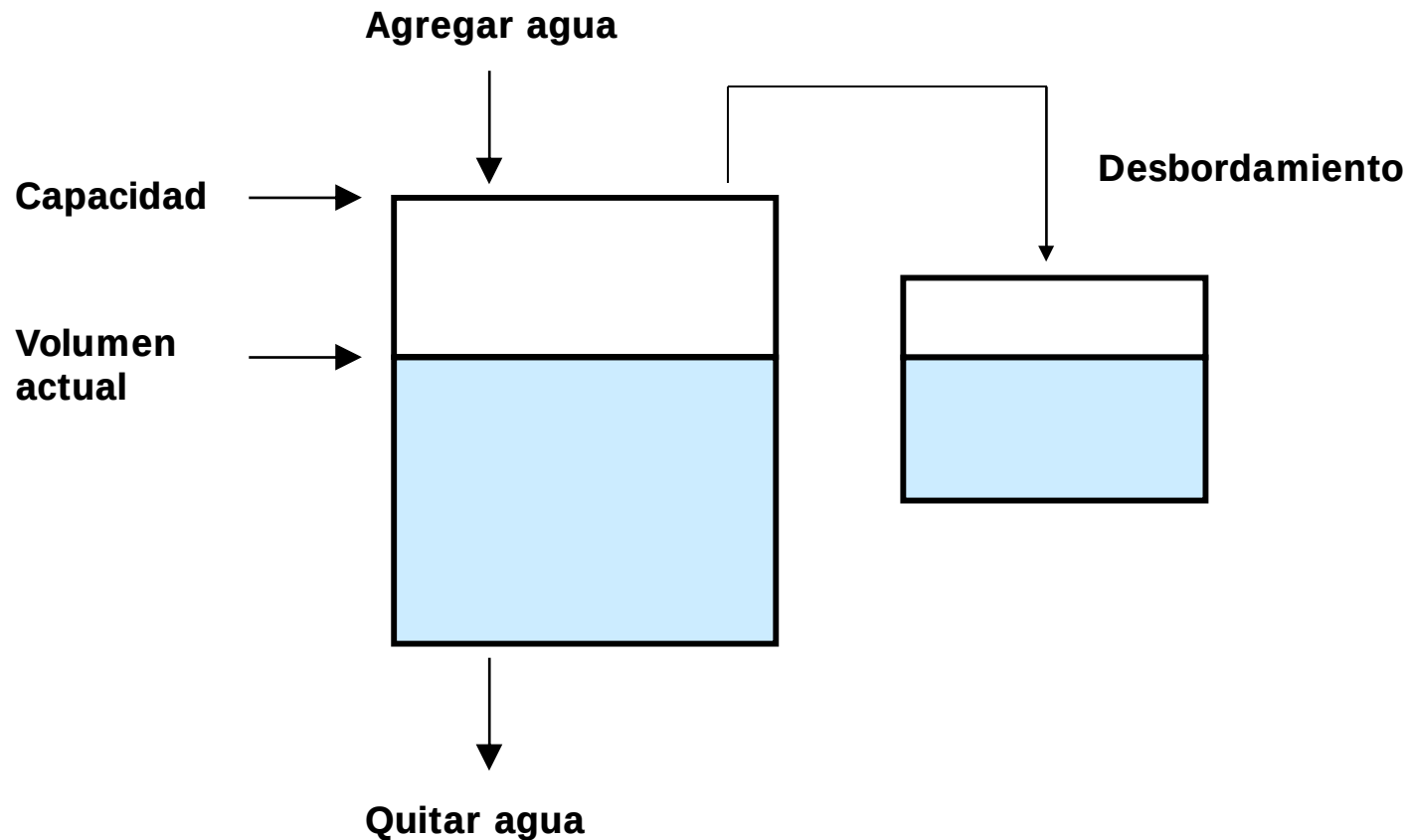
- n** Las clases se agrupan en paquetes. El uso de paquetes permite organizar diagramas de clases complejos



# Programación orientada a objetos

---

¿Cuál es el modelo de clases para un depósito de agua?



# Programación orientada a objetos

---

**¿Cuál es el modelo de clases para un depósito de agua?**

**El modelo de clases está compuesto por:**

- n Elementos tangibles (objetos)**
- n Atributos (campos)**
- n Responsabilidades (operaciones o métodos)**
- n Limitaciones de diseño (contrato del objeto)**



# Programación orientada a objetos

---

¿Cuál es el modelo de clases para un depósito de agua?

- n Elementos tangibles (objetos)**

  - n Depósito de agua**

- n Atributos (campos)**

  - n Capacidad ( $\text{m}^3$ )**

  - n Volumen actual ( $\text{m}^3$ )**

# Programación orientada a objetos

---

¿Cuál es el modelo de clases para un depósito de agua?

**n Responsabilidades (operaciones o métodos)**

**n Agregar agua ( $\text{m}^3$ )**

**n Quitar agua ( $\text{m}^3$ )**

# Programación orientada a objetos

---

¿Cuál es el modelo de clases para un depósito de agua?

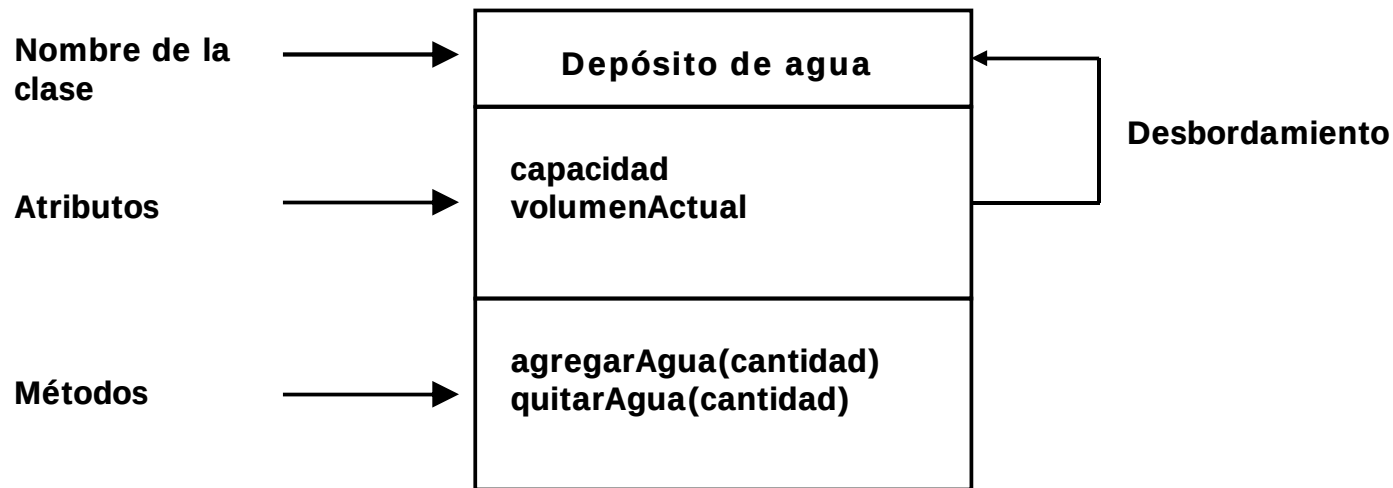
## **n** Limitaciones de diseño (contrato del objeto)

- n** La capacidad del depósito debe ser mayor que cero
- n** La capacidad es constante y se expresa en m<sup>3</sup>
- n** No se puede quitar agua de un depósito vacío
- n** No se puede añadir agua si no existe un depósito para recoger el agua en caso de desbordamiento
- n** Si al añadir agua hay desbordamiento, el agua que se desborda se debe almacenar en un depósito auxiliar
- n** No se puede agregar y quitar agua al mismo tiempo

# Programación orientada a objetos

---

¿Cuál es el modelo de clases para un depósito de agua?

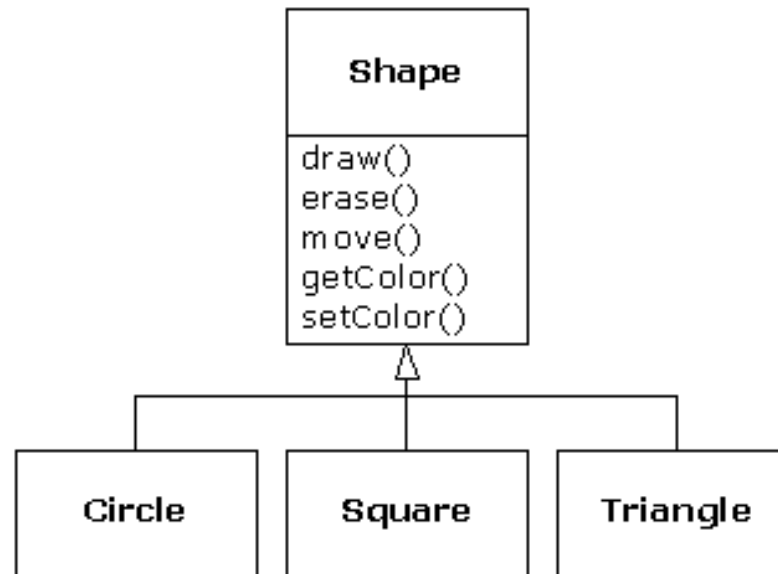


# Programación orientada a objetos

---

## Clases y objetos

- n** Todos los objetos de una clase comparten las mismas características

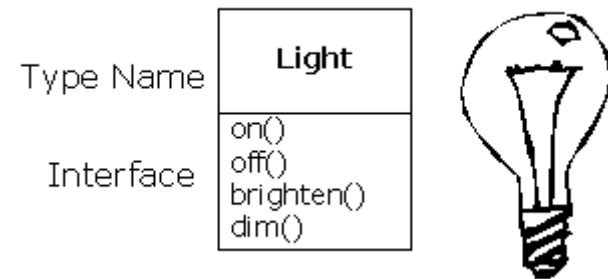


# Programación orientada a objetos

---

## Clases y objetos

- n** A partir de una clase se puede obtener un objeto (instancia) que se comunica con otros objetos mediante mensajes
- n** Cada objeto tiene un estado propio
- n** Cada objeto responde a un conjunto de peticiones definidas en la interfaz de la clase a la que pertenece



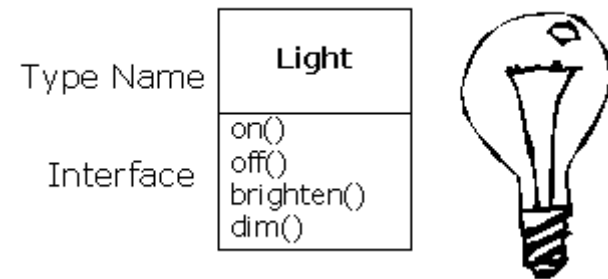
# Programación orientada a objetos

---

## Clases y objetos

- n** La interfaz ofrece al exterior el comportamiento del objeto. El código y los datos internos que se utilizan para responder a una petición se denomina implementación
- n** Un objeto se compone de:
  - n** Interfaz: qué se ofrece
  - n** Implementación: cómo se ofrece

```
Light bombilla = new Light();  
bombilla.on();
```



# Programación orientada a objetos

---

## Encapsulación

- n La encapsulación permite ocultar los datos y la funcionalidad de un objeto. Además, facilita la reutilización de objetos**
- n La interfaz de la clase de un objeto permite que otros objetos accedan a los atributos y a los métodos públicos de un objeto**

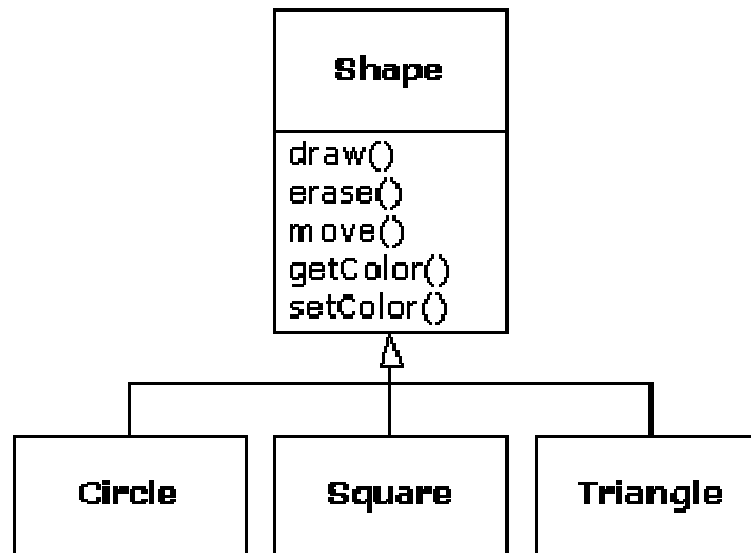


# Programación orientada a objetos

---

## Clases y subclasses

- n** La herencia permite diseñar clases a partir de otras clases. Una subclase puede añadir o modificar la funcionalidad de la clase de orden superior

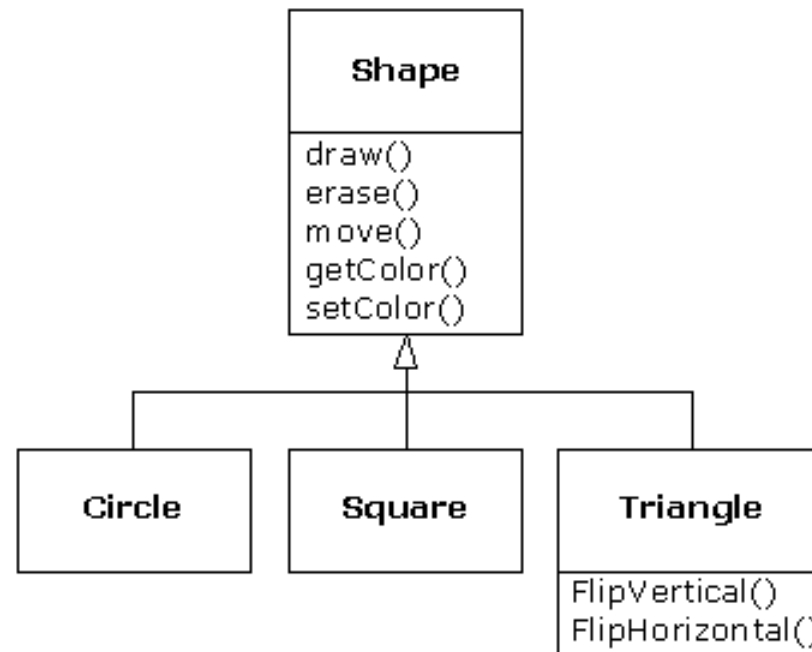


# Programación orientada a objetos

---

## Clases y subclases

- n** Una subclase puede ampliar o extender el comportamiento de la clase de orden superior

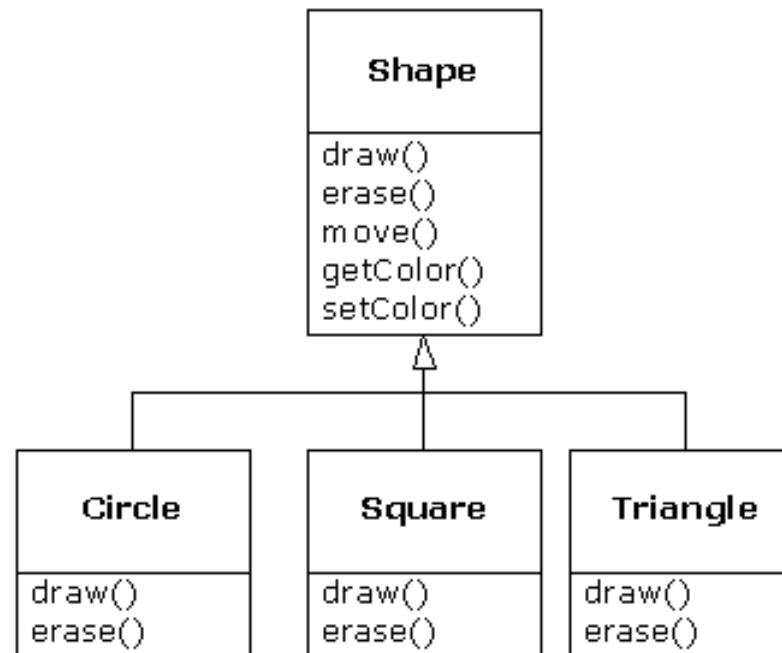


# Programación orientada a objetos

---

## Clases y subclases

- n Una subclase puede modificar el comportamiento de la clase de orden superior**

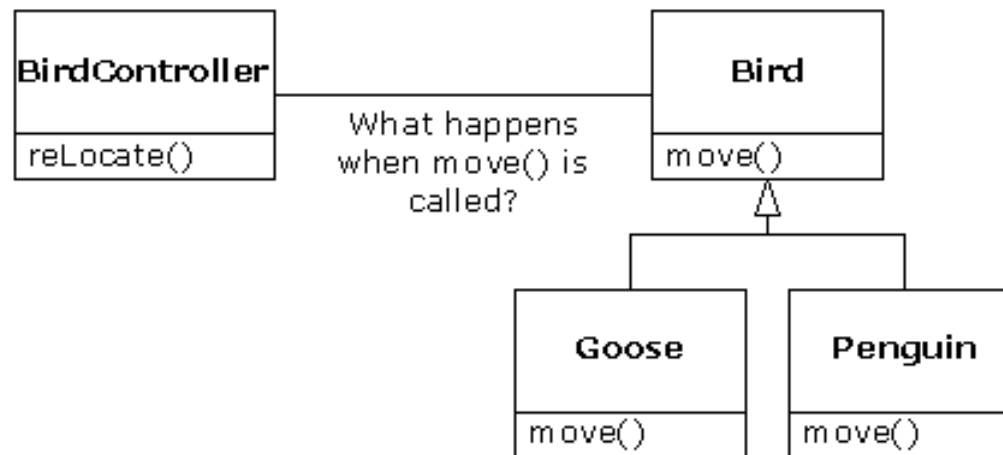


# Programación orientada a objetos

---

## Polimorfismo y sobrescritura de métodos

- n** El polimorfismo es la capacidad de modificar el comportamiento de una clase mediante la sobrescritura de métodos



# Programación orientada a objetos

---

**¿Cómo identificar las clases y los métodos de un sistema?**

- n Identificar las clases del sistema. Normalmente, las clases corresponden con los sustantivos que se utilizan para describir el sistema**
- n Definir los atributos de las clases. En general, los atributos coinciden con los adjetivos que se emplean para describir las clases**
- n Definir los métodos de cada clase. Normalmente los verbos o las acciones que describen el sistema corresponderán con los métodos**

# Programación Java

---

**Java es un lenguaje orientado a objetos**

- n En Java, todo es un objeto. Los objetos se definen en términos de la clase a la que pertenecen**

**String nombre;**

**Persona alumno;**

- n El estado de un objeto se modifica utilizando los métodos de la clase**

# Programación Java

---

## Clases y objetos

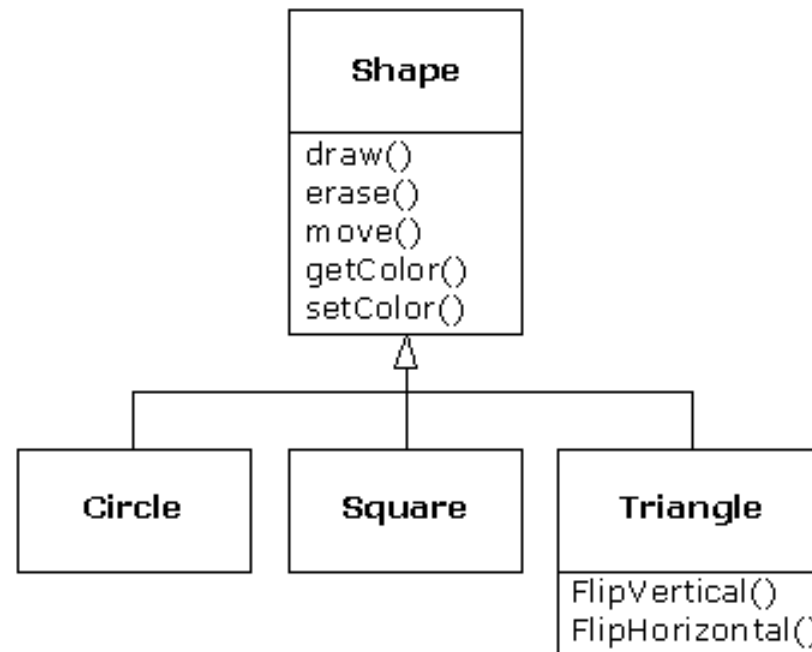
- n Las clases representan objetos de la realidad (dominio) al que se refiere un programa Java**
- n Las clases se definen a partir de un conjunto de atributos y métodos. Cada atributo puede estar limitado a un dominio de valores**
- n Las clases son conceptos abstractos. Los objetos son instancias de las clases, se crean durante la ejecución de los programas**

# Programación Java

---

## Clases y métodos

- n Los métodos de las clases realizan acciones que modifican las propiedades de los objetos**





# Programación Java

---

## El uso de métodos

- n Los métodos se utilizan para estructurar un programa Java de forma lógica**
- n El uso de métodos permite dividir un problema en problemas más pequeños que son más fáciles de resolver (“Divide y vencerás”)**
- n Los métodos encapsulan un conjunto de instrucciones que se puede ejecutar tantas veces como sea necesario. El código del método queda oculto y sólo es necesario conocer su interfaz (parámetros y valor de retorno)**

# Programación Java

---

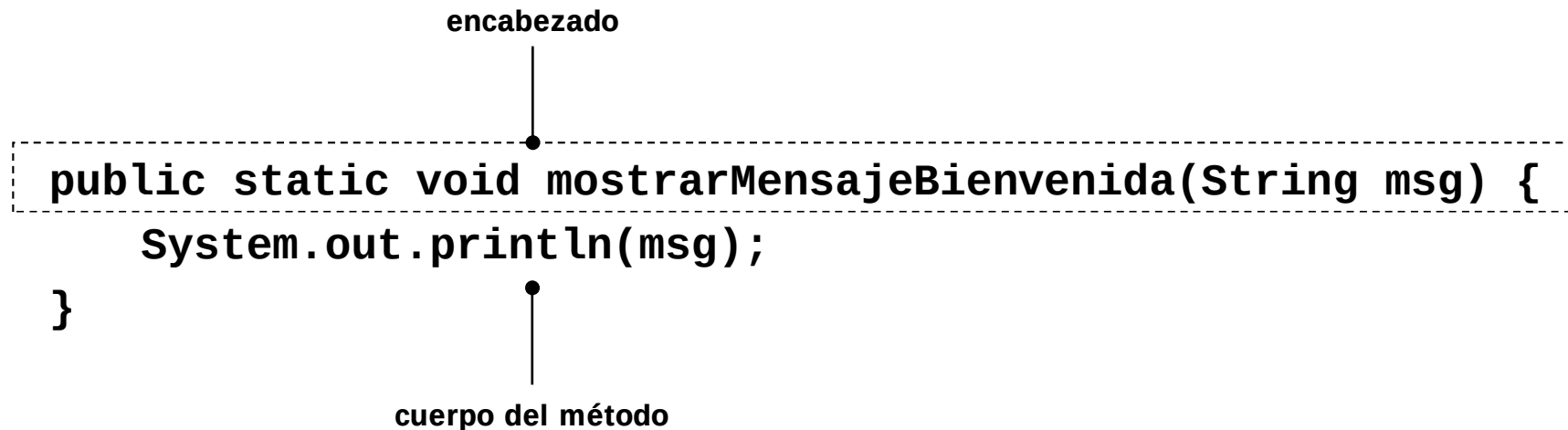
## El uso de métodos

- n Un método lleva a cabo una tarea específica de diversa naturaleza: input, output o cálculo**
- n Para crear un método se tiene que escribir la declaración del método, el encabezado y el cuerpo. Dentro del cuerpo pueden haber declaraciones de variables y constantes. Si las hay se conocen como variables locales y solo pueden ser utilizadas dentro del método**
- n En otros lenguajes, los métodos se denominan subrutinas o funciones**

# Programación Java

---

## Declaración de un método: encabezado y cuerpo



# Programación Java

---

## El encabezado de un método

The diagram illustrates the components of a Java method header. Four labels are positioned above the code: 'modificadores' points to 'public static', 'valor de retorno' points to 'void', 'nombre del método' points to 'mostrarMensajeBienvenida', and 'lista de parámetros' points to '(String msg)'. The code is as follows:

```
public static void mostrarMensajeBienvenida(String msg) {  
    System.out.println(msg);  
}
```

# Programación Java

---

## El encabezado de un método

### **n** Modificadores

|                  |                                                |
|------------------|------------------------------------------------|
| <b>public</b>    | el método puede invocarse fuera de la clase    |
| <b>protected</b> | el método puede invocarse fuera de la clase    |
| <b>private</b>   | no se puede invocar fuera de la clase          |
| <b>static</b>    | el método pertenece a la clase, no a un objeto |

### **n** Tipo del valor de retorno (“Return type”)

**int | long | double | String** o cualquier otro tipo de dato para métodos que devuelven valores, **void** si el método no devuelve un valor

# Programación Java

---

## El encabezado de un método

### **n** Nombre del método

**Debe ser un nombre descriptivo de la tarea que realiza el método**

### **n** Lista de parámetros

**La lista de parámetros es opcional. Contiene la declaración de parámetros (variables) que permiten al método recibir datos del exterior**

# Programación Java

---

## Ejecución de métodos

- n Un método se ejecuta cuando es invocado o llamado**
- n El método main se ejecuta automáticamente al comenzar la ejecución de un programa**
- n Para ejecutar un método basta indicarel nombre del método y la lista de parámetros**

**mostrarMensajeBienvenida(“Hola mundo”);**

# Programación Java

---

## Ejecución de métodos

- n Los valores que se pasan a un método cuando es invocado se denominan argumentos**
- n Para invocar un método que tiene en su lista de parámetros un valor entero se debe pasar una constante entera, una variable entera o una expresión de tipo entero**



# Programación Java

---

## Ejecución de métodos

- n Los métodos pueden ser de dos tipos: métodos sin valor de retorno o métodos con valor de retorno**
- n Los métodos sin valor de retorno se declaran con el valor de retorno void. El método main devuelve void**
- n Los métodos con valor de retorno se declaran con el tipo de dato correspondiente: int, char, long, double, String o cualquier otro tipo. Estos métodos utilizan la instrucción return para finalizar la ejecución del método, devolver el valor y el control del programa al punto donde se realizó la llamada**

# Programación Java

---

## Ejecución de métodos

```
public static int suma (int num1, int num2){  
    return (num1 + num2);  
}
```

**El valor de retorno se almacena en la variable entera c:**

```
int a = 5, b = 10, c;
```

```
c = suma(5, 10);
```

```
c= suma(a, b);
```

# Programación Java

---

## Parámetros y argumentos

- n** El tipo del argumento que se pasa a un método al invocarlo tiene que corresponder con el tipo del parámetro en la declaración.
- n** La correspondencia entre argumentos y parámetros es uno a uno según el orden
- n** En el ejemplo anterior el 5 corresponde con num1 y el 10 con num2. Las constantes 5 y 10 son argumentos, mientras que num1 y num2 son parámetros. Los argumentos se usan en tiempo de ejecución y los parámetros en la definición del método

# Programación Java

---

## Parámetros, argumentos y valor de retorno

```
public static void main() {  
    int x, y;  
  
    Scanner teclado = new Scanner(System.in);  
  
    System.out.print("Introduzca el valor de x: ");  
    x = teclado.nextInt();  
  
    System.out.print("Introduzca el valor de y: ");  
    y = teclado.nextInt();  
  
    System.out.print("La suma es: " + suma(x, y));  
}
```

# Programación Java

---

## Parámetros, argumentos y valor de retorno

```
public static int leerNumero() {  
    Scanner teclado = new Scanner(System.in);  
  
    System.out.println("Introduzca un numero: ");  
  
    int x = teclado.nextInt();  
  
    return x;  
}
```

# Programación Java

---

## Parámetros, argumentos y valor de retorno

- n Los argumentos que se pasan a un método deben ser compatibles con la el tipo de dato del parámetro**
- n El tipo de dato de la variable donde se recibe el valor de retorno de un método debe corresponder con el tipo de dato de retorno del método**
- n Java hace conversiones automáticas si el argumento es menor en tamaño que el parámetro (widening) pero no al revés (narrowing)**

# Programación Java

---

## Parámetros, argumentos y valor de retorno

```
public static void main() {  
    int x, y;  
  
    x = leerNumero();  
    y = leerNumero();  
  
    System.out.println("La suma es: " + suma(x, y));  
}
```

# Programación Java

---

## El control de errores de ejecución en los métodos

- n** Java requiere que todo método que se relacione con una entidad externa como un archivo de texto, controle los errores que se produzcan en tiempo de ejecución
- n** Java denomina excepciones a los errores en tiempo de ejecución
- n** El control de las excepciones de un programa Java se realiza con la instrucción `throws IOException`



# Programación Java

---

## El control de errores en tiempo de ejecución

- n** Una excepción es error que se produce en tiempo de ejecución de un programa Java
- n** Cuando se produce una excepción se envía un objeto (excepción) al método que ha provocado el error de ejecución
- n** Si el método que provoca el error no captura la excepción, entonces es la máquina virtual quien la captura y normalmente interrumpe la ejecución del programa

# Programación Java

---

## El control de errores en tiempo de ejecución

```
public static void main(String[] args) {  
    int suma, num[] = { 1, 2, 3 };  
  
    for (int i=0; i <= 3; i++) {  
        suma+=num[i];  
        System.out.println(num[i]);  
    }  
  
    System.out.println(suma/0);  
}
```

# Programación Java

---

## El control de errores en tiempo de ejecución

**n** En el ejemplo anterior se producen dos excepciones:

```
java.lang.ArrayIndexOutOfBoundsException  
java.lang.ArithmeticException
```

- n** `IndexOutOfBoundsException` se produce al acceder a `num[3]` en la iteración del `for` para `i = 3`. El array `num` almacena 3 valores, en los índices 0, 1 y 2, el índice 3 no existe
- n** `ArithmeticException` se produce al dividir por cero en la última instrucción del programa

# Programación Java

---

## Las instrucciones try, catch, finally

- n try** ejecuta un bloque de código y lanza la excepción a la primera sentencia catch que gestione ese tipo de excepción
- n catch** captura la excepción y realiza las acciones correspondientes
- n finally** se ejecuta al final de try y después de catch. Captura las excepciones no capturadas por sentencias catch

# Programación Java

---

## Las instrucciones try, catch, finally

```
try {  
    // bloque de código que gestiona la  
    excepción  
}  
catch ( tipoExcepción1 objeto ) {  
    // Gestión del tipo de excepción 1  
}  
catch ( tipoExcepción2 objeto ) {  
    // Gestión de tipo de excepción 2  
}  
finally {  
    // Gestión de otras excepciones  
}
```

# Programación Java

---

## Las instrucciones try, catch, finally

```
public static void main(String[] args) {  
    try {  
        int suma, num[] = { 1, 2, 3 };  
  
        for (int i=0; i <= 3; i++) {  
            suma+=num[i];  
            System.out.println(num[i]);  
        }  
    }  
    catch (ArrayIndexOutOfBoundsException e) {  
        System.out.print("i fuera de rango " + e.getMessage());  
    }  
    finally {  
        System.out.println(e.getMessage());  
    }  
}
```

# Programación Java

---

## Jerarquía de excepciones

```
java.lang.Object
|
+-- java.lang.Throwable
|
+-- java.lang.Exception
|
+-- java.lang.RuntimeException
|
+-- java.lang.IndexOutOfBoundsException
|
+--
java.lang.ArrayIndexOutOfBoundsException
```

# Programación Java

---

## Jerarquía de excepciones

```
public static void main(String[] args) {  
    try {  
        int suma, num[] = { 1, 2, 3 };  
  
        for (int i=0; i <= 3; i++) {  
            suma+=num[i];  
            System.out.println(num[i]);  
        }  
    }  
    catch (ArrayIndexOutOfBoundsException e) {  
        System.out.print("i fuera de rango " + e.getMessage());  
    }  
    catch (RuntimeException e){  
        System.out.println("error " + e.getMessage());  
    }  
}
```



# Programación Java

---

## La instrucción throws

- n throws indica al compilador las excepciones que puede lanzar un método**
- n Es necesario para todas las excepciones, excepto para las de tipo Error o RuntimeException y sus subclasses**
- n Si se utiliza input-output es necesario declarar la excepción throws IOException. Si no se hace, el compilador muestra un mensaje de error indicando que la excepción no está declarada**

# Programación Java

---

## Las excepciones de input-output

```
public static void main(String[] args) throws IOException {
    try {
        Scanner teclado = new Scanner(System.in);

        System.out.println("Introduzca un numero: ");

        int x = teclado.nextInt();

    }
    catch (NumberFormatException e) {
        System.out.print("Numero no válido !!" );
    }
    catch (IOException e) {
        System.out.print( "Error de entrada" );
    }
}
```

# Programación Java

---

## Clases y objetos

- n Una clase es una representación abstracta de un conjunto de objetos. Los objetos de una clase tienen las mismas características y el mismo comportamiento**
- n Un objeto es una instancia de una clase. En Java los objetos se crean con el operador new**
- n Cada objeto tiene sus propios atributos, lo que le diferencia de otros objetos pertenecientes a la misma clase**

# Programación Java

---

## Clases y objetos

- n La clase String permite definir objetos para almacenar cadenas de caracteres**

```
String hola = new String("Hola")  
String mundo = new String("mundo");
```

- n Los objetos hola y mundo son cadenas de caracteres, cada objeto ocupa un espacio en memoria y tiene sus propios atributos, "hola" y "mundo", respectivamente**

# Programación Java

---

## Clases y objetos

- n Las clases se utilizan para definir tipos de datos o módulos**
- n Un tipo de dato describe un conjunto de objetos y operaciones (métodos). Un módulo es una unidad de descomposición de software**

# Programación Java

---

## Clases y objetos

- n Un objeto es una instancia de una clase. Encapsula estado y comportamiento. Los objetos se crean por instanciación de las clases**
- n Un objeto puede describir una entidad física o una entidad abstracta**
- n Cada objeto tiene sus propios atributos, lo que le diferencia de otros objetos pertenecientes a la misma clase**

# Programación Java

---

## Objetos

- n Un objeto tiene las siguientes características**
  - n Identidad**
  - n Estado**
  - n Comportamiento**
- n La identidad de un objeto le permite distinguirse de otros objetos. El estado de un objeto está determinado por sus datos y el comportamiento del objeto está determinado por los métodos de la clase a la que pertenece**

# Programación Java

---

## Objetos

### **n** Identidad

**La identidad de un objeto le identifica unívocamente y no cambia durante la vida del objeto. Es independiente del estado del objeto**

### **n** Estado

**El estado de un objeto evoluciona en el tiempo, está determinado por los valores de sus atributos. Cada atributo toma un valor en un dominio**



# Programación Java

---

## Objetos

### **n** Comportamiento

**Los métodos que definen el comportamiento de un objeto describen sus acciones y agrupan sus responsabilidades**

**Las acciones de un objeto dependen de su estado y de un estímulo externo, un mensaje enviado por otro objeto**

**El estado y el comportamiento están relacionados. Por ejemplo, un avión no puede aterrizar si está en tierra**

# Programación Java

---

## Objetos: interfaz vs. implementación

- n Los objetos se relacionan a través de interfaces bien definidas. No es necesario que conozcan los detalles de la implementación de otros objetos**
- n La interfaz del objeto está definida por los métodos públicos de la clase**

# Programación Java

---

## Las clases en Java

- n Las clases Java se definen en ficheros independientes, normalmente con extensión .java**
- n La clase se carga en memoria cuando es necesario**
- n Para definir una clase en Java se utiliza la palabra reservada class**

```
public class MiClaseJava {  
  
}
```

# Programación Java

---

## Las clases en Java

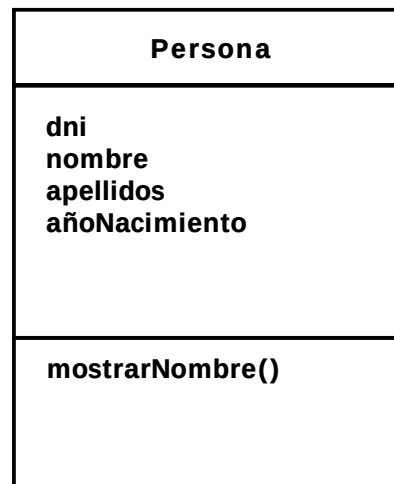
- n El nombre de una clase debe ser un identificador válido en Java**
- n Por norma, los nombres de las clases comienzan con mayúsculas**
- n Las clases públicas deben definirse en ficheros .java**

# Programación Java

---

## Las clases en Java

- n La clase Persona tiene cuatro atributos: dni, nombre, apellido y año de nacimiento**
- n El método mostrarNombre() muestra el nombre por la consola**



# Programación Java

---

## Las clases en Java

```
public class Persona {  
    public String dni, nombre, apellidos;  
    public int añoNacimiento;  
  
    public void mostrarNombre() {  
        System.out.println(nombre + " " + apellidos);  
    }  
}
```

# Programación Java

---

## Las clases en Java

```
public class NombrePersonas {  
    public static void main(String[] args) {  
  
        Persona p1 = new Persona();  
        Persona p2 = new Persona();  
  
        p1.nombre = "Juan";  
        p1.apellidos = "Gonzalez";  
  
        p2.nombre = "Luis";  
        p2.apellidos = "Gómez";  
  
        p1.mostraNombre();  
        p2.mostrarNombre();  
    }  
}
```

# Programación Java

---

## Métodos de clases

- n** Por norma, debe evitarse el uso de propiedades públicas. Es preferible definir métodos para modificar el valor de las propiedades de un objeto
- n** Los métodos set se utilizan para modificar (escribir) el valor de los atributos de un objeto
- n** Los métodos get se utilizan para consultar (leer) el valor de los atributos de un objeto

`getNombre()`

`setNombre()`



# Programación Java

---

## Métodos de clases

```
public class Persona {  
    private String dni, nombre, apellidos;  
    private int añoNacimiento;  
  
    public void mostrarNombre() {  
        System.out.println(nombre + " " + apellidos);  
    }  
  
    public setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public set añoNacimiento(int año) {  
        añoNacimiento = año;  
    }  
}
```

# Programación Java

---

## Métodos de clases

- n** En este ejemplo, el método `setNombre(String nombre)` recibe el parámetro `nombre` para modificar el atributo `nombre` de la clase. La palabra reservada `this` hace referencia a la instancia de la clase y permite diferenciar estas dos variables. El método `setNombre` asigna el valor del parámetro `nombre` al atributo `nombre` del objeto

```
this.nombre = nombre;
```

- n** El método `setAñoNacimiento` no utiliza `this` porque el identificador del parámetro (`año`) es distinto del identificador del atributo de la clase (`añoNacimiento`)

# Programación Java

---

## Métodos de clases

```
public class Cuenta {  
    private double saldo, double limiteCredito;  
  
    public void ingresar(double cantidad) {  
        saldo += cantidad;  
    }  
    public void retirar(double cantidad) {  
        saldo -= cantidad;  
    }  
    public void setLimiteCredito(double cantidad) {  
        limiteCredito = cantidad;  
    }  
    public double getLimiteCredito() {  
        return limiteCredito;  
    }  
}
```

# Programación Java

---

## Constructores

- n El operador new crea una instancia de la clase invocando un método especial de la clase, denominado constructor**
- n El método constructor inicializa el estado del objeto**

**Cuenta miCuentaCorriente = new Cuenta();**

- n Por defecto, Java crea un constructor sin parámetros. El valor inicial de los atributos del objeto depende de la inicialización de las variables de instancia**

# Programación Java

---

## Constructores

- n** Java permite definir métodos constructores para especificar los valores iniciales de los objetos
- n** El nombre de los métodos constructores coincide con el nombre de la clase

# Programación Java

---

## Constructores

```
public class Cuenta {  
    private double saldo, double limiteCredito;  
    private static final double LIMITE = 500;  
  
    public Cuenta() {  
        this.balance = 0;  
        this.limiteCredito = LIMITE;  
    }  
  
    public Cuenta(double limiteCredito) {  
        this.balance = 0;  
        this.limiteCredito = limiteCredito;  
    }  
}
```

# Programación Java

---

## Constructores

```
public class MisCuentas {  
  
    public static void main(String[] args) {  
  
        Cuenta miCuenta1 = new Cuenta();  
        Cuenta miCuenta2 = new Cuenta(6000);  
  
        System.out.print("Limite cuenta 1 (normal): ");  
        System.out.println(miCuenta1.getLimiteCredito());  
  
        System.out.print("Limite cuenta 2 (vip): ");  
        System.out.println(miCuenta2.getLimiteCredito());  
  
    }  
}
```

# Programación Java

---

## Constructores

- n** Java permite definir varios constructores diferentes para crear los objetos de la clase. La declaración de los constructores debe ser diferente y deben utilizar distintos parámetros
- n** En este ejemplo se han definido dos constructores, uno sin parámetros y otro con un parámetro para especificar el límite de crédito de la cuenta



# Programación Java

---

## Referencias a objetos

- n** Cualquier tipo definido en Java con una clase es un tipo no primitivo
- n** Cuando se declara una variable de tipo primitivo, se reserva un espacio de memoria para almacenar su valor
- n** Cuando se declara una variable de tipo no primitivo, se reserva un espacio de memoria para almacenar la referencia al objeto, no el objeto en sí. Es por esto que es necesario utilizar el operador new

# Programación Java

---

## Referencias a objetos

**n** Cuando se utilizan tipos de datos primitivos las sentencias de asignación copian el valor de las variables

```
int x = 100;
```

```
int y = 200;
```

```
int z;
```

```
z = y;      // z almacena el valor inicial de y (200)
```

```
y = x;      // y almacena el valor inicial de x (100)
```

```
x = z;      // el valor de x es 200
```

# Programación Java

---

## Referencias a objetos

- n Cuando se utilizan tipos de datos no primitivos las sentencias de asignación copian la referencia al objeto**

```
Cuenta miCuenta1 = new Cuenta();
```

```
Cuenta miCuenta2 = new Cuenta(6000);
```

```
Cuenta miCuenta3;
```

```
miCuenta3 = miCuenta2; // se copia la referencia a miCuenta2
```

```
miCuenta3.retirar(2000); // hace un retiro de miCuenta2
```

# Programación Java

---

## Inicialización por defecto de las variables de instancia

- n Las variables de tipo numérico byte, short, int o long se inicializan a cero. Las variables float se inicializan a +0.0f y las variables double a +0.0**
- n Las variables de tipo char se inicializan a '\u0000'**
- n Las variables de tipo boolean se inicializan a false**
- n Las referencias a objetos se inicializan a null**

# Programación Java

---

## Ámbito de las variables

Los principales tipos de ámbitos en Java son:

- n** **Ámbito de objeto.** Los atributos de un objeto que no se han declarado `static`
- n** **Ámbito de método.** Las variables y los objetos declarados en un método
- n** **Ámbito de clase.** Las variables declaradas `static` en una clase

# Programación Java

---

## Ámbito de las variables

Los principales tipos de ámbitos en Java son:

- n** **Ámbito de objeto.** Los atributos de un objeto están ligados al espacio de vida del objeto y son accesibles por cualquier método del objeto que no sea static.
- n** **Ámbito de método.** Las variables y los objetos declarados en un método están delimitadas por el ámbito del método

# Programación Java

---

## Ámbito de las variables

- n** **Ámbito de clase.** Las variables declaradas static en una clase viven independientemente de las instancias de la clase. Si las variables son públicas, es posible acceder a ellas desde que se declara la clase

# Programación Java

---

## Ámbito de las variables

```
public class Cuenta {  
    private double saldo, double limiteCredito;  
    public static final double LIMITE = 500;  
  
    public Cuenta() {  
        this.balance = 0;  
        this.limiteCredito = LIMITE;  
    }  
  
    public Cuenta(double limiteCredito) {  
        this.balance = 0;  
        this.limiteCredito = limiteCredito;  
    }  
}
```



# Programación Java

---

## Ámbito de las variables

- n Las variables `saldo` y `limiteCredito` son variables de objeto**
- n La variable `LIMITE` es una variable de clase. Es una variable pública compartida por todas las instancias de la clase**

# Programación Java

---

## Ámbito de los métodos de clase

- n** Un método `static` se puede invocar desde la clase, sin necesidad de una instancia

```
public static void main (String[] args) {  
    String pi = new String("3.141592");  
    double PI;  
  
    PI = Double.parseDouble(pi);  
}
```

- n** `parseDouble` es un método `static` de la clase `Double`

# Programación Java

---

## Relaciones entre clases

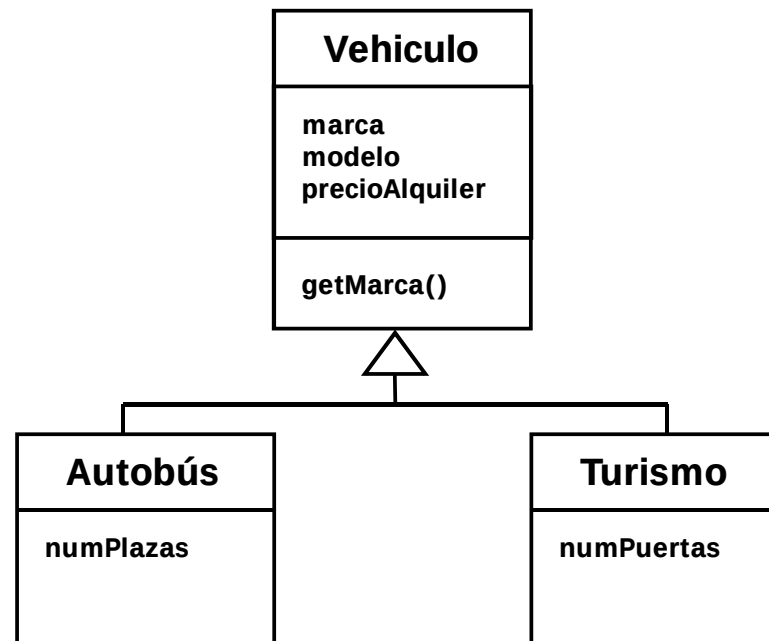
- n En el diseño orientado a objetos existen dos tipos de relaciones básicas: la herencia y la agregación**
- n La herencia se utiliza para especializar y extender las características y el comportamiento de una clase**
- n La agregación se utiliza cuando existe una relación jerárquica entre las clases. Esta relación puede ser del tipo “es parte de” o “forma parte de”**

# Programación Java

---

## Relaciones entre clases: Herencia

- n** En las relaciones de herencia las subclases heredan atributos de la superclase (marca, modelo y precio) y añaden atributos propios

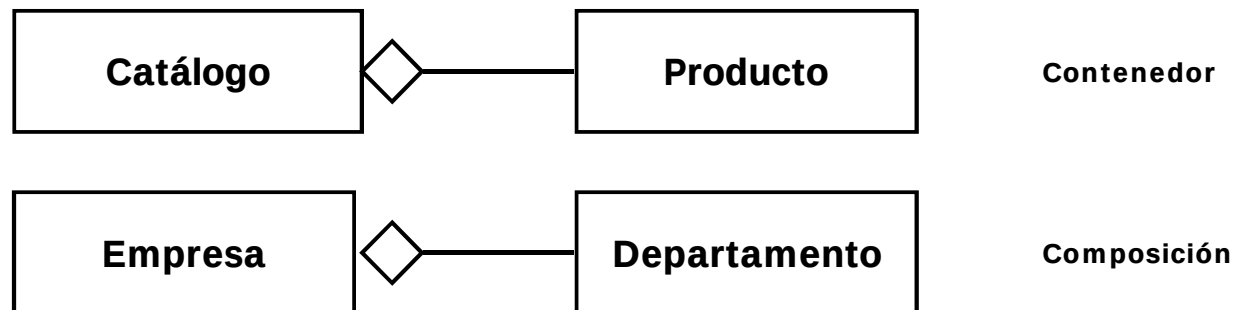


# Programación Java

---

## Relaciones entre clases: Agregación

- n** En las relaciones de agregación se existe el todo y la parte. Puede ser de tipo composición o contenedor. En la composición la parte no puede existir sin el todo



# Programación Java

---

## Herencia y polimorfismo

```
public class Vehiculo {
    private String marca, modelo;
    private double precioAlquiler;

    public Vehiculo(String marca, String modelo) {
        this.marca = marca;
        this.modelo = modelo;
    }
    public void setPrecioAlquiler(double precioAlquiler) {
        this.precioAlquiler = precioAlquiler;
    }
    public void imprimirAtributos() {
        System.out.print(" Modelo: " + marca + " " +
            modelo + "\t" + "Precio: " + precioAlquiler);
    }
}
```

# Programación Java

---

## Herencia y polimorfismo

```
public class Autobus extends Vehiculo {
    private int numPlazas;

    public Autobus(String marca, String modelo,
                   double precioAlquiler, int numPlazas) {
        super(marca, modelo, precioAlquiler);
        setPlazas(numPlazas);
    }
    public void setPlazas(int numPlazas) {
        this.numPlazas = numPlazas;
    }
    public void imprimirAtributos() {
        super.imprimirAtributos();
        System.out.print("\t" + "Plazas : " + numPlazas);
    }
}
```

# Programación Java

---

## Herencia y polimorfismo

```
public class Turismo extends Vehiculo {
    private int numPuertas;

    public Turismo(String marca, String modelo,
                   double precioAlquiler, int numPuertas) {
        super(marca, modelo, precioAlquiler);
        setPuertas(numPuertas);
    }
    public void setPuertas(int numPuertas) {
        this.numPuertas = numPuertas;
    }
    public void imprimirAtributos() {
        super.imprimirAtributos();
        System.out.print("\t" + "Puertas: " + numPuertas);
    }
}
```



# Programación Java

---

## Herencia y polimorfismo

- n Las clases Autobus y Turismo son subclases de Vehiculo. Estas clases son especializaciones de Vehiculo y “extienden” sus atributos y comportamiento**
- n Los constructores de las subclases utilizan el constructor de la clase Vehiculo. El constructor de la superclase se invoca con super**
- n El método imprimirAtributos() se sobrescribe en las subclases Autobus y Turismo para mostrar los atributos propios de la subclase**

# Programación Java

---

## Herencia y polimorfismo

```
public class Vehiculos {  
    public static void main(String[] args) {  
        List listaVehiculos = new ArrayList();  
  
        listaVehiculos.add(new Vehiculo("Mercedes", "E", 55));  
        listaVehiculos.add(new Autobus("Mercedes", "M20", 100, 20));  
        listaVehiculos.add(new Turismo("BMW", "525", 50, 4));  
        listaVehiculos.add(new Turismo("Audi", "A4", 55, 4));  
        listaVehiculos.add(new Autobus("Mercedes", "M10", 120, 30));  
        listaVehiculos.add(new Turismo("VW", "Passat", 35, 4));  
        listaVehiculos.add(new Turismo("Audi", "A3", 30, 2));  
        listaVehiculos.add(new Turismo("Mercedes", "E", 60, 4));  
        listaVehiculos.add(new Vehiculo("Audi", "A3", 30));  
        imprimirElementosLista((ArrayList)listaVehiculos);  
    }  
}
```

# Programación Java

---

## Herencia y polimorfismo

- n** La clase main utiliza el objeto listaVehiculos, una instancia de la clase List, para almacenar los vehículos
- n** Cuando se añade un objeto vehículo a listaVehiculos, se invoca al constructor de la clase correspondiente: Vehiculo, Autobus o Turismo

**Turismo("Audi","A4",55,4)**

**Vehiculo("Mercedes","E",55)**

**Autobus("Mercedes","M20",100,20)**

# Programación Java

---

## Modificadores de acceso a clases de un mismo paquete

| Modificador               | Heredado | Accesible |
|---------------------------|----------|-----------|
| default (sin modificador) | SI       | SI        |
| public                    | SI       | SI        |
| protected                 | SI       | SI        |
| private                   | NO       | NO        |

# Programación Java

---

## Modificadores de acceso a clases de otro paquete

| Modificador               | Heredado | Accesible |
|---------------------------|----------|-----------|
| default (sin modificador) | NO       | NO        |
| public                    | SI       | SI        |
| protected                 | SI       | NO        |
| private                   | NO       | NO        |

# Programación Java

---

## Modificadores de acceso

- n Los atributos y métodos private de una superclase no se heredan en las subclases**
- n Los atributos y métodos protected de una superclase se heredan y son accesibles desde las subclases**
- n Los atributos y métodos public de una clase son accesibles dentro y fuera del paquete al que pertenece la clase**
- n Si no se utiliza un delimitador, entonces el atributo o el método son públicos**

# Programación Java

---

## Modificadores de acceso

- n Utilizar `public` si el atributo o método se debe heredar y ser usado desde otro paquete**
- n Utilizar `protected` si el atributo o método se debe heredar y sólo debe ser usado desde el paquete al que pertenece la clase**
- n Utilizar `private` si el atributo o método es privado de la clase**

# Programación Java

---

## Clases abstractas

- n Las clases abstractas se utilizan para especificar una interfaz y no implementan los métodos**
- n Las clases abstractas heredan métodos abstractos, métodos declarados que no han sido implementados. Las subclases se encargan de implementar los métodos**
- n Las clases abstractas no se pueden instanciar**



# Programación Java

---

## La palabra reservada final

- n Si final precede a una variable, es una constante que no se puede modificar**
- n Si final precede a un método, no se puede sobrescribir**
- n Si final precede una clase, no permite definir subclases a partir de ella**

# Programación Java

---

## Composición y agregación

- n** La composición se utiliza para definir clases compuestas por otras clases. En este tipo de relaciones todo-parte, normalmente la parte no puede existir sin el todo (p.e. los departamentos de una empresa)
- n** Si se trata de una clase contenedor, entonces la parte puede existir sin necesidad del todo (p.e. los productos de un catálogo)
- n** Para crear contenedores es recomendable utilizar las clases Vector o List

# Programación Java

---

## Vectores

- n La clase Vector de Java es un array de longitud variable. Esta clase tiene los siguientes constructores**

|                                  |                                           |
|----------------------------------|-------------------------------------------|
| <b>Vector()</b>                  | <b>Tamaño 100 y se duplica el tamaño</b>  |
| <b>Vector(int size)</b>          | <b>Tamaño dado y se duplica el tamaño</b> |
| <b>Vector(int size, int inc)</b> | <b>Tamaño dado e incremento</b>           |

- n Para añadir un elemento al vector se invoca el método add, indicando el objeto a insertar add(Object element)**

```
Vector listaVehiculos = new Vector(50);  
listaVehiculos.add(new Turismo("Audi","A4",55,4))
```

# Programación Java

---

## Vectores

- n La clase Vector define métodos que tienen en cuenta el índice o posición dentro del Vector**
- n El primer elemento se almacena en la posición cero. Para insertar un objeto en una posición determinada se utiliza el método `insertElementAt(Object element, int position)`**
- n Cuando se utiliza este método el resto de elementos del vector son desplazados**
- n El método `size()` devuelve el número de elementos del vector, el método `capacity()` indica su capacidad máxima**

# Programación Java

---

## Vectores

- n** En el siguiente ejemplo `listaVehiculos.size()` es 1 y `listaVehiculos.capacity()` es 50

```
Vector listaVehiculos = new Vector(50);  
listaVehiculos.add(new Turismo("Audi","A4",55,4))
```

- n** El método `removeElementAt(int position)` elimina el objeto almacenado en una posición del vector y desplaza los elementos para llenar la posición libre
- n** Los métodos `elementAt(int position)` y `get(int position)` devuelven el objeto almacenado en la posición indicada

# Programación Java

---

## Listas

- n La clase ArrayList es una lista ordenada de objetos. Esta clase tiene los siguientes constructores**

|                                       |                                     |
|---------------------------------------|-------------------------------------|
| <b>ArrayList()</b>                    | <b>Tamaño por defecto</b>           |
| <b>ArrayList(int initialCapacity)</b> | <b>Tamaño dado</b>                  |
| <b>ArrayList(Collection coll)</b>     | <b>Tamaño dado por la colección</b> |

- n El métodos add(Object element) inserta un objeto en la lista y add(Object element, int position) inserta un elemento en la posición indicada. Para acceder a un objeto de la lista se utiliza el método get(int position)**

# Programación Java

---

## Listas

```
public class Vehiculos {  
    public static void main(String[] args) {  
        List listaVehiculos = new ArrayList();  
  
        listaVehiculos.add(new Vehiculo("Mercedes", "E", 55));  
        listaVehiculos.add(new Autobus("Mercedes", "M20", 100, 20));  
        listaVehiculos.add(new Turismo("BMW", "525", 50, 4));  
        listaVehiculos.add(new Turismo("Audi", "A4", 55, 4));  
        listaVehiculos.add(new Autobus("Mercedes", "M10", 120, 30));  
        listaVehiculos.add(new Turismo("VW", "Passat", 35, 4));  
        listaVehiculos.add(new Turismo("Audi", "A3", 30, 2));  
        listaVehiculos.add(new Turismo("Mercedes", "E", 60, 4));  
        listaVehiculos.add(new Vehiculo("Audi", "A3", 30));  
        imprimirElementosLista((ArrayList)listaVehiculos);  
    }  
}
```

# Programación Java

---

## Listas

```
private static void imprimirLista(ArrayList lista) {  
    for (int i = 0; i < lista.size(); i++) {  
        Vehiculo v = (Vehiculo)lista.get(i);  
  
        v.imprimirAtributos();  
  
        System.out.println("");  
    }  
}
```



# Introspección

---

## Información sobre las clases

- n** Java ofrece utilidades de introspección para obtener información interna de las clases. Los objetos Java pueden informar sobre su propia estructura
- n** La introspección permite que un objeto o una clase informe de la clase a la que pertenece
- n** La clase más importante es `java.lang.Class`. Esta clase es un descriptor o referencia de una clase y permite describir cualquier clase Java. El paquete `java.lang.reflect` es relevante para la introspección

# Introspección

---

**Conocer la clase a la que pertenece un objeto**

- n Es posible saber la clase a la que pertenece un objeto en tiempo de ejecución**

```
Persona p = new Persona();  
Class claseObjeto = p.getClass();  
System.out.print("La clase es: " + claseObjeto.getName());
```

- n El objeto claseObjeto pertenece a la clase Class, que es un descriptor de clase. Este descriptor devuelve el nombre de la clase con el método getName()**

# Introspección

---

**Conocer la clase a la que pertenece una clase**

**n Si el descriptor de clase se aplica a una clase entonces:**

**Exception.class.getName()**

**Scanner.class.getName()**

**n El método getName() devuelve el nombre de la clase y la jerarquía de paquetes**

**java.lang.Exception**

**java.util.Scanner**

# Utilidades

---

## Ordenación con Comparator

- n** Para ordenar los elementos de una lista de objetos de diverso tipo es necesario definir una clase que implemente la interfaz `Comparator`
- n** `Comparator` es una interfaz que define un conjunto de métodos para comparar objetos. El método `compare()` de `Comparator` se utiliza para comparar y ordenar. La clase que implemente `Comparator` debe sobrescribir el método `compare()`

# Utilidades

---

## Ordenación con Comparator (Clase Persona)

```
public class Persona {  
    private int id;  
    private String nombre, apellidos;  
  
    public Persona(int id) {  
        this.id = id;  
    }  
    public Persona(int id, String nombre, String apellidos) {  
        this.id = id;  
        this.nombre = nombre;  
        this.apellidos = apellidos;  
    }  
    public getId() {  
        return id;  
    }  
}
```

# Utilidades

---

## Ordenación con Comparator (Clase ComparadorPersona)

```
import java.util.Comparator;

public class ComparadorPersona implements Comparator {

    public int compare(Object a, Object b) {
        if (((Persona)a).getId() < ((Persona)b).getId())
            return -1;

        if (((Persona)a).getId() > ((Persona)b).getId())
            return 1;

        return 0;
    }
}
```

# Utilidades

---

## Ordenación con Comparator (Programa principal)

```
import java.util.*;

public class Personas {

    private static void imprimirLista(ArrayList lista) {
        System.out.println("");

        for (int i = 0; i < lista.size(); i++) {
            Persona p = (Persona)lista.get(i);

            System.out.println "[" + p.getId() + "]" + " " +
                               p.getNombre() + " " + p.getApellidos());
        }
    }
}
```

# Utilidades

---

## Ordenación con Comparator (Programa principal y 2)

```
public static void main(String[] args) {  
    List listaPersonas = new ArrayList();  
  
    listaPersonas.add(new Persona(10, "Juan", "González"));  
    listaPersonas.add(new Persona(30, "Luis", "López"));  
    listaPersonas.add(new Persona(40, "María", "Soto"));  
    listaPersonas.add(new Persona(50, "Pedro", "Sánchez"));  
    listaPersonas.add(new Persona(60, "Ana", "Rodríguez"));  
  
    imprimirLista((ArrayList)listaPersonas);  
  
    listaPersonas.remove(2);  
  
    listaPersonas.add(new Persona(20, "Mariana", "Rocha"));  
    listaPersonas.add(new Persona(90, "Beatriz", "Sada"));  
    listaPersonas.add(new Persona(70, "Mar", "García"));  
}
```



# Utilidades

---

## Ordenación con Comparator (Programa principal y 3)

```
imprimirLista((ArrayList)listaPersonas);

/* el método sort de la clase Collections permite
 * ordenar la lista con un objeto ComparadorPersona
 */

Collections.sort(listaPersonas,new ComparadorPersona());

imprimirLista((ArrayList)listaPersonas);

} // fin de static void main

} // fin de la clase Personas
```

# Utilidades

---

## Números aleatorios

- n** La clase `Random` de `java.util.Random` genera números aleatorios

```
Random r = new Random();
```

```
for (int i=1; i<=10; i++ )  
    System.out.println( r.nextDouble() );
```

- n** La clase `Random` ofrece `double nextGaussian()`, que da números aleatorios con distribución gaussiana centrada en cero y con una desviación estándar en torno a 1. `nextInt(n)` da números entre 0 y n-1

# Utilidades

---

## Funciones matemáticas

**n** La clase **Math** ofrece funciones matemáticas como:

|                        |                                                                   |
|------------------------|-------------------------------------------------------------------|
| <b>Valor absoluto</b>  | <b>abs(x)</b>                                                     |
| <b>Potencia</b>        | <b>pow(x, y)</b> para elevar $x^y$                                |
| <b>Raíz cuadrada</b>   | <b>sqrt(x)</b> para la raíz cuadrada                              |
| <b>Comparación</b>     | <b>min(X, y), max(x, y)</b>                                       |
| <b>Trigonométricas</b> | <b>sin(x), cos(x), tan(x)</b><br><b>asin(x), acos(x), atan(x)</b> |
| <b>Exponencial</b>     | <b>exp(x)</b>                                                     |
| <b>Logarítmica</b>     | <b>log(x)</b>                                                     |
| <b>e</b>               | <b>E</b>                                                          |
| <b>pi</b>              | <b>PI</b>                                                         |

# Utilidades

---

## Tareas temporizadas

- n** Una tarea temporizada se repite de forma periódica. La clase abstracta `TimerTask` tiene un método abstracto `run()` que permite ejecutar una tarea utilizando un temporizador
- n** El temporizador es un objeto de la clase `Timer` que controla el tiempo y ejecuta la tarea temporizada invocando su método `run()`. La clase `TareaTemporizada` es una subclase de `TimerTask`

```
TareaTemporizada tarea = new TareaTemporizada();  
Timer temporizador = new Timer();  
temporizador.schedule(tarea, inicio, periodo);
```



#### FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO

|                       |                                                       |
|-----------------------|-------------------------------------------------------|
| PROGRAMA DE FORMACIÓN | PROGRAMACIÓN DE APLICACIONES Y SERVICIOS PARA LA NUBE |
|-----------------------|-------------------------------------------------------|

|             |                                                                                         |                           |                                                                                                 |
|-------------|-----------------------------------------------------------------------------------------|---------------------------|-------------------------------------------------------------------------------------------------|
| COMPETENCIA | 220501113 Administrar base de datos de acuerdo con los estándares y requisitos técnicos | RESULTADOS DE APRENDIZAJE | 220501113 -01 Modelar la base de datos de acuerdo a los requerimientos del sistema informático. |
|-------------|-----------------------------------------------------------------------------------------|---------------------------|-------------------------------------------------------------------------------------------------|

|                                 |                                                                                                                                                                                                                                                                                                                                                                              |
|---------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| NÚMERO DEL COMPONENTE FORMATIVO | 1                                                                                                                                                                                                                                                                                                                                                                            |
| NOMBRE DEL COMPONENTE FORMATIVO | Conceptos generales de bases de datos.                                                                                                                                                                                                                                                                                                                                       |
| BREVE DESCRIPCIÓN               | En la sociedad de la información es imprescindible el empleo de tecnológicas, técnicas y procedimiento que a lo largo de las últimas décadas han consolidado un marco conceptual importante y necesario en todo proceso de administración de datos, este componente acerca estos conceptos de manera estructurada para avanzar en los procesos de gestión de la información. |
| PALABRAS CLAVE                  | Base de datos, relacional, integridad, dependencias.                                                                                                                                                                                                                                                                                                                         |

|                  |           |
|------------------|-----------|
| AREA OCUPACIONAL | No Aplica |
| IDIOMA           | Español   |

#### A. TABLA DE CONTENIDOS:

##### Introducción

##### 1. Conceptos generales de bases de datos

- 1.1. Tipos de datos y restricción de no nulidad.
- 1.2. Tipos de bases de datos.
- 1.3. Clasificación de bases de datos.
- 1.4. Sistema de gestión de bases de datos.

