Daniel Ruskov Vangelov
DBD 2020

DISEÑO DE BASES DE DATOS

Objetivos de la asignatura - analizar diferentes técnicas de diseño de un SGBD (conceptual, lógico, físico).
SGBD - Sistema de Gestión de Bases de Datos - software dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan.

ETAPAS DE DISEÑO

Ciclo de vida de un SBD - dentro del ciclo de vida de un sistema de información, es el proceso continuo de creación, mantenimiento, mejora y reemplazo que constituye el Ciclo de Vida de las Bases de Datos.

Etapas del ciclo de vida:

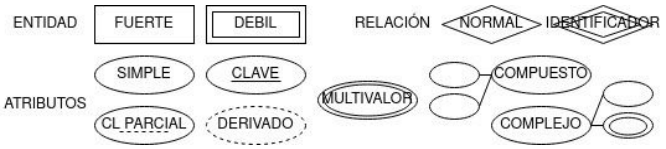
1. **Definición sistema** - estrategia para determinar las necesidades de información de un extenso periodo de tiempo, es decir, análisis, definición del ámbito, usuarios, aplicación, ventajas, viabilidad, problemas y soluciones, restricciones, objetivos, alcance, límites. El final de la etapa se marca con un informe que contenga:
   - 1.1. Necesidades de información de las áreas funcionales.
   - 1.2. Necesidades de información de los diferentes niveles de dirección.
   - 1.3. Necesidades de información de las localidades geográficas.
   - 1.4. Un esquema de estas necesidades de información.
   - 1.5. Volúmenes previstos de datos por localidad geográfica.
   - 1.6. Estimación preliminar de los costos asociados a las actualizaciones del sistema.
   - 1.7. Recomendaciones para un desarrollo de las BD.
   - 1.8. Viabilidad tecnológica (recursos), operacional (trabajo a realizar) y económica (dinero a pagar).
2. **Diseño de la BD** - proceso iterativo para satisfacer los requerimientos obteniendo una estructura lógica y física de una BD utilizando un SGBD concreto manteniendo objetivos del rendimiento como tiempo de respuesta, espacio de almacenamiento, etc. Fases:
   - 2.1. Análisis requisitos - identificar y analizar el uso que se va a realizar de la BD, requerimiento de los datos, usuarios, tipo de transacciones y frecuencia, etc.
   - 2.2. Diseño conceptual - diseño de las transacciones, diseño de esquema conceptual en modelo de datos de alto nivel independiente de cualquier SGBD como resultado para que se entiendan las estructuras de la BD y su semántica y relaciones. Identificación de las entidades y atributos, claves, restricciones de cardinalidad (1:1, 1:N, M:N) y participación (total, parcial), entidades débiles, generalización/especialización y categorías. Técnicas de aproximación centralizada (se unen todos los requerimientos de los diferentes usuarios creando un esquema, después se especifican diferentes esquemas externos para los usuarios) y aproximación por integración de vistas (se diseña esquema distinto para cada grupo de usuarios, luego se integran obteniendo esquema global). En la integración de vistas se identifican las correspondencias y conflictos entre esquemas y se reestructuran modificando y mezclando para crear el esquema global creando enlaces entre ellas. Los tipos de correspondencias pueden ser equivalentes, inclusión, solapamiento o disyunción entre el tipo de entidad A y el tipo de entidad B. También existen los conflictos de nombre (sinónimos, homónimos), de tipo, de dominio y entre restricciones. Estrategias de diseño de esquemas conceptuales usadas: Top-Down, Bottom-Up, Inside-Out y/o Mixta.
   - 2.3. Selección SGBD -tener en cuenta factores tecnicos (funciones y capacidades, lenguaje, mecanismos de seguridad e integridad, etc), económicos (adquisición y mantenimiento), organización.
   - 2.4. Diseño lógico - esquemas conceptual y externos se definen de acuerdo al modelo de datos del SGBD seleccionado. Creación de tablas. Si el modelo usado es relacional, se consideran los procesos de normalización, definición de vistas y restricciones de integridad.
   - 2.5. Diseño físico - selección de las estructuras de almacenamiento específicas y los caminos de acceso para los ficheros que forman la BD con el fin de obtener un buen rendimiento de las diferentes aplicaciones. Criterios como tiempo de respuesta, espacio utilizado, throughput (productividad medida en número medio de transacciones que se pueden ejecutar por minuto/granularidad de tiempo). Técnicas de simulación.
   - 2.6. Implementación - se compilan los diferentes esquemas (conceptual, externo e interno). Además se implementan las aplicaciones.
3. **Implementación** - escribir las definiciones de los esquemas conceptual, externo e interno, creando la BD vacía.
4. **Carga o conversión de datos** - la BD se carga introduciendo directamente o convirtiendo ficheros existentes en el formato de la BD y cargándose.
5. **Conversión de aplicaciones** - aplicaciones de sistemas previos se convierten al nuevo sistema.
6. **Comprobación y validación -** se comprueba y valida el nuevo sistema.
7. **Operación** - utilización de la BD.
8. **Control y mantenimiento** - introducir cambios en los datos y/o en las aplicaciones, necesarias reorganizaciones periódicas.

