PRIMER EXAMEN DE LA MATERIA: BASES DE DATOS			
FECHA:	2022-05-31		
ALUMNO/A:	Gerardo Tordoya		
LEGAJO:	B00048241-T4	DNI:	22.777.420
CURSO:	Comisión 2-K	TURNO:	Distancia
CARRERA:	Analista Programador (Modalidad a Distancia)		
PROFESOR/A:	María Roxana Martínez		
MODALIDAD:	Individual – Domiciliario – Teórico – Práctico – Defensa coloquial		

UNIDADES A EVALUAR DEL PROGRAMA DE LA MATERIA:

- Unidad 1: ¿Qué Es Una Base De Datos?
- Unidad 2: Modelo Entidad Relación
- Unidad 3: Modelo Relacional, Mantener La Integridad Y La Consistencia
- Unidad 5: Mantener y documentar una base de datos

RESULTADOS DE APRENDIZAJE:

- [Diseña] + [el Modelo Conceptual de Datos] + [para representar las entidades y lasrelaciones] + [utilizando un Modelo Conceptual]
- [Elabora] + [un modelo Lógico de Datos]+ [para implementar una base de datos] +[utilizando el Modelo Relacional]

CRITERIOS DE CALIFICACIÓN:
Para aprobar el examen deberá sumar 60 puntos de un total de 100 (Teórico-Práctico),siendo, al menos el 60% de los aspectos conceptuales teóricos y al menos el 60% de los aspectos prácticos.

CRITERIOS DE RESOLUCIÓN:
Los alumnos/as recibirán la consigna del examen en la fecha de evaluación previstapor el cronograma de la asignatura.
El examen constará de 2 instancias:

- *Semana 1:* Entrega de las consignas y explicación de la metodología deevaluación por parte del docente a los alumnos/as.
- *Semana 2:* Entrega de la evaluación digital (archivo en MS Word y archivo generado con el software propuesto para el desarrollo de la parte práctica)por parte de los alumnos/as al docente.

PARTE TEÓRICA (100 PTS)

Debe obtener al menos 60/100 para la aprobación de esta parte.

UNIDAD 1: ¿QUÉ ES UNA BASE DE DATOS? (30 PTS)

1. CONCEPTO DE BASE DE DATOS. VENTAJAS DEL ENFOQUE DE BASE DE DATOS FRENTE A UN SISTEMA DE ARCHIVOS.

Expondré el concepto de base de datos al final para que se comprenda mejor:

Cuanto más se han desarrollado las sociedades, mejores métodos se han desarrollado para gestionar la información y, a la vez, más datos se han necesitado gestionar. Para poder almacenar datos y cada vez más datos, el ser humano ideó nuevas herramientas: archivos, cajones, carpetas y fichas en las que se almacenaban los datos. Antes de la aparición de las computadoras, el tiempo requerido para manipular estos datos era enorme. Sin embargo, el proceso de aprendizaje era relativamente sencillo, ya que se usaban elementos que las personas manejaban desde su infancia. Por esa razón, la informática se adaptó para que la terminología en el propio ordenador se pareciera a los términos de organización de datos clásicos. Así, en informática se sigue hablado de ficheros, formularios, carpetas, directorios, etc.

Formalmente, una base de datos o banco de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso.

En la evolución de los sistemas de información ha habido dos puntos determinantes, que han formado los dos tipos fundamentales de sistemas de información electrónico:

SISTEMAS DE GESTIÓN DE FICHEROS

Este tipo de sistemas hace referencia a la forma que inicialmente se desarrolló en la informática para gestionar ficheros (y que aún se usa). En realidad, es una forma que traducía la manera clásica de gestionar sistemas de información (con sus archivadores, carpetas, etc.) al mundo electrónico. La idea es que los datos se almacenan en ficheros y se crean aplicaciones (cuyo código posee la empresa que crea dichas aplicaciones) para acceder a los ficheros. Cada aplicación organiza los datos en los ficheros como le parece mejor y si incorporamos aplicaciones nuevas, estas usarán sus propios ficheros. Cada aplicación almacena y utiliza sus propios datos de forma un tanto caótica. La ventaja de este sistema (*la única ventaja*) es que los procesos son independientes por lo que la modificación de uno no afecta al resto. Pero tiene grandes inconvenientes:

- **Programación de aplicaciones compleja.** Ya que los programadores se deben de encargar de lo que tiene que hacer la aplicación además de estructurar los datos en disco.
- **Datos redundantes.** Ya que se repiten continuamente. Podría, por ejemplo, ocurrir que una segunda aplicación utilice datos de personales, que resulta que ya estaban almacenados en los ficheros de una primera aplicación, pero como ambas son independientes, los datos se repetirán.
- **Datos inconsistentes.** En relación con el problema anterior, ya que un proceso cambia sus datos y no los del resto. Por lo que la misma información puede tener distintos valores según qué aplicación acceda a él.
- **Difícil acceso a los datos.** Cada vez que se requiera una consulta no prevista inicialmente, hay que modificar el código de las aplicaciones o incluso crear una nueva aplicación. Esto hace imposible pensar en nuevas consultas e instantáneamente obtener sus resultados; inviable para aplicaciones que requieren grandes capacidades de consultas y análisis de datos.

- **Coste de almacenamiento elevado.** Al almacenarse varias veces el mismo dato, se requiere más espacio en los discos. Además, las aplicaciones también ocupan mucho al tener que pensar en todas las posibles consultas sobre los datos que la organización precisa.
- **Dependencia de los datos a nivel físico.** Para poder saber cómo se almacenan los datos, es decir qué estructura se utiliza de los mismos, necesitamos ver el código de la aplicación; es decir el código y los datos no son independientes.
- **Dificultad para el acceso simultáneo a los datos.** El acceso simultáneo requiere que varios usuarios al puedan acceder a la misma información. Con este tipo de sistemas es extremadamente difícil conseguir esta capacidad.
- **Dificultad para administrar la seguridad del sistema.** Ya que cada aplicación se crea independientemente. Es, por tanto, muy difícil establecer criterios de seguridad uniformes. Es decir, los permisos que cada usuario tiene sobre los datos se establecen de forma muy confusa (y nada uniforme ya que cada aplicación puede variar la seguridad).

Se consideran también sistemas de gestión de ficheros a los sistemas que utilizan programas ofimáticos (como Word o Excel, por ejemplo) para gestionar sus datos. Esta última idea la utilizan muchas pequeñas empresas para gestionar los datos debido al presupuesto limitado del que disponen. Gestionar la información de esta forma produce los mismos (si no más) problemas.

SISTEMAS DE BASES DE DATOS

En este tipo de sistemas, los datos se centralizan en una base de datos común a todas las aplicaciones. Un software llamado Sistema Gestor de Bases de Datos (SGBD) es el que realmente accede a los datos y se encarga de gestionarlos. Las aplicaciones que creen los programadores no acceden directamente a los datos, de modo que la base de datos es común para todas las aplicaciones. De esta forma, hay, al menos, dos capas a la hora de acceder a los datos. Las aplicaciones se abstraen sobre la forma de acceder a los datos, dejando ese problema al SGBD. Así se pueden concentrar exclusivamente en la tarea de conseguir una interfaz de acceso a los datos para los usuarios. Cuando una aplicación modifica un dato, la modificación será visible inmediatamente para el resto de las aplicaciones ya que todas utilizarán la misma base de datos.

Ventajas:

- **Independencia de los datos y los programas.** Esto permite modificar los datos sin modificar el código de las aplicaciones y viceversa.
- **Menor redundancia.** Este modelo no requiere que los datos se repitan para cada aplicación que los requiera, en su lugar se diseñan los datos de forma independiente a las aplicaciones. Los programadores de aplicaciones deberán conocer la estructura creada para los datos y la forma en la que deben acceder a ellos.
- **Integridad de los datos.** Al estar centralizados, es más difícil que haya datos incoherentes. Es decir, que una aplicación muestre información distinta al resto de aplicaciones, ya que los datos son los mismos para todas.
- **Mayor seguridad en los datos.** El SGBD es el encargado de la seguridad y se puede centrar en ella de forma independiente a las aplicaciones. Como las aplicaciones deben atravesar la capa del SGBD para llegar a los datos, no se podrán saltar la seguridad.
- **Visiones distintas según el usuario.** Nuevamente, centralizar los datos facilita crear políticas que permitan que los usuarios vean la información de la base de datos de forma distinta.
- **Datos más documentados.** Las bases de datos tienen mucho mejor gestionados los metadatos, que permiten describir la información de la base de datos y que pueden ser consultados por las aplicaciones.
- **Acceso a los datos más eficiente.** Esta forma de organizar los datos produce un resultado óptimo en rendimiento ya que los sistemas gestores centralizan el acceso pudiendo ejecutar políticas diferentes en función de la demanda.
- **Menor espacio de almacenamiento.** Puesto que hay muy poca redundancia.
- **Acceso simultáneo a los datos.** Nuevamente el SGBD tiene más capacidad de conseguir esto. Cuando hay varias aplicaciones que intentan acceder a los datos en los sistemas orientados a los ficheros, compiten por los datos y es fácil el bloqueo mutuo. En el caso de los sistemas

orientados a bases de datos, toda petición pasa la capa del SGBD y esto permite evitar los bloqueos.