##### 2. Modelo entidad relación

- 2.1. Relaciones entre entidades.
- 2.2. Relaciones de uno a muchos.
- 2.3. Relaciones de un uno a muchos.
- 2.4. Relaciones de uno a uno

##### 3. Normalización

- 3.1. Dependencias funcionales.
- 3.2. Diseño relacional.
- 3.3. Reglas de integridad
- 3.4. Lenguajes de los sistemas administradores de bases de datos

##### 4. Sistema gestor de base de datos



## B. DESARROLLO DE CONTENIDOS:

### Introducción

Una base de datos se puede percibir como un “almacén” de información que se define y se crea una sola vez, para guardar grandes cantidades de datos de forma organizada (o estructurada), con el fin de poder encontrarla y utilizarla fácilmente. Se presentará el concepto y características de las bases de datos actuales, teniendo en cuenta que cada base de datos es diseñada para cumplir los requisitos de información de una organización o una empresa. Estos diseños están concebidos para emplear Sistemas de Gestión de Bases de Datos (en adelante SGDB), que es un sistema o servicio informático que permite a las personas definir, crear, dar soporte y mantenimiento a las bases de datos, controlando el acceso de forma segura.

### 1. Conceptos generales de bases de datos

El concepto general de bases de datos según la RAE es el “conjunto de datos organizado de tal modo que permita obtener con rapidez diversos tipos de información” (RAE ,2001). De acuerdo a esta definición, una hoja de cálculo de Excel puede considerarse una base datos, o un conjunto de archivos debidamente organizados, o la lista de nombres y teléfonos que está en nuestros *Smartphone*. En principio es correcto llamarle base de datos a estos ejemplos, sin embargo, en el contexto de desarrollo de *software*, se referirá a ese conjunto de información que puede ser almacenada en grandes cantidades de forma organizada y es gestionada desde a través de un **sistema de gestión de bases de datos** (SGBD).

Para poder organizar y definir la información de forma sistemática, las bases de datos también deben poder almacenar una descripción precisa de los datos que contiene conocida como **metadatos** a los que se le relaciona el tipo de información que conceptualmente guardada (es decir si se agrega una explicación de la naturaleza del dato en la empresa) se da origen a lo que conoce como **diccionario de datos o catálogo de datos**.

#### **Ejemplo:**

Determinar los metadatos del siguiente problema.

Una empresa tiene una base datos de llamadas telefónicas que entran y salen de su planta telefónica donde se relacionan los números de teléfonos del llamante y la extensión telefónica que recibe o hace la llamada, la fecha y la hora de la llamada, si fue o no atendida, y la duración de la misma como se muestra en la siguiente ilustración.



| calldate              | src        | dst          | duration | disposition |
|-----------------------|------------|--------------|----------|-------------|
| 2021-04-10 08:48:50.0 | 3176964335 | 444          | 00:09:23 | ANSWERED    |
| 2021-04-10 08:54:14.0 | 3217212347 | 444          | 00:07:47 | ANSWERED    |
| 2021-04-10 08:57:17.0 | 3203857798 | 444          | 00:05:17 | ANSWERED    |
| 2021-04-10 09:03:27.0 | 3226140856 | 666          | 00:04:22 | ANSWERED    |
| 2021-04-10 09:14:13.0 | 28201400   | 666          | 00:00:36 | ANSWERED    |
| 2021-04-10 09:14:45.0 | 3148915614 | 666          | 00:00:34 | ANSWERED    |
| 2021-04-10 09:15:29.0 | 3148915614 | 666          | 00:02:27 | ANSWERED    |
| 2021-04-10 09:18:13.0 | 3148915614 | 666          | 00:01:28 | ANSWERED    |
| 2021-04-10 09:07:33.0 | 3005373259 | 444          | 00:12:12 | ANSWERED    |
| 2021-04-10 09:13:19.0 | 3124523352 | 444          | 00:09:06 | ANSWERED    |
| 2021-04-10 09:21:30.0 | 7006       | 883108608263 | 00:00:54 | ANSWERED    |
| 2021-04-10 09:22:41.0 | 3125228302 | 666          | 00:03:12 | ANSWERED    |
| 2021-04-10 09:28:04.0 | 3203498937 | s            | 00:00:05 | ANSWERED    |
| 2021-04-10 09:29:13.0 | 3017363892 | tdial        | 00:00:00 | FAILED      |
| 2021-04-10 09:29:23.0 | 3017363892 | tdial        | 00:00:00 | NO ANSWER   |
| 2021-04-10 09:30:07.0 | 7006       | 883507070131 | 00:00:00 | NO ANSWER   |
| 2021-04-10 09:30:28.0 | 7006       | 883507070131 | 00:00:00 | NO ANSWER   |
| 2021-04-10 09:28:35.0 | 3203498937 | 666          | 00:02:20 | ANSWERED    |
| 2021-04-10 09:30:58.0 | 7006       | 883507070131 | 00:00:00 | NO ANSWER   |
| 2021-04-10 09:28:32.0 | 3184155040 | 666          | 00:03:20 | ANSWERED    |
| 2021-04-10 09:27:51.0 | 3235767896 | 666          | 00:04:32 | ANSWERED    |
| 2021-04-10 09:28:49.0 | 3158628495 | 444          | 00:06:57 | ANSWERED    |
| 2021-04-10 09:35:45.0 | 3214078916 | 666          | 00:00:52 | ANSWERED    |
| 2021-04-10 09:35:18.0 | 3004668735 | 1            | 00:02:07 | ANSWERED    |
| 2021-04-10 09:39:53.0 | 3226140856 | 666          | 00:02:15 | ANSWERED    |
| 2021-04-10 09:37:02.0 | 3046032130 | 666          | 00:05:30 | ANSWERED    |
| 2021-04-10 09:44:57.0 | 7006       | 883113243787 | 00:00:00 | NO ANSWER   |

Ilustración 1. Base de datos de llamadas telefónicas.

NOTA IMPORTANTE: la ilustración 1 muestra una tabla de datos que tiene las características de que en cada columna se agrupa un tipo de dato particular, y cada fila es el conjunto de dato de se llama registro que comparten una relación (la relación es que cada fila corresponde a una única llamada telefónica), a esta estructura se le llama bidimensional, filas y columnas son las dos dimensiones.

Comentado [OG1]: Captura del documento en Excel denominado "Datos ejemplo" hoja "Exportar datos de trabajo".

Comentado [OG2]: Utilizar bloque de texto destacado Imagen+texto

Comentado [OG3]: Cajón texto color

Los metadatos de la lustración anterior serian la definición de cata tipo de dato como se lista a continuación:

**calldate**: tiene la estructura YYYY-MM-DD HH:MI:SS.Z , donde:

- YYYY sería el año un dato numérico entero de cuatro dígitos.
- MM el mes un dato numérico entero de dos dígitos entre 01 y 12.
- DD un dato numérico entero de dos dígitos entre 01 y 31
- HH un dato numérico de dos dígitos entre 00 y 23.
- MI y SS un dato numérico de dos dígitos ente 00 y 59
- Z Un dato numérico de un dígito (para representar milésimas de segundo).

Comentado [OG4]: Listado no ordenado color.

Como este es el conjunto de datos de la primera columna se puede que use un término que describa lo anterior con una sola palabra que es **TIMESTAMP**, de esta forma esa primera columna puede quedar correctamente descrita como:

**calldate** **TIMESTAMP(1)**      Donde 1 es el número de dígitos que milisegundos (Z).

**src y dst** : empleado para fuente de la llamada o llamante (src) y destino de la llamada (dst) puede ser un número telefónico o de extensión telefónica, algunos tienen el código de país seguido del signo más ejemplo: 57+3155008002. Por lo tanto, se puede describir como una cadena de texto que no superará los 25 caracteres, pero puede tener menos de 25 caracteres, así el metadato iría descrito:



|                                                              |                                                                                                   |
|--------------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| <code>src VARCHAR(25)</code><br><code>dst VARCHAR(25)</code> | De esta forma decimos que es una cadena de caracteres que varía en longitud de 0 a 25 caracteres. |
|--------------------------------------------------------------|---------------------------------------------------------------------------------------------------|

**duration:** tiene la estructura HH:MI:SS y como se vio antes representa el tiempo en horas, minutos y segundos:

|                       |                                                        |
|-----------------------|--------------------------------------------------------|
| <code>src TIME</code> | TIME se usa para esta estructura de dato en particular |
|-----------------------|--------------------------------------------------------|

**disposition:** indica si la llamada es atendida ANSWER, no atendida NO ANSWER, o si fallo la FAIL. Se puede representar con una cadena de 10 caracteres.

Por lo tanto, la solución de los metadatos de la base de datos de la ilustración 1 son:

```
calldate TIMESTAM(1),  
src VARCHAR(25),  
dst VARCHAR(25),  
src TIME,  
disposition VARCHAR(10)
```

Comentado [OG5]: Utilizar fuente consola. Courier new

Note que cada uno de ellos fue separado por un coma para que se diferencie de los demás.

#### Ejemplo 2:

Conociendo el objetivo de la base datos de la ilustración 1 y los metadatos cree un diccionario de dato o un catálogo de datos.

Solución:

|             |                                                                       |        |                                                                           |
|-------------|-----------------------------------------------------------------------|--------|---------------------------------------------------------------------------|
| NOMBRE      | Base datos de llamadas telefónicas                                    |        |                                                                           |
| CREACIÓN    | 27/02/2021                                                            |        |                                                                           |
| DESCRIPCIÓN | Registro de las llamadas telefónicas de la PBX de la empresa STD LTDA |        |                                                                           |
| CAMPO       | TIPO DATO                                                             | TAMAÑO | DESCRIPCIÓN                                                               |
| calldate    | TIMESTAMP                                                             | 1      | Representa el momento exacto en que entra o sale la llamada               |
| src         | VARCHAR                                                               | 25     | Fuente (source) de la llamada, número de quien hace la llamada telefónica |
| dst         | VARCHAR                                                               | 25     | Número del destino de la llamada telefónica                               |
| duration    | TIME                                                                  | 0      | Duración de la llamada telefónica                                         |
| disposition | VARCHAR                                                               | 10     | Estado final en que se considera quedó la llamada                         |

Tabla 1 Diccionario de datos base de llamadas telefónica.

Comentado [OG6]: Utilizar bloque de texto destacado. Imagen+texto  
La imagen sería la tabla1.

Como se puede notar en la tabla anterior, un diccionario de datos aporta información de la estructura de los datos y el uso que se da a cada dato en la empresa, organización o sistema de información, pero no sin antes darle un nombre a todo el conjunto de datos (primer fila), una fecha (segunda fila) y una descripción (tercera fila) del conjunto de los datos. En la tabla 1 se puede identificar la independencia que existe entre la lógica de los datos (representado por la columna DESCRIPCIÓN del dato) y el almacenamiento físico (estructura de almacenamiento representado por las columnas TIPO DE DATO y TAMAÑO).





### 1.1. Tipos de datos y restricción de no nulidad.

Existen muchos tipos de datos y varían según el SGDB, ya que cada sistema gestor define sus propios tipos de datos, aunque existen las equivalencias notables y otras no tan notables vemos una tabla que nos relacione el tipo de datos según los SGDB más comunes del mercado como *ORACLE*, *PostgreSQL*, *MySQL* y *SQLServer*.

| Tipo de dato         | ORACLE   | PostgreSQL               | MySQL    | SQLServer  |
|----------------------|----------|--------------------------|----------|------------|
| Cadena caracteres    | VARCHAR2 | CHARACTER VARYING        | VARCHAR  | VARCHAR    |
| Cadena texto         | TEXT     | TEXT                     | TEXT     | NTEXT      |
| Entero pequeño       | SMALLINT | SMALLINT                 | SMALLINT | SMALLINT   |
| Entero               | INTEGER  | INTEGER                  | INT      | INT        |
| Fecha                | DATE     | DATE                     | DATE     | DATE       |
| Fecha y hora         | DATE     | TIMESTAMP WITH TIME ZONE | DATETIME | DATETIME 2 |
| Hora                 | DATE     | TIME                     | TIME     | TIME       |
| Entero con decimales | FLOAT    | REAL                     | FLOAT    | FLOAT      |

Tabla 2. Tipos de datos según bases de datos.

Según el tipo de sistema gestor de bases de datos cada uno define sus tipos de datos, que de alguna forma son muy parecidos entre ellos, o existen equivalencias, las diferencias son pocas y están relacionadas con aspectos técnicos del almacenamiento, se pueden ver más detalles de cada uno de ellos en los sitios oficiales de cada motor de base de datos:

| Nombre     | Enlace de acceso                                                                                                                                                                                                      |
|------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| ORACLE     | <a href="https://docs.oracle.com/cd/E11882_01/server.112/e41085/sqlqr06002.htm#SQLQR959">https://docs.oracle.com/cd/E11882_01/server.112/e41085/sqlqr06002.htm#SQLQR959</a>                                           |
| PostgreSQL | <a href="https://www.postgresql.org/docs/13/datatype.html#DATATYPE-TABLE">https://www.postgresql.org/docs/13/datatype.html#DATATYPE-TABLE</a>                                                                         |
| MySQL      | <a href="https://dev.mysql.com/doc/refman/8.0/en/data-types.html">https://dev.mysql.com/doc/refman/8.0/en/data-types.html</a>                                                                                         |
| SQLServer  | <a href="https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15">https://docs.microsoft.com/en-us/sql/t-sql/data-types/data-types-transact-sql?view=sql-server-ver15</a> |

Existen varios tipos de restricciones, por ahora veremos ya se ha visto la restricción de longitud, por ejemplo:

```
src VARCHAR(25)
```

En él se definen que **src**, es una columna que no puede exceder los 25 caracteres. De la misma forma existe una forma de definir, que es obligatorio registrar un dato en esa columna y se llama **restricción de no nulidad**, es decir, que ninguna fila puede no pueden tener valor nulo. De esta forma en el ejemplo de la base de datos de llamadas telefónicas si se define que todo y cada uno de los datos son obligatorios (usando las palabras **NOT NULL**), la metadata que define la estructura de los datos quedaría así:

```
calldate TIMESTAM(1) NOT NULL,  
src VARCHAR(25) NOT NULL,  
dst VARCHAR(25) NOT NULL,  
src TIME NOT NULL,  
disposition VARCHAR(10) NOT NULL
```



Algunos sistemas, para abreviar en la decisión de no nulidad acostumbra usar el acrónimo **NN** para representar NOT NULL.

## 1.2. Tipos de bases de datos.

Las bases de datos han evolucionado a lo largo del tiempo, usando diferentes tipos de modelos o enfoques, se comprobado que algunos de ellos tienen mejor desempeño en distintos tipos de aplicación, La estructura en la que se configuran los datos al interior de las bases de datos es una forma de clasificar:

### a. Bases de datos con estructura jerárquica

En las primeras bases de datos empleadas son la de estructura jerárquica. Las relaciones entre registros forman una estructura en árbol, en la actualidad un ejemplo de estas bases de datos es la base de datos LDAP usada en la actualidad para sistema de directorio de datos tales como usuario, dispositivos, nombres, contraseñas, direcciones etc. dentro de una red de computadores.

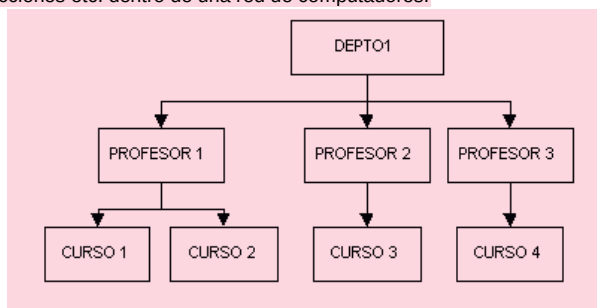


Ilustración 2. Base de datos Jerárquica (Fuente

[https://es.wikipedia.org/wiki/Sistema\\_de\\_gesti%C3%B3n\\_de\\_bases\\_de\\_datos#/media/Archivo:DB\\_jerarquica.png](https://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_bases_de_datos#/media/Archivo:DB_jerarquica.png) )

### b. Bases de datos con estructura en red

Esta estructura es un poco más compleja que la anterior porque contiene otras relaciones que las jerárquicas admitiendo relaciones de cada registro otros de manera que puede seguir por distintos caminos para acaecer la información.

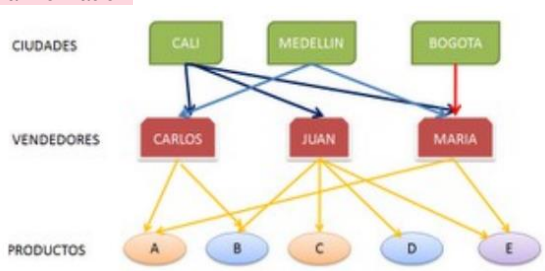


Ilustración 3. Base de datos con estructura de red. (Fuente

<https://gestionbasesdatos.readthedocs.io/es/latest/Tema1/Teoria.html>)



### c. Bases de datos con estructura relacional

Es estas bases de datos son multipropósito, lo que la hace las más extendidas en la industria y más empleada, y también la más compleja de aprender ampliamente se bordara el diseño e implementación de este tipo de bases de datos.

Se basan en tablas como las que se presentó en la ilustración 1 y en la relación de los datos de unas tablas con otras.

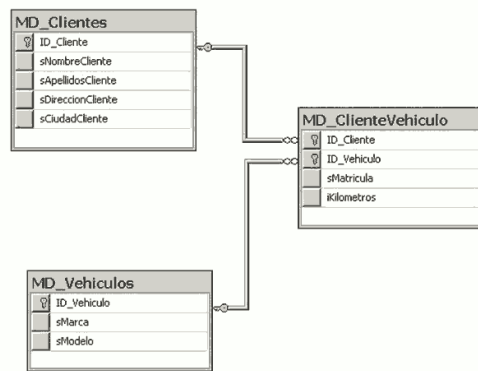


Imagen. Tablas y estructura relacional.

(Fuente <https://images.app.goo.gl/HqMzrfU1gTpSY1FG6>)

### d. Bases de datos con estructura multidimensional

Es la extensión de las bases de datos de estructuras relacionales, como se vio en las relacionales se basan en estructuras bidimensionales (tablas), mientras la multidimensional en cubos (tres dimensiones), o más complejas N dimensiones.

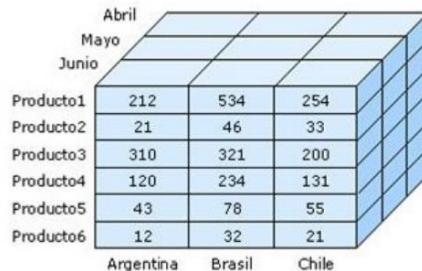


Ilustración 3. Estructura de datos de CUBO. (Fuente

<https://gestionbasesdatos.readthedocs.io/es/latest/Tema1/Teoria.html>).

### e. Bases de datos con estructura orientada a objetos

Una base de datos orientada a objetos es un sistema de gestión de base de datos mediante el cual representamos la información en forma de objetos que son utilizados en programación orientada a objetos (Kyocera, 2021).

**Comentado [OG7]:** Utilizar pestaña o tabs para que el aprendiz pueda ir navegando sobre los diferentes tipos de bases de datos:

- a. Bases de datos con estructura jerárquica
- b. Bases de datos con estructura en red
- c. Bases de datos con estructura relacional
- d. Bases de datos con estructura multidimensional
- e. Bases de datos con estructura orientada a objetos

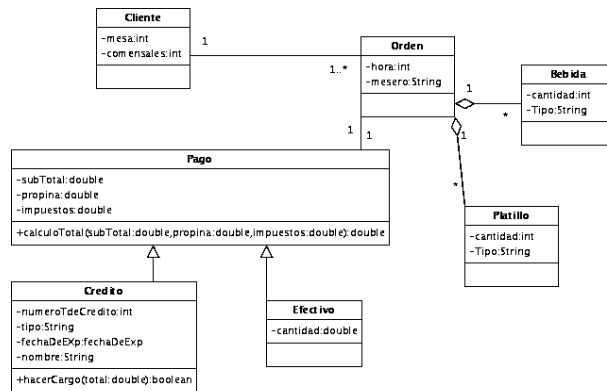


Imagen. Estructura orientada a objetos.  
(Fuente <https://images.app.goo.gl/JNk8HeHE4LZ1rKc59>)

Cada tipo de bases de datos tiene un ámbito de aplicación en el que es mejor su desempeño. Por ejemplo, las bases de datos jerárquicas tienen un mejor desempeño en operaciones de consulta de datos puntuales, mientras que las bases de datos relacionales son precisas para garantizar la calidad de los datos y la no repetición de los mismos, aunque no muy optimizadas para hacer consultas, y finalmente las multidimensionales son muy útiles para análisis estadístico de datos históricos de volúmenes inmensos de datos. Aparte del tipo de datos existe otro tipo de clasificaciones ya no según la estructura en que se almacenan los datos sino de acuerdo a la naturaleza de los datos.

### 1.3. Clasificación de bases de datos.

Según la naturaleza de los datos que se almacenan en la base de datos se pueden clasificar. La siguiente tabla muestra una compilación de estas clasificaciones.

| Criterio de clasificación           | Clasificación            | Descripción                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-------------------------------------|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Según la variabilidad de los datos. | Bases de datos estáticas | <ul style="list-style-type: none"> <li>Son bases de datos, cuyos datos son históricos, es decir ya no se pueden modificar, se usan comúnmente para estudiar el comportamiento de los datos a través del tiempo.</li> <li>Generalmente a este tipo de bases de datos se les llama bodega de datos.</li> <li>Muchas de estas bases de datos de almacenan en cubos para su análisis se las conoce como OLAP es el acrónimo en inglés de procesamiento analítico en línea (On-Line Analytical Processing).</li> </ul> |
|                                     | Bases de datos dinámicas | <ul style="list-style-type: none"> <li>Estos datos se almacenan y pueden ser modificados, agregados, borrados y consultados en cualquier momento, por ejemplo: un sistema de facturación o nuestro sistema Sofiaplus.</li> </ul>                                                                                                                                                                                                                                                                                  |



|                    |                             |                                                                                                                                                                                                                                                                                                                                                                                                                   |
|--------------------|-----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                    |                             | <ul style="list-style-type: none"><li>A estos sistemas se les denomina sistemas transaccionales, porque cada operación de guardar, borrar o editar se configura como una transacción. OLTP es la sigla en inglés de Procesamiento de Transacciones en Línea (<i>OnLine Transaction Processing</i>).</li></ul>                                                                                                     |
| Según el contenido | Bases de datos documentales | <ul style="list-style-type: none"><li>Permiten la indexación a texto completo, y en líneas generales realizar búsquedas más potentes.</li></ul>                                                                                                                                                                                                                                                                   |
|                    | Base de datos deductivos    | <ul style="list-style-type: none"><li>Un sistema de base de datos deductivos, es un sistema de base de datos, pero con la diferencia de que permite hacer deducciones a través de inferencias. Se basa principalmente en reglas y hechos que son almacenados en la base de datos. También las bases de datos deductivas son llamadas base de datos lógica, a raíz de que se basan en lógica matemática.</li></ul> |

Tabla 3. Clasificación de bases de datos.

Comentado [OG8]: Tabla. Leyenda: Clasificación de bases de datos.

Pueden existir otras clasificaciones generalmente más orientadas al ámbito de uso, sin embargo, se presentan las más comunes en el ejercicio del desarrollo de *software*. La base de datos relacionales es de obligatorio dominio en cualquiera de los casos, ya que los conceptos de esta son reutilizables en casi todos los otros tipos.

#### 1.4. Sistema de gestión de bases de datos

Un sistema de gestión de la base de datos es un programa de computadora que permite definir, crear y mantener los datos de una base de datos, controlando el acceso.

Los servicios SGDB ofrece y permite los siguientes servicios:

- Permitir la definición de la base de datos usando un lenguaje de definición de datos.
- Permitir la inserción, actualización, eliminación y consulta de datos usando un lenguaje de manejo de datos.
- Proporciona un acceso controlado a la base de dato (con autenticación, roles niveles de acceso)
- Concurrencia (varios usuarios a la vez accediendo o manipulando los datos) y multitarea.
- Algunos SGDB permiten administrar le catálogo de datos.

Comentado [OG9]: Lista de color no ordenada.

Los SGBD son una herramienta muy útil. Sin embargo, se podría decir que los SGBD son un poco más complejos, ya que los usuarios ven más datos y sus relaciones de los que realmente se necesitan.

#### 2. Modelo entidad relación

Las bases de datos relaciones inicialmente no son fáciles de modelar, por lo que se requiere de cierto nivel de abstracción, es decir se va ocultando la complejidad del diseño de la base de datos, primero analizando el requerimiento desde un punto de vista conceptual, entendido como las necesidades de la empresa o sistema de información, luego se va analizando des de un punto de vista de la lógica de los datos y finalmente se analiza físicamente como se almacenaran los datos, para ello aquí algunos conceptos.

##### A. ENTIDAD

El modelo de datos entidad-relación (E-R) está basado en una percepción del mundo real que consta de una colección de objetos básicos, llamados entidades, y de relaciones entre estos objetos. Una entidad es una «cosa»



u «objeto» en el mundo real que es distinguible de otros objetos. Por ejemplo, cada **persona** es una entidad, y las **cuentas** bancarias pueden ser consideradas entidades (Henry, 2002). Se representa con un rectángulo.

#### B. ATRIBUTOS

Las entidades están descritas por un conjunto de tributos por ejemplo una persona puede tener estos atributos: **nombres**, **apellidos**, **edad**, documento de identidad (**identificación**); o la cuenta bancaria puede tener los siguientes atributos: **número de cuenta**, **saldo**, **fecha de creación**. Se representa con una elipse.

#### C. RELACIONES

Una relación es una asociación entre varias entidades. Por ejemplo, una relación **titular** asocia un cliente con cada cuenta que tiene (de la que es titular). Se verá más claro un diagrama, pero antes de verlo considere los siguientes elementos para elaborar un diagrama. Se presentante con un rombo.

- o Rectángulos, que representan conjuntos de entidades.
- o Elipses, que representan atributos.
- o Rombo, que representan relaciones entre conjuntos de entidades.
- o Líneas, que unen los atributos con los conjuntos de entidades y los conjuntos de entidades con las relaciones.

Comentado [OG10]: Listado no ordenado.

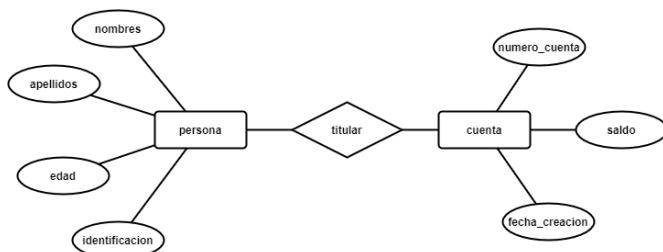


Ilustración 4. Entidad relación persona cuenta bancaria.

Fuente (<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiOdAk2c5SMXfb3o/view?usp=sharing> Page 1)

Para una base de datos de un sistema bancario consistente en clientes (es decir personas) y cuentas bancarias que tienen esos clientes. En la Ilustración 4 se muestra el diagrama Entidad Relación correspondiente. En él se indica que hay dos entidades que son **persona** y **cuenta**, con los atributos descritos para cada una anteriormente. El diagrama también representa la relación **titular** entre persona y cuenta.

Tenga en cuenta que el modelo relacional convierte las entidades en tablas por lo tanto se puede poner como ejemplo que las entidades personas se convierta en tablas como muestra la siguiente ilustración.

Comentado [OG11]: En el archivo online se encuentran relacionadas las ilustraciones que expone el experto. En el mismo se especifica la hoja donde se encuentra para que pueda ser digitalizada si se requiere.



| identificacion | nombres      | apellidos     | edad |
|----------------|--------------|---------------|------|
| 8.127.346      | Ana lis      | Mendez        | 34   |
| 212.394.762    | Jose Omar    | Ciano         | 28   |
| 545.232.546    | Luis Dario   | Marquez       | 21   |
| 125.446.732    | Armando      | Bronca Segura | 35   |
| 33.565.236.787 | Vistor José  | Diento Paz    | 24   |
| 2.343.554.768  | Jose Nicolás | Timado Ruiz   | 19   |
| 45.456.778     | Andrea       | Martinez      | 36   |

a) Tabla persona

| numero_cuenta | saldo            | fecha_creacion |
|---------------|------------------|----------------|
| 19.283.746    | \$ 2.589.954,00  | 25/08/1998     |
| 67.789.901    | \$ 23.465.212,00 | 26/08/2001     |
| 32.112.312    | \$ 2.343.242,00  | 27/08/2004     |
| 41.928.374    | \$ 2.343.423,00  | 31/08/2013     |
| 38.342.657    | \$ 8.965.211,00  | 30/08/2010     |
| 32.443.171    | \$ 11.172.594,33 | 22/06/2001     |
| 34.920.608    | \$ 14.483.578,83 | 01/09/2016     |
| 17.980.455    | \$ 17.794.563,33 | 03/09/2019     |
| 26.754.825    | \$ 21.105.547,83 | 12/04/2022     |

b) Tabla cuenta

Ilustración 5. Tablas de una entidad relación.

## D. TUPLA

Las tablas de la ilustración 5 muestran cada un conjunto de registros, la tabla persona muestra 6 registros y la tabla cuenta muestra 9 de ellos, cada uno de estos registros representa un objeto, es decir, de la tabla persona hay 6 (objetos) o personas a la agrupación de los datos de cada registro se le llama TUPLA. En términos simples una TUPLA de una base datos es el conjunto de todos los atributos (columnas) de una fila.

### 2.1. Relaciones entre entidades.

Como se observa en la Ilustraciones 4 y 5 una entidad se puede representar con una tabla, sin embargo, la relación **titular** aún no se sabe cómo representarla, para ello se debe definir antes algunas propiedades de los atributos antes.

**Principio de unicidad:** en una tabla, nunca deben existir TUPLAS repetidas, es decir, no debe haber más de una fila con exactamente los mismos valores en cada columna que otra fila de la misma tabla. En otras palabras cada fila debe diferir al menos en un elemento (columna) con relación a las demás.

En una base de datos relacional, las relaciones permiten evitar datos redundantes. En el ejemplo una persona pudiera tener varias cuentas bancarias, pero no es deseable que, por cada una de las cuentas bancarias asociadas a la misma persona, se deban repetir lo datos de la misma persona. Sino más bien que se debe encontrar un mecanismo para relacionar la información de las dos tablas. Para lo cual se introducen algunos conceptos útiles.

#### • Claves

Dado que en una tabla no hay filas repetidas, se pueden distinguir unas de otras, es decir, se pueden identificar de manera única (por el principio de unidad). Por lo tanto, la forma de identificarlas inequívocamente es según los valores de los atributos. De esta forma se conoce como claves a que los atributos que sirven para identificar un registro de forma inequívoca.

#### Ejemplo:

En la tabla persona, es sabido por todos que no existe dos personas con el mismo número de documento de identidad, por lo tanto, se puede emplear el atributo **identificación** como calve de la tabla persona. De igual forma no debe existir dos cuentas bancarias con el mismo número de cuenta, entonces, el atributo **numero\_cuenta** se puede emplear como clave de la tabla cuenta. Esto se lo puede presentar

**Comentado [OG12]:** Las tablas reales se encuentran en los anexos el documento denominado "Datos ejemplo" y la hoja Ejemplo 1

**Comentado [OG13]:** Hacer uso de tabs o pestañas para que el aprendiz pueda ir verificando los conceptos:  
A. ENTIDAD  
B. ATRIBUTOS  
C. RELACIONES  
D. TUPLA

**Comentado [OG14]:** Cajón de texto color.



en un diagrama de entidad relación cuando a los atributos que componen una clave les marcamos subrayado el nombre del atributo como se muestra en la siguiente ilustración.

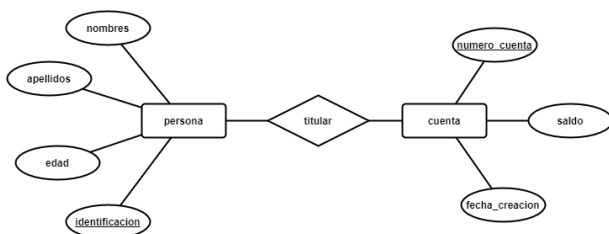


Ilustración 6. Claves de las entidades persona y cuenta

Fuente (<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiOdAk2c5SMXfB3o/view?usp=sharing> Page 2)

Notese como los atributos **identificación** y **numero\_cuenta** tiene subrayado la palabra identificadora del atributo, representado de esta forma que ese atributo es la clave (que hace que se cumpla el criterio de unidad) de su entidad.

- **Claves cantidadata, superclase y clave primaria.**

Suponga el caso en que el banco ahora es internacional, que es un banco moderno y solo diseñado para transacciones vía internet o virtuales, por lo tanto, el correo del cliente (persona) es necesario y obligatorio.

En este caso ocurren un par de asuntos a analizar: al ser internacional el banco, puede existir un cliente en un país con el mismo número de identificación de otro cliente en otro país, por lo tanto, el atributo **identificación** ya no puede ser una clave de la entidad, sin embargo es importante y obligatorio saber a qué país corresponde cada cliente, el conjunto de datos **identificación** y **país** ambos si cumplen el criterio de unidad, porque no existen dos personas en el mismo país con el mismo número de identificación.

- **Clave compuesta**

Si se escogiera el conjunto de atributos **identificación** y **país** como clave de la entidad, se está haciendo una clave compuesta porque es un conjunto de más de un atributo el que se usa para este fin.

El otro asunto a considerar es que, se puede emplear el **correo** electrónico como clave ya que no existen dos usuarios con el mismo correo electrónico, esto es válido si se definió como regla del banco que cada cliente tenga su **correo** electrónico diferente de los demás, estos dos casos se representan en la siguiente ilustración.

**Comentado [OG15]:** En el archivo online se encuentran relacionadas las ilustraciones que expone el experto. En el mismo se especifica la hoja donde se encuentra para que pueda ser digitalizada si se requiere.



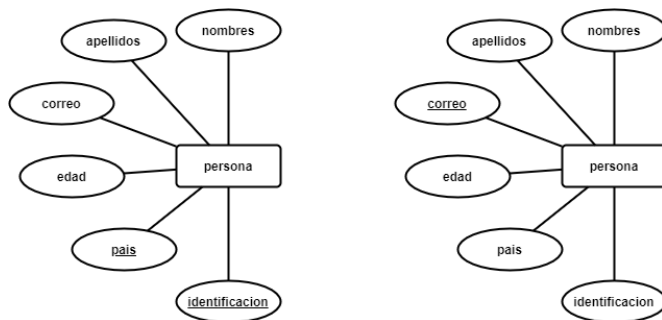


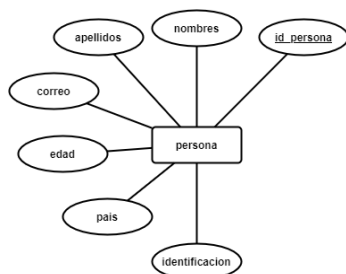
Ilustración 7. Claves Candidatas

Fuente (<https://drive.google.com/file/d/1W-ggJPezRs7noNMYiiOdAk2c5SMXfB3o/view?usp=sharing> Page 3)

- **Clave candidata**

Los dos caminos propuestos en la ilustración 7 son válidos, y se denominan claves candidatas porque con ambas se puede lograr el principio de unicidad, y es potestad de diseñador o analista de sistemas determinar cuál de las dos usa.

Sin embargo, existe otro tipo de solución y es asignar le un atributo identificador, a la entidad persona que se encargue exclusivamente de identificar inequívocamente cada tupla o fila, a este atributo le podemos llamar **id\_persona** que sería un número que el programa o software para el que estamos desarrollando la base le asignara a cada cliente cada vez que se registre por primera vez los datos de una persona (generalmente una secuencia). De esta forma tendríamos 3 claves candidatas, la ilustración 8 muestra cómo sería la entidad y la tabla que representa.



Entidad persona

**Comentado [OG16]:** En el archivo online se encuentran relacionadas las ilustraciones que expone el experto. En el mismo se especifica la hoja donde se encuentra para que pueda ser digitalizada si se requiere.



| id_persona | pais     | identificacion | correo           | nombres      | apellidos     | edad |
|------------|----------|----------------|------------------|--------------|---------------|------|
| 1          | colombia | 8.127.346      | ana@rmail.com    | Ana lis      | Mendez        | 34   |
| 2          | colombia | 212.394.762    | jose@pmail.com   | Jose Omar    | Ciano         | 28   |
| 3          | colombia | 545.232.546    | luis@jmail.com   | Luis Dario   | Marquez       | 21   |
| 4          | colombia | 125.446.732    | arm@hmail.com    | Armando      | Bronca Segura | 35   |
| 5          | colombia | 33.565.236.787 | victor@umail.com | Vistor José  | Diento Paz    | 24   |
| 6          | colombia | 2.343.554.768  | josen@pmail.com  | Jose Nicolás | Timado Ruiz   | 19   |
| 7          | perú     | 45.456.778     | andrea@email.com | Andrea       | Martinez      | 36   |

Tabla persona

Ilustración 8. Claves Candidatas secuencia

Este último enfoque es el más empleado y recomendado por los diseñadores o analistas del software, reduce significativamente la complejidad del diseño y facilita la relación con otra tablas o entidades. Por lo tanto, se usará cada vez que se tenga oportunidad de hacerlo.

#### • Super clave

Es un conjunto de uno o más atributos que, tomados juntos, permiten identificar de forma única una entidad en el conjunto de entidades. Por ejemplo, el atributo **id\_persona** del conjunto de entidades persona es suficiente para distinguir una persona cliente del banco de las otras. Así, **id\_pesona** es una superclave. Análogamente, la combinación de **identificación** y **país** es una superclave del conjunto de entidades persona. El atributo **correo** de persona es otra superclave, pero el atributo **nombres** no lo es porque varias personas podrían tener el mismo nombre.

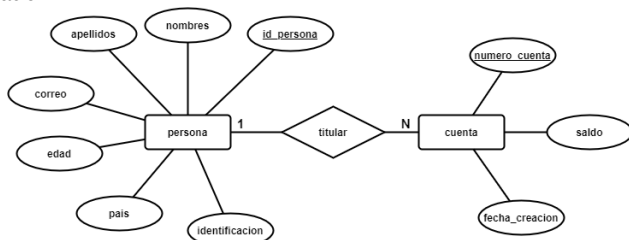
#### • Clave primaria

Se empleará el término clave primaria para nombrar una clave candidata que es elegida por el diseñador o analista de la base de datos como clave principal para identificar las tuplas dentro de una entidad. Una clave (primaria, candidata y superclave) es una propiedad del conjunto de entidades, más que de las entidades individuales. Entre dos tuplas individuales en el conjunto no pueden tener el mismo valor en sus atributos clave primaria al mismo tiempo. La designación de una clave primaria es en sí una restricción que modela una característica del mundo real.

### 2.2. Relaciones de uno a muchos.

Una relación de uno a varios o de uno a muchos es el tipo de relación más empleada. En este tipo de relación, una fila de la tabla A puede tener muchas filas coincidentes en la tabla B. Pero una fila de la tabla B solo puede tener una fila coincidente en la tabla A. Por ejemplo, las tablas **pesona** y **cuenta** tienen una relación de uno a muchos. Es decir, cada persona puede tener más de una cuenta. Pero cada título proviene de una sola editorial.

En un diagrama de entidad relación esto se representa con lo que se denomina multiplicidad como se muestra en la siguiente ilustración:



**Comentado [OG17]:** En caso de requerir los editables del esquema y tabla utilizar recursos anexos. Dato ejemplo 1 y <https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiOdAk2c5SMXfB3o/view?usp=sharing> Page 4



Ilustración 9. Relaciones de uno a muchos

Fuente (<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiOdAk2c5SMXfB3o/view?usp=sharing> Page 5)

A esta representación se le denomina multiplicidad **1:N** y se lee de la siguiente forma: **UNA persona es titular de N cuentas** bancarias, de esta forma una persona puede tener **N** (que puede ser cero, uno o más de una) cuentas bancarias. Se puede leer en forma inversa que el resultado es el mismo: muchas o varias cuentas bancarias pueden tener un mismo titular.

Cuando una relación es de uno a muchos y se mapea a un modelo de base de datos relacional, la relación **titular** (rombo) se termina convirtiendo una columna de la tabla de los muchos.

| id_persona | pais     | identificacion | correo           | nombres      | apellidos     | edad | numero_cuenta | titular | saldo            | fecha_creacion |
|------------|----------|----------------|------------------|--------------|---------------|------|---------------|---------|------------------|----------------|
| 1          | colombia | 8.127.346      | ana@rmail.com    | Ana lis      | Mendez        | 34   | 19.283.746    | 7       | \$ 2.589.954,00  | 25/08/1998     |
| 2          | colombia | 212.394.762    | jose@gmail.com   | Jose Omar    | Ciano         | 28   | 67.789.901    | 1       | \$ 23.465.212,00 | 26/08/2001     |
| 3          | colombia | 545.232.546    | luis@gmail.com   | Luis Darío   | Marquez       | 21   | 32.112.312    | 5       | \$ 2.343.242,00  | 27/08/2004     |
| 4          | colombia | 125.446.732    | arm@gmail.com    | Armando      | Bronea Segura | 35   | 41.928.374    | 2       | \$ 2.343.423,00  | 31/08/2013     |
| 5          | colombia | 33.565.236.787 | victor@gmail.com | Vistor José  | Diento Paz    | 24   | 38.342.657    | 6       | \$ 8.965.211,00  | 30/08/2010     |
| 6          | colombia | 2.343.554.768  | josen@gmail.com  | Jose Nicolás | Timado Ruiz   | 19   | 32.443.171    | 1       | \$ 11.172.594,33 | 22/06/2001     |
| 7          | perú     | 45.456.778     | andrea@email.com | Andrea       | Martinez      | 36   | 34.920.608    | 5       | \$ 14.483.578,83 | 01/09/2016     |
|            |          |                |                  |              |               |      | 17.980.455    | 3       | \$ 17.794.563,33 | 03/09/2019     |
|            |          |                |                  |              |               |      | 26.754.825    | 4       | \$ 21.105.547,83 | 12/04/2022     |

a) Tabla persona

b) Tabla cuenta

Ilustración 10. Tablas de un modelo relacional 1:N

Como se puede ver en la Ilustración 10, Ana Lis Mendez, es titular de dos cuentas bancarias (67.789.901 y 32.443.171), y todos los datos de la columna titular en la tabla cuenta se conoce como **LLAVE FORANEA** de la tabla persona; de esta forma podemos concluir.

#### REGLA DE MAPEO 1:N de Modelo entidad-relación a modelo relacional:

Una relación de uno a mucho (1:N) se transforma en una columna en la tabla que tiene la multiplicidad de los muchos (cuenta). Y todos los datos de esta columna existen como llave primaria en la tabla que tiene la multiplicidad uno (tabla persona).

Existe otro tipo de diagrama se usted aprenderá y que en realidad más empleado a la hora de diseñar una base de datos, este diagrama se llama Diagrama Relacional, donde las entidades se presentan como tablas de una vez y las relaciones son más claras. Miremos la representación del diagrama entidad relación de la ilustración a un diagrama relacional.

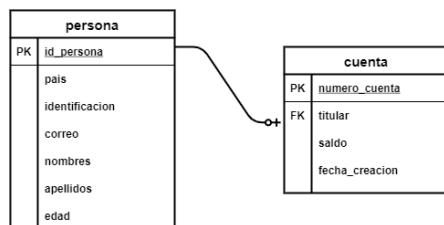


Ilustración 11. Diagrama relacional 1:N.

Fuente (<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiOdAk2c5SMXfB3o/view?usp=sharing> Page 6 -2)

En la ilustración 11 se puede ver el nombre de las entidades (persona y cuenta), los atributos haciendo especial énfasis en los que representan la llave primaria con el indicador **PK** del inglés (Primary Key), y también vemos la

**Comentado [OG18]:** En el archivo online se encuentran relacionadas las ilustraciones que expone el experto. En el mismo se especifica la hoja donde se encuentra para que pueda ser digitalizada si se requiere.

**Comentado [OG19]:** Las tablas reales se encuentran en los anexos el documento denominado "Datos ejemplo" y la hoja Ejemplo 1

**Comentado [OG20]:** En el archivo online se encuentran relacionadas las ilustraciones que expone el experto. En el mismo se especifica la hoja donde se encuentra para que pueda ser digitalizada si se requiere.



relación titular (columna de la tabla), indicando que es una clave o llave foránea de la tabla persona lo indica con el indicador **FK** del inglés (**Foreign Key**).

Algunos diagramas relacionales agregan metadatos para hacer más claro el modelo al diseñador, la naturaleza de los datos del modelo.

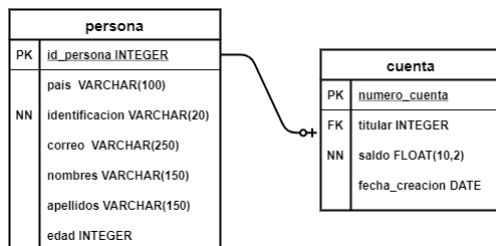


Ilustración 12. Diagrama relacional 1:N con metadatos.

Fuente (<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiOdAk2c5SMXfB3o/view?usp=sharing> Page 6 -2)

**Conclusión:** la relación de uno a muchos se termina convirtiendo en una columna en la tabla que tiene la multiplicidad d ellos muchos.

**Comentado [OG21]:** En el archivo online se encuentran relacionadas las ilustraciones que expone el experto. En el mismo se especifica la hoja donde se encuentra para que pueda ser digitalizada si se requiere.

**Comentado [OG22]:** Cajón texto color.

### 2.3. Relaciones de un muchos a muchos.

En una relación de muchos a muchos, una fila de la tabla A puede tener relación con muchas filas en la tabla B y también en sentido contrario una fila de la tabla B con respecto de varias filas de la tabla A. Esta relación se crea definiendo una tercera tabla denominada tabla de relación. Esta nueva tabla tiene una clave principal de la tabla compuesta de las claves foráneas de la tabla A y de la B.

Ejemplo:

Supongamos que necesitamos almacenar la información de unas referencias bibliográficas o publicaciones, de libros, tesis, artículos que escriben los estudiantes de una universidad, y como se sabe un libro, tesis, artículo o cualquier publicación puede ser elaborada por más de una persona, y a su vez, una persona (docente, estudiante o investigador) puede tener más de una publicación de su autoría.

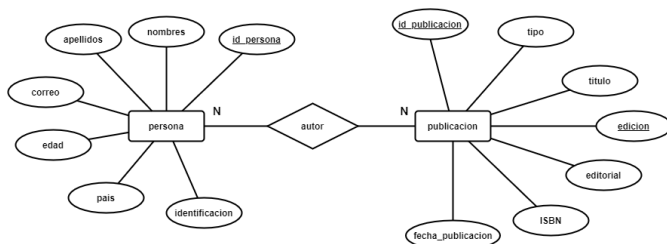


Ilustración 13. Diagrama de entidad de publicaciones bibliográficas.

Fuente (<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiOdAk2c5SMXfB3o/view?usp=sharing> Page 7 -2)

Como se puede ver en la ilustración 13, la entidad publicación tiene una llave primaria llamada identificación, y existe una relaciona llamada **autor**, y representa que unas persona pueden ser autor de una o muchas

**Comentado [OG23]:** En el archivo online se encuentran relacionadas las ilustraciones que expone el experto. En el mismo se especifica la hoja donde se encuentra para que pueda ser digitalizada si se requiere.



publicaciones, y que una publicación puede ser de autoría de una o muchas personas, la multiplicidad en este caso es de muchos a muchos y se representa como N:N.

#### REGLA DE MAPEO N:N de Modelo entidad-relación a modelo relacional

En una relación de uno a mucho (N:N) se transforma en una tabla cuya llave primaria está compuesta por las llaves primarias de las otras dos tablas.

Comentado [OG24]: Cajón texto color.

Como se puede ver en la ilustración 14, la tabla c) autor, nos dice que la persona Ana Lis, es autora de los libros de título Grandes hazañas y Pellentesque, también que el libro Pellentesque es escrito además de Ana Lis por Luis Dario y Jose Nicolas. Lo anterior haciendo referencia a las llaves primarias de las tablas a) **persona** y b) **publicacion**.

| id_persona | pais     | identificacion | correo           | nombres      | apellidos     | edad |
|------------|----------|----------------|------------------|--------------|---------------|------|
| 1          | colombia | 8.127.346      | ana@gmail.com    | Ana lis      | Mendez        | 34   |
| 2          | colombia | 212.394.762    | jose@gmail.com   | Jose Omar    | Ciano         | 28   |
| 3          | colombia | 545.232.546    | luis@gmail.com   | Luis Dario   | Marquez       | 21   |
| 4          | colombia | 125.446.732    | arm@gmail.com    | Armando      | Bronca Segura | 35   |
| 5          | colombia | 33.585.236.787 | victor@gmail.com | Vistor José  | Diento Paz    | 24   |
| 6          | colombia | 2.343.554.768  | josen@gmail.com  | Jose Nicolás | Timado Ruiz   | 19   |
| 7          | perú     | 45.456.778     | andrea@gmail.com | Andrea       | Martinez      | 36   |

a) Tabla persona

| id_publicacion | tipo     | titulo              | editorial | edicion | fecha_publicacion |
|----------------|----------|---------------------|-----------|---------|-------------------|
| 1              | libro    | Grandes Hazañas     | Universit | 6ta     | 25/08/1998        |
| 2              | tesis    | Mediciones en me    | Denver    | -       | 26/08/2001        |
| 3              | libro    | Consectetur Sem     | ISFR      | 1ra     | 27/08/2004        |
| 4              | artículo | Consectetur         | Sodales   | -       | 31/08/2013        |
| 5              | libro    | Pellentesque        | Conubia   | 3ra     | 30/08/2010        |
| 6              | revista  | Praesent            | Nascetur  | -       | 22/06/2001        |
| 7              | revista  | Fermentum           | aptent    | -       | 01/09/2016        |
| 8              | libro    | El ruido y la furia | Vulputat  | 1a      | 03/09/2019        |
| 9              | libro    | Tempus mauris       | Efficitur | 5ta     | 12/04/2022        |

b) Tabla publicacion

| id_persona | id_publicacion |
|------------|----------------|
| 1          | 1              |
| 1          | 5              |
| 6          | 5              |
| 3          | 5              |
| 6          | 2              |

c) Tabla autor

Ilustración 14. Tablas de relación N:N.

Comentado [OG25]: La ilustración se puede obtener del archivo en anexos "Datos ejemplo" la hoja "Ejemplo (2)"

La representación de la relación N:N con un diagrama relacional podemos se muestra en la ilustración 15. Donde se usan los prefijos **FK1** y **FK2**, para indicar Llave foránea uno y dos, para las entidades persona y publicación 2 respectivamente. También se puede ver el prefijo **UK** (Unique Key) en la tabla publicación, en la columna ISBN, significa en toda la tabla publicación no puede existir más de una fila con el mismo ISBN es decir, nunca ninguna publicación tendrá el mismo ISBN que otra publicación

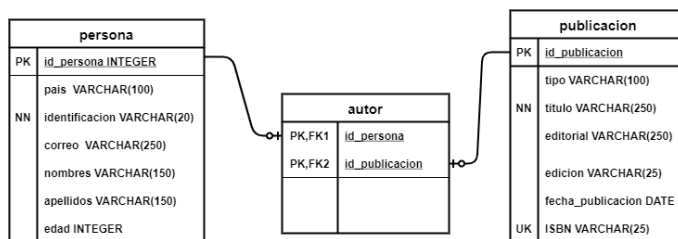


Ilustración 15. Diagrama relacional N:N.

Comentado [OG26]: El editable se puede obtener del Anexo "base de datos" Page-8

## 2.4. Relaciones de uno a uno

En una relación uno a uno, una fila de la tabla A solo puede tener una fila coincidente en la tabla B y viceversa. Se crea una relación uno a uno si ambas columnas relacionadas son claves principales o tienen restricciones

<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiQdAk2c5SMXfb3o/view?usp=sharing>



únicas. Este tipo de relación no es común porque la mayoría de la información que está relacionada de esta manera estaría en una tabla. Puede usar una relación uno a uno por alguno de estos motivos:

- Dividir una tabla con muchas columnas.
- Aísle parte de una tabla por motivos de seguridad.
- Almacenar datos de corta duración y que podrían eliminarse fácilmente borrando la tabla periódicamente.
- Almacenar información que solo se aplique a un subconjunto de la tabla principal.

Comentado [OG27]: Listado no ordenado color.

Viviendo al ejemplo del banco, supongamos esta vez, la empresa necesita información sobre sus empleados, que, a la vez, pueden ser clientes de banco.

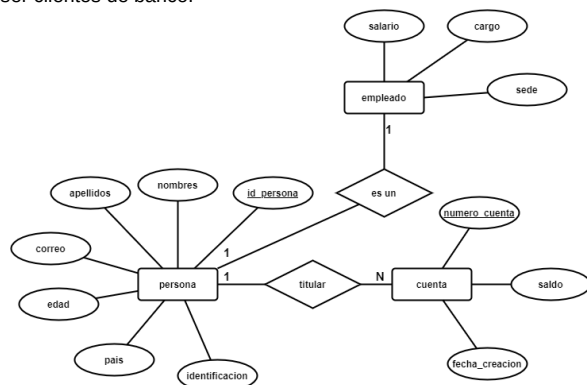


Ilustración 16. Diagrama entidad relación persona-empleado

Comentado [OG28]: El editable se puede obtener del Anexo "base de datos" Page-9

#### REGLA DE MAPEO 1:1 de Modelo entidad-relación a modelo relacional

En el momento de convertir la relaciono con multiplicidad 1:1 a tablas la relación uno a uno se convierte en una columna de la tabla empleado con restricción de unicidad (Unique Key, o Primary Key) como se muestra en el siguiente diagrama relacional.

