Fundamentos de Bases de Datos

Por
M. en C. Gilberto Pacheco Gallegos

Universidad Tecnológica de Nezahualcóyotl
División de Informática y Computación
Academia de Ingeniería de Software
Academia de Programación

Contenido

1. Introducción a los Sistemas de Información.

1. Introducción a los Sistemas de Información.

Estos sistemas satisfacen las necesidades de información de la organización en la cual están inmersos. Pueden manejar distintos tipos de datos:

- No formateados (sistemas de recuperación de información).
- Formateados (sistemas de gestión de base de datos).

En una empresa hay diferentes niveles de administración. Básicamente tenemos:
- Estratégicos (plantación).
- Táctico (control de gestión).
- Operacional (tareas administrativas).

Los niveles de más jerarquía necesitan información que refleje el comportamiento global de la empresa para tomar decisiones sobre el rumbo del negocio y elaborar planes, así como la forma de llevarlos acabo. Los niveles más bajos necesitan llevar el control detallado de las operaciones. y por ello utilizan información más específica. Los sistemas de información deben cubrir estos dos aspectos como se refleja en la siguiente figura:

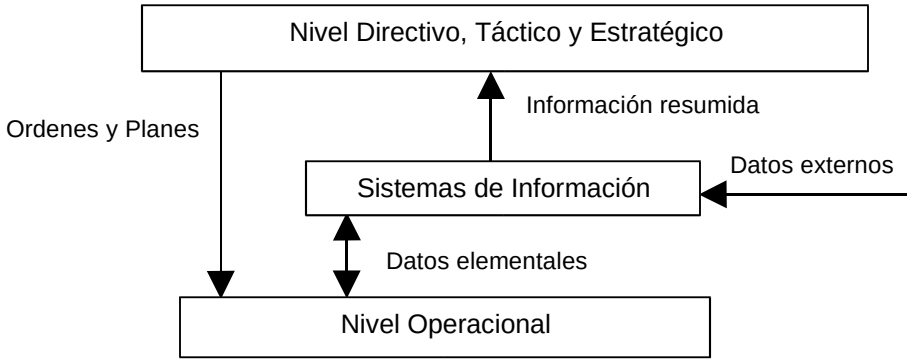


Figura 1.1 – Los sistemas de información en una empresa.

Para responder a estas necesidades tradicionalmente se utilizaban programas. Cada uno de los cuales generaban archivos. Desafortunadamente este enfoque lleva a que el formato de los archivos sea diferente para cada programa y en ocasiones se tenga información repetida. Tratando de corregir este problemas, la información se almacena en una base de datos única para la empresa.

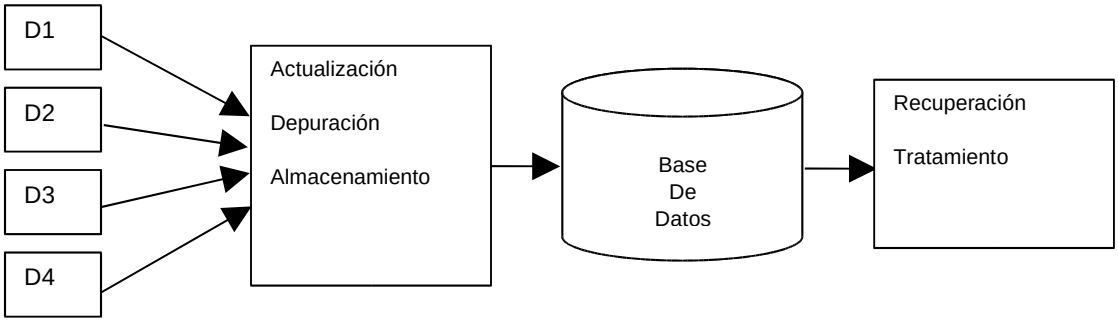


Figura 1.2 – Estructura actual de los sistemas de información.

Esta arquitectura tiene las siguientes **ventajas**:

- Independencia de los datos respecto a los procesos.
- Coherencia de resultados.
- Mayor disponibilidad de los datos para los usuarios.
- Mayor valor informativo (interrelaciones).
- Documentación de la información: normalizada e integrada a los datos.
- Mayor eficacia en la introducción: los datos se validan al introducirse al sistema.
- Reducción del espacio de almacenamiento.
- Seguridad.

Desventajas

- Implantación costosa.
- Personal especializado.
- Implantación larga y difícil.
- Falta de rentabilidad a corto plazo.
- Ausencia real de normas.
- Desfase entre teoría y práctica.

En estos sistemas se tienen tres formas de percibir al sistema (niveles de abstracción).

- **Estructura lógica de usuario:** Forma en que los usuarios perciben la estructura del sistema.
- **Estructura lógica global:** Forma en que perciben al sistema los desarrolladores y administradores.
- **Estructura física:** La manera en que realmente esta implementado.

Para poder construir los sistemas se utilizan dos tipos de lenguajes:

- **Lenguaje de Definición de Datos (DDL).** Para definir la estructura y características importantes de los datos.
- **Lenguaje de Manipulación de Datos(DML).** Para procesar y almacenar información.

2. Modelo Conceptual.

2. Modelo Conceptual.

Generalmente hay una gran diferencia entre la manera en que las personas conciben la información y su representación en un sistema de cómputo. Para lograr que nuestros modelos se acerquen más a la realidad se utilizan los modelos conceptuales como pasos intermedios.

Un **modelo conceptual** representa la información de una forma cercana a como es manejada en la vida real, sin importar como se va a implementar.

Cuando estudiamos la problemática a representar con un sistema de cómputo, conviene más utilizar un modelo conceptual, porque lo importante es entender lo que tenemos enfrente. Una vez que conocemos el problema, entonces elegiremos la herramienta más conveniente para solucionarlo. Afortunadamente, hay una serie de pasos muy sencillos para obtener implementaciones a partir de diagramas como los que se estudian en este capítulo.

Debido a que actualmente los objetos dominan la escena de la ingeniería de software, utilizaremos un modelo conceptual orientado a objetos y utilizaremos UML para hacer su representación en diagramas.

2.1. Conceptos Básicos.

2.1.1. Objetos.

Nuestro mundo está lleno de objetos con los cuales interactuamos; pero cuando pensamos o hablamos acerca de ellos no es necesario que estén presentes. Lo que hacemos es formarnos una idea de las cosas y con esta trabajamos. Al proceso de formar las ideas mentales se le llama **abstracción**.

Así pues cuando usamos la palabra objeto, nos referimos a la abstracción (o representación mental) de algo "real".

La forma de identificar los objetos, varía según lo que estemos realizando. Por ejemplo, observa la figura y di qué objetos ves:

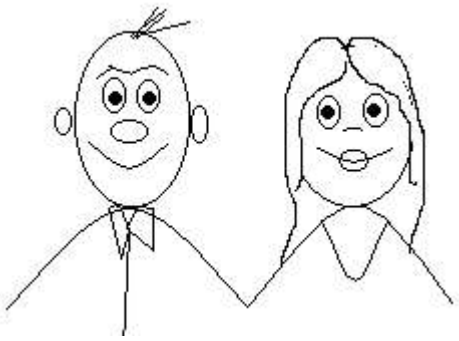


Figura 2.1 – Una imagen de objetos.

El diseñador de un juego vería una serie de personajes a animar. Un psicólogo probablemente vería una familia y sus lazos de integración. Para el registro de población son individuos registrados. En un cómic esto es una fotografía. La Secretaría de Hacienda distinguiría algunos de sus contribuyentes.

Ejercicio 2.1.

Ve a tu alrededor e identifica 4 objetos.

2.1.2. Atributos y Dominios.

Cuando trabajamos mentalmente con un objeto solamente usamos las características que nos interesan, como pueden ser el color, el nombre, etcétera. Observa que todas estas tienen asociados valores que pueden ser diferentes para cada objeto que señalemos.

Un **atributo** es la abstracción de una característica directa de un objeto completo, a la cual se le puede asociar un solo valor.

El conjunto de valores que puede tomar un atributo, recibe el nombre de **dominio**.

Una vez más, los atributos que distinguimos dependen del contexto en el que estemos trabajando. Observa la figura 2.2. Para un contador, lo que tenemos en ella podría ser una factura con dos atributos:

- *Número de factura* con valor "001" e
- *Importe* con valor "$123.00."

Para un impresor, lo que tenemos es una hoja impresa con los siguientes atributos:

- *Tamaño* con valor "carta"
- *Texto impreso* con valor "Factura: 001. Importe: $123.00"

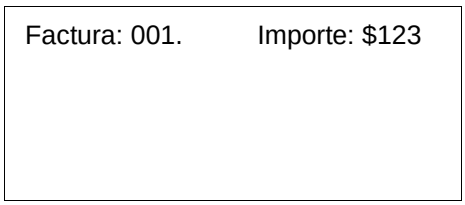


Figura 2.2 – Una factura.

Los valores posibles para el "Número de factura" (O sea el dominio del atributo "Número de factura") son números positivos.

El dominio para *Importe* es "$ + número positivo con dos decimales".

El dominio de *Tamaño* es "{carta, oficio, a4}".

El dominio de *Texto impreso* es “texto alfanumérico”.

Ejercicio 2.2.

Asigna atributos, valores y dominios para los objetos que identificaste en el ejercicio 2.1.

2.1.3. Clases.

Observa que hay objetos con los mismos atributos y que podríamos considerar del mismo tipo, que de hecho podemos agrupar.

Una clase es un conjunto de objetos del mismo tipo, con los mismos atributos, con un comportamiento similar y sobre los cuales se aplican las mismas reglas.

La información sobre una clase, sus atributos y dominios generalmente se denota de la siguiente manera:

Figura 2.3 – Representación genérica y una clase concreta.

Un objeto y sus valores asociados se pueden representar así:

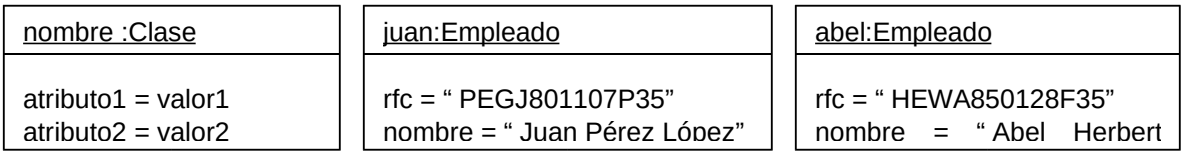


Figura 2.4 – Representación genérica y dos objetos concretos

Cabe resaltar que una clase representa a 0, 1 o más objetos; así que cuando pienses en la cajita de la clase, imagina que representa a muchos objetos.

2.1.4. Ligas y Asociaciones.

Los objetos no se utilizan aislados, generalmente están relacionados con otros.

Una **liga** es la abstracción de una relación directa entre dos objetos. En la figura 2.5 podemos decir que el objeto *andrés* está casado con el objeto *jackelín*. También indica que *andrés* es amigo de *jorge* y r*amón*.

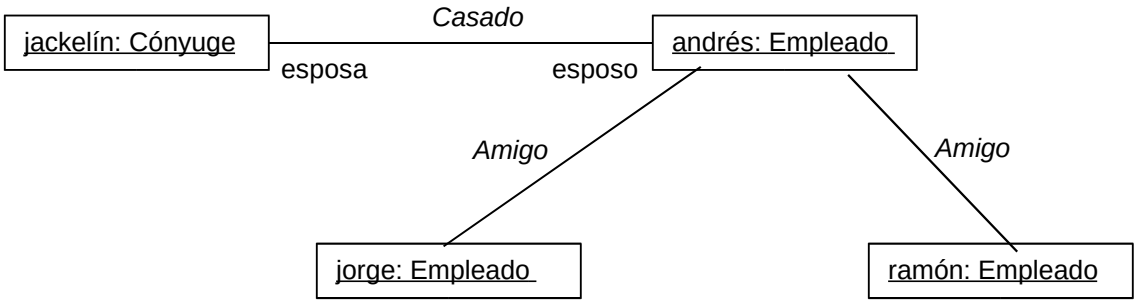


Figura 2.5 – Objetos ligados

Una **asociación** representa el tipo y cantidad de ligas que se pueden establecer entre objetos de dos o más clases. El diagrama de la figura 2.6 corresponde a la situación representada en la figura 2.5.

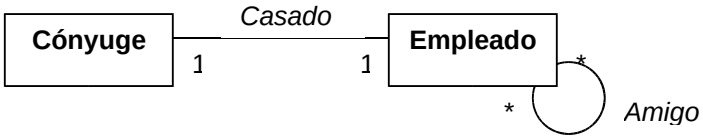


Figura 2.6 – Diagrama de clases correspondiente a la figura 2.5

En la sección 2.4 se profundiza sobre las características de una relación.

2.2. Identificación de Clases.

Tal vez te preguntes ¿Cómo puedo identificar clases? Muchas de ellas caen en alguna de las siguientes categorías:

1. Cosas tangibles.
2. Roles.
3. Incidentes.
4. Interacciones.
5. Especificaciones.

Nota que estas son sólo sugerencias o puntos de partida.

2.2.1. Cosas Tangibles.

Las cosas que ves directamente son lo más fácil de identificar, como:

- avión
- edificio

- computadora
- vehículo
- semáforo

2.2.2. Tipos de Actividades Desempeñados por Personas u Organizaciones.

- doctor
- paciente
- jardinero
- gerente
- supervisor
- propietario
- cliente
- proveedor

Cuando distingues que una persona tiene una actividad, puedes esperar que habrá más tipos de actividades que pueden ser desempeñados por la misma persona o por diferentes. Generalmente están relacionados con la organización que estás analizando.

2.2.3. Incidentes.

Nos sirven para describir cosas que suceden.

- vuelo
- falla (orden de servicio)
- transmisión
- programa
- función

2.2.4. Interacciones.

Corresponden a cosas que se realizan entre varias personas

- transacciones
- matrimonio
- compra
- contrato

También pueden describir distribuciones topográficas

- red de computadoras
- red de suministro
- vía
- espacio aéreo

2.2.5. Especificaciones y Papeles.

Los papeles generalmente hacen referencia a objetos que en ocasiones son intangibles

- Entradas de almacén
- Productos (Generalmente un inventario es el registro de un producto)
- Facturas
- Clientes
- Aspirantes a empleo (solicitudes de empleo)
- Órdenes de servicio

2.2.6. El Diccionario de Datos.

Generalmente la información contenida en un diagrama de clases no es suficiente para comunicar a otros el resultado de un análisis. Se complementa con un diccionario de datos que contiene descripciones textuales de todos los elementos que aparecen en los diagramas.

La descripción de una clase permite saber si un objeto pertenece o no a ella. Debe ser lo más clara posible. A continuación presentamos algunos consejos útiles para elaborar el texto.

