

TEMA 4
DISEÑO CONCEPTUAL Y LÓGICO DE UNA BASE DE DATOS



4.1. FORMATOS DE REPRESENTACIÓN.

- El siguiente paso de la metodología para realizar el diseño de la BD después de la definición de requerimientos es el diseño conceptual empleando para ello los conceptos del llamado Modelo Entidad relación E/R.
- Para poder iniciar con el diseño conceptual, es importante que en la etapa de análisis se hayan identificado todos los elementos del modelo de datos: Entidades, atributos, relaciones entre entidades, restricciones.
- Una vez que se ha desarrollado el diseño conceptual, se procede con la transformación del Modelo entidad relación ER en el llamado Modelo relacional empleado los conceptos vistos en el tema anterior. A este proceso se le conoce como Diseño lógico.
- En este capítulo se ilustra el uso de distintas notaciones empleadas para realizar el diseño tanto conceptual como lógico.
- Se revisan ambos modelos en paralelo debido a la correspondencia que existe entre cada uno de ellos, de una visión global y general que ofrece el modelo conceptual a un modelo particular, en este caso el relacional.

4.1.1. Principales formatos.

Para el diseño conceptual:

- ** Formato de Chen (Chen's Format) - Creador **Peter Pin-Shan Chen** en 1976.

Para el diseño lógico:

- Formato Relacional (Relational Format)
- ** Formato IE (International Engineering Format) / Formato de Martín (Martin's Format) / Modelo Pata de gallo (Crow's foot Format).
- ** Formato IDEF1X. Originalmente desarrollado por "The Computer System Laboratory of the National Institute of Standards and Technology" en diciembre del 1993.

** Estos 3 formatos son los que se emplearán en este capítulo.



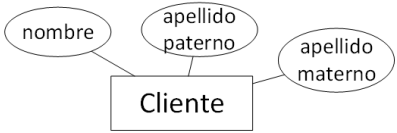
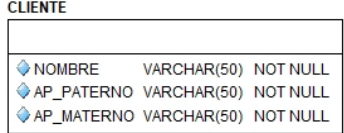
4.2. REPRESENTACIÓN DE ENTIDADES Y ATRIBUTOS.

4.2.1. Representación de entidades.

Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X
<p>Entidad</p> <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> <p>Cliente</p> </div>	<p>Tabla</p> <p>CLIENTE</p> <div style="border: 1px solid black; width: 100px; height: 40px; margin: 0 auto;"></div>

Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X
<ul style="list-style-type: none"> Se emplea un rectángulo con el nombre de la entidad en su interior iniciando en mayúsculas, sustantivo en singular Los nombres de las entidades pueden contener espacios, acentos, etc. 	<ul style="list-style-type: none"> En el modelo relacional, se emplea el concepto de tabla para representar a una entidad. El nombre se escribe arriba del rectángulo en mayúsculas. Si el nombre de la tabla es compuesto, se emplea “_” (guion bajo) para separar palabras. En el primer rectángulo se enlistan los atributos que formarán parte de la PK En el segundo rectángulo se enlistan los atributos que no forman parte de la PK.

4.2.2. Representación general de atributos.

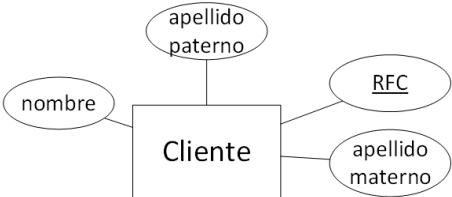
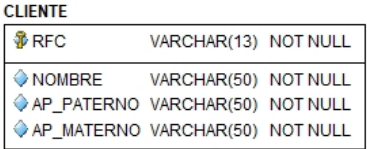
Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X
	
<ul style="list-style-type: none"> Los atributos se representan por óvalos. Los nombres pueden contener espacios 	<ul style="list-style-type: none"> Por claridad es recomendable agregar el tipo de dato de cada atributo y el constraint NULL/NOT NULL Para separar palabras compuestas se emplea guion bajo.

4.2.3. Clasificación de atributos.

- Clave principal y llave primaria natural.
- Clave candidata y llaves primaria candidata.
- Clave artificial y llave primaria artificial o subrogada
- Atributos obligatorios y opcionales
- Atributos simples y compuestos.
- Atributo de valores múltiples y de valor simple.
- Atributos derivados.



4.2.3.1. Clave principal y llave primaria natural

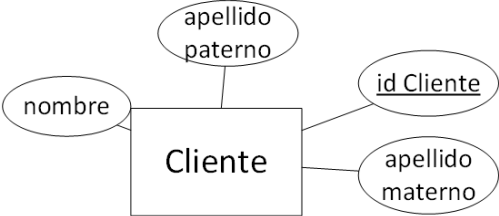












Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X
	
<ul style="list-style-type: none"> Una clave principal es un atributo cuyo valor es único para cada instancia de la entidad. Permite identificar de forma única a cada una de las instancias de la entidad. El atributo se subraya. 	<ul style="list-style-type: none"> En el modelo relacional, el concepto de clave principal corresponde con el concepto de llave primaria natural (PK) visto en el tema anterior. La PK aparece en el primer rectángulo, observar el ícono de una “Llave”.

Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X
<ul style="list-style-type: none"> La clave principal es seleccionada de la lista de los atributos 'naturales' de la entidad Todas las entidades deben contar con su clave principal. 	<ul style="list-style-type: none"> En la práctica, generalmente se reemplaza a la llave primaria natural por una llave primaria artificial por las razones vistas en el capítulo anterior.

4.2.3.2. Clave candidata y llave primaria candidata.

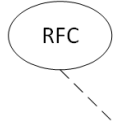









Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X																												
	<div>CLIENTE</div> <table><tr><td></td><td>NUM_CLIENTE</td><td>NUMERIC(10,0)</td><td>NOT NULL</td></tr><tr><td></td><td>RFC</td><td>VARCHAR(13)</td><td>NOT NULL</td></tr><tr><td></td><td>CURP</td><td>VARCHAR(18)</td><td>NOT NULL</td></tr><tr><td></td><td>NUM</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr><tr><td></td><td>NOMBRE</td><td>VARCHAR(50)</td><td>NOT NULL</td></tr><tr><td></td><td>AP_PATERNO</td><td>VARCHAR(50)</td><td>NOT NULL</td></tr><tr><td></td><td>AP_MATERNO</td><td>VARCHAR(50)</td><td>NOT NULL</td></tr></table>		NUM_CLIENTE	NUMERIC(10,0)	NOT NULL		RFC	VARCHAR(13)	NOT NULL		CURP	VARCHAR(18)	NOT NULL		NUM	VARCHAR(30)	NOT NULL		NOMBRE	VARCHAR(50)	NOT NULL		AP_PATERNO	VARCHAR(50)	NOT NULL		AP_MATERNO	VARCHAR(50)	NOT NULL
	NUM_CLIENTE	NUMERIC(10,0)	NOT NULL																										
	RFC	VARCHAR(13)	NOT NULL																										
	CURP	VARCHAR(18)	NOT NULL																										
	NUM	VARCHAR(30)	NOT NULL																										
	NOMBRE	VARCHAR(50)	NOT NULL																										
	AP_PATERNO	VARCHAR(50)	NOT NULL																										
	AP_MATERNO	VARCHAR(50)	NOT NULL																										
<ul style="list-style-type: none">Una entidad puede contener varias claves principales. En este caso se selecciona a uno de estos atributos y a los demás se les llama claves candidatas.Observar en uso de una X para identificar a las claves candidatas que no fueron seleccionadas como claves principales.	<ul style="list-style-type: none">En el modelo relacional el concepto de clave candidata corresponde con el concepto de llave primaria candidata.No existe una notación en particular para identificar a las PK candidatas.Puede asignarse una restricción de integridad UNIQUE.																												

4.2.3.3. Clave artificial

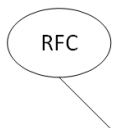









Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X												
	<p>CLIENTE</p> <table><tr><td> CLIENTE_ID</td><td>NUMERIC(10,0)</td><td>NOT NULL</td></tr><tr><td> NOMBRE</td><td>VARCHAR(50)</td><td>NOT NULL</td></tr><tr><td> AP_PATERNO</td><td>VARCHAR(50)</td><td>NOT NULL</td></tr><tr><td> AP_MATERNO</td><td>VARCHAR(50)</td><td>NOT NULL</td></tr></table>	 CLIENTE_ID	NUMERIC(10,0)	NOT NULL	 NOMBRE	VARCHAR(50)	NOT NULL	 AP_PATERNO	VARCHAR(50)	NOT NULL	 AP_MATERNO	VARCHAR(50)	NOT NULL
 CLIENTE_ID	NUMERIC(10,0)	NOT NULL											
 NOMBRE	VARCHAR(50)	NOT NULL											
 AP_PATERNO	VARCHAR(50)	NOT NULL											
 AP_MATERNO	VARCHAR(50)	NOT NULL											
<ul style="list-style-type: none">Solo para aquellos casos donde la clave principal no es tan evidente o no se especifica claramente el enunciado del problema se puede emplear una clave artificial.Por ejemplo, solo se hace mención de los atributos como son nombre, apellido paterno y apellido materno.De ser así es posible crear una clave principal empleando la notación <code>id <nombre entidad></code>.	<ul style="list-style-type: none">En el modelo relacional el concepto de clave artificial corresponde con el concepto de llave primaria artificial o subrogada. Revisada en el capítulo anterior. Recordar la notación: <code><nombre_tabla>_id</code>												

4.2.3.4. Atributos opcionales y obligatorios.

Atributo opcional: Atributo cuyos valores pueden ser nulos.








Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X									
	<p>CLIENTE</p> <table><tr><td> CLIENTE_ID</td><td>NUMERIC(10,0)</td><td>NOT NULL</td></tr><tr><td> RFC</td><td>VARCHAR(13)</td><td>NULL</td></tr><tr><td> CURP</td><td>VARCHAR(18)</td><td>NULL</td></tr></table>	 CLIENTE_ID	NUMERIC(10,0)	NOT NULL	 RFC	VARCHAR(13)	NULL	 CURP	VARCHAR(18)	NULL
 CLIENTE_ID	NUMERIC(10,0)	NOT NULL								
 RFC	VARCHAR(13)	NULL								
 CURP	VARCHAR(18)	NULL								
<ul style="list-style-type: none">La línea de conexión es punteada	<ul style="list-style-type: none">El atributo aparece con la palabra NULL.									

Atributo requerido: Atributo cuyos valores no pueden ser nulos.

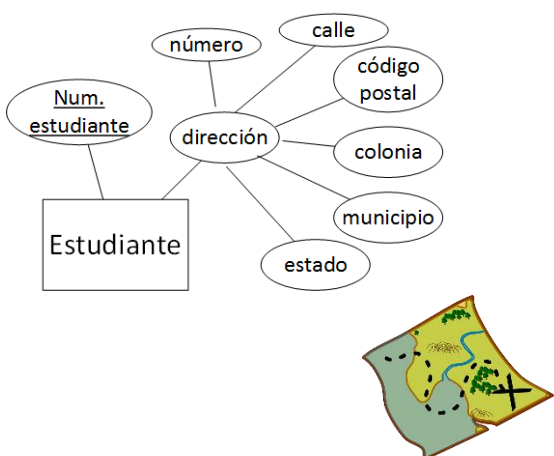
Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X									
	<p>CLIENTE</p> <table><tr><td> CLIENTE_ID</td><td>NUMERIC(10,0)</td><td>NOT NULL</td></tr><tr><td> RFC</td><td>VARCHAR(13)</td><td>NOT NULL</td></tr><tr><td> CURP</td><td>VARCHAR(18)</td><td>NOT NULL</td></tr></table>	 CLIENTE_ID	NUMERIC(10,0)	NOT NULL	 RFC	VARCHAR(13)	NOT NULL	 CURP	VARCHAR(18)	NOT NULL
 CLIENTE_ID	NUMERIC(10,0)	NOT NULL								
 RFC	VARCHAR(13)	NOT NULL								
 CURP	VARCHAR(18)	NOT NULL								
<ul style="list-style-type: none">La línea de conexión es continua.	<ul style="list-style-type: none">El atributo aparece con la palabra NOT NULL.									