<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiOdAk2c5SMXfB3o/view?usp=sharing>

Comentado [OG29]: Cajón texto color.

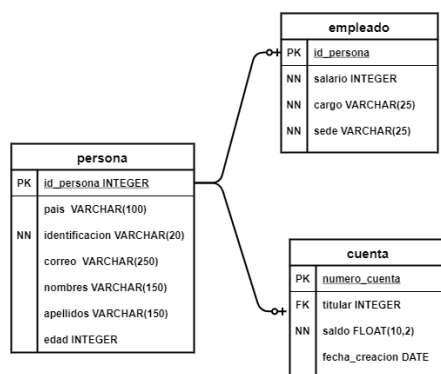


Ilustración 16. Diagrama relacional 1:1.

Comentado [OG30]: El editable se puede obtener del Anexo "base de datos" Page-6

<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiOdAk2c5SMXfB3o/view?usp=sharing>



Como se puede inferir de la ilustración 16, se crea una tabla donde se relacionan los datos de las personas que son empleados del banco, a través de una columna que se llama **id\_persona**, pero es común entre algunos diseñadores ponerle el nombre que lo relaciones con la tabla a la que perteneces es decir **id\_empleado**, pero esto queda al criterio del diseñador.

### 3. Normalización

La normalización es el procedimiento mediante el cual se aplican las reglas de mapeo o conversión de un modelo entidad-relación a un modelo relacional, es decir a tablas y relaciones, los principios rectores de este proceso son: No redundancia de datos (que no se repitan los datos) y la dependencia de los datos dependencia incoherente (una separación lógica de datos en tablas).

Los datos redundantes desperdician espacio en disco y crean problemas de mantenimiento. Por ejemplo, en la ilustración 16 se ve que los datos comunes de los clientes y los bancos están en la tabla persona, mientras que los datos que pertenecen exclusivamente a los empleados está en otra tabla que se relaciona con la de personas.

¿Qué es una "dependencia incoherente"? Aunque resulta intuitivo para un usuario buscar en la tabla de persona el correo de un cliente en particular, es posible que no tenga sentido buscar en esa columna, el salario del empleado que se relaciona con esos datos de la tabla persona. El salario del empleado está relacionado con la tabla empleado o depende de él. Las dependencias incoherentes pueden dificultar el acceso a los datos, ya que la ruta para encontrar los datos puede faltar o estar dañada.

Existen unas reglas básicas de normalización las más comunes son las 3 formas normales se listan a continuación y se irán desarrollando con un ejemplo más complejo.

| Forma Normal                | Principios                                                                                                                                                                                                                                                                 |
|-----------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Primera forma normal</b> | <ul style="list-style-type: none"><li>• Eliminar grupos de repetición en tablas individuales.</li><li>• Cree una tabla independiente para cada conjunto de datos relacionados.</li><li>• Identifique cada conjunto de datos relacionados con una clave primaria.</li></ul> |
| <b>Segunda forma normal</b> | <ul style="list-style-type: none"><li>• Crear tablas independientes para conjuntos de valores que se aplican a varios registros.</li><li>• Relacione estas tablas con una clave foránea.</li></ul>                                                                         |
| <b>Tercera forma normal</b> | <ul style="list-style-type: none"><li>• Eliminar los campos que no dependen de la clave.</li></ul>                                                                                                                                                                         |

Tabla 4. Formas normales.

Hay una cuarta forma normal, también llamada forma normal de *Boyce Codd* (BCNF), y una la quinta forma normal, pero rara vez se consideran en un diseño práctico, solo agregan complejidad al sistema y no aportan un valor funcional que importe a la solución del problema por ello no será tratada.

#### 3.1. Dependencias funcionales.

Una dependencia funcional es un tipo de restricción que termina construyendo una generalización del concepto de clave, como se estudió en el modelo E-R y el modelo relacional. Pero no es tan fácil localizar las dependencias, ya que necesitan de un análisis de los atributos (columnas) o, con más precisión, de las interrelaciones entre atributos, y frecuentemente la intuición no es suficiente a la hora de encontrar y clasificar todas las dependencias.



Aunque existe una teoría matemática para realizar este análisis, nos puede ayudar un ejemplo sencillo nos puede enseñar como analizar estas dependencias de manera intuitiva como se verá a continuación.

Ejemplo:

Un cliente le pide que desarrolle un software para llevar el inventario de una ferretería, los productos, los proveedores, el precio que cada proveedor nos vende cada producto, para lo cual nos da la siguiente tabla, y debemos identificar las dependencias funcionales para poder realizar nuestro modelo relacional.

| nit_proveedor | correo         | nombres        | telefono  | codigo_producto | producto            | precio       |
|---------------|----------------|----------------|-----------|-----------------|---------------------|--------------|
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | MS-028          | Masillas FFC        | \$ 25.890,00 |
| 212394762     | jose@pmail.com | Jose Omar      | 5475698   | PS-012          | Pistola de silicona | \$ 14.300,00 |
| 545232546     | luis@jmail.com | Luis Dario     | 3984558   | CT-003          | Cutter RDS          | \$ 8.700,00  |
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | LG-021          | Lija C20            | \$ 820,00    |
| 212394762     | jose@pmail.com | Jose Omar      | 5475698   | MS-028          | Masillas FFC        | \$ 26.000,00 |
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | LG-021          | Lija C20            | \$ 7.900,00  |
| 545232546     | luis@jmail.com | Luis Dario     | 36584114  | PS-012          | Pistola de silicona | \$ 12.990,00 |
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | LG-021          | Lija C20            | \$ 830,00    |
| 212394762     | jose@pmail.com | Jose Omar      | 5475698   | CT-003          | Cutter RDS          | \$ 9.200,00  |

Tabla 5. Proveedores, productos y precios.

Comentado [OG31]: La ilustración se puede obtener del archivo en anexos "Datos ejemplo" la hoja 6.

El principio de no repetición de datos dice que debe identificar todos los datos que se repiten y se puede en la siguiente figura, tenido en cuenta los colores de sombreado como se muestra, en la ilustración 17, note que el nit\_proveedor son iguales en todos registros que tienen el mismo valor de correo, nombres, y teléfono.

| nit_proveedor | correo         | nombres        | telefono  | codigo_producto | producto            | precio       |
|---------------|----------------|----------------|-----------|-----------------|---------------------|--------------|
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | MS-028          | Masillas FFC        | \$ 25.890,00 |
| 212394762     | jose@pmail.com | Jose Omar      | 5475698   | PS-012          | Pistola de silicona | \$ 14.300,00 |
| 545232546     | luis@jmail.com | Luis Dario     | 3984558   | CT-003          | Cutter RDS          | \$ 8.700,00  |
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | LG-021          | Lija C20            | \$ 820,00    |
| 212394762     | jose@pmail.com | Jose Omar      | 5475698   | MS-028          | Masillas FFC        | \$ 26.000,00 |
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | LG-021          | Lija C20            | \$ 7.900,00  |
| 545232546     | luis@jmail.com | Luis Dario     | 36584114  | PS-012          | Pistola de silicona | \$ 12.990,00 |
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | LG-021          | Lija C20            | \$ 830,00    |
| 212394762     | jose@pmail.com | Jose Omar      | 5475698   | CT-003          | Cutter RDS          | \$ 9.200,00  |

Ilustración 17. Dependencia de nit\_proveedor con datos del proveedor. (GIF 1 ANIMADO)

Comentado [OG32]: La ilustración corresponde al GIF 1. En caso de ajustar la imagen GIF se comparte el PPT generado para la creación de la animación.

Note también en la ilustración 18 como la columna código\_producto es el mismo para cada columna producto, pero difiere del precio que cada proveedor ofrece del mismo:





| nit_proveedor | correo         | nombres        | telefono  | codigo_producto | producto            | precio       |
|---------------|----------------|----------------|-----------|-----------------|---------------------|--------------|
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | MS-028          | Masillas FFC        | \$ 25.890,00 |
| 212394762     | jose@pmail.com | Jose Omar      | 5475698   | PS-012          | Pistola de silicona | \$ 14.300,00 |
| 545232546     | luis@jmail.com | Luis Dario     | 3984558   | CT-003          | Cutter RDS          | \$ 8.700,00  |
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | LG-021          | Lija C20            | \$ 820,00    |
| 212394762     | jose@pmail.com | Jose Omar      | 5475698   | MS-028          | Masillas FFC        | \$ 26.000,00 |
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | LG-021          | Lija C20            | \$ 7.900,00  |
| 545232546     | luis@jmail.com | Luis Dario     | 36584114  | PS-012          | Pistola de silicona | \$ 12.990,00 |
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | LG-021          | Lija C20            | \$ 830,00    |
| 212394762     | jose@pmail.com | Jose Omar      | 5475698   | CT-003          | Cutter RDS          | \$ 9.200,00  |

Ilustración 18. Dependencia de código\_producto\_ con datos del producto. (GIF 2 ANIMADO)

La dependencia funciona código\_producto no se cumple para precio del producto, por lo tanto, dice que a cada proveedor y producto debe existir un precio distinto como se ve la siguiente ilustración.

| nit_proveedor | correo         | nombres        | telefono  | codigo_producto | producto            | precio       |
|---------------|----------------|----------------|-----------|-----------------|---------------------|--------------|
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | MS-028          | Masillas FFC        | \$ 25.890,00 |
| 212394762     | jose@pmail.com | Jose Omar      | 5475698   | PS-012          | Pistola de silicona | \$ 14.300,00 |
| 545232546     | luis@jmail.com | Luis Dario     | 3984558   | CT-003          | Cutter RDS          | \$ 8.700,00  |
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | LG-021          | Lija C20            | \$ 820,00    |
| 212394762     | jose@pmail.com | Jose Omar      | 5475698   | MS-028          | Masillas FFC        | \$ 26.000,00 |
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | LG-021          | Lija C20            | \$ 7.900,00  |
| 545232546     | luis@jmail.com | Luis Dario     | 36584114  | PS-012          | Pistola de silicona | \$ 12.990,00 |
| 8127346       | ana@rmail.com  | Ana Lis Mendez | 398589854 | LG-021          | Lija C20            | \$ 830,00    |
| 212394762     | jose@pmail.com | Jose Omar      | 5475698   | CT-003          | Cutter RDS          | \$ 9.200,00  |

Ilustración 19. Dependencia de nit\_proveedor, código\_producto\_ con precio del producto. (GIF 3 ANIMADO)

En la siguiente tabla se observan las dependencias detectadas:

| Atributo                       | Es dependencia funcional de:                                                                  |
|--------------------------------|-----------------------------------------------------------------------------------------------|
| nit_proveedor                  | <ul style="list-style-type: none"><li>• correo</li><li>• nombres</li><li>• teléfono</li></ul> |
| codigo_producto                | <ul style="list-style-type: none"><li>• producto</li></ul>                                    |
| nit_proveedor, codigo_producto | <ul style="list-style-type: none"><li>• precio</li></ul>                                      |

Tabla 5. Dependencias funcionales

#### Definición formal DEPENDENCIA FUNCIONAL

Se dice que un atributo X de una relación «depende funcionalmente» de otro atributo o conjunto de atributos Y de la relación si a todo valor (o valores del conjunto) Y le corresponde siempre el mismo valor de X.

Las dependencias funcionales de la tabla 5 se pueden representar en modelo entidad relación, teniendo en cuenta que un proveedor, puede proveer muchos productos y a su vez un producto puede ser proveedor por más de un proveedor, y que por cada producto que es proveedor por un proveedor existe un precio.

**Comentado [OG33]:** La ilustración corresponde al GIF 2. En caso de ajustar la imagen GIF se comparte el PPT generado para la creación de la animación.

**Comentado [OG34]:** La ilustración corresponde al GIF 3. En caso de ajustar la imagen GIF se comparte el PPT generado para la creación de la animación.

**Comentado [OG35]:** Cajón texto color.

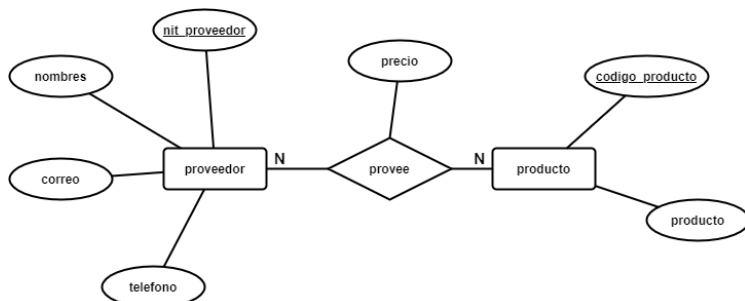


Ilustración 19. Diagrama entidad relación dependencia funcional.

Como se puede notar en la ilustración 19, la relación provee, cuando es mapeado a un diagrama relacional, se genera una tabla intermedia provee que debe tener un atributo adicional "precio".

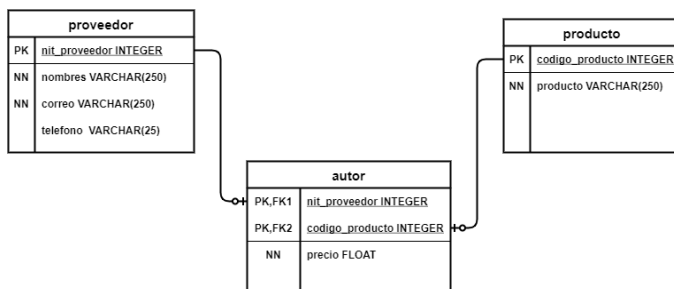


Ilustración 19. Diagrama relacional dependencia funcional.

### 3.2. Diseño relacional.

Hasta este punto se han examinado problemas concretos de las formas normales muy básicas y de la normalización. Ahora se estudiará el modo en que se encaja la normalización en proceso general de diseño de bases de datos se puede hacer de una de esas formas:

1. Convirtiendo un diagrama entidad relación a un diagrama relacional y de esa forma obtener las tablas.
2. Puede tener una tabla con todos los datos que se procesan en una aplicación e identificado las dependencias funcionales de manera que nos permite identificar las tablas y las llaves o claves primarias de cada una.
3. Se puede hacer un diseño ad hoc (para que cumpla el requerimiento) y se debe comprobar luego que satisface la forma normal deseada.

En el mundo practico y en la medida que se tiene más experticia las opciones 2 y 3 son las más empleadas.

**Comentado [OG36]:** El editable se puede obtener del Anexo "base de datos" Page-11

<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiQdAk2c5SMxfB3o/view?usp=sharing>

**Comentado [OG37]:** El editable se puede obtener del Anexo "base de datos" Page-12

<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiQdAk2c5SMxfB3o/view?usp=sharing>

**Comentado [OG38]:** Listado ordenado cuadro color.



Veamos un ejemplo completo de cómo abordar un problema de almacenamiento de datos y transformarlo en un modelo relacional, recogiendo los conceptos vistos hasta ahora, pasado de un diagrama entidad relación a un modelo relacional que es en todo caso el objetivo general de un diseño de base de datos:

#### Problema:

Se necesita de gestionar una biblioteca escolar, y nuestro cliente quiere tener ciertas herramientas a su disposición para controlar publicaciones tales como libros, revistas, folletos; también llevar un registro de los estudiantes y préstamos. Adicionalmente se necesita un control de los ejemplares de cada libro o publicación, su ubicación y su estado, registrar la retirada de libros por préstamo y restitución de ejemplares, para esto necesita información sobre editoriales datos como dirección, teléfono y nombre de la editorial.

De cada publicación se pueden tener varios ejemplares, por ejemplo: del libro “nacho lee” la biblioteca tiene 5 ejemplares, también de cada publicación se debe conocer, su autor o autores, el tema que trata la publicación (geografía, física, matemática, etc.), la editorial. Cada libro, tiene un título, un idioma, y formato (digital web, impreso, digital CD), el código ISBN y fecha de publicación.

Cada ejemplar tiene un código de barras, cuando un ejemplar ya está muy descargado se debe poder dar de baja es decir que ya se puede botar y no usarlo más.

Tanto el estudiante, como el auto de un libro tiene los siguientes datos: nombres, apellidos, correo. Adicional a un estudiante tiene los datos de documento de identidad, teléfono, dirección, grado escolar.

De cada préstamo interesa saber que ejemplar se prestó, fecha en la que se prestó, la fecha en que se devolvió y alguna nota o comentario.

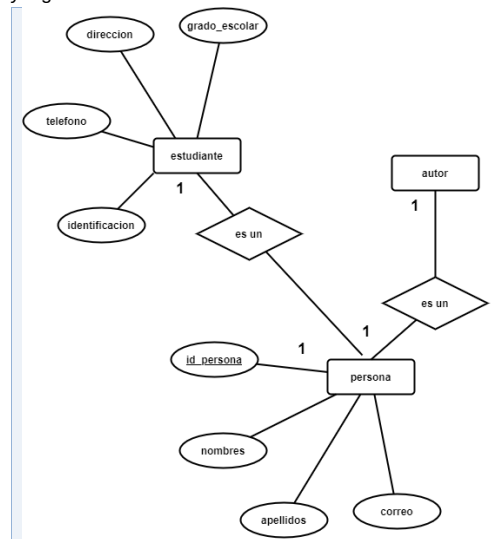


Ilustración 20. Análisis de relacionar autor

En este diagrama se presenta una posible solución analicémosla.

La relación entre persona y estudiante está muy bien corresponde al caso estudiado en las relaciones de uno a uno (entre persona y empleado del banco) de la ilustración 16.

El tema se pone complejo porque debemos analizar si aplica lo mismo para autor:

Considere que al mapear la estas entidades a diagrama relacional me resultan 3 tablas (estudiante, persona y autor), pero autor solo tendría una única columna y al no tener atributos adicionales relacionados con el autor es prácticamente lo mismo que tener la tabla persona, por lo tanto no es conveniente dejarlo de esta forma y más bien relacionar la publicación directamente con la entidad persona.

Estudie detenidamente el diagrama entidad relación, por varios minutos, asegúrese que comprende el porqué de cada elemento y su multiplicidad. Antes de continuar, realice a mano alzada diagrama el modelo relacional:

**Comentado [OG39]:** El editable se puede obtener del Anexo “base de datos” Page-15

<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiOdAk2c5SMXIB3o/view?usp=sharing>



Recuerde:

Una relación de muchos a muchos (N:N) se convierte en una tabla cuya llave primaria es computara por las llaves foráneas de las otras dos tablas, Una relación de uno a muchos (1:N) se convierte en una columna (llave foránea) de la tabla que tiene los muchos, y finalmente una relación de uno a uno (1:1) se convierte una columna en cualquier (llave foránea) de las dos tablas.

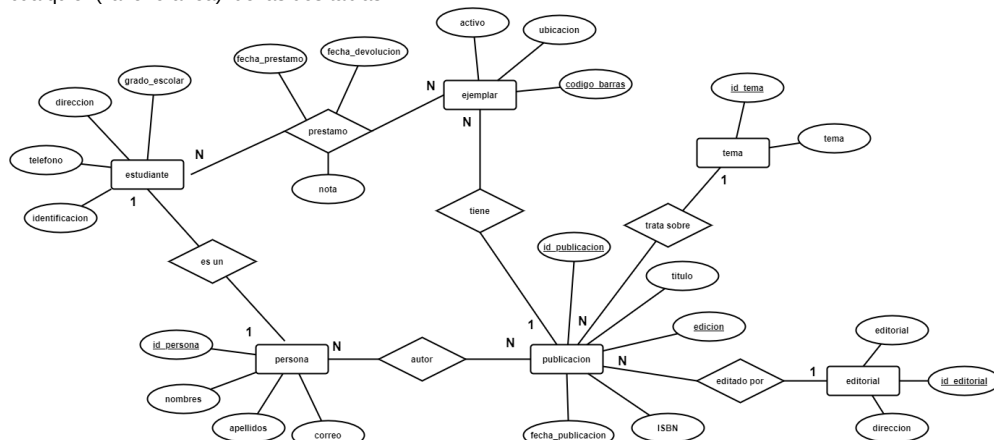
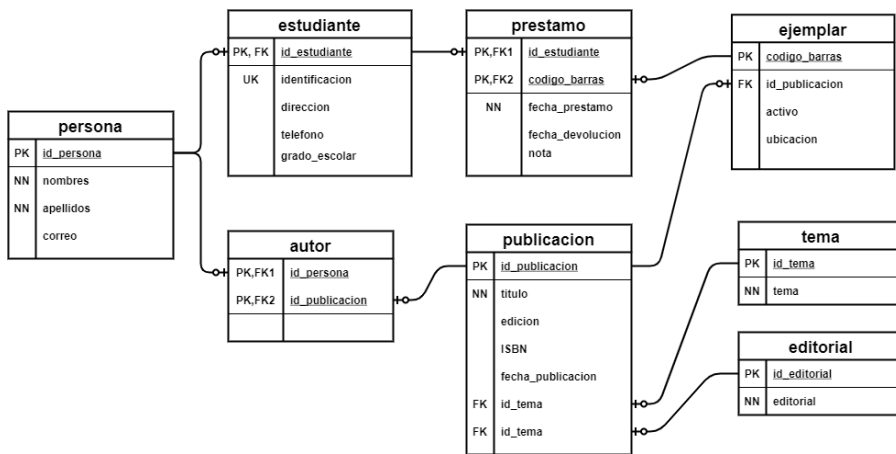


Ilustración 21. Modelo entidad relación biblioteca

En la siguiente ilustración se presenta el modelo en un diagrama relacional, este diagrama es importante ya que hace parte de uno de los subproductos que por petición del cliente muchas veces es el un entregable en un proyecto de software, no es así con el diagrama entidad relación ya que este tipo de diagrama requiere más formas y contiene menos tecnicismos.



Comentado [OG40]: El editable se puede obtener del Anexo "base de datos" Page-7

<https://drive.google.com/file/d/1W-qgJPezRs7noNMYjiOdAk2c5SMXfB3o/view?usp=sharing>



### Ilustración 22. Modelo relacional

Hasta ahora se ha visto cómo distribuir el almacenamiento de los datos en tablas, de manera que no permita la duplicidad de datos y procurando una estructura lógica, que presente la información del mundo real. Pero aún falta analizar el problema desde el punto de vista que permita garantizar que los datos que se van a insertar son correctos. Para lo cual es importante conocer las restricciones de integridad que puede tener una base de datos.

#### 3.3. Reglas de integridad.

Cuando se define un atributo si saber lo estamos escogiendo para él un dominio es decir un conjunto de valores que puede tener. Por ejemplo: en MySQL cuando escogemos un atributo de tipo entero (INT) los posibles valores que puede tener esa columna van desde -2147483648 hasta 2147483647, es decir valores positivo u negativos, de manera que si queremos solo valores positivos debemos definirlo como entero sin signo (UNSIGNED INT) por lo tanto los valores van desde 0 hasta 4294967295, vea un dato completo en el siguiente enlace

Tipos de enteros: <https://dev.mysql.com/doc/refman/8.0/en/integer-types.html>

La calidad de los datos garantiza que los datos almacenados en su base de datos cumplan con los estándares y requisitos de la organización. En otras palabras, asegura el mantenimiento de la integridad de los datos en una base de datos. Al hacerlo, aplica un conjunto de reglas a un conjunto de datos completo o específico y lo almacena en la base de datos de destino por lo consiguiente para garantizar la integridad se debe tener en cuenta:

- **Criterio de NULIDAD**

Cuando en una fila un atributo (columna) es desconocido, se dice que es nulo. Un nulo no representa el valor cero (0) ni una cadena vacía ("") ya que éstos son valores tienen un significado o valor. El nulo significa ausencia de información, bien porque al insertar la fila se desconocía el valor del atributo, o tal vez porque para dicha fila el atributo no tiene valor. Debido a que los nulos no son valores, deben tratarse de forma particular, lo que causa problemas de implementación. Actualmente casi todos los SGDB soportan valores nulos.

- **Integridad de entidad**

Ninguno de los atributos o columnas que componen una llave primaria debe ser NULO. Es decir, una clave prima es irreducible para identificar de modo único una fila, el que sea irreducible significa que ningún subconjunto de la clave primaria sirve para identificar inequívocamente la fila. Si se permitirá valor nulo en una columna de hace parte de la llave primaria, esta estaría contradiciendo la irreducibilidad de una llave primaria. Tenga en cuenta que esto no aplica para claves alternativas y solo para la clave primaria.

- **Integridad referencia.**

Analice el modelo de la ilustración 22. En particular la relación entre las tablas persona y estudiante. Como se sabe, la tabla estudiante tiene una llave primaria que a la vez es una referencia de la tabla persona, mire un ejemplo en la siguiente ilustración.

**Comentado [OG41]:** El editable se puede obtener del Anexo "base de datos" Page-13

<https://drive.google.com/file/d/1W-ggJPezRs7noNMYjiOdAk2c5SMXfB3o/view?usp=sharing>

**Comentado [OG42]:** Llamado a la acción de recurso externo versión simplificada.





| id_persona | correo           | nombres | apellidos | id_estudiante | identificacion | direccion            | telefono | grado_escolar |
|------------|------------------|---------|-----------|---------------|----------------|----------------------|----------|---------------|
| 1          | ana@gmail.com    | Ana     | lis       | 3             | 12.345.565     | arrera 3 # 18- 45    | 7825992  | 6             |
| 2          | jose@gmail.com   | Jose    | Omar      | 5             | 12.490.345     | Carrera 8a No. 7 - 8 | 3305040  | 3             |
| 3          | luis@gmail.com   | Luis    | Dario     | 1             | 13.767.234     | Calle 15 No. 9 - 56  | 7273995  | 5             |
| 4          | arm@gmail.com    | Armando | Bronca    | 7             | 13.498.543     | Carrera 7A No. 32    | 8723041  | 7             |
| 5          | victor@gmail.com | Vistor  | José      | 8             | 13.242.343     | Calle 12 No. 4 - 19  | 4210220  | 8             |
| 6          | josen@gmail.com  | Jose    | Nicolás   | 6             | 19.833.574     | Calle 7 No. 5 - 25   | 6733643  | 7             |
| 7          | andrea@gmail.com | Andrea  | Martinez  |               |                |                      |          |               |

a) Tabla persona

b) Tabla estudiante

Ilustración 23. Error de integridad referencial

Como siguiente la ilustración 23, por cada estudiante (tabla estudiante) existe uno y solo un elemento en la tabla persona, de tal forma que un estudiante es la conjunción de ambas tablas por medio de la llave foránea (que a la vez es llave primaria) id\_estudiante, ¿pero ¿qué tal que no existiera?, eso significa que la base de datos es inconsistente o que tiene datos basura en la tabla estudiante. Afortunadamente los SGDB permite definir reglas que impidan que esto ocurra en las tablas, es decir que no permita registrar un estudiante con id\_estudiante con un valor que no existe en la columna id\_persona de la tabla persona.

#### Pregunta de análisis de datos

Ahora pregúntese lo siguiente: ¿qué pasaría si se borra el registro de la tabla persona que tiene id\_persona con valor tres (3) sufrirá algún problema los datos de la tabla estudiante?, o que pasaría si se cambia el valor de tres (3) en la columna id\_persona, por otro valor como por ejemplo nueve (9). antes de continuar tome unos minutos para pensarlo.

#### RESPUESTA

La respuesta a las cuestiones planeadas es que en cualquiera de las operaciones de borrado o actualización del dato se incurrirá en una violación de la integridad referencial de la tabla estudiante, ya que no habrá manera de conocer los datos de complementarios del estudiante que se encuentran en la tabla persona.

Para evitar estos problemas de insistencia los sistemas gestores de bases de datos permiten definir unas reglas cuando se borra, o actualiza un dato que es referencia foránea de otra tabla.

**Regla de borrado:** define el comportamiento a la pregunta ¿Qué ocurre si se intenta borrar la fila referenciada por la clave foránea de otra tabla? Para lo cual el SGBD implementa una de las siguientes opciones:

- **Restringir:** no se permite borrar la fila referenciada.
- **Cascada:** se borra la fila referenciada y también se borran las filas que la referencian mediante la clave foránea en la otra tabla.
- **Poner null:** se borra la fila referenciada y las filas que la referenciaban ponen a nulo la clave foránea (sólo si acepta nulos).
- **Valor por defecto:** se borra la fila referenciada y las filas que la referenciaban ponen en la clave foránea el valor por defecto defina la regla.

**Regla de edición:** define el comportamiento a la pregunta: ¿Qué sucede si se intenta modificar el valor de la clave primaria de la fila referenciada por la clave foránea por otra tabla?

- **Restringir:** no se permite editar la fila referenciada.
- **Cascada:** se actualiza la fila referenciada y también se actualiza las filas que la referencian mediante la clave foránea en la otra tabla.

**Comentado [OG43]:** Realizar acordeón con las reglas de integridad:

- **Criterio de NULIDAD (+)**
- **Integridad de entidad (+)**
- **Integridad referencia. (+)**

**Comentado [OG44]:** Bloque de texto destacado.



- Poner null: se actualiza la fila referenciada y las filas que la referenciaban ponen a nulo la clave foránea (sólo si acepta nulos).
- Valor por defecto: se edita la fila referenciada y las filas que la referenciaban ponen en la clave foránea el valor por defecto defina la regla.

De esta forma una vez definir la estructura del modelo de datos de datos, se debe determinar el comportamiento de estas operaciones, lo cual generalmente responde a requerimiento del sistema de información.

### 3.4. Lenguajes de los sistemas administradores de bases de datos.

Los sistemas gestores de bases de datos emplean un lenguaje que se denomina SQL que corresponde al nombre en inglés (**Structured Query Language**), Generalmente, cuando un SGBD relacional implementa el lenguaje SQL, todas las acciones, y operaciones se llevan a cabo en el sistema mediante sentencias de este lenguaje. Dentro de SQL hay varios tipos de sentencias que se agrupan en cuanto conjuntos:

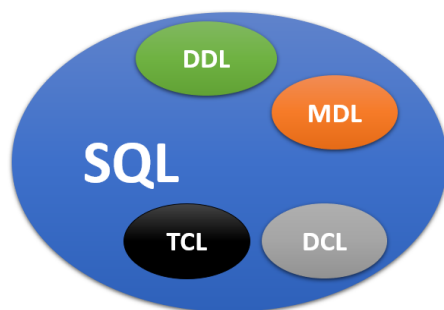


Imagen. Conjunto bases de datos SQL

- ✓ **Sentencias de definición de datos (DDL):** son las sentencias que permiten crear tablas, alterar su definición y eliminarlas. En una base de datos relacional existen otros tipos de objetos además de las tablas, como las vistas, los índices y los disparadores. Las sentencias para crear, alterar y eliminar vistas e índices también pertenecen a este conjunto de sentencia se les denomina **DDL** del inglés **Data Definition Language**.

#### Ejemplo DDL

Para crear la tabla persona de la ilustración 22. Quedaría de la siguiente forma:

```
CREATE TABLE persona (  
  id_persona INT NOT NULL ,  
  nombres VARCHAR(250) NOT NULL ,  
  apellidos VARCHAR(250) NOT NULL ,  
  correo VARCHAR(250) NOT NULL ,  
  PRIMARY KEY (id_persona)  
);
```

**Comentado [OG45]:** Usar tarjetas o dos tabs para resaltar las dos reglas:

- ✓ Regla borrado
- ✓ Regla edición

**Comentado [OG46]:** Rediseñar imagen.

<https://images.app.goo.gl/Nu6rPypE2MnAhUm79>



Note lo siguiente: cada atributo se define en una línea separado por una coma, se agregan restricciones de no nulidad (NOT NULL) al final la sentía termina con punto y coma. Ahora mire como se borraría esa misma tabla:

```
DROP TABLE persona;
```

- ✓ **Sentencias de manejo de datos (DML):** estas sentencias que permiten insertar datos en las tablas, consultarlos, editarlos y borrarlos, se denomina **DML** del inglés **Data Manipulation Language**.

#### Ejemplo DML

Veamos cómo se inserta una fila a la tabla persona, luego como se actualiza la fila insertada y posteriormente como se borra:

```
INSERT INTO persona (id_persona, nombres, apellidos, correo)  
VALUES (1, 'Ana lis', 'Mendez', 'ana@rmail.com');
```

```
UPDATE persona SET nombres = 'Ana Lis' WHERE id_persona = 1;
```

```
DELETE FROM persona WHERE id_persona = 1;
```

- ✓ **Sentencias de control (DCL):** son las sentencias empleadas por los administradores de la base de datos para realizar tareas como, por ejemplo, crear usuarios y concederles o revocar los privilegios. Se usa el término **DCL** del inglés **Data Control Language**.

#### Ejemplo DCL

En el ejemplo se le da permisos a un usuario llamado **userdb** de la base de datos de nombre biblioteca\_db para que pueda insertar, actualizar, borrar y consultar datos de la tabla persona desde el mismo computador donde está instalada la base de datos (localhost).

```
GRANT INSERT, UPDATE, DELETE, SELECT ON biblioteca_db.persona TO  
userdb@'localhost';
```

- ✓ **Sentencias de control de transacciones (TCL):** un pequeño grupo de sentencias que permiten procesar en bloque operaciones **DML** garantizando que se efectúen todas y cada una de las operaciones o ninguna. **TCL** del inglés **Transaction Control Language**.

#### 4. Sistema gestor de base de datos

En el mercado existen muchos sistemas gestores de bases de datos y a continuación se usará MySQL para realizar las prácticas, pero ello no significa ni que sea el mejor, o el más completo del mercado, sino más bien el más usado para aplicaciones web, de pequeña, mediana y algunos casos de gran complejidad. Es un buen punto de partida y todos los conceptos en MySQL vistos son reutilizables en PostgreSQL, Oracle, u otro motor de bases de datos relacional.

A continuación, podrá visualizar el paso a paso de la descarga e instalación del gestor MySQL Workbench y Web XAMPP:

VIDEO DESCARGA E INSTALACIÓN de herramienta CASE MySQL Workbench.

**Comentado [OG47]:** Utilizar acordeón para ocultar el ejemplo y que el aprendiz interactúe para visualizarlo.

**Ejemplo DDL (+)**

**Comentado [OG48]:** Utilizar acordeón para ocultar el ejemplo y que el aprendiz interactúe para visualizarlo.

**Ejemplo DML (+)**

**Comentado [OG49]:** Utilizar acordeón para ocultar el ejemplo y que el aprendiz interactúe para visualizarlo.

**Ejemplo DCL (+)**

**Comentado [OG50]:** Los vídeos se disponen en la carpeta anexos y lo ideal es generar el intro y outro y agregar los datos del experto que genera los vídeos:  
Peter Emerson Pinchao Solis  
Experto temático  
Regional Cauca, Centro de teleinformática y producción industrial

Si es posible crear carousel con los dos vídeos generados por el experto para la descarga e instalación de **Workbench y XAMPP** de lo contrario dos modals para la visualización y reproducción.





#### VIDEO DESCARGA E INSTALACIÓN de entorno desarrollo WEB XAMPP.

Finalmente revise cuidadosamente cómo usar MySQL Workbench para diseñar la base de datos del Modelo relacional de la biblioteca del numeral 3.2 y la ilustración 22:

#### VIDEO DISEÑO DE UNA BASE DATOS con MySQL Workbench relación de uno a uno

#### VIDEO DISEÑO DE UNA BASE DATOS con MySQL Workbench relación de uno a muchos

#### VIDEO DISEÑO DE UNA BASE DATOS con MySQL Workbench relación de muchos a muchos

#### C. ACTIVIDADES DIDÁCTICAS (OPCIONALES SI SON SUGERIDAS)

N/A

#### D. MATERIAL COMPLEMENTARIO:

| Referencia APA del Material                                                                                                           | Tipo de material<br>(Video, capítulo de libro,<br>artículo, otro) | Enlace del Recurso o<br>Archivo del documento o<br>material                                                                                                                                                                                     |
|---------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Espinosa, A. P. (2014, Nov. 16). L6 1 Nociones y modelos de bases de datos. Youtube.                                                  | Vídeo                                                             | <a href="https://www.youtube.com/watch?v=pPATLxiiDfw">https://www.youtube.com/watch?v=pPATLxiiDfw</a>                                                                                                                                           |
| Silberschatz, A., Korth, H. F., Sudarshan, S., Pérez, F. S., Santiago, A. I., & Sánchez, A. V. (2006). Fundamentos de bases de datos. | PDF                                                               | <a href="http://ceneval.isi.uson.mx/CENEVAL/4Implantacion%20de%20infraestructura/Fundamentos%20de%20base%20de%20datos.pdf">http://ceneval.isi.uson.mx/CENEVAL/4Implantacion%20de%20infraestructura/Fundamentos%20de%20base%20de%20datos.pdf</a> |

#### E. GLOSARIO:

| TERMINO              | SIGNIFICADO                                                                                                                                                                                                              |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Base datos           | Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.                                                                                                                                 |
| Metadatos            | Conjunto unitario de instrucciones que permite a una computadora realizar funciones diversas, como el tratamiento de textos, el diseño de gráficos, la resolución de problemas matemáticos, el manejo de bancos de datos |
| Diccionario de datos | Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.                                                                                                              |

#### F. REFERENCIAS BIBLIOGRÁFICAS:

Asale, R.-, & Rae. (2021). base | Diccionario de la lengua española. <https://dle.rae.es/base#CiiosqO>

**Comentado [OG51]:** Los vídeos se disponen en la carpeta anexos y lo ideal es generar el intro y outro y agregar los datos del experto que genera los vídeos: Peter Emerson Pinchao Solis  
Experto temático  
Regional Cauca, Centro de teleinformática y producción industrial

Si es posible crear carousel o modal con los tres vídeos generados por el experto para el uso del gestor Workbench en la relación:

- ✓Uno a uno
- ✓Uno a muchos
- ✓Muchos a muchos



Kyocera, I. Conceptos sobre base de datos orientada a objetos. (2021). <https://rb.gv/5deqdh>

López Juan Carlos, 2009, Algoritmos y programación (guía para docentes). Eduteka <http://eduteka.icesi.edu.co/articulos/GuiaAlgoritmos>

Silberschatz, A., Korth, H. F., Sudarshan, S., Pérez, F. S., Santiago, A. I., & Sánchez, A. V. (2006). Fundamentos de bases de datos. <http://ceneval.isi.uson.mx/CENEVAL/4Implantacion%20de%20infraestructura/Fundamentos%20de%20base%20de%20datos.pdf>

Resnick, Mitchel (2007): Sembrando las semillas para una sociedad más creativa. Laboratorio de medios de MIT, Massachussets. Eduteka. <http://www.eduteka.org/ScratchResnickCreatividad.php>

Maryqiu1987. (2021). Descripción de normalización de base de datos. <https://docs.microsoft.com/es-es/office/troubleshoot/access/database-normalization-description>

#### G. CONTROL DEL DOCUMENTO

|            | Nombre                        | Cargo                            | Dependencia<br>(Para el SENA indicar Regional y Centro de Formación) | Fecha     |
|------------|-------------------------------|----------------------------------|----------------------------------------------------------------------|-----------|
| Autor (es) | Henry Eduardo Bastidas Paruma | Experto temático                 | Regional Cauca, Centro de teleinformática y producción industrial    | Mayo 2021 |
|            | Peter Emerson Pinchao Solis   | Experto temático                 | Regional Cauca, Centro de teleinformática y producción industrial    | Mayo 2021 |
|            | Oscar Absalón Guevara         | Diseñador instruccional          | Centro de Gestión Industrial                                         | Mayo 2021 |
|            | Rafael Neftalí Lizcano Reyes  | Asesor Pedagógico Ecosistema RED | Centro Industrial del Diseño y la Manufactura - Regional Santander   |           |
|            |                               |                                  |                                                                      |           |

#### H. CONTROL DE CAMBIOS

(Diligenciar únicamente si realiza ajustes a la Unidad Temática)

|            | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
|------------|--------|-------|-------------|-------|------------------|
| Autor (es) |        |       |             |       |                  |

## VUELTA CICLISTA

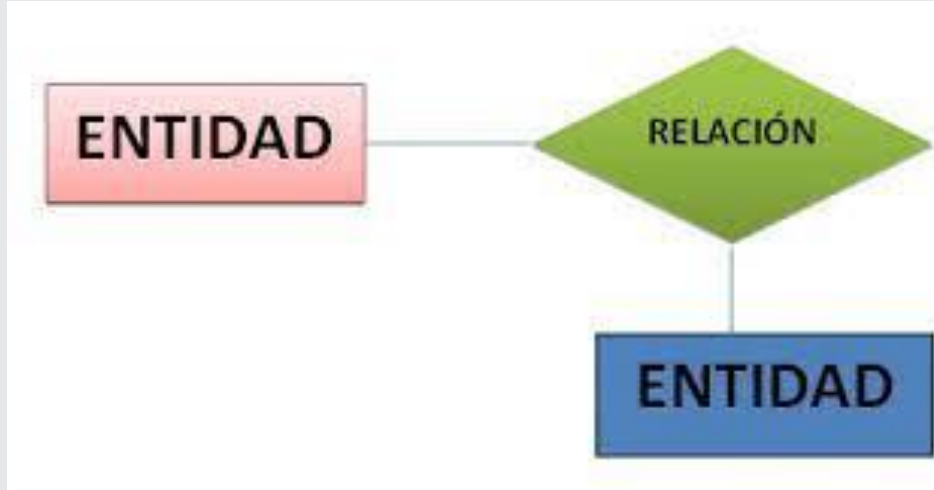
Se requiere diseñar una base de datos sobre una vuelta ciclista. En la vuelta participan una serie de equipos, que están formados por los ciclistas y un director deportivo. Cada ciclista tiene asignado un dorsal, y nos gustaría almacenar en la base de datos su nombre, edad y nombre del equipo al que pertenece. Un ciclista sólo puede pertenecer a un equipo. Los ciclistas disputan una serie de etapas. Cada etapa logra ganarla sólo un ciclista. De estas etapas nos interesa conocer su número (las etapas se numeran consecutivamente: 1, 2, los kilómetros que se recorren, el nombre de la población de salida y el de la población donde está la llegada o meta de la etapa.

Las etapas pueden tener puertos de montaña. Un puerto no puede aparecer en más de una etapa. De los puertos nos interesa conocer su nombre, el porcentaje que indica la pendiente media del puerto, la altura máxima y la categoría (primera, especial, etc.). Los ciclistas pueden ganar puertos, y un puerto sólo lo puede ganar un ciclista, que será el primero que lo alcance.

También nos interesa saber los maillots que puede llevar un ciclista en una etapa. Estos maillots otorgan una serie de premios a los ciclistas que los llevan al finalizar la vuelta. Las características más importantes que nos interesan de un maillot son: su código, tipo o clasificación que premia ese maillot (general, montaña, etc.). color de la camiseta asociada e importe del premio que corresponde al ciclista que termine la vuelta llevando el maillot. Los ciclistas pueden llevar uno o varios maillots (aunque sólo se pongan uno) en una misma etapa.



# Modelo Entidad Relación E-R



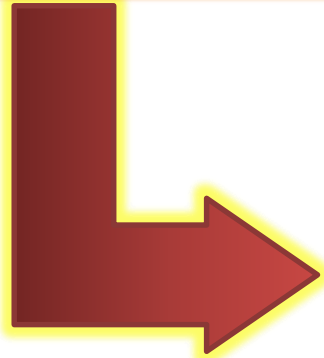
Instructor: Gustavo Salazar Escobar



@SENAcomunica

[www.sena.edu.co](http://www.sena.edu.co)

**1- Competencia 220501095 -  
Diseñar la solución de software de  
acuerdo con procedimientos y  
requisitos técnicos.**



**Actividad de Aprendizaje GA4-220501093-  
AA1 Diseñar la estructura de la base de datos  
del sistema.**

# Análisis y Desarrollo de Software ADSO



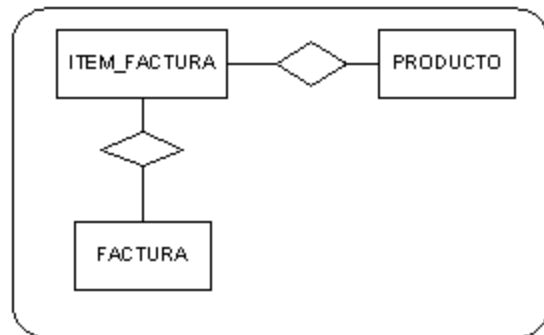
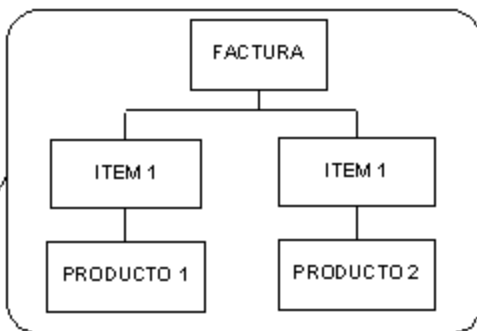
| ACTIVIDAD DE APRENDIZAJE                                                                                                                                                                                 | RESULTADO DE APRENDIZAJE (RAP)                                                                                                | Evidencia a Aprobar                                                                                                                                                                                                                                                                                                                  | Fecha de Entrega                      |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|
| <b>GA4-220501093- AA1</b><br><b><i>Diseñar la estructura de la base de datos del sistema.</i></b><br><br><b>GA4-220501095-AA1</b><br><b><i>Diseñar la estructura de la base de datos del sistema</i></b> | <b>220501095-02 -</b><br>Estructurar el modelo de datos del <i>software</i> de acuerdo con las especificaciones del análisis. | <b>GA4-220501093-AA1- EV01</b><br>Identificar entidades y relaciones planteadas en el caso de estudio<br><br><b>GA4-220501095-AA1- Diseñar el EV01 -</b> modelo conceptual y lógico de acuerdo al caso de estudio<br><br><b>GA4-220501095-AA1- conceptual EV02 -</b> Modelos y lógico para el proyecto desarrollo de <i>software</i> | 06/09/2022<br>a<br>27/09<br><br>/2022 |

# MER (Modelo Entidad Relación)

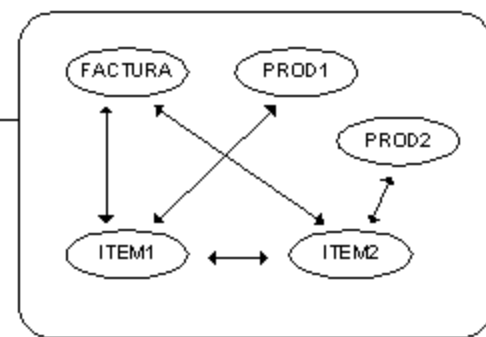


El modelado de datos es independiente del hardware o software usado para su implementación. Un modelo Entidad - Relación, puede ser implementado en bases de datos jerárquicas, red o relacionales.

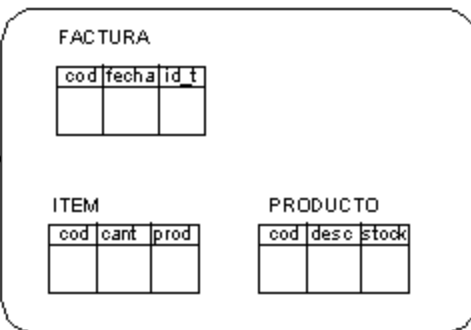
Modelo Entidad-Relación



Base Datos Red



Base Datos Relacional





# Conceptos claves



# Entidades



- Una entidad es un objeto que tiene significado o importancia, cuya información se necesita conocer.
- Un objeto de interés al negocio
- Una clase o categoría de algo
- El nombre de una cosa

# Entidades Ejemplo:



➤ En el contexto de administración de personal de una empresa las entidades pueden ser:

➤ Empleado

➤ Departamento

➤ Proyecto

# Atributos



**Describen las entidades y son las piezas específicas de información que se desea conocer**

➤ **Por ejemplo, posibles atributos para la entidad empleado serían:**

➤ **-Nombre**

➤ **-Fecha nacimiento**

➤ **-Sueldo**

➤ **-Rut**

# Atributos



**Para cada atributo hay un conjunto de valores permitidos, llamados el dominio o el conjunto de valores de ese atributo.**

**Tipos de atributos:**

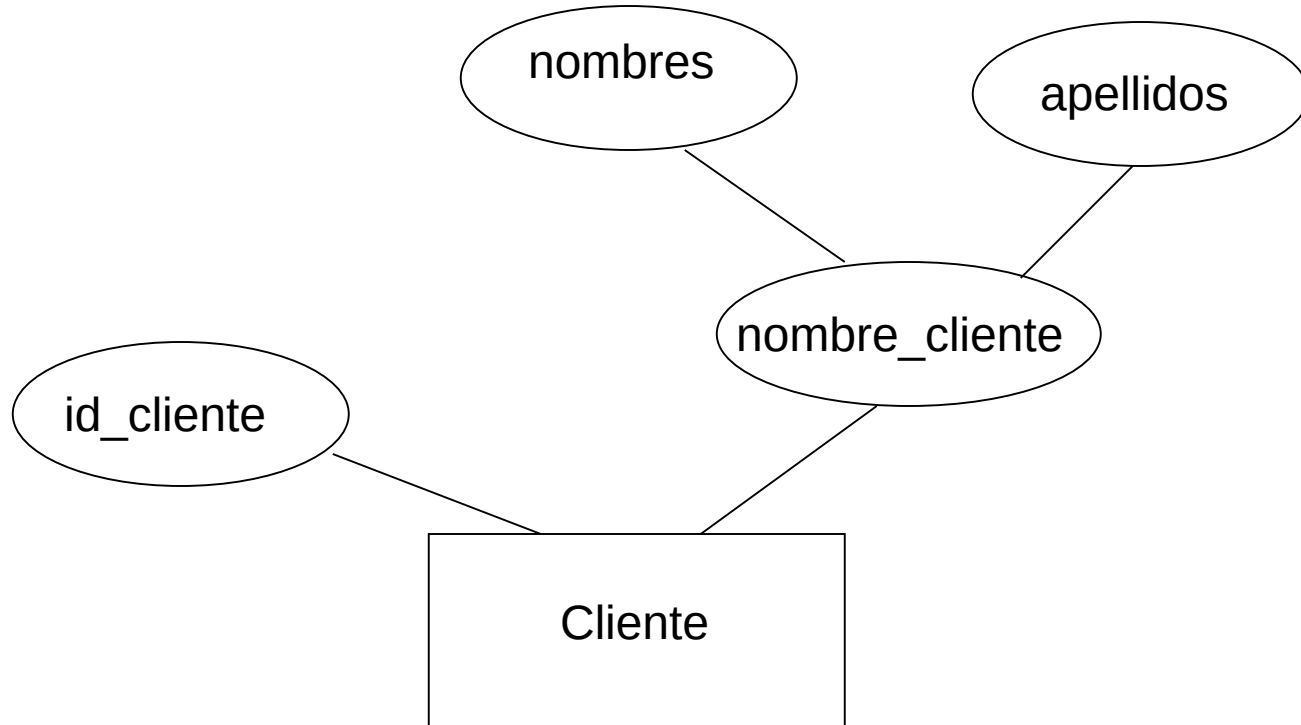
**Simple y compuestos**

**Univalorados y multivalorados**

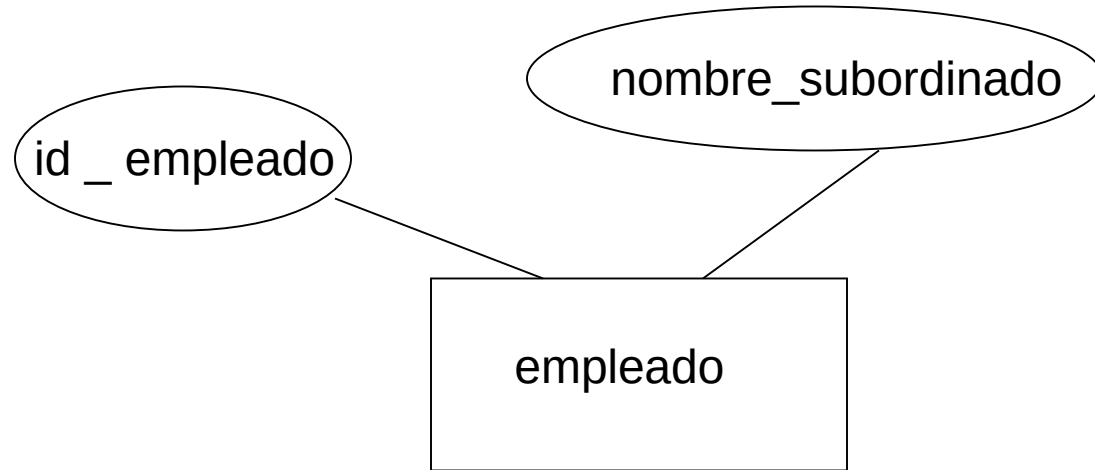
**Nulos**

**Derivados**

# Atributos simples y compuestos



# Atributos univalorados y multivalorados



id\_empleado es univalorado

nombre\_subordinado es multivalorado

# Atributos Nulos



Se usan cuando una entidad no tiene valor para un atributo

Puede indicar que el valor del atributo es desconocido

Un atributo desconocido puede ser bien perdido ( el valor existe pero no se dispone de la información) o no conocido ( no se sabe realmente si el valor existe o no)



# Atributos Derivados



Son atributos que se obtienen a partir de otros atributos.

Ejemplo:

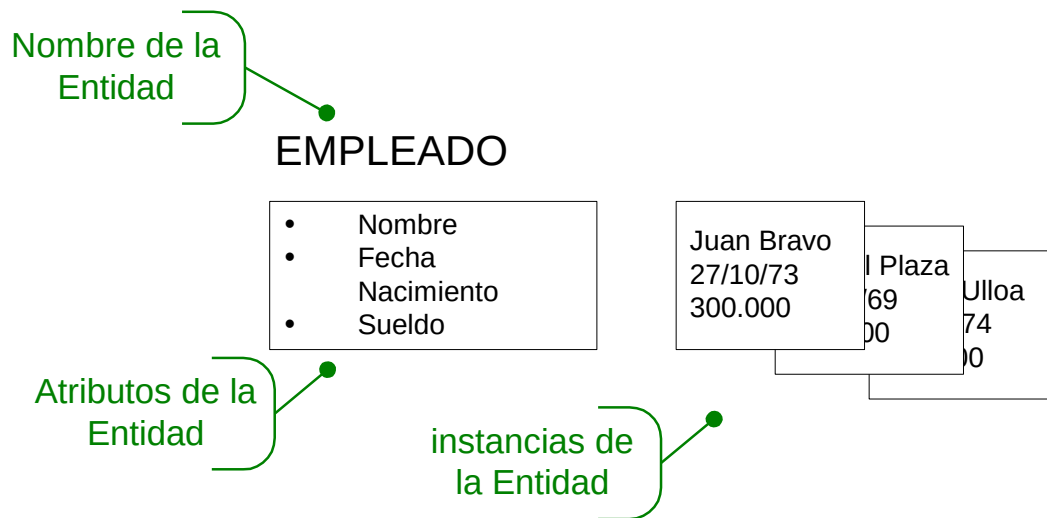
edad\_empleado

antigüedad\_cliente

# Instancias



Cada entidad debe tener múltiples ocurrencias o instancias. Por ejemplo, la entidad Empleado, tiene una ocurrencia (o instancia) por cada empleado en la empresa.



# Tuplas



Las tablas de la ilustración 5 muestran cada un conjunto de registros, la tabla persona muestra 6 registros y la tabla cuenta muestra 9 de ellos, cada uno de estos registros representa un objeto, es decir, de la tabla persona hay 6 (objetos) o personas a la agrupación de los datos de cada registro se le llama TUPLA. En términos simples una TUPLA de una base datos es el conjunto de todos los atributos (columnas) de una fila.

| identificacion | nombres      | apellidos     | edad |
|----------------|--------------|---------------|------|
| 8.127.346      | Ana lis      | Mendez        | 34   |
| 212.394.762    | Jose Omar    | Ciano         | 28   |
| 545.232.546    | Luis Dario   | Marquez       | 21   |
| 125.446.732    | Armando      | Bronca Segura | 35   |
| 33.565.236.787 | Vistor José  | Diento Paz    | 24   |
| 2.343.554.768  | Jose Nicolás | Timado Ruiz   | 19   |
| 45.456.778     | Andrea       | Martinez      | 36   |

a) Tabla persona

| numero_cuenta | saldo            | fecha_creacion |
|---------------|------------------|----------------|
| 19.283.746    | \$ 2.589.954,00  | 25/08/1998     |
| 67.789.901    | \$ 23.465.212,00 | 26/08/2001     |
| 32.112.312    | \$ 2.343.242,00  | 27/08/2004     |
| 41.928.374    | \$ 2.343.423,00  | 31/08/2013     |
| 38.342.657    | \$ 8.965.211,00  | 30/08/2010     |
| 32.443.171    | \$ 11.172.594,33 | 22/06/2001     |
| 34.920.608    | \$ 14.483.578,83 | 01/09/2016     |
| 17.980.455    | \$ 17.794.563,33 | 03/09/2019     |
| 26.754.825    | \$ 21.105.547,83 | 12/04/2022     |

b) Tabla cuenta

# Llaves o Claves



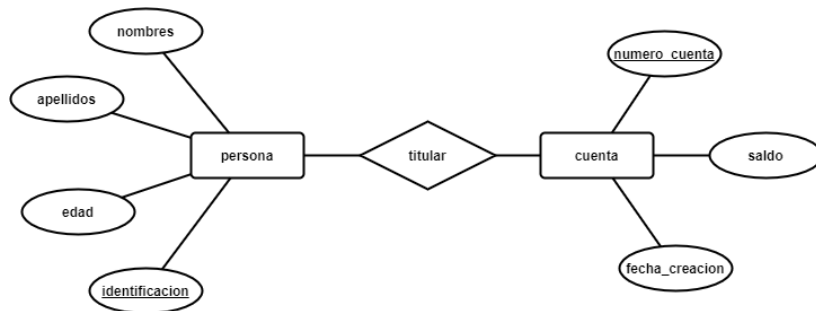
Dado que en una tabla no hay filas repetidas, se pueden distinguir unas de otras, es decir, se pueden identificar de manera única (por el principio de unidad). Por lo tanto, la forma de identificarlas inequívocamente es según los valores de los atributos. De esta forma se conoce como claves a que los atributos que sirven para identificar un registro de forma inequívoca.

# Llaves o Claves



- Ejemplo:

En la tabla persona, es sabido por todos que no existe dos personas con el mismo número de documento de identidad, por lo tanto, se puede emplear el atributo **identificación** como calve de la tabla persona. De igual forma no debe existir dos cuentas bancarias con el mismo número de cuenta, entonces, el atributo **numero\_cuenta** se puede emplear como clave de la tabla cuenta. Esto se lo puede presentar en un diagrama de entidad relación cuando a los atributos que componen una clave les marcamos subrayado el nombre del atributo como se muestra en la siguiente ilustración.



# Llaves Primaria

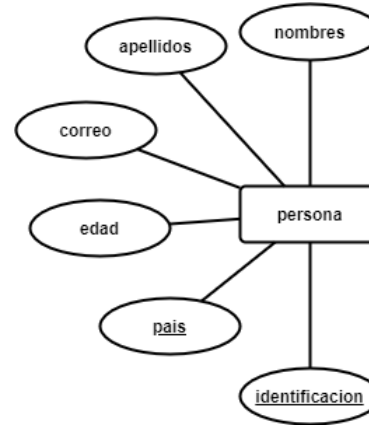
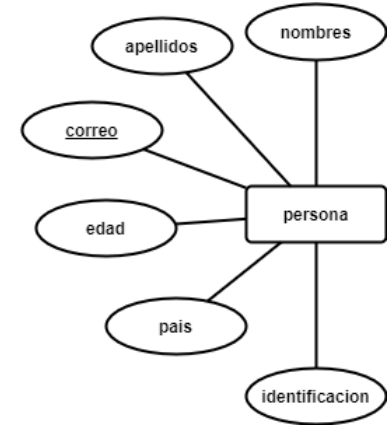


Se empleará el término clave primaria para nombrar una clave candidata que es elegida por el diseñador o analista de la base de datos como clave principal para identificar las tuplas dentro de una entidad. Una clave (primaria, candidata y superclave) es una propiedad del conjunto de entidades, más que de las entidades individuales. Entre dos tuplas individuales en el conjunto no pueden tener el mismo valor en sus atributos clave primaria al mismo tiempo. La designación de una clave primaria es en sí una restricción que modela una característica del mundo real.