MODELO ENTIDAD RELACIÓN

Definición de modelo de datos (MD) - representar los datos a un nivel más abstracto que los esquemas. Se compone de estructuras de datos, operaciones para su manejo y reglas de integridad que especifican los estados consistentes de la BD. No incluyen detalles de implementación y permiten centrarse en los aspectos de diseño lógico. Se pueden clasificar en familias como Pre-Relacional/Legados (modelo jerárquico, modelo en red), Relacional o Post-Relacional (modelo semántico, modelo orientado a objetos, NoSQL) y según los conceptos que proporcionan para describir la BD como Conceptuales (MD de alto nivel como E/R, UML), Físicos (MD de bajo nivel, detalle de almacenamiento) o MD de representación o implementación.

Modelo E/R

1. **Ventajas**:
   - 1.1. Simplicidad.
   - 1.2. Representación visual.
   - 1.3. Herramienta de comunicación efectiva.
   - 1.4. Integrado al modelo de BD relacional.
2. **Desventajas**:
   - 2.1. Representación limitada de restricciones.
   - 2.2. Representación limitada de relaciones.
   - 2.3. Representación limitada de la semántica del SI.
   - 2.4. Ningún lenguaje de manipulación de datos.
3. **Esquema conceptual** - descripción concisa de los requisitos de información de los usuarios sin detalles de implementación como tipos de datos.



