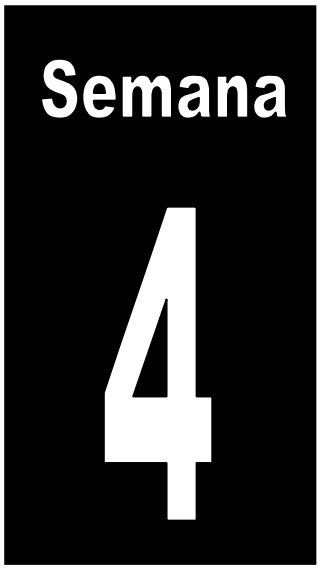


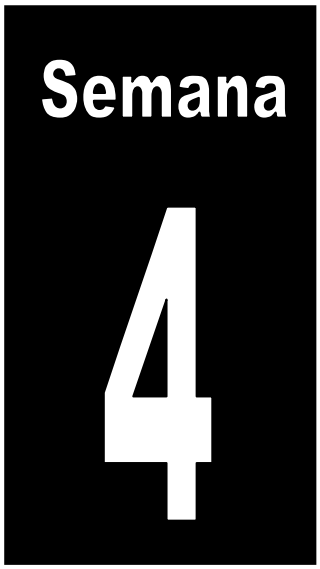
CONTENIDO:

- ✓ Conceptos.
- ✓ Datos Atómicos
- ✓ Dependencia Funcional.
- ✓ Tipos
- ✓ Normalización.
- ✓ Forma Normal 1
- ✓ Forma Normal 2
- ✓ Forma Normal 3
- ✓ Herramienta Case ERWIN 7.1
- Modelo Lógico Físico
- Optimización de tipos de datos
- Representación de un DER en un modelo Relacional (ERWIN)
- Ingeniería Directa / Reversiva
- ✓ Ejercicios.

MODELO RELACIONAL

El modelo relacional para la gestión de una base de datos es un modelo de datos basado en la lógica de predicado y en la teoría de conjuntos. Es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Tras ser postuladas sus bases en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos.

Su idea fundamental es el uso de «relaciones». Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados «tuplas». Pese a que ésta es la teoría de las bases de datos relacionales creadas por Edgar Frank Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar, esto es, pensando en cada relación como si fuese una tabla que está compuesta por registros (cada fila de la tabla sería un registro o tupla), y columnas (también llamadas campos).



CONTENIDO:

- ✓ Conceptos.
- ✓ Datos Atómicos
- ✓ Dependencia Funcional.
- ✓ Tipos
- ✓ Normalización.
- ✓ Forma Normal 1
- ✓ Forma Normal 2
- ✓ Forma Normal 3
- ✓ Herramienta Case ERWIN 7.1
- Modelo Lógico Físico
- Optimización de tipos de datos
- Representación de un DER en un modelo Relacional (ERWIN)
- Ingeniería Directa / Reversiva
- ✓ Ejercicios.

MODELO RELACIONAL

El modelo relacional para la gestión de una base de datos es un modelo de datos basado en la lógica de predicado y en la teoría de conjuntos. Es el modelo más utilizado en la actualidad para modelar problemas reales y administrar datos dinámicamente. Tras ser postuladas sus bases en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos.

Su idea fundamental es el uso de «relaciones». Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados «tuplas». Pese a que ésta es la teoría de las bases de datos relacionales creadas por Edgar Frank Codd, la mayoría de las veces se conceptualiza de una manera más fácil de imaginar, esto es, pensando en cada relación como si fuese una tabla que está compuesta por registros (cada fila de la tabla sería un registro o tupla), y columnas (también llamadas campos).

➢ CONCEPTOS

✓ DATOS ATOMICOS

Las Bases de Datos relacionales tienen en la estructura de sus tablas en realidad, datos atómicos (es así como debe de ser). Un dato atómico es aquel que no puede descomponerse en dos o más datos simples, es decir, son indivisibles en sus valores. Los datos atómicos son opuestos a los multivaluados, que pueden ser descompuestos en otros tipos de datos no atómicos. Un atributo multivaluado tiene valores de dominio con características propias (atributos propios).

Veamos un ejemplo, tenemos la siguiente tabla:

Personas (nombre, apellido, fecha_nacimiento, sexo, estado_civil)

Las tuplas en una relación son un conjunto en el sentido matemático del término, es decir una colección no ordenada de elementos diferentes. Para distinguir una tupla de otra, se recurre al concepto de "llave primaria", o sea a un conjunto de atributos que permiten identificar unívocamente una tupla en una relación. Naturalmente, en una relación puede haber más combinaciones de atributos que permitan identificar unívocamente una tupla ("llaves candidatas"), pero entre éstas se elegirá una sola para utilizar como llave primaria. Los atributos de la llave primaria no pueden asumir el valor nulo (que significa un valor no determinado), en tanto que ya no permitirían identificar una tupla concreta en una relación. Esta propiedad de las relaciones y de sus llaves primarias está bajo el nombre de integridad de las entidades (entity integrity).

A menudo, para obtener una llave primaria "económica", es decir compuesta de pocos atributos fácilmente manipulables, se introducen uno o más atributos ficticios, con códigos identificativos unívocos para cada tupla de la relación.

Cada atributo de una relación se caracteriza por un nombre y por un dominio. El dominio indica qué valores pueden ser asumidos por una columna de la relación. A menudo un dominio se define a través de la declaración de un tipo para el atributo (por ejemplo diciendo que es una cadena de diez caracteres), pero también es posible definir dominios más complejos y precisos. Por ejemplo, para el atributo "sexo" de nuestra relacion "Personas" podemos definir un dominio por el cual los únicos valores válidos son 'M' y 'F'; o bien por el atributo "fecha_nacimiento" podremos definir un dominio por el que se consideren válidas sólo las fechas de nacimiento después del uno de enero de 1960, si en nuestra base de datos no está previsto que haya personas con fecha de nacimiento anterior a esa. El DBMS se ocupará de controlar que en los atributos de las relaciones se incluyan sólo los valores permitidos por sus dominios. Característica fundamental de los dominios de una base de datos relacional es que sean "atómicos", es decir que los valores contenidos en las columnas no se puedan separar en valores de dominios más simples. Más formalmente se dice que no es posible tener atributos multivalor (multivalued). Por ejemplo, si una característica de las personas en nuestra base de datos fuese la de tener uno o más hijos, no sería posible escribir la relación Personas de la siguiente manera:

Personas (nombre, apellido, fecha_nacimiento, sexo, estado_civil, hijos)

En efecto, el atributo hijos es un atributo no-atómico, bien porque una persona puede tener más de un hijo o porque cada hijo tendrá diferentes características que lo describen. Para representar estas entidades en una base de datos relacional hay que definir dos relaciones:

Personas (*número_persona, nombrepers, apellidopers, fecha_nacimiento, sexo, estado_civil)

Hijos(*número_hijo,número_persona, *nombrehijo, apellidohijo, edad, sexo)

En las relaciones precedentes, los asteriscos (*) indican los atributos que componen sus llaves primarias. Nótese la introducción en la relación Personas del atributo número_persona, a través del cual se asigna a cada persona un identificativo numérico unívoco que se usa como llave primaria. Estas relaciones contienen sólo atributos atómicos. Si una persona tiene más de un hijo, éstos se representarán en tuplas diferentes de la relación Hijos. Las diferentes características de los hijos las representan los atributos de la relación Hijos. La unión entre las dos relaciones está constituida por los atributos número_persona que aparecen en ambas relaciones y que permiten que se asigne cada tupla de la relación hijos a una tupla concreta de la relación Personas. Más formalmente se dice que el atributo número_persona de la relación Hijos es una llave externa (foreign key) hacia la relación Personas. Una llave externa es una combinación de atributos de una relación que son, a su vez, una llave primaria para otra relación. Una característica fundamental de los valores presentes en una llave externa es que, a no ser que no sean null, tienen que corresponder a valores existentes en la llave primaria de la relación a la que se refieren. En nuestro ejemplo, esto significa que no puede existir en la relación Hijos una tupla con un valor del atributo número_persona sin que también en la relación Personas exista una tupla con el mismo valor para su llave primaria. Esta propiedad va bajo el nombre de integridad referencial

En realidad una persona puede tener uno o varios hijos, y un por lo tanto un hijo tendrá uno o dos padres, esto sería así…

NRO_PERSONA	NOMBREPERS	APELLIDOPERS	FECHA_NACI	SEXO	EST_CIVIL
PE0001	ARTURO	FLORIAN	12/10/1970	M	C
PE0002	VERONICA	DIAZ	02/05/1976	F	C
PE0003	LUIS	GUEVARA	20/10/1980	M	S

NRO_HIJO	NOMBREHIJO	APELLIDOHIJO	EDAD	SEXO
HJ0001	CESAR	FLORIAN DIAZ	15	M
HJ0002	LUCERO	FLORIAN DIAZ	20	F
HJ0003	CARLOS	GUEVARA CHAVEZ	16	M

NRO_PERSONA	NRO_HIJO	APELLIDOPERS	APELLIDOPERS
PE0001	HJ0001	FLORIAN	FLORIAN DIAZ
PE0001	HJ0002	FLORIAN	FLORIAN DIAZ
PE0002	HJ0001	DIAZ	FLORIAN DIAZ
PE0002	HJ0002	DIAZ	FLORIAN DIAZ
PE0003	HJ0003	GUEVARA	GUEVARA CHAVEZ

✓ DEPENDENCIA FUNCIONAL (DF)

Hay veces en que los atributos están relacionados entre sí de manera más específica que la de pertenecer a una misma relación. Hay veces en que es posible determinar que un atributo depende de otro funcionalmente, como si existiera una función f en el ”mundo”, tal que t[A] = f(t[B]).