# Llaves Compuesta



Si se escogiera el conjunto de atributos **identificación** y **país** como clave de la entidad, se está haciendo una clave compuesta esta porque es un conjunto de más de un atributo el que se usa para este fin.



El otro asunto a considerar es que, se puede emplear el **correo** electrónico como clave ya que no existen dos usuarios con el mismo correo electrónico, esto es válido si se definió como regla del banco que cada cliente tenga su **correo** electrónico diferente de los demás, estos dos casos se representan en la siguiente ilustración.

# Llave Candidata

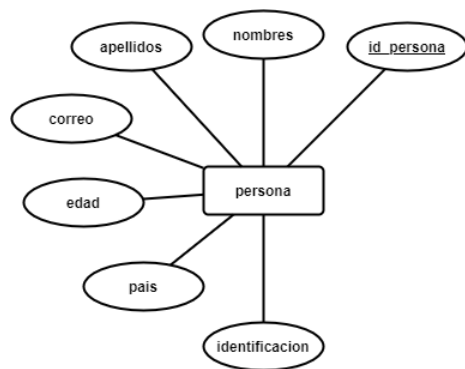


Los dos caminos propuestos en la ilustración 7 son válidos, y se denominan claves candidatas porque con ambas se puede lograr el principio de unicidad, y es potestad de diseñador o analista de sistemas determinar cuál de las dos usa.

Sin embargo, existe otro tipo de solución y es asignar le un atributo identificador, a la entidad persona que se encargue exclusivamente de identificar inequívocamente cada tupla o fila, a este atributo le podemos llamar **id\_persona** que sería un número que el programa o software para el que estamos desarrollando la base le asignara a cada cliente cada vez que se registre por primera vez los datos de una persona (generalmente una secuencia). De esta forma tendríamos 3 claves candidatas, la ilustración 8 muestra cómo sería la entidad y la tabla que representa.



# Llave Candidata



| id_persona | pais     | identificacion | correo           | nombres      | apellidos     | edad |
|------------|----------|----------------|------------------|--------------|---------------|------|
| 1          | colombia | 8.127.346      | ana@rmail.com    | Ana lis      | Mendez        | 34   |
| 2          | colombia | 212.394.762    | jose@pmail.com   | Jose Omar    | Ciano         | 28   |
| 3          | colombia | 545.232.546    | luis@jmail.com   | Luis Dario   | Marquez       | 21   |
| 4          | colombia | 125.446.732    | arm@hmail.com    | Armando      | Bronca Segura | 35   |
| 5          | colombia | 33.565.236.787 | victor@umail.com | Vistor José  | Diento Paz    | 24   |
| 6          | colombia | 2.343.554.768  | josen@pmail.com  | Jose Nicolás | Timado Ruiz   | 19   |
| 7          | perú     | 45.456.778     | andrea@email.com | Andrea       | Martinez      | 36   |

**Tabla persona**

# Notación Diagramas E-R



Entidades



Atributos



Atributos  
multivaluados

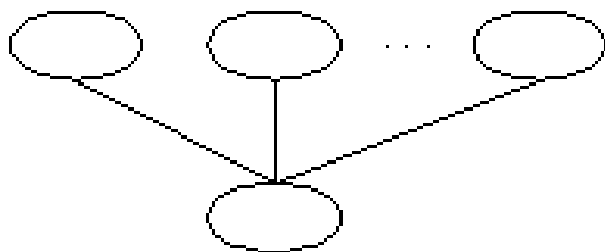


Atributos derivados

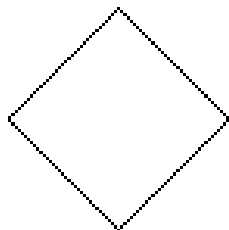


Atributo Clave

# Notación Diagramas E-R



Atributo Compuesto

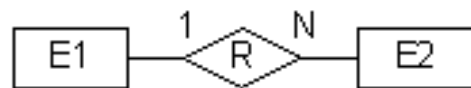


Relaciones

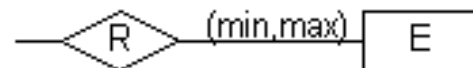
# Notación Diagramas E-R



Participación total de E2 en R

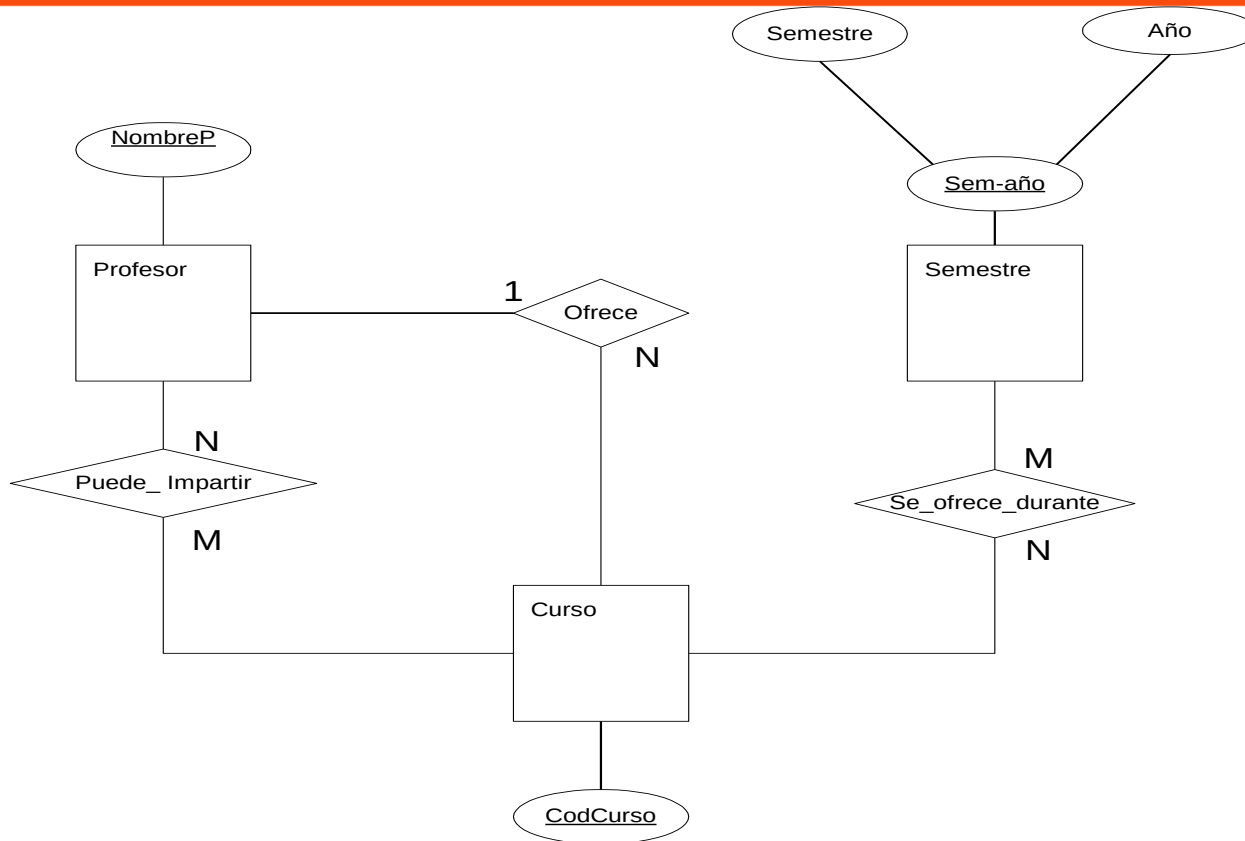


Razón de cardinalidad 1:N para E1:E2 en R



Restricción estructural (min,max) de la participación de E en R

# Notación Diagramas E-R



# Relaciones



Una relación es una asociación bi-direccional entre dos entidades, o entre una entidad consigo misma.

Ejemplo:

la relación entre las entidades INSTRUCTOR y CURSO puede ser:

Cada CURSO puede ser impartido por sólo un INSTRUCTOR.

Cada INSTRUCTOR puede ser asignado a uno a más cursos.

**Principio de unicidad:** en una tabla, nunca deben existir TUPLAS repetidas, es decir, no debe haber más de una fila con exactamente los mismos valores en cada columna que otra fila de la misma tabla. En otras palabras cada fila debe diferir al menos en un elemento (columna) con relación a las demás.

# Relaciones



Grado de opcionalidad:

- Opcional: Puede ser.
- Obligatorio: Debe ser

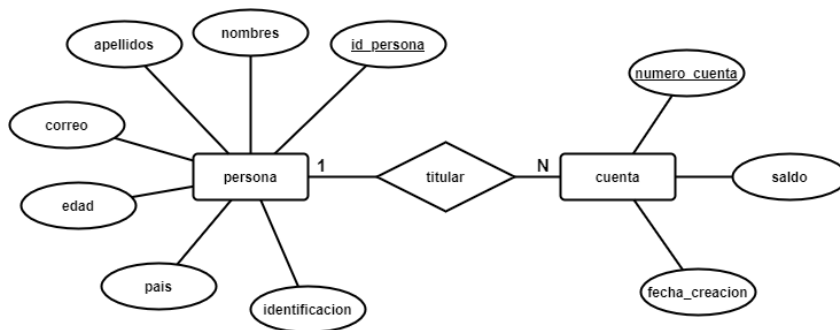
Grado de cardinalidad:

- uno o muchos
- uno a uno

# Relaciones de uno a muchos.



Una relación de uno a varios o de uno a muchos es el tipo de relación más empleada. En este tipo de relación, una fila de la tabla A puede tener muchas filas coincidentes en la tabla B. Pero una fila de la tabla B solo puede tener una fila coincidente en la tabla A. Por ejemplo, las tablas **pesona** y **cuenta** tienen una relación de uno a muchos. Es decir, cada persona puede tener más de una cuenta. Pero cada título proviene de una sola editorial.





# Relaciones de uno a muchos.



A esta representación se le denomina multiplicidad **1:N** y se lee de la siguiente forma: **UNA persona** es **titular** de **N cuentas** bancarias, de esta forma una persona puede tener **N** (que puede ser cero, uno o más de una) cuentas bancarias. Se puede leer en forma inversa que el resultado es el mismo: muchas o varias cuentas bancarias pueden tener un mismo titular.

Cuando una relación es de uno a muchos y se mapea a un modelo de base de datos relacional, la relación **titular** (rombo) se termina convirtiendo una columna de la tabla de los muchos.

| id_persona | pais     | identificacion | correo           | nombres      | apellidos     | edad |
|------------|----------|----------------|------------------|--------------|---------------|------|
| 1          | colombia | 8.127.346      | ana@rmail.com    | Ana lis      | Mendez        | 34   |
| 2          | colombia | 212.394.762    | jose@pmail.com   | Jose Omar    | Ciano         | 28   |
| 3          | colombia | 545.232.546    | luis@jmail.com   | Luis Dario   | Marquez       | 21   |
| 4          | colombia | 125.446.732    | arm@hmail.com    | Armando      | Bronca Segura | 35   |
| 5          | colombia | 33.565.236.787 | victor@umail.com | Vistor José  | Diento Paz    | 24   |
| 6          | colombia | 2.343.554.768  | josen@pmail.com  | Jose Nicolás | Timado Ruiz   | 19   |
| 7          | perú     | 45.456.778     | andrea@email.com | Andrea       | Martinez      | 36   |

a) Tabla persona

| numero_cuenta | titular | saldo            | fecha_creacion |
|---------------|---------|------------------|----------------|
| 19.283.746    | 7       | \$ 2.589.954,00  | 25/08/1998     |
| 67.789.901    | 1       | \$ 23.465.212,00 | 26/08/2001     |
| 32.112.312    | 5       | \$ 2.343.242,00  | 27/08/2004     |
| 41.928.374    | 2       | \$ 2.343.423,00  | 31/08/2013     |
| 38.342.657    | 6       | \$ 8.965.211,00  | 30/08/2010     |
| 32.443.171    | 1       | \$ 11.172.594,33 | 22/06/2001     |
| 34.920.608    | 5       | \$ 14.483.578,83 | 01/09/2016     |
| 17.980.455    | 3       | \$ 17.794.563,33 | 03/09/2019     |
| 26.754.825    | 4       | \$ 21.105.547,83 | 12/04/2022     |

b) Tabla cuenta

# Relaciones de uno a muchos.



## REGLA DE MAPEO 1:N de Modelo entidad-relación a modelo relacional:

Una relación de uno a mucho (1:N) se transforma en una columna en la tabla que tiene la multiplicidad de los muchos (cuenta). Y todos los datos de esta columna existen como llave primaria en la tabla que tiene la multiplicidad uno (tabla persona).

| id_persona | pais     | identificacion | correo           | nombres      | apellidos     | edad |
|------------|----------|----------------|------------------|--------------|---------------|------|
| 1          | colombia | 8.127.346      | ana@rmail.com    | Ana lis      | Mendez        | 34   |
| 2          | colombia | 212.394.762    | jose@pmail.com   | Jose Omar    | Ciano         | 28   |
| 3          | colombia | 545.232.546    | luis@jmail.com   | Luis Dario   | Marquez       | 21   |
| 4          | colombia | 125.446.732    | arm@hmail.com    | Armando      | Bronca Segura | 35   |
| 5          | colombia | 33.565.236.787 | victor@umail.com | Vistor José  | Diento Paz    | 24   |
| 6          | colombia | 2.343.554.768  | josen@pmail.com  | Jose Nicolás | Timado Ruiz   | 19   |
| 7          | perú     | 45.456.778     | andrea@email.com | Andrea       | Martinez      | 36   |

a) Tabla persona

| numero_cuenta | titular | saldo            | fecha_creacion |
|---------------|---------|------------------|----------------|
| 19.283.746    | 7       | \$ 2.589.954,00  | 25/08/1998     |
| 67.789.901    | 1       | \$ 23.465.212,00 | 26/08/2001     |
| 32.112.312    | 5       | \$ 2.343.242,00  | 27/08/2004     |
| 41.928.374    | 2       | \$ 2.343.423,00  | 31/08/2013     |
| 38.342.657    | 6       | \$ 8.965.211,00  | 30/08/2010     |
| 32.443.171    | 1       | \$ 11.172.594,33 | 22/06/2001     |
| 34.920.608    | 5       | \$ 14.483.578,83 | 01/09/2016     |
| 17.980.455    | 3       | \$ 17.794.563,33 | 03/09/2019     |
| 26.754.825    | 4       | \$ 21.105.547,83 | 12/04/2022     |

b) Tabla cuenta

# Relaciones de muchos a muchos.



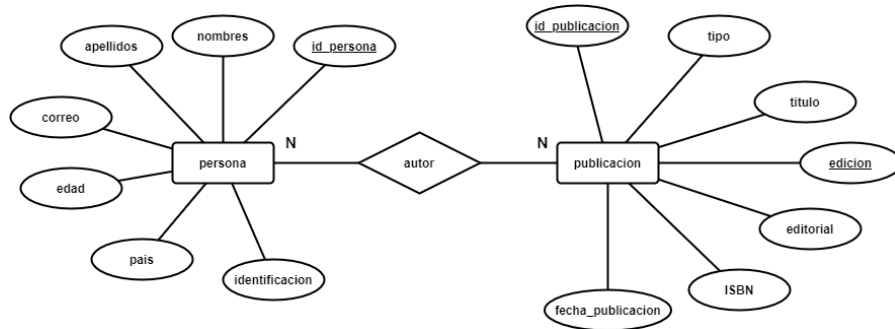
En una relación de muchos a muchos, una fila de la tabla A puede tener relación con muchas filas en la tabla B y también en sentido contrario una fila de la tabla B con respecto de varias filas de la tabla A. Esta relación se crea definiendo una tercera tabla denominada tabla de relación. Esta nueva tabla tiene una clave principal de la tabla compuesta de las claves foráneas de la tabla A y de la B.

# Relaciones de muchos a muchos.



Ejemplo:

Supongamos que necesitamos almacenar la información de unas referencias bibliográficas o publicaciones, de libros, tesis, artículos que escriben los estudiantes de una universidad, y como se sabe un libro, tesis, artículo o cualquier publicación puede ser elaborada por más de una persona, y a su vez, una persona (docente, estudiante o investigador) puede tener más de una publicación de su autoría.



# Relaciones de muchos a muchos.



**REGLA DE MAPEO N:N de Modelo entidad-relación a modelo relacional:** En una relación de uno a mucho (N:N) se transforma en una tabla cuya llave primaria está compuesta por las llaves primarias de las otras dos tablas.

Como se puede ver en la ilustración, la tabla c) autor, nos dice que la persona Ana Lis, es autora de los libros de título Grandes hazañas y Pellentesque, también que el libro Pellentesque es escrito además de Ana Lis por Luis Dario y Jose Nicolas. Lo anterior haciendo referencia a las llaves primarias de las tablas a) **persona** y b) **publicacion**.

| id_persona | pais     | identificacion | correo           | nombres      | apellidos     | edad |
|------------|----------|----------------|------------------|--------------|---------------|------|
| 1          | colombia | 8.127.346      | ana@gmail.com    | Ana lis      | Mendez        | 34   |
| 2          | colombia | 212.394.762    | jose@gmail.com   | Jose Omar    | Ciano         | 28   |
| 3          | colombia | 545.232.546    | luis@gmail.com   | Luis Dario   | Marquez       | 21   |
| 4          | colombia | 125.446.732    | arm@gmail.com    | Armando      | Bronca Segura | 35   |
| 5          | colombia | 33.585.236.787 | victor@gmail.com | Victor José  | Diento Paz    | 24   |
| 6          | colombia | 2.343.554.768  | josen@gmail.com  | Jose Nicolás | Timado Ruiz   | 19   |
| 7          | perú     | 45.456.778     | andrea@gmail.com | Andrea       | Martinez      | 36   |

a) Tabla persona

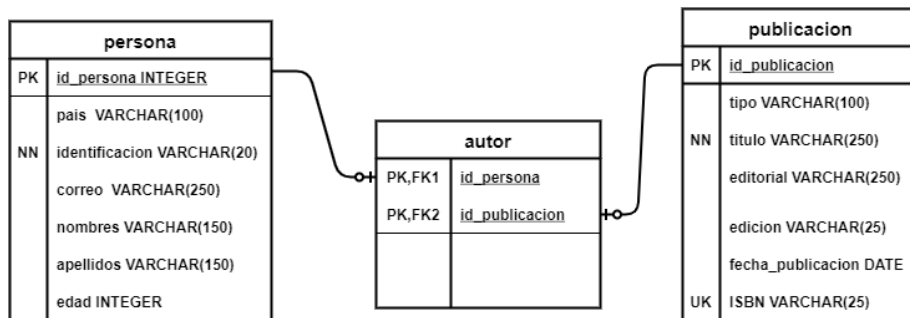
| id_publicacion | tipo     | título              | editorial | edición | fecha_publicacion |
|----------------|----------|---------------------|-----------|---------|-------------------|
| 1              | libro    | Grandes Hazañas     | Universit | 6ta     | 25/08/1998        |
| 2              | tesis    | Mediciones en me    | Denver    | -       | 26/08/2001        |
| 3              | libro    | Consectetur Sem     | ISFR      | 1ra     | 27/08/2004        |
| 4              | artículo | Consectetur         | Sodales   | -       | 31/08/2013        |
| 5              | libro    | Pellentesque        | Conubia   | 3ra     | 30/08/2010        |
| 6              | revista  | Praesent            | Nascetur  | -       | 22/06/2001        |
| 7              | revista  | Fermentum           | aptent    | -       | 01/09/2016        |
| 8              | libro    | El ruido y la furia | Vulputat  | 1a      | 03/09/2019        |
| 9              | libro    | Tempus mauris       | Efficitur | 5ta     | 12/04/2022        |

b) Tabla publicacion

| id_persona | id_publicacion |
|------------|----------------|
| 1          | 1              |
| 1          | 5              |
| 6          | 5              |
| 3          | 5              |
| 6          | 2              |

c) Tabla autor

# Relaciones de muchos a muchos.



# Relaciones de uno a uno



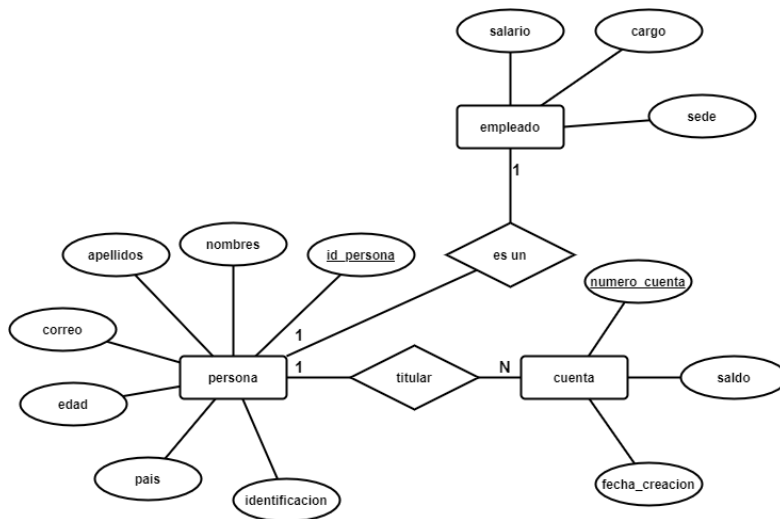
En una relación uno a uno, una fila de la tabla A solo puede tener una fila coincidente en la tabla B y viceversa. Se crea una relación uno a uno si ambas columnas relacionadas son claves principales o tienen restricciones únicas. Este tipo de relación no es común porque la mayoría de la información que está relacionada de esta manera estaría en una tabla. Puede usar una relación uno a uno por alguno de estos motivos:

- Dividir una tabla con muchas columnas.
- Aísle parte de una tabla por motivos de seguridad.
- Almacenar datos de corta duración y que podrían eliminarse fácilmente borrando la tabla periódicamente.
- Almacenar información que solo se aplique a un subconjunto de la tabla principal.

# Relaciones de uno a uno



Viviendo al ejemplo del banco, supongamos esta vez, la empresa necesita información sobre sus empleados, que, a la vez, pueden ser clientes de banco.



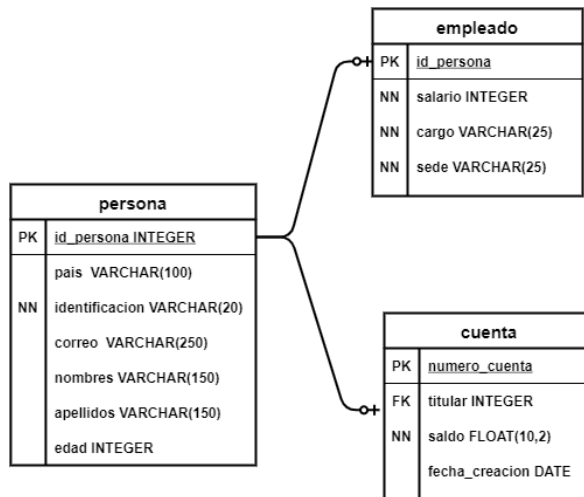


# Relaciones de uno a uno



## REGLA DE MAPEO 1:1 de Modelo entidad-relación a modelo relacional

En el momento de convertir la relaciono con multiplicidad 1:1 a tablas la relación uno a uno se convierte en una columna de la tabla empleado con restricción de unicidad (**U**nique **K**ey, o **P**rimary **K**ey) como se muestra en el siguiente diagrama relacional.





**G R A C I A S**

Línea de atención al ciudadano: 018000 910270  
Línea de atención al empresario: 018000 910682



@SENAcomunica

[www.sena.edu.co](http://www.sena.edu.co)

## EJERCICIOS

1. Se desea calcular la distancia recorrida (m) por un móvil que tiene velocidad constante (m/s) durante un tiempo  $T$  (Sg), considerar que es un MRU (Movimiento Rectilíneo Uniforme)
2. Se necesita obtener el promedio simple de un estudiante a partir de sus tres notas parciales.
3. Elaborar un algoritmo que solicite el número de respuestas correctas, incorrectas y en blanco, correspondientes a postulantes, y muestre su puntaje final considerando, que por cada respuesta correcta tendrá 4 puntos, respuestas incorrectas tendrá -1 y respuestas en blanco tendrá 0.
4. Elaborar un algoritmo que permita ingresar el número de partidos ganados, perdidos y empatados, por algún equipo en el torneo apertura, se debe de mostrar su puntaje total, teniendo en cuenta que por cada partido ganado obtendrá 3 puntos, empatado 1 punto y perdido 0 puntos.
5. Se requiere el algoritmo para elaborar la planilla de un empleado. Para ello se dispone de sus horas laboradas en el mes, así como de la tarifa por hora.
6. De un triángulo se tiene la longitud de la base y la longitud de la altura. Determine el valor de su área.
7. Se requiere llenar un tanque que tiene una capacidad de 50 metros cúbicos. Haga un algoritmo que imprima las horas que tarda en llenarse dicho tanque con una manguera que tiene una capacidad de  $L$  litros de agua por minuto.
8. Un amigo suyo acaba de iniciar un negocio de venta de zapatos. Por ahora sólo vende tres tipos de zapatos: sandalias, tenis y mocasines. Cada tipo de zapato lo adquiere a un costo distinto y para venderlos, supone una ganancia del 55%. Cuando un cliente llega debe comprar de los tres tipos de zapatos y la cantidad que desee de cada uno de ellos. El cliente tiene derecho a un 8% de descuento sobre la compra que realiza. Ayúdele a su amigo a crear un programa que, para un cliente dado, muestre su nombre, el valor de la venta sin descuento, el descuento, valor de la venta con descuento y valor de la venta incluyendo IVA (venta neta final).



**Tecnólogo de análisis y desarrollo de software ficha  
técnica 2455274**

**Fundamentos de programación estructurada y  
estructuras cíclicas GA3-220501093-AA2-EV01**

**LUZ ANGELICA SANCHEZ RAMIREZ**  
**Aprendiz**

**Jesus Andres Silva Plazas**  
**Instructor**

**Centro agroempresarial y desarrollo pecuario del  
Huila**

**2022**

 **1** |  **0**



## INTRODUCCION

Hemos leído el componente informativo donde tenemos que tener el análisis, diseño e implementación de cada problema para tener claro para poder desarrollarlo de la forma efectiva para que sea resuelto, fácil y entendido de la mejor forma, vamos a plantear los siguientes ejercicios:

1. **Algoritmo que a partir de una fecha de nacimiento y una fecha actual determine la edad en años actual de una persona.**

### *Análisis:*

- **Formular el problema:** consiste desde la fecha de nacimiento y fecha actual en edad en años actuales
- **Resultados esperados:** se espera que el algoritmo funcione sin ningún error al ser ejecutado
- **Datos disponibles:** son como fecha de nacimiento como día, mes y año y con la fecha actual igualmente como mes día y año
- **Restricciones:** ninguna
- **Procesos necesarios:** solicitar que el usuario indique los respectivos datos como día, mes y año

### *Diseño:*

## **Algoritmo Años\_Actual\_Persona**



**Definir Edad, DiaActual, MesActual,  
AñoActual Como Entero**

**Definir DiaNacim, MesNacim,  
AñoNacim Como Entero**

**Escribir "Por favor escriba el dia de la fecha de  
nacimiento "**

**leer DiaNacim**

**Escribir "Por favor escriba el mes de la fecha  
de nacimiento "**

**leer MesNacim**

**Escribir "Por favor escriba el año de la fecha  
de nacimiento "**

**leer AñoNacim**

**Escribir "Por favor escriba la fecha en dia  
actual"**

**leer DiaActual**

**Escribir "Por favor escriba la fecha mes actual  
"**

**leer MesActual**

**Escribir "Por favor escriba la fecha año  
actual"**

**leer AñoActual**

**Edad = AñoActual - AñoNacim**

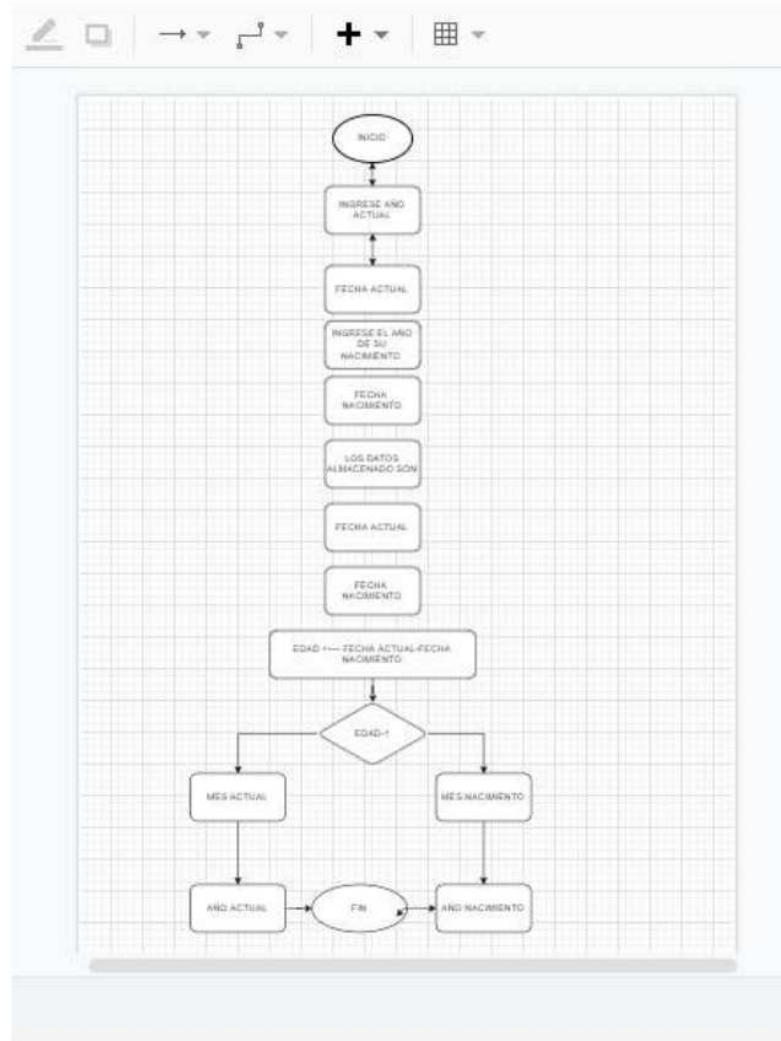
**si MesNacim > MesActual Entonces**

**Edad = Edad-1**

**SiNo**



```
    si MesNacim == MesActual
Entonces
    si DiaNacim > DiaActual
Entonces
    Edad = Edad-1
FinSi
si DiaNacim == DiaActual Entonces
    Escribir "¡Oh!Felicidades por tu
cumpleaños"
FinSi
FinSi
FinSi
Escribir "Tu edad actual es: ",edad, "años"
FinAlgoritmo
```



## 2. Algoritmo que permita determinar si un año indicado es o no un año bisiesto.

*Análisis:*

- **Formular el problema:** consiste diferenciar que año es Bisiesto y que año no es bisiesto





- **Resultados esperados:** se espera que el algoritmo funcione sin ningún error al ser ejecutado y sin ningún problema
- **Datos disponibles:** año actual, años pasados o futuros
- **Restricciones:** ninguna
- **Procesos necesarios:** solicitar que el usuario indique los respectivos datos como año sea vigente, pasado o futuro para obtener la información clara

*Diseño:*

#### **Algoritmo Año\_Bisiesto**

**Escribir " Por favor dígame si el año es Bisiesto o No Bisiesto"**

**Leer Año**

**si (Año mod 4=0 Y Año mod 100<>0) Entonces**

**Escribir "El Año " Año " Es Bisiesto"**

**sino**

**si (Año mod 400 =0 y Año mod 100=0)**

**Escribir "El Año " Año "Es Bisiesto"**

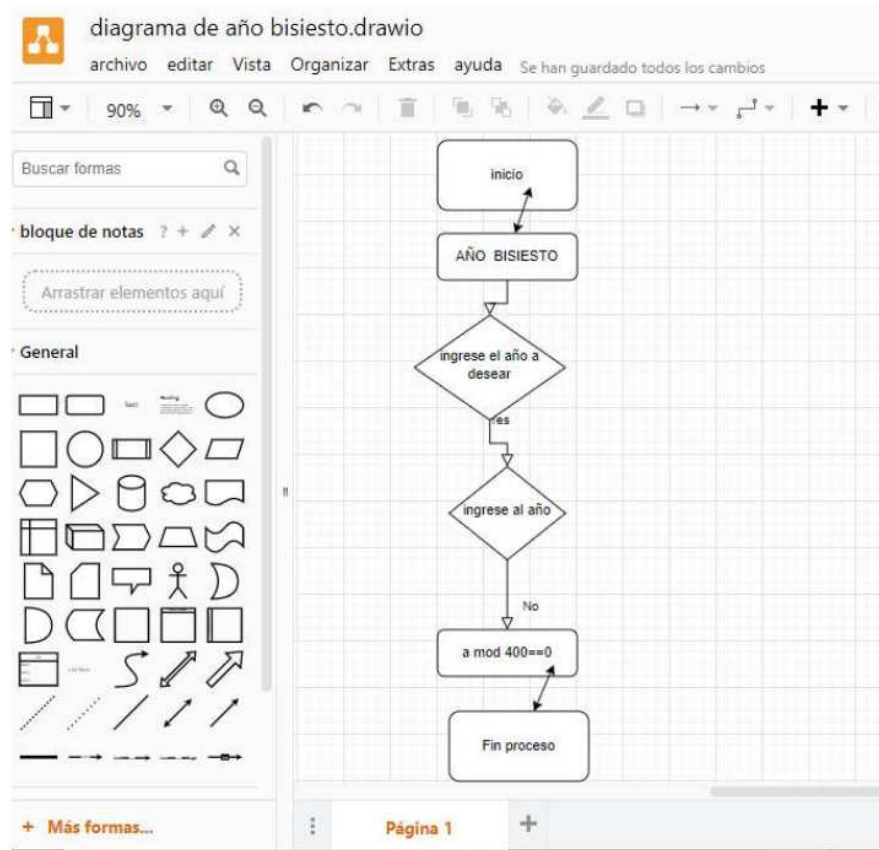
**sino**

**Escribir "El Año " Año " No es Bisiesto"**

**FinSi**

**FinSi**

**FinAlgoritmo**





## PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL FORMATO GUÍA DE APRENDIZAJE

### 1. IDENTIFICACIÓN DE LA GUÍA DE APRENDIZAJE

- **Denominación del programa de formación:** Análisis y Desarrollo de Software
- **Código del programa de formación:** 228118
- **Nombre del proyecto:** Construcción de software integrador de tecnologías orientadas a servicios
- **Fase del proyecto:** Planeación
- **Actividad de proyecto:** Estructurar procesos lógicos para la construcción de algoritmos.
- **Competencias:**

#### **Técnicas:**

- 220501093- Evaluar requisitos de la solución de software de acuerdo con metodologías de análisis y estándares.

#### **Claves:**

#### **Transversales:**

- **Resultados de aprendizaje a alcanzar**

#### **Técnicos:**

- 220501093-03 - Desarrollar procesos lógicos a través de la implementación de algoritmos.

#### **Claves:**

#### **Transversales:**

- **Duración de la Guía:** 240 horas

**Técnicas:** 96 horas.

**Claves:** 48 horas.

**Transversales:** 96 horas.

### 2. PRESENTACIÓN

Estimado aprendiz, el SENA extiende una cordial bienvenida a este proceso de formación, en el cual se desarrollarán los temas que fundamentan todo lo correspondiente a la programación de software, es decir las bases algorítmicas relacionadas con las estructuras secuenciales, condicionales y cíclicas sobre la que se sustentan la construcción de la lógica que se usará para dar soluciones a problemas utilizando lenguajes de programación.

Permanece atento, y ¡**APROVECHA ESTA OPORTUNIDAD DE APRENDIZAJE!**



### 3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

En este apartado se describirán las actividades de aprendizaje planteadas para el desarrollo de la competencia, así como la indicación respectiva para que logre la elaboración de las evidencias en un total de 240 horas.

De este programa de formación, en cada una de las actividades se detallan las evidencias a entregar. La entrega de las evidencias se realizará durante (6) semanas planeadas para esta guía. En el siguiente cuadro se observa para la competencia, el momento y las evidencias a entregar según las actividades de aprendizaje asociadas a la misma.

| Competencia                                                                                            | Semana 1                                                                                                                                                                                                                                             | Semana 2                                                                                                                 | Semana 3                                                                                            | Semana 4                                                                                                                      | Semana 5                                                                                                            | Semana 6                                                                                                                          |
|--------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|
|                                                                                                        | Evidencias                                                                                                                                                                                                                                           |                                                                                                                          |                                                                                                     |                                                                                                                               |                                                                                                                     |                                                                                                                                   |
| 220501093 –<br>Evaluar requisitos de la solución de acuerdo con metodologías de análisis y estándares. | Evidencia de conocimiento:<br>Bases conceptuales de lógica proposicional.<br>GA3-220501093-AA1-EV01<br><br>Evidencia de producto:<br>Taller de resolución de problemas de algoritmos en pseudocódigo y diagramas de flujo.<br>GA3-220501093-AA1-EV02 | Evidencia de conocimiento:<br>Fundamentos de programación estructurada y estructuras cíclicas.<br>GA3-220501093-AA2-EV01 | Evidencia de desempeño:<br>Foro temático:<br>Estructuras de programación.<br>GA3-220501093-AA2-EV02 | Evidencia de Producto:<br>Taller aplicando funciones y procedimientos en la solución de algoritmos.<br>GA3-220501093-AA2-EV03 | Evidencia de conocimiento:<br>Bases teóricas de estructuras de almacenamiento en memoria.<br>GA3-220501093-AA3-EV01 | Evidencia de desempeño:<br>Resolución a problemas algorítmicos aplicando estructuras de almacenamiento.<br>GA3-220501093-AA3-EV02 |
| Horas por semana                                                                                       | 16                                                                                                                                                                                                                                                   | 16                                                                                                                       | 16                                                                                                  | 16                                                                                                                            | 16                                                                                                                  | 16                                                                                                                                |

El proceso de formación se centra en el aprendizaje basado en problemas y posibilitará la realización de una gran cantidad de ejercicios prácticos, los cuales permitirán desarrollar habilidades de análisis para asociar las características de los problemas con las estructuras algorítmicas requeridas para el desarrollo de una solución.



### **3.1. Actividades de aprendizaje de la competencia: - Evaluar requisitos de la solución de software de acuerdo con metodologías de análisis y estándares (220501093).**

Esta competencia se centra en la realización de actividades de evaluación de las mejores estrategias que permitan abordar cada uno de los requisitos de la solución de software de una manera organizada y efectiva, específicamente para esta guía se abordarán las bases requeridas para la construcción de algoritmos los cuales son la esencia de la programación

#### **3.1.1. Actividad de aprendizaje GA3-220501093 - AA1: Realizar actividades para el desarrollo del pensamiento algorítmico.**

En esta actividad se estudiarán los conceptos básicos asociados al concepto de algoritmo, se abordará la definición formal de algoritmo, las características del pensamiento algorítmico y cómo esto nos ayuda a plantear soluciones a problemas reales y problemas asociados a la programación. Luego se abordan cada uno de los componentes requeridos para abordar el problema e identificar cada uno de los elementos de este.

**Duración:** 14 horas

**Materiales de formación a consultar:** para el desarrollo de esta actividad es importante la lectura y análisis del material de formación: "Introducción a la algoritmia"

**Evidencias:** a continuación, se describen las acciones y las correspondientes evidencias que conforma la actividad de aprendizaje:

- **Evidencia GA3-220501093-AA1-EV01: Bases conceptuales de lógica proposicional.**

Esta evidencia se centra en consolidar los conceptos relacionados con los componentes de lógica matemática indispensables para el desarrollo de estructuras de control algorítmicas

Para su desarrollo es importante la lectura del componente formativo. Se resolverá un cuestionario que se emplea como técnica de evaluación. Las preguntas del cuestionario pueden ser:

- ✓ De carácter formativo, las cuales permiten determinar el nivel alcanzado de desarrollo de la competencia durante la formación.
- ✓ De carácter evaluativo, que determinan el dominio y el nivel máximo alcanzado en la competencia.

#### **Lineamientos generales para la entrega de la evidencia:**

- **Duración:** 30 minutos.
- Para responder el cuestionario remítase al área de la actividad correspondiente y acceda al espacio: **Bases conceptuales de lógica proposicional. GA3-220501093-AA1-EV01.**
- **Evidencia GA3-220501093-AA1-EV02: Taller de resolución de problemas de algoritmos en pseudocódigo y diagramas de flujo.**



Teniendo en cuenta las recomendaciones para el análisis de problemas desarrollado durante el componente formativo, en esta actividad se plantean algunos problemas los cuales deberá leer detenidamente para luego identificar los elementos de entrada y los resultados esperados. Adicionalmente se introduce el tema de diagramas de flujo para lo cual deberá realizar una investigación corta respondiendo algunos lineamientos establecidos en esta actividad

### **Elementos para tener en el desarrollo del taller:**

#### **Sección 1 Taller**

Teniendo en cuenta los siguientes problemas identificar los posibles elementos de entrada y resultados esperados de cada uno:

- Se desea elaborar un algoritmo que permita identificar la cantidad de dólares equivalentes a una cantidad de pesos colombianos.
- Se desea elaborar un algoritmo que permita determinar la temperatura equivalente en grados centígrados a la cantidad de grados Fahrenheit actuales en la ciudad de New York.
- Suponiendo que nos encontramos descansando en una nuestra casa en una ciudad de Colombia requiero hacer un plan detallado para llegar a mi sitio de trabajo a tiempo el día siguiente.
- Suponiendo que tengo habilidades en la elaboración de comida necesito elaborar un arroz con pollo para 5 personas.

#### **Respecto a los problemas planteados:**

- Para precisar los elementos de entrada de cada una de las situaciones planteadas no olvide las preguntas guía:
  - ¿Qué información es importante y necesaria para resolver el problema?
  - ¿Qué información no es importante y se puede prescindir?
  - ¿Cuáles son los datos de entrada conocidos?
  - ¿Cuál es la incógnita o que debemos calcular?
  - ¿Los datos se pueden agrupar en categorías?
  - ¿Qué información adicional hace falta para resolver el problema?
- Para precisar los resultados esperados se recomienda utilizar las preguntas guía:
  - ¿Qué información están solicitando?
  - ¿En qué formato se debe entregar esta información?

#### **Sección 2 Taller**

Elabore una investigación corta usando los materiales disponibles en la biblioteca o internet respecto a los fundamentos para la resolución de problemas con algoritmos, selecciones por lo menos tres fuentes que le permitan resolver las siguientes preguntas:

- Definición de diagrama de flujo.
- Símbolos mas importantes.
- Seleccionar uno de los problemas de la sección 1 y representarlo en su equivalente diagrama de flujo.

#### **Respecto a los diagramas de flujo:**

- Referenciar las fuentes de información utilizadas para resolver cada una de las preguntas utilizando formato APA.

### **Lineamientos generales para la entrega de la evidencia:**

- **Productos para entregar:** Un documento con la solución del taller propuesto.
- **Extensión:** Libre.



- **Formato:** PDF o Word.
- Para hacer el envío de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **Taller de resolución de problemas de algoritmos en pseudocódigo y diagramas de flujo. GA3-220501093-AA1-EV02.**

### **3.1.2. Actividad de aprendizaje GA3-220501093-AA2 - Desarrollar soluciones algorítmicas a través de estructuras de control.**

En esta actividad se estudiarán algunas formas de representación de algoritmos y se introducen algunos conceptos clave como: constantes, variables, acumuladores, contadores, operadores, jerarquía de los operadores. Finalmente se abordan las estructuras de control secuencial y herramientas existentes para la creación y pruebas de algoritmos.

**Duración:** 40 horas

**Materiales de formación a consultar:** para el desarrollo de esta actividad es importante la lectura y análisis del material de formación: "**Análisis y solución de problemas aplicando algoritmos**"

**Evidencias:** a continuación, se describen las acciones y las correspondientes evidencias que conforma la actividad de aprendizaje:

- **Evidencia GA3-220501093-AA2-EV01: Fundamentos de programación estructurada y estructuras cíclicas.**

Esta evidencia se centra en consolidar los conceptos relacionados con las metodologías de algoritmos y sus tres fases: análisis, diseño e implementación. Respecto a la implementación se aborda los pseudocódigos y diagramas de flujo.

Para su desarrollo es importante la lectura del componente formativo. Se resolverá un cuestionario que se emplea como técnica de evaluación. Las preguntas del cuestionario pueden ser:

- ✓ De carácter formativo, las cuales permiten determinar el nivel alcanzado de desarrollo de la competencia durante la formación.
- ✓ De carácter evaluativo, que determinan el dominio y el nivel máximo alcanzado en la competencia

#### **Lineamientos generales para la entrega de la evidencia:**

- **Duración:** 30 minutos.
- Para responder el cuestionario remítase al área de la actividad correspondiente y acceda al espacio: **Fundamentos de programación estructurada y estructuras cíclicas. GA3-220501093-AA2-EV01.**

- **Evidencia GA3-220501093-AA2-EV02: Foro temático: Estructuras de programación.**

Teniendo en cuenta las diferentes estructuras de control y su respectiva forma de representación en diagrama de flujo y pseudocódigo, realizar su participación en el foro argumentando cual de las dos



representaciones le resulta más conveniente para cada una de las estructuras de control trabajadas. Tome en cuenta las siguientes preguntas orientadoras:

- ✓ ¿Qué beneficios tiene una representación en comparación con la otra?
- ✓ ¿Respecto a las herramientas existentes, cuál tiene más representación?
- ✓ ¿Cuál sería la más fácil para entender las estructuras algorítmicas?
- ✓ ¿Cuál permitirá una transición más fácil hacia la codificación?

Realice seguimiento al foro y responda a un comentario crítico-reflexivo de acuerdo con la intervención que hayan realizado sus compañeros demostrando construcción de conocimiento frente a los planteamientos que cada uno proponga.

#### **Lineamientos generales para la entrega de la evidencia:**

- **Productos para entregar:** participación en el foro.
  - Para hacer el envío de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **Foro temático: Estructuras de programación. GA3-220501093-AA2-EV02.**
- 
- **Evidencia GA3-220501093-AA2-EV03: Taller aplicando funciones y procedimientos en la solución de algoritmos.**

En esta actividad deberá aplicar todos los conocimientos adquiridos a lo largo del componente formativo para dar solución a problemas utilizando la notación de pseudocódigo y diagramas de flujo usando las herramientas establecidas por el instructor. Tenga en cuenta los diferentes tipos de estructuras de control básicas: secuenciales, condicionales y repetitivas.

La descripción de cada uno de los enunciados a resolver se encuentra en el **Anexo 1**.

#### **Elementos para tener para la solución de este taller:**

- Cada problema se debe solucionar usando las dos notaciones (pseudocódigo y diagrama de flujo).
- Debe utilizar la herramienta o el modelo de representación de cada notación sugerida por el instructor.

#### **Lineamientos generales para la entrega de la evidencia:**

- **Productos para entregar:** Un único archivo comprimido con el nombre del estudiante donde se encuentre todas las soluciones a cada uno de los ejercicios presentados en el anexo 1.
- **Formato:** ZIP.
- Para hacer el envío de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **Taller aplicando funciones y procedimientos en la solución de algoritmos. GA3-220501093-AA2-EV03.**

### **3.1.3. Actividad de aprendizaje GA3-220501093-AA3 - Solucionar planteamientos aplicando el concepto de arreglos (vectores y matrices).**

Esta actividad se centra en la introducción a los lenguajes de programación abordando conceptos básicos, características de los lenguajes, sus tipos y el proceso de estación. Luego se identifican los tipos de datos, operadores, comentarios y expresiones admitidas por el lenguaje de programación seleccionado.





Finalmente se repasan cada una de las estructuras de control y se incluye adicionalmente las estructuras de datos tipo vector y matriz.

**Duración:** 42 horas

**Materiales de formación a consultar:** para el desarrollo de esta actividad es importante la lectura y análisis del material de formación: " **Codificación de Algoritmos**"

**Evidencias:** a continuación, se describen las acciones y las correspondientes evidencias que conforma la actividad de aprendizaje:

- **Evidencia GA3-220501093-AA3-EV01: Bases teóricas de estructuras de almacenamiento en memoria.**

Esta evidencia se centra en consolidar los conceptos básicos relacionados con los lenguajes de programación, entornos de codificación e instalación y la sintaxis del lenguaje de JavaScript.

Para su desarrollo es importante la lectura del componente formativo. Se resolverá un cuestionario que se emplea como técnica de evaluación. Las preguntas del cuestionario pueden ser:

- ✓ De carácter formativo, las cuales permiten determinar el nivel alcanzado de desarrollo de la competencia durante la formación.
- ✓ De carácter evaluativo, que determinan el dominio y el nivel máximo alcanzado en la competencia

**Lineamientos generales para la entrega de la evidencia:**

- **Duración:** 30 minutos.
- Para responder el cuestionario remítase al área de la actividad correspondiente y acceda al espacio: **Bases teóricas de estructuras de almacenamiento en memoria. GA3-220501093-AA3-EV01.**
- **Evidencia GA3-220501093-AA3-EV02: Resolución a problemas algorítmicos aplicando estructuras de almacenamiento.**

En esta actividad deberá aplicar todos los conocimientos adquiridos a lo largo del componente formativo para dar solución a problemas utilizando todas las estructuras de control requeridas y el lenguaje JavaScript.

La descripción de cada uno de los enunciados a resolver se encuentra en el **Anexo 2**.

**Elementos para tener para la solución de este taller:**

- Cada problema se debe solucionar usando el lenguaje JavaScript.
- Si bien existen muchas formas posibles de resolver un problema se recomienda utilizar las estructuras que permitan el desarrollo de acciones de una manera simple y eficiente.

**Lineamientos generales para la entrega de la evidencia:**

- **Productos para entregar:** Un único archivo comprimido con el nombre del estudiante donde se encuentre todas las soluciones a cada uno de los ejercicios presentados en el anexo 2.
- **Formato:** ZIP.



- Para hacer el envío de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **Resolución a problemas algorítmicos aplicando estructuras de almacenamiento. GA3-220501093-AA3-EV02.**

#### 4. ACTIVIDADES DE EVALUACIÓN

Tome como referencia la técnica e instrumentos de evaluación citados en la guía de Desarrollo Curricular

| Evidencias de Aprendizaje                                                                                                                            | Criterios de Evaluación                                                                                          | Técnicas e Instrumentos de Evaluación                      |
|------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------|
| <b>Evidencia de conocimiento:</b><br><br>Bases conceptuales de lógica proposicional.<br>GA3-220501093-AA1-EV01                                       | Soluciona problemas de lógica proposicional incorporando habilidades propias en el oficio como programador.      | <b>Cuestionario:</b><br><br>IE- GA3-220501093-AA1-EV01     |
| <b>Evidencia de producto:</b><br><br>Taller de resolución de problemas de algoritmos en pseudocódigo y diagramas de flujo.<br>GA3-220501093-AA1-EV02 |                                                                                                                  | <b>Lista de chequeo:</b><br><br>IE- GA3-220501093-AA1-EV02 |
| <b>Evidencia de conocimiento:</b><br><br>Fundamentos de programación estructurada y estructuras cíclicas.<br>GA3-220501093-AA2-EV01                  | Crea soluciones a problemas mediante algoritmos que incluyen estructuras secuenciales, condicionales y cíclicas. | <b>Cuestionario:</b><br><br>IE- GA3-220501093-AA2-EV01     |
| <b>Evidencia de desempeño:</b><br><br>Foro temático: Estructuras de programación.<br>GA3-220501093-AA2-EV02                                          |                                                                                                                  | <b>Lista de chequeo:</b><br><br>IE- GA3-220501093-AA2-EV02 |
| <b>Evidencia de Producto:</b><br><br>Taller aplicando funciones y procedimientos en la solución de algoritmos.<br>GA3-220501093-AA2-EV03             |                                                                                                                  | <b>Lista de chequeo:</b><br><br>IE- GA3-220501093-AA2-EV03 |



|                                                                                                                                              |                                                                                                                                                                                               |                                                            |
|----------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------|
| <b>Evidencia de conocimiento:</b><br><br>Bases teóricas de estructuras de almacenamiento en memoria.<br>GA3-220501093-AA3-EV01               | Crea funciones y procedimientos en la solución de algoritmos para ordenar y simplificar los códigos.<br><br>Manipula arreglos en diferentes dimensiones para dar solución a problemas reales. | <b>Cuestionario:</b><br><br>IE- GA3-220501093-AA3-EV01     |
| <b>Evidencia de desempeño:</b><br><br>Resolución a problemas algorítmicos aplicando estructuras de almacenamiento.<br>GA3-220501093-AA3-EV02 | Administra la información de los usuarios por medio de archivos, permitiendo el ingreso, modificación y eliminación de los datos.                                                             | <b>Lista de chequeo:</b><br><br>IE- GA3-220501093-AA3-EV02 |

## 5. GLOSARIO DE TÉRMINOS

**Archivo ejecutable:** archivo en código binario que se genera a partir de la compilación de un programa y el cual representa el programa a ser usado por un cliente.

**JavaScript:** Es un lenguaje de programación interpretado orientado a objetos.

**Metodología:** ciencia que consta de métodos y técnicas, que se aplican sistemáticamente durante un proceso de investigación o para solucionar una problemática.

## 6. REFERENTES BIBLIOGRÁFICOS

- Aguilar, J. L. (2008). *Fundamentos de programación (Spanish Edition)* (4th ed.). McGraw-Hill Interamericana de España S.L.
- M. (2021a). *Introducción a la programación estructurada en C (Spanish Edition)* (#REF! ed.). PEARSON.
- Sierra, F. J. C. (2000). *C/C++ curso de programación*. Alianza Editorial.
- Vázquez, J. (2012). *Análisis y diseño de algoritmos*. Red tercer milenio, Tlalnepantla, Estado de México. ISBN: 978-607-733-053-0



## 7. CONTROL DEL DOCUMENTO

|                   | Nombre                    | Cargo            | Dependencia                                       | Fecha          |
|-------------------|---------------------------|------------------|---------------------------------------------------|----------------|
| <b>Autor (es)</b> | Jonathan Guerrero Astaiza | Experto temático | Centro de Teleinformática y Producción Industrial | Noviembre 2021 |

## 8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

|                   | Nombre | Cargo | Dependencia | Fecha | Razón del Cambio |
|-------------------|--------|-------|-------------|-------|------------------|
| <b>Autor (es)</b> |        |       |             |       |                  |

# INTRODUCCIÓN A LA ALGORITMIA

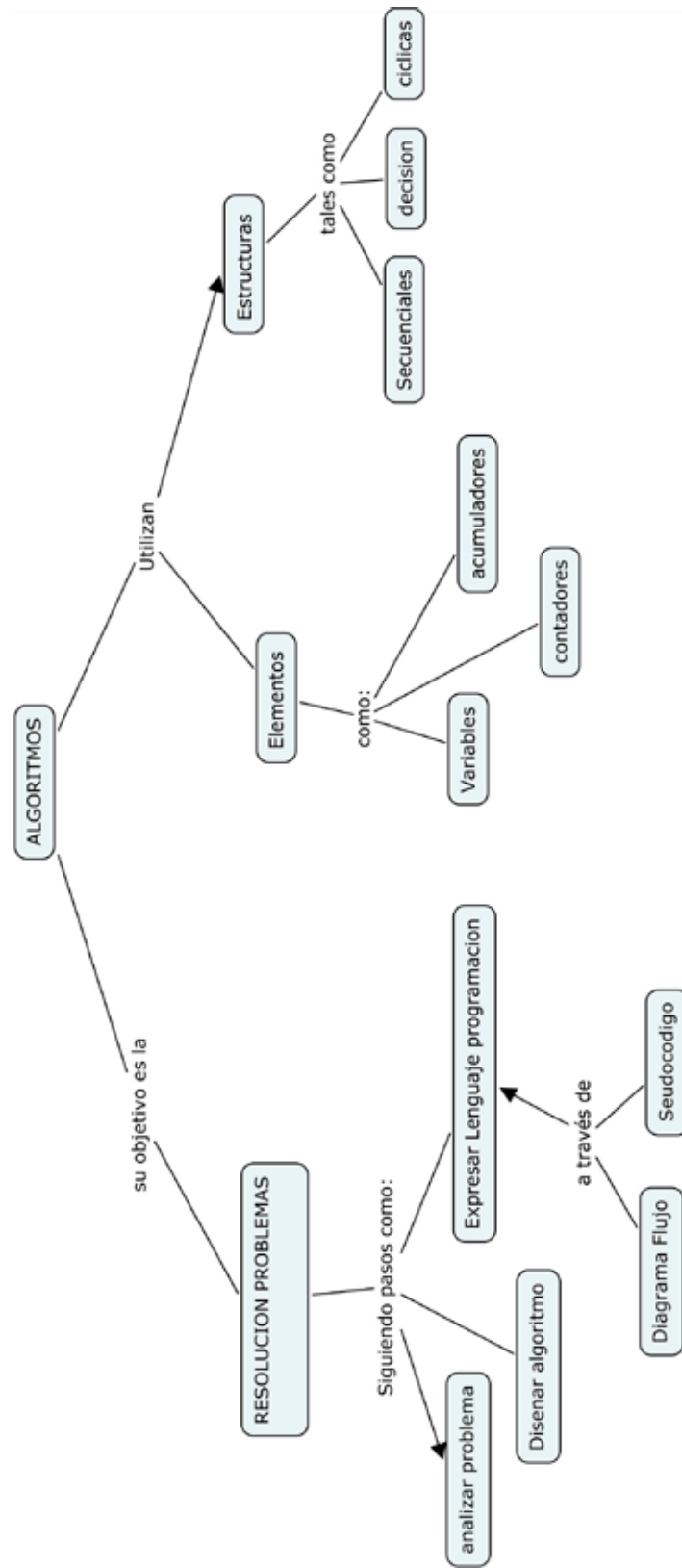
---

Estructura de contenidos

|                                                                                          |    |
|------------------------------------------------------------------------------------------|----|
| INTRODUCCIÓN .....                                                                       | 3  |
| 1. CONCEPTO DE ALGORITMOS .....                                                          | 3  |
| 2. UN POCO DE HISTORIA .....                                                             | 3  |
| 3. RESOLUCIÓN DE PROBLEMAS CON ALGORITMOS.....                                           | 4  |
| 4. SOLUCIÓN DE PROBLEMAS CON DIAGRAMAS DE FLUJO Y LENGUAJE<br>NATURAL (SEUDOCÓDIGO)..... | 14 |
| 5. ESTRUCTURAS ALGORÍTMICAS O DE PROGRAMACION.....                                       | 18 |



# Mapa conceptual INTRODUCCIÓN A LA ALGORITMIA



# INTRODUCCIÓN

Nuestro actuar como personas se basa en obedecer por medio de ordenes Secuenciales y lógicas a nuestro cerebro; hoy en día un computador realiza tareas y maneja datos obedeciendo secuencias de pasos lógicos para lo cual ha sido programado a través de algoritmos, los cuales permiten solucionar problemas por medio de la escritura secuencial (paso a paso) de lenguaje natural y organizado, para luego llevarlos a un computador mediante un lenguaje de programación.