   - 3.1. Tipo entidad/intensión (ej: Libro) - agrupa a un conjunto de entidades que posee el mismo conjunto de atributos. No puede haber ningún tipo entidad sin ningún atributo definido. Cada tipo de entidad se describe con su nombre y lista de atributos. Sus entidades son el conjunto de entidades o extensión. El tipo entidad se llama intensión de su conjunto de entidades.
   - 3.2. Entidad (ej: Libro_1) - algo físico o conceptual del mundo real con existencia independiente. Descrito mediante su conjunto de atributos. El conjunto de entidades de un tipo entidad se denomina extensión. Al ser conjunto, no puede haber entidades repetidas. Cada entidad del tipo de entidades contiene un valor para cada uno de los atributos definidos en el tipo de entidades.
   - 3.3. Entidad débil - entidad que carece de atributos clave propios. Tiene clave parcial y se relaciona con tipo de entidad propietario/identificador. Sus entidades se identifican por la clave parcial concatenada con la clave de la entidad relacionada mediante el tipo de relación identificador. Eso supone que la participación con este tipo de relación debe ser siempre (1,1) cuando no hay clave parcial, para que haya siempre un valor que complete la clave y sólo uno. Puede ser (1, N) con la clave parcial. Puede haber varios niveles de tipo de entidad débil. También, un tipo de entidad débil puede tener varios tipos de entidad propietarios y que su clave está compuesta por la concatenación de las claves de esas entidades identificadoras. En casos particulares se puede representar como atributo complejo siempre que no existan otros tipos de relación. Una solución artificial es inventar un código para usarlo como clave de la entidad débil convirtiéndola en fuerte.
   - 3.4. Atributo (ej: Nombre) - propiedad de las entidades del tipo entidad.
   - 3.5. Valor de atributo (ej: The witcher) - valor asignado en una entidad para un atributo.
   - 3.6. Tipos de atributos:
     - 3.6.1. Monovalor - valor único.
     - 3.6.2. Multivalor - varios valores (ej: localidades - Logroño, Ventosa, Navarrete).
     - 3.6.3. Almacenado - simple.
     - 3.6.4. Derivado - calculado a partir de otro (ej: edad - calculado a partir de la fecha de nacimiento).
     - 3.6.5. Simple - atómico (monovalor, derivado, etc).
     - 3.6.6. Compuesto - jerarquía de varios niveles que representa la concatenación de todas sus componentes simples.
     - 3.6.7. Complejo - anidaciones arbitrarias de atributos compuestos y multivalor.
     - 3.6.8. Clave - tiene valor único para cada entidad del tipo entidad (restricción de clave o unicidad). Se deberá cumplir para cualquier extensión. Puede ser un atributo compuesto. Algunos tipo entidad pueden tener más de un atributo clave. Puede haber tipo entidad sin clave denominado tipo de entidad débil. Un atributo clave nunca puede ser nulo.
     - 3.6.9. Valor nulo - no aplicable (no existe, ej: título universitario), falta (existe, pero no se conoce, ej: altura de una persona) o no se sabe si existe (ej: teléfono de una persona).
   - 3.7. Dominio de un atributo - conjunto de todos los posibles valores que puede tomar el atributo (tipo de datos).
   - 3.8. Tipo de relación - define un conjunto de asociaciones entre varios tipos de entidad, o conjunto de relaciones entre las entidades de los tipos de entidad. Cada relación es una asociación de entidades que incluye una única entidad de cada tipo de entidad que participa en el tipo de relación (participación, cardinalidad). Propiedades:

     - 3.8.1. Grado - es el número de tipos de entidad que participan en un tipo de relación.
     - 3.8.2. Razón de cardinalidad - número de relaciones en las que puede participar una entidad (1:1, 1:N, M:N).
     - 3.8.3. Restricción de participación:
       - 3.8.3.1. Total - dependencia de existencia. Toda entidad debe participar en al menos una relación (ej: todo departamento debe tener un director).
       - 3.8.3.2. Parcial - algunas entidades no participan en ninguna relación (ej: no todos los empleados son directores).

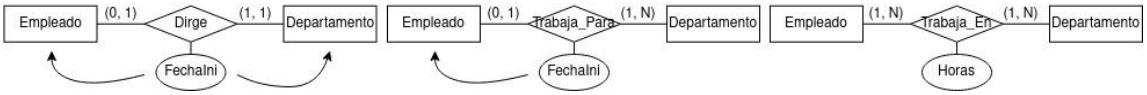
3.8.4. Notación (MIN, MAX) - permite números diferentes de 0, 1 y/o N. Indican respectivamente el mínimo y máximo de relaciones en las que participa una entidad (0<=MIN<=MAX>=1). MIN=0 indica participación parcial y MIN>0 indica participación total.

3.8.5. Notación alternativa - menos precisa. Tener en cuenta que la información sobre una entidad se encuentra repartida a ambos lados del tipo de relación (la participación al lado y la cardinalidad en el lado opuesto).

3.8.6. Rol - indica el papel desempeñado por las entidades en las relaciones. No son necesarios si los tipos de entidad relacionados son distintos, pero si son obligatorios si se repiten tipos de entidad en el tipo de relación (normalmente recursivos).