- **Criterio de inclusión.** Describe las características de los objetos que pertenecen a la clase.

- **Criterio de exclusión.** Puedes mencionar también si hay algunos objetos que no pertenecen a la clase.

- **Relaciones con otros objetos.** Explica como los objetos de una clase están relacionados con objetos de otras clases.

- **Información adicional.** Se puede incluir información de cómo ocurren los procesos en los que la clase aparece.

- **Estandarización.** Generalmente la descripción es un texto breve en tiempo presente de indicativo. Evita términos ambiguos.

- Se recomienda que las descripciones se elaboren al comenzar a realizar los diagramas.

2.2.7. Los Nombres de las Clases.

Una buena elección de nombres ayuda a mejorar la comprensión de los modelos. Trata de que sean claros y directos, aunque esto no siempre es fácil. Esta parte es muy importante, porque como una base de datos es el núcleo de las operaciones organizacionales, fija un estándar de cómo llamarle a las cosas que se manejan dentro de la empresa. Las organizaciones en ocasiones usan el mismo nombre para diferentes cosas o de maneras ambiguas. Estos problemas deben resolverse con un diccionario de datos que tenga nombres bien seleccionados. A continuación te presentamos algunos consejos para realizar esta labor.

- Lo mejor es usar términos de uso común, siempre y cuando no presenten los problemas que se mencionaron antes. Esta es una buena oportunidad para romper ambigüedades.
- Usa nombres concisos y cortos.
- Usa nombres que contrasten unos de otros en los mismos contextos.
- Precisión. Agrega adjetivos a nombres cortos para hacerlos más descriptivos.
- Selecciona nombres que describan la esencia de los objetos.
- Selecciona nombres que correspondan al contexto que estás trabajando.

- Evita abusar de términos comunes.

2.2.8. Prueba tus Clases.

Cuando empiezas el proceso de identificar clases, se corre el riesgo de considerar cosas que no lo son. He aquí algunas pruebas para saber si realmente son clases o no.

- **Uniformidad**. Debes asegurarte que todos los objetos que están contemplados para la misma clase tengan exactamente los mismos atributos, dominios, el mismo comportamiento y están sujetos a las mismas reglas.

- **Identifica atributos.** Todo objeto tiene atributos. Si los objetos de la clase no tienen otro atributo aparte de su descripción, es probable que tengas un atributo o alguna otra cosa, pero no una clase.

- **La prueba de conectivos.** Si el criterio de inclusión para la descripción de la clase utiliza la palabra "o" de una manera relevante, probablemente tengas un conglomerado de varias clases. Este problema se corrige añadiendo más clases.

- **Evita listas.** Si el criterio de inclusión contiene un listado de objetos específicos, probablemente no estés hablando de una clase.

2.3. Identificación de Atributos.

Cuando trabajas con una clase le debes identificar un conjunto de atributos que tengan las siguientes características:

- Se contempla toda la información requerida por el problema que estás resolviendo.
- Cada atributo representa aspectos diferentes de los objetos.
- Cada atributo toma sus valores independientemente de los otros.

Para identificar un atributo puedes hacerte las siguientes preguntas:

- ¿Qué características son comunes a todos los objetos de la clase que estoy modelando?
- ¿Qué información necesito saber de un objeto para poder considerarlo parte de la clase que estoy modelando?

Los atributos generalmente caen en una de las siguientes categorías:

- **Atributos descriptivos.** Proporcionan características intrínsecas del objeto.

- **Atributos para asignar nombres.** Son nombres arbitrarios que se le ponen a los objetos para poder distinguir unos de otros. Nota que sólo sirven para distinguir. Si les cambias el nombre no significa que se convierta en un objeto diferente; simplemente le has cambiado la forma de reconocerlo.

- **Atributos referenciales.** Sirven para identificar que un objeto está ligado con otro. **Cuando se trate de modelos conceptuales debemos excluir los atributos de este tipo.** En vez de ellos pondremos ligas y relaciones.

- **Identificadores.** Es un conjunto de atributos que distinguen de manera única cada objeto y no se deben modificar. **En nuestros diagramas conceptuales solamente incluiremos estos atributos sin forman parte de forma natural en los objetos que estemos modelando.**

2.3.1. Diccionario de Datos.

El diccionario de datos también debe incluir una descripción corta para los atributos. En caso de usar alguna convención en la estructura de los valores, ésta debe aparecer en el diccionario.

También se debe indicar el dominio de cada atributo. En caso de que los dominios tengan un nombre específico, también se debe tener una entrada para describirlos.

2.3.2. Dominios.

Los dominios se pueden describir de las siguientes maneras:

- **Enumeración:** proporcionar una lista explícita de sus valores.
- **Citas:** mencionar un documento donde se describa el dominio.
- **Reglas de aceptación:** mencionar una regla que permita calcular si un valor propuesto forma o no parte del dominio.
- **Rangos:** proporcionar las unidades y los valores de los extremos de un rango de números.

2.3.3. Valida tus Atributos.

Para saber si tus atributos están bien elegidos realiza las siguientes observaciones:

- Asegúrate que los atributos estén definidos para todos los objetos que correspondan a la misma clase.

- En algunas ocasiones la información de varios atributos se junta en uno solo. Esto no nos ayuda mucho. Debes asignar atributos diferentes para cada aspecto diferente del objeto.

- Los atributos deben corresponder a todo el objeto y no a una parte del mismo.

- Los atributos que no sean identificadores deben ser independientes uno del otro. Si están relacionados, posiblemente estás juntando dos clases de objetos en uno solo y debes separarlas.

2.4. Asociaciones.

Cuando se analiza o desarrolla un sistema, los objetos tienen asociaciones entre ellos. Para modelarlas en un diagrama de clases utilizamos **asociaciones**. Ellas nos indican las restricciones que se deben aplicar al relacionar dos objetos de diferentes clases.

Las asociaciones pueden establecerse entre objetos de la misma clase. También se permite relacionar objetos de dos o más clases

2.4.1. Asociaciones Binarias.

Las primeras relaciones que estudiaremos son las que se establecen entre objetos de dos clases, también conocidas como binarias. Se pueden agrupar en cuatro tipos diferentes conocidos como:

1. Uno a uno.
2. Uno a muchos.
3. Muchos a muchos.
4. Generalización.

A continuación mostramos la forma de representar en diagramas cada una de esas formas, también indicamos la forma de leerlas y su significado, pero antes hacemos algunas observaciones.

La **multiplicidad** de una asociación indica cuantas ligas puede establecer un elemento de una clase con elementos distintos de otra clase y viceversa. En la siguiente figura se muestra que un ganadero puede tener muchas vacas, pero que cada vaca puede pertenecer a un solo ganadero. Esto implicaría poner dos flechas.

Figura 2.8 – Ejemplo vaca ganadero original.

Para ahorrar espacio, las dos flechas se compactan en una sola línea que no se marca con flechas.

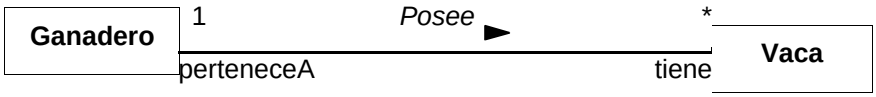


Figura 2.8 – Ejemplo vaca ganadero con una sola línea.

La relación puede tener un nombre, que en este caso es *Posee*. El triangulito negro indica el sentido en que se lee y es opcional. Es decir, podemos leer "Los ganaderos poseen vacas". Las dos palabras que están pegadas a los nombres de las clases reciben el nombre de **roles** o **extremos de asociación** e indican como ve una clase al otro extremo de una asociación. Podemos ver que una vaca pertenece a un ganadero y que un ganadero puede tener muchas vacas.

Nota que indicamos que un ganadero puede tener vacas, pero dejamos abierta la posibilidad a que no tenga. Por ello, se puede indicar un mínimo en la multiplicidad de la siguiente manera:

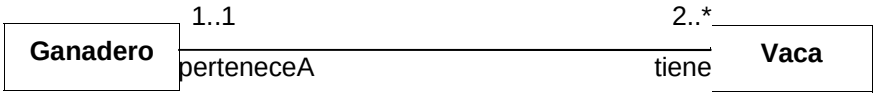


Figura 2.9 – Ejemplo vaca ganadero con multiplicidades mínimas y máximas.

Esto indica que un ganadero al menos debe tener dos vacas y que cada vaca debe tener forzosamente un dueño.

2.4.1.1. Asociaciones Uno a Uno.

En una asociación uno a uno, un objeto *a* de una clase *A*, está relacionado con un solo objeto *b* de otra clase *B*. Ese mismo objeto *b* sólo está relacionado con un objeto de la clase *A*, que resulta ser el mismo objeto *a* que se mencionó al inicio. Observa el siguiente ejemplo de objetos con ligas uno a uno.

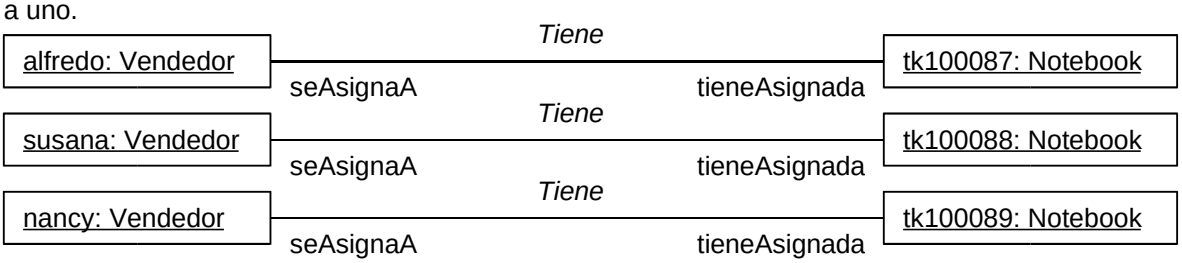


Figura 2.10 – Ejemplo 1 – 1 Vendedor – Notebook (objetos).

Observa como el vendedor *alfredo* tiene asignada una notebook, que está identificada con el número de serie *tk100087*. Ahora observa que esta última notebook (*tk100087*) solamente está conectada con *alfredo*. Esta relación se expresa con las siguientes dos frases:

- Un vendedor tiene asignada **una** notebook.
- Una notebook se asigna a **un** vendedor.

Observa que las frases empiezan en singular. El lado derecho de ambas frases es el que nos indica el tipo de relación y aparece marcado con negrillas.

Recuerda que para cada línea que veas, se tiene en realidad dos flechas y debe leerse con dos frases.

El diagrama de clases correspondiente a esta situación es el siguiente:

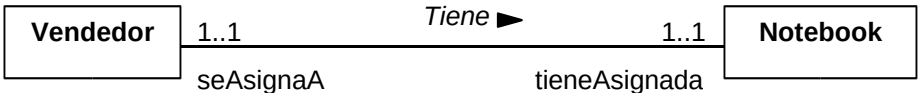


Figura 2.11 – Diagrama de clases Vendedor – Notebook 1 – 1 con roles.

El nombre de la relación es: *Tiene.*

Los roles son *seAsignaA* y *tieneAsignada*.

2.4.1.2. Asociaciones Uno a Muchos.

En una asociación uno a uno a muchos, un objeto *a* de una clase *A*, está relacionado con varios objetos de la clase *B*. Esos mismos objetos de *B* sólo están relacionados con un sólo objeto de la clase *A*, que resulta ser el mismo objeto *a* que se mencionó al inicio. Observa el siguiente ejemplo de objetos con ligas uno a muchos; en él se han omitido los roles por motivos de espacio.

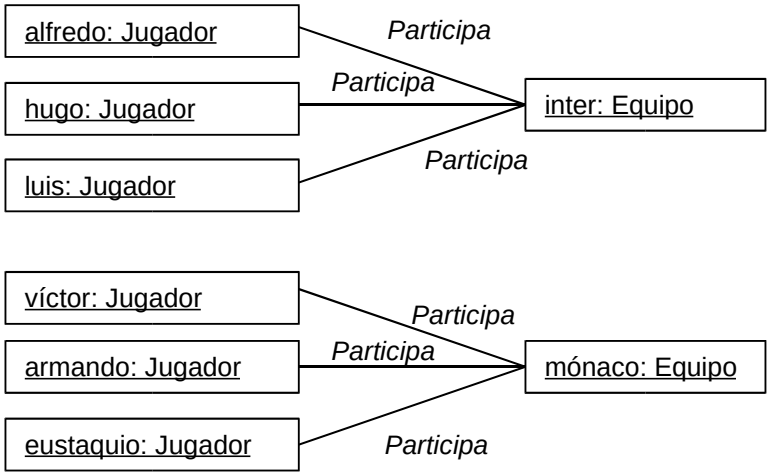


Figura 2.12 – Diagrama 1 – muchos Jugador – Equipo (objetos).

Observa como el equipo *inter* tiene como jugadores a *alfredo*, *hugo* y *luis*. Así mismo, estos jugadores participan en un solo equipo (precisamente *inter*). Esta relación se expresa con las siguientes dos frases:

- Un jugador participa en **un** equipo.
- Un equipo consta de **varios** jugadores.

Observa que las frases empiezan en singular. El lado derecho de ambas frases es el que nos indica el tipo de relación y se muestra en negrillas.

Recuerda que para cada línea que veas, se tienen en realidad dos flechas y debe leerse con dos frases.

El diagrama de clases correspondiente a esta situación es el siguiente:

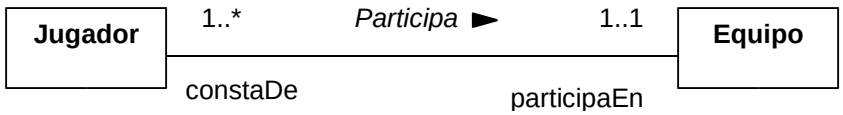


Figura 2.13 – Diagrama de clases Equipo – Jugador 1 – muchos con roles.

El nombre de la relación es: *Participa.*

Los roles son *participaEn* y *constaDe*.

2.4.1.3. Asociaciones Muchos a Muchos.

En una relación muchos a muchos, un objeto de una clase *A*, está relacionado con varios objetos de la clase *B*. Estos mismos objetos de *B* están relacionados con un varios objetos de la clase *A*. Observa el siguiente ejemplo de objetos con ligas muchos a muchos. (Por cuestiones de espacio se han omitido los roles y los nombres de las ligas.)