Desventajas:

- **Instalación costosa.** El control y administración de bases de datos requiere de un software y hardware poderoso.
- **Requiere personal cualificado.** Debido a la dificultad de manejo de este tipo de sistemas.
- **Implantación larga y difícil.** En relación con los puntos anteriores. La adaptación del personal y del equipamiento es mucho más complicada y lleva bastante tiempo.
- **Ausencia de estándares totales.** Lo cual significa una excesiva dependencia hacia los sistemas comerciales del mercado. Aunque, hoy en día, hay un funcionamiento base y un lenguaje de gestión (SQL) que desde hace tiempo se considera estándar (al menos en las bases de datos relacionales).

2. DESARROLLE SISTEMA DE ADMINISTRACIÓN DE BASE DE DATOS (DBMS). EJEMPLIFICAR MEDIANTE GRÁFICO.

Un sistema gestor de base de datos (SGBD) o Database Management System (DBMS) es un conjunto de programas invisibles para el usuario final con el que se administra y gestiona la información que incluye una base de datos. Los gestores de datos o gestores de base de datos permiten administrar todo acceso a la base de datos, pues tienen el objetivo de servir de interfaz entre esta, el usuario y las aplicaciones. Entre sus funciones se encuentran la de permitir a los usuarios de negocio almacenar la información, modificar datos y acceder a los activos de conocimiento de la organización. Asimismo, el gestor de base de datos también se ocupa de realizar consultas y hacer análisis para generar informes. Además, los sistemas de gestión de base de datos pueden entenderse como una colección de datos interrelacionados, estructurados y organizados en el ecosistema formado por dicho conjunto de programas que acceden a ellos y facilitan su gestión.

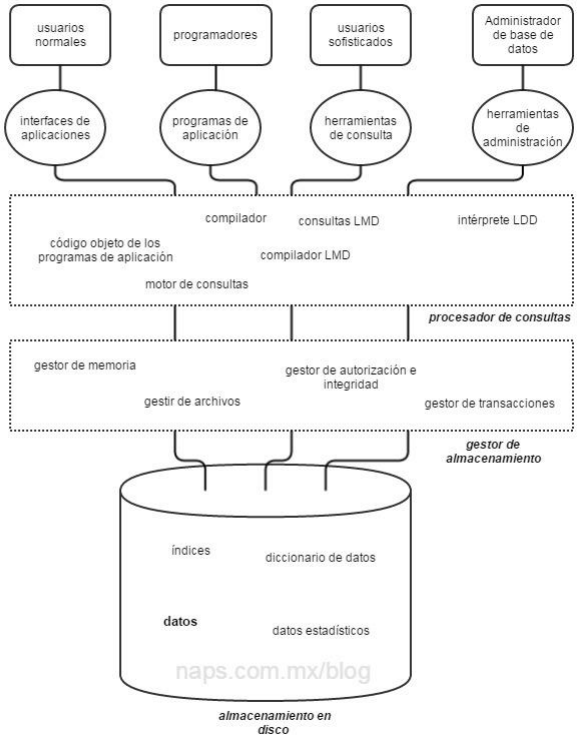
Formalmente, entre la BD física y los usuarios existe una capa de Software denominada SISTEMA ADMINISTRADOR DE BASE DE DATOS (SMBD o DBMS), y todos los requerimientos de acceso a la BD son manejados por el SMBD.

Qué permiten los SGBD:

En pocas palabras, el gestor de base de datos controla cualquier operación ejecutada por los distintos tipos de usuarios contra la BD. Para desarrollar esta función, es normal que se requieran herramientas específicas, como por ejemplo, sistemas de búsqueda y de generación de informes, así como distintas aplicaciones. Los gestores de base de datos también permiten lo siguiente:

- Que las interacciones con cualquier base de datos gestionada puedan desarrollarse siempre separadamente a los programas o aplicaciones que los gestionan.
- La manipulación de bases de datos, garantizando su seguridad, integridad y consistencia.
- La definición de bases de datos a diferentes niveles de abstracción.

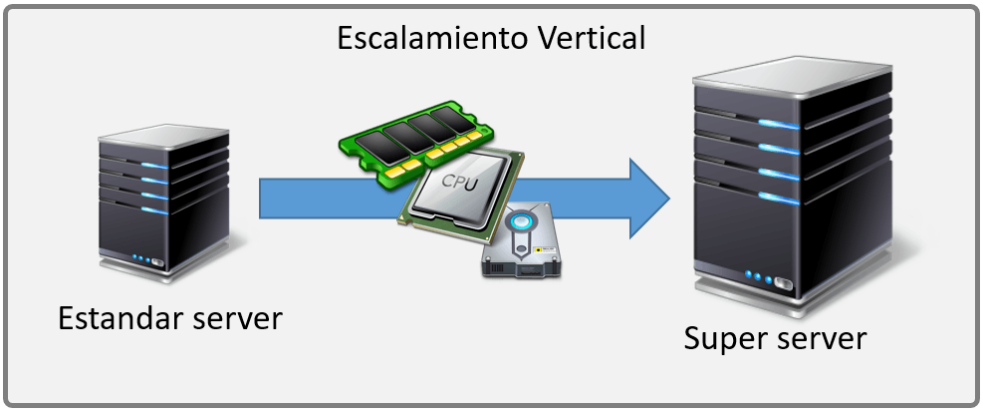
Una arquitectura típica de un gestor de base de datos es la siguiente:



3. DIFERENCIA ENTRE ESCALABILIDAD VERTICAL Y ESCALABILIDAD HORIZONTAL. DESARROLLE Y GRAFIQUE.

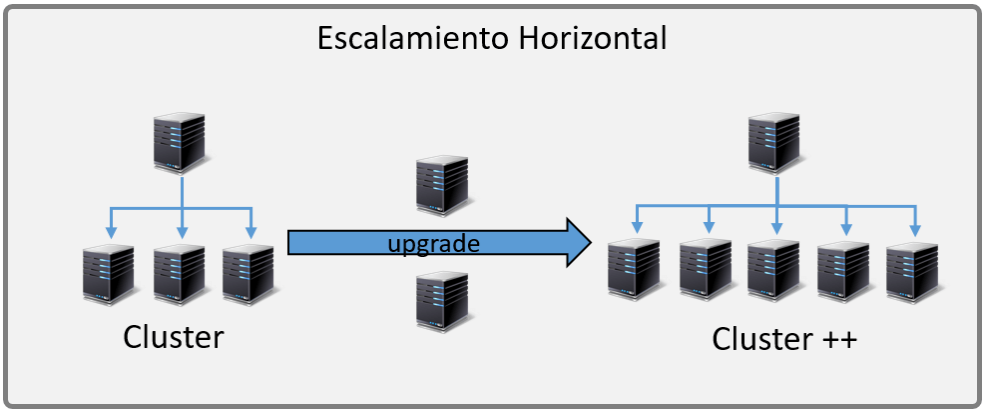
La escalabilidad de la base de datos (database scalability) es la capacidad que tienen las bases de datos de mejorar la disponibilidad y su comportamiento cuando el negocio demanda más recursos. Esta escalabilidad se puede abordar desde dos enfoques: Vertical y Horizontal.

Vertical Scaling. Este enfoque implica añadir más recursos físicos y virtuales al servidor subyacente que aloja la base de datos. Ampliar más capacidad de computación (CPU), más memoria o más capacidad de almacenamiento. Este es el enfoque tradicional, consistente en utilizar un servidor más grande para soportar todos los datos:



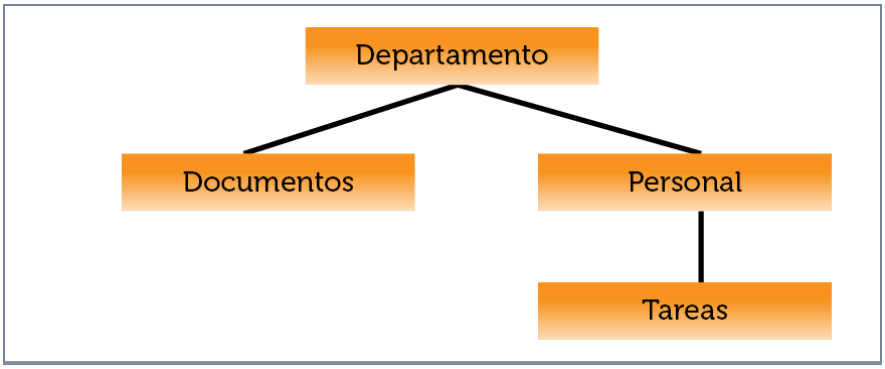
Horizontal Scaling. Este enfoque está relacionado con añadir más instancias o nodos a la base de datos para tratar de lidiar con una mayor carga de trabajo. Así, cuando una organización necesita una mayor capacidad, simplemente añade más servidores al clúster. Además, el clúster utilizado tiende a formar

parte de servidores más pequeños y baratos. El inconveniente de esta aproximación es que la mayoría de los productos no escalan de este modo y dependiendo de cómo están implementados, las aplicaciones necesitarán ser reprogramadas para trabajar con la base de datos. Todo un reto para los desarrolladores de sistemas:



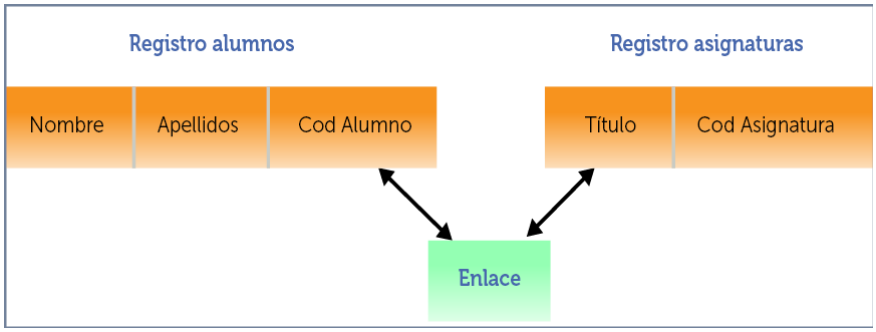
4. **DESCRIBA EL ENFOQUE JERÁRQUICO VERSUS ENFOQUE DE RED.**

JERÁRQUICO:



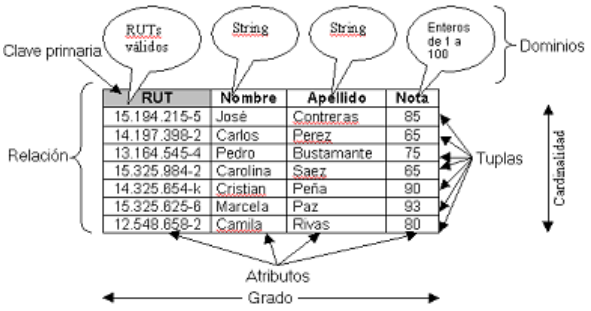
Era utilizado por los primeros SGBD, desde que IBM lo definió para su IMS (Information Management System, Sistema Administrador de Información) en 1970. Se le llama también modelo en árbol debido a que utiliza una estructura en árbol para organizar los datos. La información se organiza con una jerarquía en la que la relación entre las entidades de este modelo siempre es del tipo padre/hijo. De esta forma hay una serie de nodos que contendrán atributos y que se relacionarán con nodos hijos de forma que puede haber más de un hijo para el mismo padre (pero un hijo sólo tiene un padre). Los datos de este modelo se almacenan en estructuras lógicas llamadas segmentos. Los segmentos se relacionan entre sí utilizando arcos. La forma visual de este modelo es de árbol invertido, en la parte superior están los padres y en la inferior los hijos. Este esquema está en absoluto desuso ya que no es válido para modelar la mayoría de los problemas de bases de datos. Su virtud era la facilidad de manejo ya que sólo existe un tipo de relación (padre/hijo) entre los datos; su principal desventaja es que no basta para representar la mayoría de las relaciones. Además, no mantenía independencia física de la base de datos.

RED (CODASYL):



Es un modelo que ha tenido una gran aceptación (aunque apenas se utiliza actualmente). En especial se hizo popular la forma definida por el estándar Codasyl a principios de los 70 que se convirtió en el modelo en red más utilizado. El modelo en red organiza la información en registros (también llamados nodos) y enlaces. En los registros se almacenan los datos, mientras que los enlaces permiten relacionar estos datos. Las bases de datos en red son parecidas a las jerárquicas sólo que en ellas puede haber más de un padre. En este modelo se pueden representar perfectamente cualquier tipo de relación entre los datos (aunque el Codasyl restringía un poco las relaciones posibles), pero hace muy complicado su manejo. Poseía un lenguaje poderoso de trabajo con la base de datos. El problema era la complejidad para trabajar con este modelo tanto para manipular los datos como programar aplicaciones de acceso a la base de datos. Tampoco mantenía una buena independencia con la física de la base de datos.

5. **DESCRIBA, EJEMPLIFIQUE Y GRAFIQUE:**
   a. ¿QUÉ RELACIÓN HAY ENTRE CARDINALIDAD Y RELACIÓN?



- **Cardinalidad:** Es el número de tuplas que contiene una tabla.
- **Tabla o Relación:** Es un conjunto de tuplas que han de ser por toda fuerza distintas. Esto también implica que el orden de las tuplas es irrelevante. El conjunto vacío es una relación particular: la relación *nula* o *vacía*.

   b. ¿QUÉ ES EL DOMINIO DE UN ATRIBUTO?

Un dominio contiene todos los posibles valores que puede tomar un determinado atributo. Dos atributos distintos pueden tener el mismo dominio. Un dominio en realidad es un conjunto finito de valores del mismo tipo. A los dominios se les asigna un nombre y así podemos referirnos a ese nombre en más de un atributo (*véase imagen anterior*).

6. CONCEPTO DE LA INDEPENDENCIA FÍSICA Y LÓGICA DE DATOS.

Edgar Frank Codd a finales definió las bases del modelo relacional a finales de los 60. En 1970 publica el documento "A Relational Model of data for Large Shared Data Banks" ("Un modelo relacional de datos para grandes bancos de datos compartidos"). Actualmente se considera que ese es uno de los documentos más influyentes de toda la historia de la informática. Codd perseguía estos objetivos con su modelo:

- **Independencia física.** La forma de almacenar los datos no debe influir en su manipulación lógica. Si el almacenamiento físico cambia, los usuarios no tienen ni siquiera porque enterarse, seguirán funcionando sus aplicaciones.
- **Independencia lógica.** Las aplicaciones que utilizan la base de datos no deben ser modificadas por que se modifiquen elementos de la base de datos. Es decir, añadir, borrar y suprimir datos, no influye en las vistas de los usuarios.

Formalmente, la independencia de datos es el objetivo esencial de un DBMS y consiste en mantener la inmunidad de las aplicaciones ante los cambios en la estrategia de acceso o en la estructura de almacenamiento de los datos.

UNIDAD 2: MODELO ENTIDAD RELACIÓN (30 PTS)

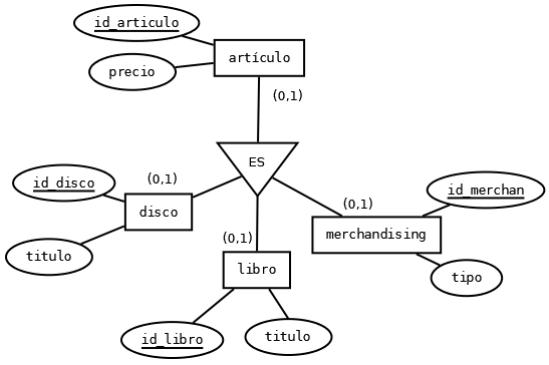
1. EXPLIQUE Y EJEMPLIFIQUE COBERTURA EN GENERALIZACIÓN. CASO COBERTURA TOTAL Y EXCLUSIVA.

En la bibliografía en la que he encontrado este tema, recibe el título de "Relaciones ISA". Y dícese de las relaciones que indican relaciones que permiten distinguir tipos de entidades, es decir tendremos entidades que son un (*is a*, en inglés) tipo de entidad respecto a otra entidad más general. Se utilizan para unificar entidades agrupándolas en una entidad más general (generalización) o bien para dividir una entidad en entidades más específicas (especificación): aunque hoy en día a todas ellas se las suele llamar generalización e incluso (quizá más adecuadamente) relaciones de herencia. Se habla de superentidad refiriéndonos a la entidad general sobre las que derivan las otras (que se llaman subentidades). En la superentidad se indican los atributos comunes a todas las subentidades, se sobreentiende que las subentidades también tienen esos atributos, pero no se indican de nuevo esos atributos en el diagrama.

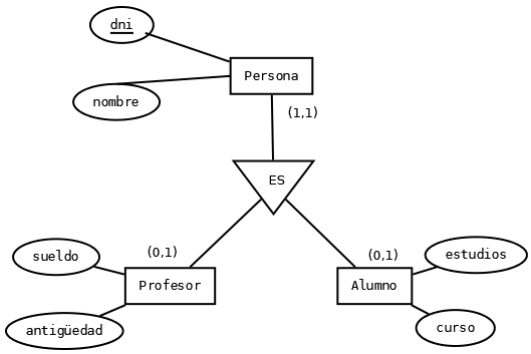
NOTA

En otro artículo, se especifica una distinción entre generalización y especialización:

- **Generalización.** En toda jerarquía los campos se heredan del supertipo, pero es posible que cada entidad tenga su propia clave principal, de forma que se puede identificar de forma independiente. En estos casos la cardinalidad mínima será siempre en todas las entidades (0,1). Podemos identificar una generalización cuando los supertipos y subtipos tienen sus propias claves:

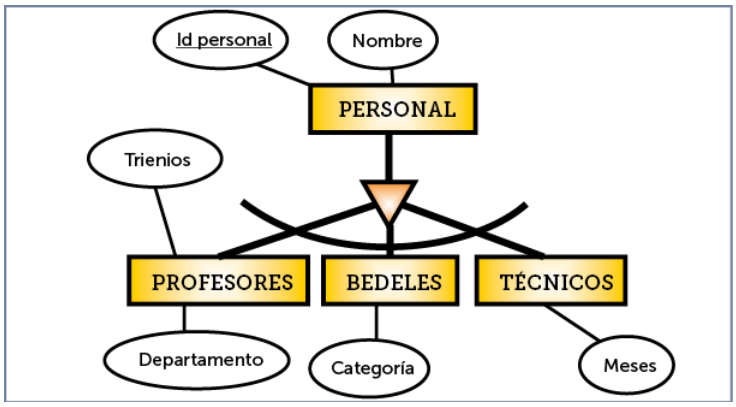


- **Especialización.** En este tipo de jerarquía también se heredan los campos del supertipo, pero la diferencia que hay con la generalización, es que en la especialización todas las entidades intervinientes comparten la clave del supertipo. En estos casos la la cardinalidad del supertipo será (1,1):



EXCLUSIVIDAD

En las relaciones ISA (y también en otros tipos de relaciones) se puede indicar el hecho de que cada ejemplar sólo puede participar en una de entre varias ramas de una relación. Este hecho se marca con un arco entre las distintas relaciones. En las relaciones ISA se usa mucho para indicar que cada ejemplar de la superentidad sólo se relaciona con una subentidad, por ejemplo:



En el ejemplo, el personal sólo puede ser o bedel o profesor o técnico; una y sólo una de las tres cosas (es por cierto la forma más habitual de relación ISA).