Es muy importante que usted como aprendiz logre desarrollar y despertar su mentalidad algorítmica, ya que durante todo su proceso estará inmerso en aprender a solucionar problemas durante la etapa lectiva y esto se logra por medio del diseño, creación e implementación de un algoritmo.

## 1. CONCEPTO DE ALGORITMOS

Un algoritmo se puede definir como un conjunto de instrucciones que conducen a la solución de un problema determinado, las cuales deben estar relacionadas lógicamente y ordenadamente.

## 2 UN POCO DE HISTORIA

Los algoritmos fueron creados con la necesidad de resolver problemas matemáticos, sus inicios se dan con la creación de las reglas de sumar, multiplicar y dividir números decimales (Mohammed al-Khowarizmi-matemático persa), de ahí nace el nombre de la "Algoritmo", Su apellido fue traducido al latín como Algorismus y posteriormente paso al español como Algoritmo; más adelante Euclides, gran matemático griego utilizó algoritmos cuando inventó el método para encontrar el máximo común divisor de dos números; y por último se puede decir que el gran padre de la algoritmia es Niklaus Wirth el cual tuvo el gran invento de Pascal.

Los algoritmos fueron creados con el fin de utilizarlos para resolver problemas, para luego llevarlos a un computador mediante un lenguaje

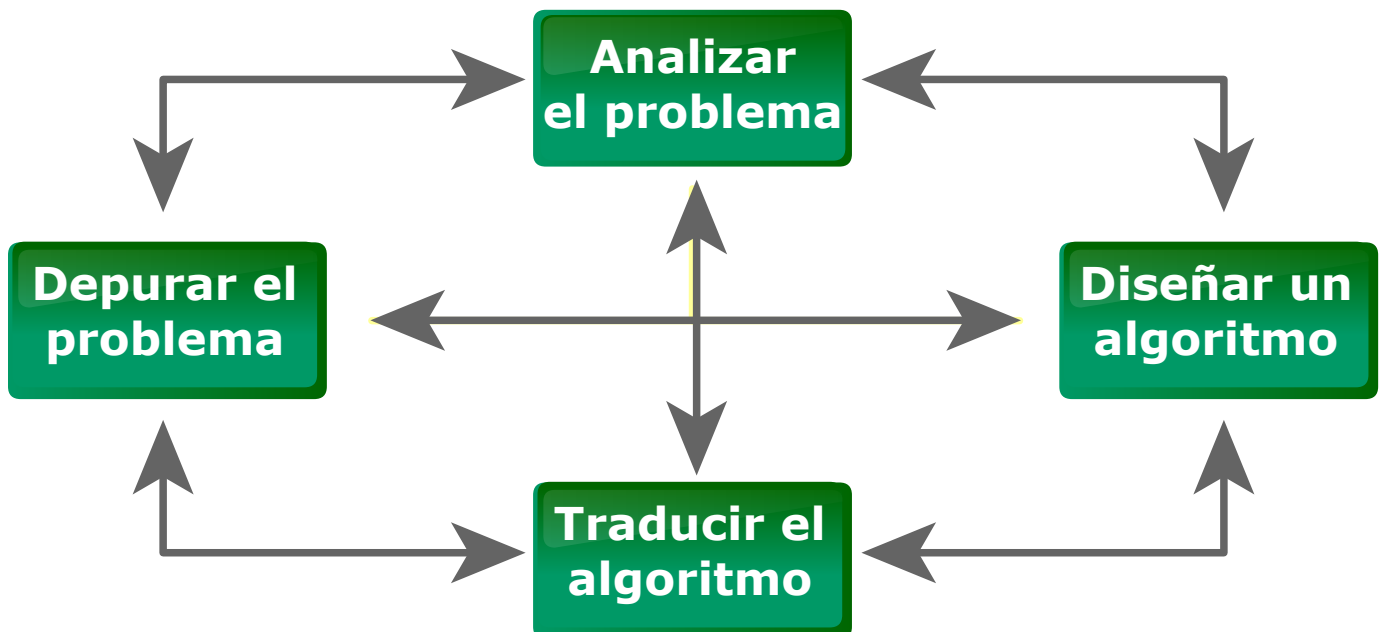


de programación; para realizar un algoritmo se debe establecer una metodología de desarrollo, una de estas es la llamada resolución de problemas, la cual permite realizar un diseño organizado del algoritmo.

### 3 RESOLUCIÓN DE PROBLEMAS CON ALGORITMOS

Para solucionar un problema mediante un algoritmo es necesario seguir un orden, uno de los primeros pasos es el diseño previo de un algoritmo, con la resolución de problemas podemos seguir este orden de la siguiente manera:

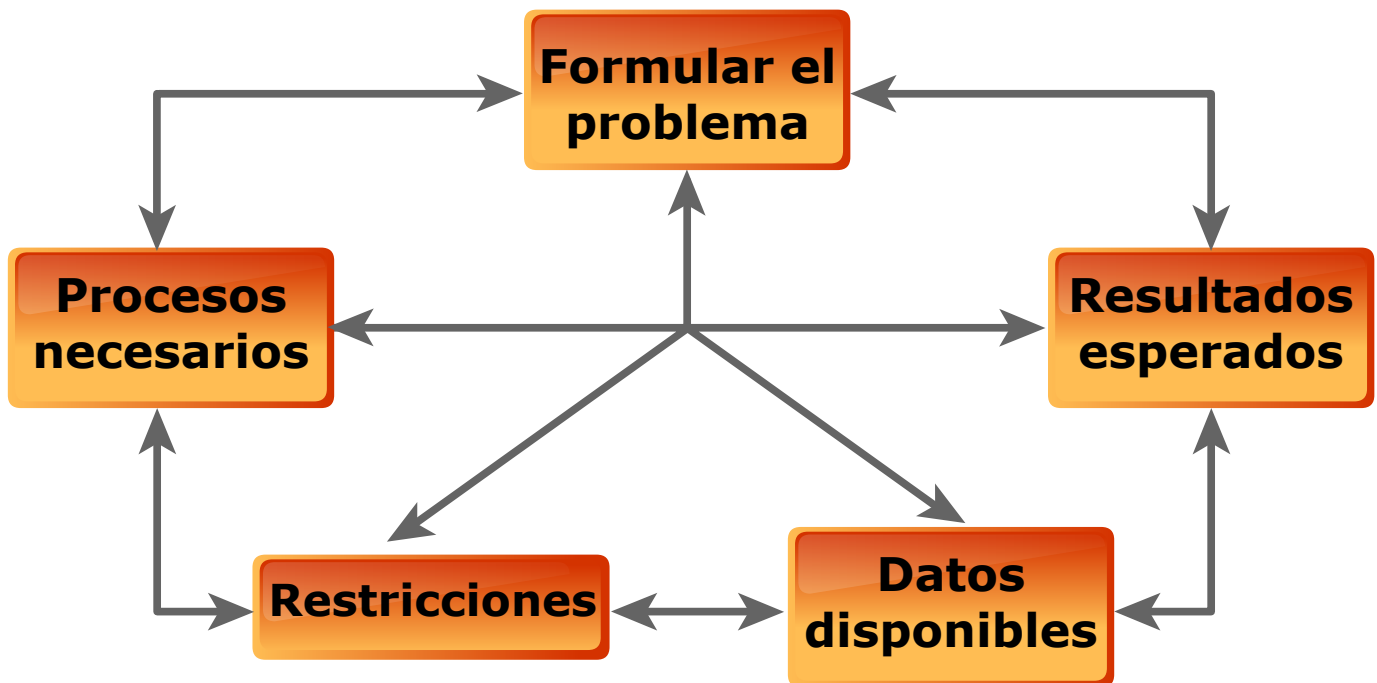
- Analizar el problema (entender el problema)
- Diseñar el algoritmo: En este paso se describe la secuencia ordenada de pasos que conduce a la solución del problema citado (diagrama de flujo o pseudocódigo).
- Expresar el algoritmo: el algoritmo se debe expresar como un programa en un lenguaje de programación adecuado. (Fase de codificación.)
- Ejecución y validación: Se pone en ejecución el programa realizado en un computador.





### 3.1 Analizar el problema

La finalidad como ya se ha mencionado de un algoritmo es resolver un problema, lo primero y esencial es analizar el problema para lograr un entendimiento claro



**Reformular el problema:** Si el problema que se ha planteado no se encuentra bien formulado, redactado o no se entiende al momento de leerse, debe reformularlo de tal manera que usted lo entienda y tenga claro la situación problema a solucionar).

**Resultados Esperados:** Debe especificar, describir y/o escribir los resultados que espera, por ejemplo (cuál es el producto final que se quiere tener para dar solución al problema, cual es la información a la que se necesita llegar, que se espera del problema citado)

**Datos Disponibles:** Identificar la información disponible se resuelve haciéndose las siguientes preguntas: que información es importante o relevante para solucionar el problema? Cuáles son los datos de entrada?, cual es la incógnita?, que información me falta para resolver el problema?,

es decir todos los datos que tengo disponibles que me permitan solucionar el problemas.

**Restricciones:** Determina las condiciones que plantea el problema para lograr el resultado, lo que está permitido, lo prohibido.

**Procesos necesarios:** En esta fase debe definir los procesos para poder convertir la información disponible, en resultados esperados que den solución al problema ya que se determinan los procesos que se necesitan, las formulas a utilizar y el orden de lo que se debe realizar.

### Ejemplo 1

Carlos Andrés, se encuentra ahorrando para comprar una patineta que vale 55.000 pesos. Su papá le ha dado para sus gastos 5.000 pesos durante 7 semanas. Por lavar el auto de su tío tres veces recibió 8.000 pesos. Su hermano Juan Antonio ganó 10.000 pesos por hacer los mandados de su mamá y 4.000 por sacar a pasear el perro. *¿Carlos Andrés tiene ahorrado el dinero suficiente para comprar la patineta o aún le falta?*

**Formular el problema:** Ya se encuentra claramente planteado, pero si no entiende la redacción puede redactarlo a su manera, hasta es posible realizar un resumen de lo planteado.

**Resultados esperados:** Saber si Carlos Andrés tiene o no tiene ahorrado el dinero para comprar su patineta, la cual cuesta 55.000 pesos.

**Datos disponibles:** Los ingresos de Carlos Andrés 5.000 pesos por 7semanas + 8.000 pesos, los datos irrelevantes serian: Los 10.000 y 4.000 pesos qué ganó el hermano ya que no aportan información para la solución de este problema y se pueden omitir.

**Restricciones:** no se encuentra ninguna.



**Procesos:** Calcular el valor ahorrado por Carlos para saber si le alcanza para comprar la patineta.

$$\text{ValorAhorrado} = 5000 + 8000 = 13.000$$

Es decir no le alcanza para la patineta.

## Ejemplo 2

Se necesita calcular el área de un triángulo rectángulo cuya Base mide 3 cm, la Altura 4 cm y la Hipotenusa 5 cm.

**Formular el problema:** Ya se encuentra claramente planteado, pero si no entiende la redacción puede redactarlo a su manera, hasta es posible realizar un resumen de lo planteado.

**Resultados esperados:** El área de un triángulo rectángulo.

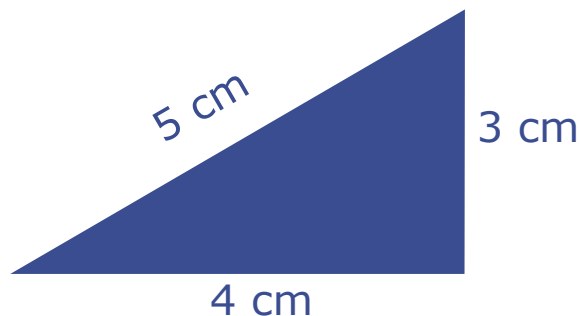
**Datos disponibles:** Base, Altura, Hipotenusa, tipo de triángulo.

La incógnita es el área y todos los valores son constantes. El valor de la hipotenusa se puede omitir.

El aprendiz debe preguntarse si sus conocimientos actuales de matemáticas le permiten resolver este problema; de no ser así, debe plantear una estrategia para obtener los conocimientos requeridos.

**Determinar las restricciones:** Utilizar las medidas dadas, y saber que se debe aplicar la fórmula del área de un triángulo rectángulo.

**Procesos necesarios:** Guardar en dos variables los valores de Base y Altura; Guardar en una constante el divisor 2; aplicar la fórmula  $\text{área} = \text{base} * \text{altura} / 2$ ; comunicar el resultado (área).



### 3.2 Diseño de un algoritmo

El diseño de un algoritmo se puede realizar mediante un diagrama de flujo o mediante pseudocódigo. Los algoritmos tienen las siguientes características

#### ***Características de los algoritmos***

Un algoritmo debe:

1. Ser preciso e indicar el orden de realización de cada paso.
2. Estar definido. Si se sigue un algoritmo dos veces, se debe obtener el mismo resultado cada vez.
3. Ser finito. Si se sigue un algoritmo, se debe terminar en algún momento.

Un algoritmo debe contener como mínimo las siguientes partes:

Entrada

Proceso

Salida.

Por ejemplo para realizar una receta de comida por medio de un algoritmo, cada parte podrá estar determinada así:

Entrada: ingredientes y utensilios empleados.

Proceso: elaboración de la receta en la cocina.

Salida: terminación del plato (por ejemplo, cordero).

#### **3.2.1 Diseño de un algoritmo mediante diagrama de flujo**

Los algoritmos pueden representarse de varias maneras, entre ellas está la representación de diagramas de flujo. La representación en diagramas de flujo tiene como objetivo seguir paso a paso la solución de un problema mediante gráficos



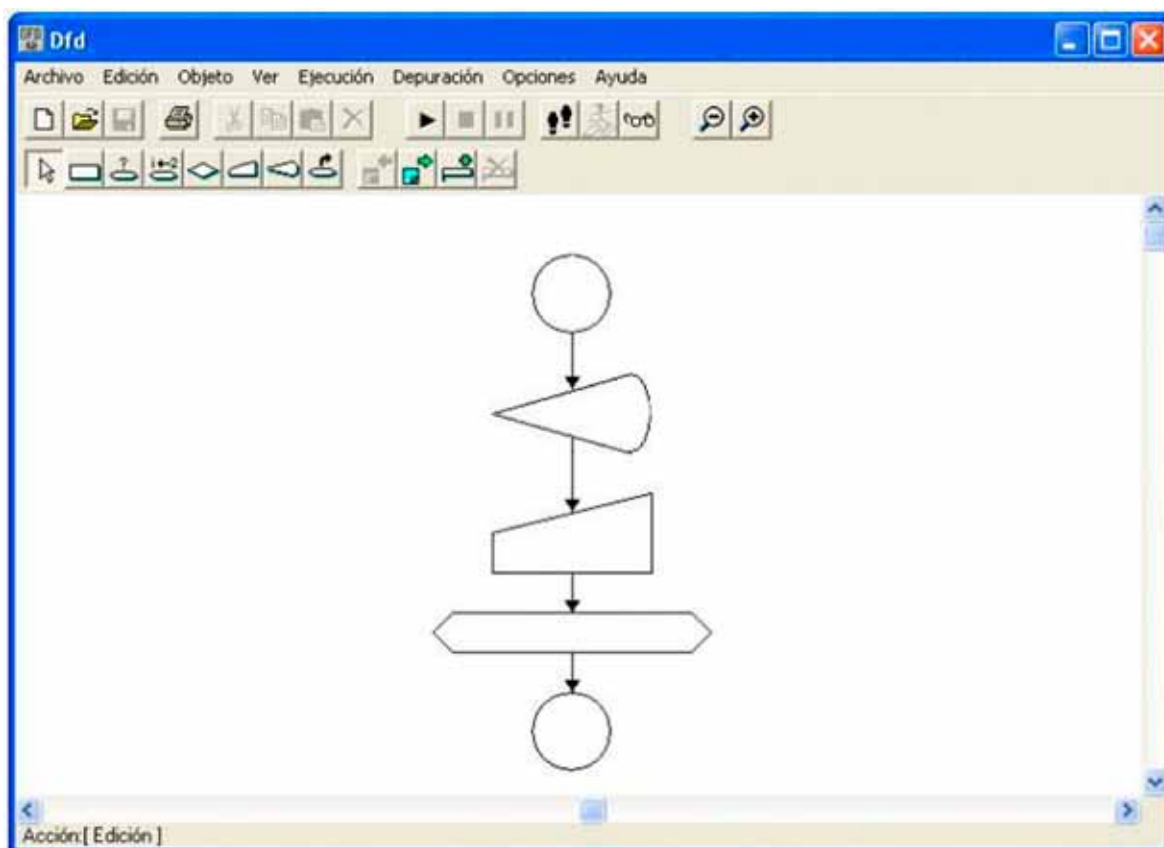
### 3.2.1.1 Definición Diagrama de flujo

Un diagrama de flujo es un conjunto secuencial de figuras geométricas standard conectadas lógicamente entre sí para dar solución a un problema específico, cada figura tiene un significado propio .

La secuencia lógica se da por medio de flechas llamadas líneas de flujo que indican el flujo lógico del algoritmo. Al ser un diagrama grafico facilita la visión de la ejecución del algoritmo.

La simbología utilizada en estos diagramas ha sido estandarizada por las organizaciones ANSI(American National Institute) y por ISO (International Estándar Organization).

Existen herramientas software que permiten realizar los gráficos mencionados anteriormente, una de las más utilizadas es DFD, el cual es de fácil uso y libre acceso en la web, algunos beneficios de utilizar estos son:



- Almacenar digitalmente los diagramas construidos.
- Introducirles modificaciones fácilmente.
- Imprimir copias de los diagramas para compartirlos con compañeros o documentar sus trabajos.
- Exportarlos en varios formatos gráficos para utilizarlos en otros programas.
- Alinear y organizar los símbolos automáticamente

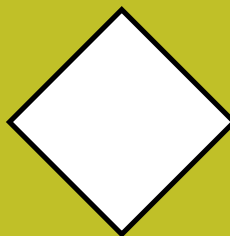
### Simbología Estándar de un diagrama de flujo



**Para iniciar y terminar un diagrama**



**Para representar un proceso**



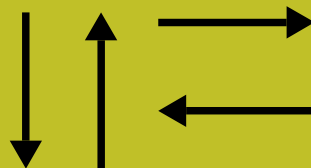
**Para toma de decisiones:  
utilizado también en estructuras cíclicas**



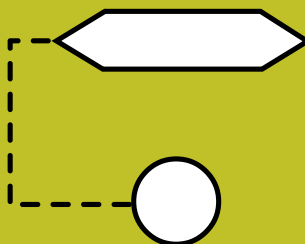
**Para representar entrada de datos**



**Para representar salida de datos**



**Flechas conectoras**



**Iteración: Indica  
que una instrucción  
o un grupo de  
instrucciones deben  
ejecutarse repetida-  
mente.**

### 3.2.1.2 Reglas para la elaboración de un diagrama de flujo:

1. Los diagramas se deben realizar de arriba hacia abajo y de izquierda hacia derecha.
2. Los símbolos de inicio y final deben aparecer solo una vez
3. La ejecución de un programa siempre empieza en la parte superior del programa.
4. La dirección del flujo se debe representar por medio de flechas.
5. Todas las líneas de flujo deben llegar a un símbolo o a otra línea.
6. Se deben inicializar las variables que se utilicen o permitir la asignación de valores mediante la consulta a un usuario.

### 3.2.2 Diseño de un algoritmo mediante pseudocódigo

La representación de pseudocódigo sigue paso a paso la solución de un problema con lenguaje natural, pero recuerde que debe realizar la metodología resolución de problemas para poder iniciar su algoritmo con pseudocódigo. Podría recordarse la imagen de resolución de problemas explicada anteriormente.

```
Paso 1: Inicio  
Paso 2: ...  
Paso 3...  
Paso n...Fin
```

#### Ejemplo1:

Se retoma el ejemplo de Carlos Andrés para llevarlo en forma de algoritmo:

Carlos Andrés, se encuentra ahorrando para comprar una patineta que vale 55.000 pesos. Su papá le ha dado para sus gastos 5.000 pesos durante 7 semanas. Por lavar el auto de su tío tres veces recibió 8.000 pesos. Su hermano Juan Antonio ganó 10.000 pesos por hacer los mandados de su mamá y 4.000 por sacar a pasear el perro. ¿Carlos Andrés tiene ahorrado el dinero suficiente para comprar la patineta o aún le falta?





Paso 1: Inicio

Paso 2: Asignarle el valor del dinero ahorrado por Carlos Andres a la Variable ValorAhorrado = 5.000+8000

Paso 3: Mostrar el ValorAhorrado, indicando si le alcanza o no para la patineta.

Paso 4: Fin.

### Ejemplo 2

Se retoma el ejemplo anterior también por continuidad.

Se necesita calcular el área de un triángulo rectángulo cuya Base mide 3 cm, la Altura 4 cm y la Hipotenusa 5 cm.

Paso1:Inicio

Paso2:Indicar que la variable Base tiene un valor de 3cm

Paso3: Indicar que la variable Altura es de 4cm

Paso4: Calcular el área  $a = ( \text{Bases} * \text{Altura} ) / 2$

Paso 5 : Mostar el área

Paso 6 Fin.

### Ejemplo 3

También se pueden resolver problemas cotidianos: por medio de pseudocódigo escriba un algoritmo para poder pasarse los semáforos, una vez usted se encuentra esperando pasar como peatonal.

Paso1:Inicio

Paso2:Ver el color del semáforo

Paso3: Si el semáforo esta en rojo : hay que detenerse

Paso 4: Mostar mensaje Hay que detenerse.

Paso 5: Si el semáforo esta en amarillo: alistarse para pasar

Paso 6: Mostar mensaje alistarse.

Paso7: Si el semáforo esta en verde: pasarse la calle.

Paso 8: Mostar mensaje puede pasarse y la persona se pasa la calle.

Paso 0 Fin.

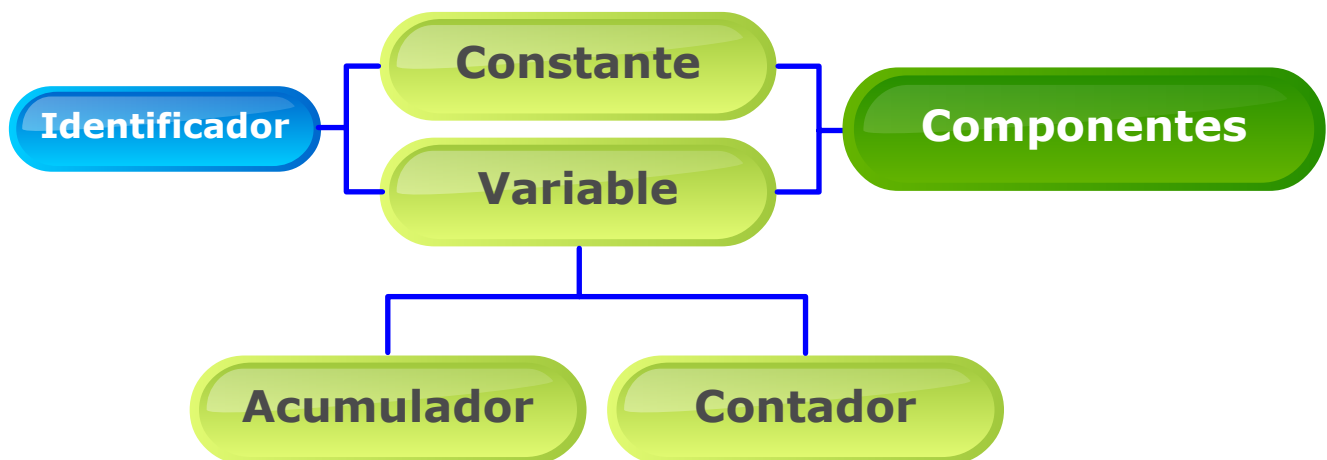


### 3.3 Expresar el algoritmo en un lenguaje de programación:

Una vez el algoritmo este diseñado en diagrama de flujo o en lenguaje natural (seudocódigo), se pone en marcha la “traducción” de este a un lenguaje de programación específico.

Existen diferentes tipos de lenguajes que permiten “traducir” el algoritmo para que pueda ser entendido por el computador entre los cuales son java, c#, visual basic, entre otros, los cuales son tratados específicamente más adelante.

## 4 SOLUCIÓN DE PROBLEMAS CON DIAGRAMAS DE FLUJO Y LENGUAJE NATURAL (SEUDOCÓDIGO).



Recuerde cuando se plantee un problema ya sabe que puede resolverlo por medio de diagrama de flujo o por lenguaje natural.

Es importante conocer y aprender a utilizar los siguientes conceptos que están inmersos al momento de realizar algoritmos.

#### 4.1.1 Variables:

Recordemos que cuando se inicia con el análisis del problema se identifican los datos iniciales, estos datos se estructuran o se definen como variables;

en ellas se pueden almacenar valores y son nombradas con identificadores, es decir nombres para poder identificarlas dentro del algoritmo. Por ejemplo, si en el problema de calcular el área de un triángulo tengo como datos iniciales la base y la altura, estas dos anteriores serian llamadas variables y sus nombres serán "BASE" y "ALTURA".

Un aspecto importante de las variables es que pueden cambiar su valor durante la ejecución del algoritmo.

Se debe tener en cuenta que una variable puede ser declarada, asignada o solicitada mediante un algoritmo para poder ser utilizada.

**Declaración de una variable:** Para utilizar una variable tanto en pseudocódigo como en diagrama de flujo es necesario siempre declararla, es decir indicarle al algoritmo que va a utilizar una variable por ejemplo nombre, quiere decir que usted necesita una variable que almacene los nombres de las personas.

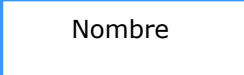
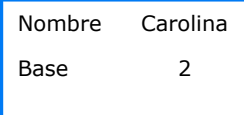
**Asignación de valor a una variable:** Luego de declararla puede usted necesitar que esa variable inicie por defecto con un valor por ejemplo Saldo = 2000, quiere decir que utilizara la variable saldo pero que esta inicializada con un valor de 2000, puede inicializar una variable según el valor que necesite para solucionar el problema.

Asignarle un valor a una variable, constante, acumulador o contador, expresiones complejas o simples, por ejemplo.

```
Variable = expresión  
Por ejemplo: Nombre = Carmen  
Saldo = 2000  
Área = (Base*altura)
```

**Solicitar una variable:** Cuando se necesita que el valor de la variable pueda ser ingresado por un usuario al computador, entonces debe solicitarse este valor de la variable.

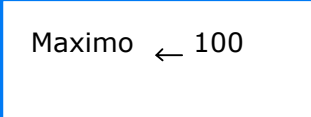


| Diagrama de flujo                                                                                                       | Uso                                                                                                               | Seudocódigo                                                                  |
|-------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|
| <p>Solicitar una variable</p>          | <p>Cuando se necesita que el valor de la variable pueda ser ingresado por un usuario al computador.</p>           | <p>Solicitar una variable<br/>cont=0<br/>Leer Nombre<br/>Conocer Nombre</p>  |
| <p>Declarar una variable</p>           | <p>Indica al algoritmo que va a utilizar una variable por ejemplo nombre, esta variable almacena nombres.</p>     | <p>Declarar una variable<br/><br/>Declarar Variable Nombre</p>               |
| <p>Asignarle valor a una variable</p>  | <p>Para asignarle un valor a una variable, constante, acumulador o contador, expresiones complejas o simples.</p> | <p>Asignarle valor a una variable<br/><br/>Nombre= Carolina<br/>Base = 2</p> |

#### 4.1.2 Constantes:

Almacenan datos al igual que las variables, pero su gran particularidad es que sus datos no cambian durante la ejecución del algoritmo, es decir siempre el valor de la constante va ser el mismo.

Las constantes se deben declarar e inicializar.

| Diagrama de flujo                                                                                                                 | Seudocódigo                                                                     |
|-----------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------|
| <p>Declarar Una constante e inicializarla</p>  | <p>Declarar Una constante e inicializarla</p> <p><br/>Constante Máximo =100</p> |

### 4.1.3 Contadores:

Permiten almacenar valores con el objetivo de llevar a cabo un conteo en las secuencias de ejecución.

Como su palabra lo dice permiten contar, para poder utilizar un contador es necesario inicializarlo en un valor y luego incrementar su valor de una manera constante para permitir realizar el conteo.

| Diagrama de flujo                                       | Seudocódigo                                               |
|---------------------------------------------------------|-----------------------------------------------------------|
| Declarar e iniciar un contador<br><pre> cont ← 0 </pre> | Declarar e inicializar un contador<br><pre> cont=0 </pre> |
| Utilizar contador<br><pre> cont ← cont+1 </pre>         | Utilizar un contador<br><pre> cont=cont+1 </pre>          |

### 4.1.4 Acumuladores:

Permiten almacenar el valor de una variable "A", sumando el valor de ella más el valor de una variable "B"

Como su palabra lo dice permiten acumular el valor de una variable, para poder utilizar un acumulador es necesario inicializarlo en un valor y luego iniciar con la acumulación del valor.

| Diagrama de flujo                                       | Seudocódigo                                               |
|---------------------------------------------------------|-----------------------------------------------------------|
| Declarar e iniciar un contador<br><pre> cont ← 0 </pre> | Declarar e inicializar un contador<br><pre> acum=0 </pre> |
| Utilizar contador<br><pre> cont ← cont+1 </pre>         | Utilizar un contador<br><pre> acum=acum+suma </pre>       |



### 4.1.5 Identificadores:

Los identificadores son nombres que se dan a las variables, constantes, acumuladores y contadores para así poder diferenciarlos. Para asignar los nombres se debe tener en cuenta lo siguiente:

- Los nombres pueden estar formados por una combinación de letras y números (saldoMes, salario, fecha2, baseTriángulo, etc).
- El primer carácter de un nombre debe ser una letra.
- La mayoría de los lenguajes de programación diferencian las mayúsculas de las minúsculas.
- Los nombres deben ser nemotécnicos, con solo leerlos se puede entender lo que contienen. Deben ser muy descriptivos; no utilizar abreviaturas, a menos que se justifique plenamente.
- No utilizar caracteres reservados (% , + , / , > , etc).
- No utilizar palabras reservadas por los lenguajes de programación.
- Para cumplir con convenciones ampliamente utilizadas (Jiménez, 2002), los nombres de procedimientos, variables y constantes deben empezar con minúscula. Ejemplo, fecha, suma, etc. Si es un nombre compuesto por varias palabras, cada una de las palabras (con excepción de la primera) debe empezar con mayúscula. Ejemplo: fechaInicial, baseTriángulo, etc.

## 5. ESTRUCTURAS ALGORÍTMICAS O DE PROGRAMACION

Las estructuras algorítmicas están creadas para orientarnos en la forma de diseñar algoritmos, cada una de ellas representa cierto concepto que permite lograr encontrar la solución del problema, entre ellas tenemos:

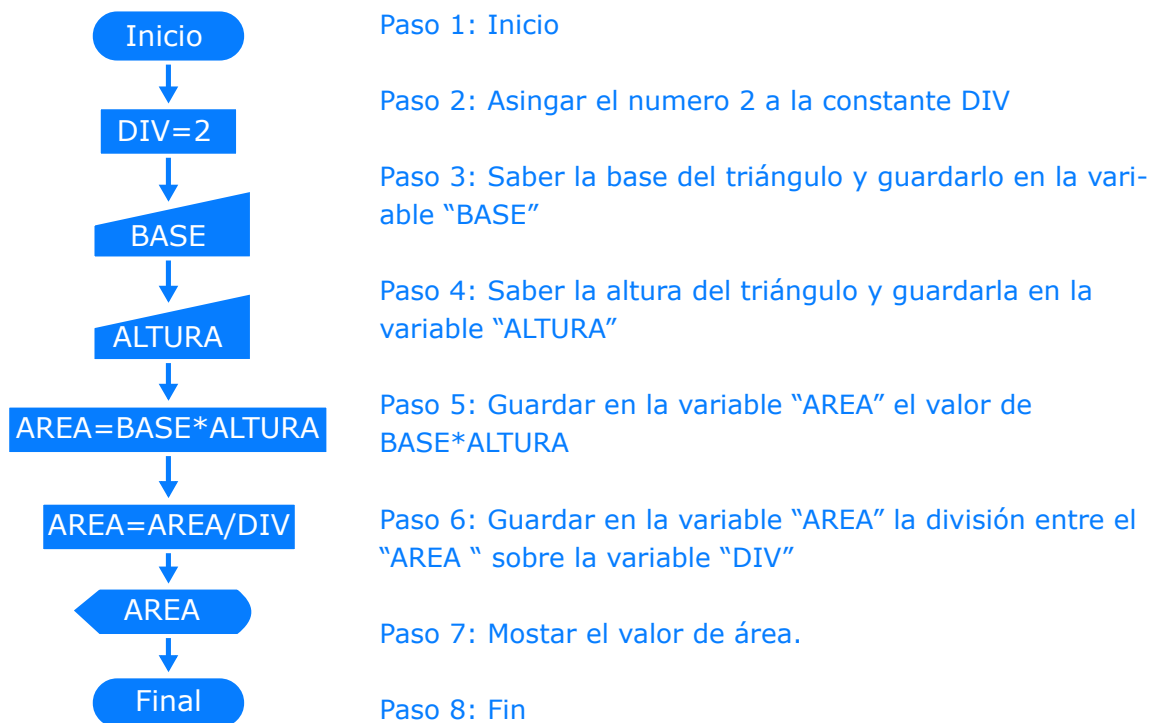
### 5.1 Estructura secuencial

Es una de las estructuras más sencilla, conocida también como estructura lineal y se compone de instrucciones que deben ejecutarse secuencialmente.



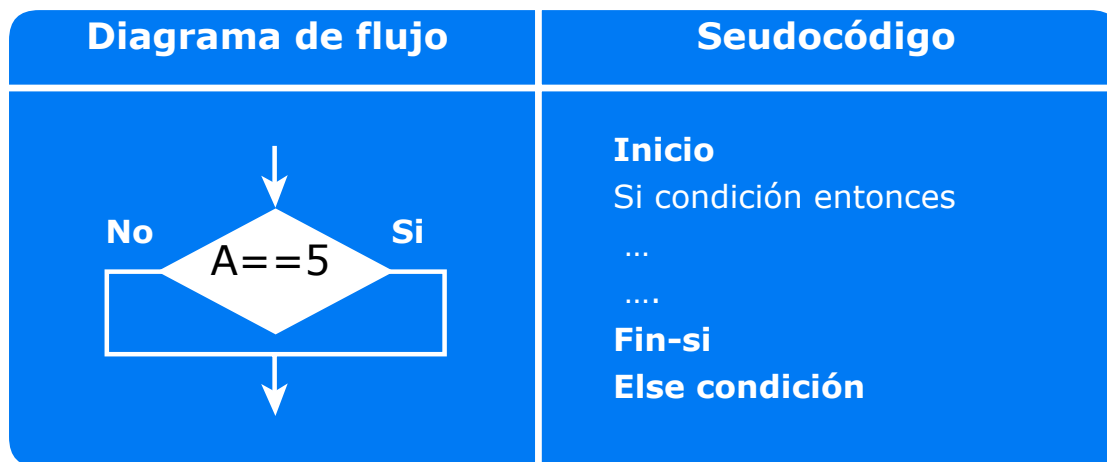


Correspondencia de pseudocódigo a diagrama de flujo



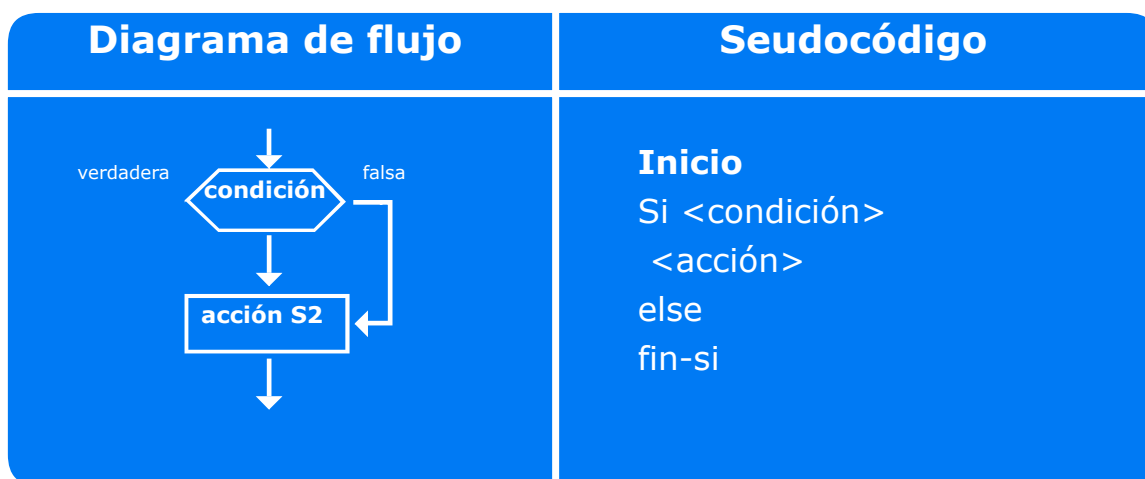
## 5.2 Estructuras de decisión

Son utilizadas para tomar decisiones lógicas, llamadas también estructuras selectivas o alternativas, en ellas se evalúa una condición y en función del resultado de la misma se realiza una opción u otra.



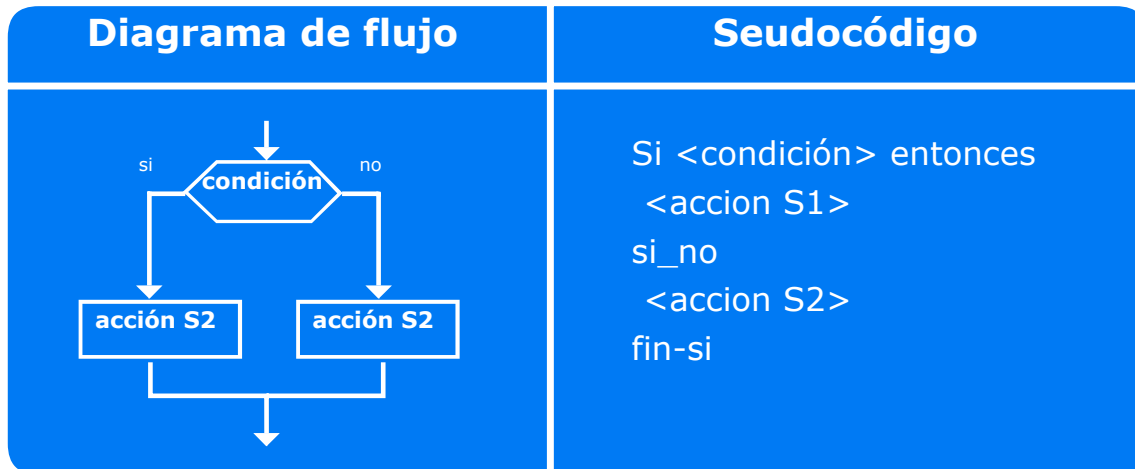
Las estructuras de decisión pueden ser:

**Simples:** (llamada si-entonces), ejecuta una determinada acción cuando se cumple una determinada condición. La selección si-entonces evalúa la condición y si la condición es verdadera, entonces ejecuta la acción; si la condición es false entonces no hace nada.





**Dobles:** La estructura anterior es muy limitada y normalmente se necesitará una estructura que permita elegir entre dos opciones o alternativas posibles, en función del cumplimiento o no de una determinada condición



### 5.3 Estructuras cíclicas y/o repetitivas

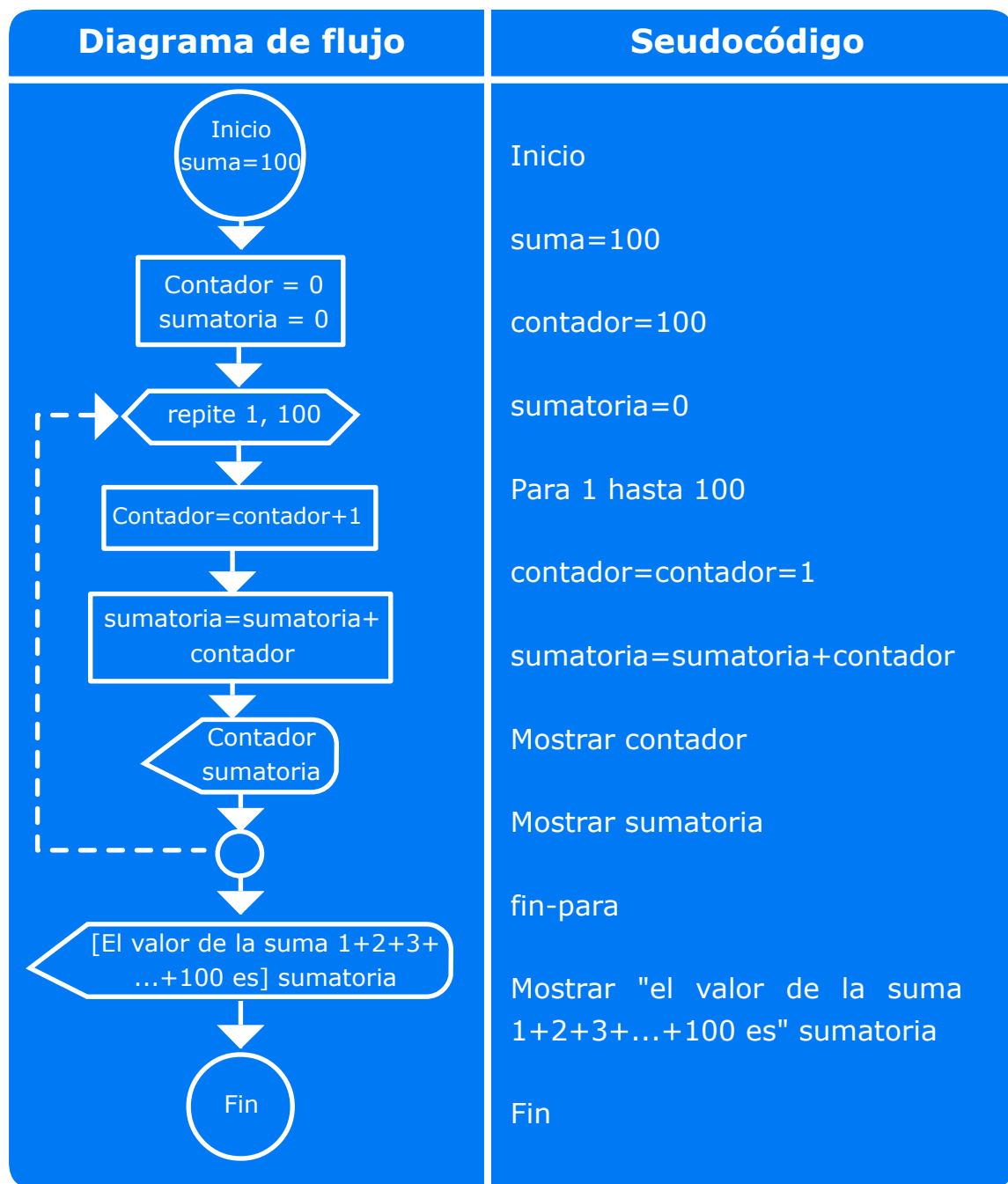
Un tipo muy importante de estructura es el algoritmo necesario para repetir una o varias acciones un número determinado de veces. Las estructuras que repiten una secuencia de instrucciones un número determinado de veces se denominan bucles, y se llama iteración al hecho de repetir la ejecución de una secuencia de acciones.

Y así sucesivamente para cada número de la lista. En otras palabras, el algoritmo repite muchas veces las acciones.

Tales opciones repetidas se denominan bucles o lazos. La acción (o acciones) que se repite en un bucle se denomina iteración. Las dos principales preguntas a realizarse en el diseño de un bucle son: ¿qué contiene el bucle? y ¿cuántas veces se debe repetir?

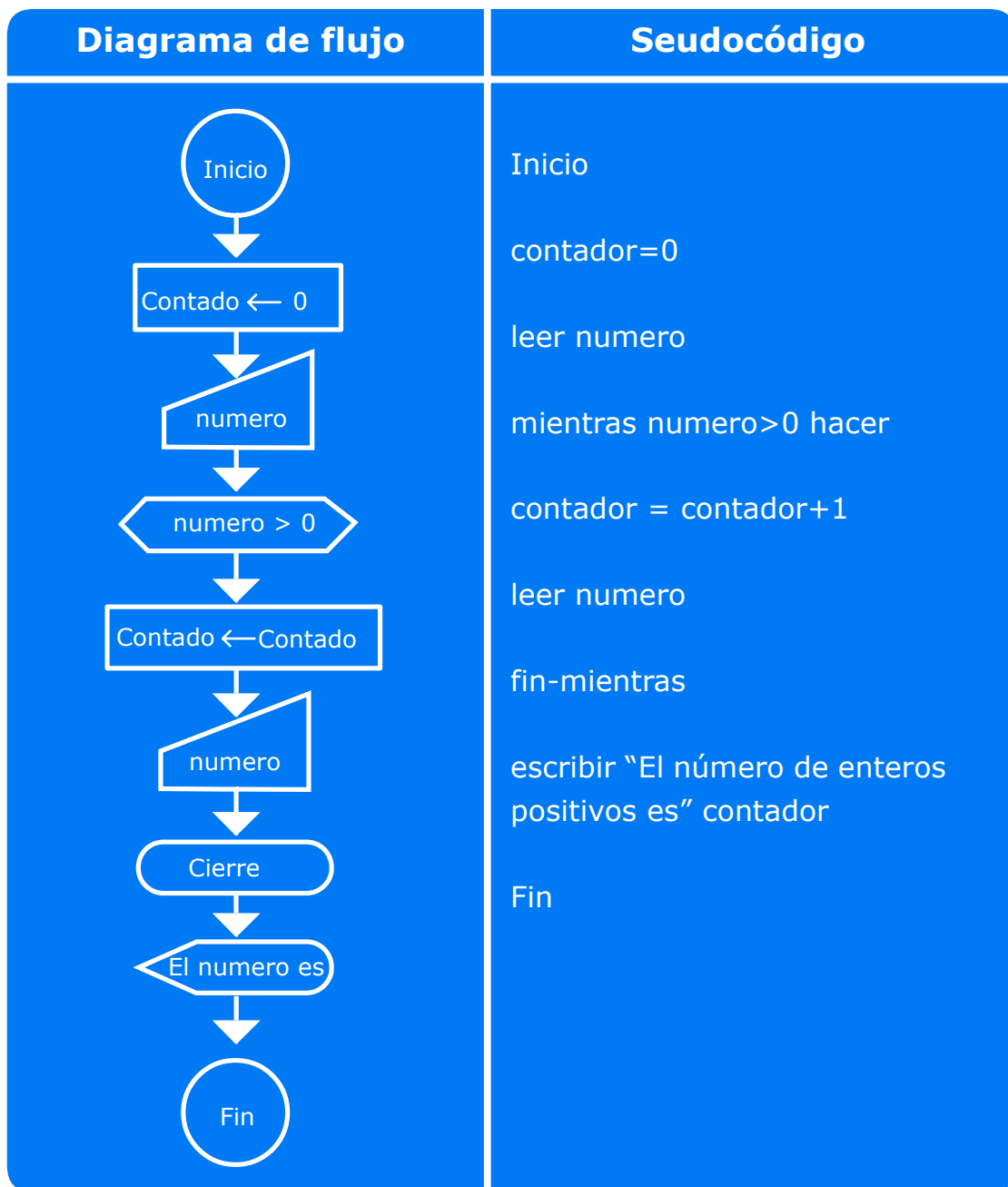
Se manejan entonces 3 tipos de estructuras cíclicas que permiten dar respuesta a los anteriores interrogantes

## 5.3.1 Estructura para



En muchas ocasiones se conoce de antemano el número de veces que se desean ejecutar las acciones de un bucle. Esta estructura permite ejecutar una o varias instrucciones un determinado de veces finita, fija.

## 5.3.2 Estructura mientras



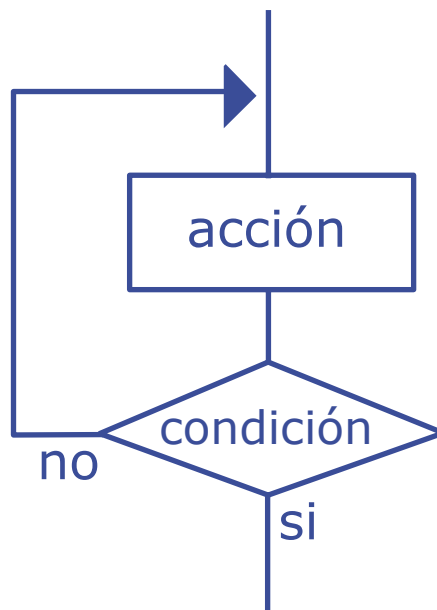
Se llama Mientras a la estructura algorítmica que se ejecuta mientras la condición evaluada resulte verdadera. Se evalúa la expresión booleana y, si es cierta, se ejecuta la instrucción especificada, llamada el cuerpo del bucle. Entonces se vuelve a evaluar la expresión booleana, y si todavía

es cierta se ejecuta de nuevo el cuerpo. Este proceso de evaluación de la expresión booleana y ejecución del cuerpo se repite mientras la expresión sea cierta.

Cuando se ejecuta la instrucción *mientras*, la primera cosa que sucede es que se evalúa la condición (una expresión booleana). Si se evalúa falsa, ninguna acción se toma y el programa prosigue en la siguiente instrucción del bucle. Si la expresión booleana es verdadera, entonces se ejecuta el cuerpo del bucle, después de lo cual se evalúa de nuevo la expresión booleana. Este proceso se repite una y otra vez mientras la expresión booleana (condición) sea verdadera.

### 5.3.3 Estructura repita

La estructura cíclica *REPITA*, al igual que la estructura cíclica *mientras*, se ejecuta un número indeterminado de veces, estas dos estructuras tienen un comportamiento similar, presentando su principal diferencia en el lugar de estructura donde se evalúa la condición, dado que la estructura *MIENTRAS* evalúa la condición del ciclo al inicio del mismo y la estructura *REPITA* lo hace al final del mismo, de este modo, en la estructura cíclica *REPITA*, el programador garantiza que el ciclo se ejecuta al menos una vez.



## GLOSARIO

---

**ALGORITMO:** Es un conjunto de instrucciones que conducen a la solución de un problema determinado, las cuales deben estar relacionadas lógicamente y ordenadamente.

**ANSI (American National Standards Institute):** El Instituto Nacional Estadounidense de Estándares, es una organización sin ánimo de lucro que supervisa el desarrollo de estándares para productos, servicios, procesos y sistemas en los Estados Unidos.

**ISO (International Organization for Standardization):** La Organización Internacional para la Estandarización, es una federación mundial que agrupa a representantes de cada uno de los organismos nacionales de estandarización (como lo es el ICONTEC en Colombia), y que tiene como objeto desarrollar estándares internacionales que faciliten el comercio internacional entre otros.

**SEUDOCODIGO:** Es una descripción informal de alto nivel de un algoritmo, que utiliza las convenciones estructurales de un lenguaje de programación verdadero, pero que está diseñado para la lectura.



## BIBLIOGRAFÍA

---

- [1]. JOYANES Luis. Fundamentos de programación, Algoritmos y estructura de datos. McGraw – Hill
- [2] Lopes Garcia, Juan Carlos. Algoritmos Y Programación (Guía Para Docentes), Segunda Edición, 2007, 2009.



| <b>Control de documento</b><br><b>Construcción Objeto de Aprendizaje</b><br><b>Introducción a la Algoritmia</b> |                                                          |
|-----------------------------------------------------------------------------------------------------------------|----------------------------------------------------------|
| Desarrollador de contenido<br>Experto temático                                                                  | Leydy Carolina Muñoz                                     |
| Asesor pedagógico                                                                                               | Rafael Neftalí Lizcano Reyes<br>Claudia Milena Hernández |
| Producción Multimedia                                                                                           | Luis Fernando Botero Mendoza<br>Victor Hugo Tabares      |
| Programadores                                                                                                   | Daniel Eduardo Martínez                                  |
| Líder expertos temáticos                                                                                        | Ana Yaqueline Chavarro Parra                             |
| Líder línea de producción                                                                                       | Santiago Lozada Garcés                                   |



## RETO JUEGO DE BLACKJACK

En esta actividad deberán programar un juego básico de Blackjack. Las soluciones que se pueden aportar son muchísimas y queda abierto a la imaginación de cada programador. El juego debe contener lo siguiente:

1. El juego deberá permitir barajar las 52 cartas aleatoriamente
2. El juego debe permitir repartir cartas (las primeras 2 cartas del mazo barajado son para el jugador) para 1 jugador que va a jugar contra la casa o el Crupier en un casino.
3. Se debe verificar si desde la partida el jugador ya hizo Blackjack (sumar 21 puntos).
4. Se debe permitir al jugador tomar cartas de la baraja y añadirlas a su mano, se debe contar cuantas cartas va tomando.
5. El juego debe permitir que una vez Jugador ya ha terminado y le toca el turno al Crupier, salvo que previamente el Jugador hubiera tenido Blackjack o se hubiera pasado, en cuyo caso el juego finaliza.

Aquí el resultado de una posible solución.



```
Simulación básica del juego de Blackjack
Generamos las 52 cartas de la Baraja
p1 c8 r6 c4 p4 c2 p2 c3 r13 c1 p7 r9 r3 c2 c10 d12 d7 p5 d8 r11 c13 c12 c3 r12 p8 p10 d10 r7 d11 p12 d4 r4 d12 c7 d5 d13 c6 r10 c5 p11 p3 p13 p6 d5 d8 c3 r11 d9 r1 r5 r6 p9
Mano inicial del Jugador
Mano inicial del Jugador: p1 - c8
Puntos iniciales del Jugador: 19
El Jugador se planta.
Le toca el turno al Crupier
Mano inicial del Crupier
Mano inicial del Crupier: r6 - c8
Puntos iniciales del Crupier: 10
Regla: El Crupier puede con menos de 17 y se planta con 17 o más.
El Crupier pide cartas
El Crupier pide nueva carta.
-La mano del Crupier es: r6 - c8 - p4
La puntuación del Crupier es 14
El Crupier pide nueva carta.
-La mano del Crupier es: r6 - c8 - p4 - d7
La puntuación del Crupier es 15
El Crupier pide nueva carta.
-La mano del Crupier es: r6 - c8 - p4 - d2 - p2
La puntuación del Crupier es 18
El Jugador gana porque tiene más puntos.
```

Se debe consultar para tener en cuenta las reglas básicas del Blackjack.



# Análisis y Desarrollo de Software

## ADSO



El empleo  
es de todos

Mintrabajo

Evidencias de Conocimientos:

**GA3-220501093-AA1-EV01:** Bases conceptuales de lógica proposicional.

**GA3-220501093-AA1-EV02:** Taller de resolución de problemas de algoritmos en pseudocódigo y diagramas de flujo.



@SENAcomunica

[www.sena.edu.co](http://www.sena.edu.co)

Por:

Gustavo Salazar

Instructor Técnico

Centro De Comercio y Servicios  
Regional CAUCA

**GA3-220501093-AA1-EV01:**

Bases conceptuales de lógica proposicional.

**GA3-220501093-AA1-EV02:**

Taller de resolución de problemas de algoritmos en pseudocódigo y diagramas de flujo



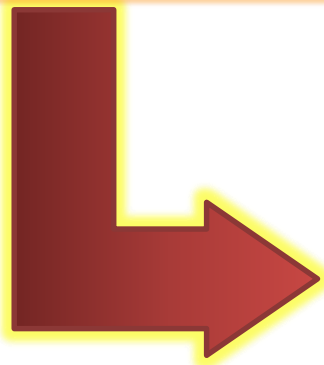
# Análisis y Desarrollo de Software

## AGENDA



- BIENVENIDA
- Socialización Evidencia **GA3-220501093-AA1-EV01**  
y **GA3-220501093-AA1-EV02**
- Resolución de inquietudes
- Conclusiones

**1- Competencia 220501093** - Evaluar requisitos de la solución de software de acuerdo con metodologías de análisis y estándares



**Actividad e Aprendizaje GA3-220501093-AA1:**  
Realizar actividades para el desarrollo del pensamiento algorítmico

# Análisis y Desarrollo de Software ADSO



| ACTIVIDAD DE APRENDIZAJE                                                                          | RESULTADO DE APRENDIZAJE (RAP)                                                                  | Evidencia a Aprobar                                                                                                                                                                                          | Fecha de Entrega                      |
|---------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|
| <b>GA3-220501093-AA1:</b><br>Realizar actividades para el desarrollo del pensamiento algorítmico. | <b>220501093-03</b> - Desarrollar procesos lógicos a través de la implementación de algoritmos. | <b>GA3-220501093-AA1-EV01:</b><br>Bases conceptuales de lógica proposicional.<br><br><b>GA3-220501093-AA1-EV02:</b><br>Taller de resolución de problemas de algoritmos en pseudocódigo y diagramas de flujo. | 06/09/2022<br>a<br>04/10<br><br>/2022 |

# Análisis y Desarrollo de Software ADSO



## Evidencias de Conocimiento

**GA3-220501093-AA1-EV01**

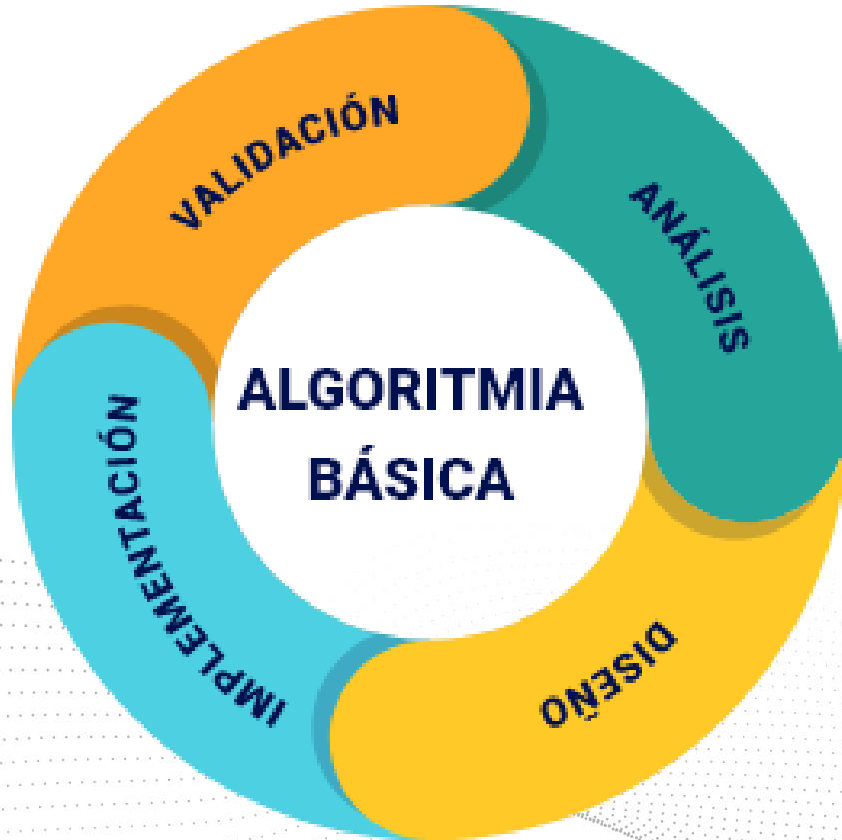
**GA3-220501093-AA1-EV02**



Materiales de formación: para el desarrollo de estas actividades, es importante la lectura y el análisis del componente formativo

**“Introducción a la algoritmia”.**

# 1. Introducción a la algoritmia



Es la aplicación apropiada de una serie de pasos detallados que garantizan una solución correcta de un problema;

La programación de computadores o construcción de software se basa en esta práctica.