4.2.3.5. Atributos simples y compuestos.

Atributo simple: Atributo que ya no puede ser dividido en sub atributos

Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X						
	<div>CLIENTE</div> <table><tr><td> CLIENTE_ID</td><td>NUMERIC(10,0)</td><td>NOT NULL</td></tr><tr><td> COLONIA</td><td>VARCHAR(100)</td><td>NOT NULL</td></tr></table>	 CLIENTE_ID	NUMERIC(10,0)	NOT NULL	 COLONIA	VARCHAR(100)	NOT NULL
 CLIENTE_ID	NUMERIC(10,0)	NOT NULL					
 COLONIA	VARCHAR(100)	NOT NULL					
<ul style="list-style-type: none">En el ejemplo, la colonia ya no puede dividirse en sub atributos.No existe notación el particular para este tipo de atributos en ambos diseños.							












Atributo compuesto: Es aquel que puede ser descompuesto en al menos 2 sub atributos relevantes para las reglas de negocio o caso de estudio.

Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X																																													
	<p>Opción 1:</p> <p>ESTUDIANTE</p> <table><tr><td>NUM_ESTUDIANTE</td><td>NUMERIC(10,0)</td><td>NOT NULL</td></tr><tr><td>NOMBRE</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr><tr><td>APELLIDO_PATERNO</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr><tr><td>APELLIDO_MATERNO</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr><tr><td>DIRECCION</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr></table> <p>Opción 2:</p> <p>ESTUDIANTE</p> <table><tr><td>NUM_ESTUDIANTE</td><td>NUMERIC(10,0)</td><td>NOT NULL</td></tr><tr><td>NOMBRE</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr><tr><td>APELLIDO_PATERNO</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr><tr><td>APELLIDO_MATERNO</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr><tr><td>CALLE</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr><tr><td>NUM_EXTERIOR</td><td>NUMERIC(5,0)</td><td>NOT NULL</td></tr><tr><td>COLONIA</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr><tr><td>CODIGO_POSTAL</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr><tr><td>MUNICIPIO_DELEGACION</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr><tr><td>ESTADO</td><td>VARCHAR(30)</td><td>NOT NULL</td></tr></table>	NUM_ESTUDIANTE	NUMERIC(10,0)	NOT NULL	NOMBRE	VARCHAR(30)	NOT NULL	APELLIDO_PATERNO	VARCHAR(30)	NOT NULL	APELLIDO_MATERNO	VARCHAR(30)	NOT NULL	DIRECCION	VARCHAR(30)	NOT NULL	NUM_ESTUDIANTE	NUMERIC(10,0)	NOT NULL	NOMBRE	VARCHAR(30)	NOT NULL	APELLIDO_PATERNO	VARCHAR(30)	NOT NULL	APELLIDO_MATERNO	VARCHAR(30)	NOT NULL	CALLE	VARCHAR(30)	NOT NULL	NUM_EXTERIOR	NUMERIC(5,0)	NOT NULL	COLONIA	VARCHAR(30)	NOT NULL	CODIGO_POSTAL	VARCHAR(30)	NOT NULL	MUNICIPIO_DELEGACION	VARCHAR(30)	NOT NULL	ESTADO	VARCHAR(30)	NOT NULL
NUM_ESTUDIANTE	NUMERIC(10,0)	NOT NULL																																												
NOMBRE	VARCHAR(30)	NOT NULL																																												
APELLIDO_PATERNO	VARCHAR(30)	NOT NULL																																												
APELLIDO_MATERNO	VARCHAR(30)	NOT NULL																																												
DIRECCION	VARCHAR(30)	NOT NULL																																												
NUM_ESTUDIANTE	NUMERIC(10,0)	NOT NULL																																												
NOMBRE	VARCHAR(30)	NOT NULL																																												
APELLIDO_PATERNO	VARCHAR(30)	NOT NULL																																												
APELLIDO_MATERNO	VARCHAR(30)	NOT NULL																																												
CALLE	VARCHAR(30)	NOT NULL																																												
NUM_EXTERIOR	NUMERIC(5,0)	NOT NULL																																												
COLONIA	VARCHAR(30)	NOT NULL																																												
CODIGO_POSTAL	VARCHAR(30)	NOT NULL																																												
MUNICIPIO_DELEGACION	VARCHAR(30)	NOT NULL																																												
ESTADO	VARCHAR(30)	NOT NULL																																												

Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X
<ul style="list-style-type: none"> Observar en el ejemplo, los sub atributos se modelan como sub atributos del atributo compuesto. Es decir, la dirección se puede descomponer en calle, número, colonia, etc. 	<ul style="list-style-type: none"> Se tienen 2 opciones: Opción 1: Especificar solo el atributo compuesto Opción 2: Eliminar el atributo compuesto y especificar solo los atributos simples. La decisión depende de las reglas de negocio, lo que resulte más adecuado para la solución. <p><u>Ejemplo:</u> Si se requiere procesar o consultar atributos simples por separado, lo adecuado es la opción 2.</p>

4.2.3.6. Atributos derivados.

- Es un atributo cuyo valor es generado a partir de algún cálculo o algoritmo empleando los valores de otros campos de la tabla.
- El valor de un atributo derivado siempre se puede calcular a partir de los valores de los otros campos.

Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X									
	<div><div>ESTUDIANTE</div><table><tr><td> ESTUDIANTE_ID</td><td>NUMERIC(10,0)</td><td>NOT NULL</td></tr><tr><td> FECHA_NACIMIENTO</td><td>DATE</td><td>NOT NULL</td></tr><tr><td> EDAD</td><td>NUMERIC(3,0)</td><td>NOT NULL</td></tr></table></div> 	 ESTUDIANTE_ID	NUMERIC(10,0)	NOT NULL	 FECHA_NACIMIENTO	DATE	NOT NULL	 EDAD	NUMERIC(3,0)	NOT NULL
 ESTUDIANTE_ID	NUMERIC(10,0)	NOT NULL								
 FECHA_NACIMIENTO	DATE	NOT NULL								
 EDAD	NUMERIC(3,0)	NOT NULL								
<ul style="list-style-type: none">En el ejemplo, el campo edad es derivado ya que su valor puede ser calculado a partir de la fecha de nacimiento.Se representa por un óvalo punteado.	<ul style="list-style-type: none">No existe representación en particular. Si se decide conservar al atributo derivado, aparecerá como un atributo más.									

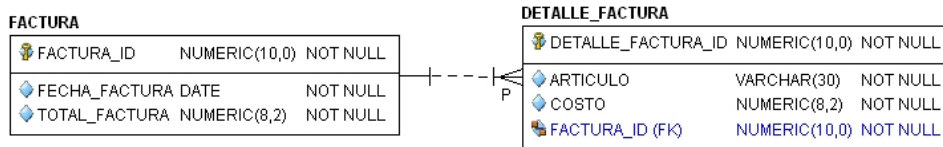
Ventajas de conservar el atributo derivado:

- Ahorro del cálculo requerido para obtener su valor, en especial si este se considera como costoso o complejo.
- Útil si el cálculo es costoso o complejo.

Desventajas de conservar el atributo derivado:

- En sentido estricto, un campo derivado puede considerarse como redundante.
- Al ser redundante puede causar inconsistencias. ¿Qué sucede si se actualiza la fecha de nacimiento, y no se actualiza la edad?
- Posible impacto en almacenamiento. Se requiere espacio para almacenar los valores del atributo derivado.

Algunos manejadores ofrecen algunas opciones para eliminar las desventajas de un atributo derivado Oracle ofrece el concepto de **columna virtual**. Esta idea permite eliminar los problemas de posibles inconsistencias y posibles impactos en el almacenamiento. (Ver la parte de SQL para mayores detalles).

Ejemplo:

El campo `total_factura` es derivado, ya que en la tabla `detalle_factura` se guarda el costo de cada artículo que contiene la factura. Su valor se obtendría sumando los costos de cada artículo.

4.2.3.7. Atributo de valores múltiples y de valor simple.

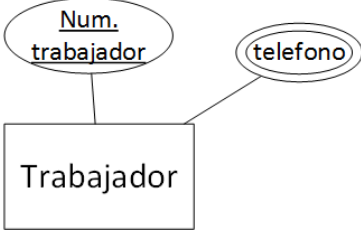

Atributo de valor simple.

- Es un atributo que solo puede tener un único valor por cada registro o instancia de una entidad.
- Observar que un atributo simple no es necesariamente un atributo de valor simple.

Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X						
<p>Atributo de valor simple</p> <pre>graph LR; E([Estudiante]) --- A1([<u>Num. estudiante</u>]); E --- A2([fecha nacimiento]);</pre>	<p>Atributo de valor simple</p> <table><caption>ESTUDIANTE</caption><tr><td> ESTUDIANTE_ID</td><td>NUMERIC(10,0)</td><td>NOT NULL</td></tr><tr><td> FECHA_NACIMIENTO</td><td>DATE</td><td>NOT NULL</td></tr></table>	ESTUDIANTE_ID	NUMERIC(10,0)	NOT NULL	FECHA_NACIMIENTO	DATE	NOT NULL
ESTUDIANTE_ID	NUMERIC(10,0)	NOT NULL					
FECHA_NACIMIENTO	DATE	NOT NULL					
<ul style="list-style-type: none">• En el ejemplo, la fecha de nacimiento de un estudiante es un atributo de valor simple, ya que por cada estudiante (instancia) solo se tiene un valor para el campo fecha de nacimiento.• Visto de otra forma: el estudiante tiene una sola fecha de nacimiento.• No existe notación en particular para distinguir a un atributo de valor simple en ambos diseños.							

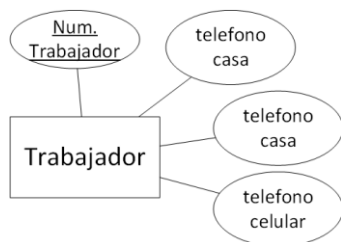
Atributo de valores múltiples.

- Llamado también atributo multi-valorado.
- Es un atributo que puede tener más de un valor para cada registro de la tabla
- En el ejemplo: Un trabajador puede tener varios teléfonos: teléfono de casa, celular, teléfono de oficina, etc.
- En este caso, cada registro o instancia de `Trabajador` puede tener varios valores para el campo `teléfono`. Se trata de un atributo multi-valor.

Diseño conceptual	Diseño lógico Crow's Foot e IDEF1X
<p>Atributo multivalor</p> 	<p>Atributo multivalor</p> <ul style="list-style-type: none"> En el modelo relacional un atributo no puede tener múltiples valores para un mismo registro. Para resolver este caso se emplean estrategias alternas que se muestran a continuación, sin embargo, algunos manejadores permiten la definición de colecciones como tipo de dato y así poder asociar una lista de valores a un registro. 
<ul style="list-style-type: none"> Observar que un atributo multi – valor se distingue por un doble ovalo. 	

Estrategia 1.

Agregar un nuevo campo para cada tipo de teléfono. ¿Qué ventajas y desventajas presenta?



TRABAJADOR

TRABAJADOR_ID	NUMERIC(10,0)	NOT NULL
NOMBRE	VARCHAR(30)	NOT NULL
APELLIDO_PATERO	VARCHAR(30)	NOT NULL
APELLIDO_MATERNO	VARCHAR(30)	NOT NULL
TELEFONO_CASA	VARCHAR(30)	NULL
TELEFONO_CELULAR	VARCHAR(30)	NULL
TELEFONO_OFICINA	VARCHAR(30)	NULL

Desventajas:

- ¿Qué pasa si el trabajador agrega un nuevo teléfono, por ejemplo, telefono_depto?, Se tendría que modificar la estructura de la tabla.
- Observar que los 3 campos de teléfono se tienen que definir como nulos ya que no todos los trabajadores cuentan con los mismos tipos de teléfono.

Ventajas:

- Los 3 campos están en la misma tabla lo que permite una explotación y /o recuperación de los 3 valores de forma directa.

Estrategia 2.

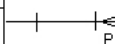
Agregar una tabla que permita almacenar un registro por cada tipo de teléfono. ¿Qué ventajas y desventajas presenta?

TRABAJADOR

TRABAJADOR_ID	NUMERIC(10,0)	NOT NULL
NOMBRE	VARCHAR(30)	NOT NULL
APELLIDO_PATERO	VARCHAR(30)	NOT NULL
APELLIDO_MATERNO	VARCHAR(30)	NOT NULL

TELEFONO_TRABAJADOR

TIPO	VARCHAR(30)	NOT NULL
TRABAJADOR_ID (FK)	NUMERIC(10,0)	NOT NULL
NUMERO	VARCHAR(30)	NOT NULL



Ventajas:

- El modelo puede guardar N números telefónicos, no solo 3. Observar el uso de una nueva tabla formado por una PK compuesta. Cada trabajador puede tener varios teléfonos, clasificados por tipo.
- Para cada trabajador y para cada tipo se almacena el número telefónico.
- El número telefónico ya no tiene que definirse como null.