3.8.7. Tipos recursivos - un mismo tipo de entidad participa varias veces.

3.8.8. Atributos de un tipo de relación - no pueden ser clave. La clave de los tipos de relación 1:1 puede ser cualquiera de los dos tipos de entidad. en 1:N el tipo de entidad con cardinalidad 1 y en M:N ambos tipos de entidad de manera conjunta. Estos atributos no modifican la clave de un tipo de relación. Según sea 1:1 o 1:N el atributo del tipo de vínculo puede situarse alternativamente en alguno de los tipos de entidad participantes, como muestran las flechas en los ejemplos.



3.8.9. Relaciones N-arias. Propiedades:

3.8.9.1. Se utiliza un doble etiquetado (ver imagen del apartado 3.8.10).

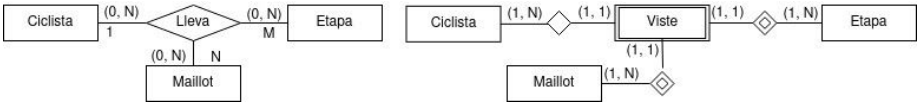
3.8.9.2. La notación mínimo-máximo se mantiene con el mismo significado que para las binarias (la alternativa no se puede utilizar).

3.8.9.3. La N de CICLISTA significa que cada entidad de CICLISTA puede participar en varias relaciones de LLEVA.

3.8.9.4. La otra notación (1:N:M en el ejemplo), añade la idea de clave. Por tanto, la clave del tipo de relación LLEVA la forman la entidad de MAILLOT y la de ETAPA, ya que en una etapa sólo se puede imponer un determinado maillot a un único ciclista.

3.8.9.5. Si no se indica esta segunda notación se entiende que es: M:N:P.

3.8.10. Transformación de un tipo de relación ternario en un tipo de entidad débil.

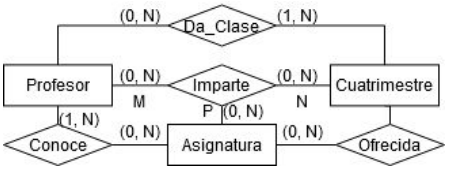


3.8.10.1. El tipo de entidad débil VISTE representa lo mismo que el tipo de relación ternario LLEVA.

3.8.10.2. El tipo de relación entre VISTE y CICLISTA no es identificador, porque no forma parte de la clave de LLEVA (pero el tipo de relación con CICLISTA es necesario).

3.8.10.3. Obsérvese que en este caso VISTE no precisa usar clave parcial.

3.8.11. Posibilidad de tener un tipo de relación ternario y los tres binarios correspondientes, pero es preciso definir restricciones de manera que haya coherencia entre los trios.



3.9. Convenio para los nombres de los elementos de los esquemas E/R (no usado en las imágenes) . Pautas:

3.9.1. Tipos de entidad - nombres en singular y mayúsculas.

3.9.2. Tipos de relación - verbos en mayúsculas. Ordenado de izda a dcha o de arriba hacia abajo, de manera que se pueda leer con sentido.

3.9.3. Atributos - nombres en singular con primera letra en mayúscula.

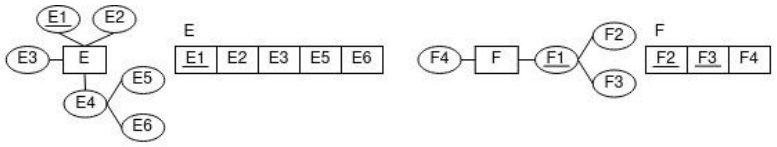
3.9.4. Roles - en minúscula.

4. **Esquema lógico** - obtención a partir del esquema conceptual.