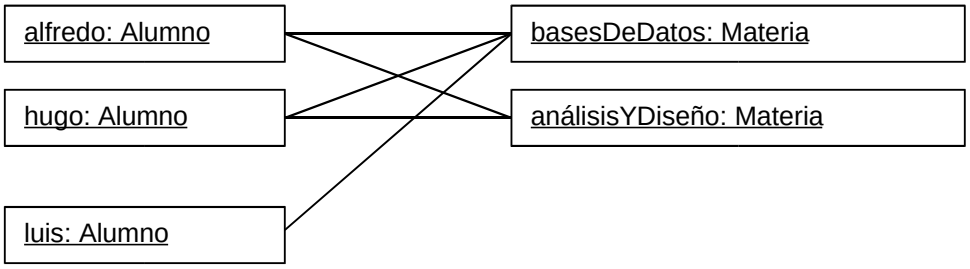


Figura 2.14 – Diagrama muchos – muchos Alumno – Materia (objetos).

Observa como el alumno *alfredo* cursa las materias *basesDeDatos* y *análisisYDiseño*. Ahora observa de manera aislada la materia *análisisYDiseño*, que es cursada por *alfredo* y *hugo.* Estas relaciones se expresan con las siguientes dos frases:

- Un alumno cursa **varias** materias
- Una materia es cursada por **varios** alumnos

Observa que las frases empiezan en singular. El lado derecho de ambas frases es el que nos indica el tipo de relación.

Recuerda que para cada línea que veas, se tienen en realidad dos flechas y debe leerse con dos frases. El diagrama de clases correspondiente a esta situación es el siguiente:

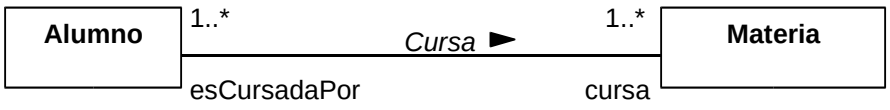


Figura 2.15 – Diagrama de clases Alumno – Materia muchos – muchos.

El nombre de la relación es: *Cursa*.

Los roles son *cursa* y *esCursadaPor.*

2.4.1.4. Generalización.

A diferencia de los conceptos vistos anteriormente, la generalización sirve cuando se necesita clasificar por tipos y para organizar la información de manera que no se duplique. En el siguiente ejemplo, se muestra el caso de un centro deportivo en el cual tienen lugar diferentes competencias en diferentes deportes. Se necesita manejar información sobre cada deportista en general por cuestiones de seguros, para cada premio a los jugadores se necesita registrar información especifica de cada deporte. El siguiente diagrama muestra como se puede organizar la información en clases.

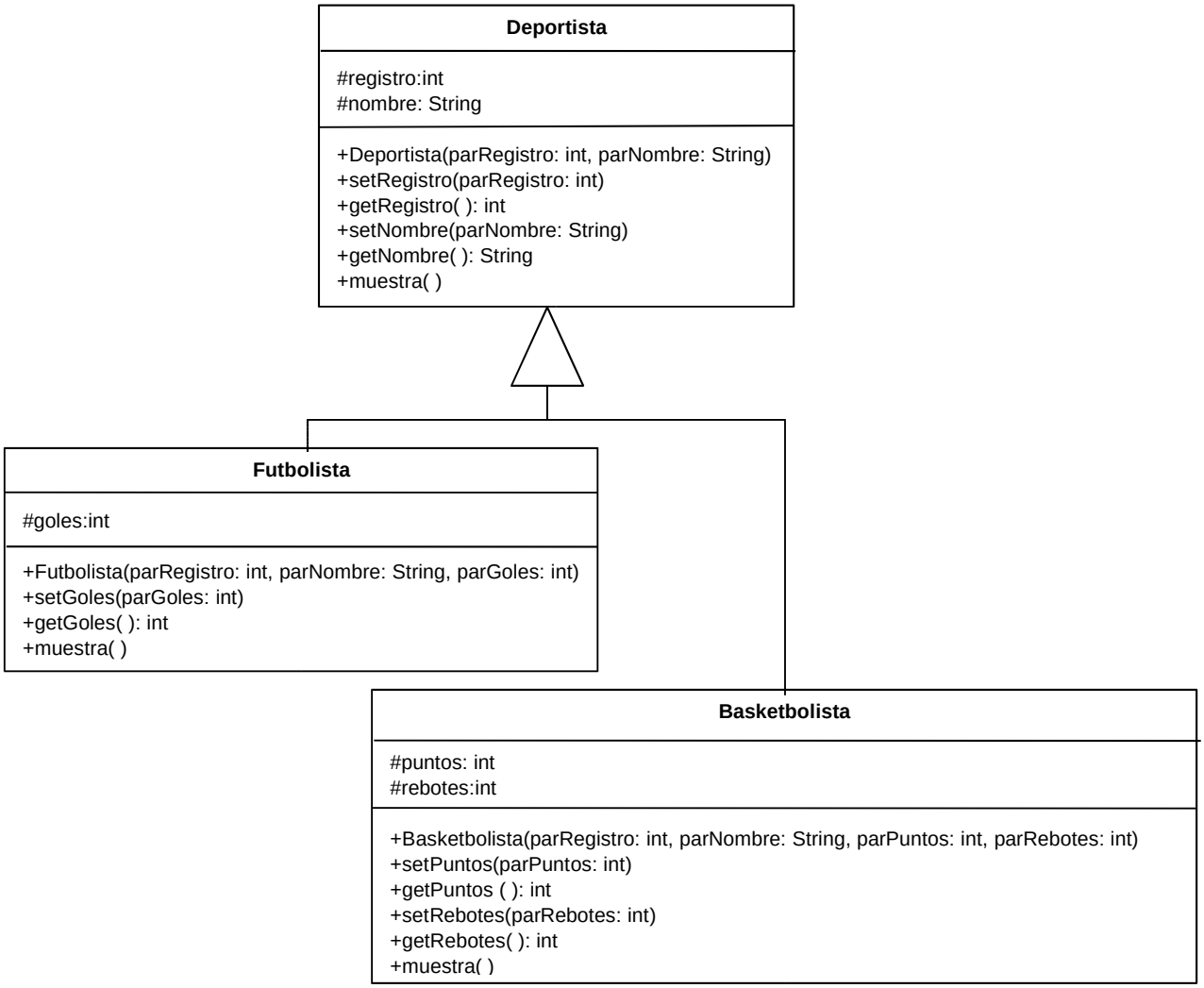


Figura 2.16 – Diagrama de clases para ejemplificar la generalización.

Como caso excepcional, en el diagrama de clases se han incluido los atributos y operaciones con el fin de hacer más clara la explicación. La clase que se encuentra en la punta del triángulo se llama clase **padre**. Las que están del otro lado del triángulo se conocen como clases hijas. Los objetos que pertenecen a estas últimas tienen los mismos atributos y operaciones que los objetos que pertenecen a la clase padre, pero adicionalmente poseen los atributos que marcan sus propios descriptores. En la siguiente figura ejemplificamos como quedarían internamente objetos de las tres clases.

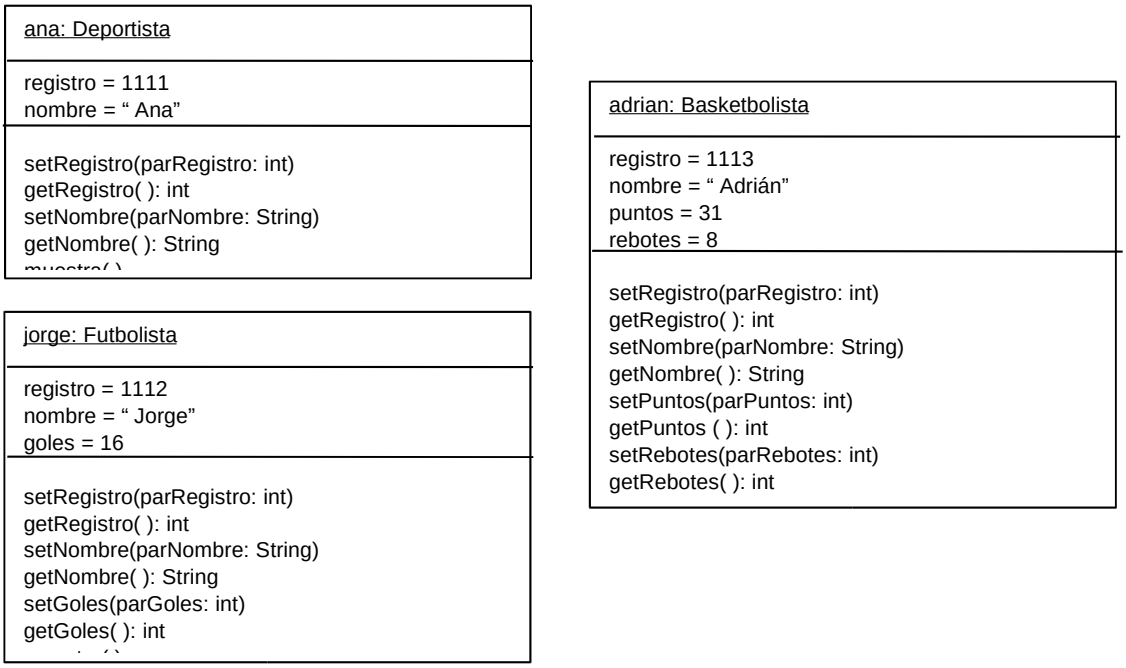


Figura 2.17 – Diagrama de objetos para ejemplificar la generalización.

Se dice que los miembros del descriptor de la clase padre se **heredan** a los descriptores de las clases hijas. La asociación entre clases padres e hijas se conoce como **generalización**. Dado que los objetos de las clases Futbolista y Basketbolista contienen todos los atributos y operaciones de un Deportista, se considera que también pertenecen a esta clase. Así pues, las clases hijas son subconjuntos de la clase padre.

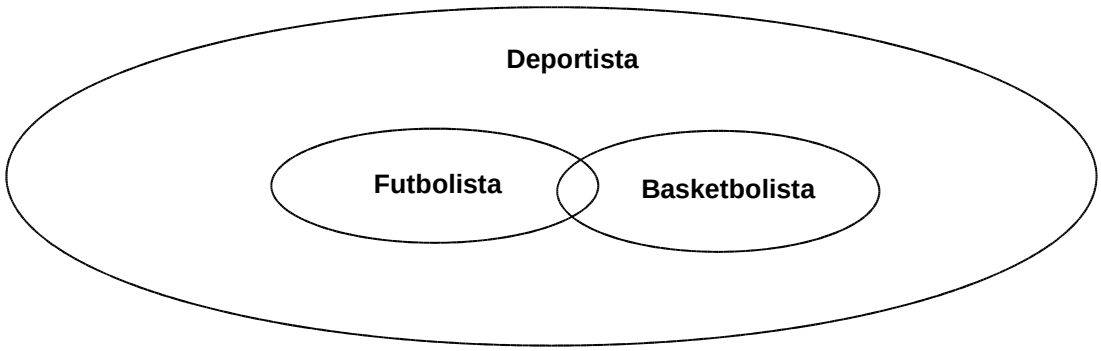


Figura 2.18 – Diagrama de Venn para ejemplificar la generalización.

Nota que los subconjuntos se pueden intersectar, como sería en el caso de deportistas que juegan ambos deportes. Muchos lenguajes de programación no permiten que las clases hijas se intersecten, pero hay algunas técnicas para simular este comportamiento. Otro caso especial de la generalización es que una clase hija puede tener varias clases padre, pero este tipo de relaciones debe manejarse con sumo cuidado.

Las frases que describen esta situación son las siguientes:
- Todo deportista **puede ser** futbolista o basketbolista.
- Todo basketbolista **es un** deportista.
- Todo futbolista **es un** deportista.

Otra forma de leer el diagrama es:
- Un futbolista **es un tipo de** deportista.
- Un basketbolista **es un tipo de** deportista.

En muchas ocasiones, al momento de modelar se observa que objetos que aparentemente pertenecen a la misma clase tienen más atributos que otros. En estos casos, posiblemente necesites usar la generalización. Los atributos que todos los objetos tienen, son colocados en una clase padre y se generan clases hijas para atributos que son comunes solo a una parte de los objetos. La generalizaciones pueden tener varios niveles. Por ejemplo, los *futbolistas* se pueden clasificar en base a su posición, por ejemplo en *Defensas*, los cuales todavía se pueden clasificar en *Laterales* y *Centrales*.

2.4.2. Asociaciones n-arias.

Este tipo de asociaciones se establecen entre objetos de tres o más clases. En ciertos casos, es posible descomponer la relación en otras de grado menor, pero no siempre. Se debe cuidar que las relaciones elegidas representen **claramente** el problema en cuestión. He aquí un ejemplo.

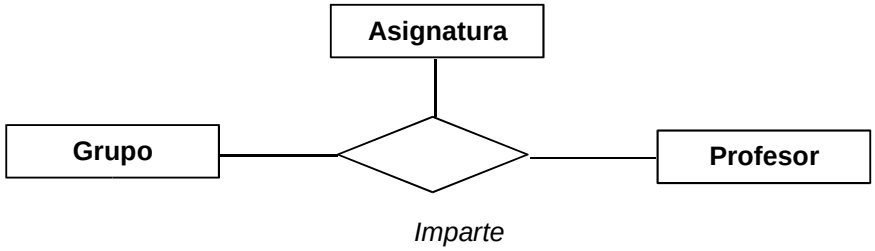


Figura 2.19 – Diagrama de Venn para ejemplificar la generalización.

2.4.3. Clases Asociativas.

Puede suceder que las asociaciones tengan atributos, como en los ejemplos que siguen.

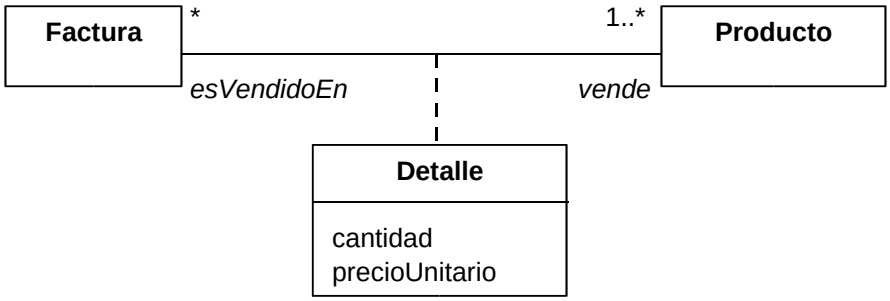


Figura 2.20 – Clase asociativa con dos extremos de asociación.

El nombre de la asociación se pone en el rectángulo que se une con línea punteada a la línea de asociación. En ese mismo rectángulo se ponen los atributos de la asociación. A continuación se muestra como estos conceptos se aplican a relaciones n-arias.