- ¿Qué pasa si un trabajador no cuenta con algún número telefónico?: Simplemente, no existirán registros en la tabla `telefono_trabajador` para un trabajador en particular.
- Los datos se verían así:

TRABAJADOR_ID	NOMBRE	APELLIDO_PATERNO	APELLIDO_MATERNO
1	JUAN	LOPEZ	LARA
2	PABLO	LUNA	MONTES
3	MARIA	LARA	MARISCAL

TIPO	TRABAJADOR_ID	NUMERO
CASA	1	55909023902
CELULAR	1	55029309233
OFICINA	1	33892898323
CASA	2	55902930212
CELULAR	2	55902899323

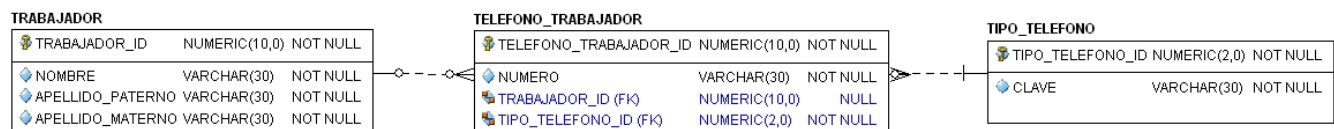
Desventajas:

- La recuperación de la información acerca de los teléfonos de un trabajador implica hacer una operación y lectura extra con la tabla nueva, es decir, se requiere hacer una operación “join” entre ambas tablas para recuperar los datos. Esto pudiera generar problemas de desempeño sobre todo con tablas con una gran cantidad de registros. Sin embargo, este detalle en términos generales se prefiere a la estrategia anterior.
- Observar que la PK es compuesta y está formada por una cadena de texto. ¿Qué pasa si en lugar de guardar la palabra CELULAR, se guarda “CEL”? Este problema se resuelve con la siguiente estrategia:

Estrategia 3.

Optimización con una PK artificial y creando catálogo de tipos de teléfonos.

- Ahora tenemos 2 entidades: `trabajador` y `tipo_telefono`. Por lo anterior, podemos escribir lo siguiente:
- Un trabajador puede tener varios tipos de teléfonos.
- Un tipo de teléfono puede ser asociado con varios trabajadores



Los datos se verían así:

TRABAJADOR

TRABAJADOR_ID	NOMBRE	APELLIDO_PATERNO	APELLIDO_MATERNO
1	JUAN	LOPEZ	LARA
2	PABLO	LUNA	MONTES
3	MARIA	LARA	MARISCAL

TELEFONO_TRABAJADOR

TELEFONO_TRABAJADOR_ID	TRABAJADOR_ID	TIPO_TELEFONO_ID	NUMERO
1	1	1	55909023902
2	1	2	55029309233
3	1	3	33892898323
4	2	1	55902930212
5	2	2	55902899323

TIPO_TELEFONO

TIPO_TELEFONO_ID	NOMBRE
1	CASA
2	CELULAR
3	OFICINA

- Observar, se agrega una PK artificial para mejorar el desempeño al momento de recuperar los datos.
- En esta última estrategia, se tiene el mejor nivel de integridad, pero la explotación se ve afectada.

Ejercicio en clase 1

Para el siguiente enunciado realizar:

- Propuesta de diseño conceptual empleando un modelo ER
- Propuesta de un diseño lógico empleando un modelo relacional.



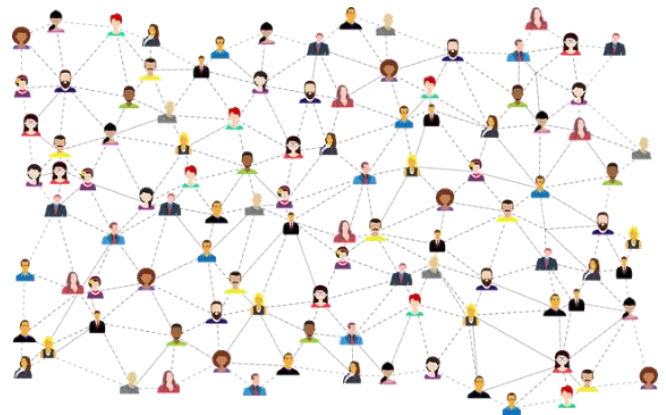
Se desea construir una pequeña base de datos para almacenar el registro de solicitudes de profesores candidatos para ocupar las vacantes de un colegio privado. Como parte inicial del proceso se requiere almacenar nombre y apellidos del candidato, número de cédula profesional, descripción de su título, año de titulación. Para establecer una comunicación segura, a cada aspirante se le permite registrar hasta 3 correos electrónicos.

Se requiere registrar la lista de asignaturas que el profesor puede o desea impartir (álgebra, cálculo, etc.). Finalmente, en caso que el aspirante haya trabajado anteriormente en el colegio, se registran los números de contrato (folios de 5 dígitos) realizados en el pasado.

4.3. REPRESENTACIÓN DE RELACIONES.

Resumen:

- Representación general.
- Niveles de dependencia (débil y fuerte).
- Cardinalidad
- Dependencia de existencia.
- Participación de una entidad en una relación.
- Entidades débiles.
- Grado de una relación



4.3.1. Representación general de relaciones.

Diseño conceptual.

- Asociación o correspondencia existente entre entidades.
- Representación a través de un rombo con el nombre de la relación (verbo en 3ª persona).
- El nombre que se especifica en el rombo debe permitir leer la relación en 2 posibles sentidos dependiendo la forma en la que ambas entidades están organizadas en el diagrama.
 - De izquierda hacia la derecha
 - De arriba hacia abajo

Ejemplo:



4.3.1.1. Papel o rol

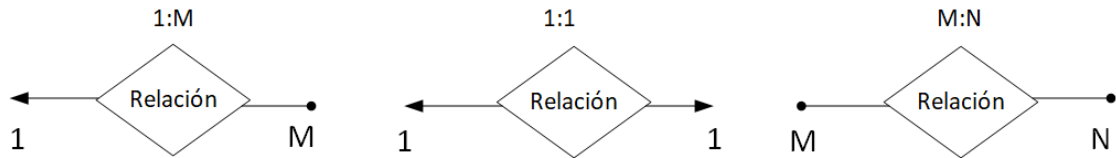
- Es la función que realiza cada tipo de entidad en una relación
- Se representa con un pequeño texto que se coloca sobre la línea que une a las entidades.

- En la práctica es poco común agregar el rol.



- En este caso, la relación se lee de izquierda a derecha: Un empleado “Administra” una agencia.

4.3.1.2. Representación de tipos de relaciones.



- En una relación 1:M la flecha apunta a la tabla padre
- En una relación M:N se especifican 2 puntos finales en lugar de flechas.
- En una relación 1:1 Se especifican ambas flechas.

Ejemplos:

Relación	Representación diseño conceptual
Sucursal y Ciudad (1:M)	<pre> graph LR Sucursal[Sucursal] -- pertenece --> Ciudad[Ciudad] </pre>
Factura y Renta (1:1)	<pre> graph LR Renta[Renta] -- genera --> Factura[Factura] </pre>
Sucursal y Empleado (Agente) (M:N)	<pre> graph LR Empleado[Empleado] -- Agente de Ventas -- pertenece --> Sucursal[Sucursal] </pre>

Diseño lógico.

Recordando del tema anterior, para relacionar 2 entidades se emplea el concepto de llave foránea (FK). Para asociar a las 2 entidades es necesario conocer el tipo de relación:

- Relación uno a uno (1:1)
- Relación uno a muchos (1:M)
- Relación muchos a muchos (M:N)

El siguiente paso es identificar la tabla que contendrá a la FK. Este proceso se realiza en la siguiente sección.

4.3.2. Niveles de dependencia.

El nivel de dependencia indica que tan fuerte es la relación entre 2 entidades. Existen 2 niveles representados por 2 tipos de relaciones:

- Relaciones suaves, débiles o no identificativas. -----
- Relaciones fuertes, duras o identificativas. _____

4.3.2.1. Relaciones no identificativas, suaves o débiles.

- Consiste en relacionar 2 entidades a través de una **línea punteada**. Normalmente se emplea para representar relaciones 1:M aunque también puede emplearse para representar relaciones 1:1, y en algunos casos, relaciones M:N.
- Al asociar 2 tablas con una relación no identificativa (línea punteada) la llave primaria (PK) de la tabla padre pasa como un campo más de la tabla hija como llave foránea (FK).

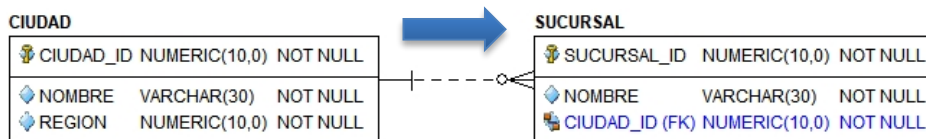
Ejemplos:

Considerar las parejas de entidades que se revisaron en temas anteriores.

i. Sucursal y Ciudad (1:M)

- **Una** sucursal se ubica en **una** ciudad. (1:1)
- En **una** ciudad pueden existir **varias** sucursales. (1:M)

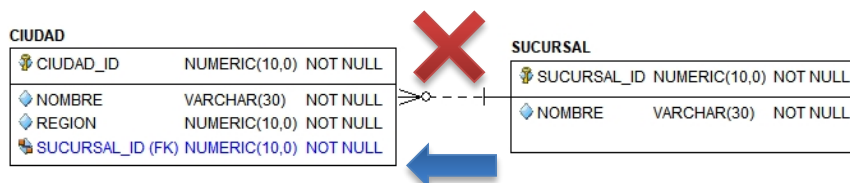
En una relación 1:M, la FK debe quedar del lado “Muchos”. Observando el ejemplo, la FK debe quedar del lado de la entidad Sucursal.



Para verificar lo anterior, basta con imaginar los datos:

- Cada registro de *sucursal*, se asocia con un registro de *ciudad*.
- Cada registro de *ciudad* puede estar asociado con varios registros de *sucursal*.

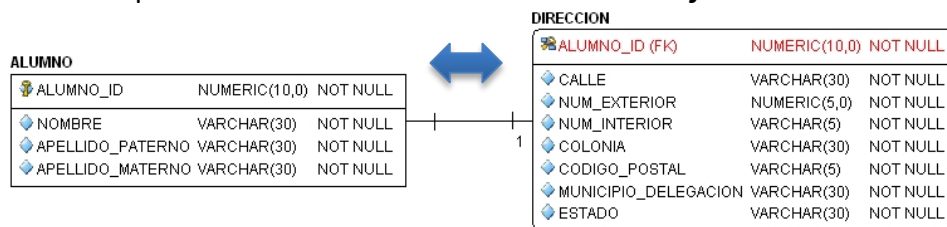
¿Qué sucedería si la FK estuviera del lado de la tabla *ciudad*?



- Como se puede observar, a cada registro de *ciudad* solo se le puede asociar un registro de *sucursal*, pero, ¿Qué sucede si 2 sucursales están en la misma ciudad?
- Esta condición puede ocurrir, pero con la FK de esta forma, no es posible registrar los datos correspondientes.

4.3.2.2. Relaciones identificativas, fuertes o duras.

- Consiste en relacionar 2 entidades a través de una *línea continua*. Se emplea para representar relaciones 1:1 y para modelar relaciones M:N
- Al asociar 2 tablas con una relación identificativa (línea continua) la llave primaria (PK) de la tabla padre pasa como llave primaria (PK) y también como llave foránea (FK) en la tabla hija.
- La diferencia con la relación no identificativa es que la (FK) forma parte de la PK de la tabla hija.
- Si la tabla hija no tiene una PK propia, la PK de la tabla padre y de la tabla hija es **compartida**.
- Por lo anterior y por la definición de PK, la relación entre ambas tablas es 1:1, un registro de la tabla padre, **solo** puede asociarse con un registro de la tabla hija.
- Al compartir la PK con la tabla hija, está también **identificará** a cada registro de la tabla hija. Por tal razón a este tipo de relaciones se les conoce como **identificativas**.