La función se anotaría como f : A ! B, pero como f es desconocida (o sino B sería un atributo derivado), sólo nos quedamos con A ! B, la dependencia funcional, que se lee: ”A determina B”.

Formalmente, X ! Y en R se cumple si y sólo si 8s, t 2 R, s[X] = t[X]) s[Y] = t[Y]. Esto es análogo a las funciones: 8x1, x2 2 X, x1 = x2) f(x1) = f(x2), con f : X ! Y...

UTILIDAD EN EL DISEÑO DE BASES DE DATOS

Las dependencias funcionales son restricciones de integridad sobre los datos. Conocer las dependencias funcionales en el momento del diseño de la base de datos permite crear mecanismos para evitar la redundancia (y los potenciales problemas de integridad que eso conlleva) y mejorar la eficiencia.

¿COMO OBTENER LAS DEPENDENCIAS FUNCIONALES?

La mejor manera de obtenerlas es a través del conocimiento del problema. Es lo más disponible en la fase de diseño de una base de datos. Sin embargo, esto puede tornarse bastante difícil, como en el caso del vehículo (honestamente, esto puede ocurrir cuando la base de datos modela conocimiento técnico, que escapa al sentido común).

Otra manera, relacionada con el ejemplo anterior, es comprobar dependencias funcionales sobre una gran población de datos usando la definición.

EJEMPLO

Una dependencia funcional es una relación de dependencia entre uno o más atributos. Por ejemplo si conocemos el valor FechaDeNacimiento podemos conocer el valor de Edad.

Las dependencias funcionales se escriben utilizando una flecha, de la siguiente manera:

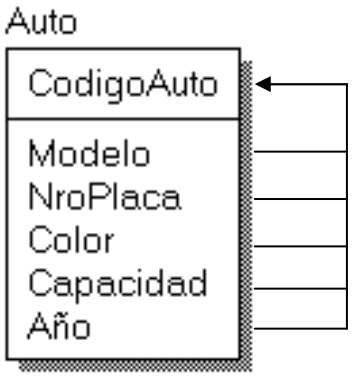
FechaDeNacimiento -> Edad

Aquí a FechaDeNacimiento se le conoce como un determinante. Se puede leer de dos formas FechaDeNacimiento determina a Edad o Edad es funcionalmente dependiente de FechaDeNacimiento. De la normalización (lógica) a la implementación (física o real) puede ser sugerible tener éstas dependencias funcionales para lograr mayor eficiencia en las tablas.

VEAMOS OTRO EJEMPLO:

Tenemos la entidad

Entidad Auto (CodigoAuto, Modelo, NroPlaca, Color, Capacidad, Año)



Todos los atributos de esta entidad tienen dependencia funcional con la clave principal de la entidad. Podemos decir entonces que conociendo el código del auto asignado, podemos conocer los demás detalles de cada auto.

✓ DEPENDENCIA FUNCIONAL TRANSITIVA

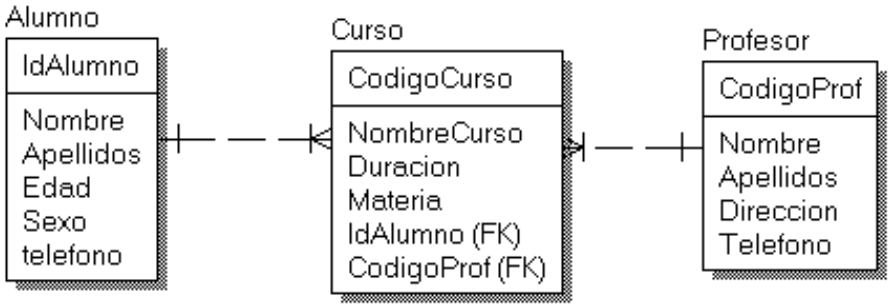
Supongamos que en una relación en la que los estudiantes solo pueden estar matriculados en un solo curso y supongamos que los profesores solo pueden dar un curso.

ID_Estudiante -> Curso_Tomando

Curso_Tomando -> Profesor_Asignado

ID_Estudiante -> Curso_Tomando -> Profesor_Asignado

Entonces tenemos que ID_Estudiante determina a Curso_Tomando y el Curso_Tomando determina a Profesor_Asignado, indirectamente podemos saber a través del ID_estudiante el Profesor_Asignado. Entonces en la relación tenemos una dependencia transitivaentre alumno y profesor.



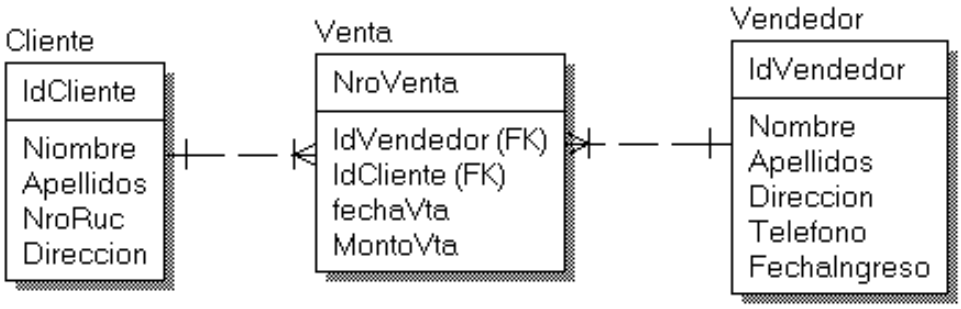
VEAMOS OTRO EJEMPLO:

IdCliente -> Venta realizada

Venta realizada -> Vendedor encargado

IdCliente -> Venta realizada -> Vendedor encargado

Entonces tenemos que el IdCliente determina a quién se le hizo la venta, y la venta realizada determina qué vendedor llevó a cabo la venta. Entonces en la relación tenemos una dependencia transitiva entre el cliente y el vendedor.



➢ NORMALIZACION DE DATOS

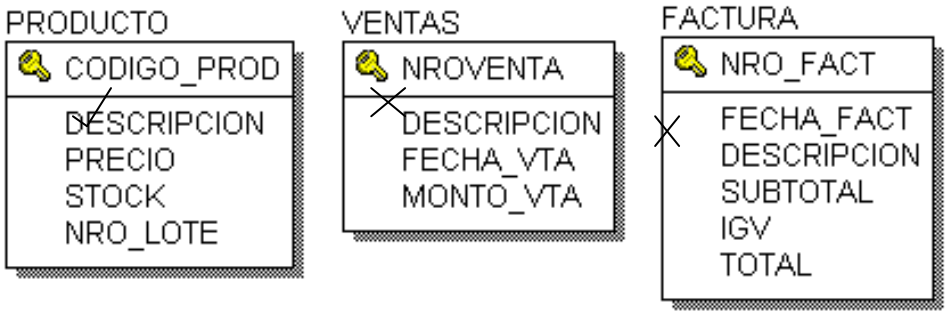
La normalización de datos es el proceso de transformación de las entidades complejas en entidades simples, siempre que se normaliza se crean por lo menos dos entidades nuevas. Esta es otra forma de encontrar las entidades del proceso de negocio, por medio de los documentos que son los que se puede normalizar, podemos diseñar los modelos de datos.

¿CUÁL ES EL OBJETIVO DE LA NORMALIZACIÓN?

- El objetivo principal es el de evitar la redundancia de los datos en las tablas, mejorar u optimizar el diseño del sistema para brindar una mejor performance de los procesos. Solo un diseño normalizado puede garantizar que nuestro sistema cumple con los requisitos de los usuarios.
- Además Evitar problemas de actualización de los datos en las tablas.
- Proteger la integridad de los datos.

¡EVITAR LA REDUNDANCIA!

Ejemplo:



En el proceso de normalizar datos, nos vamos a encontrar con que existen procedimientos para lograr la optimización de nuestro diseño de datos, estos procedimientos son conocidos como formas normales, las cuales a su vez tienen sus propias características, veamos cada uno de ellos.

Existen 5 formas normales, de las cuales podemos decir que cumplidas las 3 primeras formas normales tendremos un diseño adecuado de datos.

✓ 1ª FORMA NORMAL (1FN)

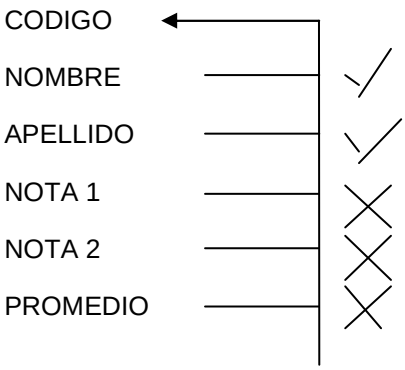
Una relación se encuentra en primera forma normal si y sólo si sus atributos son atómicos, es decir son no descomponibles. El objetivo de la 1FN es hallar aquellos los atributos que tienen dependencia funcional directamente con la PK.

DEPENDENCIA FUNCIONAL (DF)

Es la relación que existe entre los atributos no primos (no claves) y la clave primaria de la entidad.

Ejemplo:

Alumno (código, nombre, apellido, nota1, nota2, promedio)



Diremos entonces: El campo Nombre y Apellido tienen DF con la clave Código.