## 1. Introducción a la algoritmia



### Definición de algoritmo

Un algoritmo está definido como un “**conjunto ordenado y finito de operaciones que permite hallar la solución de un problema**”.

Computacionalmente las máquinas están en capacidad de hacer unas pocas operaciones matemáticas muy básicas y sencillas, Se considera que computan enormes cantidades de estas operaciones en unidades de tiempo muy corto.





# 1. Introducción a la algoritmia



**¿QUÉ ES UN  
ALGORITMO?**

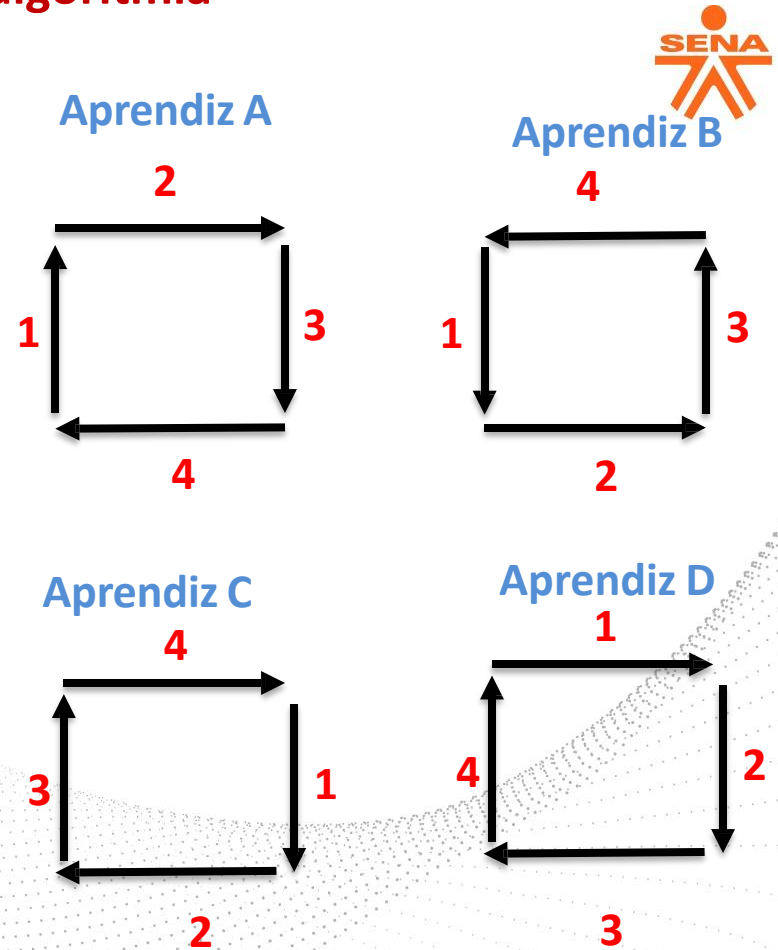


## 1. Introducción a la algoritmia

### Pensamiento algorítmico

Un problema puede ser abordado de varias maneras y **apoyaría la misma solución**, pero lo que siempre tendrían en común es que para resolverlo usarían una **secuencia de pasos ordenados**.

Por ejemplo, si se pasa una **hoja de papel cuadriculado** a los aprendices de una clase y se les pide que **dibujen un cuadrado** sin levantar el lápiz, habrá quienes lo hagan de diferente manera y obtengan el mismo resultado como lo muestra la siguiente figura:



Existe una gran cantidad de soluciones al mismo problema.

# 1. Introducción a la algoritmia



En un problema más complejo se concluye que existen tantas soluciones como el enfoque o lógica empleada por cada aprendiz o persona. Pero lo que sí es común en todos los casos es que se realizaron **unas fases mentales secuencialmente ordenadas**, las cuales son:

1. Entender el problema enunciado por el instructor (**entender el problema**).
2. Escoger una ruta a seguir del rectángulo (**hacer un plan**).
3. Plasmar en el papel cuadriculado ese plan (**ejecutar el plan**).
4. Revisar el resultado y ver que se alcanzó el objetivo (**revisar**).



Cuando el problema es muy complejo de resolver, muchas veces toca **volver y analizar el problema**, o trazar un nuevo plan o plasmar ese nuevo plan una y otra vez, revisando los resultados y volviendo a analizar hasta encontrar la solución del problema.

## 1. Introducción a la algoritmia

### Solución de problemas y programación

“Los algoritmos son una herramienta que permite describir claramente un conjunto finito de instrucciones”



“En el ámbito de la computación **los algoritmos son una herramienta** que permite describir claramente **un conjunto finito de instrucciones ordenadas secuencialmente y libres de ambigüedad**, que debe llevar a cabo un computador para lograr un resultado previsible. Vale la pena recordar que un programa de computador consiste de una serie de instrucciones muy precisas y escritas en un lenguaje de programación que el computador entiende (Logo, Java, Pascal, etc.)”

# 1. Introducción a la algoritmia



**Ejemplo 1: se propone diseñar un algoritmo para apagar una computadora.**

Algoritmo: apagar computadora.

INICIO

- 1.-Dar clic en el icono INICIO.
- 2.-Dar clic en el icono que dice APAGAR.
- 3.-Esperar que se apague completamente.
- 4.-Desconectar los cables del enchufe.

FIN

**Ejemplo 2: se propone diseñar un algoritmo para cargar un celular.**

Algoritmo: cargar un celular.

INICIO

1. Buscar el cargador.
2. Agarrar el cargador.
3. Agarrar el celular.
4. Poner el enchufe en el celular.
5. Conectarlo al tomacorriente.
6. Esperar a que se ponga a cargar.
7. Desconectarlo del tomacorriente.
8. Ponerlo en un lugar seguro.

FIN



# 1. Introducción a la algoritmia

## algoritmo debe ser



### Realizable:

El **proceso algorítmico** debe terminar después de una cantidad finita de pasos. Se dice que **un algoritmo es inaplicable** cuando se ejecuta con un conjunto de datos iniciales y **el proceso resulta infinito** o durante la ejecución se encuentra con un obstáculo insuperable sin arrojar un resultado.

### Comprensible:

**Debe ser claro lo que hace**, de forma que quien ejecute los pasos (ser humano o máquina) sepa **qué, cómo y cuándo hacerlo**. Debe existir un procedimiento que determine el proceso de ejecución.

### Preciso:

**El orden de ejecución de las instrucciones** debe estar perfectamente indicado. Cuando se ejecuta varias veces, con los mismos datos iniciales, **el resultado debe ser el mismo siempre**. La precisión implica determinismo





# 1. Introducción a la algoritmia

## Fases para elaborar un programa de computador



El proceso de diseño de algoritmos de programas informáticos, están basados en un algoritmo que posteriormente es escrito en un lenguaje de programación

# 1. Introducción a la algoritmia



## Análisis del problema (entendéilo)

Como los programas de computador tienen por objetivo **resolver problemas muy puntuales**, lo primero que se debe realizar para resolverlos es lograr una mejor comprensión posible de estos.

Para realizar esta actividad se debe:

1. Formular claramente el problema.
2. Especificar los resultados que se desean obtener con la solución.
3. Identificar la información disponible en los datos.





### 1. Formular el problema

A continuación, algunas preguntas previas a la formulación del problema:

- ✓ ¿Es posible definir de forma más clara el problema?
- ✓ ¿Qué palabras no son conocidas o se desconoce su significado?
- ✓ ¿Se ha resuelto antes algún problema similar?
- ✓ ¿Qué información es importante para resolver el problema?
- ✓ ¿Qué información no es importante y se puede omitir?

# 1. Introducción a la algoritmia



## Ejemplo

Don Juan necesita decidir cómo comprar un teléfono celular que cuesta **\$870.000 de contado** o **\$980.000 a crédito**; él tiene **\$630.000** pesos en efectivo.

Luego de revisar las preguntas previas se puede concluir:

Como el efectivo que tiene don Juan no le alcanza para comprar el teléfono celular de contado, entonces él tiene dos opciones: **comprarlo completamente a crédito** o también **pagar una parte de contado (como cuota inicial) y el resto diferirlo a crédito**.

Para poder resolver este problema se debe conocer **el número de cuotas en el que difiere si desea pagarlo totalmente a crédito** o conocer el número de cuotas y el valor total del celular si se da una cuota inicial de **\$630.000 pesos**.



## 2. Precisar elementos de entrada

Otro punto importante en la fase de análisis del problema es determinar cuál es la información disponible, por lo tanto:

- ✓ ¿Qué información es importante y necesaria para resolver el problema?
- ✓ ¿Qué información no es importante y se puede prescindir?
- ✓ ¿Cuáles son los datos de entrada conocidos?
- ✓ ¿Cuál es la incógnita o qué se debe calcular?
- ✓ ¿Los datos se pueden agrupar en categorías?
- ✓ ¿Qué información adicional hace falta para resolver el problema?

### 3. Precisar resultados esperados

Para determinar con claridad cuál es el resultado final esperado (o producto) que se quiere obtener es necesario determinar o definir qué resultados se solicitan y, en qué tipo o formato deben estar (mostrados en pantalla, diagramación, orden, etc.). Para ello, es importante tener en cuenta:

- ✓ ¿Qué información están solicitando?
- ✓ ¿En qué formato se debe entregar esta información? ¿Qué información adicional hace falta para resolver el problema?

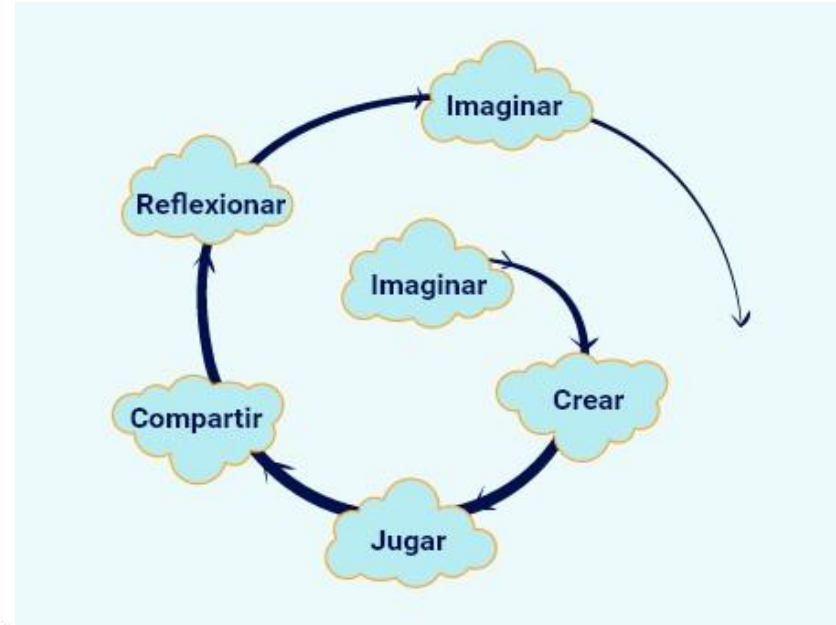
# 1. Introducción a la algoritmia



## Desarrollo de la creatividad: elementos, modelos, fases y sus objetivos

**Resolver algoritmos** es una actividad que requiere en la mayoría de los casos **creatividad por parte del programador**, esta es una habilidad que se va fortaleciendo en la medida que se desempeña más y más en procesos de desarrollo de software.

Para estimular “el pensamiento creativo propuesto por Resnick (2007) **el programador debe imaginar qué quiere hacer**; crea un proyecto basado en sus ideas; juega y comparte sus ideas y creaciones con otros para reflexionar sobre sus experiencias y permitirse imaginar nuevas ideas y nuevos proyectos. La espiral genera **un proceso indefinido de mejoramiento continuo**”



Espiral del Pensamiento Creativo diseñada por el Dr. Mitchel Resnick

### En conclusión

**Un algoritmo** es la definición de una secuencia lógica de pasos que permiten solucionar un problema

**Formular algoritmos** es la base de los grandes adelantos en las ciencias matemáticas y la tecnología, porque se pueden aplicar a cualquier área del conocimiento.



## 2. Lógica matemática



**La lógica matemática** es la disciplina que trata sobre los métodos de razonamiento. En un nivel elemental, la lógica proporciona reglas y técnicas para determinar si es o no válido un argumento dado.

**El razonamiento lógico** se emplea en matemáticas para demostrar teoremas; en ciencias de la computación para verificar si son o no correctos los programas; en las ciencias física y naturales, para sacar conclusiones de experimentos; y en las ciencias sociales y en la vida cotidiana, para resolver una multitud de problemas. Ciertamente se usa en forma constante el razonamiento lógico para realizar cualquier actividad.





## 2. Lógica matemática





## 2. Lógica matemática



### Componentes de la lógica matemática

Todo razonamiento lógico está compuesto por una serie de elementos que se interrelacionan entre sí, y la correcta interpretación de estos componentes y de sus relaciones corresponde al análisis

deductivo. **Estos componentes son las preposiciones y conectores lógicos**



## 2. Lógica matemática



### Proposiciones

Es una oración que puede ser **falsa o verdadera**, pero no ambas a la vez.

**Las proposiciones pueden ser simples o compuestas.** Las primeras están formadas por una sola proposición, por el contrario, las proposiciones compuestas están conformadas por dos o más proposiciones unidas por un conector.

### proposiciones o Enunciados

**p**: la tierra es plana.

**q**:  $-17 + 38 = 21$

**r**: hola ¿cómo estás?

**s**: lava el carro por favor.

**p** y **q** son proposiciones simples válidas porque pueden tomar un valor verdadero o falso; sin embargo, los enunciados **r** y **s** son proposiciones no válidas, porque no pueden tomar un valor de falso o verdadero, uno de ellos es un saludo y el otro es una orden.

## 2. Lógica matemática



### Conectores lógicos y proposiciones compuestas

Existen las proposiciones compuestas que son las que se componen de varias proposiciones, las cuales utilizan para su conexión **operadores o conectores lógicos**

#### Operador AND (y) conjunción

Conecta dos proposiciones que se deben unir para que se pueda obtener un resultado verdadero, se representa por el siguiente símbolo  **$\wedge$** .

#### Ejemplos:

- ✓ Nuestra bandera es de color amarillo, azul **y** rojo.
- ✓ El número 9 es mayor que 7 **y** menor que 12.
- ✓ El carro enciende cuando tiene corriente eléctrica **y** gasolina en el tanque.



## 2. Lógica matemática



### Conectores lógicos y proposiciones compuestas

Existen las proposiciones compuestas que son las que se componen de varias proposiciones, las cuales utilizan para su conexión **operadores o conectores lógicos**

#### Operador OR (o) disyunción

Conecta dos proposiciones y se obtiene una proposición compuesta verdadera cuando por lo menos unas de las proposiciones son verdaderas., se representa por el siguiente símbolo **V**

#### Ejemplo:

- ✓ Puedo ir a Bogotá en avión **o** en bus.
- ✓ Puedo entrar al estadio si compro mi boleta **o** si tengo un pase de cortesía.
- ✓ Voy a desayunar con huevos revueltos **o** huevos pericos.



## 2. Lógica matemática



### Conectores lógicos y proposiciones compuestas

Existen las proposiciones compuestas que son las que se componen de varias proposiciones, las cuales utilizan para su conexión **operadores o conectores lógicos**

#### Operador NOT (no) negación

Su propósito es negar una proposición, es decir, que si tenemos una proposición verdadera al aplicarle este tipo de operador se obtiene la negación, es decir, es falso.

Se representa por el siguiente símbolo  $\neg$

#### Ejemplo:

- ✓ **No** es cierto que Gabriel García Márquez escribió la Ilíada.
- ✓ **No** es cierto que el América de Cali es de la ciudad de Bogotá.
- ✓ **Es falso** que el agua es un hidrocarburo.



## 2. Lógica matemática

### Tablas de verdad

Una estrategia de la lógica que tiene como objetivo determinar **la validez de varias propuestas** en alguna situación, es decir, **determina las condiciones necesarias en las que un enunciado es verdadero**, estas condiciones se representan con una tabla de verdad que determina el valor de verdad de todas las proposiciones que componen la situación.

**una proposición  
puede ser  
verdadera o  
falsa  
únicamente,  
nunca ambas.**

## 2. Lógica matemática



- ❑ El valor de una **proposición verdadera** se representa por “**V**” o un “**1**”.
- ❑ El valor de una **proposición falsa** se representa por “**F**” o un “**0**”.
- ❑ **Existe una tabla de verdad** para cada uno de los conectores lógicos, lo que se tiene que evaluar para responder si una proposición compuesta es verdadera o falsa es el estatus actual de las proposiciones simples.





## 2. Lógica matemática



Tabla de verdad conjunción (Y)

| P | Q | $P \wedge Q$ |
|---|---|--------------|
| V | V | V            |
| V | F | F            |
| F | V | F            |
| F | F | F            |



| P | Q | $P \wedge Q$ |
|---|---|--------------|
| 1 | 1 | 1            |
| 1 | 0 | 0            |
| 0 | 1 | 0            |
| 0 | 0 | 0            |

Tabla de verdad negación (no)

| P | $\neg P$ |
|---|----------|
| V | F        |
| F | V        |



| P | $\neg P$ |
|---|----------|
| 1 | 0        |
| 0 | 1        |

Tabla de verdad disyunción (O)

| P | Q | $P \vee Q$ |
|---|---|------------|
| V | V | V          |
| V | F | V          |
| F | V | V          |
| F | F | F          |



| P | Q | $P \vee Q$ |
|---|---|------------|
| 1 | 1 | 1          |
| 1 | 0 | 1          |
| 0 | 1 | 1          |
| 0 | 0 | 0          |



## 2. Lógica matemática



Tabla de verdad Condicional Si o para que ( $\Rightarrow$ )

| P        | Q        | $P \Rightarrow Q$ |
|----------|----------|-------------------|
| V        | V        | V                 |
| <b>V</b> | <b>F</b> | <b>F</b>          |
| F        | V        | V                 |
| F        | F        | V                 |



| P        | Q        | $P \Rightarrow Q$ |
|----------|----------|-------------------|
| 1        | 1        | 1                 |
| <b>1</b> | <b>0</b> | <b>0</b>          |
| 0        | 1        | 1                 |
| 0        | 0        | 1                 |

Tabla de verdad sí y solo sí ( $\Leftrightarrow$ )

| P        | Q        | $P \Leftrightarrow Q$ |
|----------|----------|-----------------------|
| <b>V</b> | <b>V</b> | <b>V</b>              |
| V        | F        | F                     |
| F        | V        | F                     |
| <b>F</b> | <b>F</b> | <b>V</b>              |



| P        | Q        | $P \Leftrightarrow Q$ |
|----------|----------|-----------------------|
| <b>1</b> | <b>1</b> | <b>1</b>              |
| 1        | 0        | 0                     |
| 0        | 1        | 0                     |
| <b>0</b> | <b>0</b> | <b>1</b>              |

## 2. Lógica matemática



### Ejemplo1

## Si no llueve se arruina la cosecha.

Este tipo de proposición se conoce como la condición “**para que**”, de manera que para que no se arruine la cosecha se requiere que llueva.

p: si no llueve

q: se arruina la cosecha

Al aplicar el operador  $p \rightarrow q$  su tabla de verdad quedaría de la siguiente manera:

| P | Q | $P \rightarrow Q$ |
|---|---|-------------------|
| V | V | V                 |
| V | F | F                 |
| F | V | V                 |
| F | F | V                 |

O puede ser

| P | Q | $P \rightarrow Q$ |
|---|---|-------------------|
| 1 | 1 | 1                 |
| 1 | 0 | 0                 |
| 0 | 1 | 1                 |
| 0 | 0 | 1                 |

## 2. Lógica matemática

### Ejemplo 2



### Es buen estudiante.

Es buen estudiante, si y solo si tiene promedio de cinco, donde se puede concluir lo siguiente:

p: es un buen estudiante

q: tiene promedio de 5

al aplicar el operador  $p \leftrightarrow q$  su tabla de verdad quedaría de la siguiente manera:

| P | Q | $P \leftrightarrow Q$ |
|---|---|-----------------------|
| V | V | V                     |
| V | F | F                     |
| F | V | F                     |
| F | F | V                     |

O puede ser

| P | Q | $P \leftrightarrow Q$ |
|---|---|-----------------------|
| 1 | 1 | 1                     |
| 1 | 0 | 0                     |
| 0 | 1 | 0                     |
| 0 | 0 | 1                     |

## 2. Lógica matemática



### Ejemplo 3

#### Las palomas vuelan.

p: todas las palomas vuelan

q: todas las aves tienen plumas

r: luego todas las palomas son aves

Estas proposiciones se pueden escribir utilizando los operadores lógicos de la siguiente manera:

$$(p \wedge q) \rightarrow r$$

Para este ejercicio se construye la tabla de verdad:

| P | Q | R | $P \wedge Q$ | R | $(P \wedge Q) \rightarrow R$ |
|---|---|---|--------------|---|------------------------------|
| V | V | V | V            | V | V                            |
| V | V | F | V            | F | F                            |
| V | F | V | F            | V | V                            |
| V | F | F | F            | F | V                            |
| F | V | V | F            | V | V                            |
| F | V | F | F            | F | V                            |
| F | F | V | F            | V | V                            |
| F | F | F | F            | F | V                            |

## 2. Lógica matemática



### Ejercicios de lógica proposicional

Estos ejercicios con su respectiva solución permiten identificar *un problema, reconocer los datos de entrada y los procesos a ser aplicados como estrategia a seguir con la utilización de la lógica proposicional en la identificación y solución de problemas.*

#### Ejemplo 1

Se necesita personal  
femenino que sepa  
Microsoft Office o  
Gsuite.

#### Pasos a seguir:

- ✓ Identificar el problema.
- ✓ Descomponer el enunciado en proposiciones atómicas.
- ✓ Evaluar los posibles casos derivados del enunciado anterior.
- ✓ Determinar los casos donde se cumpla la condición y hacer la selección de las personas de acuerdo con el requerimiento.

## 2. Lógica matemática



1. **Identificar el problema:** Se trata de buscar o contratar personal femenino que sepa utilizar Microsoft Office o Google Suite.
2. **Descomponer el enunciado:** Descomponé en p<sup>ro</sup>posiciones atómicas del enunciado (cuando están sepa<sup>ra</sup>adas del contexto del p<sup>ro</sup>blema se llaman p<sup>ro</sup>posiciones atómicas).
  - ✓ **p:** es personal femenino.
  - ✓ **q:** sabe utilizar el Mic<sup>ro</sup>soft Office.
  - ✓ **r:** sabe utilizar la suite de Google.
3. **Dete<sup>er</sup>minar los casos:** Dete<sup>er</sup>mina<sup>er</sup> las <sup>er</sup>elaciones lógicas de acue<sup>er</sup>do con lo planteado en el enunciado, utilizando las p<sup>ro</sup>posiciones atómicas definidas en el punto ante<sup>er</sup>io<sup>er</sup>.  $p \wedge (q \vee r)$
4. **Evaluar los posibles casos derivados del enunciado anterior**

## 2. Lógica matemática

### Ejemplo 2



Juan sabe utilizar Microsoft Office y Gsuite, Martha sabe utilizar Gsuite, Natalia sabe utilizar Gsuite y Microsoft Office.

Crear la tabla de verdad y aplicar la lógica proposicional para elegir las personas que cumplan con los requisitos exigidos.

Para la construcción de la tabla de verdad se evalúa cada una de las preposiciones atómicas y, se van validando para verificar si se cumple lo que la haría verdadera o si no se cumple, lo cual haría que la proposición sea falsa.

## 2. Lógica matemática



Se analiza cada posible caso:

por lo que se **evalúa a Juan**, para la primera proposición sería un valor falso, ya que **p requiere que sea personal femenino**, el cual se pondrá en la **columna 2**,

luego se evalúa si **Juan sabe utilizar Microsoft Office o la Suite de Google**, el valor va en la **columna tres** y por último, se evalúa **la conjunción** para determinar la solución.

| Nombre   | P | $(q \vee r)$ | $p \wedge (q \vee r)$ |
|----------|---|--------------|-----------------------|
| Juan     | F | V            | F                     |
| Martha   | V | V            | V                     |
| Nathalia | V | V            | V                     |

De acuerdo con la anterior tabla, se concluye que Martha y Natalia son aptas para el trabajo, lo que permite seleccionarlas.



## 2. Lógica matemática



### Ejemplo 3

Sandra dice  
adelgazo o  
no hago  
dieta.

Para este enunciado además se propone una serie de alternativas que podrían ser válidas o no.

Las siguientes son las posibles respuestas de las cuales se deberá elegir la que se considera es real al problema planteado.

- 1 Si adelgazo, hago dieta.
- 2 Si no adelgazo, hago dieta.
- 3 Si hago dieta, no adelgazo.
- 4 Si hago dieta adelgazo.
- 5 Hago dieta o no adelgazo.

# Solución:

## 1. Problema

Identificar la respuesta correcta al problema presentado por Sandra.

## 2. Definir

Definir las proposiciones atómicas:

**p**: adelgazo.

**q**: hago dieta.

## 3. Determinar

Determinar la relación lógica.

$p \vee \neg q$ .

## 4. Crear

Crear la tabla de verdad.

| p | q | $\neg q$ | $(p \vee \neg q)$ |
|---|---|----------|-------------------|
| V | V | F        | V                 |
| V | F | V        | V                 |
| F | V | F        | F                 |
| F | F | V        | V                 |

# Análisis y Desarrollo de Software ADSO



## Evidencias de Conocimiento

**GA3-220501093-AA1-EV01:** Bases conceptuales de lógica proposicional.

**GA3-220501093-AA1-EV02:** Taller de resolución de problemas de algoritmos en pseudocódigo y diagramas de flujo.

En Esta actividad se estudia:

- ✓ **Los conceptos básicos asociados al concepto de algoritmo**
- ✓ **Definición formal de algoritmo**
- ✓ **Las características del pensamiento algorítmico**

Estos elementos ayudan a plantear **soluciones a problemas reales y problemas asociados a la programación.**

Evidencia de conocimiento: GA3-220501093-AA1-EV01  
bases conceptuales de lógica proposicional

Teniendo en cuenta los conceptos desarrollados durante el componente formativo, construir **un documento donde se dé solución a los siguientes problemas de lógica proposicional** registrando cada uno de los pasos elaborados para llegar a la solución:



Evidencia de conocimiento: GA3-220501093-AA1-EV01  
bases conceptuales de lógica proposicional

$$(2 * 5) < 8 \text{ OR } ((4 * 6) > (2 * 5))$$

$$(4 + 5) < 3 \text{ AND } ((5 * 5) + (4 + 25 < 3))$$

Construir la tabla de verdad para las  
siguientes expresiones

$$\neg (P \wedge Q)$$

$$\neg (P \vee \neg Q)$$



### Lineamientos generales para la entrega de la evidencia:

- **Producto para entregar:** *Un único archivo* con el nombre del estudiante donde se encuentre todas las soluciones a cada uno de los ejercicios presentados con el paso a paso realizado para encontrar la solución.
- **Formato:** PDF.
- Para hacer el desarrollo de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **bases conceptuales de lógica proposicional GA3-220501093-AA1-EV01**



# Análisis y Desarrollo de Software ADSO



## INSTRUMENTO DE EVALUACIÓN

### Criterios de Evaluación:

- Soluciona problemas de lógica proposicional incorporando habilidades propias en el oficio como programador.

### LISTA DE VERIFICACIÓN

| No. | VARIABLES/INDICADORES DE LOGRO                                                                                                       | CUMPLE |    | Observaciones |
|-----|--------------------------------------------------------------------------------------------------------------------------------------|--------|----|---------------|
|     |                                                                                                                                      | SI     | NO |               |
| 1   | En el documento se presenta el paso a paso para la solución de la proposición:<br>$(2 * 5) < 8 \text{ OR } ((4 * 6) > (2 * 5))$      |        |    | 10%           |
| 2   | La proposición $(2 * 5) < 8 \text{ OR } ((4 * 6) > (2 * 5))$ es resuelta correctamente.                                              |        |    | 15%           |
| 3   | En el documento se presenta el paso a paso para la solución de la proposición<br>$(4 + 5) < 3 \text{ AND } ((5 * 5) + (4 + 25 < 3))$ |        |    | 10%           |
| 4   | La proposición $(4 + 5) < 3 \text{ AND } ((5 * 5) + (4 + 25 < 3))$ es resuelta correctamente.                                        |        |    | 15%           |
| 5   | Se presenta la tabla de verdad completa para la expresión $\neg (P \wedge Q)$ .                                                      |        |    | 10%           |
| 6   | La tabla de verdad de $\neg (P \wedge Q)$ es correcta.                                                                               |        |    | 15%           |
| 7   | Se presenta la tabla de verdad completa para la expresión $\neg (P \vee \neg Q)$ .                                                   |        |    | 10%           |
| 8   | La tabla de verdad de $\neg (P \vee \neg Q)$ es correcta.                                                                            |        |    | 15%           |

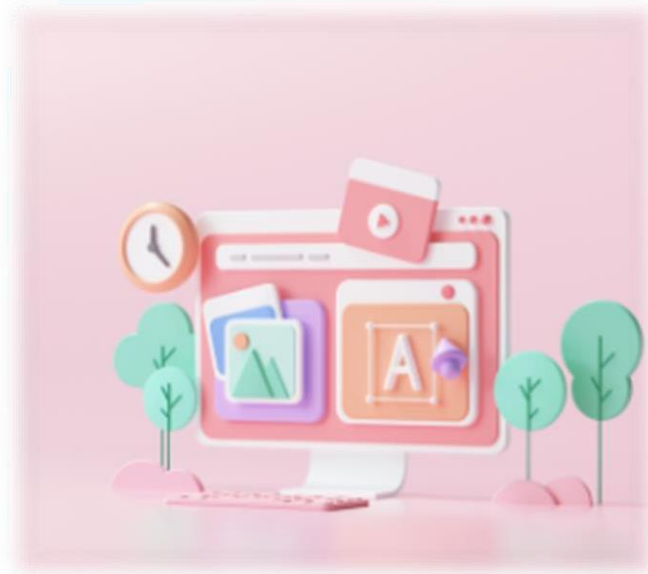


# Análisis y Desarrollo de Software ADSO



**Evidencia de producto: GA3-220501093-AA1-EV02**  
**Taller de resolución de problemas de algoritmos en pseudocódigo y diagramas de flujo**

Teniendo en cuenta las recomendaciones para el análisis de problemas desarrollado durante el componente formativo en esta actividad se plantean algunos problemas, los cuales deberá leer detenidamente para luego **identificar los elementos de entrada y los resultados esperados**. Adicionalmente, se introduce el tema de diagramas de flujo para lo cual deberá **realizar una investigación corta respondiendo algunos lineamientos establecidos en esta actividad**



### Elementos para tener en el desarrollo del taller:

#### Sección 1

Teniendo en cuenta los siguientes problemas identificar los posibles elementos de entrada y resultados esperados de cada uno

- ☐ Se desea elaborar un algoritmo que permita identificar la cantidad de dólares equivalentes a una cantidad de pesos colombianos.
- ☐ Se desea elaborar un algoritmo que permita determinar la temperatura equivalente en grados centígrados a la cantidad de grados Fahrenheit actuales en la ciudad de New York.
- ☐ Suponiendo que nos encontramos descansando en una nuestra casa en una ciudad de Colombia requiero hacer un plan detallado para llegar a tiempo a mi sitio de trabajo el día siguiente.
- ☐ Suponiendo que tengo habilidades en la elaboración de comida necesito elaborar un arroz con pollo para 5 personas.

**Elementos para tener en el desarrollo del taller:**

### **Sección 2 Taller**

**Elabore una investigación corta** usando los materiales disponibles en la biblioteca o Internet respecto a los fundamentos para la resolución de problemas con algoritmos, seleccione por lo menos tres fuentes que le permitan resolver las siguientes preguntas:

- ☐ Definición de diagrama de flujo.
- ☐ Símbolos más importantes.
- ☐ Seleccionar uno de los problemas de la sección 1 y representarlo en su equivalente diagrama de flujo.

### **Respecto a los diagramas de flujo:**

- Referenciar las fuentes de información utilizadas para resolver cada una de las preguntas utilizando el formato de las Normas APA.

### Lineamientos generales para la entrega de la evidencia:

- **Producto para entregar:** Documento con la solución del taller propuesto.
- **Formato:** PDF.
- **Extensión:** libre
- Para hacer el desarrollo de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **taller de resolución de problemas de algoritmos en pseudocódigo y diagramas de flujo - GA3-220501093-AA1-EV02**



# Análisis y Desarrollo de Software AD SO

## GA3-220501093-AA1-EV02



### INSTRUMENTO DE EVALUACIÓN

#### Criterios de Evaluación:

- Soluciona problemas de lógica proposicional incorporando habilidades propias en el oficio como programador.

### LISTA DE VERIFICACIÓN

| No. | VARIABLES/INDICADORES DE LOGRO                                                            | CUMPLE |    | Observaciones |
|-----|-------------------------------------------------------------------------------------------|--------|----|---------------|
|     |                                                                                           | SI     | NO |               |
|     | Sección 1                                                                                 |        |    |               |
| 1   | Se identifican claramente los elementos de entrada y resultados esperados del problema 1. |        |    | 15%           |
| 2   | Se identifican claramente los elementos de entrada y resultados esperados del problema 2. |        |    | 15%           |
| 3   | Se identifican claramente los elementos de entrada y resultados esperados del problema 3  |        |    | 15%           |
| 4   | Se identifican claramente los elementos de entrada y resultados esperados del problema 4. |        |    | 15%           |

# Análisis y Desarrollo de Software AD SO

## GA3-220501093-AA1-EV02



### INSTRUMENTO DE EVALUACIÓN

#### Criterios de Evaluación:

- Soluciona problemas de lógica proposicional incorporando habilidades propias en el oficio como programador.

### LISTA DE VERIFICACIÓN

| No. | VARIABLES/INDICADORES DE LOGRO                                                                                   | CUMPLE |    | Observaciones |
|-----|------------------------------------------------------------------------------------------------------------------|--------|----|---------------|
|     |                                                                                                                  | SI     | NO |               |
|     | Sección 2                                                                                                        |        |    |               |
| 5   | Se incluye una definición clara de diagrama de flujo.                                                            |        |    | 5%            |
| 6   | Se incluye los símbolos representativos de un diagrama de flujo.                                                 |        |    | 5%            |
| 7   | Representa correctamente un algoritmo de los problemas de la sección 1 por medio del diagrama de flujo.          |        |    | 20%           |
| 8   | Incluye referencias en formato APA.                                                                              |        |    | 5%            |
| 9   | El documento presenta normas básicas de presentación de un documento escrito (portada, contenido y referencias). |        |    | 5%            |



**G R A C I A S**

Línea de atención al ciudadano: 018000 910270  
Línea de atención al empresario: 018000 910682



@SENAcomunica

[www.sena.edu.co](http://www.sena.edu.co)

# Análisis y Desarrollo de Software

## ADSO



El empleo  
es de todos

Mintrabajo

Evidencias de Conocimientos:

# GA3-220501093-AA2-EV01:

## Fundamentos de programación estructurada y estructuras cíclicas.

Por:

Gustavo Salazar Escobar  
Instructor Técnico



Centro De Comercio y Servicios  
Regional Cauca



@SENAcomunica

[www.sena.edu.co](http://www.sena.edu.co)



## **GA3-220501093-AA2-EV01:** Fundamentos de programación estructurada y estructuras ciclicas.



# Análisis y Desarrollo de Software

## AGENDA

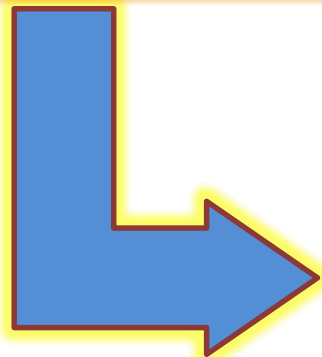


- BIENVENIDA
- Socialización Evidencia **GA3-220501093-AA2-EV01**
- Resolución de inquietudes
- Conclusiones

# Análisis y Desarrollo de Software ADSO



**1- Competencia 220501093** - Evaluar requisitos de la solución de software de acuerdo con metodologías de análisis y estándares



**Actividad de Aprendizaje GA3-220501093-AA2:** Desarrollar soluciones algorítmicas a través de estructuras de control.

# Análisis y Desarrollo de Software ADSO



| ACTIVIDAD DE APRENDIZAJE                                                                             | RESULTADO DE APRENDIZAJE (RAP)                                                                 | Evidencia a Aprobar                                                                               | Fecha de Entrega              |
|------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|-------------------------------|
| <b>GA3-220501093-AA2:</b><br>Desarrollar soluciones algorítmicas a través de estructuras de control. | <b>220501093-03:</b> Desarrollar procesos lógicos a través de la implementación de algoritmos. | <b>GA3-220501093-AA2-EV01:</b><br>Fundamentos de programación estructurada y estructuras cíclicas | 27/09/2022<br>a<br>11/10/2022 |

## Evidencias de Conocimiento

GA3-220501093-AA2-EV01

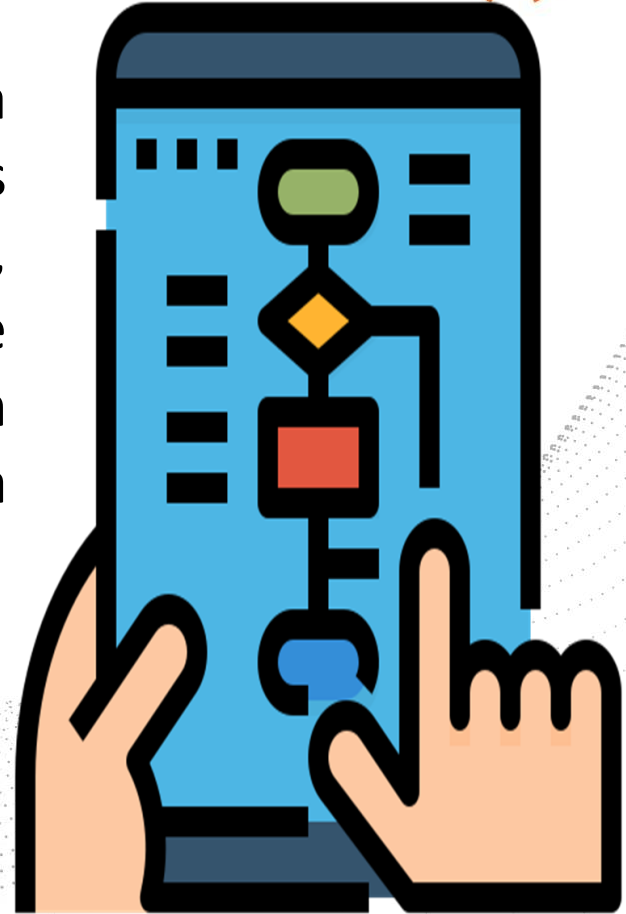


Materiales de formación: para el desarrollo de esta actividad, es importante la lectura y el análisis del material de formación “**Análisis y solución de problemas aplicando algoritmos**”.

# Análisis y solución de problemas aplicando algoritmos

El avance tecnológico lleva implícito la frontera del conocimiento, esto a su vez genera nuevos conceptos, terminologías, notaciones, nomenclaturas y su representación a través de **diagramas, planos o artefactos** que terminan convirtiéndose en un lenguaje natural de la tecnología en cuestión.

En este componente se presentan los **artefectos, diagramas y nomenclaturas** con los que se describen los algoritmos.



# 1. Metodología de algoritmos

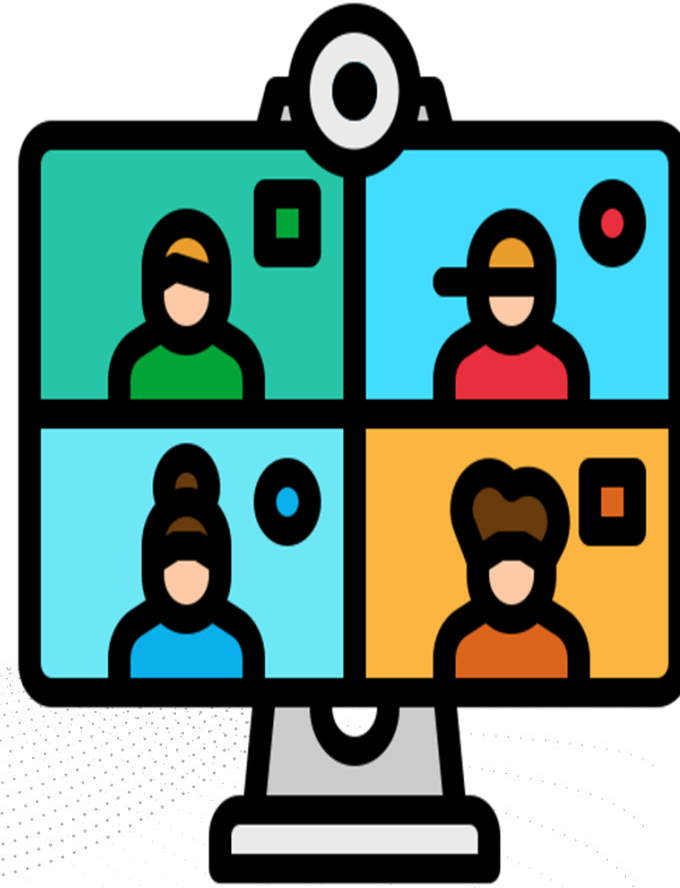


De manera implícita, en las distintas áreas productivas de la sociedad se **usan algoritmos** por ejemplo, **en la construcción de una casa**, el ingeniero o el arquitecto **analiza el terreno, diseña unos planos, realiza maquetas** y así se sigue una **secuencia lógica de procesos**, para la resolución del problema. En el segundo ejemplo un abogado antes de defender a un cliente **analiza las pruebas, elabora un plan, practica sus argumentos** y sigue **un método para lograr la defensa de su cliente**. Se puede concluir, que una solución a la medida de un problema requiere una metodología que lleve a la solución final.

## 1. Metodología de algoritmos



En el proceso de desarrollo de programas informáticos, aparte de conocer esta metodología, se necesita conocer las funciones que puede realizar un computador y la representación de las mismas.

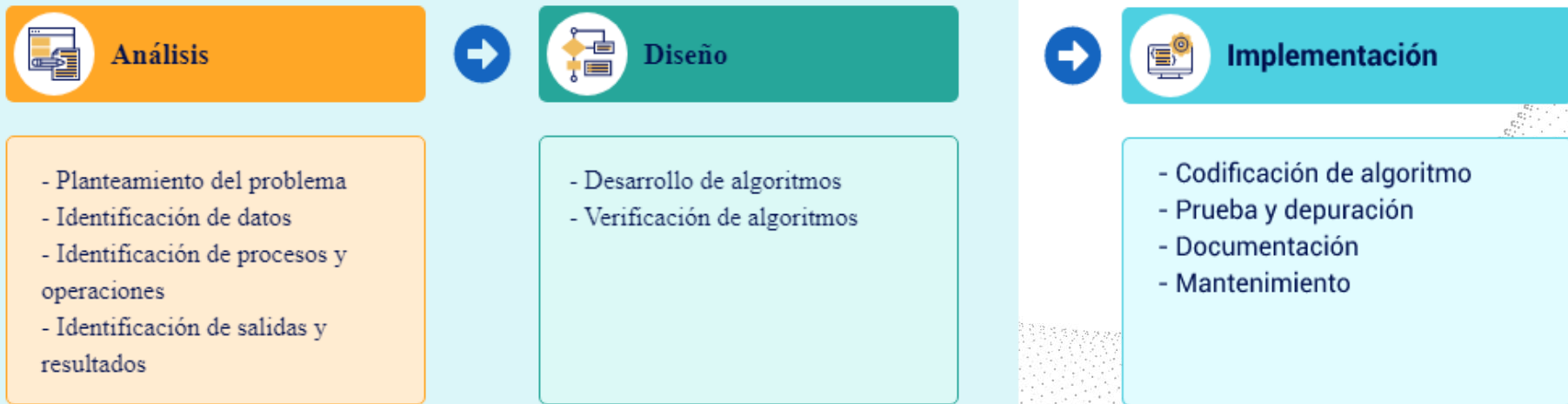




# 1. Metodología de algoritmos



Existen tres (3) fases secuenciales para la **elaboración de un algoritmo** representadas en la siguiente figura y de las cuales se abordarán las dos primeras:



## 1.1. Análisis



la fase de análisis trata de responder a la pregunta: ¿**Qué es lo que se debe hacer?**, y esto es necesario para **entender el problema**. Para ello es preciso *relacionar los datos de entrada, salida o resultados con los procesos a ejecutar con esta información y los datos auxiliares que se puedan requerir en las tareas.*



## 1.1. Análisis

El éxito de un algoritmo consiste en reducir a cero la brecha de lo que el cliente necesita y los resultados que el algoritmo entrega.

Para lograrlo en su totalidad, es importante indagar otros aspectos que corresponden a la naturaleza del problema, las siguientes preguntas modeladoras son de referencia casi obligante a tener en cuenta:

**¿Qué es?**  
**¿Quiénes**  
**intervienen?**  
**¿Cómo interviene**  
**o se comporta?**  
**¿Cuándo?**  
**¿Cómo lo hace?**  
**¿Por qué? y**  
**¿Para qué?**

## 1.2. Diseño

Se responde, básicamente, a la pregunta **¿Cómo se va a hacer?**, refiriéndose a **la manera en la que el sistema va a realizar la tarea solicitada**. Debe quedar bien definido el problema, sin dar espacio a **más de una interpretación** de la acción o acciones que debe hacer el sistema.

*En el proceso de desarrollo del algoritmo, las siguientes actividades son obligatorias:*



- Definir las partes del algoritmo



- Pasos para el desarrollo del algoritmo



- La representación del algoritmo  
(notación, diagrama o pseudocódigo)

Para la verificación del algoritmo, se aplica una técnica denominada **pruebas de escritorio o trazas**, que trata de la **validación y verificación del algoritmo** mediante la ejecución de las sentencias u operaciones que lo componen (**proceso**), y así determinar sus resultados de salida a partir de una base determinada de elementos en la entrada del proceso.

Como el resultado del proceso de diseño es un algoritmo, es obligatorio saber representarlo de alguna manera, y se empleará inicialmente con la notación de pseudocódigo para hacer este tipo de representaciones.



### 1.2.1. Notación de diseño: pseudocódigo



Para expresar el contenido de un programa de computadora, se necesita de un **lenguaje de programación que las computadoras pueden interpretar**. Sin embargo, se puede expresar ese mismo contenido en **el lenguaje natural** lengua materna (español), empleando expresiones sencillas que lo simplifican y abrevian. Cuando se emplea esta técnica, se dice que está escrito en pseudocódigo.





**Ejemplo:** El cliente pide que se desarrolle un programa de computadora **que le solicite al usuario un número, luego el sistema debe mostrar en pantalla el número digitado.**

**Análisis:** el dato de entrada debe ser un número ingresado por el usuario, pero antes de eso el sistema debe solicitarle al usuario que lo ingrese (**digite**) al programa, si no, *¿de qué otra manera el usuario sabría que el programa está esperando el ingreso de ese dato?* Por lo tanto, una posible operación inicial dentro del algoritmo sería **escribir en pantalla** que “**por favor se ingrese un número**”.

La segunda operación es la de **recibir ese número desde el teclado**, para ingresarlo al programa, pero esto implica que se debe definir un dato **intermedio donde se guardará ese valor** y el cual identificará el valor ingresado (**a este dato se denomina variable porque puede tener cualquier valor entero según digite el usuario**). Para guardar el dato (variable) se debe definir dónde se almacenará o como podría referirse al dato ingresado dentro del programa, por lo tanto, antes de empezar el algoritmo se debe indicar al computador que **reserve un espacio para guardar un número entero**.

Una vez definida la variable, es posible referirse a ella, leer desde el teclado el número entero y almacenarlo en dicha variable.

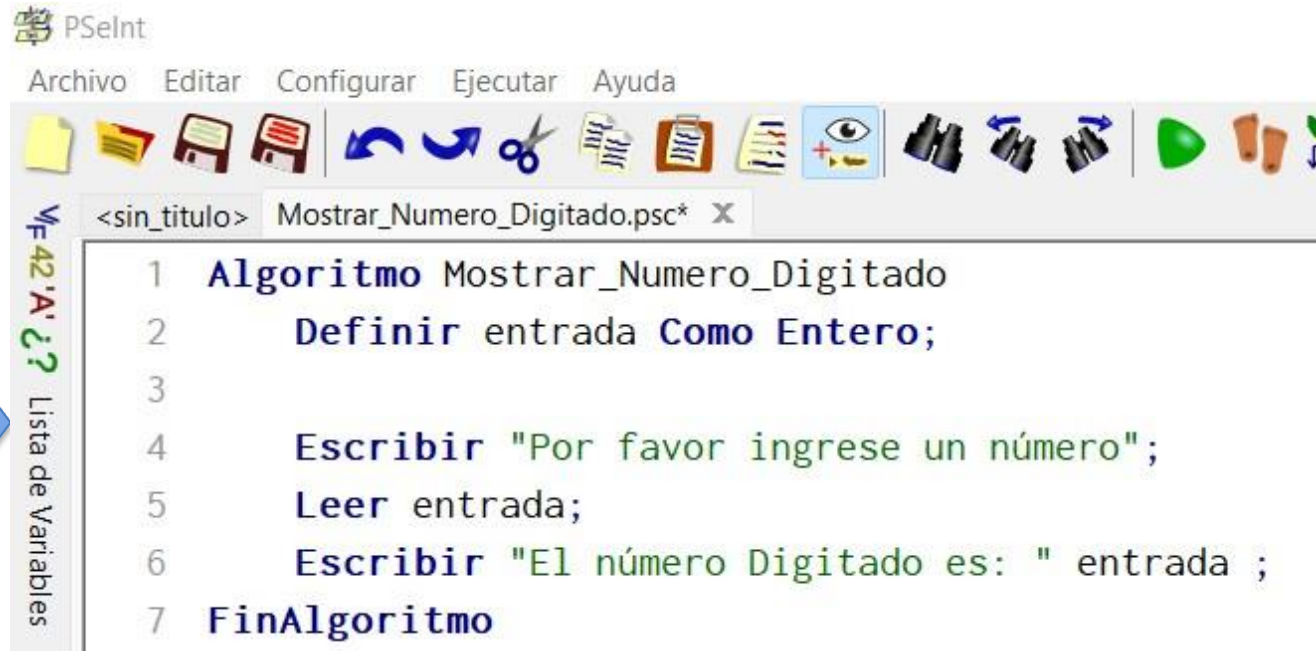
Por último, **el sistema debe imprimir** el contenido de la variable (**dato de salida**), con lo cual se puede dar final al programa.



**Diseño:** la siguiente figura tiene la representación del resultado al análisis anterior.



Para el  
Diseño  
utilizaremos  
el Programa  
PSint

A screenshot of the PSint program interface. The window title is "PSint". The menu bar includes "Archivo", "Editar", "Configurar", "Ejecutar", and "Ayuda". The toolbar contains icons for file operations (new, open, save, print), editing (undo, redo, cut, copy, paste), and execution (run, stop). The main editor shows a file named "Mostrar\_Numero\_Digitado.psc\*" with the following pseudocode:

```
1  Algoritmo Mostrar_Numero_Digitado
2      Definir entrada Como Entero;
3
4      Escribir "Por favor ingrese un número";
5      Leer entrada;
6      Escribir "El número Digitado es: " entrada ;
7  FinAlgoritmo
```

On the left side of the editor, there is a vertical toolbar with icons for "Ver" (eye), "F4" (key), "42" (number), "A" (letter), "?", and "Lista de Variables" (list icon).

pseudocódigo imprimir un número



**ALGORITMO** indica el nombre del algoritmo, después se declaran las variables que serán usadas en él (con **Definir**) indicando su tipo de dato (un valor de tipo **ENTERO**, para este ejemplo).

Entre **Algoritmo** y **FinAlgoritmo** se escriben las diferentes acciones que realizará el algoritmo y usamos **ESCRIBIR** para indicar con esta palabra que se **mostrará un mensaje por pantalla** (el texto que se va a mostrar 'tal cual' debe ir entre comillas dobles), y **LEER** para recibir el dato desde el teclado.

Se debe tener en cuenta que cada línea termina con un enter o en punto y coma (excepto **Algoritmo** y **FinAlgoritmo**), Enter o punto y coma indica que lo que se ejecuta es una **sentencia** que hace una única operación, mientras que lo que está entre **Algoritmo** y **FinAlgoritmo** agrupa varias sentencias que se ejecutan en un orden determinado.

### 1.2.2. Pruebas de escritorio o trazas

Son las simulaciones de ejecución de un algoritmo que permite determinar la **validez o efectividad** del mismo. Se trata de escribir en una tabla con tantas columnas como variables tiene el algoritmo seguir las instrucciones poniendo los valores correspondientes.

| — | — | == |
|---|---|----|
| — |   |    |
|   |   |    |
|   |   | —  |



Para entenderlo se tiene que recorrer línea a línea el pseudocódigo y así identificar tanto las variables como condiciones.



**Ejemplo:** Escribir un algoritmo que **calcule el cuadrado y el cubo de un número** introducido por teclado y **mostrar los resultados para el cubo y el cuadrado.**

**Análisis:** El dato de entrada debe ser un **número ingresado por el usuario.**

Antes de realizar las *operaciones de elevar al cuadrado y al cubo*, se necesita **definir las dos variables que recibirán estos datos** (el cuadrado, y el cubo), por lo tanto, el algoritmo debe definirlas antes de realizar las operaciones y asignarles a ellas los resultados.

Una vez realizadas las operaciones **el algoritmo debe mostrar en pantalla el contenido de estas variables**, con un mensaje texto que indique si es el resultado del cuadrado o del cubo.

**Diseño:** El siguiente pseudocodigo en Psint tiene la representación del resultado al análisis anterior.

```
1  Algoritmo Calcular_el_Cuadrado_y_Cubo
2      Definir entrada Como Entero;
3      Definir  cuadrado Como Entero;
4      Definir cubo Como Entero;
5
6      Escribir "Por favor ingrese un número";
7      Leer entrada;
8      cuadrado = entrada * entrada;
9      cubo = entrada * entrada * entrada;
10     Escribir "El cuadrado es " cuadrado;
11     Escribir  "El cubo es " cubo
12 FinAlgoritmo
13
```

# Análisis y Desarrollo de Software ADSO



Las **pruebas de escritorio** permiten verificar que el pseudocódigo realice lo que se espera al resolver el problema. Sin embargo.

La mayoría de las pruebas de escritorio se representan en una **tabla**, donde en cada columna se anotan **las variables y condiciones** si las hay, en el orden en que aparecen en el algoritmo, para entenderlo se tiene que recorrer línea a línea el pseudocódigo y así identificar tanto las variables como condiciones e ir anotándolas en la tabla de la prueba de escritorio.

Es importante que en la tabla no aparezcan **variables o condiciones repetidas** y en la última columna se pueden anotar los datos de salida para entonces también anotar “Salida”.

| — | — | == |
|---|---|----|
| — |   |    |
|   |   |    |
|   |   |    |



```
1  Algoritmo Calcular_el_Cuadrado_y_Cubo
2      Definir entrada Como Entero;
3      Definir  cuadrado Como Entero;
4      Definir cubo Como Entero;
5
6      Escribir "Por favor ingrese un número";
7      Leer entrada;
8      cuadrado = entrada * entrada;
9      cubo = entrada * entrada * entrada;
10     Escribir "El cuadrado es " cuadrado;
11     Escribir "El cubo es " cubo
12 FinAlgoritmo
```

| Número | Cuadrado | Cubo | Salida                                                                           |
|--------|----------|------|----------------------------------------------------------------------------------|
| 4      | 16       | 64   | Por Favor ingrese<br>un número:<br>4<br><br>El cuadrado es: 16<br>El cubo es: 64 |
|        |          |      |                                                                                  |
|        |          |      |                                                                                  |
|        |          |      |                                                                                  |

## 2. Diagramas de flujo



Consiste en una **representación por medio de un gráfico**. Mientras en pseudocódigo las sentencias se representaban con preposiciones, En diagramas de flujo la preposición es reemplazada por una figura.

Algoritmo Mostrar\_Numero\_Digitado

Inicio del algoritmo

Definir entrada Como E...

Declara la variable  
entrada como número entero

'Por favor ingrese un ...

Muestra en Pantalla  
el mensajes que  
ingrese un número

entrada

Leemos del teclado un número

'El número Digitado es...

Muestra en Pantalla  
el valor del número  
digitado

FinAlgoritmo

Final del algoritmo



Algoritmo Calcular\_el\_Cuadrado\_y...

Inicio del algoritmo

Definir entrada Como E...

Definir cuadrado Como ...

Definir cubo Como Entero

Declaración de variables

'Por favor ingrese un ...

Muestra en Pantalla el mensajes que ingrese un número

entrada

Leemos del teclado un número

$\text{cuadrado} \leftarrow \text{entrada} * \text{ent}...$

Calcula el cuadrado del número digitado

$\text{cubo} \leftarrow \text{entrada} * \text{entrada}...$

Calcula el cubo del número digitado

'El cuadrado es ',cuad...

Muestra el cuadrado del número digitado

'El cubo es ',cubo

Muestra el cubo del número digitado

FinAlgoritmo

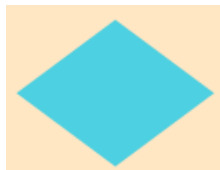
Final del algoritmo

# Los siguientes son los principales simbolos para crear diagramas de flujo:



## Inicio/Final

Se utiliza para indicar el **inicio y el final de un diagrama**; del Inicio solo puede salir una línea de flujo y al Final solo debe llegar una línea



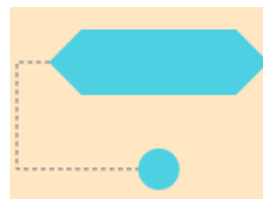
## Decisión

Indica la **comparación de dos datos** y dependiendo del **resultado lógico** (falso o verdadero) se toma la decisión de seguir un camino del diagrama u otro



## Entrada General

Entrada/Salida de datos



## Iteración

Indica que una instrucción o grupo de instrucciones **deben ejecutarse varias veces**



## Entrada por teclado

Instrucción de entrada de datos por teclado. Indica que el computador debe esperar a **que el usuario teclee un dato que se guardará en una variable o constante**



## Salida Impresa

Indica la presentación de uno o varios resultados en forma impresa



## Llamada a subrutina

Indica la llamada a una subrutina o procedimiento determinado



## Salida en Pantalla

Instrucción de presentación de mensajes o resultados en pantalla



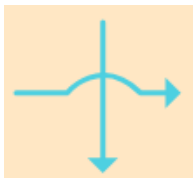
## Acción/Proceso General

Indica una acción o instrucción general que debe realizar el computador (cambios de valores de variables, asignaciones, operaciones aritméticas, etc.)