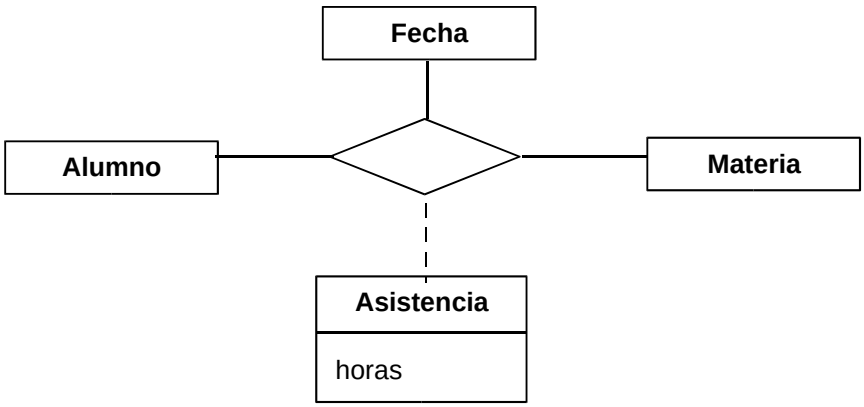


Figura 2.21 – Clase asociativa con tres extremos de asociación.

En este último ejemplo, la clase Fecha tal vez no se implemente como tabla en un sistema de bases de datos, pero nos sirve para entender como registrar la asistencia de un alumno a clases. Aún así, al diseñar la tabla para Asistencia, si es conveniente considerar la fecha como una clase. En el caso de programación, la clase Fecha si se considera.

2.5. El aspecto temporal.

La información que se almacena cambia con el tiempo. Así, tenemos cambios de parejas, jugadores que cambian de equipos, etc. Los modelos representados hasta ahora representan la situación presente de los datos. ¿Qué podemos hacer si nos interesa guardar información histórica y saber a qué periodo de tiempo pertenece?

Una solución es añadir fechas como atributos:

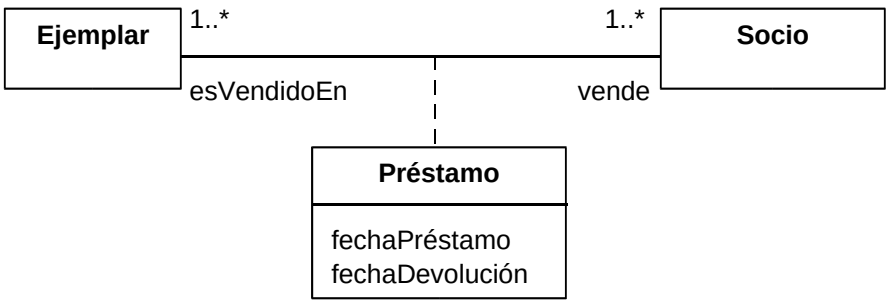


Figura 2.22 – Ejemplo de manejo del tiempo usando atributos.

Cuando el aspecto temporal está delimitado por periodos bien establecidos, tales como temporadas, ciclos escolares, etc, el tiempo se puede representar con clases.

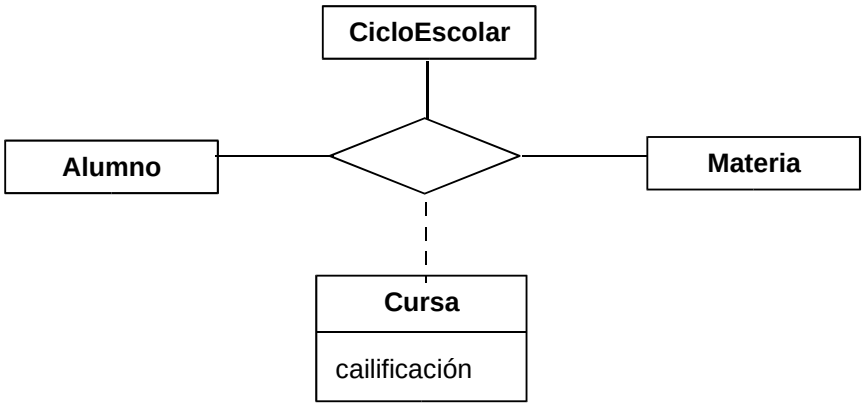


Figura 2.23 – Ejemplo de manejo del tiempo usando clases.

También pueden suceder que con el paso del tiempo, las propiedades que nos interesan sobre un objeto cambien. En este caso, pueden cambiar de un tipo hacia uno de los subtipos.

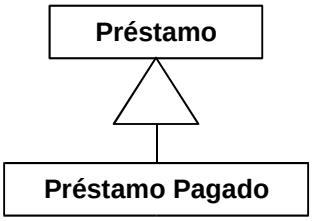


Figura 2.24 – Ejemplo de generalización para modelar la evolución de un objeto en el tiempo.

2.6. Agregación.

Hay algunos objetos cuya existencia depende de la existencia de otros. A esta relación se le conoce como **agregación de composición**. En la figura siguiente se muestra como los temas de una asignatura dependen de la existencia de la asignatura en sí. Si desaparece la asignatura, desaparecen los temas.

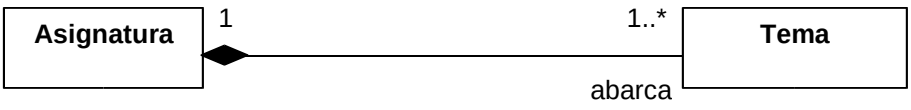


Figura 2.25 – Ejemplo de agregación.

2.7. Atributos Calculados.

También puede darse que un atributo se puede calcular a partir de otros, como en el caso de subtotales y totales. En este caso, se les antepone una diagonal.

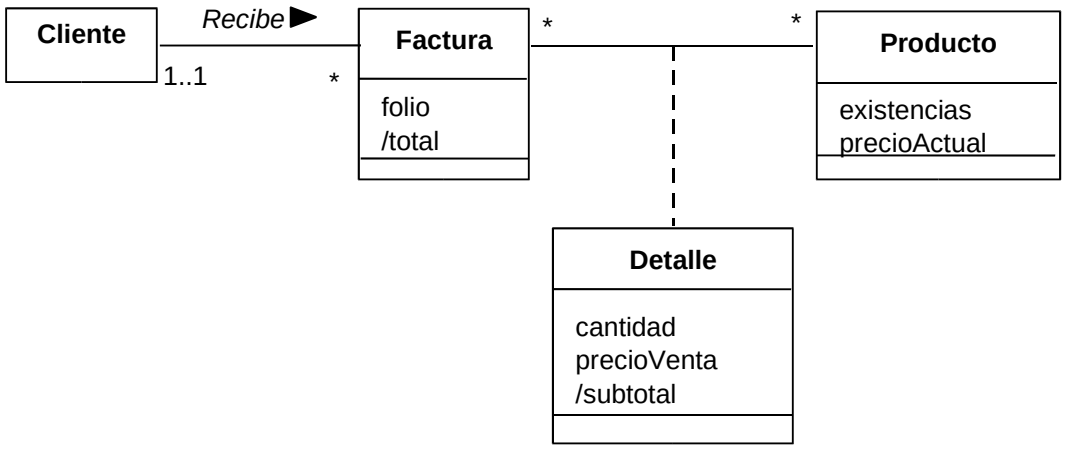


Figura 2.25 – Ejemplo de atributos calculados.

3. Modelo Relacional.

3. Modelo Relacional

Es el modelo más difundido para implementar bases de datos. En el, todos los datos se organizan de la manera que se esquematiza a continuación

Nombre de la relación

Atributo 1	**Atributo 2**	**.....**	**Atributo n**	
Valor 1	Valor 2	.....	Valor n	*Tupla 1*
Xxxxxxx	xxxxxxxx		Xxxxxxxx	*Tupla 2*
. . .	. . .	. . .	. . .	. . .
Xxxxxxx	xxxxxxxx	.....	Xxxxxxxxx	*Tupla m*

Figura 3.1 – Estructura genérica de una relación en el modelo relacional.

Cada **tupla** representa un objeto. Los atributos representan características del mismo. Todas las tuplas tienen varios **atributos**. Cada atributo tiene un valor asociado. El conjunto de los valores que puede tomar un atributo se conoce como **dominio**. Los valores de un dominio no tienen estructura interna (es decir que son **atómicos**) y no contiene estructuras repetitivas (**univaluados**). Un conjunto de tuplas se pueden juntar para formar una **relación**. En ella, todas las tuplas tienen los mismos nombres y dominios para un número dado de atributo. Así mismo, todas las tuplas de una relación tienen el mismo número de atributos.

En la práctica los sistemas de bases de datos relacionales utilizan objetos llamados **tablas**, que normalmente se manejan como si fueran relaciones.

3.1. Claves.

- **Candidatas.** Conjunto no vacío de atributos que identifican unívoca y minimamente cada tupla, las llaves representan la identidad de un objeto.

- **Primarias**. De entre todas las llaves candidatas para una relación, hay una que se escoge como oficial.

- **Alternativas**. Toda las claves candidatas que no se seleccionaron como primaria.

- **Foráneas**. Conjunto no vacío de atributos, cuyos valores deben coincidir con los valores de la clave primaria de otra relación.

3.2. Restricciones.

Como parte del modelo, hay ciertas restricciones o reglas que deben cumplir los datos. Estas se determinan al estudiar un problema. Las principales son:

- De dominio
- De integridad referencial

Como ejemplo estas últimas, supóngase el caso de un deportivo donde se lleva el registro de equipos y sus jugadores. Se tiene la tabla de equipos, donde la clave primaria es la combinación del nombre del equipo y la categoría. Por otro lado, se tiene una tabla con los datos de los jugadores. En esta, la llave primaria es el número de registro. Para saber en que equipo participa cada jugador, añadimos una llave foránea, que coincide con la llave primaria del equipo.

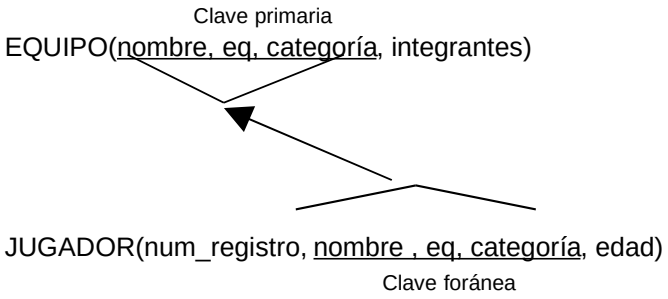


Figura 3.2 – Ejemplo de integridad referencial entre dos relaciones.

Un ejemplo de cómo quedarían los datos es el siguiente:

EQUIPO

nombre eq.	**Categoría**	**integrantes**	
Inter.	Master	20	
Inter.	Infantil A	18	
Santos	Juvenil	19	

JUGADOR

num_registro	**nombre**	**nombre eq**	**categoría**	**edad**
M1234	José Smith	Inter.	Master	45
1431	Pedro Negrete	Inter.	Infantil A	7
J9012	Susano Miguel	Santos	Juvenil	18
K3011	Eduardo Bisnes	Inter.	Master	38

Figura 3.3 – Ejemplo de cómo pueden almacenarse datos con integridad referencial.

La relación entre estas dos tablas debe cumplirse siempre. Así pues, un cambio en la tabla equipo repercutirá en el contenido de la tabla jugador. Las operaciones de inserción, borrado y modificado cuando hay integridad referencial nos llevan a las siguientes situaciones:

Para poder insertar en una relación que tiene claves foráneas, es necesario que los datos referidos ya estén integrados a la otra tabla. Por ejemplo, primero se debe insertar en la tabla equipo. Algunas formas de manejar borrados y modificaciones son los siguientes:

- **Operaciones restringidas (RESTRIC).** Solo se puede borrar aquellas tuplas que no son referidas por ninguna otra tupla. Se prohíbe modificar una llave primaria que esté referenciada.
- **Operaciones con transmisión en cascada (CASCADE).** El borrado o modificación de una clave primaria conlleva el borrado o modificación de los valores relacionados en otras tuplas.

- **Operaciónes con puesta en nulos (SET NULL).** El borrado o modificación de llaves primarias llevan a poner como nulos los valores correspondientes en las tuplas que les hacían referencia.
    - **Asignar un valor por defecto.**
    - **Invocar una acción de usuario.**

3.3. Seguridad y Valores Nulos.

Otra característica de los manejadores de bases de datos que utilizan el modelo relacional es que permite el uso de vistas (control de acceso y visibilidad de los datos) y el manejo de valores nulos.

3.4. Álgebra Relacional.

Adicionalmente el modelo relacional incluye un conjunto de operaciones que permiten realizar las siguientes operaciones:

- Inserción.
- Borrado.
- Modificación
- Consulta. En este caso tenemos los operadores
    - Restricción.
    - Proyección.
    - Unión.
    - Diferencia.
    - Producto cartesiano.
    - Combinación.
    - Intersección.
    - División.

3.5. FORMAS NORMALES.

Un mal diseño de tablas pueden llevarnos a problemas como los siguientes:

- Incapacidad para almacenar ciertos datos.
- Redundancias e incoherencias.
- Perdida de información.
- Perdida de dependencias.
- Aparición de estados no validos en el mundo real (aparición de tuplas espúreas).
- Anomalías de modificación, inserción y borrado.

Después de muchos estudios, se han obtenido algunas reglas que indican como diseñar tablas que no presenten de los problemas antes mencionados. Estas reglas han evolucionado con el paso del tiempo y se conocen como formas normales. La forma en que se relacionan se muestra en el siguiente diagrama de Venn.