EJEMPLO PEDIDO EN LA CONSIGNA:

Consideremos el caso de un banco cualquiera y su política hacia las personas con respecto a su calidad como empleados y/o clientes. Si esa política establece que todas las personas son empleados o clientes del banco y no ambas cosas simultáneamente, estamos ante un caso de cobertura total y exclusiva. Es decir, hablamos de cobertura **total** (todas las personas están clasificadas como empleados o clientes, obligatoriamente, no pueden pertenecer a la superentidad) y **exclusiva** (si una persona se clasifica como empleado, no puede clasificarse como cliente, y viceversa).

2. **REALIZAR UN CUADRO CON LAS CARACTERÍSTICAS DE LOS SIGUIENTES MODELOS (EJEMPLIFIQUE EN CADA CASO):**
   a. Modelo Entidad Relación
   b. Modelo Conceptual
   c. Modelo Relacional
   d. Modelo Lógico

Formalmente, cada modelo se caracteriza por:

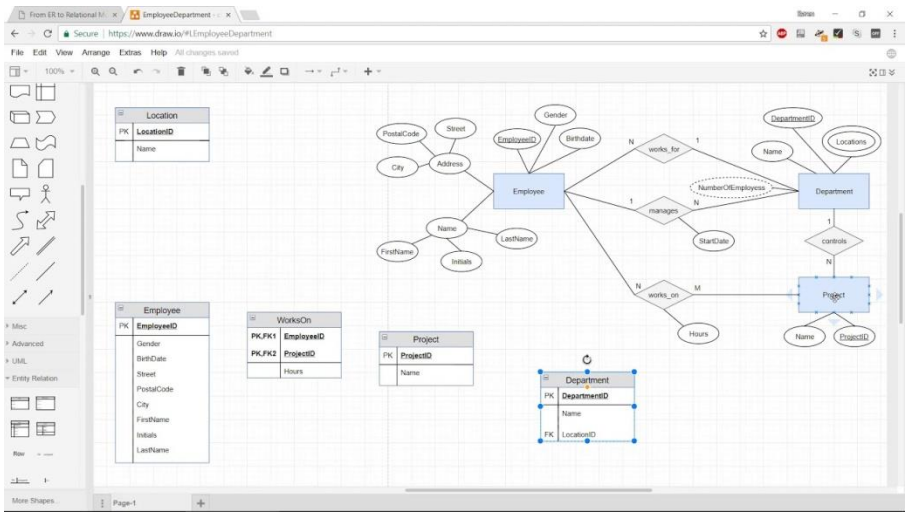
MODELO ENTIDAD INTERRELACIÓN	MODELO CONCEPTUAL	MODELO RELACIONAL	MODELO LÓGICO
Es el enfoque más natural del mundo real (que lo conceptúa en entidades y sus interrelaciones).	Es el orientado a la descripción de estructuras de datos y restricciones de integridad de un problema dado y está enfocado en representar los elementos y las relaciones que intervienen.	Es el modelo de datos para la gestión de una BD a fin de administrar datos dinámicamente de problemas reales.	Es el orientado a las operaciones más que a la descripción de una realidad.

El primer par comparable es Modelo Entidad-Relación y Modelo Relacional

Cuadro:

Comparativa	**Modelo ER**	**Modelo Relacional**
Uso	Se utiliza para describir un conjunto de objetos conocidos como entidades, así como las relaciones entre ellos.	Se utiliza para representar una colección de tablas, así como las relaciones entre ellas.
Tipo	Es un modelo de alto nivel o conceptual.	Es el modelo de implementación o representacional.
Componentes	Representa componentes como Entidad, Tipo de entidad y Conjunto de entidades.	Representa componentes como dominio, atributos y tuplas.
Usuarios	Este modelo es útil para aquellas personas que no tienen ningún conocimiento sobre cómo se implementan los datos.	Este modelo es sobre todo famoso entre los programadores.
Relación	Es fácil entender la relación entre entidades.	En comparación con el modelo ER, es más fácil derivar la relación entre tablas en el modelo relacional.

Ejemplo:

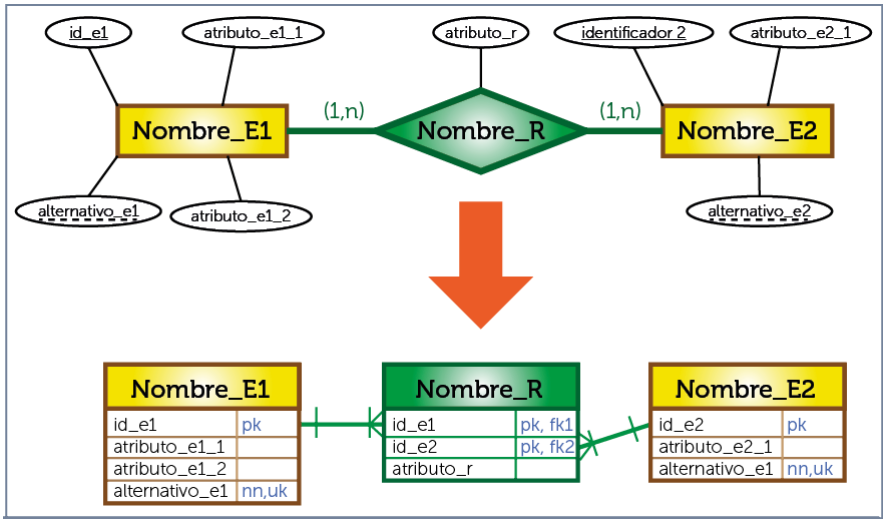


En la imagen se puede ver un modelo conceptual (a la derecha, parte superior) siendo convertido al modelo relacional (a la izquierda, parte inferior). Se puede apreciar cómo se van plasmando las entidades del modelo conceptual en las futuras “tablas” del modelo relacional.

El segundo par comparable es el Modelo Conceptual y el Modelo Lógico

Modelo Conceptual	Modelo Lógico
Un modelo que ayuda a identificar las relaciones de más alto nivel entre las diferentes entidades.	Un modelo que describe los datos con el mayor detalle posible, sin importar cómo se implementarán físicamente en la base de datos.
Consta de entidades y las relaciones entre las entidades.	Consta de entidades, atributos, relaciones, claves primarias y claves foráneas.
No especifica claves primarias y claves foráneas.	Especifica las claves principales y las claves externas.
Base para desarrollar el modelo lógico de datos.	Base para desarrollar el modelo físico de datos.
Más simple que el modelo de datos lógicos.	Más complejo que el modelo de datos conceptual.

Ejemplo:



El ejemplo, muy parecido al anterior, es un ejemplo terminado de una conversión conceptual a lógico.

3. EXPLIQUE Y EJEMPLIFIQUE: INTERRELACIÓN UNARIA, BINARIA Y N-ARIA.

Relaciones Binarias. Son las relaciones típicas. Se trata de relaciones que asocian dos entidades.

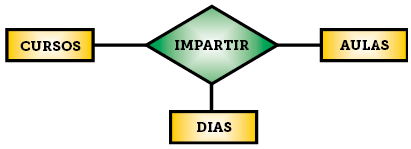
Relaciones Ternarias. Relacionan tres entidades. A veces se pueden simplificar en relaciones binarias, pero no siempre es posible.

Relaciones n-arias. Relacionan n entidades (por ejemplo, relaciones cuaternarias, quinquenarias, ...). Son muy raras.

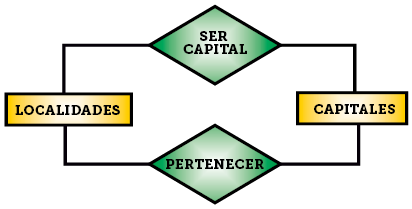
Relaciones dobles. Se llaman así a dos relaciones distintas que sirven para relacionar a las mismas relaciones. Son las más difíciles de manejar ya que al manipular las entidades hay que elegir muy bien cuál es la relación adecuada para hacerlo.

Relación reflexiva (unarias). Es una relación que sirve para relacionar dos ejemplares de la misma entidad (personas con personas, piezas con piezas, etc.)

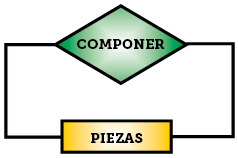
Relación binaria



Relación ternaria



Relación doble



Relación reflexiva

4. ¿Cuál es la diferencia ente el Diseño Conceptual y el Diseño Lógico?

El **modelo conceptual** no se ocupa de implementar la base de datos real. Funciona a un nivel muy alto, traduciendo los problemas del mundo real y los requisitos comerciales en un marco conceptual que es fácil de entender y que, por esta razón, puede comunicarse también a los no expertos. Solo se enfoca en identificar las entidades que serán parte de la base de datos y las relaciones entre ellas, es decir, las operaciones o asociaciones que existen entre ellas. Es muy eficaz para comprender y definir los procesos comerciales de la empresa. Entonces, si intentáramos una definición de modelo de datos conceptual, podríamos decir que un modelo de datos conceptual es una representación de la estructura general de datos requerida para llevar a cabo los requisitos comerciales (lo que significa respaldar los procesos de la empresa, registrar eventos comerciales y realizar un seguimiento del desempeño) y esto independientemente de cualquier software específico, sistema de gestión de base de datos o estructura de almacenamiento de datos.