## Conector

Indica el enlace de dos partes de un diagrama dentro de la misma página



## Flujo

Indica el seguimiento lógico del diagrama. También indica el sentido de ejecución de las operaciones



## Conector

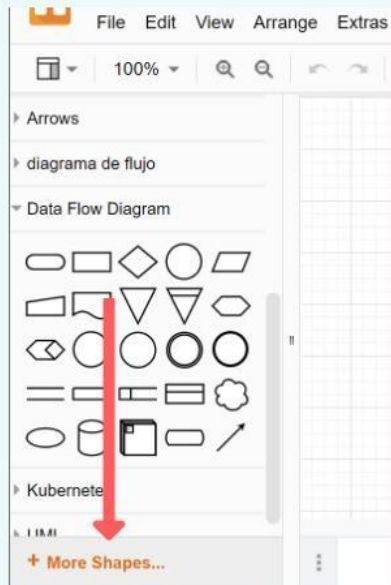
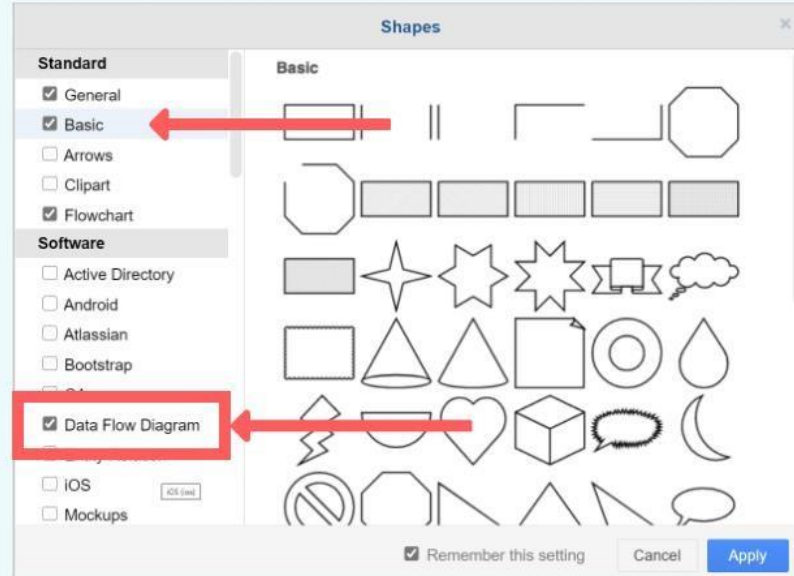
Indica el enlace de dos partes de un diagrama en páginas diferentes



# Herramientas digitales para creación y prueba de algoritmos



Para aprender a reconocer los elementos que componen los diagramas de flujo se puede hacer uso de la herramienta denominada **Draw.io** a través del enlace <https://app.diagrams.net/> con la que puede usar una cuenta de correo de Google, o misena.edu.co para acceder y crear un nuevo diagrama.



En caso de contar con intermitencia de internet se puede trabajar con el programa de libre distribución disponible en <http://dia-installer.de/> o puede acceder a la versión de escritorio de Draw.io disponible en <https://app.diagrams.net/>



The screenshot shows the Dia Editor website. The header includes the Dia logo and the text "Editor de diagramas Dia". Below the header is a navigation bar with links: "Productos", "Descargas", "Formas", "Documentación", and "Apoyo". Under "Descargas", there are sub-links: "Diámetro", "Diashapes", "vsd2svg", "dia2code", and "Parse-Dia-Sql". The main content area is titled "Visión general" and describes Dia as a program for drawing structured diagrams. A large blue button with a white download arrow icon contains the text "Día 0.97.2", "Descargar gratis", and "Windows, 60 idiomas (20 MB)". Below this button are links for "Notas" and "Mac OS X Linux". The section "Captura de pantalla" shows a preview of the Dia application window. The window title is "Diagramm1.dia - diaw.exe". The menu bar includes "File", "Edit", "View", "Layers", "Objects", "Select", "Tools", "Input Methods", "Dialogs", "Debug", "Help", and "Test". The toolbar contains various icons for file operations, editing, and viewing. The main canvas shows a diagram with a ruler at the bottom.

El programa de mayor uso para el diseño y depuración de algoritmos en pseudocódigo es **PSeInt** disponible en <http://pseint.sourceforge.net/>.



...una invitación a entrar en el maravilloso mundo de la programac

Portada Noticias Descargar Documentación Foros

**PSeInt** es es una herramienta para asistir a un estudiante en sus primeros pasos en programación.

Descárguelo ahora (versión: 20210906)...

 **PSeInt para GNU/Linux**

 **PSeInt para Windows**

 **PSeInt para macOS**

Descubra sus principales funcionalidades...

Información para docentes...



### 3. Uso de identificadores y palabras reservadas



- ✓ En la elaboración de diagramas de flujo o algoritmos muchas veces se necesitan datos auxiliares.
- ✓ Los identificadores son nombres que se dan a los elementos utilizados para resolver un problema y poder diferenciar unos de otros

- ☐ Pueden estar formados por una combinación de letras y números (**saldoMes**, **salario**, **fecha2**, etc.).
- ☐ El primer carácter de un nombre debe ser una letra.
- ☐ La mayoría de los lenguajes de programación **diferencian las mayúsculas de las minúsculas**.
- ☐ Los nombres deben ser **nemotécnicos**, con solo leerlos se puede entender lo que contienen.
- ☐ Deben ser muy descriptivos; no utilizar abreviaturas, a menos que se justifique.
- ☐ Es conveniente utilizar una **sola palabra** para nombrar páginas, controles, variables, etc.
- ☐ No utilizar caracteres reservados (% , + , / , > , etc.) tampoco letras acentuadas (á , é , ñ , ó , ú).
- ☐ No utilizar palabras reservadas por los lenguajes de programación.

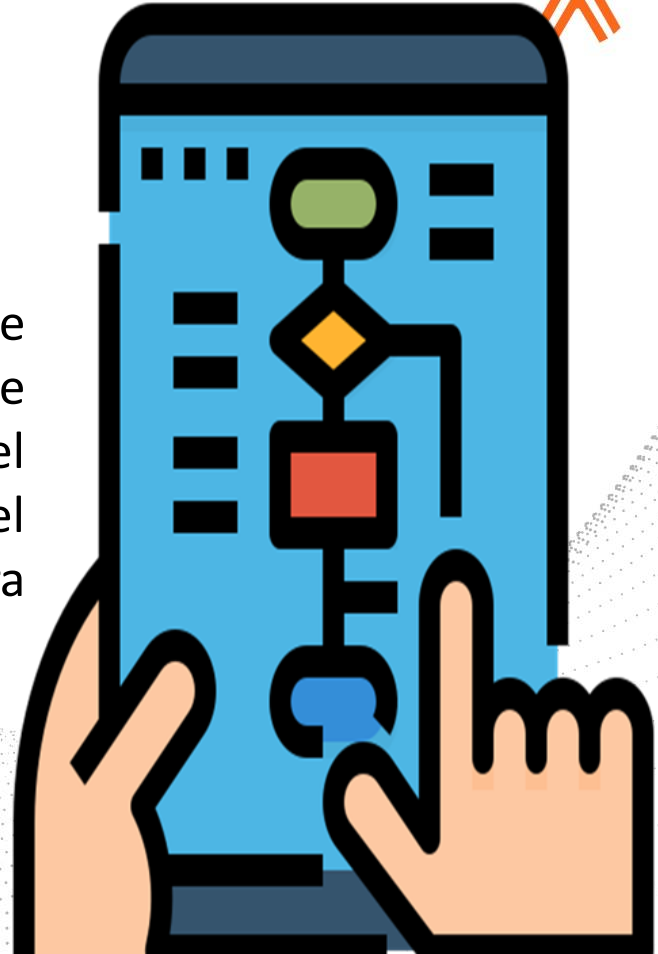
## Ejemplo: listar los identificadores usados en los algoritmos previos.



**Solución:** en el algoritmo de imprimir un número entero usando el identificador de **entrada** para representar el número digitado.

En el algoritmo de imprimir el **cuadrado** y el **cubo** de un número entero empleando el identificador de entrada para representar el número digitado, el identificador cuadrado para representar el valor del cuadrado de entrada y el identificador cubo para almacenar el cubo entrada.

Todos los lenguajes de programación tienen unas palabras para nombrar sus comandos, instrucciones y funciones; estas palabras no se pueden usar como indicadores.





## a. Variables

Para los algoritmos con diversos conjuntos de datos iniciales, se debe determinar una independencia **entre los datos iniciales de un problema y la estructura de su solución**. Esto se hace mediante la utilización de variables (cantidades que se suelen emplear y que durante la ejecución del algoritmo **pueden tomar cualquier valor de un intervalo de valores posibles**).

Las variables son espacios de trabajo (contenedores) dispuestos para almacenar datos (valores). El valor de una variable puede ir cambiando en algún paso del algoritmo o permanecer sin cambios; por lo tanto, el valor de una variable es el del último valor o dato asignado a esta. En el segundo ejemplo de algoritmos, las variables eran los indicadores **cuadrado y cubo**.



## b. Constantes

Las constantes se crean de la misma forma que las variables, a diferencia que **se procura su definición o creación a inicio del algoritmo**. Las constantes consisten en datos que, luego de ser asignados, **no cambian en ninguna instrucción del algoritmo**; pueden contener constantes matemáticas (pi).



## c. Contadores

Un **contador** es una variable que estfi en ambos lados de una asignación (a la derecha y a la izquierda) interna, a la que se le suma un valor **constante**. Es necesario haberla iniciado al principio del algoritmo, ya que va a ser lerdido su contenido.



sin\_titulo>\* X

```
1  Algoritmo Contador_  
2      Definir contador Como Entero;  
3  
4      contador = 0;  
5      contador = contador + 1;  
6      contador = contador + 1;  
7      contador = contador + 1;  
8      Escribir "El valor de contdor es: " contador;  
9  FinAlgoritmo
```

En este algoritmo, **tres veces se incrementa en 1 (uno) el valor del contador** y al final de la ejecución **su valor es de 3**.

Contador inicia con un valor, el **valor 0**, si esta asignación inicial no se hace el computador pondrra un valor aleatorio y el valor al final del algoritmo serra indeterminado.

## **d. Acumuladores**

Los acumuladores son una versión ampliada de contadores, tienen las mismas características que un contador excepto el valor que se va incrementando no es una constante sino una que es un valor variable.

## 4. Operadores y jerarquía en los operadores



**Los operadores son símbolos que sirven para manipular datos.** Y en un algoritmo es necesario dejar un espacio en blanco a cada lado del signo aritmético.

- ✓ **Aritméticos:** *permiten realizar operaciones entre datos de tipo numérico y dan como resultado otro valor de tipo numérico.* Ejemplo: producto o **multiplicación** (\*); **división** (/); **suma** (+); **resta** (-); **asignación** (=).
- ✓ **Alfanuméricos:** *permiten operar con datos de tipo carácter (letras) o cadenas de texto.* La mayoría de los lenguajes de programación admiten el operador **+** para realizar la concatenación (unión) de caracteres o cadenas.
- ✓ **Relacionales:** *permiten la comparación entre datos del mismo tipo de dato y dan como resultado dos valores posibles: verdadero o falso.* Ejemplo: **igual a** (=); **menor que** (<); **mayor que** (>), **mayor e igual que** (>=).
- ✓ **Lógicos:** *posibilitan la evaluación lógica de dos expresiones de tipo lógico.* Dan como resultado uno de dos valores posibles: **verdadero o falso**. Ejemplo: negación (no); conjunción (y); disyunción (o).



# Jerarquía en los operadores

Las computadoras ejecutan los operadores en un orden establecido y el siguiente es el orden (jerarquía) para ejecutar operadores:



1. **Paréntesis** (se ejecutan primero los más internos).
2. **Signo** (-2), si un valor es positivo o negativo.
3. **Potencias** (^) y Raíces (sqrt); Productos y Divisiones (\* y /), en este mismo orden.
4. **Sumas y Restas** (+ y -).
5. **Concatenación** (+).
6. **Relacionales** (=, <, >).
7. **Negación** (no).
8. **Conjunción** (y).
9. **Disyunción** (o).

$$(2*3-2)^2 - 4*8/(2^3/4) = ?$$

$$\begin{array}{ccc} \underbrace{2*3-2}_{6} & & \underbrace{2^3/4}_{8} \\ \underbrace{\quad}_{4} & \underbrace{4*8}_{32} & \underbrace{\quad}_{2} \\ \underbrace{\quad}_{16} & \underbrace{\quad}_{16} & \end{array}$$

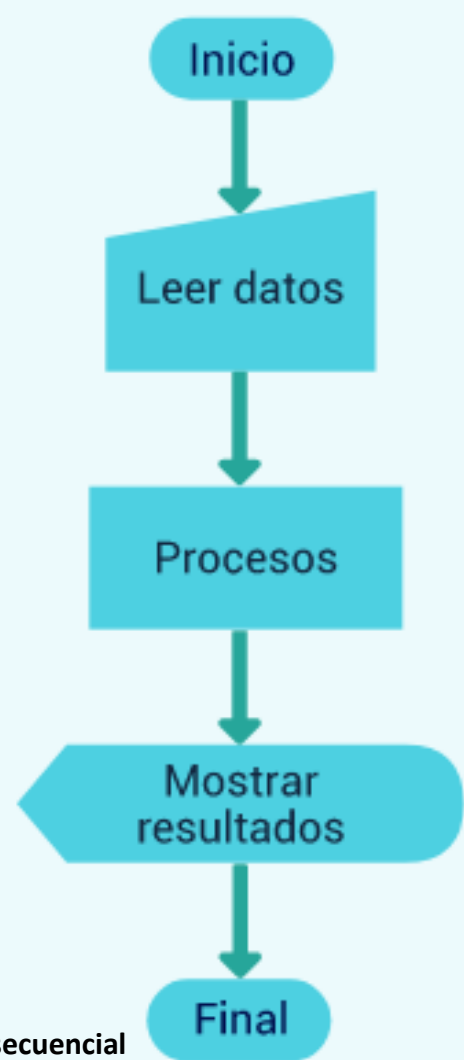
$$16 - 16 = 0$$

## 5. Estructuras de control secuencial

Una estructura es definida como un esquema con determinada distribución y orden que permite representar una **idea de forma simplificada**.

La **estructura de control secuencial** es la más sencilla y es conocida como estructura lineal. Se compone de instrucciones que deben ejecutarse en forma consecutiva, una tras otra, siguiendo una línea de flujo.

Los problemas más sencillos pueden resolverse haciendo uso únicamente de esta estructura. Por lo general, la estructura secuencial forma parte de soluciones a problemas complejos, en las que se la utilizan otras estructuras iterativas (repetir varias veces un conjunto de instrucciones) y condicionales



## 5.1 Ejemplo estructura de control secuencial

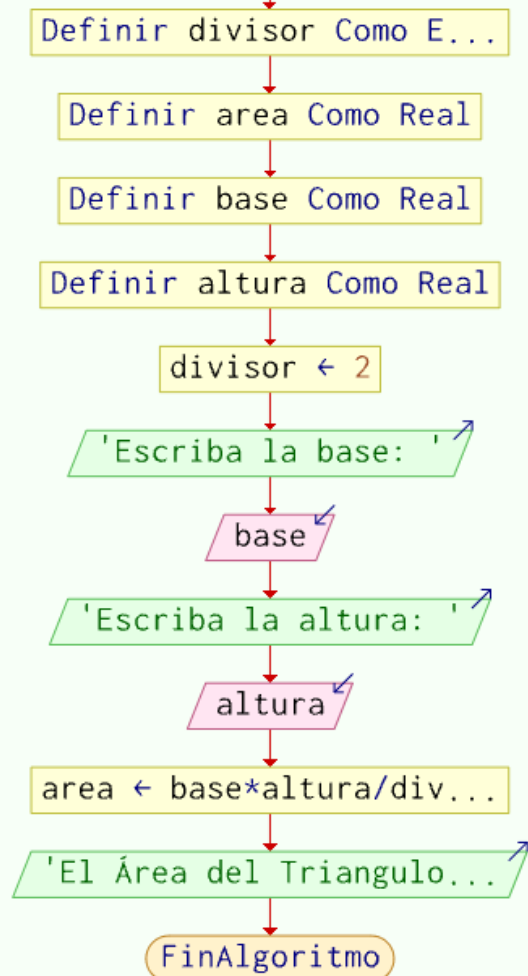
Escribir un algoritmo para **calcular el área de cualquier triángulo rectángulo**, en él se debe pedir al usuario que *ingrese los valores de la altura y la base del triángulo*

### Análisis

- ❑ **Formular el problema:** está claramente planteado y consiste en calcular el área de un rectángulo.
- ❑ **Resultados esperados:** se espera que el algoritmo sea capaz de *calcular el área del rectángulo* y mostrar en la pantalla el resultado de ese cálculo.
- ❑ **Datos disponibles:** dos datos, números que representan **la base y la altura** del triángulo (*se deben solicitar al usuario*). Como aprendiz se debe preguntar si mis conocimientos actuales de matemáticas me permiten resolver este problema, de no ser así, tendrá que plantear una estrategia para obtener los conocimientos requeridos. Para este ejemplo recordar que un triángulo rectángulo es aquel que uno de sus lados tiene un ángulo recto (90 grados).
- ❑ **Restricciones:** los valores de base y altura son variables y se deben solicitar al usuario.
- ❑ **Procesos necesarios:** definir variables; asignar **el valor 2** a la constante usando para dividir (divisor); solicitar al usuario el valor de la altura del triángulo; solicitar al usuario el valor de la base; aplicar la fórmula de área; mostrar el resultado.



## Algoritmo Area\_De\_Un\_Triangulo



## Diseño: El algoritmo resultado del análisis



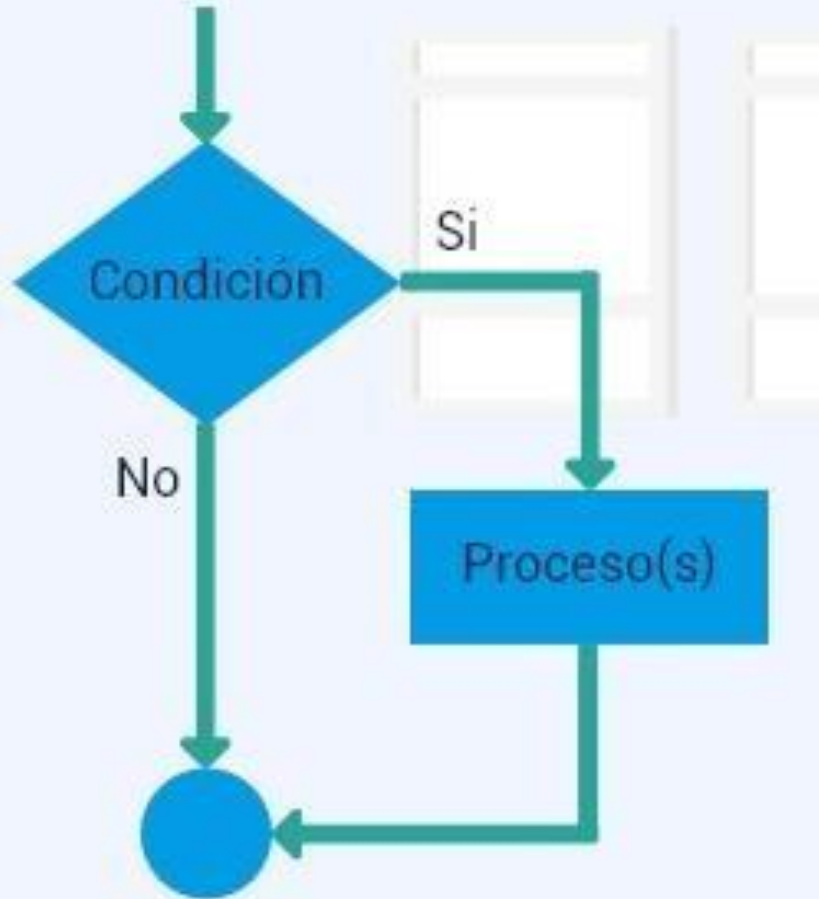
Diagrama de Flujo

Pseudocódigo

<sin\_titulo>\* X

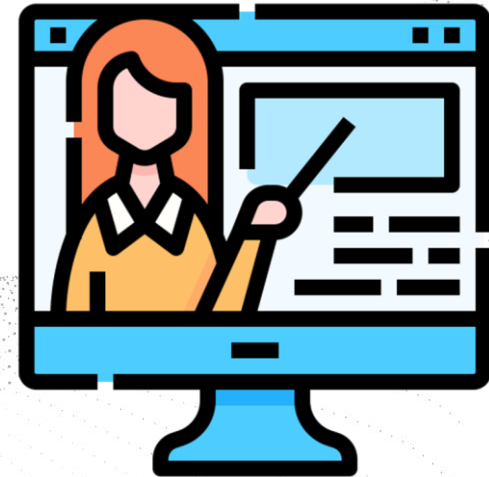
```
1  Algoritmo Area_De_Un_Triangulo
2      Definir divisor Como Entero
3      Definir area Como Real
4      Definir base Como Real
5      Definir altura Como Real
6      divisor ← 2
7      Escribir 'Escriba la base: '
8      Leer base
9      Escribir 'Escriba la altura: '
10     Leer altura
11     area ← base*altura/divisor
12     Escribir 'El Área del Triangulo es: ' area
13 FinAlgoritmo
14
```

## 5.2. Estructura



La **estructura condicional** se emplea para indicar al computador que debe evaluar una condición y, según el resultado, ejecutar el bloque de instrucciones.

La forma más común está compuesta por una condición que se evalúa y dos bloques de instrucciones que se ejecutan: **uno cuando la condición es verdadera y otro cuando la condición es falsa.**



**Ejemplo:** un aprendiz aprueba un examen cuando la calificación de este es mayor o igual a 3. Elaborar un algoritmo que pida al usuario una calificación, aplicar el criterio de aprobación e **imprimir “Aprobado” o “Reprobado”**, según sea el caso.

- **Formular el problema:** es un problema sencillo de condición con dos operaciones o condición doble.
- **Resultados esperados:** un aviso que reporte si el estudiante obtiene **“Aprobado”** o **“Reprobado”** el examen.
- **Datos disponibles:** la calificación ingresada por el usuario. Para aprobar, la nota debe ser mayor o igual a 3.
- **Restricciones:** aplicar el criterio de aprobación (calificación mayor o igual a 3).
- **Procesos necesarios:** solicitar al usuario que ingrese la calificación. Evaluar si la calificación es igual o superior a 3; en caso de ser verdadero, reportar **“Aprobado”**; en caso contrario, reportar **“Reprobado”**.

Algoritmo Calificacion\_del\_Usuario

Definir calificacion C...

'Escriba la calificacion'

calificacion

(calificacio...

('Reprobado')

'Aprobado'

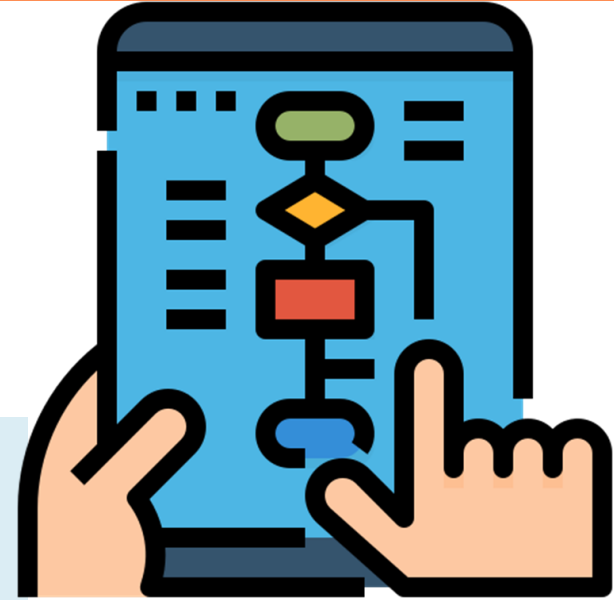
FinAlgoritmo

Diseño:



Area\_De\_Un\_Triangulo.psc <sin\_titulo>\* X

```
1  Algoritmo Calificacion_del_Usuario
2      Definir calificacion Como Real
3
4      Escribir "Escriba la calificacion"
5      Leer calificacion
6      Si (calificacion ≥ 3)
7          Escribir "Aprobado"
8      SiNo
9          Escribir ("Reprobado")
10     FinSi
11 FinAlgoritmo
```



**Evidencias de Conocimiento**

**GA3-220501093-AA2-EV01:**

Fundamentos de programación  
estructurada y estructuras ciclicas.

En esta actividad se estudiarán **algunas formas de representación de algoritmos** y se introducen algunos conceptos clave como: *constantes, variables, acumuladores, contadores, operadores, jerarquía de los operadores*. Finalmente, se abordan las **estructuras de control secuencial** y herramientas existentes para la **creación y pruebas de algoritmos**

# Análisis y Desarrollo de Software ADSO



Evidencia de conocimiento: GA3-220501093-AA2-EV01 Fundamentos de programación estructurada y estructuras ciclicas

Esta evidencia se centra en consolidar los conceptos relacionados con las **metodologías de algoritmos y sus tres fases**: análisis, diseño e implementación. Respecto a la implementación se aborda los **seudocódigos y diagramas de flujo**.



# Análisis y Desarrollo de Software ADSO



Evidencia de conocimiento: GA3-220501093-AA2-EV01 Fundamentos de programación estructurada y estructuras ciclicas

Esta evidencia se centra en consolidar los conceptos relacionados con las **metodologías de algoritmos y sus tres fases**: análisis, diseño e implementación. Respecto a la implementación se aborda los **seudocódigos y diagramas de flujo**.



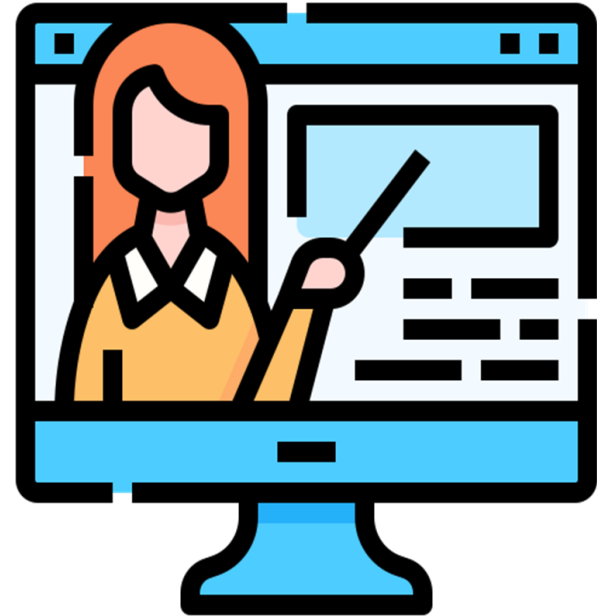


# Análisis y Desarrollo de Software ADSO



Evidencia de conocimiento: GA3-220501093-AA2-EV01 Fundamentos de programación estructurada y estructuras ciclicas

Para su desarrollo es importante la lectura del **componente formativo**. Debe **construir un documento donde registre el análisis, diseño e implementación de algoritmos** que permitan resolver los siguientes problemas:



# Análisis y Desarrollo de Software ADSO



Evidencia de conocimiento: GA3-220501093-AA2-EV01 Fundamentos de programación estructurada y estructuras ciclicas

- ✓ **Algoritmo** que a partir de una fecha de nacimiento y una fecha actual determine la **edad en años actual de una persona**.
- ✓ Algoritmo que permita determinar si un año indicado es o no un año bisiesto.

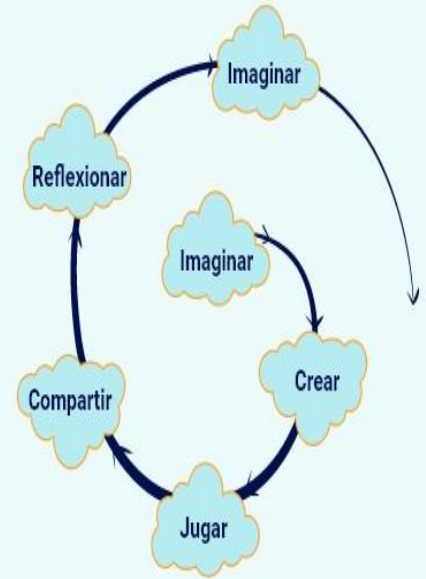


# Análisis y Desarrollo de Software ADSO



Evidencia de conocimiento: GA3-220501093-AA2-EV01 Fundamentos de programación estructurada y estructuras ciclicas

Tener en cuenta respecto al **análisis del algoritmo**, deberá registrar como minimo los elementos de **entrada, salida y los procesos requeridos** para cada uno de los algoritmos solicitados. Respecto a la fase de diseño e implementación plantear **una solución usando diagramas de flujo**.



### Lineamientos generales para la entrega de la evidencia:

- **Producto para entregar:** Un documento con la solución del taller propuesto.
- **Extensión:** Libre.
- **Formato:** PDF.
- Para el desarrollo de la evidencia remitase al área de la actividad correspondiente y acceda al espacio **fundamentos de programación estructurada y estructuras ciclicas GA3-220501093-AA2-EV01**



# Análisis y Desarrollo de Software ADSO



## INSTRUMENTO DE EVALUACIÓN

### Criterios de Evaluación:

- Crea soluciones a problemas mediante algoritmos que incluyen estructuras secuenciales, condicionales y cíclicas..

### LISTA DE VERIFICACIÓN

| No. | VARIABLES/INDICADORES DE LOGRO                                                                                                          | CUMPLE |    | Observaciones |
|-----|-----------------------------------------------------------------------------------------------------------------------------------------|--------|----|---------------|
|     |                                                                                                                                         | SI     | NO |               |
| 1   | El documento entregado presenta el análisis para el primer algoritmo indicando entradas, salidas y proceso requerido para la solución.  |        |    | 10%           |
| 2   | El documento entregado presenta el análisis para el segundo algoritmo indicando entradas, salidas y proceso requerido para la solución. |        |    | 10%           |
| 3   | El documento entregado presenta la implementación del primer algoritmo utilizando diagrama de flujo.                                    |        |    | 20%           |
| 4   | El diagrama de flujo del primer algoritmo utiliza la simbología correctamente y presenta una solución coherente.                        |        |    | 20%           |
| 5   | El documento entregado presenta la implementación del segundo algoritmo utilizando diagrama de flujo.                                   |        |    | 20%           |
| 6   | El diagrama de flujo del segundo algoritmo utiliza la simbología correctamente y presenta una solución coherente.                       |        |    | 20%           |



**G R A C I A S**

Línea de atención al ciudadano: 018000 910270  
Línea de atención al empresario: 018000 910682



@SENAcomunica

[www.sena.edu.co](http://www.sena.edu.co)



¿CÓMO USARLO PARA HACER EL  
TRABAJO EN EQUIPO MÁS EFICIENTE?

## ¿PARA QUÉ SIRVE?:

- Herramienta para asignación de tareas, con responsables y fechas de entrega.
- Se puede utilizar en su versión web o su versión móvil.
- Herramienta gratuita.



# ¿CÓMO USARLO PARA ASIGNACIÓN DE TAREAS?



1. Ingresar a [www.trello.com](https://www.trello.com) ,  
y registrarse.

2. Cada miembro del equipo  
deberá registrarse.

3. Recomendado poner una  
foto, para identificar al  
miembro del equipo.



**Trello es la manera gratuita, flexible y visual de  
organizarlo todo con cualquiera.**

Deje a un lado las largas cadenas de correos electrónicos, las  
hojas de cálculo sin actualizar, las notas rápidas ya no tan  
adhesivas y el software inadecuado para gestionar sus  
proyectos. Trello le deja ver todo sobre su proyecto de un  
solo vistazo.

**Regístrese. ¡Es gratis!**

[Iniciar sesión en...](#)

# ¿CÓMO USARLO PARA ASIGNACIÓN DE TAREAS?



## Crear una cuenta de Trello

Nombre

Correo electrónico

Contraseña

☒ Acepto las [Condiciones del Servicio](#) y la [Política de Privacidad](#)

¿Tiene una cuenta de Google?

4. Diligenciar los datos solicitados y seleccionar registrarse.

5. Una vez registrado, inicie sesión.



## Iniciar sesión en Trello

Correo electrónico (o nombre de usuario)

Contraseña

¿Olvidó su contraseña? [Restablezca la](#)

[Iniciar sesión con SSO](#)

# ¿CÓMO USARLO PARA ASIGNACIÓN DE TAREAS?



6. Define los proyectos en los que vas a trabajar.



## CONSIDERACIONES A TENER EN CUENTA

- Los *tableros*, sirven para identificar proyectos específicos y/o fases del mismo.
- Una vez tu equipo tenga sus cuentas de acceso. Incluye a todos los miembros en el tablero. En la opción de “*configuración*”.
- Es importante que el correo con el que se registre como usuario de Trello, sea de uso “frecuente”. Ya que todas las notificaciones en esta herramienta, serán notificadas en el mail de contacto.

## FORMAS DE CLASIFICAR UN PROYECTO

- El tablero *"Por Hacer"*, se asignan las tareas que hay que desarrollar.
- El tablero *"Haciendo"*, son las tareas que están en desarrollo.
- El tablero *"Revisión"*, son las tareas terminadas por la persona encargada de ejecutarla, pero cuyo responsable del proyecto debe revisar , antes de darla por terminada.
- El tablero *"Periódico"*, son las tareas repetitivas que se deben hacer periódicamente, por ejemplo, presentar informes, actualizar presupuestos, etc.

# ¿CÓMO USARLO PARA ASIGNACIÓN DE TAREAS?



Tableros

IDL DIGITAL

Privado

Trello

Tamara Paez

Por Hacer

- Check list EMAILING  
0/7
- Extraer contenidos valiosos desde Enoch Bartlett Sears AIA
- Manejo de objeciones  
1
- Protocolo comercial  
26 de dic. de 2016 0/1
- Cambiar presentacion  
1

Haciendo

Añadir una tarjeta...

Revision

Añadir una tarjeta...

Periodicos

Añadir una tarjeta...

Añadir una tarjeta...

# ¿CÓMO USARLO PARA ASIGNACIÓN DE TAREAS?

“Añadir una tarjeta” es la forma de asignar una tarea.

7. Al final del cuadro en la parte de abajo dice Añadir una tarjeta, dar *Click*.



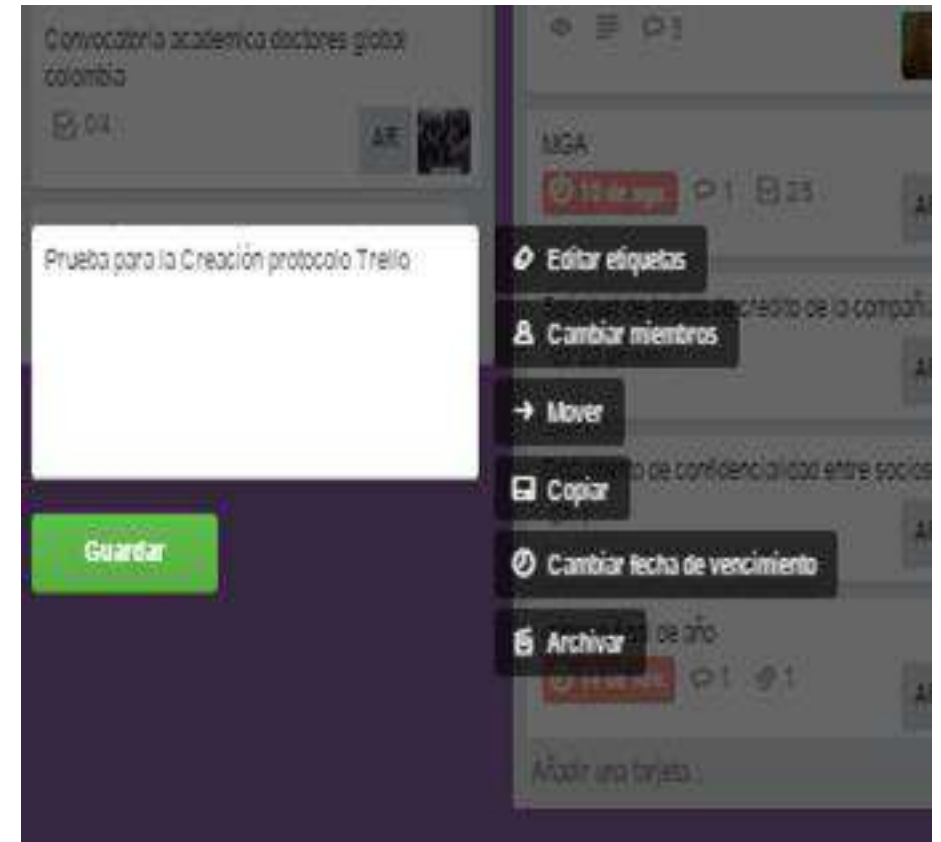
8. Selecciona un título a la tarea, para luego seleccionar “el lapiz”



# ¿CÓMO USARLO PARA ASIGNACIÓN DE TAREAS?

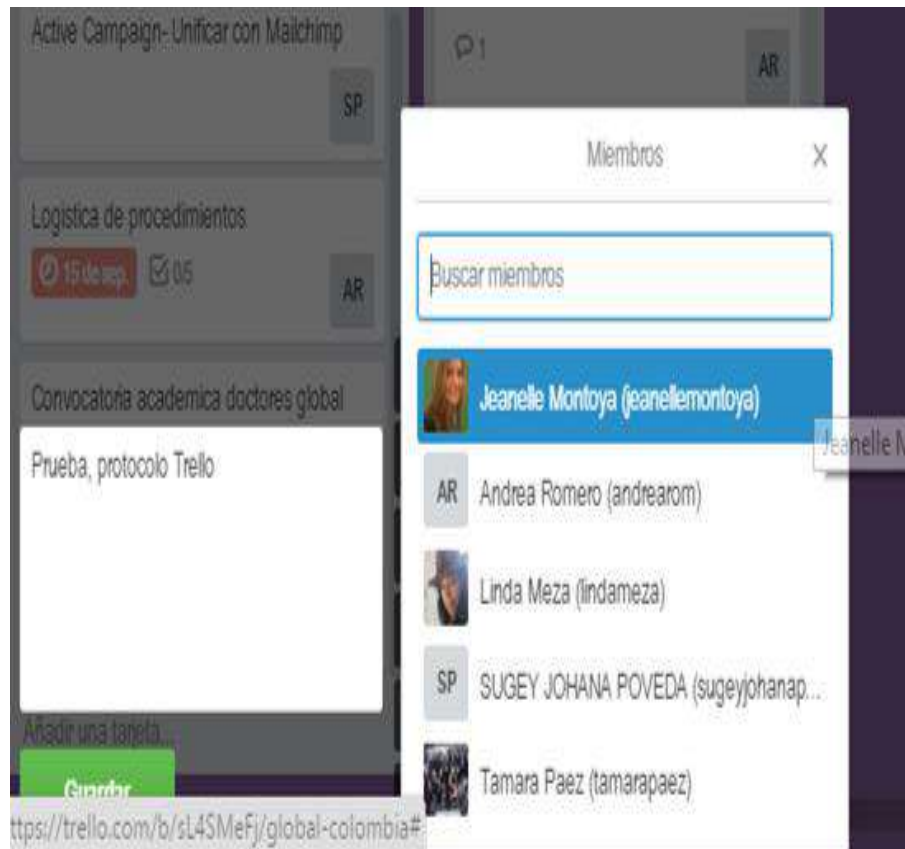


9. Puedes definir, cuando debe estar entregada, puedes incluir archivos; y lo más importante, asignar a la persona del equipo que debe llevarla a cabo.





# ¿CÓMO USARLO PARA ASIGNACIÓN DE TAREAS?



10. Una vez asignada la tarea,  
a la persona le llegará  
Una notificación a su correo.

# ¿CÓMO USARLO PARA ASIGNACIÓN DE TAREAS?

11. Al seleccionar la tarea recién creada, esta se abre para añadir los comentarios respectivos, o si es necesario hacer un Checklist, o adjuntar algún archivo, entre otras opciones.



# ¿CÓMO USARLO PARA ASIGNACIÓN DE TAREAS?



- La tarea se traslada del tablero *"Por Hacer"* a *"Haciendo"* cuando se inicia a ejecutar la tarea.
- Cuando se termina la tarea, se traslada al tablero *"Revisión"*.
- Únicamente quien crea la tarea es quien la archiva, por lo tanto el proceso va hasta *"Revisión"*.

- En el tablero *"Periódico"*: este tablero tiene asignadas las funciones que se hacen de forma periódica y su respectivo responsable.



# CONVERSIÓN MER A MODELO RELACIONAL

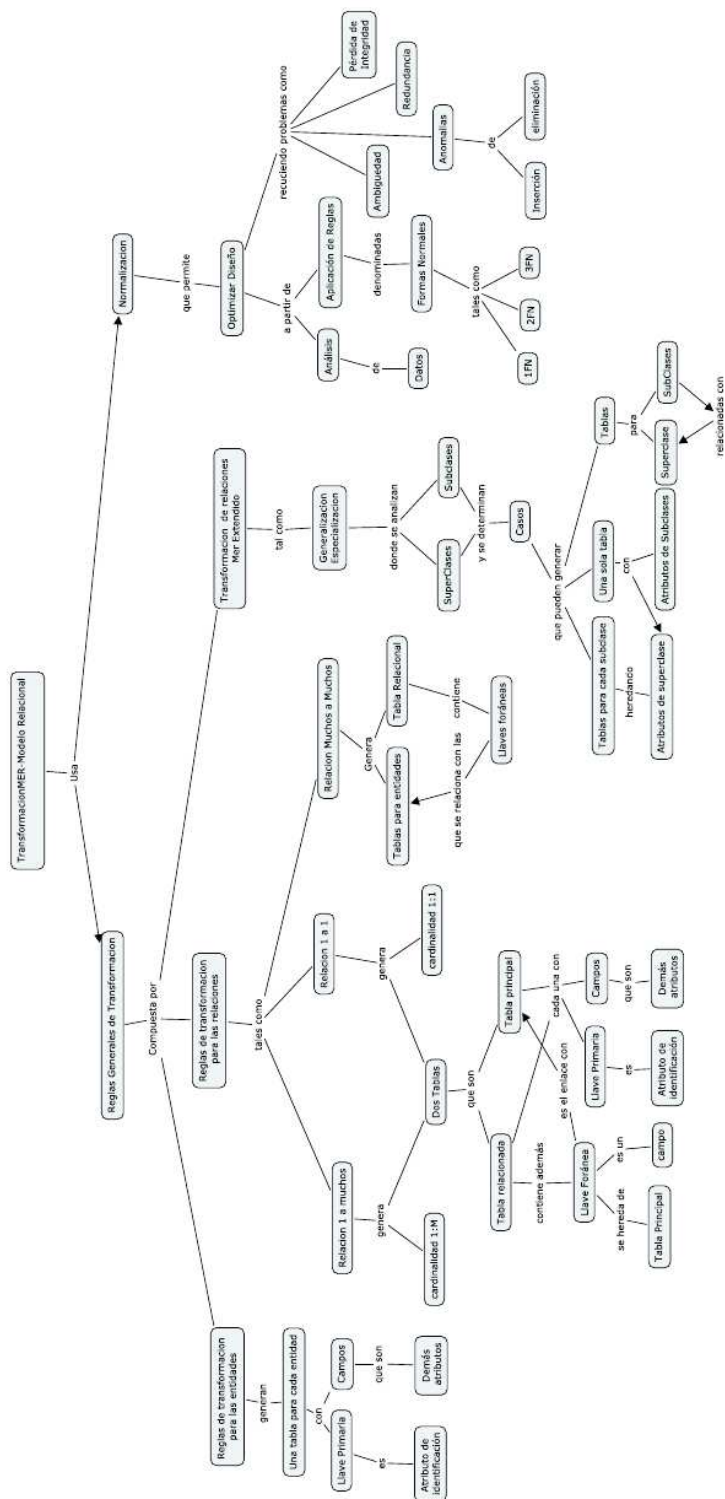
---

## Estructura de contenidos

|                                                                    |    |
|--------------------------------------------------------------------|----|
| 1. INTRODUCCIÓN .....                                              | 3  |
| 2. TRANSFORMACIÓN MER A MODELO RELACIONAL.....                     | 3  |
| 3. REGLAS GENERALES DE TRANSFORMACIÓN .....                        | 5  |
| 3.1 Transformación para las entidades y atributos.....             | 5  |
| 3.2 Transformación para las Relaciones.....                        | 7  |
| 4. TRANSFORMACIÓN MER- EXTENDIDO A RELACIONAL .....                | 11 |
| 5. CONSTRUCCIÓN TRANSFORMACIÓN MER- EXTENDIDO A<br>RELACIONAL..... | 14 |
| 5.1 Relación especialización y generalización .....                | 15 |
| 5.2 Relación Recursiva .....                                       | 20 |
| 6. NORMALIZACIÓN .....                                             | 23 |
| 6.1 Grados de Normalizacion.....                                   | 24 |



# Mapa conceptual CONVERSIÓN MER A MODELO RELACIONAL



### 1. INTRODUCCIÓN

Almacenar información es uno de los usos más frecuentes que una organización o empresa realiza día a día en un computador. La construcción e implementación de bases de datos permite que la organización pueda almacenar la información relevante del negocio a través de la construcción de una base de datos que se realiza por medio del modelado de datos. En este proceso están identificados tres modelos:

1. Modelo Entidad Relación (MER)
2. Modelo Relacional
3. Modelo Físico

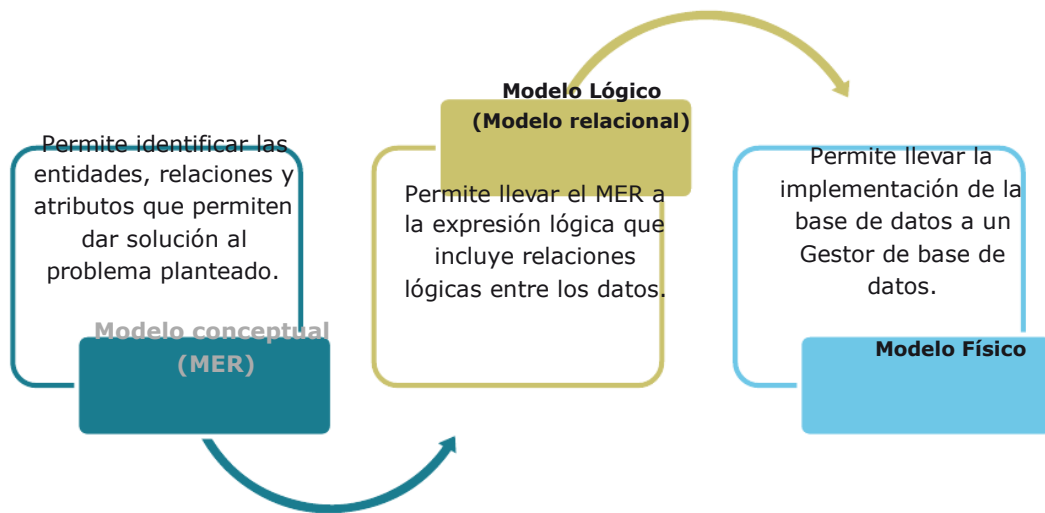
En este material de consulta se presentará el proceso de construcción de un modelo relacional a partir de un MER, lo cual se denomina: **“Transformación del MER a Modelo Relacional”**.

La intención es que el aprendiz logre desarrollar y despertar su interés en el modelado de datos, utilizando dicho modelo para la creación e implementación de una base de datos relacional.

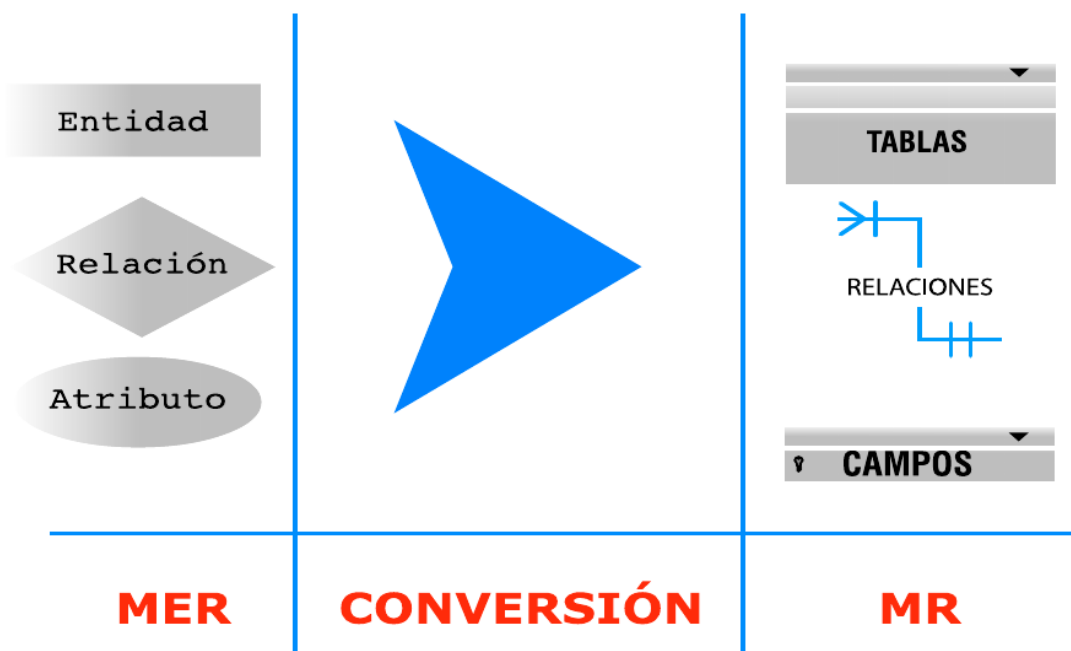
### 2. TRANSFORMACIÓN MER A MODELO RELACIONAL

Uno de los pasos iniciales para la construcción de una base de datos es el modelado de ellos por medio de un Modelo Entidad Relación (MER), modelo que permite por medio de entidades, atributos y relaciones acercarse a una representación preliminar del diseño de una base de datos.

Luego de terminar un MER debe entrarse en el diseño de un modelo lógico, el cual es llamado modelo relacional. La forma como se logra desarrollar una base de datos relacional es a través del modelado y la metodología de diseño de una base de datos puede estar enfocada en los siguientes 3 pasos:

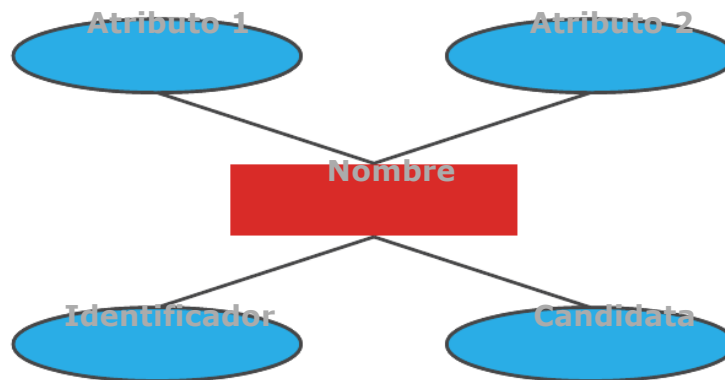


En estos momentos se abordará el proceso de la construcción de un modelo relacional a partir de un MER.



### 3. REGLAS GENERALES DE TRANSFORMACIÓN

La transformación de un Modelo Entidad Relación a Relacional, se puede expresar mediante un esquema relacional o un modelo relacional.



El esquema relacional consta de palabras y relaciones mediante flechas que indican la asociación de las entidades; el modelo relacional es un dibujo que representa las tablas, atributos, llaves primarias y llaves foráneas según la transformación realizada.

A continuación se explica detalladamente la transformación de un MER según las reglas para las entidades y se expresa en esquema relacional y modelo relacional.

NombreTabla(Identificador, Atributo1, Atributo2, Candidata)

#### 3.1 Transformación para las entidades y atributos

- Entidades. Las entidades pasan a ser tablas
- Atributos. Los atributos pasan a ser columnas o atributos de la tabla.
- Identificadores principales. Pasan a ser claves primarias
- Identificadores candidatos. Pasan a ser claves candidatas.

La siguiente imagen ilustra los anteriores elementos:



| EMP_ID | EMP_MOMBRE | EMP_CARGO  | EMP_JEFE | DEP_ID |               |
|--------|------------|------------|----------|--------|---------------|
| 1000   | LOPEZ      | PRESIDENTE |          | 10     | Campo Nulo    |
| 1100   | BENAVIDEZ  | GERENTE    | 1000     | 30     |               |
| 1200   | CORDOBA    | GERENTE    | 1000     | 10     | Filo / Tupla  |
| 1300   | JIMENEZ    | GERENTE    | 1000     | 20     |               |
| 2100   | MARTINEZ   | VENDEDOR   | 1100     | 30     | Celda / Campo |
| 2200   | ARTEAGA    | VENDEDOR   | 1100     | 30     |               |
| 2300   | TOLEDO     | VENDEDOR   | 1100     | 30     |               |
| 2400   | JARAMILLO  | OFICINISTA | 1100     | 30     |               |
| 2500   | ZAMBRANO   | VENDEDOR   | 1100     | 30     |               |

Llave Primaria PK      Columna Atributo      Llave Foranea FK

Por ejemplo:

Si se tiene la siguiente entidad Aprendiz se convierte entonces:

|                           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>MER</b>                |                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| <b>Esquema Relacional</b> | <p>TblAprendiz(AprCedula, AprNombre, AprApellido, AprTelefono, AprEmail)</p> <p>En este esquema la llave primaria se identifica porque se encuentra subrayado el atributo que será llave primaria</p>                                                                                                                                                                                                                                                              |
| <b>Modelo Relacional</b>  | <div> <div>TblAprendiz</div> <div> <div>  AprCedula: INTEGER         </div> <div>  AprNombre: VARCHAR         </div> <div>  AprApellido: VARCHAR         </div> <div>  AprTelefono: INTERGER         </div> <div>  Apremail: VARCHAR         </div> </div> </div> <p>La tabla se llama TblAprendiz y la llave primaria se identifica porque el nombre puede estar: subrayado, tener al frente el estereotipo &lt;&lt;pk&gt;&gt; o una llave en color amarillo.</p> |

**Nota:** para garantizar unicidad de estilos recuerde que debe tener una estandarización para nombrar las tablas y los atributos. Por ejemplo para este caso la entidad aprendiz se convierte en la tabla aprendiz y es llamada como TblAprendiz, el estilo de nombrado es iniciar con las tres primeras letras Tbl para indicar que es tabla, seguido del nombre de la entidad e iniciando con la primera letra en mayúscula.

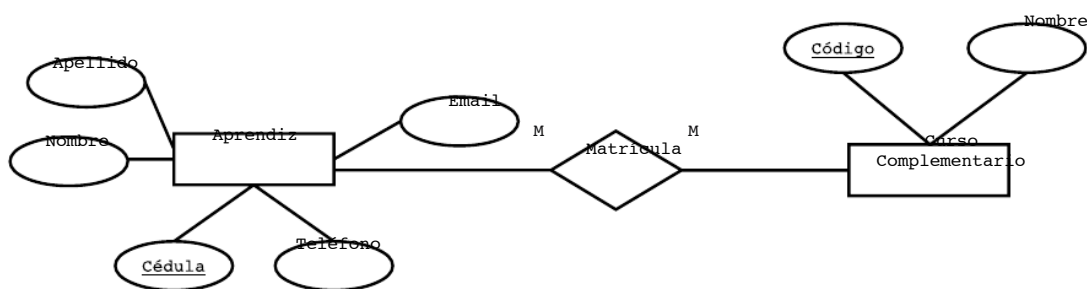
Para el atributo Apellido, en el modelo relacional se convierte en AprApellido indicando Apr que viene de la tabla Aprendiz y seguido del nombre del atributo cada uno iniciando con Mayuscula la primera letra.

### 3.2 Transformación para las Relaciones

#### 3.2.1 Relación Muchos a Muchos (M..M):

Esta relación se distingue en un MER porque su cardinalidad entre las dos entidades esta designada por (M..M) o (N..M).

Ejemplo:

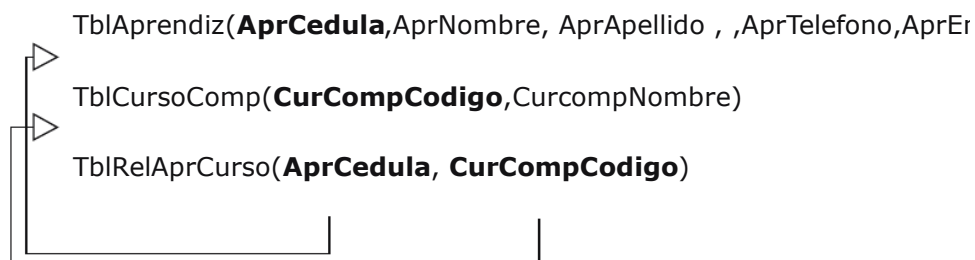


#### Transformación Relación Muchos a Muchos (M..M):

Cuando se tiene una relación de este tipo se debe crear una nueva relación (tabla) la relación se transforma en una tabla cuyos atributos son: los atributos de la relación y las claves de las entidades relacionadas (que pasarán a ser claves externas). La clave de la tabla la forman todas las claves externas.

## Transformación a Esquema Relacional

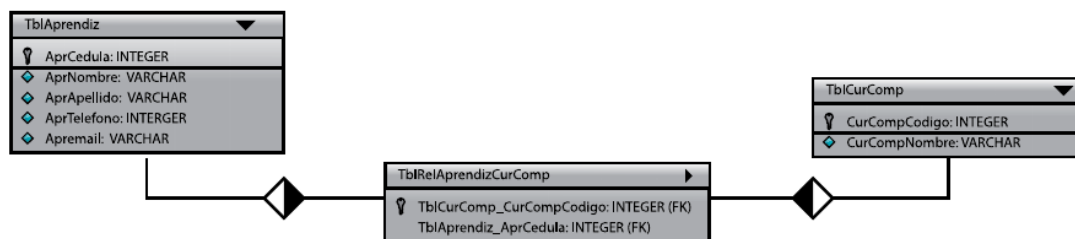
Cuando se hace la transformación a esquema relacional se debe tener un esquema de nombrado para las tablas que se generaran, por ejemplo para este caso la entidad aprendiz la tabla generada se denomina TblAprendiz, anteponiéndole tres letras Tbl que indican que es una tabla; de la misma manera para los atributos de esta entidad se le antepone las tres letras Apr para indicar que son atributos de la tabla Aprendiz. Así se continuara el estilo de nombrado para todos los casos de estudio y ejemplos de este material.



## Transformación a Modelo Relacional

Para realizar las transformaciones a modelo relacional puede utilizar herramientas case, para estos ejemplos se ha utilizado DbDesigner, el uso de ésta última se detallará en los videotutoriales que acompañan el laboratorio correspondiente.

Se visualiza la nueva tabla donde las llaves primarias son a la vez llaves foráneas, nótese que la cardinalidad de la relación está representada por los rombos en blanco (1) o sombreados (M)



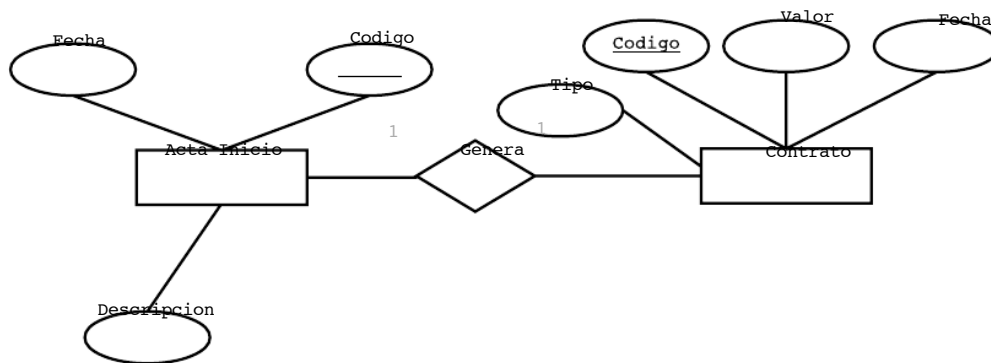
La nueva tabla creada es llamada TblRelAprendizCurComp, que tiene incluida las dos llaves primarias de la tabla TblAprendiz y TblCurComp. Estas llaves en la nueva tabla son llamadas llaves foráneas y son representadas por un <fk> en el modelo.