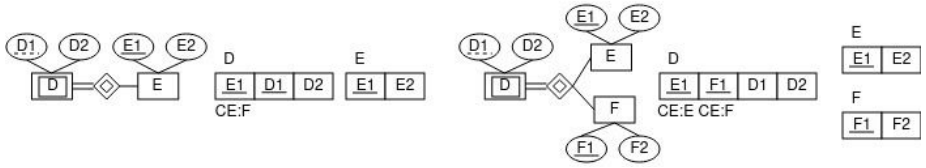
5. **Esquema físico** - depende de las transacciones (consultas, inserciones, modificaciones).

Transformacion modelo E/R al modelo relacional:

1. **Entidades fuertes.**

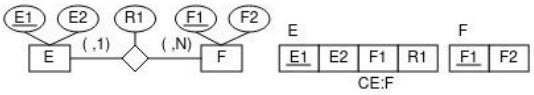


2. **Entidades débiles.**



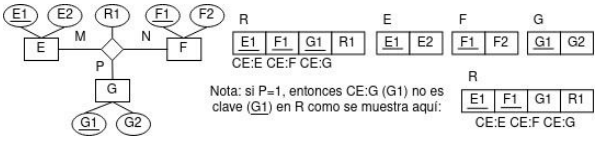
3. **Herencias simple y múltiple** - métodos A, B, C y/o D.
4. **Vínculos 1:1.**

5. **Vínculos 1:N.**

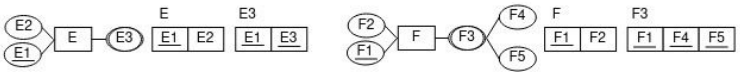


6. **Vínculos M:N** - se puede usar también para 1:1 y 1:N (donde la clave de R en 1:N es la de la entidad del lado N y en 1:1 la del lado con participación total). Mejor cuando haya pocos vínculos (menos nulos).

7. **Vínculos N-arios.**

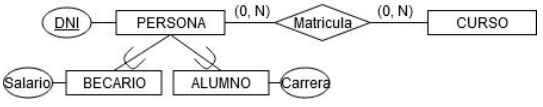


8. **Atributos multivaluados.**



Modelo E/R+ (EER, extendido) - para requisitos más complejos. Añade al E/R mecanismo de herencia de atributos y relaciones. El mecanismo de herencia se ha desarrollado en otras áreas de la informática como los modelos semánticos de datos, el modelo orientado a objetos o la representación del conocimiento.

1. **Características:**
   1.1. Cada subclase representa a un subconjunto de las entidades de la superclase.
   1.2. La relación entre superclase con cualquiera de sus subclases se llama relación IS-A (es un).
   1.3. Toda entidad de una subclase será al mismo tiempo entidad de su superclase.
   1.4. Una entidad puede ser al mismo tiempo miembro de varias subclases: por ejemplo el secretario “jon” puede ser también un asalariado.
   1.5. Una entidad de la superclase no tiene por qué ser al mismo tiempo miembro de alguna subclase.
2. **Definición de subclases**:
   2.1. Por condición o atributo - cuando todas las subclases están definidas por una condición sobre el mismo atributo de la superclase que define la subclase a la que pertenece. La condición se especifica encima del círculo de especialización.
   2.2. Por el usuario - cuando no existe una condición para determinar la pertenencia a una subclase.
3. **Herencia** - es el mecanismo por el cual entidades más específicas (de subclases) incorporan atributos y relaciones definidos para entidades más generales (de superclases). Ejemplo: Becario y Alumno tienen sus propios atributos específicos, pero los dos heredan los atributos de Persona y Curso



   3.1. Jerarquía - toda subclase participa en una única relación superclase/subclase. Se da lugar a una estructura en árbol.

3.2. Herencia desde la raíz - se hereda de la superclase, pero además de la superclase de la primera y así sucesivamente hasta la superclase raíz.

3.3. Relaciones de la raíz - una subclase con sus atributos específicos más los atributos y relaciones heredados es un tipo de entidades. por eso se representan con rectángulo.