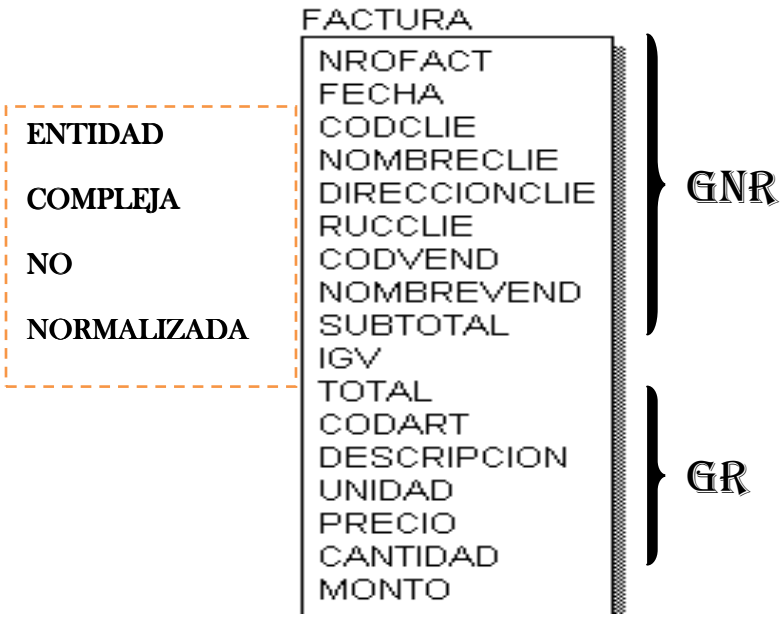
Nota1, Nota2 y Promedio no tienen DF con la clave Código.

Sólo aquellos atributos que pertenezcan a las características propias de la entidad, tienen dependencia funcional con la PK, sin no dependen funcionalmente de la clave principal, entonces no pertenecen a la entidad.

PASOS DE LA 1FN:

1. Identificar los grupos repetitivos y no repetitivos (GR, GNR).
2. Remover los GR y crear una nueva entidad con ellos.
3. Llevar la clave a la nueva entidad.

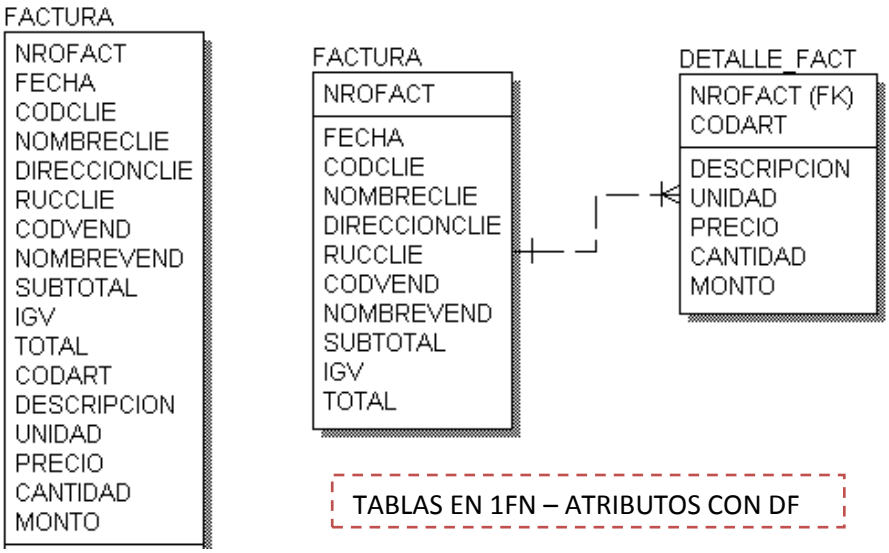
Para explicar las formas normales, utilizaremos una factura de venta la cual iremos descomponiendo paso a paso.
Tenemos una factura cuyo modelo es simple, una típica factura de una bodega o una farmacia por ejemplo, debemos ubicar todos aquellos datos que representan información importante para el negocio, las listamos para luego proceder a normalizarlo. Aquí la lista de atributos encontrados…



Veamos la factura en forma de tabla:

NROFACT	FECHA	CODCLIE…	CODVEND….	CODART	DESCRIPCION…
F0001	17/10/07	C0001	V0001	A0001	Televisor
F0001	17/10/07	C0001	V0001	A0002	Plancha
F0001	17/10/07	C0001	V0001	A0003	Cocina

GNR **GR**



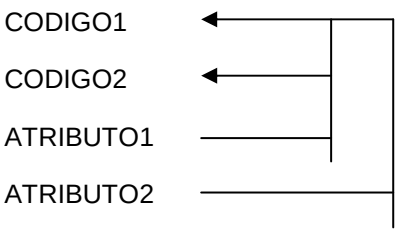
✓ 2ª FORMA NORMAL (2FN)

Una relación estará en 2FN si y sólo si está en 1FN y además se cumple que los atributos no primos tienen dependencia funcional completa con respecto a la clave concatenada o compuesta.

DEPENDENCIA FUNCIONAL COMPUESTA (DFC)

Es la relación que existe entre los atributos no primos (no claves) y la clave concatenada, una clave concatenada es aquella que está compuesta por dos o más atributos claves, la tienen las entidades asociadas y las entidades con relación identificada.

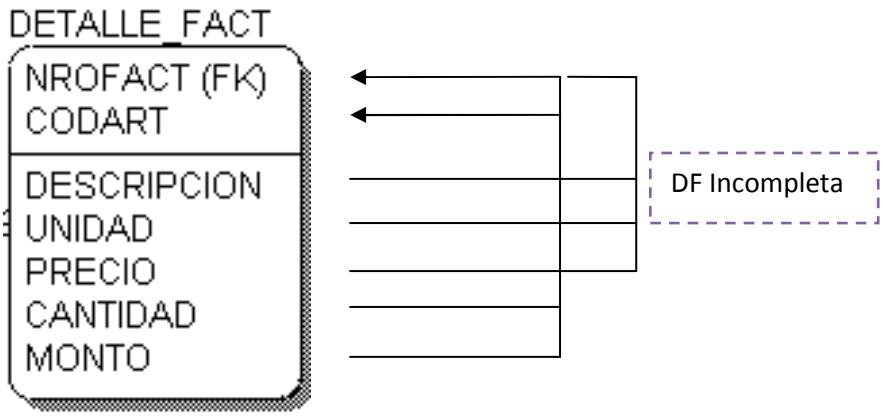
Ejemplo: Una entidad que tiene una clave compuesta.



Diremos: Atributo 1 tiene DFC con ambas claves, Atributo 2 no tiene DFC con ambas claves, entonces remover Atributo 2.

PASOS DE LA 2FN

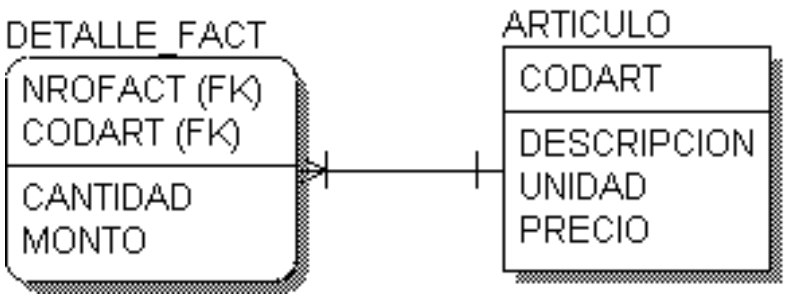
1. Identificar los atributos con dependencia funcional incompleta.
2. Remover los atributos con DF incompleta y crear una nueva entidad.
3. Llevar la clave a la nueva entidad.



Veamos esto en forma de tabla:

NROFACT	CODART	CANTIDAD	MONTO
F0001	A0001	2	400.00
F0001	A0002	3	390.00
F0001	A0003	1	540.00

CODART	DESCRIPCION	UNIDAD	PRECIO
A0001	Televisor	Unid	200.00
A0002	Plancha	Unid	130.00
A0003	Cocina	unid	540.00



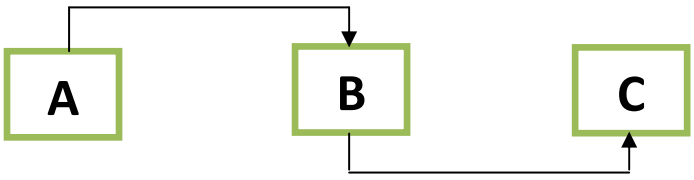
TABLAS EN 2FN – ATRIBUTOS CON DFC

✓ 3ª FORMA NORMAL (3FN)

Una relación estará en 3FN si y sólo si está en 2FN y además existen atributos no claves que dependen de otros atributos no claves de la entidad compleja. Estos atributos no claves tienen relación transitiva con la entidad principal.

- **DEPENDENCIA TRANSITIVA**

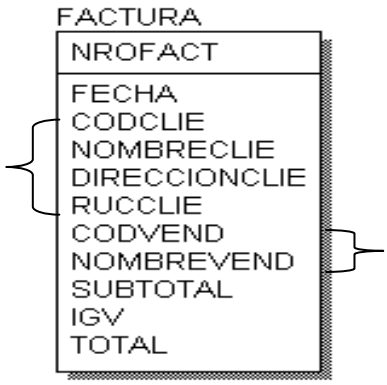
Se refiere a la relación indirecta entre dos o más entidades, esta relación indirecta se da por medio de otra entidad que funge de puente entre ambas.