**Nota:** las llaves primarias pasada a la nueva tabla se llaman llaves foráneas.

### 3.2.2 Relación 1 a 1 (1..1):

Esta relación se distingue en un MER porque su cardinalidad entre las dos entidades esta designada por (1..1).

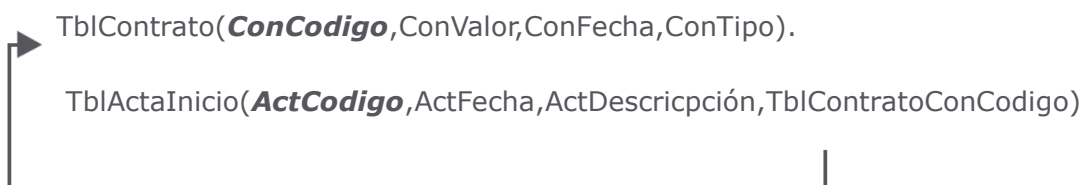
Ejemplo:



### Transformación Relación 1 a 1 (1..1):

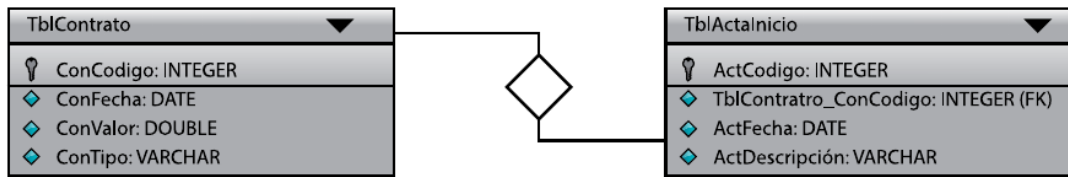
Cuando se tiene una relación de este tipo se envía la llave primaria de una tabla a la otra tabla que participa de la relación, se envía la llave primaria que se considere sea más relevante para enviar a la otra tabla.

### Esquema Relacional



Para este ejemplo específico se ha pasado la llave primaria del contrato (ConCodigo) a la tabla ActaInicio.

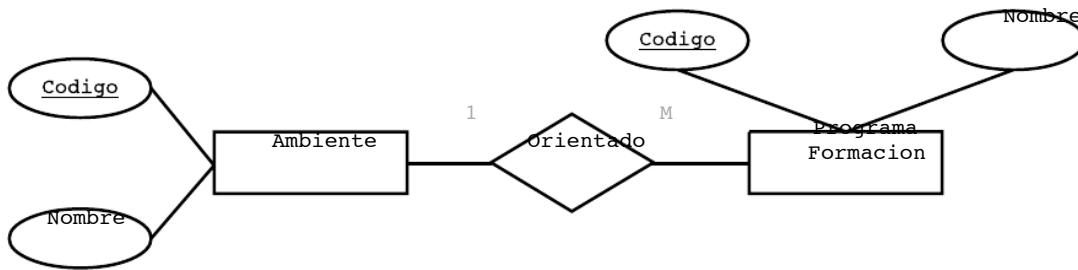
## Transformación a Modelo Relacional



En la tabla TblActaInicio se incluye la clave primaria de la tabla TblContrato (ConCodigo) quedando como llave foránea.

### 3.2.3 Relación 1 a Muchos (1..M):

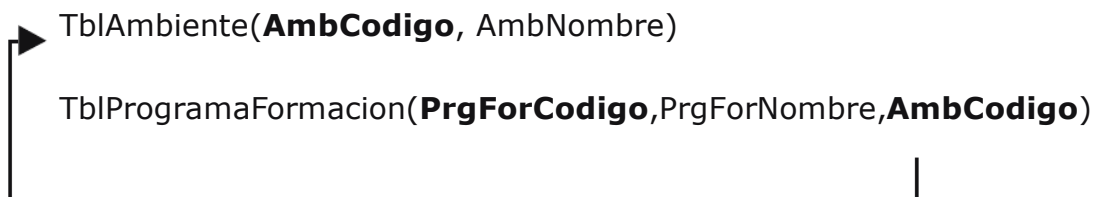
Esta relación se distingue en un MER porque su cardinalidad entre las dos entidades esta designada por (1..M).



### Transformación Relación 1 a Muchos (1..M):

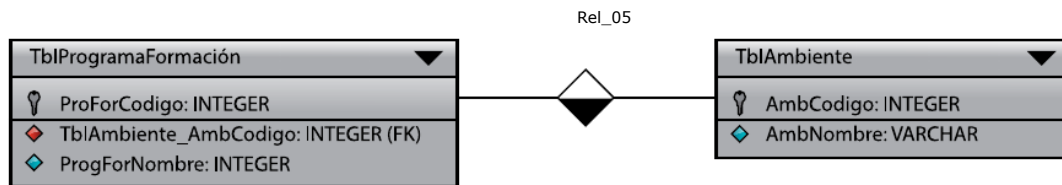
Cuando se tiene una relación de este tipo se debe colocar o enviar la llave primaria de la tabla que tiene en cardinalidad 1 a la tabla que tiene cardinalidad M

#### Esquema relacional



Visualice que La tabla TblAmbiente tiene como cardinalidad 1, por la tanto la clave primaria de esta tabla es pasada a la tabla TblProgramaFormacion.

### Transformación a Modelo Relacional



Note que la clave de la tabla TblAmbiente es pasada a la Tabla TblProgramaFormacion la cual tiene su estereotipo <fk>.

## 4. TRANSFORMACIÓN MER- EXTENDIDO A RELACIONAL

A continuación se presenta un ejemplo de transformación de modelo entidad relación a Relacional, que aplica las reglas descritas. Para este ejemplo se toma una situación problemática y su modelo conceptual, para lo cual se presenta la descripción del problema, el MER y el modelo relacional resultante.

### Descripción del Problema

En el SENA Regional Cauca del Centro de Teleinformática y Producción industrial el coordinador requiere un sistema de información que permita gestionar los horarios de los instructores pertenecientes a un centro de formación que orientan formación profesional integral, tanto de planta como de contrato, tenga en cuenta que un instructor de contrato puede tener su relación de contrato por horas o asignación mensual.

Para lo cual es necesario conocer: Un instructor orienta formación profesional (clase) en un programa de formación (el programa de formación se refiere a las tecnologías o tecnólogos que ofrece el SENA), o en un curso complementario (los cuales son cursos de duración de 2 a 3 meses y solicitados por empresas del sector para capacitar a sus empleados).

Ambos: Programa de formación y complementarios son identificados por un código.

Cuando un aprendiz se matricula a un programa de formación o curso complementario tiene que durante toda su etapa lectiva (clase), cursar competencias laborales.

Una competencia podría asemejarse como una asignatura que un aprendiz debe cursar, pero para este caso siempre manejaremos el término competencia.

**Asignación de Competencias:** Para asignar una competencia a un instructor es necesario que esté legalmente contratado por el centro o que tenga resolución de nombramiento y según su especialidad se le asignan las competencias que va a orientar.

**Ambientes de formación:** Es el sitio específico (salón) donde se orientaran las competencias asignadas. Cada ambiente tiene una codificación asignada.

**Reglas del negocio:** Un instructor de contrato puede iniciar labores una vez tenga el acta de inicio.

Tenga en cuenta que un instructor puede laborar de lunes a viernes de 7:00 a 10:00 p.m. de la noche, sábados y domingos de 8:00 a 6:00 p.m.

Un instructor puede orientar diferentes competencias en diferentes programas de formación y cursos complementarios.

Un instructor no puede orientar una competencia en diferentes ambientes.

Dos o más instructores pueden orientar una o más competencias en un mismo ambiente.

**Usuarios:** Los usuarios que interactúan con la base de datos

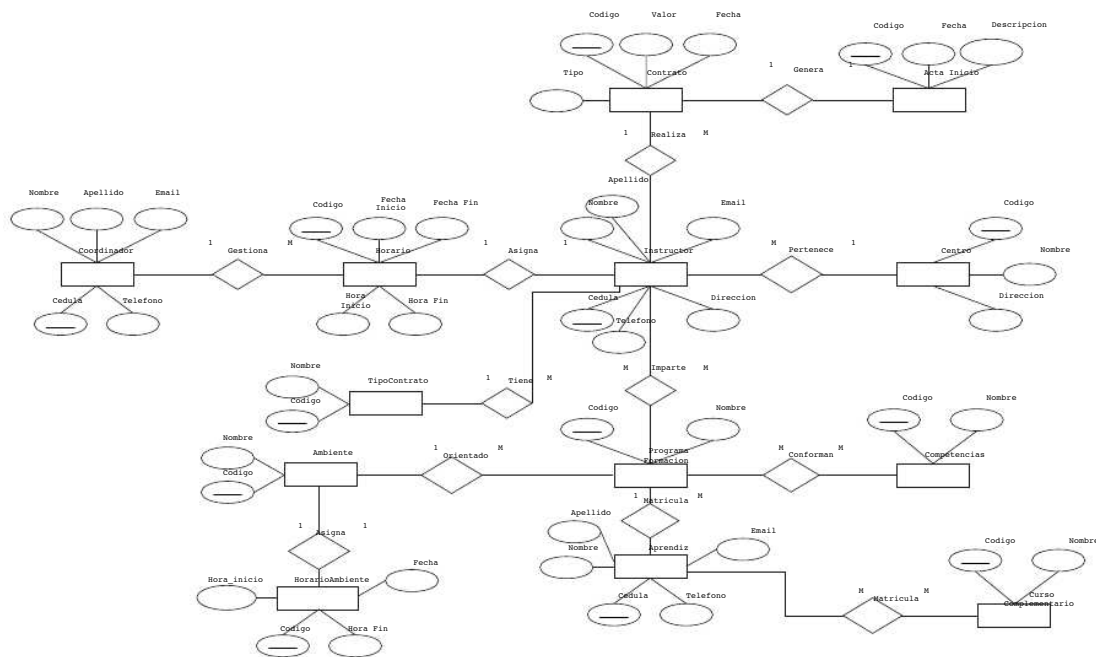
Coordinador académico

Instructores del SENA centro de Teleinformática y PI Regional Cauca

**Informes por Usuario:** Los informes necesarios para mostrar a los usuarios identificados son:

- Visualizar el horario de todos los instructores de planta.
- Visualizar el horario de todos los instructores de contrato
- Visualizar el horario de un solo instructor.
- Visualizar el horario de todos los ambientes
- Visualizar el horario de un solo ambiente.
- Visualizar el horario de todos los programas de formación.

### Modelo Entidad Relación



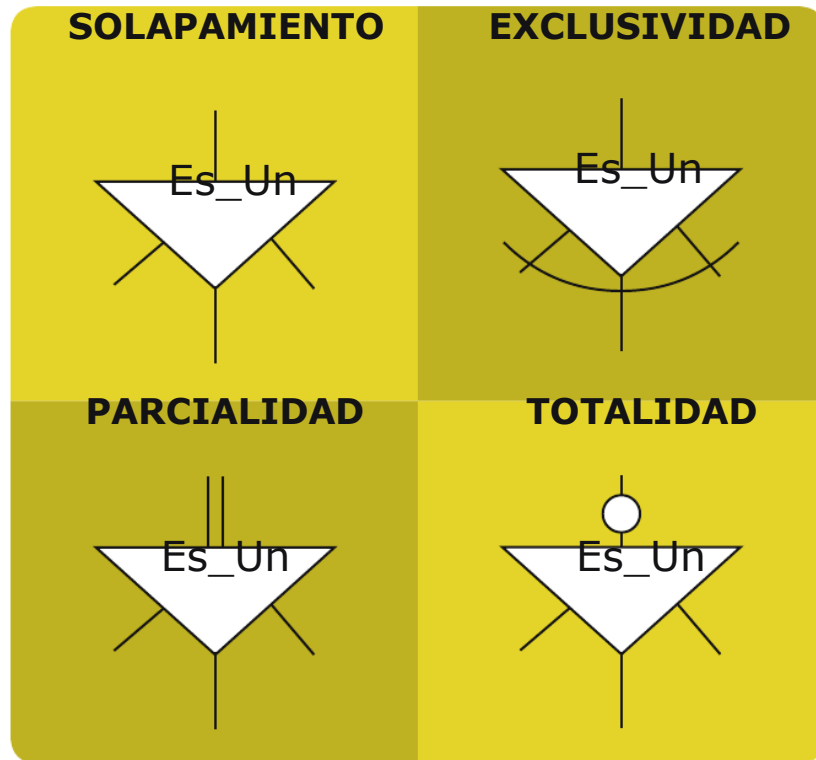


[illegible]

El MER-Extendido ha sido creado para modelar otros tipos de relaciones que no contempla el MER tradicional, los cuales son la generalización y especialización, recursivas, inclusividad y exclusividad. A continuación se muestra como se transforma cada una de estas relaciones a un modelo relacional.

## 5.1 Relación especialización y generalización

Recordemos que en un MER-Extendido la generalización puede identificarse como solapamiento, exclusividad, parcialidad o totalidad Figura 2, en estas relaciones se encuentran entidad superclase o supertipos y entidades subclase o subtipos,

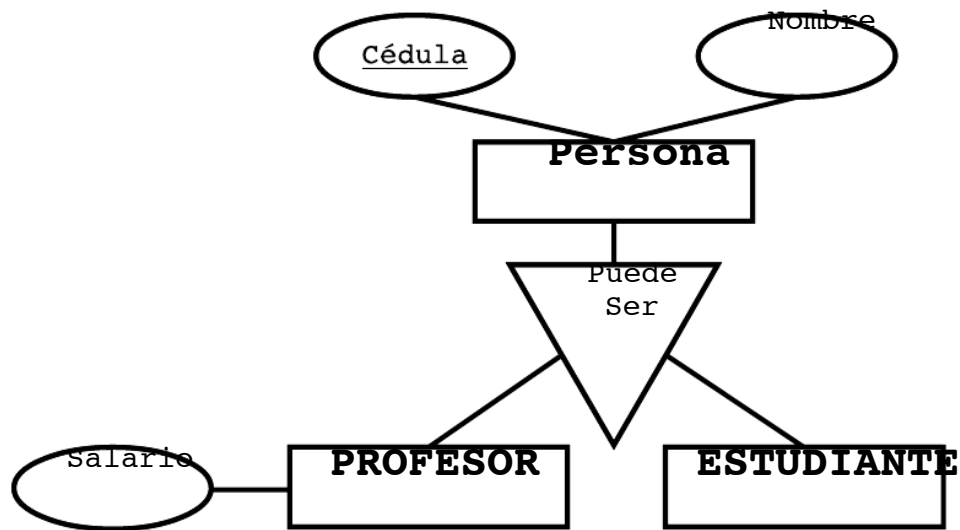


*Tipos de relación Generalización en un Mer Extendido*

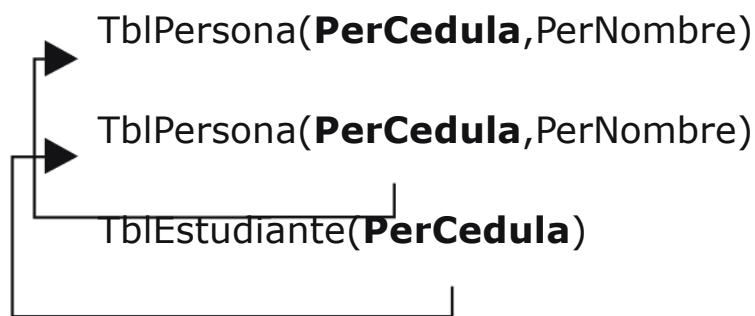
Existen cuatro opciones para la transformación de estos tipos de relaciones

**1. Si es solapada y parcial:** Crear una relación (**tabla**) para la superclase, con sus atributos correspondientes y una relación (**tabla**) para cada subclase con sus atributos incluyendo la clave primaria de la superclase.

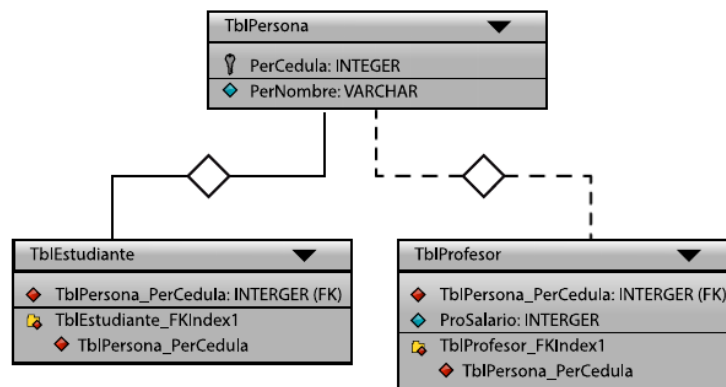
**Generalización:** esta representación es solapada y parcial, es decir una persona puede ser profesor y estudiante a la vez y no es obligatorio que asuma un subtipo.



### Transformación Esquema Relacional



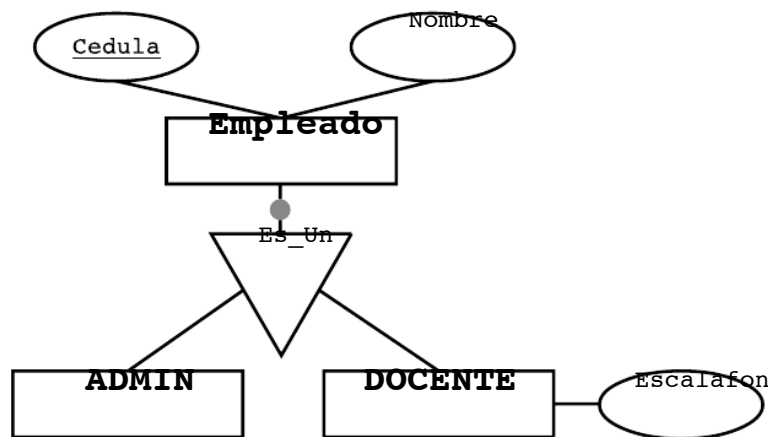
### Transformación Modelo Relacional



Observe que se creó la tabla persona con sus campos y se crearon las tablas profesor y estudiante que heredan la llave primaria de la tabla persona (PerCedula) y se convierte en una llave foránea.

**2. Si es total:** Se crea una relación (tabla) para las entidades subtipo, donde cada una tiene los atributos del supertipo. No se crea una relación para el supertipo.

**Relación de totalidad:** Obligatoriamente el empleado tiene que tomar algún subtipo o administrador y/o docente.



### Transformación Esquema Relacional

TblAdmin(AdminCedula,AdminNombre)

TblDocente(DocCedula,DocNombre)

### Transformación Modelo Relacional

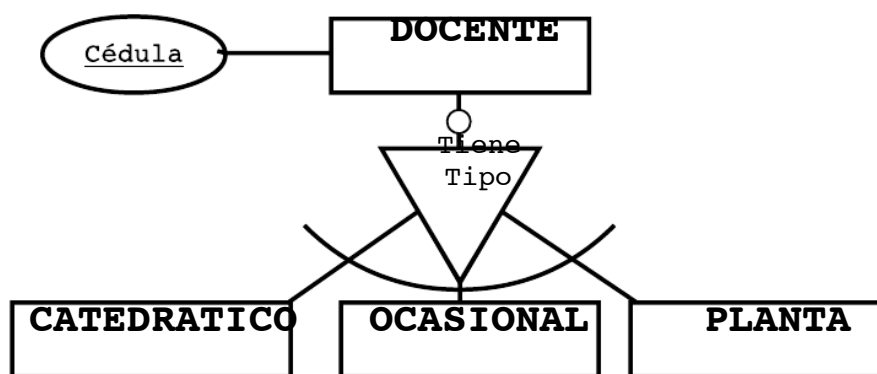
| TblAdmin                                                 |
|----------------------------------------------------------|
| <div> <div></div> <div>AdminCedula: INTEGER</div> </div> |
| <div> <div></div> <div>AdminNombre: VARCHAR</div> </div> |

| TblDocente                                                |
|-----------------------------------------------------------|
| <div> <div></div> <div>AdminCedula: INTEGER</div> </div>  |
| <div> <div></div> <div>AdminNombre: VARCHAR</div> </div>  |
| <div> <div></div> <div>DocEscalafón: INTEGER</div> </div> |

Observe que quedan dos tablas y se suprime la superclase (empleado), cada una de las tablas hereda los atributos de la superclase y continúan con los atributos propios.

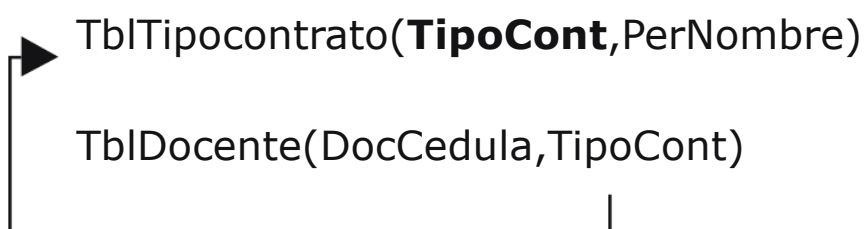
3. Crear una relación (tabla) para almacenar los subtipos y la relación (tabla) superclase. Teniendo en cuenta si es parcial, total, solapado o de exclusividad; Miremos los dos casos que pueden darse en este sentido:

**Caso 1:** Si la descripción del problema indica que obligatoriamente **pertenece a un solo tipo** es decir la relación es (total y exclusiva), entonces se crea una relación para la SuperClase y otra para el tipo y se le pasa la llave primaria de la tabla tipo a la tabla superclase.



En esta representación el docente puede tener un solo contrato (ocacional, catedrático y planta) y obligatoriamente tiene que tener asociado un contrato.

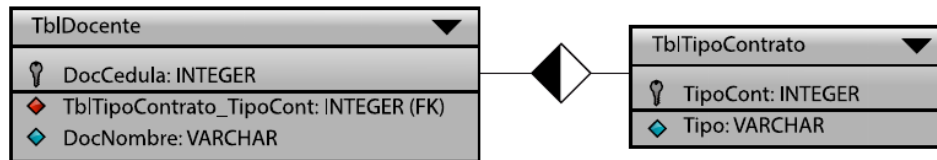
### Transformación esquema relacional



Observe que se crea una relación (tabla) que almacena los diferentes tipos de contrato que un docente puede tener y adicionalmente a la tabla TblDocente se le pasa la llave primaria de la tabla TipoContrato.

De esta manera se garantiza que cada docente obligatoriamente tenga un solo tipo de contrato.

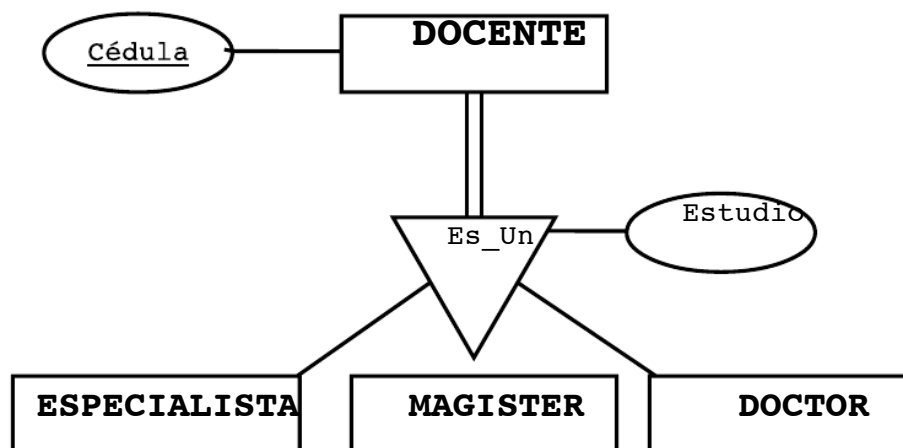
### Transformación modelo relacional



Observe que se genera una nueva tabla llamada TblTipCont que almacenara los diferentes tipos de contrato que puede tener un docente, para mayor explicación esta tabla se almacenara con los siguientes datos.

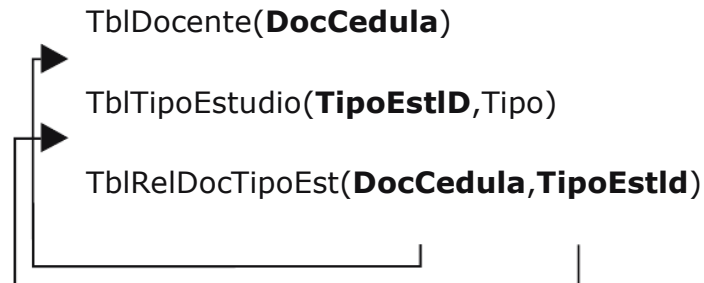
| TblTipCont |             |
|------------|-------------|
| TipoContID | Tipo        |
| 1          | Catedratico |
| 2          | Ocasional   |
| 3          | Planta      |

**Caso 2:** Si la descripción del problema indica que puede **pertenecer a varios tipos** es decir la relación es (parcial y solapada), entonces se crea una relación para la SuperClase, otra para el tipo y se trata la relación superclase y Tipo como una relación de muchos a muchos.

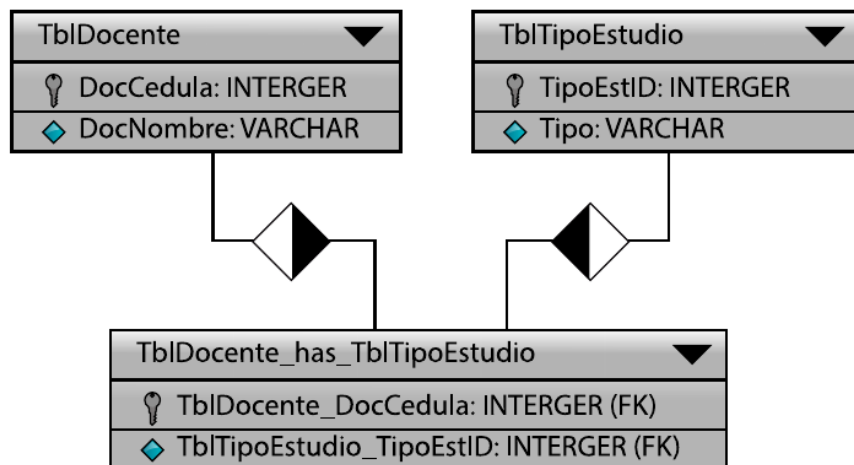


En esta representación un docente puede tener o no un tipo de estudio y puede tenerlos todos a la vez.

### Transformación Esquema Relacional



Observe que se crea una relacion (tabla) para la superclase y otra para el tipo y se tratan estas dos relaciones como muchos a muchos, por lo tanto se crea una tabla intermedia que contenga las llaves primarias de las 2 tablas creadas.



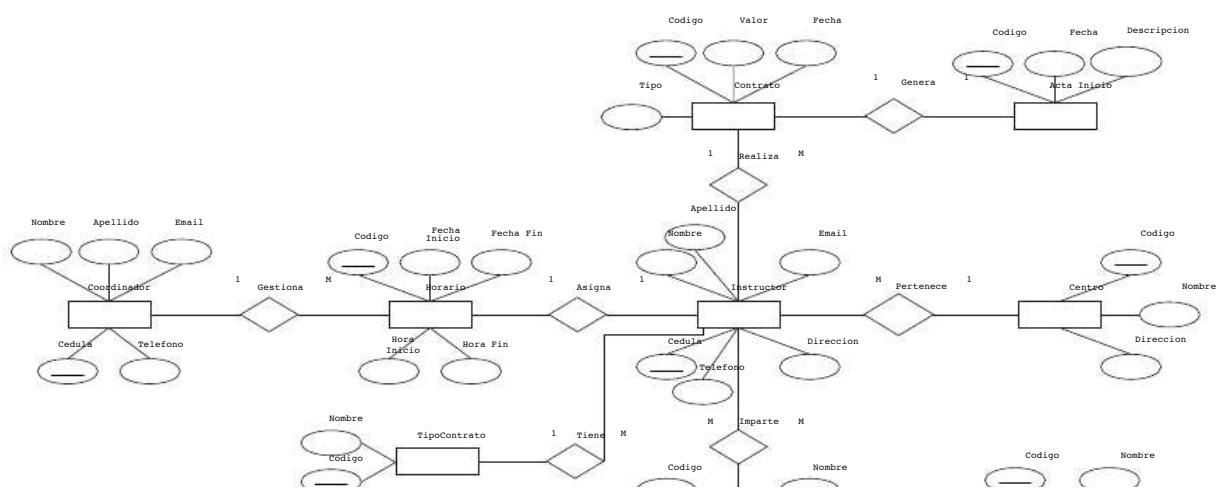
## 5.2 Relación Recursiva

Las relaciones recursivas se tratan de la misma forma que las otras, sólo que hay que imaginar que la tabla se divide en dos, una por cada rol. Teniendo en cuenta eso, la solución es idéntica a lo ya resuelto en los casos anteriores.

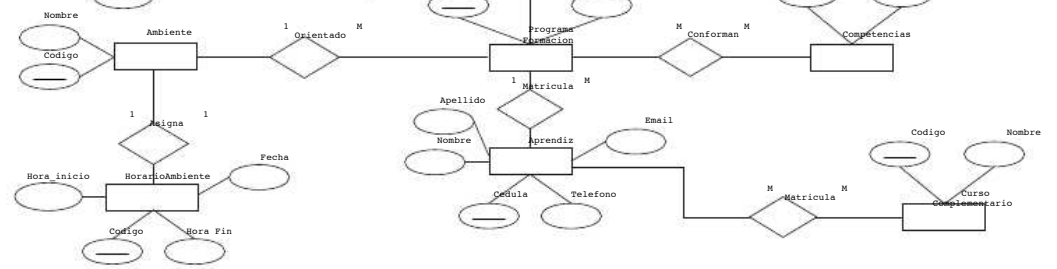
## Conversión Mer a Modelo Relacional

- Visualizar el horario de todos los instructores de planta.
- Visualizar el horario de todos los instructores de contrato
- Visualizar el horario de un solo instructor.
- Visualizar el horario de todos los ambientes
- Visualizar el horario de un solo ambiente.
- Visualizar el horario de todos los programas de formación.

## Modelo Entidad Relación

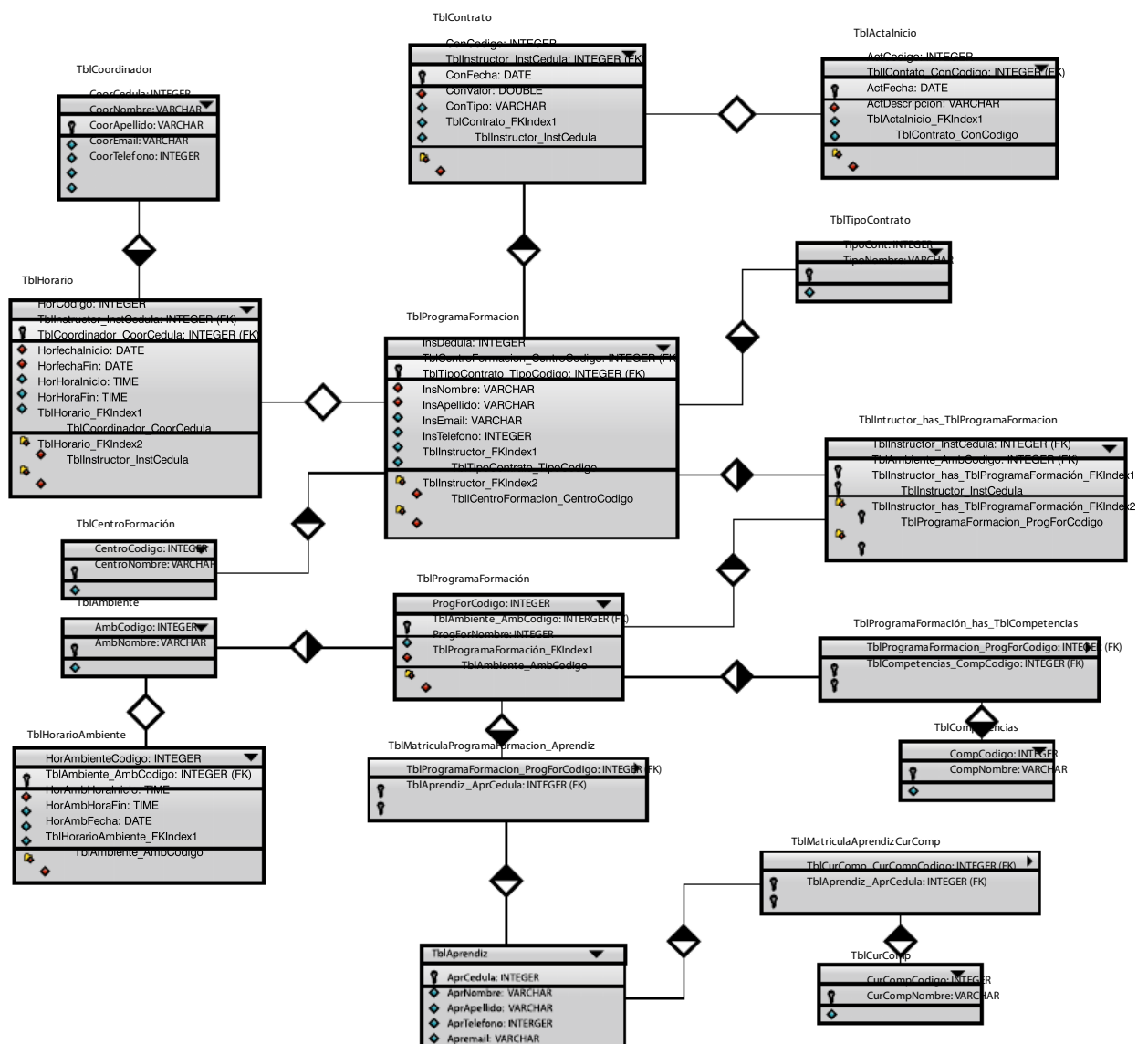






## Conversión Mer a Modelo Relacional

### Modelo Relacional



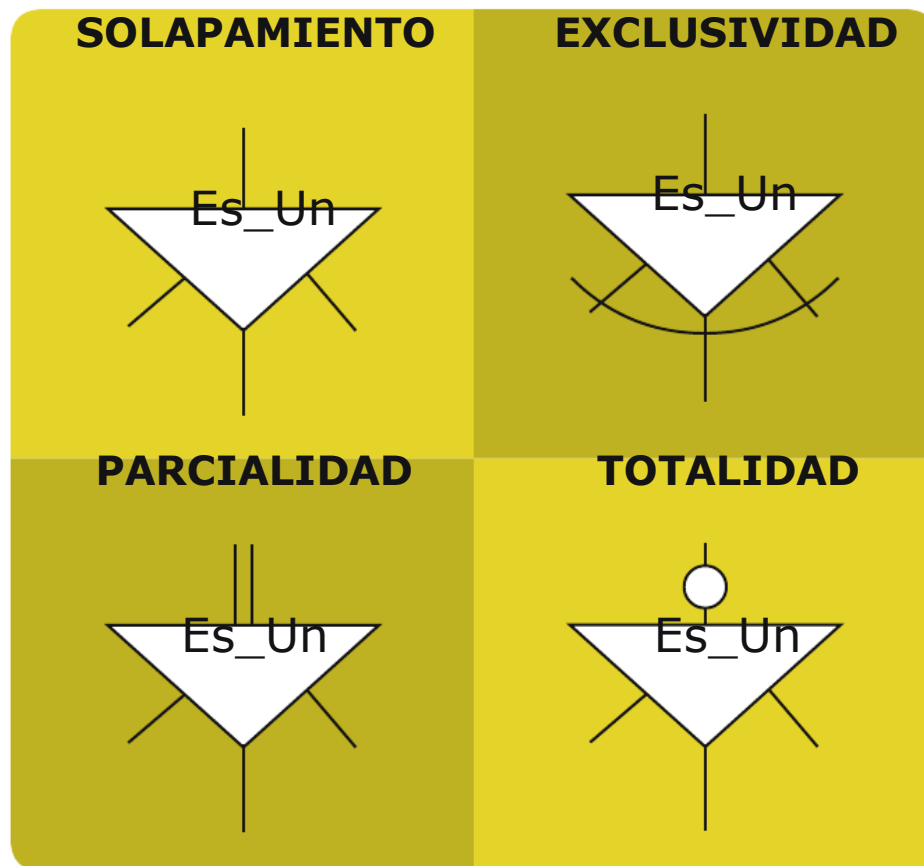
## 5. CONSTRUCCIÓN TRANSFORMACIÓN MER-EXTENDIDO A RELACIONAL

El MER-Extendido ha sido creado para modelar otros tipos de relaciones que no contempla el MER tradicional, los cuales son la generalización y especialización, recursivas, inclusividad y excusividad. A continuación se muestra como se transforma cada una de estas relaciones a un modelo relacional.



## 5.1 Relación especialización y generalización

Recordemos que en un MER-Extendido la generalización puede identificarse como solapamiento, exclusividad, parcialidad o totalidad Figura 2, en estas relaciones se encuentran entidad superclase o supertipos y entidades subclase o subtipos,



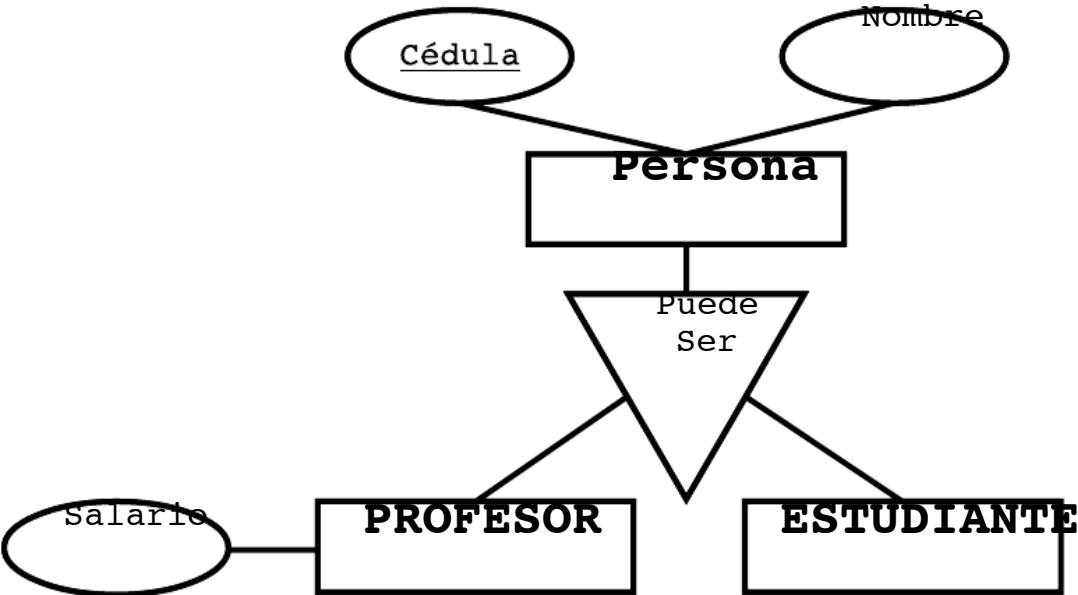
*Tipos de relación Generalización en un Mer Extendido*

Existen cuatro opciones para la transformación de estos tipos de relaciones

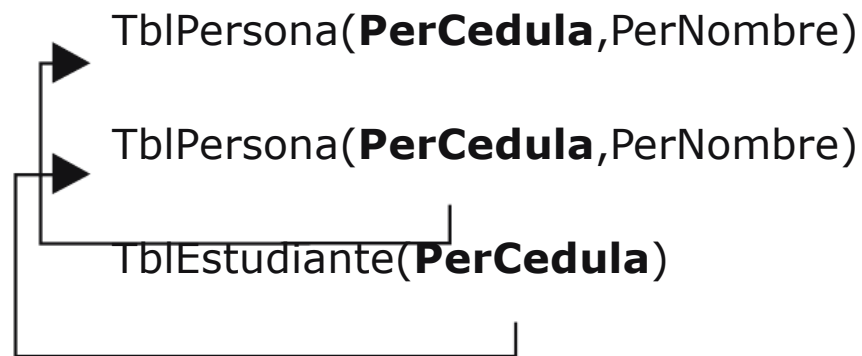
**1. Si es solapada y parcial:** Crear una relación (**tabla**) para la superclase, con sus atributos correspondientes y una relación (**tabla**) para cada subclase con sus atributos incluyendo la clave primaria de la superclase.

**Generalización:** esta representación es solapada y parcial, es decir una persona puede ser profesor y estudiante a la vez y no es obligatorio que asuma un subtipo.

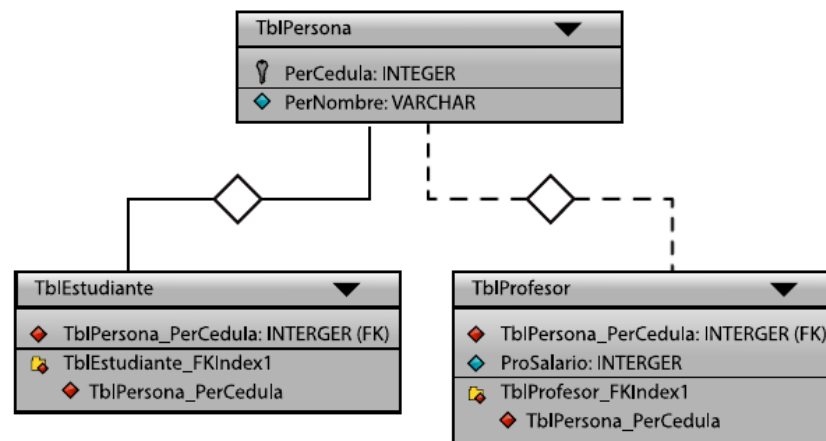
Conversión Mer a Modelo Relacional



## Transformación Esquema Relacional



## Transformación Modelo Relacional



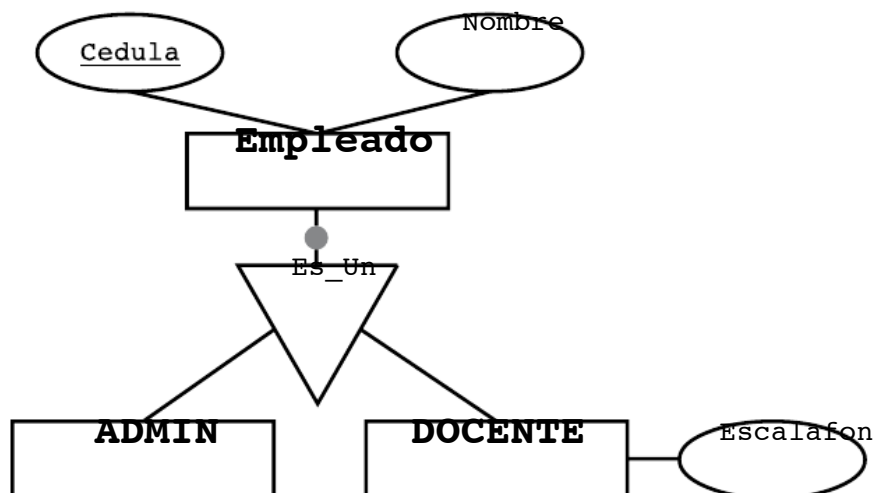
## Conversión Mer a Modelo Relacional

---

Observe que se creó la tabla persona con sus campos y se crearon las tablas profesor y estudiante que heredan la llave primaria de la tabla persona (PerCedula) y se convierte en una llave foránea.

**2. Si es total:** Se crea una relación (tabla) para las entidades subtipo, donde cada una tiene los atributos del supertipo. No se crea una relación para el supertipo.

**Relación de totalidad:** Obligatoriamente el empleado tiene que tomar algún subtipo o administrador y/o docente.



## Transformación Esquema Relacional

TblAdmin(AdminCedula,AdminNombre)

TblDocente(DocCedula,DocNombre)

## Transformación Modelo Relacional

| TblAdmin ▼ |                      |
|------------|----------------------|
| 🔑          | AdminCedula: INTEGER |
| ◆          | AdminNombre: VARCHAR |

| TblDocente ▼ |                       |
|--------------|-----------------------|
| 🔑            | AdminCedula: INTEGER  |
| ◆            | AdminNombre: VARCHAR  |
| ◆            | DocEscalafón: INTEGER |

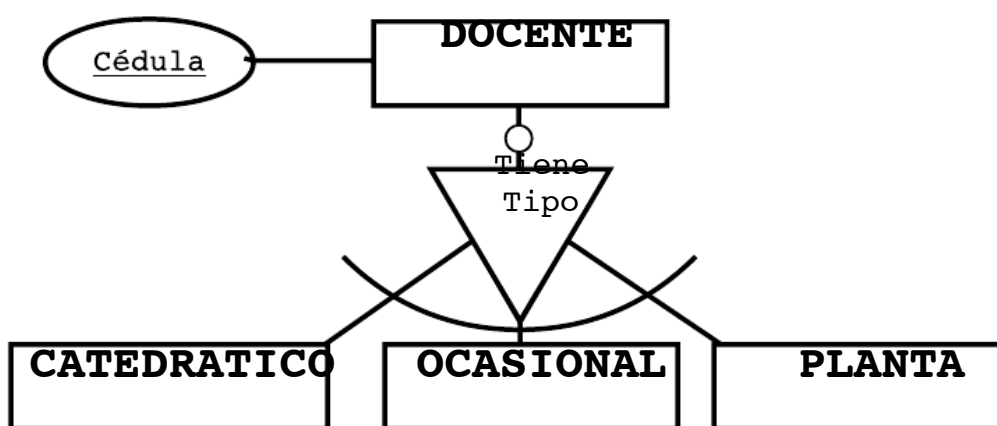




Observe que quedan dos tablas y se suprime la superclase (empleado), cada una de las tablas hereda los atributos de la superclase y continúan con los atributos propios.

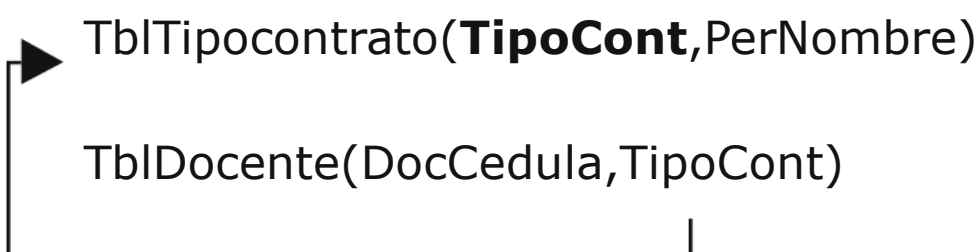
3. Crear una relación (tabla) para almacenar los subtipos y la relación (tabla) superclase. Teniendo en cuenta si es parcial, total, solapado o de exclusividad; Miremos los dos casos que pueden darse en este sentido:

**Caso 1:** Si la descripción del problema indica que obligatoriamente **pertenece a un solo tipo** es decir la relación es (total y exclusiva), entonces se crea una relación para la SuperClase y otra para el tipo y se le pasa la llave primaria de la tabla tipo a la tabla superclase.



En esta representación el docente puede tener un solo contrato (ocacional, catedrático y planta) y obligatoriamente tiene que tener asociado un contrato.

### Transformación esquema relacional



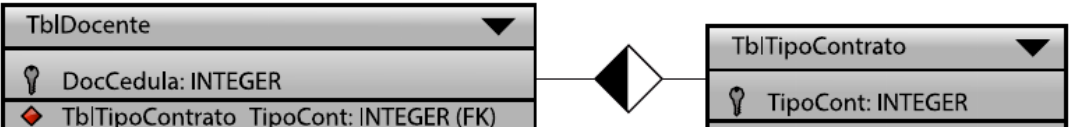
Observe que se crea una relación (tabla) que almacena los diferentes tipos de contrato que un docente puede tener y adicionalmente a la tabla TblDocente se le pasa la llave primaria de la tabla TipoContrato.

### Conversión Mer a Modelo Relacional

---

De esta manera se garantiza que cada docente obligatoriamente tenga un solo tipo de contrato.

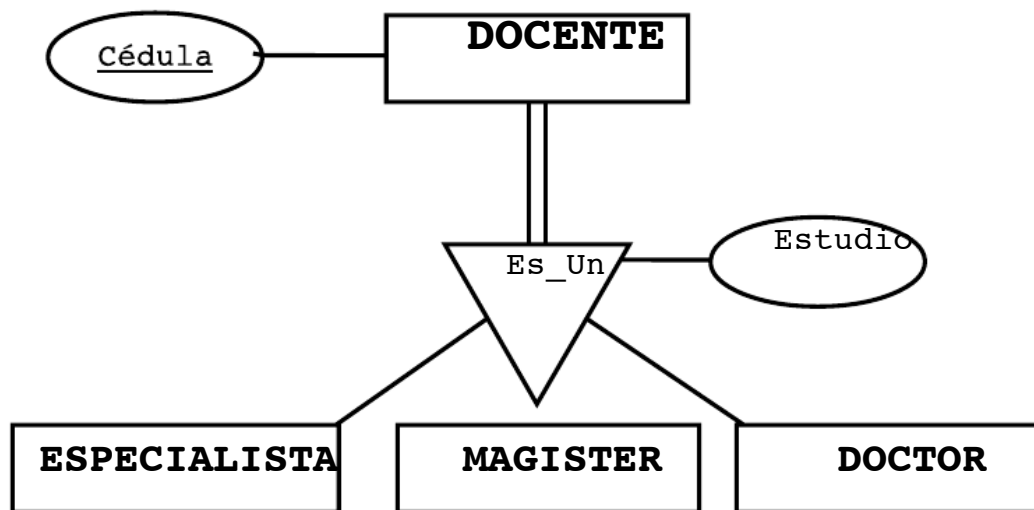
### Transformación modelo relacional



Observe que se genera una nueva tabla llamada TblTipCont que almacenara los diferentes tipos de contrato que puede tener un docente, para mayor explicación esta tabla se almacenara con los siguientes datos.

| TblTipCont |             |
|------------|-------------|
| TipoContID | Tipo        |
| 1          | Catedratico |
| 2          | Ocasional   |
| 3          | Planta      |

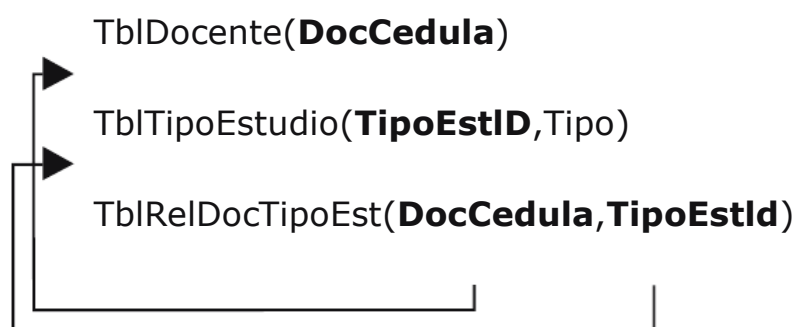
**Caso 2:** Si la descripción del problema indica que puede **pertenecer a varios tipos** es decir la relación es (parcial y solapada), entonces se crea una relación para la SuperClase, otra para el tipo y se trata la relación superclase y Tipo como una relación de muchos a muchos.



## Conversión Mer a Modelo Relacional

En esta representación un docente puede tener o no un tipo de estudio y puede tenerlos todos a la vez.

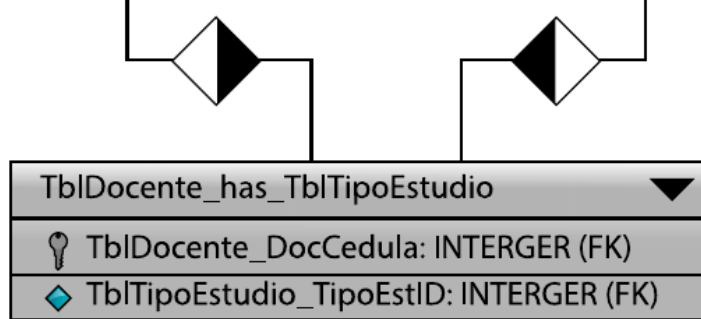
### Transformación Esquema Relacional



Observe que se crea una relacion (tabla) para la superclase y otra para el tipo y se tratan estas dos relaciones como muchos a muchos, por lo tanto se crea una tabla intermedia que contenga las llaves primarias de las 2 tablas creadas.

| TblDocente          |
|---------------------|
| DocCedula: INTERGER |
| DocNombre: VARCHAR  |

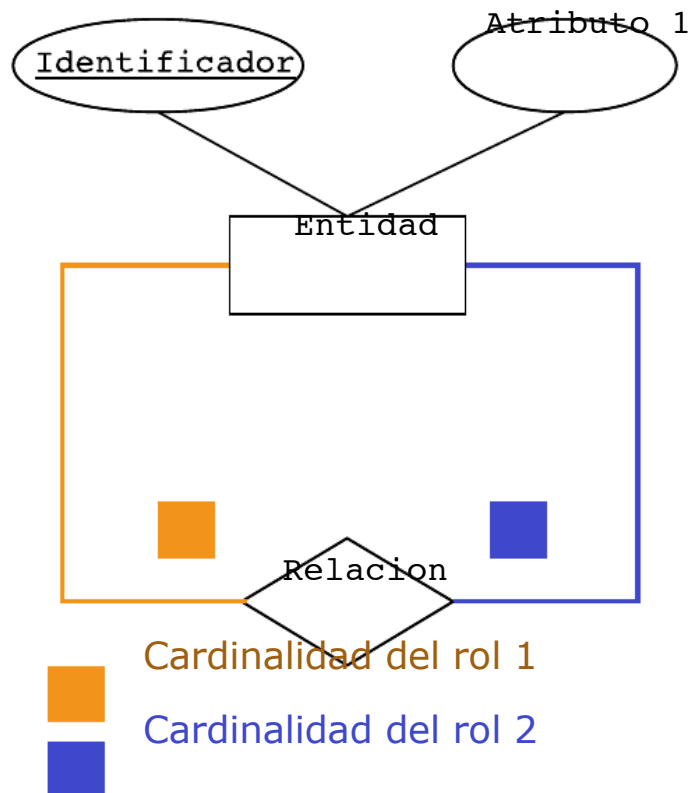
| TblTipoEstudio      |
|---------------------|
| TipoEstID: INTERGER |
| Tipo: VARCHAR       |



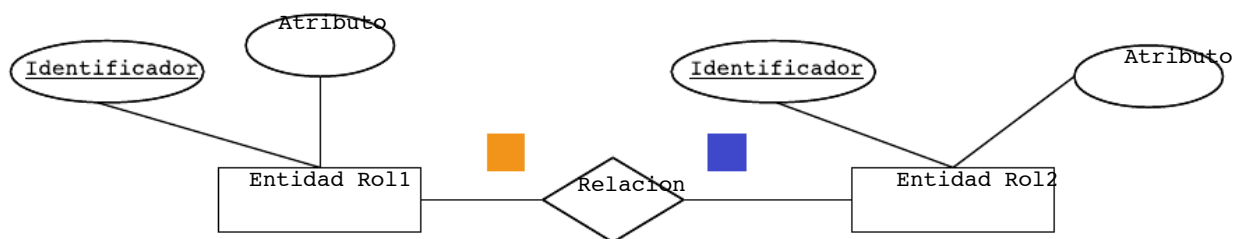
## 5.2 Relación Recursiva

Las relaciones recursivas se tratan de la misma forma que las otras, sólo que hay que imaginar que la tabla se divide en dos, una por cada rol. Teniendo en cuenta eso, la solución es idéntica a lo ya resuelto en los casos anteriores.

## Conversión Mer a Modelo Relacional



**La forma de imaginarse las relaciones recursivas para resolverlas es:**

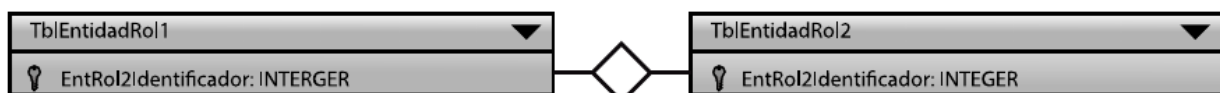


Donde se "duplica la entidad" y se conservan las cardinalidades, luego dependiendo de ésta, se aplica la regla de transformación correspondiente teniendo las siguientes posibilidades:

1 caso: si la cardinalidad de la relación es de 1 a 1:

## Conversión Mer a Modelo Relacional

---

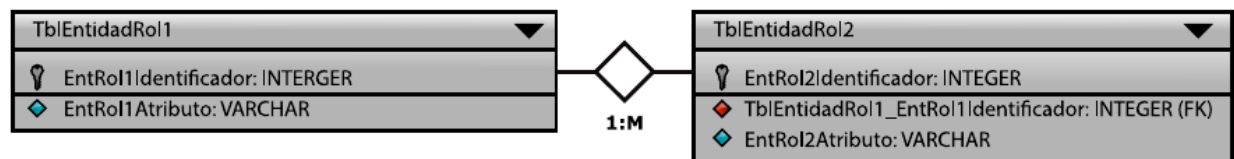


◆ EntRol1Atributo: VARCHAR

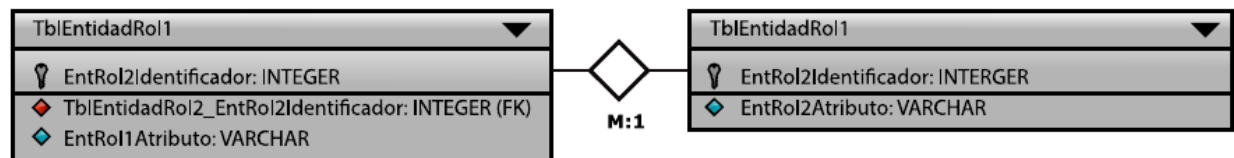
◆ TblEntidadRol1\_EntRol1Identificador: INTEGER (FK)  
◆ EntRol2Atributo: VARCHAR

La entidad con Rol débil hereda el atributo de identificación de la otra entidad y pasa a ser su llave foránea

2 caso: si la cardinalidad de la relación es de 1 a M :

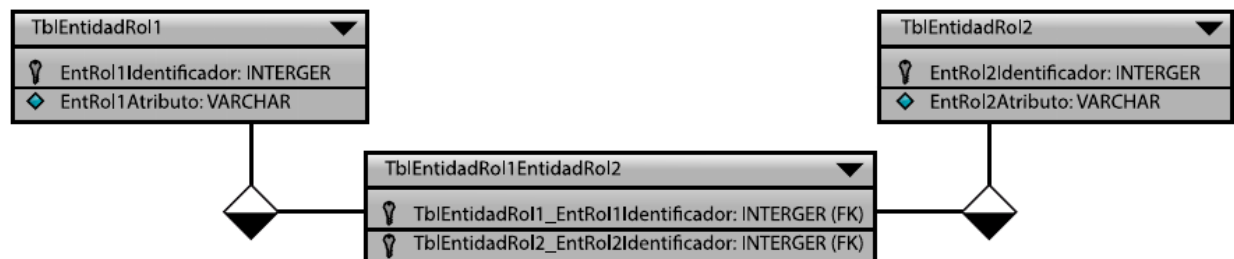


O de M a 1:



En cualquiera de los casos, la entidad con relación muchos hereda el atributo de identificación de la otra y esta se convierte en la llave foránea

3 caso: si la cardinalidad de la relación es de M a M:



Se genera en este caso una tercera tabla que relaciona las dos entidades, está compuesta por los atributos de identificación que hereda de cada entidad, más los atributos propios de la relación (si existieran).