NOTA: Es esencial disponer de un modelo de datos conceptual para obtener una comprensión precisa de los tipos de datos disponibles y necesarios, para llegar a un acuerdo entre las diferentes unidades de negocio y para definir nombres, tipos de datos y características entre ellos. Es importante comprender que cuando se trata de un modelo de datos conceptual, se trata de objetos a un nivel muy alto. Por ejemplo, si te refieres a un automóvil, te refieres al objeto del mundo real «automóvil» y no a una representación abstracta; un conjunto de partes metálicas conectadas de cierta manera. Esta es la razón por la que no todo lo que aparece en un modelo de datos conceptual puede traducirse al modelo de datos físicos.

El **modelo de datos lógicos** es un paso más allá y es el primer paso para construir realmente la arquitectura de las aplicaciones. Sigue siendo independiente del sistema particular utilizado (por ejemplo, un sistema de gestión de base de datos en particular) pero a partir del modelo conceptual intenta traducir las vistas de alto nivel que se pueden encontrar en este último en un esquema formal y abstracto, agregando un nivel adicional de detalle al nivel conceptual. A diferencia del modelo conceptual, el modelo de datos lógicos utiliza índices y claves externas para representar las relaciones de datos, pero aun así se mantiene independiente de cualquier implementación de DBMS. Podemos considerarlo como un puente entre el modelo de datos conceptual (vista empresarial) y el modelo de datos físicos (vista del desarrollador) y podemos usarlo para verificar si la implementación real cumple con los requisitos establecidos en el modelo conceptual.

Entonces, **la diferencia** entre el modelo de datos conceptual y lógico es clara. El modelo de datos lógicos es una representación abstracta de una posible implementación, sin estar vinculado a ninguna implementación específica, mientras que el modelo de datos conceptual es una representación de alto nivel de los requisitos comerciales y los conjuntos y relaciones de datos conectados. El modelo lógico, a través de un proceso de normalización, brinda una representación más estructurada de las entidades y sus relaciones y detalla más sus relaciones y características: se resuelven las redundancias, las relaciones de

muchos a muchos, las ambigüedades y las incertidumbres de atribución de entidades y surge obviamente una posible implementación de base de datos.

UNIDAD 3: MODELO RELACIONAL, MANTENER LA INTEGRIDAD Y LA CONSISTENCIA (20 PTS)

1. EXPLIQUE Y EJEMPLIFIQUE CLAVE PRIMARIA, SUPERCLAVES, CLAVES CANDIDATAS Y CLAVES FORÁNEAS.

Vamos con las tres primeras:

Toda la información que contiene una BD debe poderse identificar de alguna forma. En el caso particular de las BBDD que siguen el modelo relacional, para identificar los datos que la BD contiene, se pueden utilizar las claves candidatas de las relaciones. A continuación, definimos qué se entiende por clave candidata, clave primaria y clave alternativa de una relación. Para hacerlo, será necesario definir el concepto de superclave: Una superclave de una relación de esquema R(A1, A2, ..., An) es un subconjunto de los atributos del esquema tal que no puede haber dos tuplas en la extensión de la relación que tengan la misma combinación de valores para los atributos del subconjunto. Una superclave, por lo tanto, nos permite identificar todas las tuplas que contiene la relación.

> *En la relación de esquema EMPLEADOS(DNI, NSS[1], nombre, apellido, teléfono), algunas de las superclaves de la relación serían los siguientes subconjuntos de atributos: {DNI, NSS, nombre, apellido, teléfono}, {DNI, apellido}, {DNI} y {NSS}.*

Una clave candidata de una relación es una superclave C de la relación que cumple que ningún subconjunto propio de C es superclave. Es decir, C cumple que la eliminación de cualquiera de sus atributos da un conjunto de atributos que no es superclave de la relación. Intuitivamente, una clave candidata permite identificar cualquier tupla de una relación, de manera que no sobre ningún atributo para hacer la identificación.

> *En la relación de esquema EMPLEADOS(DNI, NSS, nombre, apellido, teléfono), sólo hay dos claves candidatas: {DNI} y {NSS}.*

Habitualmente, una de las claves candidatas de una relación se designa clave primaria de la relación. La clave primaria es la clave candidata cuyos valores se utilizarán para identificar las tuplas de la relación. El diseñador de la BD es quien elige la clave primaria de entre las claves candidatas. Las claves candidatas no elegidas primaria se denominan claves alternativas. Utilizaremos la convención de subrayar los atributos que forman parte de la clave primaria en el esquema de la relación. Así pues, R(A1, A2, ..., Ai, ..., An) indica que los atributos A1, A2, ..., Ai forman la clave primaria de R.

> *En la relación de esquema EMPLEADOS(DNI, NSS, nombre, apellido, teléfono), donde hay dos claves candidatas, {DNI} y {NSS}, se puede elegir como clave primaria {DNI}. Lo indicaremos subrayando el atributo DNI en el esquema de la relación EMPLEADOS(DNI, NSS, nombre, apellido, teléfono). En este caso, la clave {NSS} será una clave alternativa de EMPLEADOS.*

[1] Número de Seguridad Social. Común en muchos países, no en Argentina.

Es posible que una clave candidata o una clave primaria conste de más de un atributo.

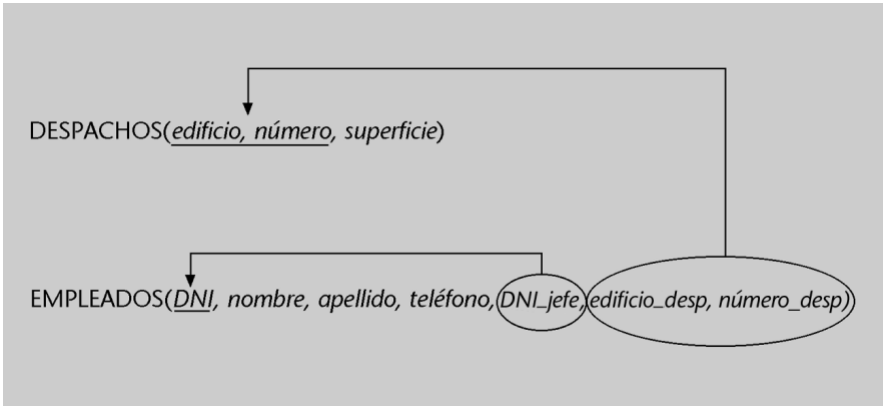
Formalmente:

- **Superclave** es un conjunto de uno o más atributos que, tomados colectivamente, permiten identificar de forma única una entidad en el conjunto de entidades.
- **Clave Primaria** es un campo (o conjunto de campos) que identifica inequívocamente un registro. Es decir, es un campo que no admite valores duplicados en los registros.
- **Clave Candidata** es aquella que podría utilizarse como Clave Primaria.

Ahora vamos con la última:

Hasta ahora hemos estudiado las relaciones de forma individual, pero debemos tener en cuenta que una BD relacional normalmente contiene más de una relación para poder representar distintos tipos de hechos que suceden en el mundo real. Por ejemplo, podríamos tener una pequeña BD que contuviese dos relaciones: una denominada EMPLEADOS, que almacenaría datos de los empleados de una empresa, y otra con el nombre DESPACHOS, que almacenaría los datos de los despachos que tiene la empresa. Debemos considerar también que entre los distintos hechos que se dan en el mundo real pueden existir lazos o vínculos. Por ejemplo, los empleados que trabajan para una empresa pueden estar vinculados con los despachos de la empresa, porque a cada empleado se le asigna un despacho concreto para trabajar. En el modelo relacional, para reflejar este tipo de vínculos, tenemos la posibilidad de expresar conexiones entre las distintas tuplas de las relaciones. Por ejemplo, en la BD anterior, que tiene las relaciones EMPLEADOS y DESPACHOS, puede ser necesario conectar tuplas de EMPLEADOS con tuplas de DESPACHOS para indicar qué despacho tiene asignado cada empleado. En ocasiones, incluso puede ser necesario reflejar lazos entre tuplas que pertenecen a una misma relación. Por ejemplo, en la misma BD anterior, puede ser necesario conectar determinadas tuplas de EMPLEADOS con otras tuplas de EMPLEADOS para indicar, para cada empleado, quién actúa como su jefe. El mecanismo que proporcionan las BD relacionales para conectar tuplas son las **claves foráneas** de las relaciones. Las claves foráneas permiten establecer conexiones entre las tuplas de las relaciones. Para hacer la conexión, una clave foránea tiene el conjunto de atributos de una relación que referencian la clave primaria de otra relación (o incluso de la misma relación).

En la figura siguiente, la relación EMPLEADOS(DNI, nombre, apellido, teléfono, DNI_jefe, edificio_desp, número_desp), tiene una clave foránea formada por los atributos edificio_desp y número_desp que se refiere a la clave primaria de la relación DESPACHOS(edificio, número, superficie). Esta clave foránea indica, para cada empleado, el despacho donde trabaja. Además, el atributo DNI_jefe es otra clave foránea que referencia la clave primaria de la misma relación EMPLEADOS, e indica, para cada empleado, quien es su jefe:



Las claves foráneas tienen por objetivo establecer una conexión con la clave primaria que referencian. Por lo tanto, los valores de una clave foránea deben estar presentes en la clave primaria correspondiente, o bien deben ser valores nulos. En caso contrario, la clave foránea representaría una referencia o conexión incorrecta.

Formalmente, **Clave Foránea** es un campo (o conjunto de campos) cuyos valores están limitados a los que se hayan definido en otra tabla como Clave Principal.