Diremos entonces que: La entidad A es transitiva a la entidad C, relación indirecta por medio de la entidad B.

PASOS DE LA 3FN

1. Identificar los atributos no claves con DF con otros atributos no claves.
2. Remover los atributos transitivos y crear una nueva entidad.
3. Llevar la clave a la nueva entidad.
4.

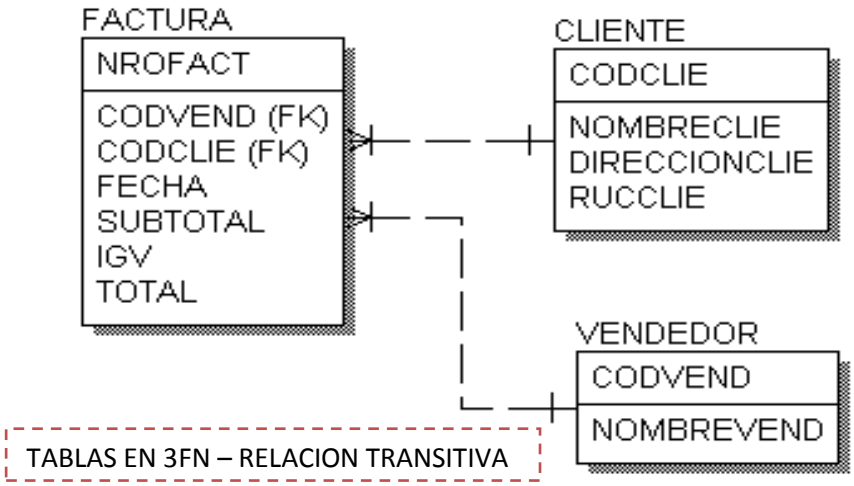


NROFACT	CODCLIE	NOMBREC	DIRECCIONC	RUCC	CODVEND	NOMBVEND
F0001	C0001	Augusto	San Isidro	1235698	V0001	Nelly
F0001	C0001	Pedro	Surco	5698745	V0002	Eduardo
F0001	C0001	María	Miraflores	5630214	V0001	Teresa

Clave Entidad Principal

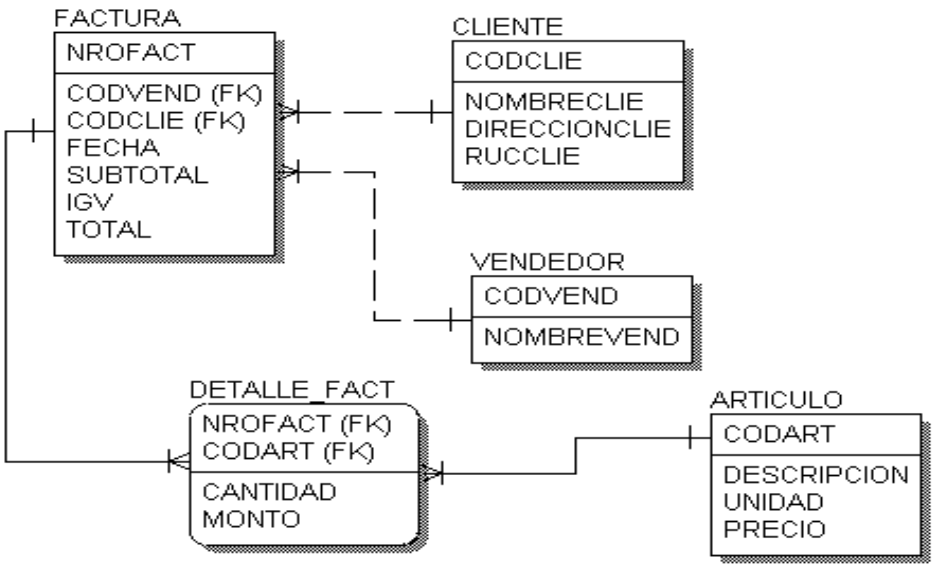
Atributos con DF con atributo no clave

Atributos con DF con atributo no clave



TABLAS EN 3FN – RELACION TRANSITIVA

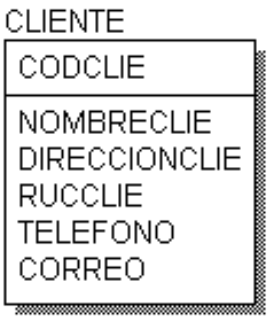
Para que un diseño de datos tenga credibilidad y de suficiente soporte al cumplimiento de requerimiento de los usuarios, se acepta hasta la 3FN, es decir, si el diseño se encuentra normalizado hasta la 3FN entonces cumple con los requisitos del sistema, este ejemplo quedaría así:



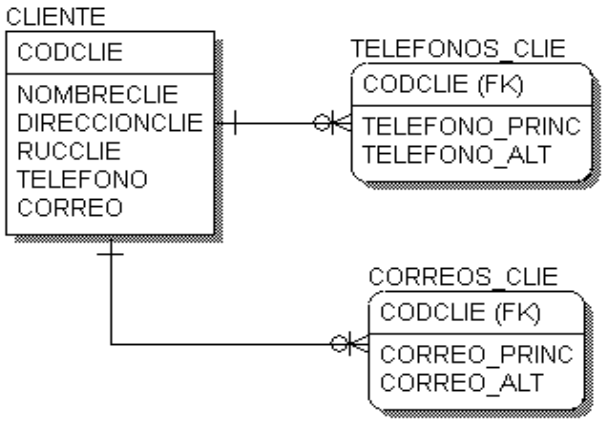
✓ 4ª FORMA NORMAL (4FN)

Una relación está en 4FN si y solo si se encuentra en 3FN y se cumple que no existan dependencias multivaluadas en alguno de los atributos no claves. Un atributo multivaluado es aquel que tiene varios posibles valores para una sola instancia de la entidad.

Por ejemplo: tenemos una entidad Cliente, cada uno de los clientes registrados puede tener varios teléfonos y correos alternativos, entonces estamos ante una dependencia multivaluada entre ambos atributos y el atributo no clave Nombreclie.



Debemos resolver creando dos nuevas entidades con los atributos multivaluados para evitar así la redundancia de datos.



¿QUÉ TAN LEJOS SE DEBE LLEVAR LA NORMALIZACIÓN?

La siguiente decisión es ¿qué tan lejos debe llevar la normalización? La normalización es una ciencia subjetiva. Determinar las necesidades de simplificación depende de nosotros. Si nuestra base de datos va a proveer información a un solo usuario para un propósito simple y existen pocas posibilidades de expansión, normalizar los datos hasta la 3FN quizá sea algo exagerado. Las reglas de normalización existen como guías para crear tablas que sean fáciles de manejar, así como flexibles y eficientes. A veces puede ocurrir que normalizar los datos hasta el nivel más alto no tenga sentido.

¿Se están dividiendo tablas sólo para seguir las reglas o estas divisiones son en verdad prácticas?. Éstas son el tipo de cosas que nosotros como diseñadores de la base de datos, necesitamos decidir, y la experiencia y el sentido común nos pueden auxiliar para tomar la decisión correcta. La normalización no es una ciencia exacta, más bien subjetiva.

Existen seis niveles más de normalización que no se han discutido aquí. Ellos son Forma Normal Boyce-Codd, Cuarta Forma Normal (4NF), Quinta Forma Normal (5NF) o Forma Normal de Proyección-Unión, Forma Normal de Proyección-Unión Fuerte, Forma Normal de Proyección-Unión Extra Fuerte y Forma Normal de Clave de Dominio. Estas formas de normalización pueden llevar las cosas más allá de lo que necesitamos. Éstas existen para hacer una base de datos realmente relacional. Tienen que ver principalmente con dependencias múltiples y claves relacionales.

EN RESUMEN

La normalización es una técnica que se utiliza para crear relaciones lógicas apropiadas entre tablas de una base de datos. Ayuda a prevenir errores lógicos en la manipulación de datos. La normalización facilita también agregar nuevas columnas sin romper el esquema actual ni las relaciones.

Existen varios niveles de normalización: Primera Forma Normal, Segunda Forma Normal, Tercera Forma Normal, Forma Normal Boyce-Codd, Cuarta Forma Normal, Quinta Forma Normal o Forma Normal de Proyección-Unión, Forma Normal de Proyección-Unión Fuerte, Forma Normal de Proyección-Unión Extra Fuerte y Forma Normal de Clave de Dominio. Cada nuevo nivel o forma nos acerca más a hacer una base de datos verdaderamente relacional.

Se discutieron las primeras tres formas. Éstas proveen suficiente nivel de normalización para cumplir con las necesidades de la mayoría de las bases de datos. Normalizar demasiado puede conducir a tener una base de datos ineficiente y hacer a su esquema demasiado complejo para trabajar. Un balance apropiado de sentido común y práctico puede ayudarnos a decidir cuándo normalizar.

✓ DESNORMALIZACIÓN

Se puede definir como el proceso de poner la misma información en varios lugares. Una normalización reduce problemas de integridad y optimiza las actualizaciones, quizás con el costo del tiempo de recuperación. Cuando se pretende evitar esta demora resultado de la combinación de muchas tablas entonces se puede utilizar la desnormalización.