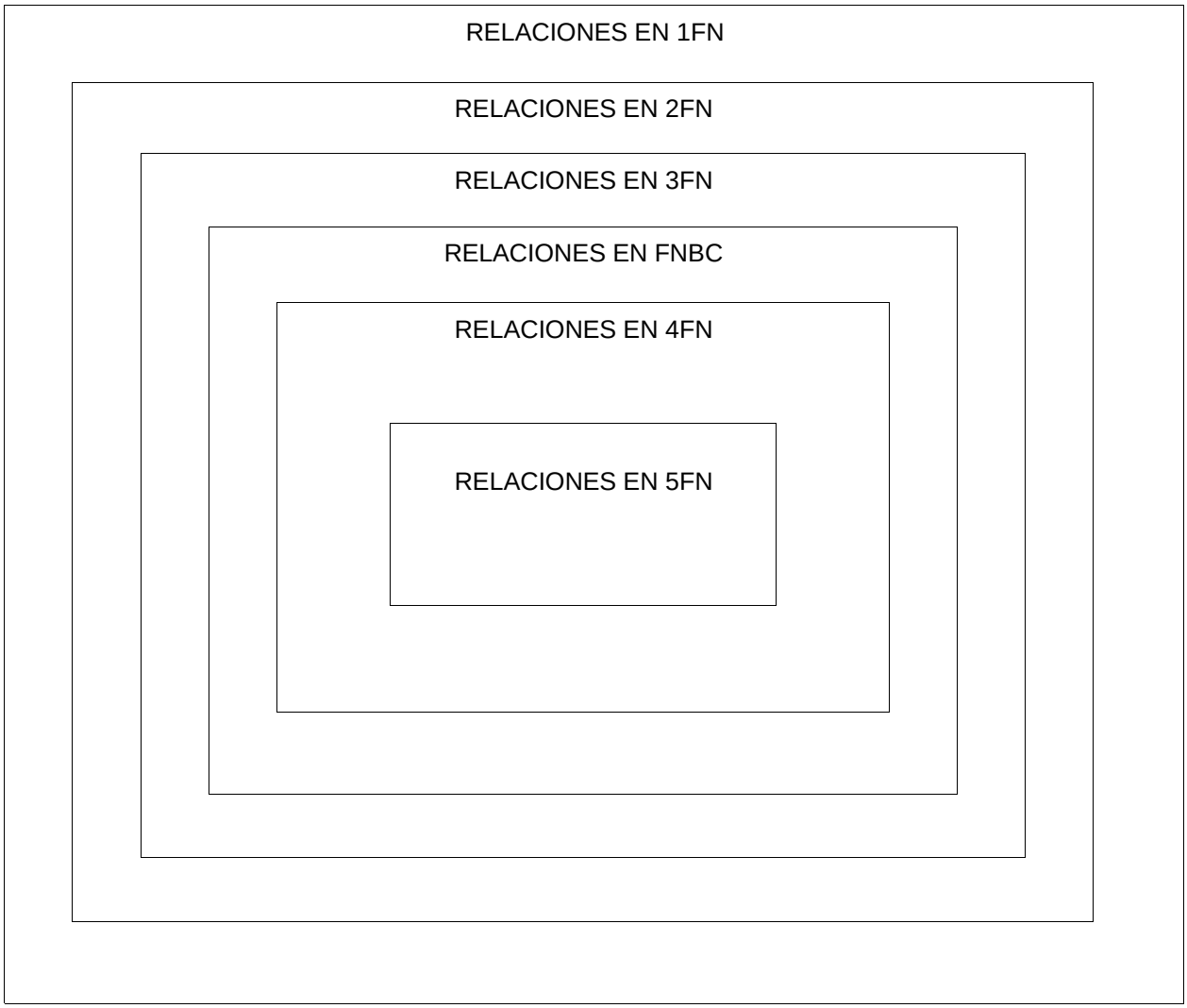


Figura 3.4 – Diagrama de Venn de las formas normales.

3.5.1. Primera Forma Normal (1FN).

Toda tabla que no tenga grupos repetitivos ni valores nulos está en **primera forma normal** y es una relación (también llamada **afinidad**).

3.5.2. Segunda Forma Normal (2FN).

Primero se necesita definir dependencia funcional

X --> Y Si solo si, cada valor de x tiene asociado en todo momento un valor de Y. También se puede leer como:

> Para conocer el valor de Y en un objeto, primero debes conocer el valor de X.
> Es suficiente conocer el valor de X para conocer el valor de Y.

X determina funcionalmente a Y.
Y depende funcionalmente de X.

En este ejemplo, X recibe el nombre de **determinante** o implicante, mientras que Y recibe el nombre de **implicado**.

Del ejemplo de jugadores podemos decir :

num_registro --> nombre

Una relación esta en **segunda forma normal** si

1. Esta en 1FN.
2. Cada atributo no clave depende funcionalmente de toda clave.

Del ejemplo de facturas tenemos:
folio, cve_prod --> cantidad
Otro ejemplo:
matricula, asignatura, parcial --> calificación

Dado que una clave trata de representar un objeto de la realidad, podríamos ver a la segunda forma normal como un intento de expresar que un atributo debe describir una característica de un objeto y que se debe tener una tabla por objeto. El enunciado aquí presentado no expresa del todo esa idea y ha dado lugar a algunos de los problemas que se mencionaron al inicio de este capitulo. Por ello es que se han definido otras formas normales.

3.5.3. Tercera Forma Normal (3FN).

En este caso tenemos el concepto de **dependencia funcional transitiva**. Si X-->Y, Y-->Z, X-->Z y no Y-->X se dice que Z tiene una dependencia transitiva respecto a X.

Una relación está en **tercera forma normal** si:

1. Esta en 2FN.
2. No tiene dependencias funcionales transitivas.

La idea que se ha tratado de expresar con la segunda forma normal aparece aquí en una mejor aproximación, pero aún no se logra completamente.

3.5.4. Forma Normal de Boyce/Codd(FNBC).

Una relación está en **forma normal de Boyce/Codd** si y solo si cada determinante es clave candidata.

Esta forma normal puede dar lugar a la perdida de dependencias funcionales, pero no presenta las anomalías ocasionadas por las formas normales anteriores. Aún no se ha expresado del todo la idea que se buscaba desde la segunda forma normal y por ello también presenta anomalías, que se tratan de corregir con las formas normales que se muestran a continuación.

3.5.5. Cuarta Forma Normal(4FN).

Dependencias multivaluadas:

X-->>Y si y solo si, si un cierto valor de X implica un conjunto bien definido de valores de Y con independencia de los demás atributos de la relación. Se lee: X multidetermina a Y .

Ejemplo: en este caso, un profesor tiene diferentes especialidades independientemente de la institución en la que trabaja. También el hecho de que trabaje en distintas instituciones es independiente de la especialidades que tenga. Por eso podemos escribir:

PROFESOR(profesor, especialidad, institución)
profesor-->>especialidad
profesor-->>institución

Una relación esta en **cuarta forma normal** si y solo si esta en FNBC y si toda dependencia multivaluada esta determinada por una llave candidata.

Una relación que no esta en cuarta forma normal de alguna manera ha mezclado dos o más entidades ligadas por relaciones de uno a muchos, obteniendo una sola tabla.

3.5.6. Quinta Forma Normal(5FN).

Observemos la siguiente tabla:

EDITA

Editorial	**Idioma**	**Tema**
Addison Wesley	Ingles	Base de Datos
Addison Wesley	Español	Case
Prentice Hall	Español	Base de Datos
Addison Wesley	Español	Base de Datos

Figura 3.5 – Tabla base.

Por alguna razón necesitamos fragmentar esta tabla. Para ello, dejamos el campo idioma en ambas tablas esperando poder juntarlas después. Obtenemos pues las siguientes proyecciones:

EDITA 1

Editorial	**Idioma**
Addison Wesley	Ingles
Addison Wesley	Español
Prentice Hall	Español
Addison Wesley	Español

EDITA2

Idioma	Tema
Ingles	Base de Datos
Español	Case
Español	Base de Datos
Español	Base de Datos

Figura 3.6 – Fragmentación incompleta de la tabla anterior.

Si tratamos de juntar estas tablas, aparecerá la tupla adicional <Prentice Hall, español, CASE>. Para evitar esto, debemos juntar EDITA1 y EDITA2 con la siguiente:

EDITA3

Editorial	Tema
Addison Wesley	Base de Datos
Addison Wesley	Case
Prentice Hall	Base de Datos
Addison Wesley	Base de Datos

Figura 3.7 – Tabla adicional para poder recuperar la tabla original.

Decimos entonces que EDITA tiene una dependencia de combinación respecto a EDITA1, EDITA2 y EDITA3.

Cuando se cumplen las formas normales anteriores y en toda dependencia de combinación esta implicada una llave candidata, tenemos una relación en quinta forma normal.

Una relación está en **quinta forma normal** si y solo si toda dependencia funcional, multivaluada o de combinación es consecuencia de claves candidatas.

Esta forma normal nos indica que al fragmentar una tabla, debemos incluir la llave primaria en cada una de sus proyecciones.

3.5.7. Forma Normal de Dominio/ Clave(FNDC).

Una relación se encuentra en **Forma Normal de Dominio/Clave** si y solo si, toda restricción es consecuencia lógica de restricciones de dominio y de clave.

No se ha demostrado si hay un algoritmo que permita transformar relaciones en 1FN hasta esta forma normal, pero si se sabe que cualquier afinidad que esté en FNDC también esta en 5FN.

3.5.8. Resultado Importante de las Formas Normales.

La moraleja aquí, es que hay que centrarse en describir objetos y dominios. Después de todo, una clave candidata representa a un objeto. Hay que dedicar una tabla para cada objeto y solo poner en ella atributos y restricciones que describan directamente a los objetos de la clase.

4. Diseño de Tablas.

4. Diseño de Tablas.

Para proceder al diseño de tablas se recomiendan los siguientes pasos:

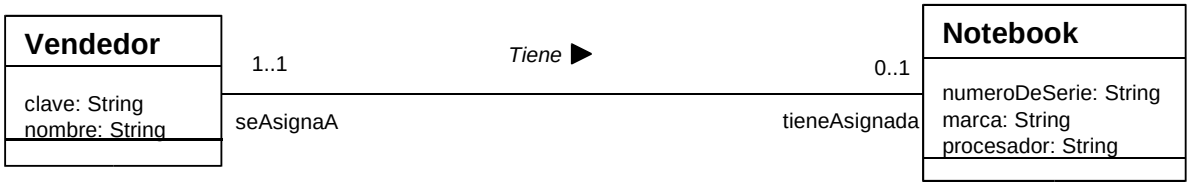
1. **Hacer un diagrama conceptual**. En este **no se marcan llaves primarias**. Solamente se colocan estas si forman parte de forma natural de los datos que se están analizando. Las operaciones del modelo no nos interesan para esta parte. Serán útiles al desarrollar otros elementos del sistema. Evita también agregar llaves foráneas en este instante; estas se agregarán en fases posteriores.
2. **Para cada clase diseñar una tabla** que tenga los mismos atributos, pero hay que adecuar sus tipos de datos a los permitidos para tu manejador de bases de datos.
3. **Agregar llaves primarias.** En el caso de que el objeto tenga atributos que se comporten como llaves candidatas, se elige una de esas llaves como primaria. Si no hay ninguna clave candidata, se agrega un atributo extra para que cumpla esta función.
4. **Añadir llaves foráneas y tablas adicionales** de acuerdo a las explicaciones que se presentan más adelante en este capítulo, según el tipo de relaciones que se establezcan entre las clases.

En las siguientes secciones se muestra como implementar cada uno de los casos estudiados en el capítulo dedicado al modelo conceptual. Para facilitar la comprensión del tema, cada caso incluye los siguientes subtemas:

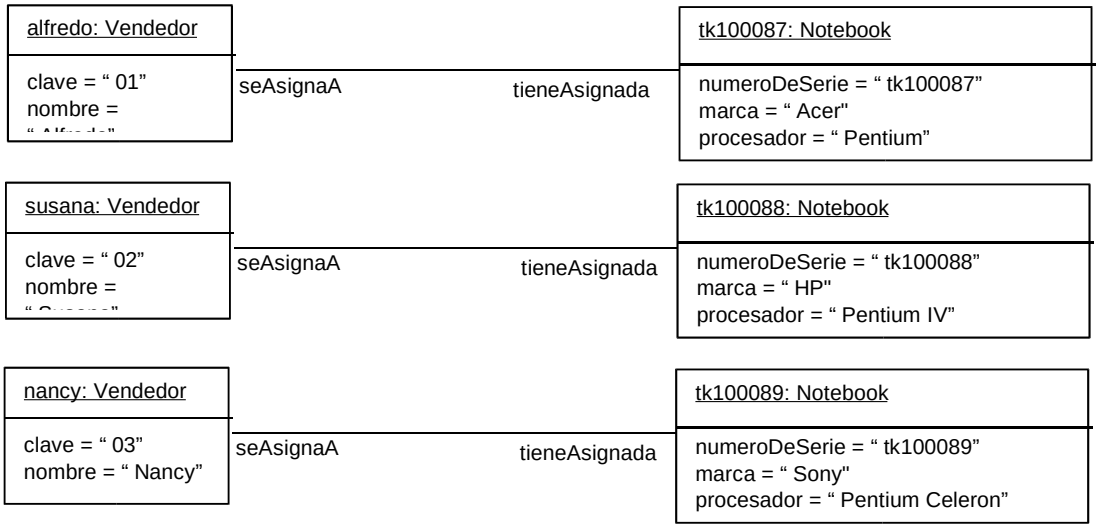
- Diagrama conceptual
- Diagrama de objetos
- Frases.
- Se muestra también el diseño de tablas, indicando que atributos o tablas extras añadir, utilizando el **UML Database Profile.**
- Ejemplo de como se almacenan los datos en tablas.
- Instrucciones en SQL genérico para construir los ejemplos y mostrar como pueden evolucionar con el paso del tiempo.

4.1. Relaciones Uno a Uno.

4.1.1. Diagrama Conceptual.



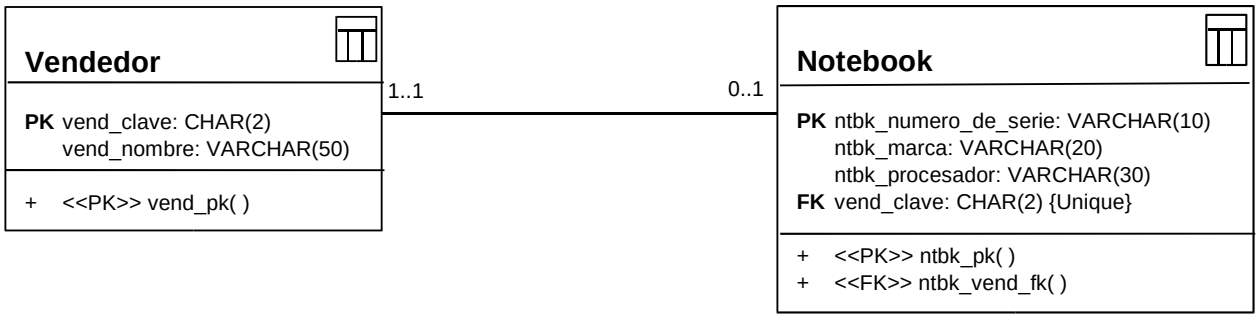
4.1.2. Diagrama de Objetos.



4.1.3. Frases.

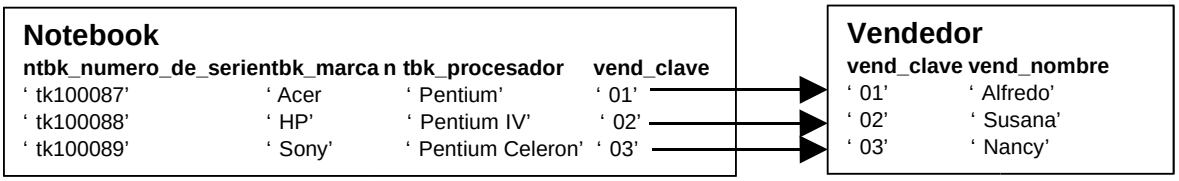
Un vendedor tiene asignada **una** notebook.
Una notebook se asigna a **un** vendedor

4.1.4. Diseño de la Base de Datos.



Se pone una llave foránea del lado que más convenga y esa llave restringe para no permitir duplicados, ya sea marcándola como *unique* o creando un índice sin duplicados para ese campo. Nota que el lado de la llave foránea queda opcional y el opuesto queda obligatorio.