2. ¿Qué son las Operaciones fundamentales del álgebra relacional? Desarrollo ejemplo de Producto cartesiano versus Proyección.

El álgebra relacional se inspira en la teoría de conjuntos para especificar consultas en una BD relacional. Para especificar una consulta en álgebra relacional, es preciso definir uno o más pasos que sirven para ir construyendo, mediante operaciones de álgebra relacional, una nueva relación que contenga los datos que responden a la consulta a partir de las relaciones almacenadas. Los lenguajes basados en el álgebra relacional son procedimentales, dado que los pasos que forman la consulta describen un procedimiento. Las operaciones del álgebra relacional han sido clasificadas según distintos criterios; de todos ellos indicamos los tres siguientes:

Según se pueden expresar o no en términos de otras operaciones:

- **Operaciones primitivas:** son aquellas operaciones a partir de las cuales podemos definir el resto. Estas operaciones son la unión, la diferencia, el producto cartesiano, la selección y la proyección.
- **Operaciones no primitivas:** el resto de las operaciones del álgebra relacional que no son estrictamente necesarias, porque se pueden expresar en términos de las primitivas; sin embargo, las operaciones no primitivas permiten formular algunas consultas de forma más cómoda. Existen distintas versiones del álgebra relacional, según las operaciones no primitivas que se incluyen. Las operaciones no primitivas que se utilizan con mayor frecuencia son la intersección y la combinación.

Según el número de relaciones que tienen como operandos:

- **Operaciones binarias:** son las que tienen dos relaciones como operandos. Son binarias todas las operaciones, excepto la selección y la proyección.
- **Operaciones unarias:** son las que tienen una sola relación como operando. La selección y la proyección son unarias.

Según se parecen o no a las operaciones de la teoría de conjuntos:

- **Operaciones conjuntistas:** son las que se parecen a las de la teoría de conjuntos. Se trata de la unión, la intersección, la diferencia y el producto cartesiano.
- **Operaciones específicamente relacionales:** son el resto de las operaciones; es decir, la selección, la proyección y la combinación.

CLASIFICACIÓN ACORDE A LOS APUNTES DE CÁTEDRA

Tradicionales:

- Unión: UNION
- Intersección: INTER
- Diferencia: MENOS
- Producto cartesiano: VECES

Especiales:

- Selección: DONDE
- Proyección
- Reunión: REUNION
- División: DIVISION

DESARROLLO EJEMPLO PRODUCTO CARTESIANO

El producto cartesiano es una operación que, a partir de dos relaciones, obtiene una nueva relación formada por todas las tuplas que resultan de concatenar tuplas de la primera relación con tuplas de la segunda. El producto cartesiano es una operación binaria. Siendo T y S dos relaciones que cumplen que sus esquemas no tienen ningún nombre de atributo común, el producto cartesiano de T y S se indica como T x S.

Esquema y extensión de *EDIFICIOS_EMP*:

EDIFICIOS_EMP	
edificio	*sup_media_desp*
Marina	15
Diagonal	10

Esquema y extensión de *DESPACHOS*:

DESPACHOS		
Edificio	*número*	*Superficie*
Marina	120	10
Marina	230	20
Diagonal	120	10
Diagonal	440	10

El producto cartesiano de las relaciones *DESPACHOS* y *EDIFICIOS_EMP* del ejemplo se puedehacer como se indica (es necesario renombrar atributos previamente):

> *EDIFICIOS(nombre_edificio, sup_media_desp) := EDICIOS_EMP(edificio, sup_media_desp).*
>
> *R := EDIFICIOS x DESPACHOS.*

Entonces, la relación *R* resultante será:

R				
nombre_edificio	*sup_media_desp*	*edificio*	*número*	*superficie*
Marina	15	Marina	120	10
Marina	15	Marina	230	20
Marina	15	Diagonal	120	10
Marina	15	Diagonal	440	10
Diagonal	10	Marina	120	10
Diagonal	10	Marina	230	20
Diagonal	10	Diagonal	120	10
Diagonal	10	Diagonal	440	10

Conviene señalar que el producto cartesiano es una operación que raramente se utiliza de forma explícita porque el resultado que da no suele ser útil para resolver las consultas habituales. A pesar de ello, el producto cartesiano se incluye en el álgebra relacional porque es una operación primitiva; a partir de la cual se define otra operación del álgebra, la combinación, que se utiliza con mucha frecuencia.

DESARROLLO EJEMPLO PROYECCIÓN

Podemos considerar la proyección como una operación que sirve para elegir algunos atributos de una relación y eliminar el resto. Más concretamente, la proyección es una operación que, a partir de una relación, obtiene una nueva relación formada por todas las (sub)tuplas de la relación de partida que resultan de eliminar unos atributos especificados. La proyección es una operación unaria. Siendo {A_i, A_j,

..., A_k} un subconjunto de los atributos del esquema de la relación *T*, la proyección de *T* sobre {A_i, A_j, ..., A_k} se indica como T[A_i, A_j, ..., A_k].

Esquema y extensión de *EMPLEADOS_ADM*:

EMPLEADOS_ADM				
DNI	*Nombre*	*apellido*	*edificio_desp*	*número_desp*
40.444.255	Juan	García	Marina	120
33.567.711	Marta	Roca	Marina	120

Si queremos obtener una relación *R* con el nombre y el apellido de todos los empleados de administración de la BD del ejemplo, haremos la siguiente proyección:

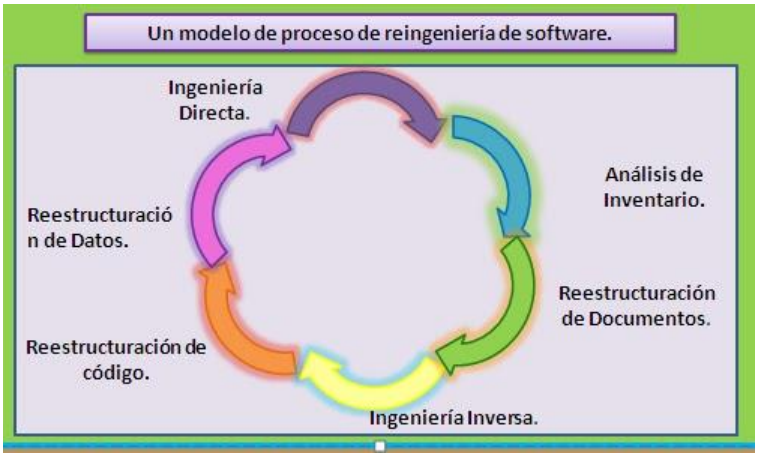
> *R := EMPLEADOS_ADM[nombre, apellido].*

Entonces, la relación *R* resultante será:

R	
nombre	*apellido*
Juan	García
Marta	Roca

UNIDAD 5: MANTENER Y DOCUMENTAR UNA BASE DE DATOS (20 PTS)

1. **¿CUÁLES SON LAS DIFERENTES ACTIVIDADES QUE INVOLUCRA LA REINGENIERÍA DEL SOFTWARE? EXPLIQUE Y GRAFIQUE PASOS.**



Reestructuración de código. El tipo más común de reingeniería (en realidad, la aplicación del término reingeniería sería discutible en este caso) es la reestructuración del código. Algunos sistemas heredados tienen una arquitectura de programa relativamente sólida, pero los módulos individuales han sido codificados de una forma que hace difícil comprenderlos, comprobarlos y mantenerlos. En estos casos, se puede reestructurar el código ubicado dentro de los módulos sospechosos. Para llevar a cabo esta actividad, se analiza el código fuente mediante una herramienta de reestructuración, se indican las violaciones de las estructuras de programación estructurada, y entonces se reestructura el código (esto se puede hacer automáticamente). El código reestructurado resultante se revisa y se comprueba para asegurar que no se hayan introducido anomalías. Se actualiza la documentación interna del código.

Reestructuración de Datos. Un programa que posea una estructura de datos débil será difícil de adaptar y de mejorar. De hecho, para muchas aplicaciones, la arquitectura de datos tiene más que ver con la viabilidad a largo plazo del programa que el propio código fuente. A diferencia de la reestructuración de código, que se produce en un nivel relativamente bajo de abstracción, la estructuración de datos es una actividad de reingeniería a gran escala. En la mayoría de los casos, la reestructuración de datos comienza por una actividad de ingeniería inversa. La arquitectura de datos actual se analiza minuciosamente y se definen los modelos de datos necesarios. Se identifican los objetos de datos y atributos y, a continuación, se revisan las estructuras de datos a efectos de calidad. Cuando la estructura de datos es débil (por ejemplo, se implementan archivos planos cuando un enfoque relacional simplificaría muchísimo el procesamiento) se aplica una reingeniería a los datos. Dado que la arquitectura de datos tiene una gran influencia sobre la arquitectura del programa, y también sobre los algoritmos que lo pueblan, los cambios en datos darán lugar invariablemente a cambios o bien de arquitectura o bien de código.

Ingeniería Directa. En un mundo ideal, las aplicaciones se reconstruyen utilizando un «motor de reingeniería» automatizado. En el motor se insertaría el programa viejo, que lo analizaría, reestructuraría y después regeneraría la forma de exhibir los mejores aspectos de la calidad del software. Después de un espacio de tiempo corto, es probable que llegue a aparecer este «motor», pero los fabricantes de CASE han presentado herramientas que proporcionan un subconjunto limitado de estas capacidades y que se enfrentan con dominios de aplicaciones específicos (por ejemplo, aplicaciones que han sido implementadas empleando un sistema de bases de datos específico). Lo que es más importante, estas herramientas de reingeniería cada vez son más sofisticadas. La ingeniería directa, que se denomina también renovación o reclamación, no solamente recupera la información de diseño de un software ya existente, sino que, además, utiliza esta información para alterar o reconstruir el sistema existente en un esfuerzo por mejorar su calidad global. En la mayoría de los casos, el software procedente de una reingeniería vuelve a implementar la funcionalidad del sistema existente, y añade además nuevas funciones y/o mejora el rendimiento global.