Antes de denormalizar es importante considerar:

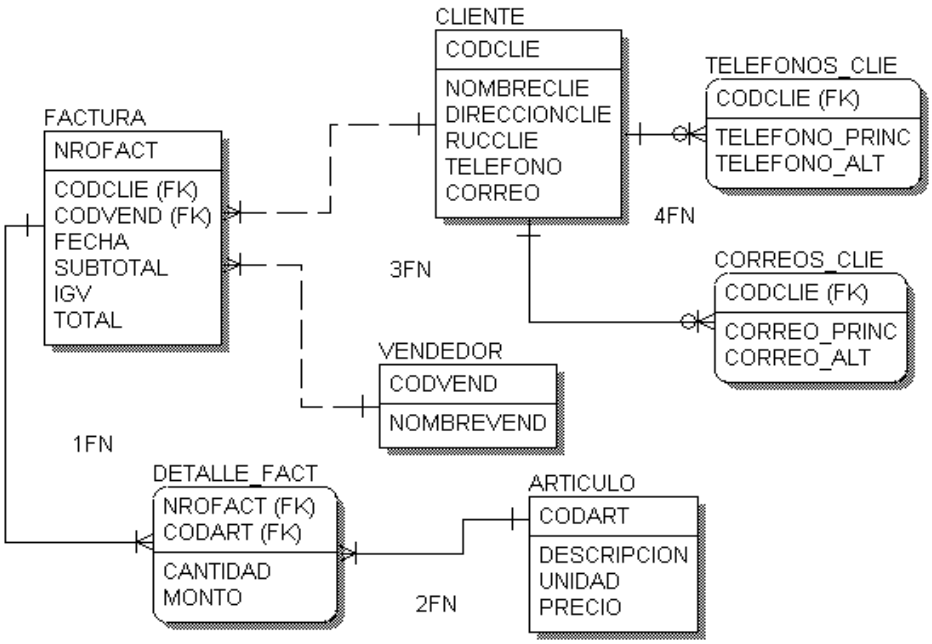
- ¿El sistema puede tener un desempeño aceptable sin la desnormalización?
- ¿Aún con la denormalización el desempeño será siendo malo?
- ¿El sistema será menos confiable debido a la desnormalización?

Candidatos a desnormalización:

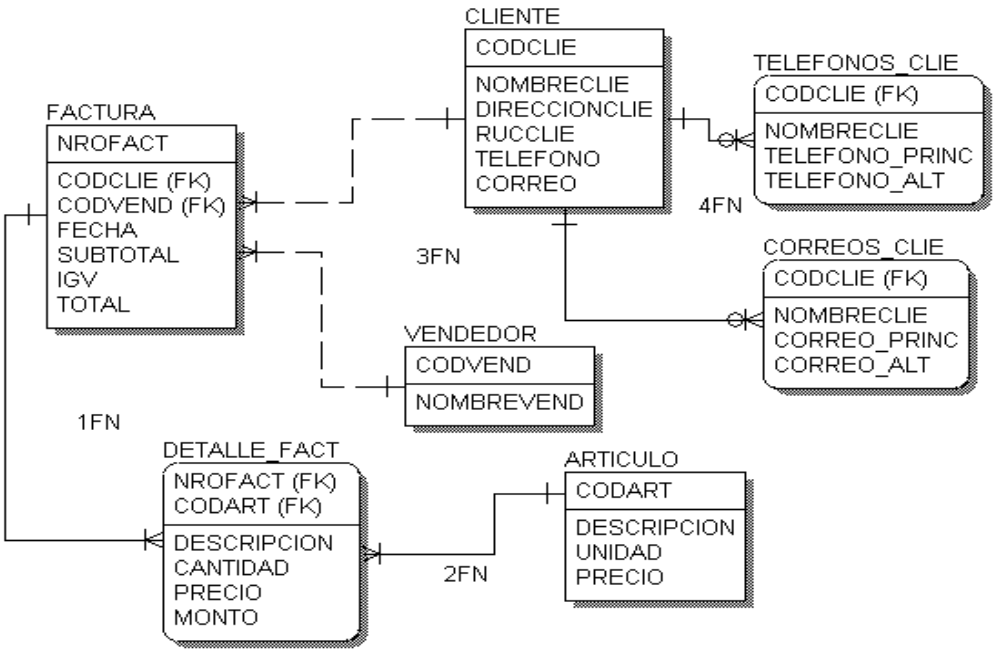
- Numerosas consultas críticas o reportes incluyen datos que incluyen más de una tabla.
- Grupos repetidos de elementos necesitan ser procesados en un grupo en lugar de individualmente.
- Muchos cálculos necesitan realizarse a una o más columnas antes de procesar las consultas.
- Las tablas necesitan ser accesadas de diferentes maneras por diferentes usuarios durante el mismo lapso de tiempo.
- Llaves primarias mal diseñadas que requieren tiempo al usarlas en relaciones.
- Algunas columnas son interrogadas un gran porcentaje del tiempo.

Importante: nunca se realiza un desnormalización en un modelo lógico.

Comparación: Veamos como quedó nuestro ejemplo de Normalización hasta la 4FN:



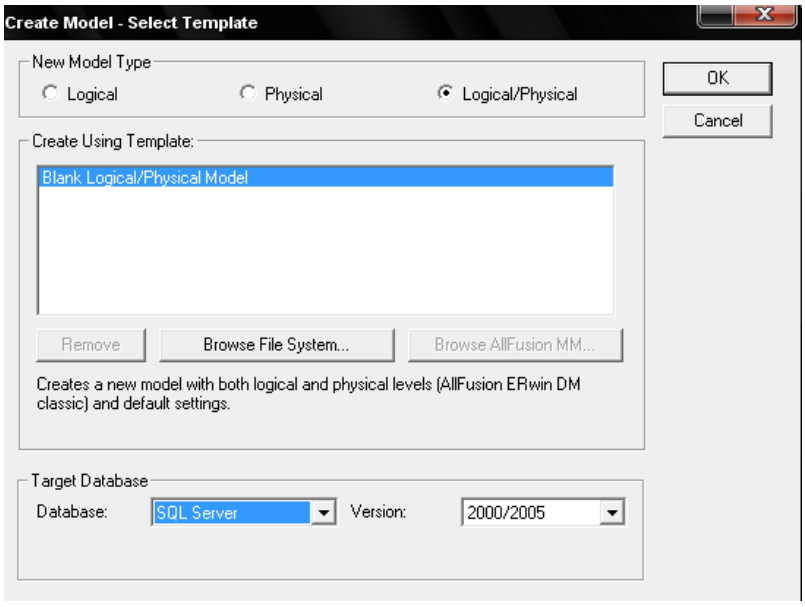
Veamos ahora como debe quedar el mismo diseño con la técnica de Desnormalización:



LABORATORIO # 4

MODELO LOGICO- FISICO

Al iniciar un nuevo archivo tenemos la opción de seleccionar el tipo de modelo a trabajar, bien podemos trabajar solo a nivel de modelo lógico, o sólo a nivel de modelo físico, pero lo recomendable es trabajar ambos modelos, para poder seleccionar un motor de base de datos.



➢ REPRESENTACION DE UN DER EN UN MODELO RELACIONAL

A continuación veremos un caso de estudio en la cual se muestra el modelo lógico y el modelo físico resultante…

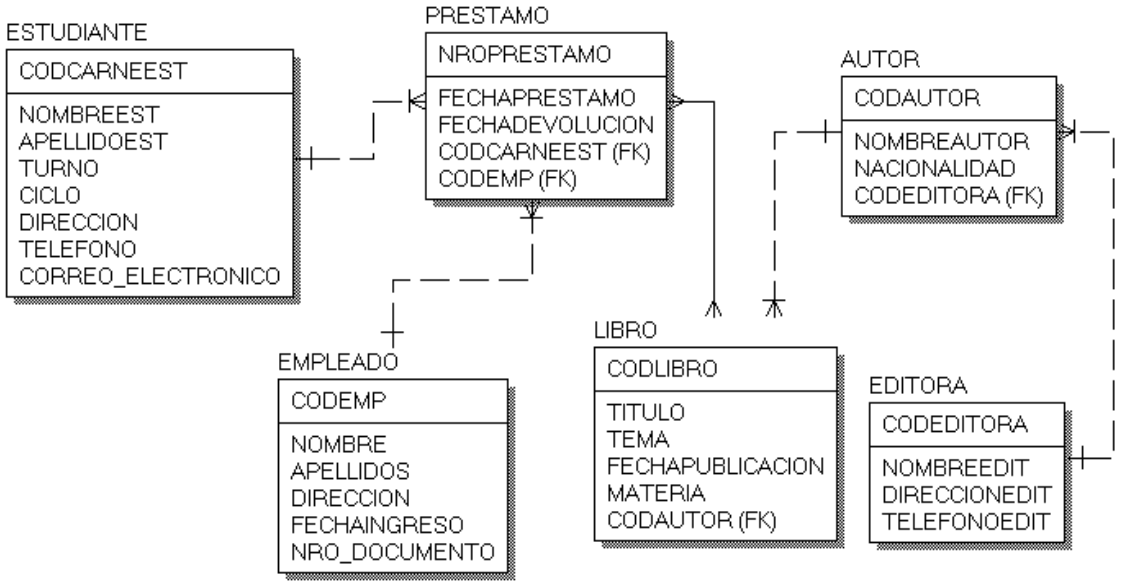
PROCESO DE NEGOCIO BIBLIOTECA

La biblioteca del instituto tecnológico desea implementar un sistema de control de préstamos de libros a los estudiantes, para lo cual se nos brindó la información necesaria.