4.1.5. Ejemplo de Tablas.



4.1.6. Instrucciones para Crear Tablas en SQL.

```
CREATE TABLE Vendedor
(
    vend_clave CHAR(2) NOT NULL,
    vend_nombre VARCHAR(50) NOT NULL,

    CONSTRAINT vend_pk PRIMARY KEY (vend_clave)
);

CREATE TABLE Notebook
(
    ntbk_numero_de_serie VARCHAR(10) NOT NULL,
    ntbk_marca VARCHAR(20) NOT NULL,
    ntbk_procesador VARCHAR(30) NOT NULL,
    vend_clave CHAR(2) UNIQUE NOT NULL,

    CONSTRAINT ntbk_pk PRIMARY KEY (ntbk_numero_de_serie)
);

ALTER TABLE Notebook
ADD CONSTRAINT ntbk_vend_fk FOREIGN KEY (vend_clave)
       REFERENCES Vendedor(vend_clave);

CREATE UNIQUE INDEX ntbk_uni_vend_clave ON Notebook(vend_clave);
```

Nota: Si el campo Notebook(vend_clave) se puede definir como UNIQUE, no es necesario crear el índice y viceversa.

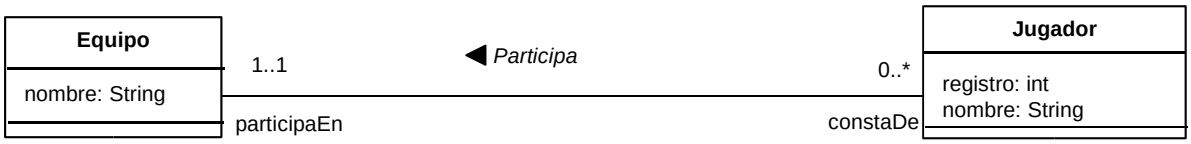
4.1.7. Instrucciones para Armar la Red en SQL.

```
/* Se crean y se conectan los nodos */
INSERT INTO Vendedor(vend_clave, vend_nombre) VALUES('01','Alfredo');
INSERT INTO Vendedor(vend_clave, vend_nombre) VALUES('02','Susana');
INSERT INTO Vendedor(vend_clave, vend_nombre) VALUES('03','Nancy');

INSERT INTO Notebook
       (ntbk_numero_de_serie, ntbk_marca, ntbk_procesador, vend_clave)
    VALUES('tk100087','Acer', 'Pentium', '01');
INSERT INTO Notebook
       (ntbk_numero_de_serie, ntbk_marca, ntbk_procesador, vend_clave)
    VALUES('tk100088','HP', 'Pentium IV', '02');
INSERT INTO Notebook
       (ntbk_numero_de_serie, ntbk_marca, ntbk_procesador, vend_clave)
    VALUES('tk100089','Sony', 'Pentium Celeron', '03');

/*Consulta información de alfredo y susana */
SELECT Vendedor.vend_clave, vend_nombre, ntbk_numero_de_serie,
    ntbk_marca, ntbk_procesador
FROM Vendedor, Notebook
WHERE
    Vendedor.vend_clave = Notebook.vend_clave
AND (vend_nombre = 'Alfredo' OR vend_nombre = 'Susana');
```

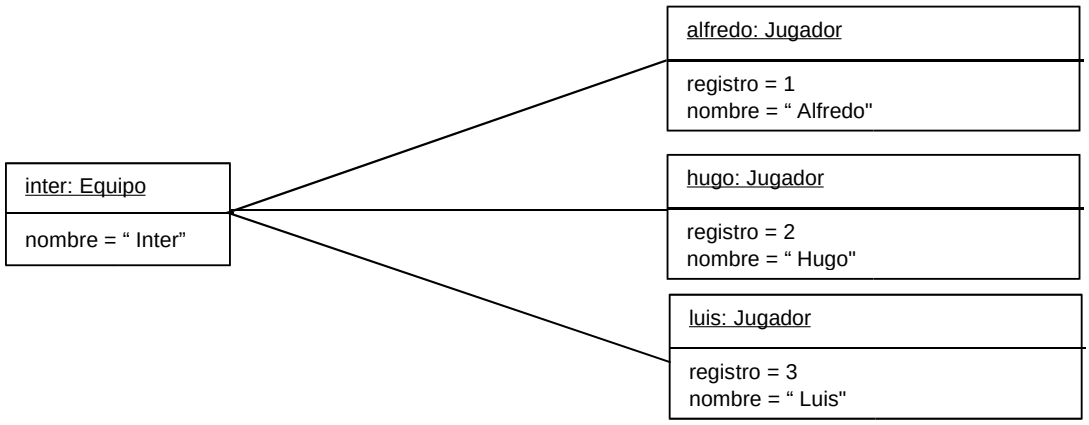
```
/*
  A susana le quitamos la computadora y se la damos a un nuevo vendedor
*/
INSERT INTO Vendedor(vend_clave, vend_nombre) VALUES('04','Penélope');

UPDATE Notebook
SET vend_clave = '04'
WHERE vend_clave = '02';

/*Otra forma de hacerlo */
UPDATE Notebook
SET vend_clave = '04'
WHERE vend_clave IN
    (SELECT vend_clave FROM Vendedor WHERE vend_nombre = 'Susana');

/*Muestra información de todos */
SELECT *
FROM Vendedor, Notebook
WHERE
    Vendedor.vend_clave = Notebook.vend_clave;
```

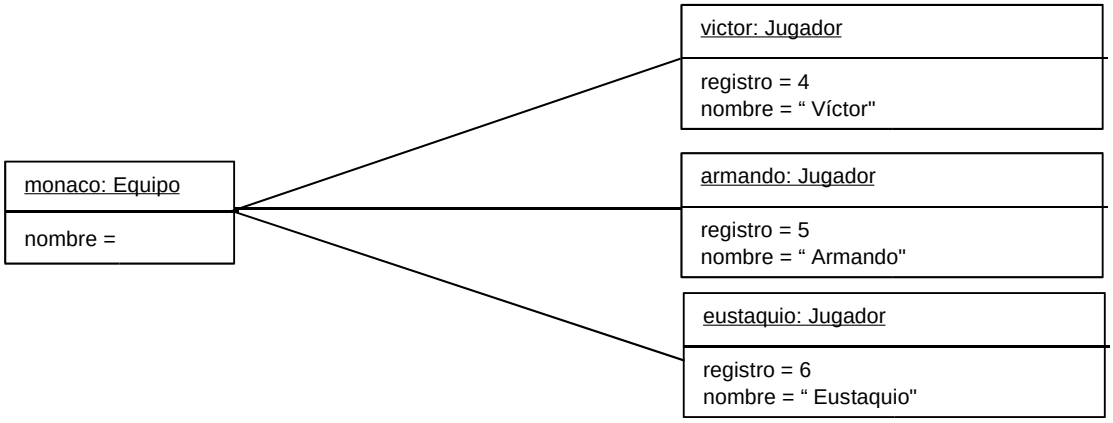
4.2. Relaciones Uno a Muchos.

4.2.1. Diagrama Conceptual.



4.2.2. Diagrama de Objetos.

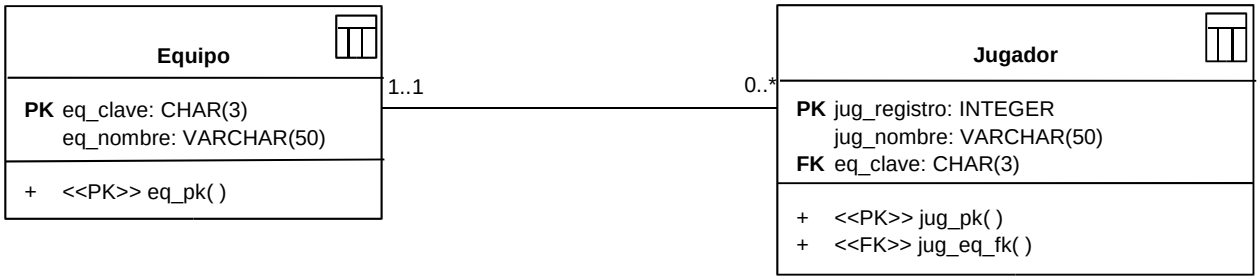




4.2.3. Frases.

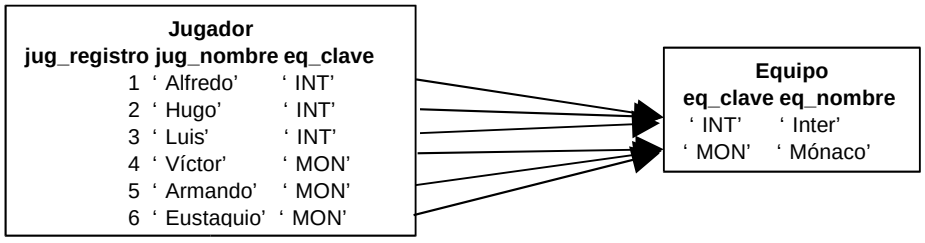
Un jugador participa en **un** equipo.
Un equipo consta de **varios** jugadores.

4.2.4. Diseño de la Base de Datos.



Se pone una llave foránea del lado muchos.

4.2.5. Ejemplo de Tablas.



4.2.6. Instrucciones para Crear Tablas en SQL.

```
CREATE TABLE Equipo
(
    eq_clave CHAR(3) NOT NULL,
    eq_nombre VARCHAR(50) NOT NULL,

    CONSTRAINT eq_pk PRIMARY KEY (eq_clave)
);

CREATE TABLE Jugador
(
    jug_registro VARCHAR(10) NOT NULL,
    jug_nombre VARCHAR(20) NOT NULL,
    eq_clave VARCHAR(30) NOT NULL,

    CONSTRAINT jug_pk PRIMARY KEY (jug_registro)
);

ALTER TABLE Jugador
ADD CONSTRAINT jug_eq_pk FOREIGN KEY (eq_clave)
       REFERENCES Equipo (eq_clave);
```

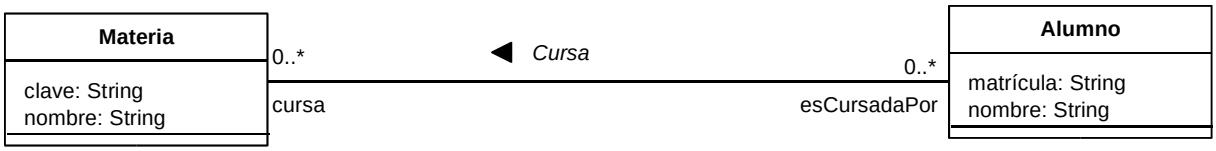
4.2.7. Instrucciones para Armar la Red en SQL.

```
/* Se crean y se conectan los nodos */
INSERT INTO Equipo(eq_clave, eq_nombre) VALUES('INT','Inter');
INSERT INTO Equipo(eq_clave, eq_nombre) VALUES('MON','Mónaco');

INSERT INTO Jugador(jug_registro, jug_nombre, eq_clave)
    VALUES(1, 'Alfredo','INT');
INSERT INTO Jugador(jug_registro, jug_nombre, eq_clave)
    VALUES(2, 'Hugo','INT');
INSERT INTO Jugador(jug_registro, jug_nombre, eq_clave)
    VALUES(3, 'Luis','INT');
INSERT INTO Jugador(jug_registro, jug_nombre, eq_clave)
    VALUES(4, 'Víctor','MON');
INSERT INTO Jugador(jug_registro, jug_nombre, eq_clave)
    VALUES(6, 'Eustaquio','MON');

/* Consulta información de los equipos */
SELECT *
FROM Equipo, Jugador
WHERE
    Equipo.eq_clave = Jugador.eq_clave
ORDER BY Equipo.eq_clave, jug_registro ;

/*
    Cambios en los equipos.
    Eustaquio se retira
*/
DELETE FROM Jugador WHERE jug_nombre = 'Eustaquio';
```

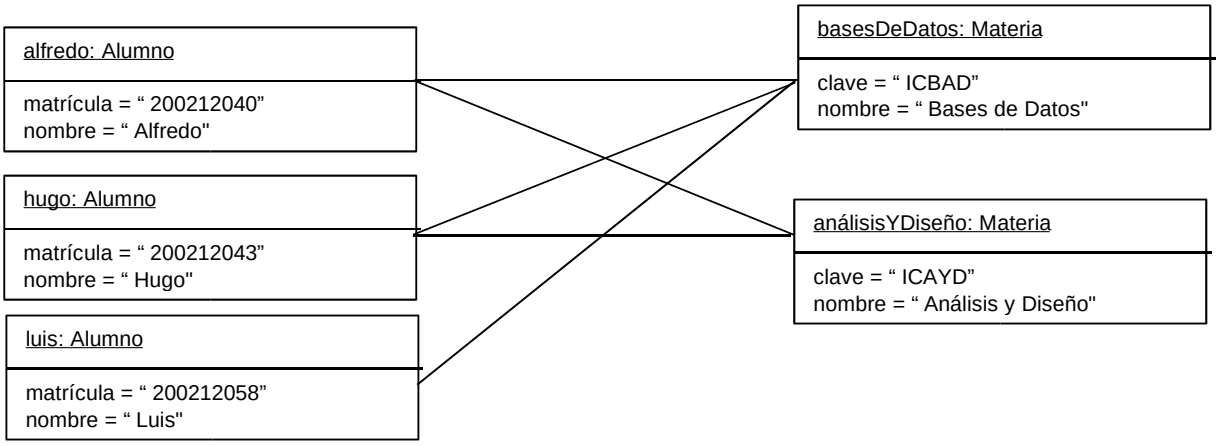
```
/*Luis se cambia al mónaco */
UPDATE Jugador
SET eq_clave = 'MON'
WHERE jug_registro IN
    (SELECT jug_registro FROM Jugador WHERE jug_nombre = 'Luis');

/* El Mónaco se cambia de nombre */
UPDATE Equipo
SET eq_nombre = 'San Luis'
WHERE eq_nombre = 'Mónaco';

/* Consulta información de los equipos */
SELECT *
FROM Equipo, Jugador
WHERE
    Equipo.eq_clave = Jugador.eq_clave
ORDER BY Equipo.eq_clave, jug_registro ;
```

4.3. Relaciones Muchos a Muchos.

4.3.1. Diagrama Conceptual.



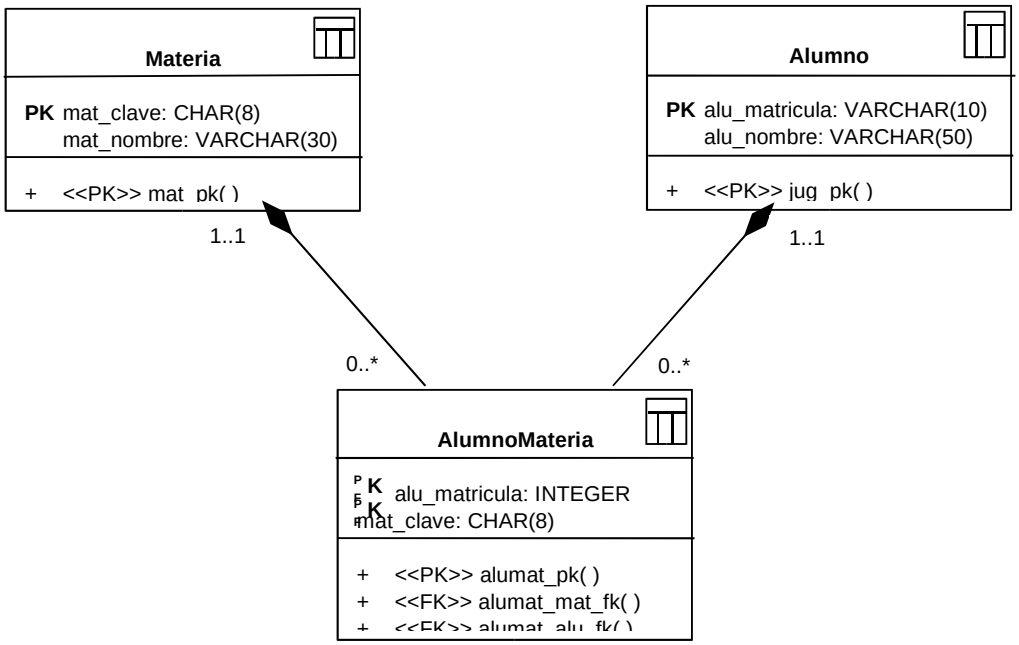
4.3.2. Diagrama de Objetos.