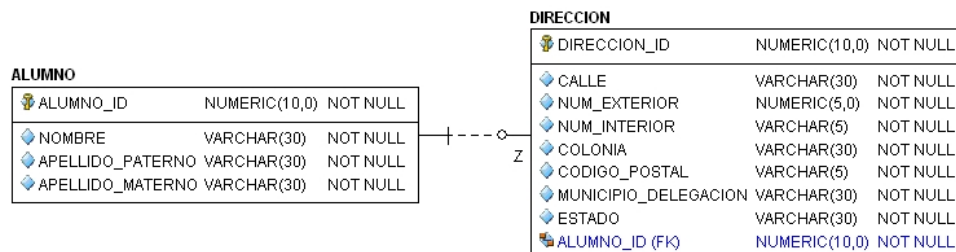
Ejemplos:

i. Alumno y Dirección (1:1)

- **Un** alumno tiene **una** dirección. (1:1)
- **Una** dirección le pertenece a **un** alumno. (1:1)
- Observar que la PK se comparte. Para cada registro de `alumno` le corresponde un registro de `direccion`. No es posible, por ejemplo, registrar 2 direcciones asociadas al mismo alumno. Por tal razón, este tipo de relaciones se emplea para representar relaciones 1:1
- Observar que la PK de `direccion` también es FK.
- ¿Qué pasa si invertimos la PK?, es decir, ahora la PK es `direccion_id` y es PK y FK en `alumno`. En realidad, no pasa nada, es equivalente. La relación sigue siendo 1:1
- Observar que en el diseño lógico la tabla "hija" que recibe la PK tiene las **esquinas redondeadas**.

Estrategia 2:

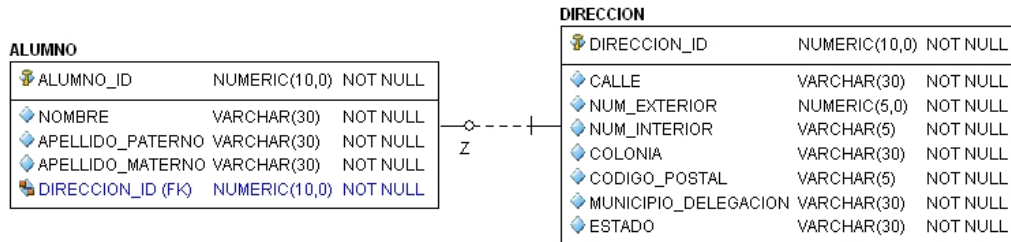
- El hecho de que la tabla `direccion` tenga una PK y FK `alumno_id` pareciera verse un poco rara. Lo común es ver su propio identificador: `direccion_id`. Para respetar este estilo, se puede aplicar la siguiente variante:



- En sentido estricto, la relación anterior es un 1:M, al estar la PK en `direccion`, podríamos tener que el campo `alumno_id` se repitiera ya que este ya no es PK: Un alumno puede tener varias direcciones.

direccion_id(pk)	alumno_id (fk)
1	1001
2	1001
3	1002
4	1002

- Para respetar la relación 1:1, se puede agregar un índice tipo `unique` a la FK, y con esta técnica el diseño es equivalente al anterior.
- ¿Qué pasa si la FK la escribimos del lado de alumno?



- Ahora podríamos decir que el campo `direccion_id` se puede duplicar, es decir, en una dirección pueden vivir varios alumnos.

alumno_id (pk)	direccion_id (fk)
1001	1
1002	1
1003	2
1004	2

- Si se desea respetar la relación 1:1, podemos emplear la misma técnica, agregar un índice tipo `unique` a la FK, y, por lo tanto, la FK puede quedar en **cualquiera** de las 2 tablas, esto, *siempre y cuando se trate de una relación 1:1*

Estrategia 3:

- Otra solución es eliminar la tabla `direccion` y agregar todos sus campos a `alumno`. Esto es válido al tratarse de una relación 1:1. En realidad se decide separar en 2 tablas, cuando se desee tener una clara definición y separación de entidades. Si para el caso de estudio es relevante manejar ambas entidades por separado, se recomienda separarlas, de otra forma pueden fusionarse.

ALUMNO

ALUMNO_ID	NUMERIC(10,0)	NOT NULL
NOMBRE	VARCHAR(30)	NOT NULL
APELLIDO_PATERNO	VARCHAR(30)	NOT NULL
APELLIDO_MATERNO	VARCHAR(30)	NOT NULL
CALLE	VARCHAR(30)	NOT NULL
NUM_EXTERIOR	NUMERIC(5,0)	NOT NULL
NUM_INTERIOR	VARCHAR(5)	NOT NULL
COLONIA	VARCHAR(30)	NOT NULL
CODIGO_POSTAL	VARCHAR(30)	NOT NULL
MUNICIPIO_DELEGACION	VARCHAR(30)	NOT NULL
ESTADO	VARCHAR(30)	NOT NULL

Ejercicio en clase 2

Considerar las siguientes entidades:

Sucursal y Gerente (1:1)

- **Una** sucursal es administrada por **un** gerente. (1:1)
- **Un** gerente solo puede administrar **una** sucursal. (1:1)

- ¿Cuál de las 3 estrategias anteriores podría implementar esta relación?
- Construir el modelo relacional correspondiente.



4.3.3. Cardinalidad.

- La cardinalidad expresa el número mínimo y máximo de ocurrencias de una entidad (instancias o registros) asociados con una ocurrencia (instancia o registro) de otra entidad.
- Se emplea el formato (X,Y) y se escribe a un costado de cada entidad.
 - X: valor mínimo
 - Y: valor máximo. Si el valor máximo no se conoce con exactitud se escribe "*" (muchos).
- Los usos de estos valores son de mayor relevancia para la aplicación. La cardinalidad no se implementa en la base de datos, pero es importante que se exprese en el modelo. A nivel de desarrollo de la aplicación es útil contar con esta información.

Ejemplo:

- Un** profesor imparte como máximo **4** cursos.
- Un** curso es impartido por **un** profesor.

Para cada entidad se realizan las siguientes preguntas:

Entidad **profesor**:

- ¿Cuántas instancias de la entidad **curso** (min y máx.) se asocian con una instancia de **profesor**?
Respuesta: **(1,4)**
 - Un profesor imparte como mínimo 1 curso, máximo 4.*
- El resultado se escribe del lado de la entidad contraria, en este caso al lado de la entidad **curso**.

Entidad **curso**:

- ¿Cuántas instancias de la entidad **profesor** (min y máx.) se asocian con una instancia de **CURSO**?
Respuesta: **(1,1)**.
 - Un curso es impartido mínimo por un profesor, máximo por 1.*
- El resultado se escribe del lado de la entidad contraria, en este caso al lado de la entidad **profesor**.

Los datos se verían así:

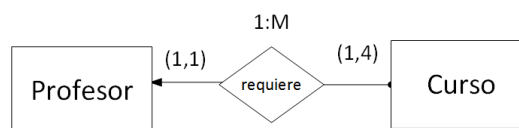
CURSO

CURSO_ID	NOMBRE	PROFESOR ID
1	ALGEBRA	1
2	CALCULO	1
3	GEOMETRIA	1
4	ESTADISTICA	3
5	ESTATICA	2

PROFESOR	
PROFESOR_ID	NOMBRE
1	JUAN
2	LORENA
3	MARCOS

Observar que para determinar de forma correcta la cardinalidad, es importante conocer las reglas de negocio.

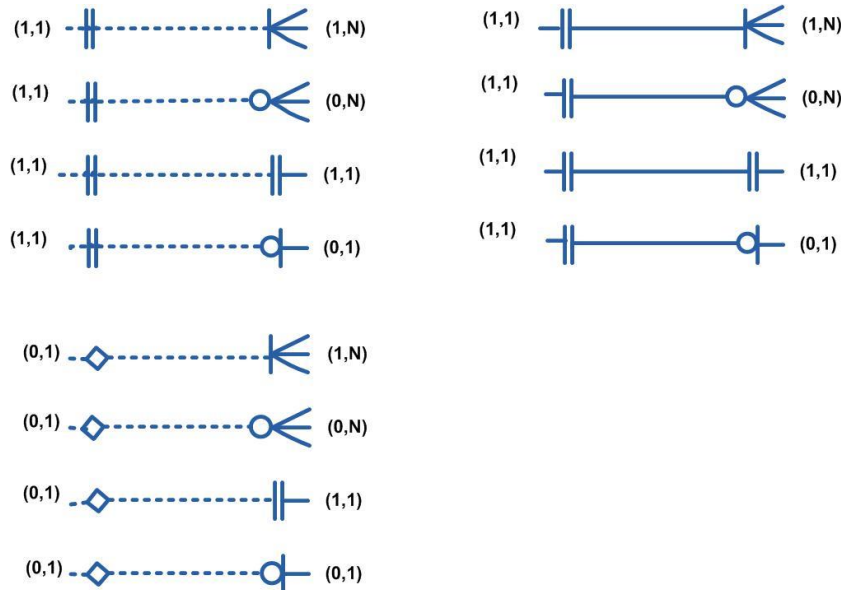
Diseño conceptual



Diseño lógico.

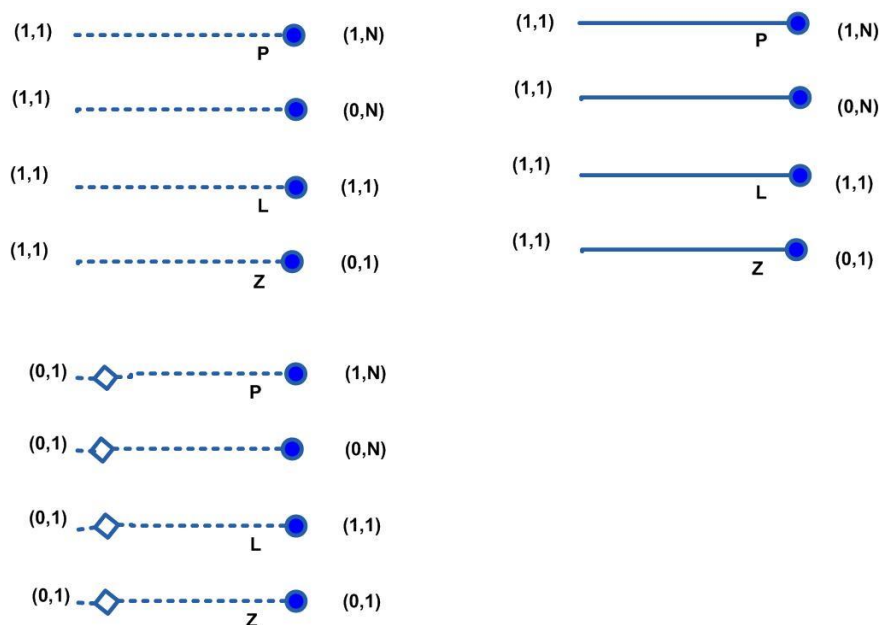
- Similar al diseño conceptual, se emplea la notación (min, max) en ambos lados de las entidades.
- Adicionalmente las relaciones identificativas y no identificativas se complementan con las siguientes notaciones:

4.3.3.1. Cardinalidad con notación crow's Foot



- Observar que no existen relaciones identificativas con cardinalidad (0,1) del lado izquierdo. **¿Cuál es la razón?** Respuesta: la cardinalidad (0,1) del lado de la tabla padre implica una FK que puede tener valores nulos, cosas que no ocurre con una identificativa, ya que la FK forma parte de la PK de la tabla hija, y las llaves primarias no deben tener valores nulos.

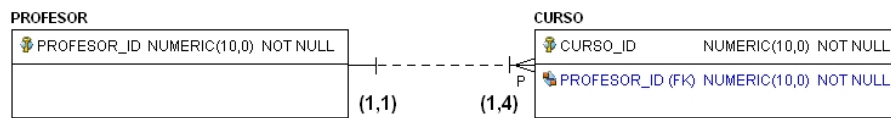
4.3.3.2. Cardinalidad con notación IDEF1X



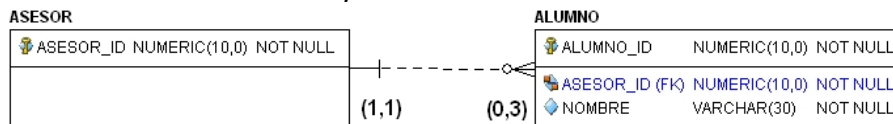
Ejemplo:

Retomando el ejemplo anterior:

- **Un** profesor imparte como máximo **4** cursos.
- **Un** curso es impartido por **un** profesor.