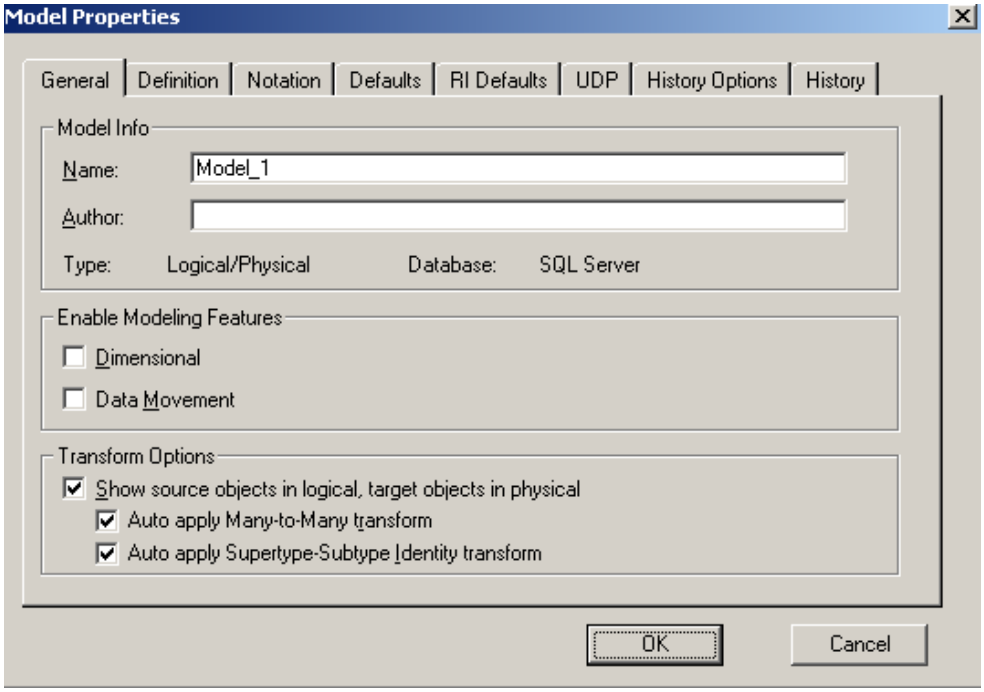
El estudiante debe solicitar el préstamo al empleado encargado de la biblioteca quién entregará el libro en cuestión. El estudiante debe dejar un documento personal para que se le entregue el libro.

Por cada préstamo el estudiante puede solicitar hasta un máximo de tres libros, de distintas especialidades, se desea registrar cada libro clasificándolos según el autor y la editora que la distribuye.

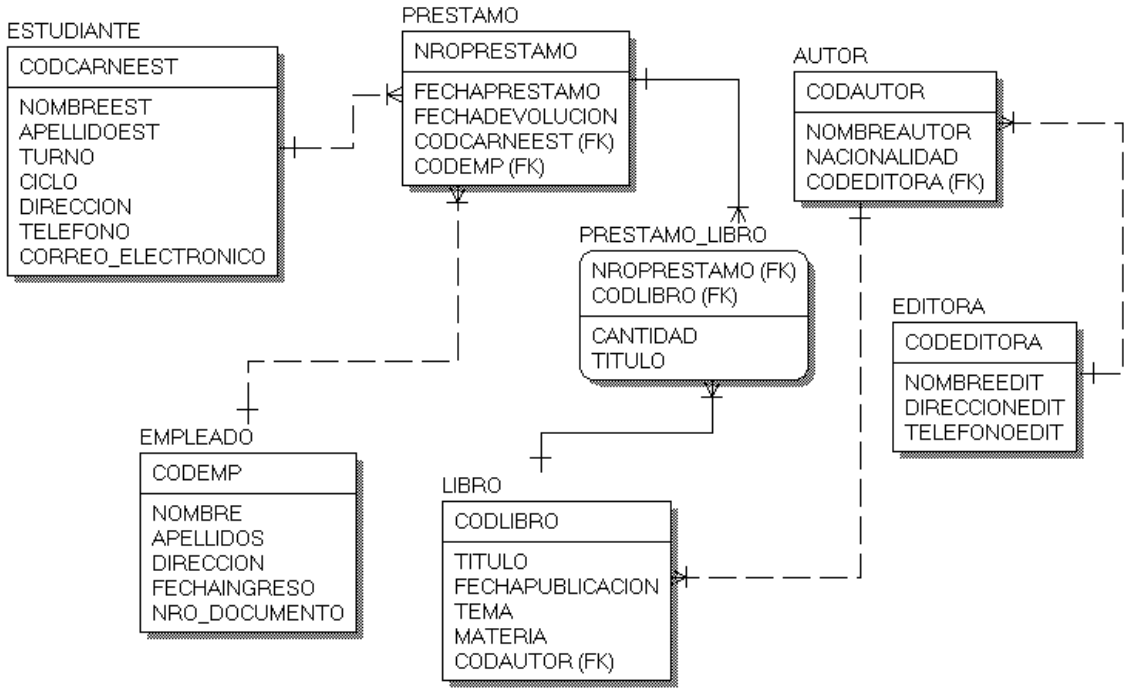
EL MODELO LOGICO



En el modelo físico resolvemos la relación de Muchos a Muchos entre las entidades Libro y Préstamo, para ello no olvidar que debemos activar la opción Auto apply Many-to-Many transform, desde el menú Model – Model Properties, tal como se muestra en la siguiente imagen…

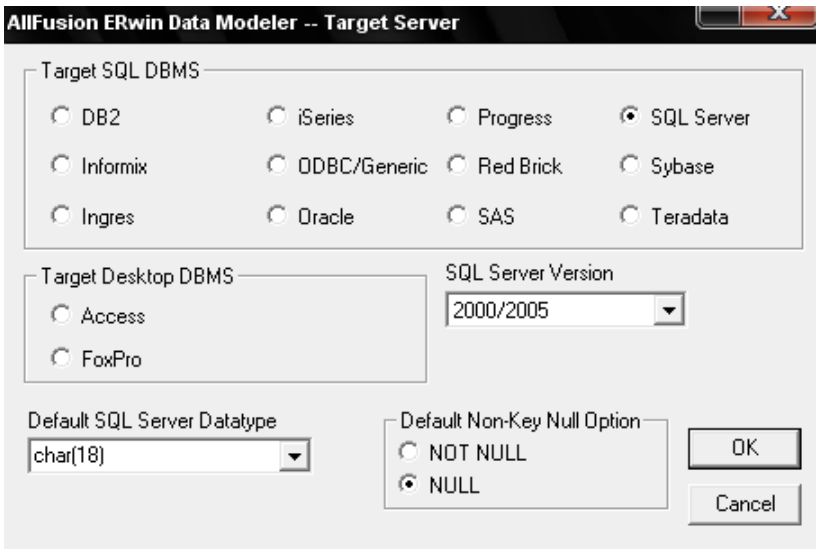


EL MODELO FISICO



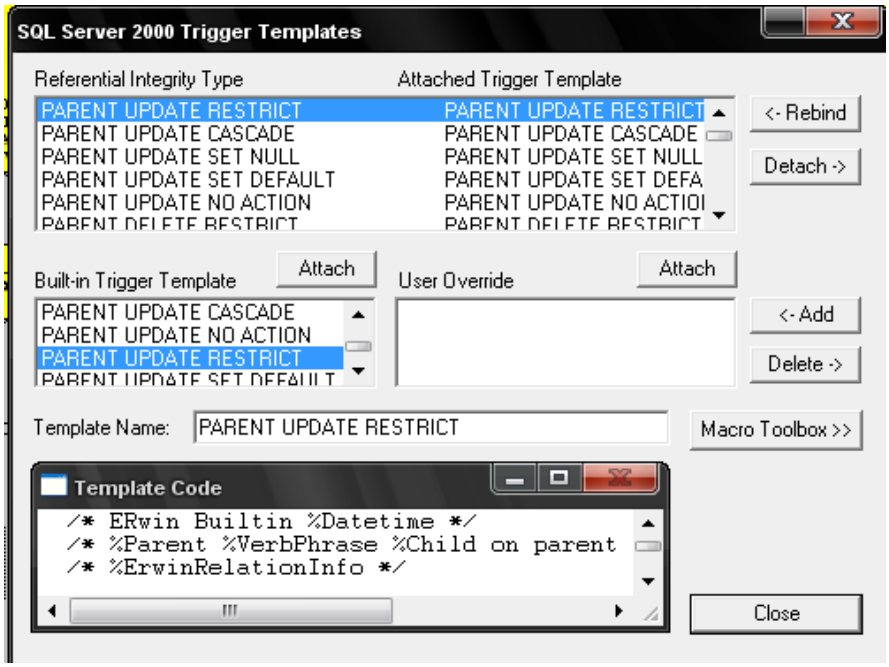
➢ OPTIMIZACION DE TIPO DE DATOS

- ✓ ***SELECCIONAR EL SERVIDOR DE DATOS***
  - ✓ Hacemos clic en el menú Databases – Choose Database.
  - ✓ Aparece la siguiente ventana:

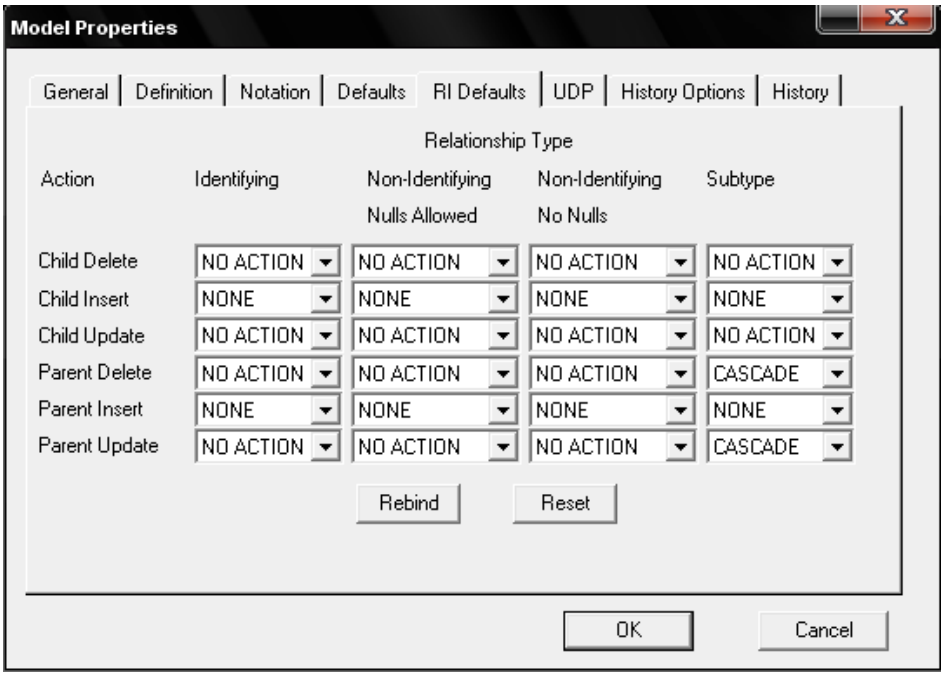


Aquí debemos seleccionar el DBMS con la que nos conectaremos.
Luego la versión del servidor disponible.
El tipo de dato por defecto y los valores no llaves por defecto, en este caso No Null.
Luego de ello le damos clic en Ok.

Ahora debemos chequear las opciones por defecto de Integridad Referencial, para ello nos vamos al menú Database y elegimos RI Trigger Templates..

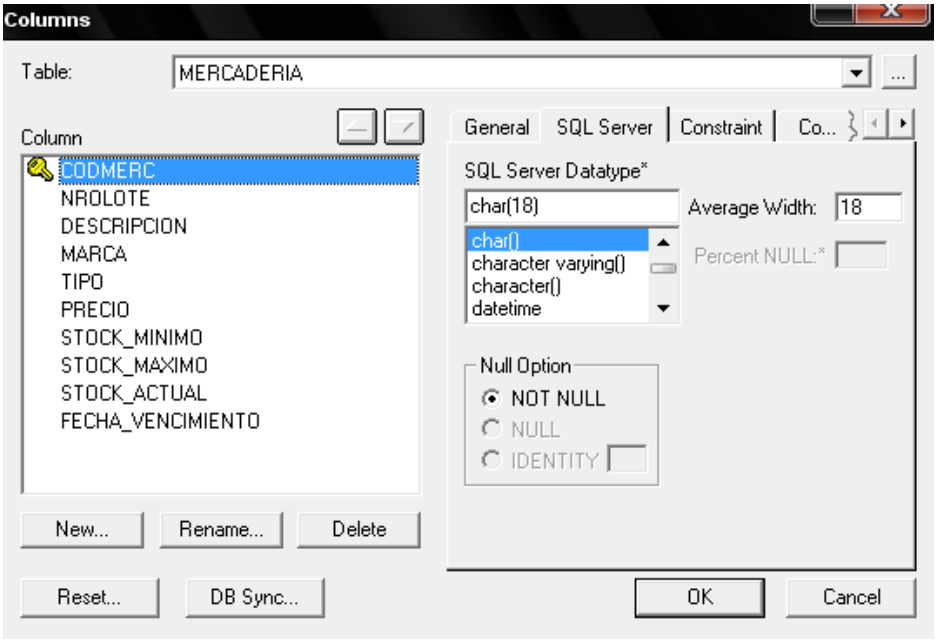


Podemos trabajarlo también desde elmenú Model – Model Properties en lapestana RI Defaults…

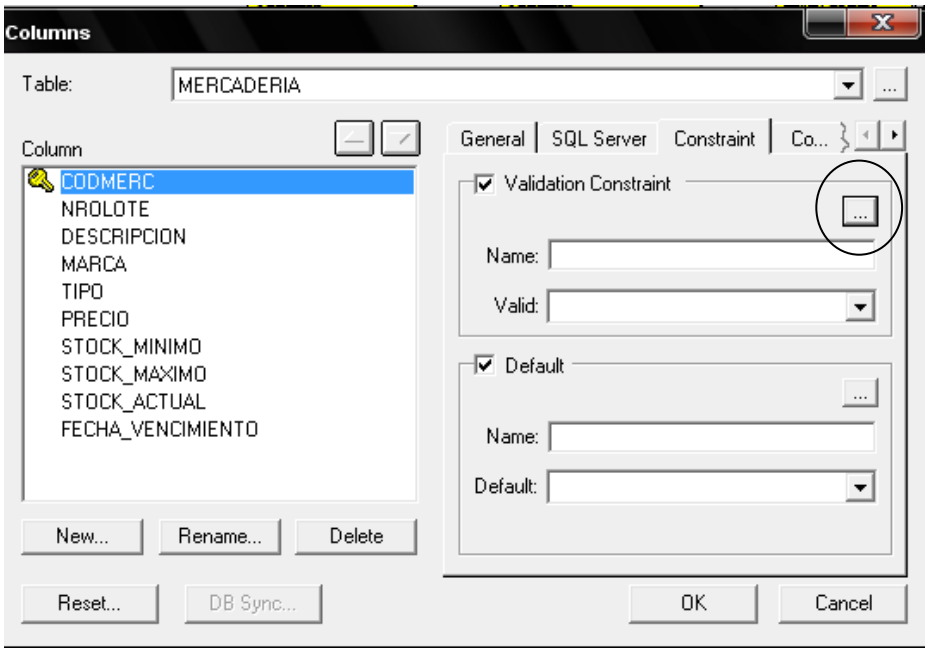


- ✓ **ASIGNANDO PROPIEDADES DE COLUMNAS**

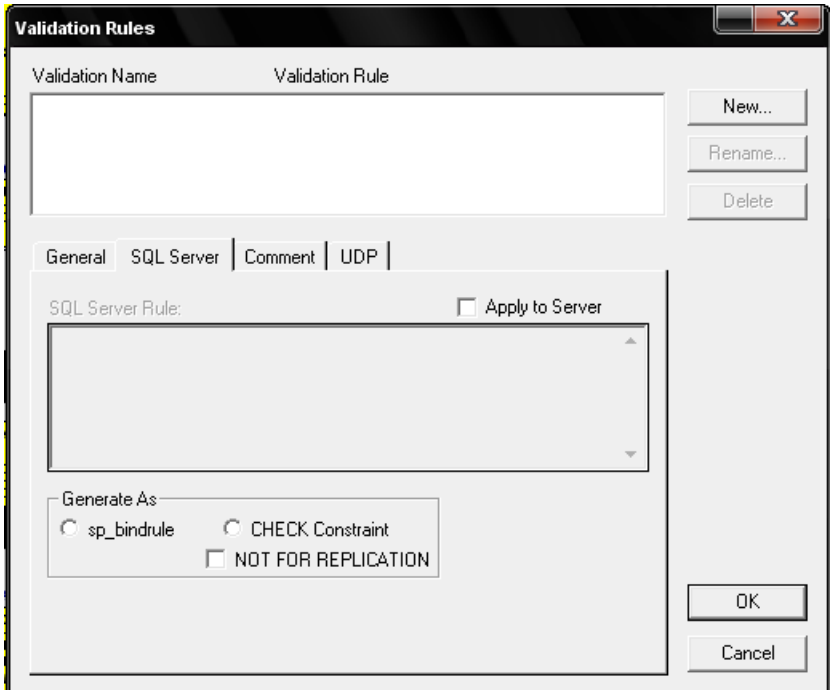
- ✓ Hacemos clic derecho sobre las tablas, elegimos la opción Column.
- ✓ Aparece la siguiente ventana.