4.3.3. Frases.

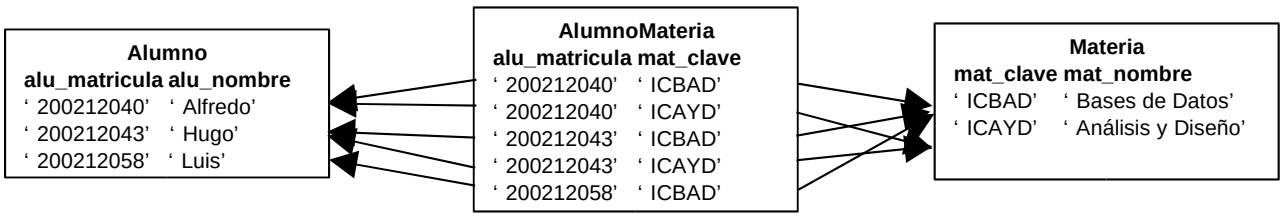
Un alumno cursa **varias** materias
Una materia es cursada por **varios** alumnos

4.3.4. Diseño de la Base de Datos.



Se añade una tabla intermedia que tiene por contiene las llaves primarias de las dos tablas originales. Ambos campos forman la llave primaria y cada uno de ellos es llave foránea hacia la tabla original. Nota que las relaciones está marcadas como agregaciones. En el caso de bases de datos vamos a poner el signo de agregación siempre que la llave primaria de una tabla pase como llave primaria y foránea a otra.

4.3.5. Ejemplo de Tablas.



4.3.6. Instrucciones para Crear Tablas en SQL.

```
CREATE TABLE Materia
(
    mat_clave CHAR(8) NOT NULL,
    mat_nombre VARCHAR(30) NOT NULL,

    CONSTRAINT mat_pk PRIMARY KEY (mat_clave)
);
```

```
CREATE TABLE Alumno
(
    alu_matricula VARCHAR(10) NOT NULL,
    alu_nombre VARCHAR(50) NOT NULL,

    CONSTRAINT alu_pk PRIMARY KEY (alu_matricula)
);

CREATE TABLE AlumnoMateria
(
    alu_matricula VARCHAR(10) NOT NULL,
    mat_clave CHAR(8) NOT NULL,

    CONSTRAINT alumat_pk PRIMARY KEY (alu_matricula, mat_clave)
);

ALTER TABLE AlumnoMateria
ADD CONSTRAINT alumat_mat_fk FOREIGN KEY (mat_clave)
      REFERENCES Materia (mat_clave);

ALTER TABLE AlumnoMateria
ADD CONSTRAINT alumat_alu_fk FOREIGN KEY (alu_matricula)
            REFERENCES Alumno (alu_matricula);
```

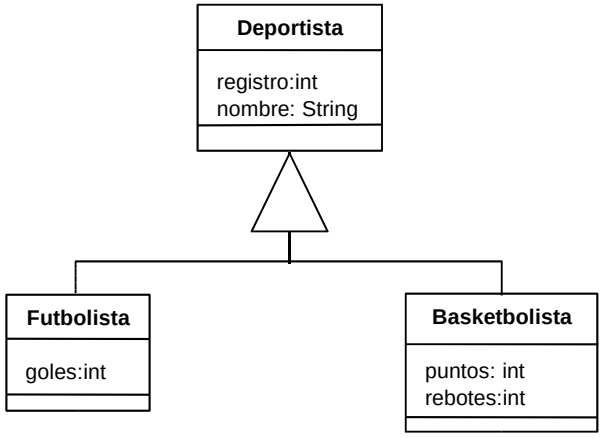
4.3.7. Instrucciones para Armar la Red en SQL.

```
// Se crean y se conectan los nodos
INSERT INTO Materia(mat_clave, mat_nombre)
    VALUES('ICBAD','Bases de Datos');
INSERT INTO Materia(mat_clave, mat_nombre)
    VALUES('ICAYD','Análisis y Diseño');

INSERT INTO Alumno(alu_matricula, alu_nombre)
    VALUES('200212040', 'Alfredo');
INSERT INTO Alumno(alu_matricula, alu_nombre)
    VALUES('200212043', 'Hugo');

INSERT INTO Alumno(alu_matricula, alu_nombre)
    VALUES('200212058', 'Luis');

INSERT INTO AlumnoMateria(alu_matricula, mat_clave)
    VALUES('200212040', 'ICBAD');
INSERT INTO AlumnoMateria(alu_matricula, mat_clave)
    VALUES('200212040', 'ICAYD');
INSERT INTO AlumnoMateria(alu_matricula, mat_clave)
    VALUES('200212043', 'ICBAD');
INSERT INTO AlumnoMateria(alu_matricula, mat_clave)
    VALUES('200212043', 'ICAYD');
INSERT INTO AlumnoMateria(alu_matricula, mat_clave)
    VALUES('200212058', 'ICBAD');
```

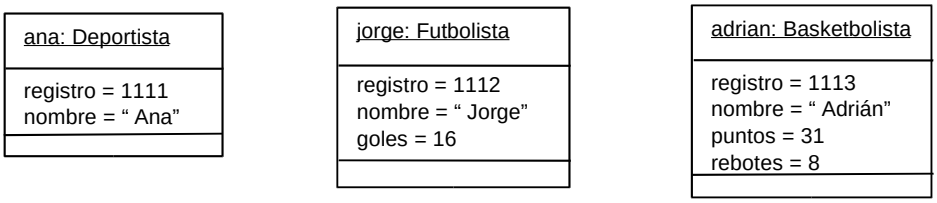
```
/* Consulta información */
SELECT *
FROM Alumno, AlumnoMateria, Materia
WHERE
    Alumno.alu_matricula = AlumnoMateria.alu_matricula
    AND AlumnoMateria.mat_clave = Materia.mat_clave
ORDER BY Materia.mat_clave, Alumno.alu_matricula;

/* Hugo se dá de baja de bases de datos */
DELETE FROM AlumnoMateria
WHERE alu_matricula = '200212043' AND mat_clave = 'ICBAD';

/* Consulta información */
SELECT *
FROM Alumno, AlumnoMateria, Materia
WHERE
    Alumno.alu_matricula = AlumnoMateria.alu_matricula
    AND AlumnoMateria.mat_clave = Materia.mat_clave
ORDER BY Materia.mat_clave, Alumno.alu_matricula;
```

4.4. Generalización.

4.4.1. Diagrama Conceptual(1).



4.4.2. Diagrama de Objetos.



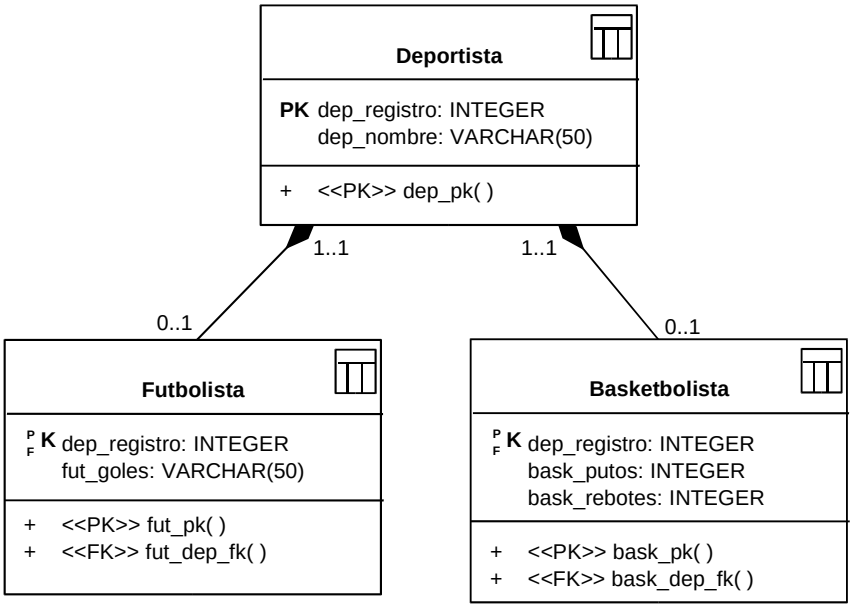
4.4.3. Frases.

Todo deportista **puede ser** futbolista o basketbolista.
Todo basketbolista **es un** deportista.
Todo futbolista **es un** deportista.

Otra forma de leer el diagrama es:

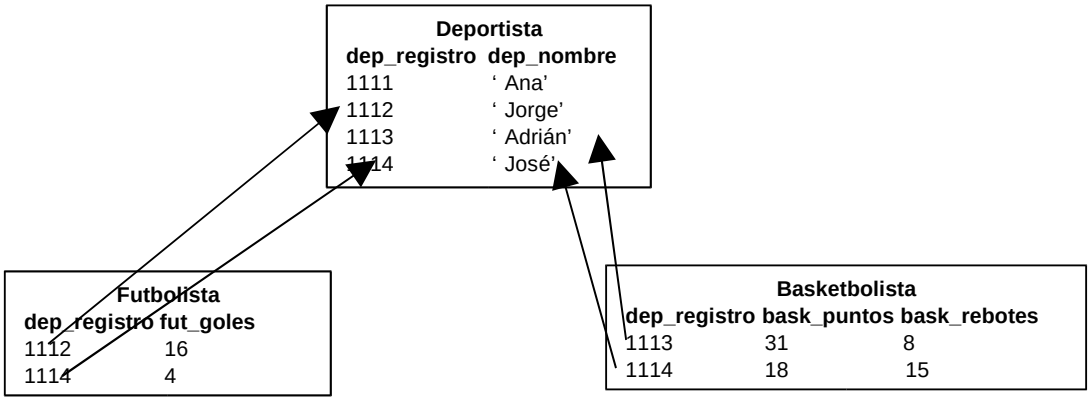
Un futbolista **es un tipo de** deportista.
Un basketbolista **es un tipo de** deportista.

4.4.4. Diseño de la Base de Datos(1).



La llave primaria del padre se coloca como llave primaria y foránea del hijo. Dado que la llave foránea representa la esencia de un objeto, es natural que se use como llave para almacenar la información de una misma persona, aunque esté en distintas tablas. Un modelado de este tipo cumple con la quinta forma normal.

4.4.5. Ejemplo de Tablas.



4.4.6. Instrucciones para Crear Tablas en SQL(1).

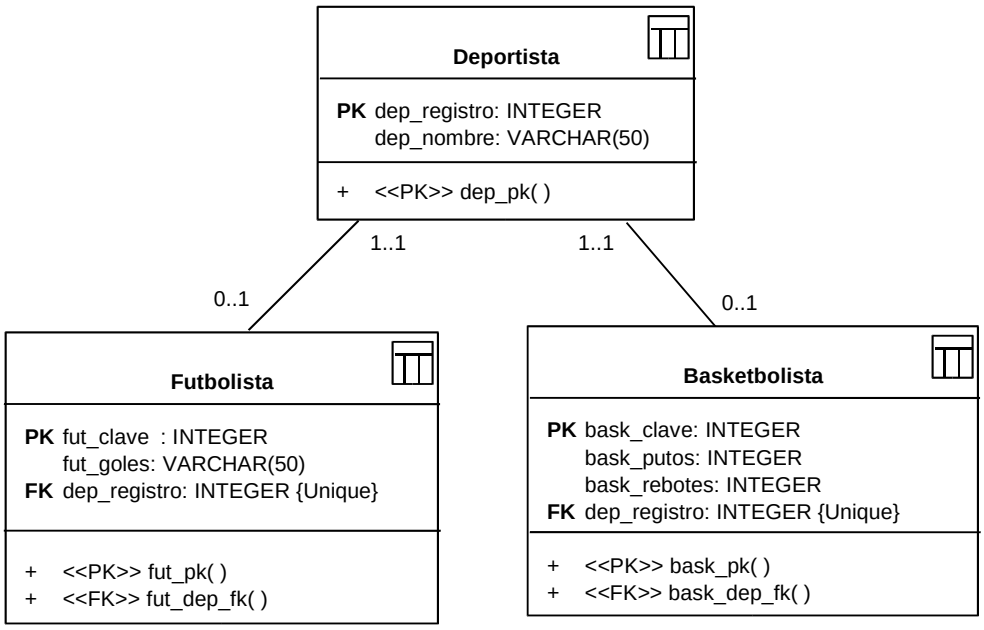
```
CREATE TABLE Deportista
(
    dep_registro INTEGER NOT NULL,
    dep_nombre VARCHAR(50) NOT NULL,

    CONSTRAINT dep_pk PRIMARY KEY (dep_registro)
);

CREATE TABLE Futbolista
(
    dep_registro INTEGER NOT NULL,
    fut_goles INTEGER NOT NULL,

    CONSTRAINT fut_pk PRIMARY KEY (dep_registro)
);

CREATE TABLE Basketbolista
(
    dep_registro INTEGER NOT NULL,
    bask_puntos INTEGER NOT NULL,
    bask_rebotes INTEGER NOT NULL,

    CONSTRAINT bask_pk PRIMARY KEY (dep_registro)
);

ALTER TABLE Futbolista
ADD CONSTRAINT fut_dep_fk FOREIGN KEY (dep_registro)
       REFERENCES Deportista(dep_registro);

ALTER TABLE Basketbolista
ADD CONSTRAINT bask_dep_fk FOREIGN KEY (dep_registro)
            REFERENCES Deportista(dep_registro);
```