Análisis de Inventario. Todas las organizaciones de software deberán disponer de un inventario de todas sus aplicaciones. El inventario puede que no sea más que una hoja de cálculo con la información que proporciona una descripción detallada (por ejemplo: tamaño, edad, importancia para el negocio) de todas las aplicaciones activas. Los candidatos a la reingeniería aparecen cuando se ordena esta información en función de su importancia para el negocio, longevidad, mantenibilidad actual y otros criterios localmente importantes. Es entonces cuando es posible asignar recursos a las aplicaciones candidatas para el trabajo de reingeniería. Es importante destacar que el inventario deberá revisarse con regularidad. El estado de las aplicaciones (por ejemplo, la importancia con respecto al negocio) puede cambiar en función del tiempo y, como resultado, cambiarán también las prioridades para la reingeniería.

Reestructuración de Documentos. Una documentación escasa es la marca de muchos sistemas heredados. ¿Qué se puede hacer al respecto?:

- Opción 1: La creación de documentación consume muchísimo tiempo. El sistema funciona, y ya nos apañaremos con lo que tengamos. En algunos casos, éste es el enfoque correcto. No es posible volver a crear la documentación para cientos de programas de computadoras. Si un programa es relativamente estático está llegando al final de vida útil, y no es probable que experimente muchos cambios.
- Opción 2: Es preciso actualizar la documentación, pero se dispone de recursos limitados. Se utilizará un enfoque «del tipo documentar si se modifica». Quizá no sea necesario volver a documentar por completo la aplicación. Más bien se documentarán por completo aquellas partes del sistema que estén experimentando cambios en ese momento. La colección de documentos Útil y relevante irá evolucionando con el tiempo.
- Opción 3: El sistema es fundamental para el negocio, y es preciso volver a documentarlo por completo. En este caso, un enfoque inteligente consiste en reducir la documentación al mínimo necesario

Todas y cada una de estas opciones son viables. Las organizaciones del software deberán seleccionar aquella que resulte más adecuada para cada caso.

Ingeniería Inversa. El término «ingeniería inversa» tiene sus orígenes en el mundo del hardware. Una cierta compañía desensambla un producto de hardware competitivo en un esfuerzo por comprender los «secretos» del diseño y fabricación de su competidor. Estos secretos se podrán comprender más fácilmente si se obtuvieran las especificaciones de diseño y fabricación de este. Pero estos documentos son privados, y no están disponibles para la compañía que efectúa la ingeniería inversa. En esencia, una ingeniería inversa con éxito precede de una o más especificaciones de diseño y fabricación para el producto, mediante el examen de ejemplos reales de ese producto. La ingeniería inversa del software es algo bastante similar. Sin embargo, en la mayoría de los casos, el programa del cual hay que hacer una ingeniería inversa no es el de un rival, sino, más bien, el propio trabajo de la compañía (con frecuencia efectuado hace muchos años). Los «secretos» que hay que comprender resultan incomprensibles porque nunca se llegó a desarrollar una especificación. Consiguientemente, la ingeniería inversa del software es el proceso de análisis de un programa con el fin de crear una representación de programa con un nivel de abstracción más elevado que el código fuente. La ingeniería inversa es un proceso de recuperación de diseño. Con las herramientas de la ingeniería inversa se extraerá del programa existente información del diseño arquitectónico y de proceso, e información de los datos.

2. DESARROLLE HERRAMIENTAS CASE.

Los CASE fueron, desde sus inicios, un caso interesante. Me gusta mucho la explicación (que he leído en un libro de divulgación) sobre lo que son:

“Voy a desvelar un secreto: ¿Qué es, en realidad, una Metodología? Una serie ordenada de tareas y procedimientos, junto con las técnicas asociadas, que permite a los profesionales del Desarrollo de Aplicaciones cumplir con su trabajo, es decir, escribir software y luego implantarlo y mantenerlo funcionando con la máxima calidad y eficiencia, en el menor tiempo factible, y siempre con el menor coste posible. ¿Sí? ¿Y para qué sirve una Herramienta CASE? Para ayudar a realizar estas tareas y procedimientos con un soporte informático que nos ayude, por un lado, a seguir rigurosamente el método seleccionado y, por otro, a asegurar en todo caso y tiempo la corrección formal de nuestro trabajo, la salvaguarda de la información, su mantenibilidad a lo largo del tiempo… ¿Correcto? Todo esto está muy bien, pero… A nosotros (los informáticos) nos pagan para proporcionar a nuestros Usuarios (el resto de la empresa) una serie ordenada de tareas y procedimientos apoyados por un soporte de software informático que les ayude a, por un lado, seguir los procedimientos de trabajo rigurosamente y, por otro, asegurar la corrección formal de la información crítica de la empresa, su salvaguarda, la capacidad de recuperación ante errores, su mantenibilidad en el tiempo… Volved a leer algunos de párrafos más arriba, si os place, allí donde describía a grosso modo para qué servían las Metodologías. ¿Veis, quizás, alguna coincidencia? ¡Exacto! Con el advenimiento de las Metodologías y de las Herramientas CASE, lo que estaba ocurriendo era, de facto, que estábamos informatizando al Departamento de Informática. Ni más ni menos. ¡Sólo que no lo sabíamos! O no queríamos saberlo, más bien. A mí siempre me ha parecido muy curiosa esta esquizofrenia colectiva de nosotros, los informáticos: nos hemos opuesto sistemáticamente a que alguien nos mecanice nuestro trabajo, cuando eso es precisamente lo que hacemos nosotros continuamente: mecanizar a los demás… Por consiguiente, si lo que deseamos es mecanizar al Departamento de Informática, lo lógico es ni más ni menos que aplicarnos la misma medicina que nosotros aplicamos a nuestros usuarios: Estudio del Sistema Actual, Diseño del Sistema Propuesto, Realización del Sistema e Implantación. Fácil, ¿no os parece?” [Tomado de Memorias de un Viejo Informático].

3. ¿CUÁLES SON LAS HERRAMIENTAS PARA APLICAR LA INGENIERÍA INVERSA?

Los Depuradores. Un depurador es una aplicación que permite correr otros programas, permitiendo al usuario ejercer cierto control sobre los mismos a medida que los estos se ejecutan, y examinar el estado del sistema (variables, registros, banderas, etc.) en el momento en que se presente algún problema. El propósito final de un depurador consiste en permitir al usuario observar y comprender lo que ocurre "dentro" de un programa mientras el mismo es ejecutado.”

Las Herramientas de Inyección de Fallos. Es una técnica para la mejora de la cobertura de una prueba mediante la introducción de faltas para probar las rutas de código, en particular, las rutas de código de gestión de errores que, de otro modo, raramente pueden seguirse.

Los Desensambladores. Se trata de una herramienta que convierte código máquina en lenguaje ensamblador. El lenguaje ensamblador es una forma legible para los humanos del código máquina. Los desensambladores revelan qué instrucciones máquinas son usadas en el código. El código máquina normalmente es específico para una arquitectura dada del hardware. De forma que los desensambladores son escritos expresamente para la arquitectura del hardware del software a desensamblar.

Los compiladores Inversos o Decompiladores. Es una herramienta que transforma código en ensamblador o código máquina en código fuente en lenguaje de alto nivel. También existen decompiladores que transforman lenguaje intermedio en código fuente en lenguaje de alto nivel. Estas herramientas son sumamente útiles para determinar la lógica a nivel superior como bucles o declaraciones if-then de los programas que son decompilados. Los decompiladores son parecidos a los desensambladores pero llevan el proceso un importante paso más allá.

Las Herramientas CASE. Este es un caso especial. Yo conceptuaba a las CASE como herramientas propias de la ingeniería directa, así que tuve que ponerme a buscar por qué fueron listadas dentro de este rubro, el de la ingeniería inversa. Teniendo en mente lo contestado en el punto anterior:

«Los sistemas CASE permiten establecer una relación estrecha y fuertemente formalizable entre los productos generados a lo largo de distintas fases del ciclo de vida, permitiendo actuar en el sentido especificaciones-código (ingeniería "directa") y también en el contrario (ingeniería "inversa"». [Tomado de un apunte del instituto tecnológico David Ausubel, Quito]

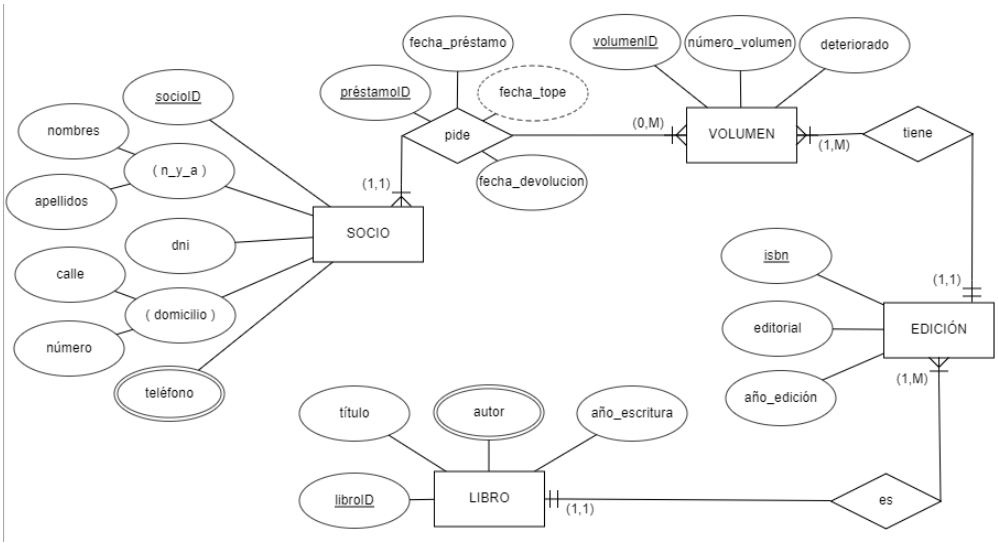
PRÁCTICA (100 PTS)

Debe obtener al menos 60/100 para la aprobación de esta parte. Para todos los puntos, se deberán realizar los diagramas correspondientes con unsoftware de modelado de datos.