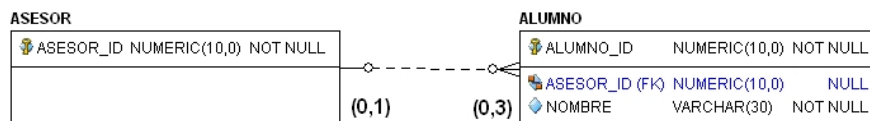
Ejemplo:

- Un asesor si lo desea, puede ser asesor de hasta 3 alumnos.
- Un alumno **debe** contar con su asesor y es uno solo durante su estancia en la escuela.

Ejemplo:

Suponer que se modifica la regla anterior:

- Un asesor si lo desea puede ser asesor de hasta 3 alumnos.
- Un alumno **puede** solicitar a un único asesor durante su estancia en la escuela.

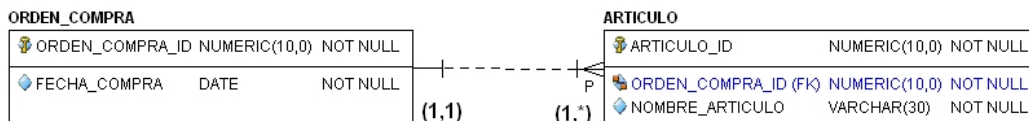


Observar las diferencias en los 2 diagramas anteriores:

- La cardinalidad de alumno hacia asesor es (0,1), ya que el alumno puede o no tener asesor.
- Observar que, en este caso, la FK debe ser declarada como null. De lo anterior, siempre que en valor mínimo de la cardinalidad sea 0, la FK se declara como opcional (null).

Ejemplo:

- Una orden de compra está integrada por varios artículos.
- Un artículo pertenece a una orden de compra.



- Los valores de los lados izquierdos de la cardinalidad, solamente deben tener los valores (0,1), o (1,1)

4.3.4. Dependencia de existencia.

- Una entidad se considera **dependiente de existencia** cuando sus instancias o registros requieren la existencia de una instancia de la entidad con la que se relaciona.

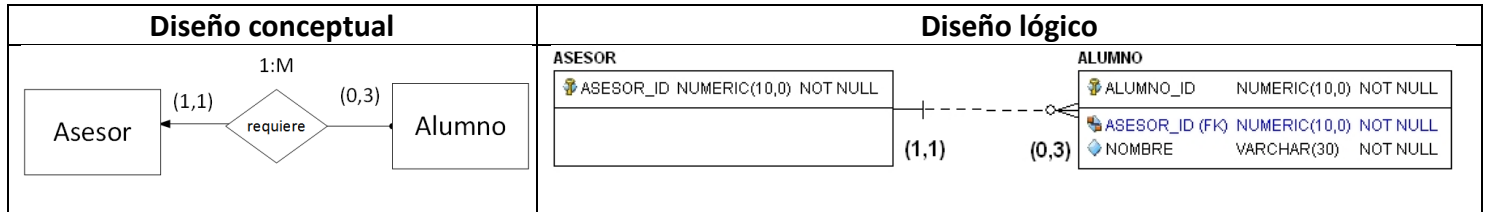


- La dependencia de existencia siempre se verifica del lado de la entidad “**hija**”, la cual puede ser dependiente o independiente de existencia con respecto a la tabla padre.

Ejemplo:

Considerando nuevamente el ejemplo de las entidades `asesor` y `alumno`.

- Un profesor si lo desea, puede ser asesor de hasta 3 alumnos.
- Un alumno **debe** contar con su asesor y es uno solo durante su estancia en la escuela.



Si observamos las reglas de negocio con respecto a la tabla hija: `alumno`, podemos decir, que `alumno` es **dependiente de existencia**, ya que la regla nos dice que todo alumno debe contar con su asesor. Es decir, para poder registrar un alumno es necesario que se le asigne su asesor, por lo tanto, requiere que exista una instancia de `asesor`.

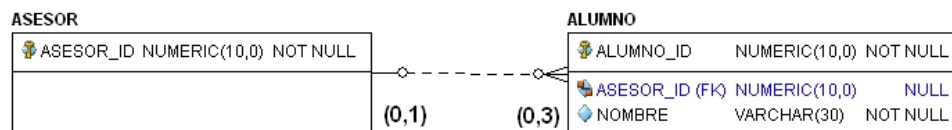
La base de datos nos permite implementar esta condición definiendo la FK como `not null`. Esta restricción impide que se registre un alumno sin su asesor.

Por otro lado, observar la cardinalidad de la tabla `alumno` hacia `asesor` (lado izquierdo). Cuando una entidad es dependiente de existencia, el valor de la cardinalidad siempre es (1,1). Adicionalmente, observar la notación. En Crow's foot, del lado **izquierdo** se emplean 2 líneas para indicar la relación (1,1).

Ejemplo:

Retomando la variante de las entidades `asesor` y `alumno`

- Un profesor si lo desea puede ser asesor de hasta 3 alumnos.
- Un alumno **puede** solicitar a un único asesor durante su estancia en la escuela.



Ahora observamos las reglas de negocio, podemos concluir que la tabla `alumno` es **independiente de existencia** con respecto a `asesor`.

- El alumno puede ser registrado sin su asesor, tal y como lo dicen la regla de negocio.
- La FK está declarada como `null`.
- La cardinalidad del lado de `asesor`, toma el valor de (0,1). Esto siempre ocurre con una entidad independiente de existencia.
- Observar la notación, tanto en Crow's foot como en IDEF1X, se emplea un “rombo” para representar la independencia de existencia.



4.3.5. Participación de una entidad en una relación

A diferencia de la dependencia de existencia, este concepto se verifica observando la cardinalidad en la tabla padre. La participación de una entidad en una relación puede ser obligatoria u opcional:

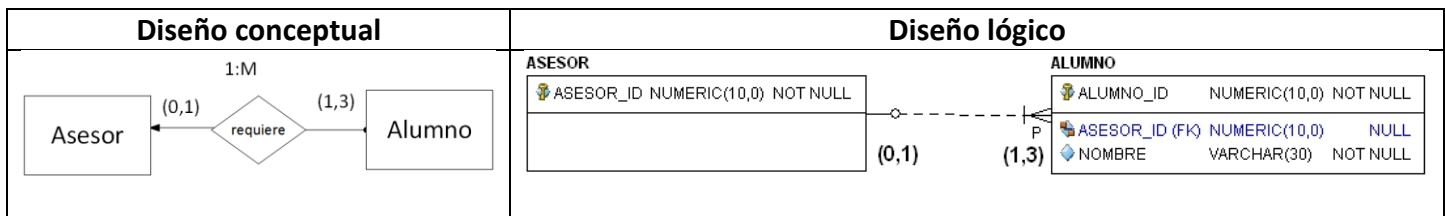
4.3.5.1. Participación obligatoria.

Una entidad “padre” tiene participación obligatoria en la relación con su entidad hija, cuando cada una de las instancias de la entidad padre se asocia con al menos una instancia de la entidad hija.

Ejemplo:

Considerando la siguiente variante entre `asesor` y `alumno`.

- Un profesor **debe** ser asesor de uno o hasta 3 alumnos.
- Un alumno **puede** solicitar a un único asesor durante su estancia en la escuela.



- Ahora nos concentramos en revisar las reglas de negocio con respecto a la entidad padre. Para determinar el tipo de participación podemos formular la siguiente pregunta:

¿Puede existir una instancia de `asesor` sin asociarse con `alumno`?

- En este caso la respuesta es NO, ya que cada profesor debe asesorar al menos a un alumno. Por lo tanto, la entidad `ASESOR` tiene **participación obligatoria**.
- Observar la cardinalidad del lado derecho, cuando una entidad padre tiene participación obligatoria, el valor mínimo de cardinalidad deber ser **1**. Adicionalmente, observar la notación, en crow's foot tiene el símbolo P y en IDEF1X, solo se escribe la “P”.



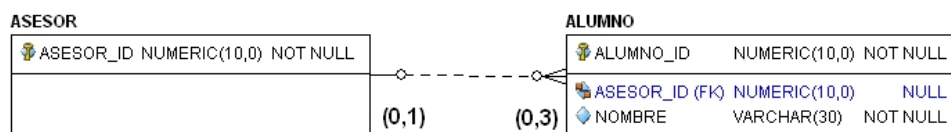
Es importante no confundir con el concepto de dependencia de existencia, es decir, no importa si la FK está definida como `null` o `not null`.

La base de datos **no tiene** forma de restringir esta condición, solo se representa empleando las notaciones y observando el valor mínimo de la cardinalidad del lado derecho.

Ejemplo:

Considerando la siguiente variante entre `asesor` y `alumno`.

- Un profesor **si así lo desea**, puede ser asesor máximo de 3 alumnos.
- Un alumno **puede** solicitar a un único asesor durante su estancia en la escuela.



Aplicando la misma pregunta:

¿Puede existir una instancia de `asesor` sin asociarse con `alumno`?

- En este caso la respuesta es SI, ya que las reglas nos indican que, si un profesor no lo desea, no asesora, pueden existir instancias de `asesor` que no se relacionen con `alumno`. Por lo tanto, la entidad `asesor` tiene **participación opcional**.
- Observar que el valor mínimo de la cardinalidad del lado derecho en una participación opcional es 0. Nuevamente, no confundir con el concepto de dependencia de existencia, no importa la forma en la que se define la FK (`not null` o `null`).

Ejercicio en clase 3:

Considerar nuevamente el enunciado del ejercicio en clase 1 que hace referencia a una base de datos empleada para guardar los datos de los profesores aspirantes de un proyecto.

- Asignar las cardinalidades a cada una de las relaciones obtenidas.
- Para cada relación, aplicar los conceptos de dependencia e independencia de existencia, participación opcional y obligatoria.



4.3.6. Dependencia de identificación

- Representa una extensión del concepto de dependencia de existencia.
- Este tipo de relación se verifica en la tabla hija al igual que en dependencia de existencia.
- La tabla hija define su propia PK, pero sus valores no son suficientes para poder garantizar unicidad, es decir, la tabla hija requiere de la PK de la tabla padre para funcionar correctamente, formando una **PK compuesta**.
- Esta situación se produce generalmente cuando los valores de la PK original de la tabla hija se reinician o se repiten por cada valor de la PK de la tabla padre.

Ejemplo:

- Una librería cuenta con un catálogo de libros.
- Un libro se identifica de manera única con su ISBN
- Por cada ejemplar que se compra de dicho libro, se guarda un registro en base de datos identificándolo por su número iniciando en 1, por ejemplo, el ejemplar 1, del libro con ISBN 12, el ejemplar 2 del libro con ISBN 12, el ejemplar 1 del libro con el ISBN 13, etc.
- Se observa que el número de ejemplar se puede repetir por cada libro, es decir, ejemplar 1 del libro 12, el ejemplar 1 del libro 13, etc.
- Por lo anterior, la llave primaria de la entidad `Ejemplar` estará formada por la pareja `<ISBN, numEjemplar>` para garantizar combinaciones únicas en `ejemplar`.



Diseño conceptual	Diseño lógico
<p>Para hacer énfasis en que la llave primaria de la entidad <code>Libro</code> debe propagarse hacia la entidad <code>Ejemplar</code>, el rombo y el rectángulo de la tabla hija se marca con doble línea.</p>	<p>Observar que la PK de <code>LIBRO</code> se propaga a <code>EJEMPLAR</code> empleando una relación identificativa. De esta forma es posible identificar de forma única a cada ejemplar</p>

- Los datos se verían así:

ISBN (PK)	NUM_EJEMPLAR_ID (PK)(FK)
1001	1
1001	2
1001	3
1002	1
1002	2
1003	1

Ejercicio en clase 4:

Generar el modelo ER y el modelo relacional para el siguiente enunciado:



- En una sala de teatro se reparten boletos foliados por cada función iniciando en 001.
- Adicional al folio, se almacena el importe del boleto, el nombre de la función y el número de asiento.
- Un cliente puede comprar varios boletos, se almacena el nombre, apellidos y el email (único por cliente).
- Se desea almacenar en la base de datos, los boletos que ha comprado cada cliente.

4.3.7. Grado de una relación.

Consiste en clasificar a las relaciones por el número de entidades (no instancias) que participan en una relación:

- Relaciones unarias
- Relaciones binarias
- Relaciones ternarias.

4.3.7.1. Relaciones unarias.

Ocurre al existir una relación empleando una sola entidad, es decir, una instancia de una entidad se relaciona con otra instancia de la **misma** entidad. A este tipo de relaciones también se les conoce como relaciones **recursivas**.