4.4.7. Instrucciones para Armar la Red en SQL.

```
/* Se crean y se conectan los nodos */
INSERT INTO Deportista (dep_registro, dep_nombre) VALUES(1111, 'Ana');

INSERT INTO Deportista (dep_registro, dep_nombre) VALUES(1112, 'Jorge');
INSERT INTO Futbolista (dep_registro, fut_goles) VALUES(1112, 16);

INSERT INTO Deportista (dep_registro, dep_nombre) VALUES(1113, 'Adrián');
INSERT INTO Basketbolista (bask_puntos, bask_rebotes)
    VALUES(1113, 31, 8);

INSERT INTO Deportista (dep_registro, dep_nombre) VALUES(1114, 'José');
INSERT INTO Futbolista (dep_registro, fut_goles) VALUES(1114, 4);
INSERT INTO Basketbolista (bask_puntos, bask_rebotes)
    VALUES(1114, 18, 15);

/* Consulta información de deportistas */
```

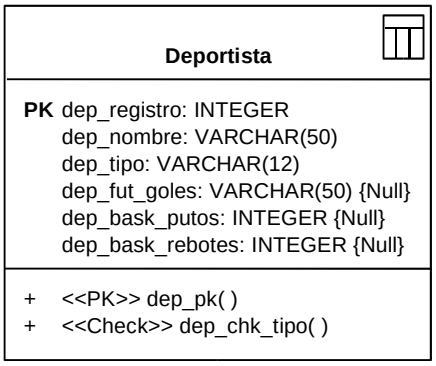
```
SELECT *
FROM Deportista;

/* Consulta información de futbolistas */
SELECT *
FROM Deportista, Futbolista
WHERE
    Deportista.dep_registro = Futbolista.dep_registro
ORDER BY Deportista.dep_registro;

/* Consulta información de basketbolistas */
SELECT *
FROM Deportista, Basketbolista
WHERE
    Deportista.dep_registro = Basketbolista.dep_registro
ORDER BY Deportista.dep_registro;
```

4.4.8. Diseño de la Base de Datos(2).



Aquí la clase hija añade la llave primaria de la clase padre como llave foránea. De hecho esta estrategia consiste en implementar una asociaciones uno a uno. Se utiliza con los hijos que de forma natural incorporan su propia llave primaria.

4.4.9. Diseño de la Base de Datos(3).

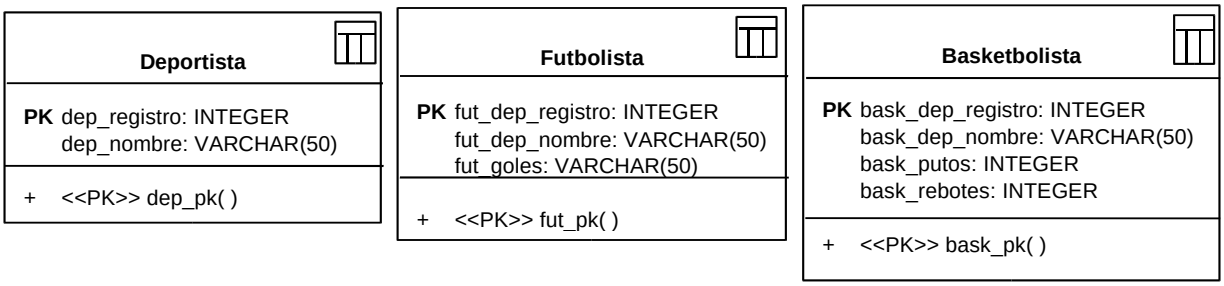


En este enfoque todas las clases se juntan en una sola. Los datos del padre son obligatorios y los datos de los hijo son opcionales. Se puede usar un campo que indique a que tipo corresponde el registro, sea deportista, futbolista o basketbolista. Este esquema se recomienda menos que los dos primeros.

4.4.10. Instrucciones para Crear Tablas en SQL(2).

```
CREATE TABLE Deportista
(
    dep_registro INTEGER NOT NULL,
    dep_nombre VARCHAR(50) NOT NULL,
    dep_tipo VARCHAR(20) NOT NULL,

    dep_fut_goles INTEGER,
    dep_bask_puntos INTEGER,
    dep_bask_rebotes INTEGER,

    CONSTRAINT dep_pk PRIMARY KEY (dep_registro)
    CONSTRAINT dep_chk_tipo
        CHECK(dep_tipo IN('deportista', 'futbolista', 'basketbolista'))
);
```

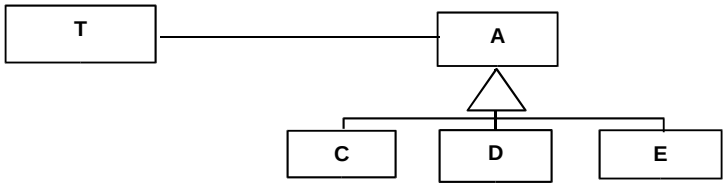
4.4.11. Diseño de la Base de Datos(4).



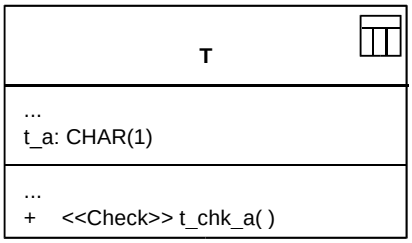
Este enfoque incorpora todos los atributos de los padres en los hijos. Debido esto mantiene aisladas a las tablas, pero es muy engorroso para darle mantenimiento, pues un cambio en la clase padre implica modificar también todas las tablas hijas.

4.4.12. Diagrama Conceptual(2).

En este caso se supone que las clases *A*, *C*, *D* y *E* no tienen atributos.



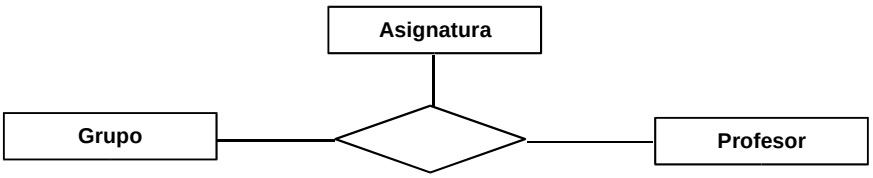
4.4.13. Diseño de la Base de Datos(5).



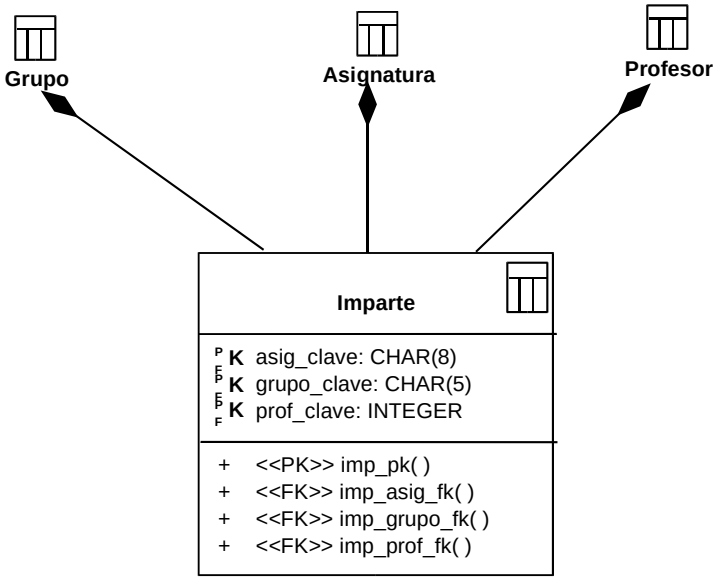
Se añade a *T* una campo adicional *a*. En la restricción *t_chk_a* se valida que los valores de *a* solo sean ‘A’, ‘C’, ‘D’ o ‘E’.

4.5. Clases n-arias.

4.5.1. Diagrama Conceptual.



4.5.2. Diseño de la Base de Datos.



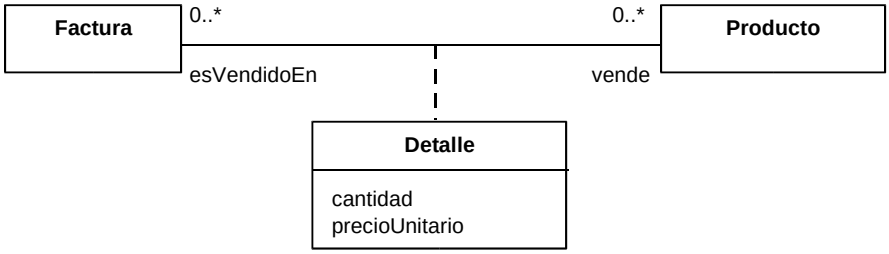
Se crea una tabla para la asociación n-aria, cuya llave primaria está formada por las llaves primarias de cada tabla que enlaza. Estas llaves son a su vez foráneas hacia su propia tabla.

4.5.3. Instrucciones para Crear Tablas en SQL.

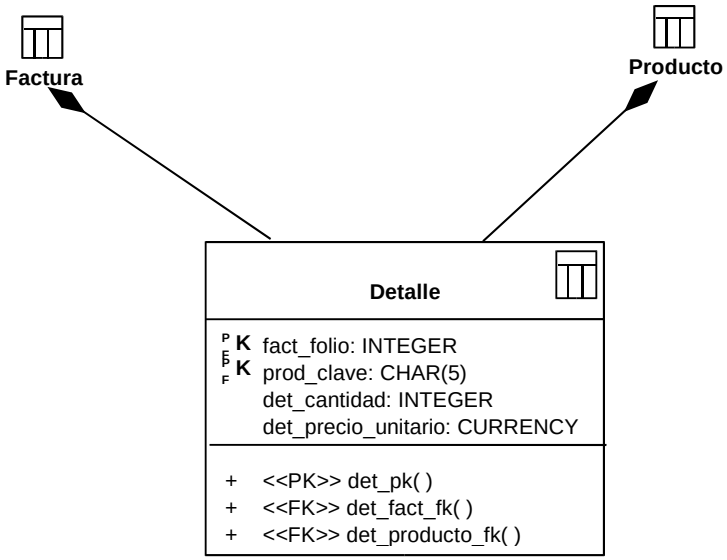
```
CREATE TABLE Imparte
(
    asig_clave CHAR(8) NOT NULL,
    grupo_clave CHAR(5) NOT NULL,
    prof_clave INTEGER NOT NULL,

    CONSTRAINT imp_pk PRIMARY KEY (asig_clave, grupo_clave, prof_clave)

);

ALTER TABLE Imparte
ADD CONSTRAINT imp_asig_fk FOREIGN KEY (asig_clave)
       REFERENCES Asignatura(asig_clave);
ALTER TABLE Imparte
ADD CONSTRAINT imp_grupo_fk FOREIGN KEY (grupo_clave)
       REFERENCES Grupo(grupo_clave);
ALTER TABLE Imparte
ADD CONSTRAINT imp_prof_fk FOREIGN KEY (prof_clave)
       REFERENCES Profesor(prof_clave);
```

4.6. Clases Asociativas.

4.6.1. Diagrama Conceptual(1).



4.6.2. Diseño de la Base de Datos(1).

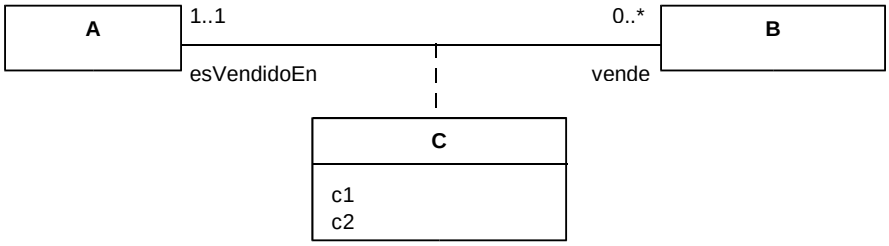


Se crea una tabla para la clase asociativa, cuya llave primaria está formada por las llaves primarias de cada tabla que enlaza. Estas llaves son a su vez foráneas hacia su propia tabla. Se añaden los atributos propios de la clase asociativa. Este enfoque se vale para cualquier tipo de asociación binaria o n-aria.

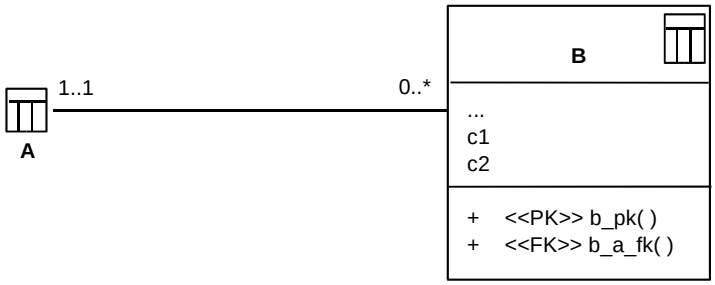
4.6.3. Instrucciones para Crear Tablas en SQL.

```
CREATE TABLE Detalle
(
    fact_folio INTEGER NOT NULL,
    prod_clave CHAR(5) NOT NULL,
    det_cantidad INTEGER NOT NULL,
    det_precio_unitario CURRENCY NOT NULL,

    CONSTRAINT det_pk PRIMARY KEY (fact_folio, prod_clave),
);


ALTER TABLE Detalle
ADD CONSTRAINT det_fact_fk FOREIGN KEY (fact_folio)
       REFERENCES Factura(fact_folio);
ALTER TABLE Detalle
ADD CONSTRAINT det_prod_fk FOREIGN KEY (prod_clave)
       REFERENCES Producto(prod_clave)
```

4.6.4. Diagrama Conceptual(2).



4.6.5. Diseño de la Base de Datos(2).



Si la relación es de **uno a muchos**, los atributos de la clase asociativa se ponen en la clase del lado muchos y el resto de los atributos se colocan como una relación normal. Este enfoque también se puede usar en una relación **uno a uno** y en ella los atributos de la clase asociativa se pueden colocar en cualquiera de las dos clases implicadas.