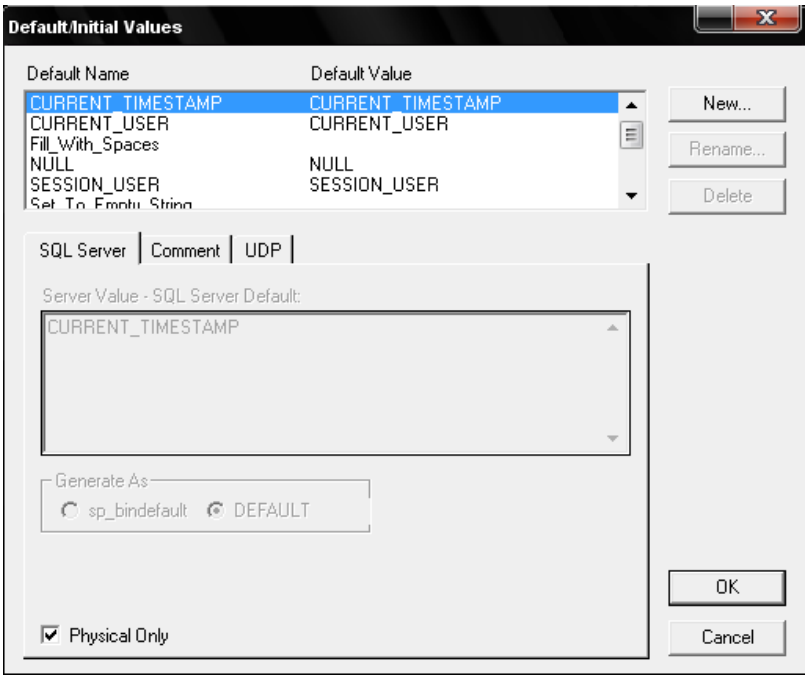
Aquí seleccionamos cada columna y el tipo de dato compatible con el servidor de datos seleccionado, además asignamos la opción de colocar valores nulos a los campos. Podemos asignar una regla de validación a cada campo, desde la pestaña Constraint…



En Validation Constraint hacemos clic en el botón Agregar para poder asignar una regla de validación de datos, aparecerá la siguiente ventana…



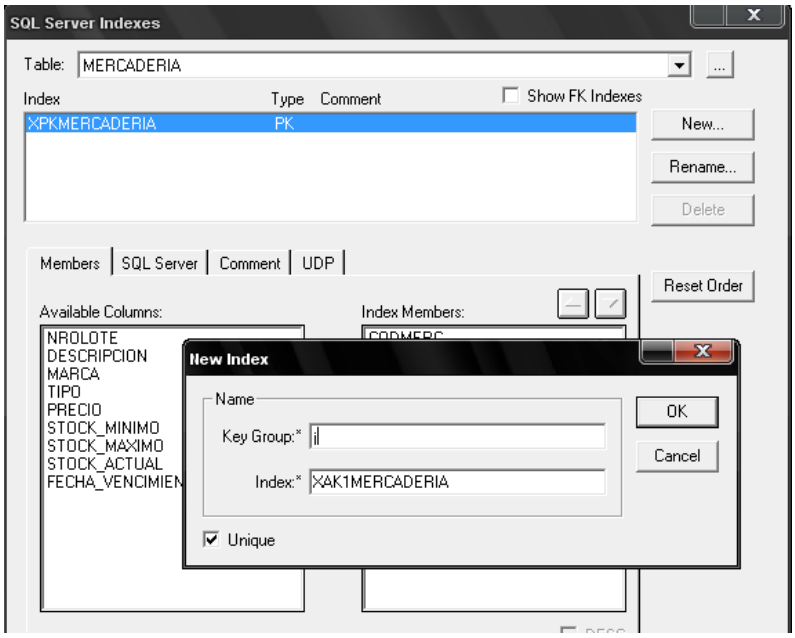
Si lo que deseamos es asignar un valor por defecto, hacemos clic en el botón Agregar del área Default, aparecerá la siguiente ventana en donde tenemos un listado de funciones disponibles…



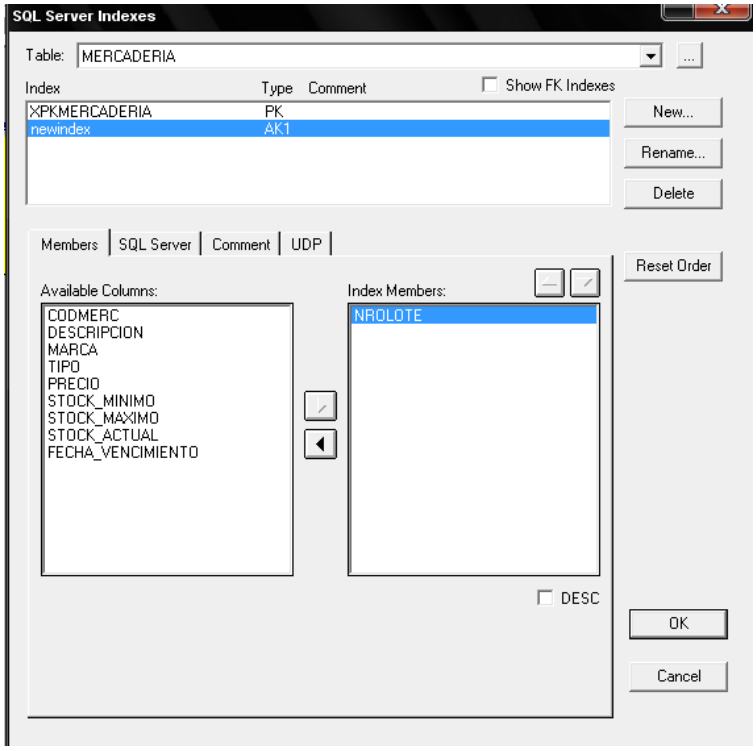
Por ejemplo tenemos la opción de colocar el usuario actual del sistema (CURRENT_USER), o podemos colocar un valor nuevo creándolo desde el botón New…

✓ CREAR INDICES

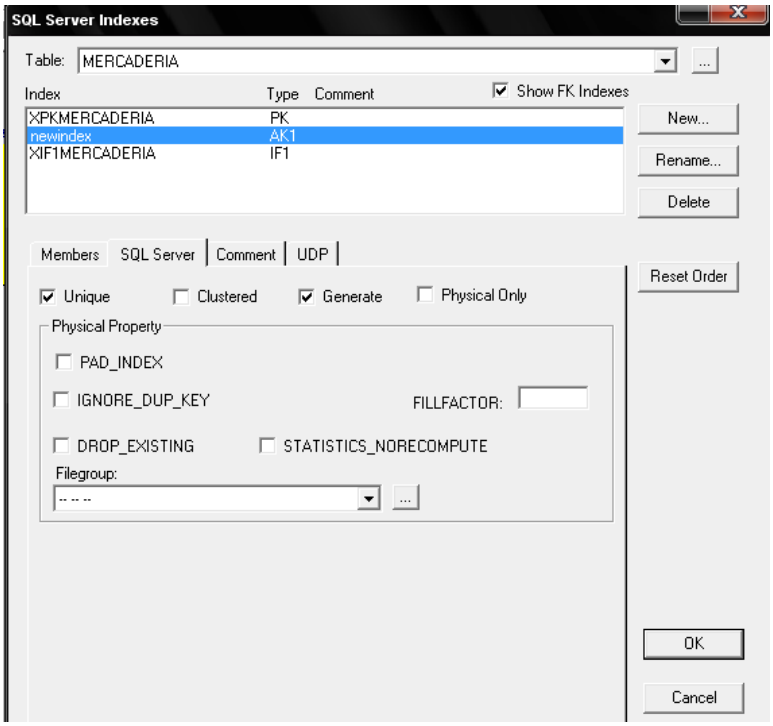
Hacemos clic derecho sobre la tabla, luego elegimos la opción Indexes, en la ventana veremos el único índice creado que es la Llave primaria (Primary Key). Para agregar un nuevo índice damos clic en el botón New…



Le damos un nombre al índice, luego en la ventana principal seleccionamos el o los campos que pertenecerán al índice.



Los índices nos ayudan a crear un patrón de búsquedas para los registros de datos de las tablas, podemos seleccionar los campos asignados como claves alternas como el índice, para ello debemos tener seleccionada la opción Unique en la pestaña SQL Server…



- **INGENIERIA DIRECTA / REVERSIVA**

  (FORWARD ENGINEER / REVERSE ENGINEER)

Ingeniería Directa: Es el proceso de producción del código de una aplicación a partir de sus especificaciones. La ingeniería directa reduce el tiempo de creación de una base de datos ya que será la herramienta de diseño quien genere los códigos DDL dentro del DBMS.

Ingeniería Inversa: Conjunto de tareas destinadas a obtener las especificaciones de un sistema de información partiendo del propio sistema. Es una actividad típica del mantenimiento de equipos lógicos, cuando no existen las especificaciones del diseño del equipo a mantener.

La ingeniería inversa tiene la misión de interpretar las fuentes de los sistemas informáticos. Consiste básicamente en recuperar el diseño de una aplicación a partir del código fuente generado. Esto se realiza mediante herramientas que extraen información de los datos, procedimientos y arquitectura del sistema existente.

La Ingeniería inversa es aplicable a aquellos sistemas cuya documentación es inexistente, demanda grandes cambios en su estructura frecuentemente y contiene bloques de código muy grandes y sin reestructurar.

ERWIN – SQL SERVER

Ca Erwin Data Modeler facilita en gran medida la gestión de modelos para el análisis del negocio y la recogida de requisitos, así como el diseño y la implementación de bases de datos y aplicaciones data warehouse de calidad. Además, permite a los profesionales de las bases de datos que utilicen SQL Server diseñar visualmente y generar objetos gestionados para utilizarlos en el entorno de desarrollo de Microsoft Visual Studio.

Los profesionales de las bases de datos también pueden aprovechar CA ERwin DM para generar sus esquemas de datos en SQL Server y así acelerar la implantación de proyectos Visual Studio.

EJERCICIOS

Realizar el modelo Lógico y Físico en Erwin del siguiente caso de estudio:

PROCESO DE NEGOCIO

FARMACIA

Una farmacia ‘X’ desea implementar un pequeño sistema para el control de su almacén, Cada salida de medicamento hacia ventas debe estar registrado en una orden de venta, sin este documento no podrá salir. Cada medicamento tiene un tipo de lo clasifica, como jarabe, comprimidos, etc. Cada adquisición de nuevos lotes de medicamentos debe estar registrada en una orden de compra, se debe registrar el laboratorio proveedor de los medicamentos así como la orden de compra que se le envía, cada medicamento se clasifica según la especialidad, como pediatría, urología, tipo de enfermedad, etc.