Ejemplo:

Considerar las siguientes reglas de negocio.

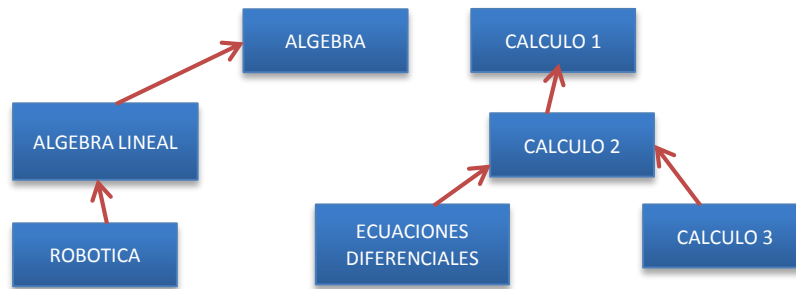
Asignatura y Asignatura antecedente. (1:M)

- Una asignatura puede requerir de una asignatura antecedente para poder cursarse. (1:1)
- Una asignatura puede ser antecedente de varias asignaturas. (1:M)



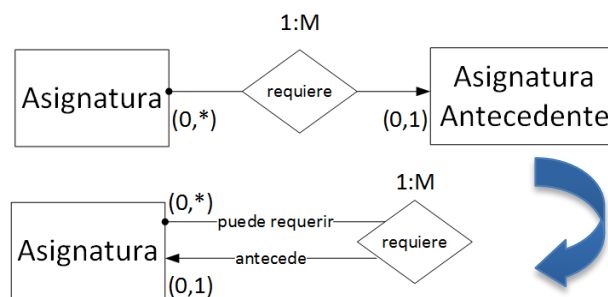
Estas 2 reglas de negocio se pueden reflejar en estos escenarios:

- La asignatura Álgebra lineal requiere de la asignatura Álgebra para que un alumno la pueda cursar.
- La asignatura Robótica requiere de la asignatura Álgebra lineal para poder cursarse.
- La asignatura Calculo 2 debe cursarse antes de cursar Calculo 3 y ecuaciones diferenciales



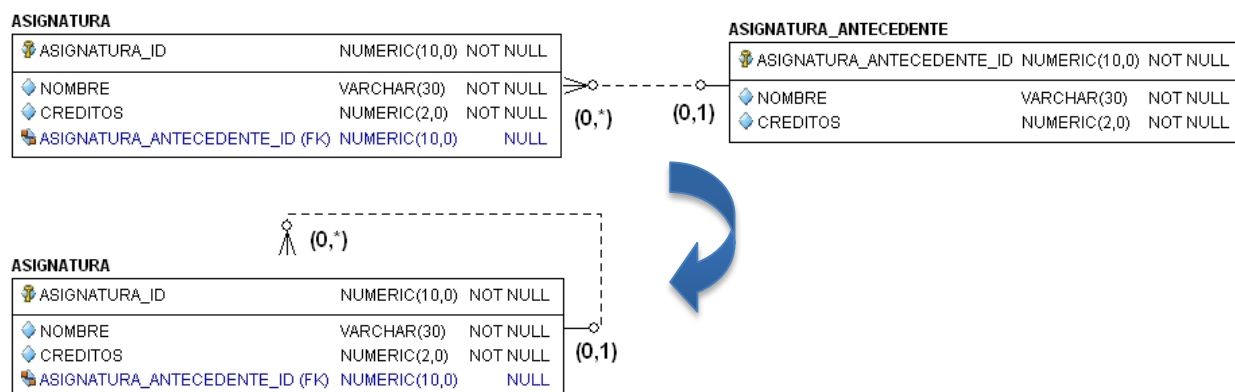
- Observar que este tipo de relación permite asociar instancias de una entidad en una estructura jerárquica, en donde la instancia raíz no se asocia con alguna otra instancia.
- La forma más sencilla es imaginar a las 2 entidades como si existieran por separado:

Diseño conceptual.



Una asignatura puede requerir de una asignatura antecedente para poder cursarse.
Una asignatura puede ser antecedente de varias asignaturas.

Diseño lógico.



- Dependencia de existencia: **Independiente**. No todas las asignaturas (tabla hija) tienen asignatura antecedente (tabla padre). La FK es null
 - ¿Qué pasaría si la FK se declara como not null?
- Participación de una entidad: **obligatoria**. Toda asignatura antecedente (tabla padre) debe asociarse con al menos una asignatura subsecuente.

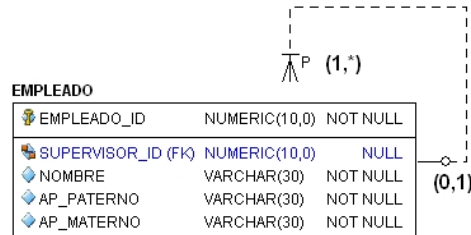
Los datos se verían así:

ASIGNATURA_ID	NOMBRE	CREDITOS	ASIGNATURA_REQUERIDA_ID
1	ALGEBRA	6	NULL
2	ALGEBRA LINEAL	7	1
3	CALCULO 1	6	NULL
4	CALCULO 2	7	3
5	CALCULO 3	7	4
6	ECUACIONES DIFERENCIALES	7	4
7	ROBOTICA	8	2

Ejemplo:

Entidad Empleado y Supervisor.

- A cada empleado se le asigna su supervisor el cual también es un empleado.
- Un supervisor puede tener asignados a varios empleados.



- En este caso se tiene una independencia de existencia. No todos los empleados tienen supervisor, es decir, los supervisores en si no tienen supervisor.
- Participación: obligatoria. Un supervisor debe tener al menos un empleado asignado.

Los datos se verían así:

EMPLEADO_ID	NOMBRE	AP_PATERNO	AP_MATERNO	SUPERVISOR_ID
1	JUAN	MARTINEZ	AGUIRRE	4
2	LILIANA	LOPEZ	HURTADO	NULL
3	GERARDO	JIMENEZ	BENITEZ	2
4	MARIA	MONDRAGON	MENDEZ	NULL
5	JORGE	LUNA	AGUILAR	2

4.3.7.2. Relaciones binarias.

Ocurren entre 2 entidades diferentes. Este tipo de relaciones son las que se revisaron en la sección 4.5

4.3.7.3. Relaciones ternarias.

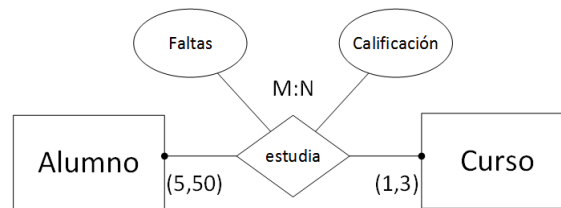
Ocurren cuando se relacionan 3 entidades entre si para poder cumplir una regla de negocio. ¿En qué casos se podrían relacionar 3 entidades? Suponer el siguiente escenario:

Entidades Curso y Alumno (M:N).

- Un curso está integrado mínimo de 5 alumnos, máximo de 50 (1:M)
- Un alumno puede tomar uno o hasta 3 cursos. (1:M)
- Se requiere registrar la calificación y el número de faltas que obtuvieron los alumnos.



Diseño conceptual:



- Observar que los atributos que dependen de la combinación de ambas instancias se asocian al rombo y no a las entidades.
- Tanto la calificación como las faltas dependen de la combinación de los valores de cada alumno y de cada curso.

Diseño lógico

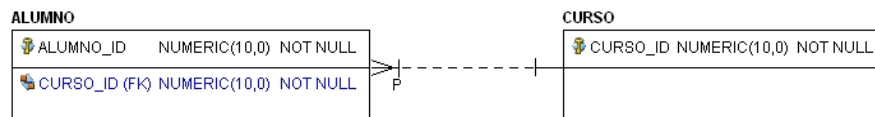
Si se intenta modelar una relación M:N asignando la FK en alguna de las 2 tablas ocurren las siguientes situaciones:

- FK en curso



En esta estrategia, se cumple la regla “un alumno puede tomar varios cursos” ya que un mismo `alumno_id` puede aparecer en cursos diferentes. Sin embargo, la regla “un curso está formado por varios alumnos” no se cumple. Como se puede observar en el diagrama, por cada curso únicamente se puede registrar a un alumno.

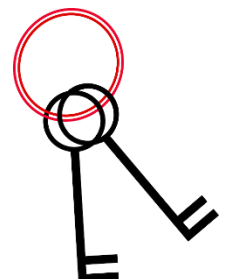
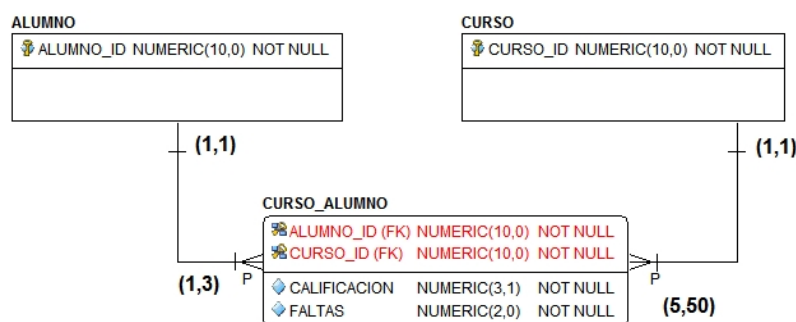
- FK en alumno



En esta estrategia, se cumple la regla “un curso está formado por varios alumnos”. El mismo `curso_id` puede aparecer en varios registros de la tabla `alumno`. Sin embargo, la regla “Un alumno puede tomar varios cursos” no se cumple, por alumno solo se puede registrar a un alumno.

Para resolver el problema anterior, en una relación M:N, se debe incorporar una tabla intermedia. El modelo relacional no soporta la implementación de relaciones M:N de forma directa.

Estrategia 1: PK compuesta.



- Observar en este caso la cardinalidad en ambos lados de la tabla intermedia:
 - Un alumno toma como mínimo un curso (participación obligatoria), y como máximo 3.
 - Un curso está formado por mínimo 5 alumnos (participación obligatoria), máximo 50.
- Observar que la dependencia de existencia no tiene sentido verificarla, las FKs forman parte de la llave primaria de la tabla intermedia, por lo tanto, no pueden ser definidas como NULL.
- En una relación M:N siempre existirá dependencia de existencia.
- La participación puede ser opcional u obligatoria.
- Los datos se verían así:

ALUMNO

ALUMNO_ID	NOMBRE
1	JUAN
2	MARIO
3	LAURA
4	LORENA

CURSO

CURSO_ID	NOMBRE
1000	PROGRAMACIÓN
1001	ALGORITMOS
1002	INTEGRACION
1003	COCINA

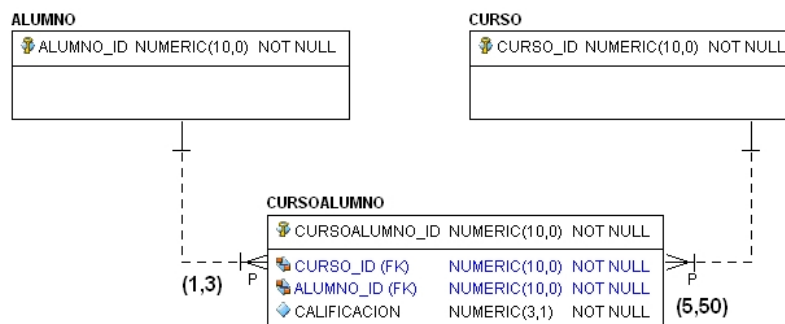
CURSO_ALUMNO

CURSO_ID(PK)(FK)	ALUMNO_ID(PK)(FK)	CALIFICACION
1000	1	10
1001	1	9
1002	2	8
1002	3	7
1003	4	9



- Observar que la tabla intermedia permite implementar correctamente las reglas de negocio al incorporar una PK compuesta.
 - Un alumno puede tomar varios cursos (primeros 2 registros)
 - Un curso está formado por varios alumnos (registro 3 y 4)
- Observar que los campos que pueden aparecer en la tabla intermedia dependen de la combinación de los valores de la PK. Por ejemplo, la calificación, es un atributo que depende tanto del curso como del alumno (para cada curso y para cada alumno se asigna una calificación).

Estrategia 2: PK artificial.