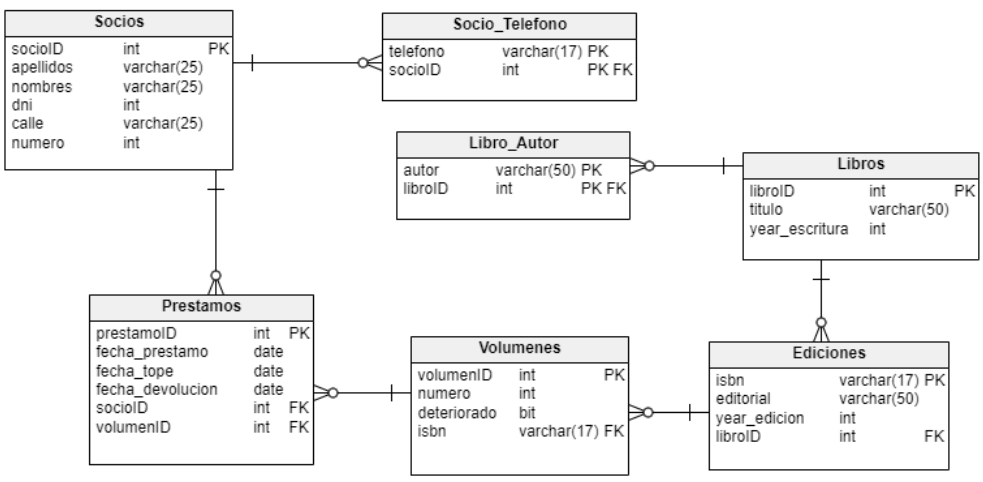
1. **DESARROLLE UN MODELO DE ENTIDAD RELACIÓN (CONCEPTUAL) PARA EL SIGUIENTE ENUNCIADO (30 PTS):**

Crear un diseño entidad relación que permita gestionar los datos de una biblioteca de modo que las personas socias de la biblioteca disponen de un código de socio y datos engeneral, como ser: dni, dirección, teléfono, nombre y apellidos. La biblioteca almacena libros que presta a los socios. De los mismos se almacena su título, su editorial, el año en el que se escribió el libro, el nombre completo del autor (o autores), el año en que se editó, en qué editorial fue y el ISBN. Necesitamos poder indicar si un volumen en la biblioteca está deteriorado o no. Además,queremos controlar cada préstamo que se realiza almacenando la fecha en la que se realiza, la fecha tope para devolver (que son 15 días más que la fecha en la que se realizael préstamo) y la fecha real en la que se devuelve.

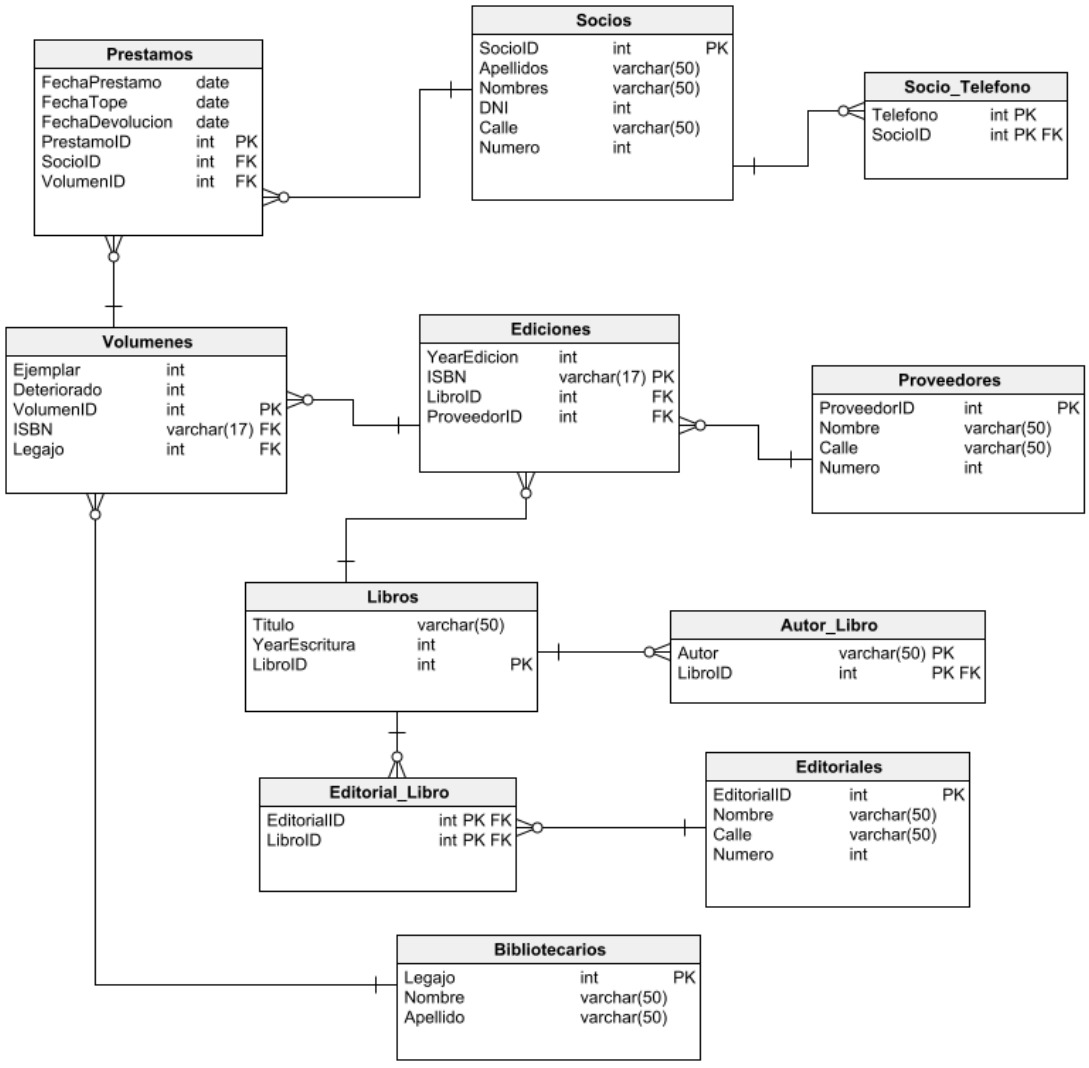
Se necesita la administración de Libros, Editoriales y Proveedores de libros. (Véase punto 3 sobre este último requerimiento).



2. **REALIZAR EL MODELO LÓGICO DEL PUNTO ANTERIOR (30 PTS).**



3. **EN BASE AL MODELO FINAL LÓGICO (DER LÓGICO), INCORPORAR 3 ENTIDADES JUSTIFICANDO SU CARDINALIDAD Y RELACIÓN CON EL RESTO DEL DIAGRAMA (40 PTS).**



JUSTIFICACIÓN:

Se han incorporado 3 entidades (desde el punto de vista del diagrama conceptual): Editoriales, Proveedores y Bibliotecarios, para que atiendan las administraciones pedidas en la última línea del punto 1:

- Se ha agregado una relación de Editoriales hacia Libros.
- Se ha agregado una relación de Proveedores hacia Ediciones.
- Se ha agregado una relación de Bibliotecarios hacia Volúmenes (en el entendimiento de que la tarea de un bibliotecario es, entre otras, el mantenimiento de los ejemplares de un libro).

Editoriales: la cardinalidad (que está dada por una tabla intermedia) dice que cualquier libro en la biblioteca ha sido publicado por una o muchas editoriales (en el caso de los libros de dominio público, es así: por cada libro en biblioteca, varias editoriales pueden estar publicando el mismo libro; por ejemplo, El Quijote de la Mancha, cada una con su propio número de edición).

Proveedores: la cardinalidad viene dada por el hecho de que cada edición en existencia ha sido vendida por un proveedor. A su vez, un proveedor puede vender uno o muchos de los libros en existencia.

Bibliotecarios: esto viene dado, más que nada, por la evaluación que hace un bibliotecario sobre un ejemplar (volumen) deteriorado y, sobre esa evaluación, recuperarlo de la biblioteca y reacondicionarlo

para que se pueda seguir usando. En este sentido, si hay un solo empleado que haga esta tarea, todo libro pasa por él. Por otro lado (en el caso de una biblioteca nueva) puede que exista un momento en el que no haya tenido que hacer aún ningún recupero.