En esta estrategia se emplea una PK simple y artificial, y las 2 tablas se relacionan a través de relaciones no identificativas. Aunque la BD lo permite, las FKs no deben declararse como `null` por lo siguiente:

- ¿Qué sentido tendría un registro en la tabla intermedia en donde un alumno tuviera como valor de `curso_id` un valor nulo?
- ¿Qué sentido tendría un registro en la tabla intermedia en donde un curso tuviera un `alumno_id` con un valor nulo?
- ¿Qué pasa si un alumno no se inscribe a algún curso? Las FKs aun así no deben ser marcadas como `null`. En este caso, simplemente no existiría un registro en la tabla intermedia, por lo tanto, estaríamos hablando de una participación opcional, y la cardinalidad del lado izquierdo cambiaría a (0,3).
- Lo mismo ocurre con el curso. Si las reglas nos dijeran que pudiera existir un curso sin alumnos, no habría registros en la tabla intermedia asociados a dicho curso, las FKs siguen siendo definidas como `not null` y la cardinalidad cambiaría a (0,50).

Ventajas de esta estrategia:

- Al tener una PK artificial el desempeño tiende a mejorar ya que al realizar consultas y relacionar tablas, el procesamiento con llaves compuestas siempre es mayor al requerido para ligar tablas con PKs simples.
- Las consultas SQL requeridas para explotar los datos de esta tabla intermedia se simplifican con PKs simples (esto se verá en la parte de SQL).

Desventajas de esta estrategia:

- Se agrega un nuevo campo (la PK artificial), sin embargo, no representa un problema mayor en cuanto a espacio de almacenamiento.
- Posibilidad de insertar datos inconsistentes, situación que no ocurre con la estrategia anterior. En este caso, se le delega a la aplicación o al usuario insertar datos consistentes.

Ejemplo:

Los siguientes registros son inconsistentes. La base de datos no tiene forma de detectar o restringir la inconsistencia, a menos que se creara un trigger para verificar.

CURSO_ALUMNO

CURSO_ALUMNO_ID(PK)	CURSO_ID(FK)	ALUMNO_ID(FK)	CALIFICACION
1	1000	1	10
2	1000	1	9

- En este caso, el registro es duplicado, y lo peor es que se tienen 2 calificaciones distintas para el mismo alumno y curso.
- Una solución adicional, podría ser, crear un UNIQUE index que aplique a ambos campos, tanto a `curso_id` como a `alumno_id`

En conclusión, se recomienda más la segunda estrategia, a menos que hubiera algún requerimiento muy particular que lo impida.

4.4. TIPOS DE DATOS EMPLEADOS PARA REALIZAR MODELOS RELACIONALES.

Un punto importante dentro del diseño lógico es la selección del tipo de dato. Cuando se agrega cada una de las columnas de una tabla es indispensable indicar el tipo de dato que debe contener la columna. En esta sección se revisan los diferentes tipos de datos SQL que pueden ser empleados durante el proceso de diagramado de un caso de estudio.



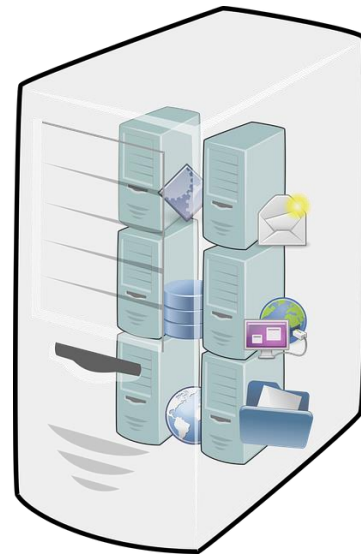
Adicional a las restricciones que pudiera tener una columna en una tabla, es importante definir el tipo de dato y su dominio. Dicha definición debe reflejar de forma correcta la naturaleza de los datos:

- Valores monetarios
- Fechas
- Cantidades
- Nombres, etc.

Para implementar este requerimiento, el estándar SQL ha definido una serie de tipos de datos que todo manejador debe implementar. La realidad es que no todos los manejadores cumplen al 100% con el estándar. Algunos definen variantes o sus propios tipos de datos.

Clasificación de los tipos de datos SQL:

- Cadenas
 - Cadenas de caracteres
 - De longitud fija
 - De longitud variable
 - Cadenas binarias
- Numéricos
 - Exactos
 - Aproximados
- Tiempo y fechas
- Otros:
 - boolean (Estándar,MySQL, PostgreSQL)
 - ROWID (Oracle)
 - UROWID (Oracle)
 - BFILE (Oracle)
 - XML



4.4.1. Cadenas de caracteres

Conjunto o secuencia de caracteres que pertenecen a un “Juego de caracteres” predefinido. La longitud de la cadena está definida por el número de caracteres que contiene. El número de bytes de espacio requerido para almacenar una cadena depende del **juego de caracteres** con el que se represente. Internamente, cada carácter es representado generalmente por un valor numérico que se obtiene a través de la aplicación de algún algoritmo de codificación. Algunas normas de codificación (juego de caracteres):

- ISO 646
- ASCII
- UNICODE
- ISO 8859
- Juego de caracteres de Windows
- UTF8

Ejemplos:

Carácter	ISO-8859-1	UTF-8	UTF-16
a	0x61	0x61	0x00 0x61
b	0x62	0x62	0x00 0x62

Uno de los puntos importantes al momento de instalar una base de datos, es la asignación del **juego de caracteres** que se empleará para almacenar los datos. Esto permitirá la correcta codificación, representación y almacenamiento de las cadenas de caracteres en la base de datos. Dependiendo de la diversidad de símbolos y caracteres que puedan ser enviados a la base de datos, se deberá elegir el juego de caracteres a configurar. Por default, se selecciona el juego de caracteres que emplea el sistema operativo.

4.4.1.1. Cadenas de caracteres de longitud fija:

Si un atributo en la BD se define como una cadena de longitud fija N, como máximo se podrán insertar N caracteres. Si la cadena es menor a N, esta se rellena con espacios.

Ejemplo: cadena de caracteres de longitud 8 (fija)

A	R	P	A				
---	---	---	---	--	--	--	--

M	U	S	I	C	A		
---	---	---	---	---	---	--	--

P	I	A	N	O			
---	---	---	---	---	--	--	--



El tipo de dato que representa a una cadena de longitud fija es CHAR. (Ver tabla siguiente).

Sintaxis:

CHAR[ACTER] [(n)]

EL contenido que esta entre [], es opcional. El valor de “n” representa al número de caracteres de la cadena. Si no se especifica, es equivalente a CHAR (1) , un carácter.

Ejemplos:

Ejemplo	Descripción
CHAR (10)	Cadena de caracteres de longitud 10
CHAR	Cadena de caracteres de longitud 1

4.4.1.2. Cadenas de caracteres de longitud variable.

A diferencia de las cadenas de longitud fija, solo se almacena el número real de caracteres que contiene la cadena. A nivel de definición del campo solo se define la capacidad máxima.

Ejemplo: cadena de caracteres de longitud 8 (variable)

A	R	P	A
---	---	---	---

M	U	S	I	C	A
---	---	---	---	---	---

P	I	A	N	O
---	---	---	---	---

El tipo de dato que representa a una cadena de longitud fija es VARCHAR. (Ver tabla siguiente).

Sintaxis:

VARCHAR (n)

En este caso, “n” define el número máximo de caracteres que puede tener la cadena: VARCHAR (1) , VARCHAR (100) , etc.

Para el caso particular de Oracle, se recomienda emplear `VARCHAR2` (este será el tipo de dato a emplear en el curso para las prácticas de SQL).

4.4.1.3. Cadenas de caracteres de gran tamaño.

En algunas ocasiones no se conoce con precisión el tamaño máximo que pudiera tener una cadena a insertar en la base de datos. Lo único que se conoce es que pueden ser cadenas muy grandes. Para estos casos, existe en tipo de dato `CLOB` (Character Large Object).

Se emplea para almacenar grandes cantidades de caracteres, textos, artículos, contenido de un libro, etc.



4.4.1.4. Cadenas de caracteres en diferente juego de caracteres.

Los tipos de datos `NCHAR` Y `NCHAR VARYING(N)`, `NVARCHAR2` (para Oracle) se emplean para guardar cadenas representadas en un juego de caracteres diferente al configurado en la base de datos. Para el caso de Oracle, se emplea como juego de caracteres `UNICODE`, aunque el estándar define `CHARACTER VARYING(n) CHARACTER SET <name>` en el que se especifica el nombre del juego de caracteres a emplear, Oracle siempre emplea `UNICODE`. La principal razón: `UNICODE` es el juego de caracteres universal, por lo que se puede representar cualquier carácter.

4.4.1.5. Clasificación de los tipos de datos para representar cadenas de caracteres

La siguiente tabla muestra la clasificación de las cadenas de caracteres. Del lado izquierdo, representa al tipo de dato definido por el estándar SQL, y las demás columnas representan la forma en la que cada manejador implementa al tipo de dato.

SQL Estándar (especificación)	ORACLE 11g	DB2 9.5	SQL Server 2008	PostgreSQL 8.x	MySQL 5.x
CHAR [ACTER] [(n)]	CHAR[ACTER] [(n)]	CHAR[ACTER] [(n)]	CHAR[ACTER] [(n)]	CHAR[ACTER] [(n)]	CHAR[(M)]
CHAR[ACTER] VARYING(n) ó VARCHAR(n)	CHAR[ACTER] VARYING(n) ó VARCHAR(n) ó VARCHAR2(n)	CHAR[ACTER] VARYING(n) ó VARCHAR(n) ó LONGVARCHAR	CHAR[ACTER] VARYING(n) ó VARCHAR[(n)] TEXT	CHAR[ACTER] VARYING(n) ó VARCHAR(n)	VARCHAR(M) ó TINYTEXT ó TEXT ó MEDIUMTEXT ó LONGTEXT
CLOB ó CHARACTER LARGE OBJECT	CLOB ó LONG[VARCHAR]	CLOB[(n) [K M,G]]	VARCHAR(MAX)	TEXT	BINARY[(M)] VARBINARY(M)
NATIONAL CHAR[ACTER] [(n)] ó NCHAR [(n)] ó CHARACTER[(n)] CHARACTER SET <charset-name>	NATIONAL CHAR[ACTER] [(n)] ó NCHAR [(n)]	GRAPHIC[(n)]	NATIONAL CHAR[ACTER] [(n)] ó NCHAR[(n)] ó NTEXT	-----	NATIONAL CHARACTER(n) ó CHAR(n) CHARACTER SET <name> ó NCHAR(n)
NATIONAL CHAR[ACTER] VARYING(n) ó NCHAR VARYING(n) ó CHARACTER VARYING(n) CHARACTER SET <name>	NATIONAL CHAR[ACTER] VARYING(n) ó NCHAR VARYING(n) ó NVARCHAR2(n)	VARGRAPHIC(n) ó LONG VARGRAPHIC(n)	NATIONAL CHARACTER VARYING [(n)] ó NTEXT	-----	VARCHAR(n) CHARACTER SET <name> ó NATIONAL VARCHAR(n) ó NCHAR VARCHAR(n) ó NATIONAL CHAR[ACTER] VARYING(n)
NATIONAL CHARACTER LARGE OBJECT	NCLOB	DBCLOB[(n) [K M G]]	NVARCHAR(MAX)	-----	-----

- Las palabras entre `[]` son opcionales, las palabras entre `<>` se sustituyen por un valor.

4.4.1.6. Rangos soportados por los principales manejadores para cadenas de caracteres

Tipo de dato	Oracle	Tipo de dato	DB2	Tipo de dato	SQL Server	Tipo de dato	MySQL
CHAR	1-2000	CHAR	1-254	CHAR	1-8000	CHAR	1-255
VARCHAR2	1-4000	VARCHAR	1-32672	VARCHAR	1-8000	VARCHAR	1-65535
NCHAR (unicode)	1-2000	LONG VARCHAR	1-32700	TEXT	2 GB	TINYTEXT	0-255
NVARCHAR (unicode)	1-4000	CLOB	2 GB	NCHAR	1-4000	TEXT	0-65535
CLOB	8 TB	GRAPHIC	1-127	NTEXT	1 GB	MEDIUM TEXT	0-16777215
NCLOB	8 TB	VARGRAPHIC	1-16, 336	VARCHAR (MAX)	2 GB	LONGTEXT	0-4294967295
		LONG VARGRAPHIC	16,350				

Tipo de dato	PostgreSQL
CHAR	1 GB
VARCHAR	1-GB
TEXT	1-GB



4.4.2. Cadenas binarias

Se emplean para almacenar secuencia de bytes en la base de datos, es decir, se almacenan archivos binarios: fotos, música, videos, huellas, documentos, etc. El tipo de dato más común empleado es BLOB (Binary Large Object), de forma similar a CLOB, no es necesario escribir la longitud máxima que puede almacenarse en la BD.

Clasificación:

SQL Estándar (especificación)	ORACLE 11g	DB2 9.5	SQL Server 2008	PostgreSQL 8.x	MySQL 5.x
BLOB ó BINARY LARGE OBJECT	BLOB ó LONGRAW ó RAW (n)	BLOB [(n) [K M G]]	VARBINARY (MAX) ó VARBINARY [(n)] ó IMAGE	BYTEA	BINARY [(n)] ó VARBINARY (M) ó TINYBLOB ó BLOB ó MEDIUM BLOB ó LONG BLOB

4.4.2.1. Rangos cadenas binarias.

Tipo de dato	Oracle	Tipo de dato	DB2	Tipo de dato	SQL Server
BLOB	8 TB	BLOB	2 GB	BINARY	8000
RAW	4000 BYTES			VARBINARY	8000
				IMAGE	2147483647
				VARBINARY (MAX)	2 GB

¡Observar que en el caso de Oracle es posible almacenar hasta un archivo de 8 TB!

4.4.3. Tipos de datos numéricos.

Existen 2 clasificaciones de los tipos de datos numéricos:

- Tipos de datos numéricos exactos
- Tipos de datos numéricos aproximados.



4.4.3.1. Tipos de datos numéricos exactos

Los tipos de datos numéricos exactos pueden ser números enteros finitos o números con parte decimal finita.

Al definir un atributo de una tabla con un tipo de dato numérico exacto se puede definir 2 elementos:

- Precisión: Número total de dígitos: Parte entera + parte decimal
- Escala: Número de dígitos que formarán a la parte decimal.

Por lo anterior, el número máximo de dígitos que podrá tener la parte entera de un valor numérico será:

#digitosParteEntera = precisión- escala

Ejemplo.

¿Cuál será el dominio de la siguiente columna? precio number(10,4)

Precision = 10

Escala = 4

Parte entera = 10-4 =6

Dominio: [0,999999.9999]



4.4.3.2. Clasificación de los tipos de datos numéricos exactos.

SQL Estándar (especificación)	ORACLE 11g	DB2 9.5	SQL Server 2008	PostgreSQL 8.x	MySQL 5.x
INT[EGER]	NUMBER (n)	INT[EGER] ó BIGINT	INT[EGER] ó BIGINT	INTEGER ó BIGSERIAL ó SERIAL ó BIGINT	INT[(M)] ó MEDIUMINT[(M)] ó BIGINT[(M)]
SMALLINT	SMALLINT NUMBER (n)	SMALLINT	SMALLINT ó TINYINT	SMALLINT	SMALLINT[(M)] ó TINYINT[(M)]
NUMERIC[(P[,S])] ó DEC[IMAL] [(P[,S])]	NUMERIC [(P[,S])] ó DEC[IMAL] [(P[,S])] ó NUMBER[(P[,S])]	NUMERIC [(P[,S])] ó DEC[IMAL] [(P[,S])]	NUMERIC [(P[,S])] ó DEC[IMAL] [(P[,S])] MONEY SMALLMONEY	NUMERIC [(P[,S])] ó DEC[IMAL] [(P[,S])]	

4.4.3.3. Tipos de datos numéricos aproximados.

Se emplean cuando no se conoce con exactitud los valores de la precisión y/o escala que puede tener los valores de una columna.

4.4.3.4. Clasificación de los tipos de datos numéricos aproximados.

SQL Estándar (especificación)	ORACLE 11g	DB2 9.5	SQL Server 2008	PostgreSQL 8.x	MySQL 5.x
FLOAT[(P)]	FLOAT[(P)] ó NUMBER	FLOAT[(P)]	FLOAT[(P)]	-----	FLOAT[(P[,S])]
REAL	REAL NUMBER	REAL	REAL	REAL	REAL[(P[,S])]
DOUBLE PRECISION	DOUBLE PRECISION NUMBER	DOUBLE [PRECISION]	DOUBLE PRECISION	DOUBLE PRECISION	DOUBLE [PRECISION]

4.4.3.5. Rangos tipos de datos numéricos para Oracle:

- INTEGER, SMALLINT se convierten a NUMBER (38)
- NUMERIC, DECIMAL se convierten a NUMBER



- Rango: 1×10^{-130} a 9×10^{125} (38 nuevos).
- Oracle permite la definición de escalas negativas empleadas para redondear. Por ejemplo:
 - `NUMBER(10,2)`, `6,345,768,903.678` será redondeado a `6,345,768,904`

Observar que, para el caso de Oracle, en todos los tipos que define el estándar, Oracle lo representa a través del tipo de dato `NUMBER`. Por lo anterior, se recomienda que todos los tipos de datos numéricos (tanto exactos como aproximados) sean representados empleando `NUMBER`.

4.4.4. Tipos de datos para el manejo de fechas.

Esta clasificación de tipos de dato es la más complicada en el sentido de que prácticamente ningún manejador sigue el estándar SQL:

A nivel del estándar:

- `DATE` representa una fecha hasta el nivel de días (Año, Mes, Día)
- `TIME` Representa tiempo a nivel de HH:MM:SS
- `TIMESTAMP` Representa una combinación: Año, Mes, Día, Hora, Minuto, Segundo, Milisegundo



Es importante recalcar que, en la mayoría de las aplicaciones o clientes gráficos empleados para consultar los datos, estos emplean un formato predefinido para representar a una fecha. Ejemplos:

- 02/03/08
- 02-03-2008
- 02 de marzo de 2008, etc.

Lo anterior no significa que en la base de datos una fecha se almacene como una cadena con un determinado formato. Normalmente las fechas se almacenan internamente como datos numéricos.

4.4.5. Otros tipos de datos.

4.4.5.1. Clasificación de los tipos de datos para manejo de fechas.

SQL Estándar (especificación)	ORACLE 11g	DB2 9.5	SQL Server 2008	PostgreSQL 8.x	MySQL 5.x
<code>DATE</code>	<code>DATE</code>	<code>DATE</code>	<code>DATETIME</code> ó <code>SMALLDATETIME</code>	<code>DATE</code>	<code>DATE</code>
<code>TIME[WITH TIME ZONE]</code>	<code>DATE</code>	<code>TIME</code>	<code>DATETIME</code> ó <code>SMALLDATETIME</code>	<code>TIME[(P)] [WITH[OUT] TIMEZONE]</code>	<code>TIME</code>
<code>TIMESTAMP[(P)] [WITH TIME ZONE]</code>	<code>DATE</code> ó <code>TIMESTAMP[(P)] [WITH TIME ZONE]</code>	<code>TIMESTAMP</code>	<code>DATETIME</code> ó <code>SMALLDATETIME</code>	<code>TIMESTAMP [(P)] [WITH[OUT] TIMEZONE]</code>	<code>DATETIME</code> ó <code>TIMESTAMP</code>
<code>INTERVAL</code>	<code>INTERVAL DAY [(P)] TO SECOND [(P)]</code> ó <code>INTERVAL YEAR[(P)] TO MONTH</code>	-----	-----	<code>INTERVAL[fields] [(p)]</code> Fields: YEAR, MONTH, DAY, HOUR, etc.	<code>YEAR</code>

Observar nuevamente, para el caso de Oracle, se emplea el tipo de dato `DATE` para representar fechas, tiempo o ambas.

4.4.5.2. BOOLEAN

- Se agrega al estándar SQL 2003. Posibles valores para este tipo de dato: `TRUE`, `FALSE`.
- Oracle, DB2, SQL server no tienen este tipo de dato. Para el caso de Oracle se emplea `NUMBER(1,0)`. Típicamente 1 significa `TRUE`, 0 significa `FALSE`.

- En MySQL: BOOL, BOOLEAN, TINYINT (1)
- En PostgreSQL: BOOLEAN

4.4.5.3. ROWID

Como se revisó en temas anteriores, ROWID es exclusivo de Oracle, se emplea para almacenar una dirección única para cada registro de una tabla que se crea para localizar los datos de una forma directa.

4.4.5.4. UROWID

Similar a ROWID, solo que este se emplea para tablas indexadas.

4.4.5.5. BFILE

Tipo dato exclusivo de Oracle, se emplea para leer datos binarios que están almacenados fuera de la base de datos (se considera una especie de puntero hacia la ubicación física del archivo).

4.4.5.6. XML:

Tipo de dato empleado para almacenar documentos XML. Permite el manejo y explotación directa del documento a través de SQL en combinación de lenguaje empleado para navegar sobre los elementos de un documento XML.

4.5. MODELADO DE CASOS DE ESTUDIO

En esta sección se revisará el modelado de casos de estudio empleando los conceptos vistos en este capítulo. El objetivo es la creación de un modelo de datos que cumpla con las reglas de negocio especificadas en cada caso.



4.5.1. Base de datos para una empresa que administra colegios.

Las siguientes reglas de negocio resultaron de las entrevistas con los dueños de una empresa que administra colegios particulares la cual ha solicitado el diseño de una base de datos.

Ejercicio en clase 5:



- Generar una lista de entidades candidatas, realizar el análisis correspondiente para descartar falsas entidades.
 - Considerar el modelo ER mostrado a final del enunciado. Analizar los elementos de diseño, escribir los nombres de las entidades según corresponda.
 - Realizar el diseño lógico empleando para ello la transformación del modelo E/R obtenido en el punto anterior a un modelo relacional con notación crow's foot.
- Cada colegio está formado por varias escuelas. De cada colegio se requiere almacenar su nombre.
 - Para cada escuela se requiere almacenar el nombre y su clave. Cada escuela es administrada por un director el cual a su vez es un profesor. Cada director puede administrar una sola escuela. Para un profesor no es requisito que administre una escuela.
 - Cada escuela está formada por varios departamentos. Se requiere almacenar el nombre del departamento.
 - Cada departamento está dividido en un conjunto de áreas funcionales. Se desea almacenar los nombres de las áreas funcionales del departamento.
 - Cada departamento ofrece cursos; otros departamentos se clasifican, por ejemplo, como departamentos de investigación por lo que no ofrecen cursos. Se debe almacenar el tipo de departamento ("de investigación", o "regular"). Cuando se registra el curso se debe indicar en

- nombre del curso, el cupo máximo, número de alumnos inscritos y el departamento al que pertenece.
- Cada curso se imparte en varias clases a la semana. Para cada clase se indica la hora inicio, hora fin, el día de la semana y el salón (un mismo curso puede impartirse en varios salones).
 - Para cada curso, en una semana se pueden dar como mínimo una clase, y como máximo 5 clases. El curso lo imparte un único profesor.
 - El colegio requiere guardar los siguientes datos de cada salón: clave de 5 caracteres, y cupo máximo. No en todos los salones se imparten cursos.
 - Para contar con una mejor comprensión de un curso, en algunos casos se recomienda tomar un curso previo. Para dichos casos se requiere asociar el curso recomendado.
 - Cada profesor pertenece a un departamento. Uno de estos profesores es el encargado de administrarlo (Jefe de departamento). Los datos que se registran de un profesor son: Nombre, apellidos, y RFC. Algunos profesores no imparten cursos.
 - Un estudiante puede estar inscrito en 1 y hasta en 6 cursos en un mismo periodo. Cada curso puede tener como mínimo 5 estudiantes, y como máximo 35. Al final del periodo se registra la calificación del estudiante. Se requiere almacenar nombre, apellidos del estudiante y número de matrícula. El estudiante debe proporcionar al menos un correo electrónico como medio de contacto.
 - Cada estudiante pertenece a una carrera. El colegio cuenta con un catálogo de carreras en el que se almacena clave de la carrera, nombre de la carrera, y total de créditos.
 - Cada estudiante puede tener un asesor, el cual es un profesor. Solo aquellos profesores que así lo deseen pueden asesorar hasta 5 estudiantes.
 - Cada clase se imparte en un salón, y cada salón se encuentra localizado en un edificio perteneciente a la escuela. Se requiere almacenar los datos del edificio: Clave de 2 letras, y descripción. No en todos los edificios se imparten cursos.
 - Cada año un asesor puede recibir varios bonos por su buen desempeño. Se requiere registrar la fecha de pago, el monto del bono y un número consecutivo (folio de pago) que indica el número de pago. El folio está formado por 5 dígitos. Ejemplo: 00001 para el primer pago, 0002 para el segundo pago, etc.