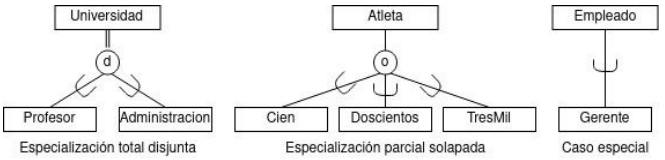
3.4. Herencia múltiple - se hereda el mismo atributo de varias superclases. Se produce un conflicto donde el diseñador resuelve explícitamente de dónde se hereda.

3.5. Retículas - hay alguna subclase que participa en más de una relación. Estas se llaman subclases compartidas y heredan todo desde la raíz y todas sus superclases mediante herencia múltiple. Cuando los mismos atributos se heredan desde varios caminos, las reglas de herencia establecen que debería incluirse una sola vez cada elemento y no hay conflicto.

4. **Generalización** - se identifican rasgos comunes entre varios tipos de entidad y se crea una superclase para todos ellos. Refinamiento Bottom-Up.

5. **Especialización** - proceso de dividir un tipo de entidad en subclases a partir de alguna característica distintiva con una misma superclase. Lo contrario a generalizar, Top-Down.

5.1. Propiedades:



5.1.1. De completitud:

5.1.1.1. Total (||)- cada entidad de la superclase tiene que formar parte de alguna subclase.

5.1.1.2. Parcial (|)- algunas entidades de la superclase no aparecen en las subclases.

5.1.2. De disyunción o solapamiento:

5.1.2.1. Especialización disjunta (d) - una entidad de la superclase solo podrá ser miembro de una subclase. Si las subclases se definen por atributo, éste será monovalor.

5.1.2.2. Especialización solapada (overlapped, o) - una entidad de la superclase podrá ser miembro de varias subclases. Es la opción por defecto si no se especifica en el esquema.

5.1.2.3. Caso especial - el predicado da lugar a una sola subclase.

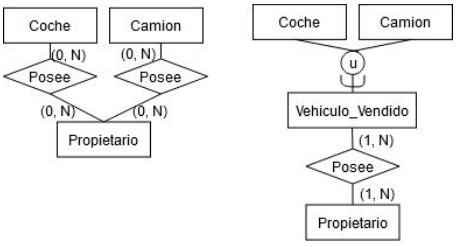
5.2. Reglas de insercion y eliminacion:

5.2.1. Eliminación - una entidad c1 se elimina de una superclase C, entonces se elimina automáticamente de todas las subclases a las que pertenece.

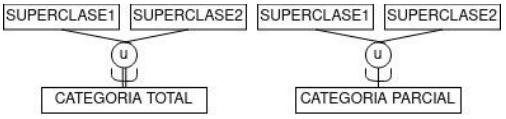
5.2.2. Inserción - una entidad en una superclase. Si las subclases están definidas por una condición o atributo, se podrá insertar automáticamente en las subclases que le corresponden. Se insertará al menos en una subclase si la especialización es total.

6. **Necesidad de especialización/generalización** - atributos que se aplican sólo a algunas entidades del tipo de entidades o relaciones donde sólo pueden participar algunas entidades del tipo de entidades.

7. **Categoría/tipo unión** (u) - se usa para reducir el número de relaciones (ej: las entidades de Vehiculo_Vendido son un subconjunto de la unión de las entidades de coche y camión. en este caso en Vehiculo_Vendido están sólo los coches y camiones vendidos y el resto no. Vehiculo_Vendido es una subclase compartida categoria).



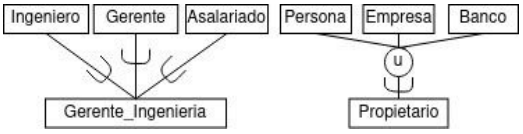
7.1. Tipos de categorías**:**



7.1.1. Total - contiene la unión de todas las entidades de sus superclases.

7.1.2. Parcial - contiene un subconjunto de la unión de las entidades de sus superclases.

7.2. Diferencia entre subclase compartida y categoría. Ejemplo:



7.2.1. Subclase compartida GERENTE_INGENIERIA:

7.2.1.1. Cualquiera de sus entidades debe existir en las tres superclases.

7.2.1.2. Es un subconjunto de intersección de las entidades de sus superclases.

7.2.1.3. Cada entidad hereda los atributos de todas sus superclases.

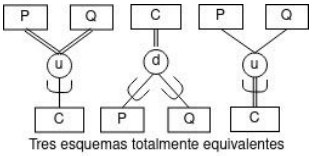
7.2.2. Categoría PROPIETARIO:

7.2.2.1. Cualquier entidad de propietario debe existir sólo en una de sus superclases.

7.2.2.2. Es un subconjunto de la unión de las entidades de sus superclases.

7.2.2.3. Cada entidad hereda los atributos de una de sus superclases.

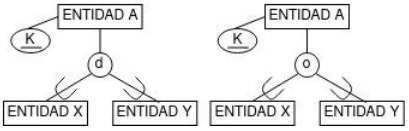
7.3. Equivalencia entre especialización y categoría - si una categoría es total, puede representarse alternativamente como una especialización total y disjunta. En el caso de que la categoría no incluyese todas las entidades, la especialización no sería una alternativa válida.



Tres esquemas totalmente equivalentes

Transformación esquema E/R+ al modelo relacional:

1. **Especialización/generalización.**



1.1. Método A - para TODO TIPO de especialización/generalización se pasa la clave de la superclase en las subclases. Operación de equijoin sobre la clave primaria, entre cualquier subclase y la superclase, produce todos los atributos específicos y heredados de las entidades subclase.

ENTIDAD A, K + ENTIDAD X, K + ENTIDAD Y

1.2. Método B - para especialización/generalización DISJUNTAS y TOTALES. No se representa la superclase (no se necesita equijoin). Siempre que busquemos una entidad arbitraria de la superclase, deberemos buscar en todas las tablas de las subclases. Otros autores mejoran la propuesta con una vista (vista para la Entidad A). Si solapada, redundancia (de los atributos heredados). Si parcial, elementos no representados. La superclase se obtiene con OUTER UNION.

ENTIDAD A + ENTIDAD X, ENTIDAD A + ENTIDAD Y

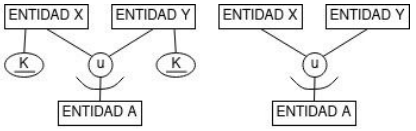
1.3. Método C - para especialización/generalización DISJUNTAS. Aparece un nuevo atributo T, que indica el tipo de la subclase que se utiliza. No se recomienda si X o Y tienen muchos atributos. Si tienen pocos, mejor (evitan JOIN o UNION), Si parcial, T=NULL (y valores Nulos para los atributos específicos de las subclases).

ENTIDAD A + T + ENTIDAD X + ENTIDAD Y

1.4. Método D - para especialización/generalización SOLAPADAS. Secuencia de bits = bb (tantos como subclases) que indica si los atributos de x e y son válidos para la instancia (m campos de tipo booleano, uno por cada subclase, 1 campo por cada m bits). No se recomienda si X o Y tienen muchos atributos, Si tienen pocos, mejor (evitan JOIN o UNION).

ENTIDAD A + bb + ENTIDAD X + ENTIDAD Y

2. **Categoría.**



2.1. Claves iguales - si X e Y comparten clave K, A la usa como clave.

K + ENTIDAD A, ENTIDAD X, ENTIDAD Y

2.2. Claves diferentes - se usa clave sustituta Ks para A, y clave extranjera en X e Y.

Ks + ENTIDAD A, ENTIDAD X + Ks, ENTIDAD Y + Ks.

3. **Subclase compartida.**

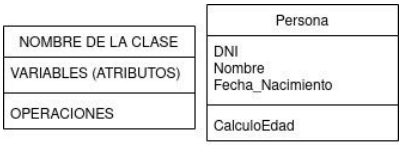
3.1. Superclases con claves iguales - como especialización, recomendado método A.

3.2. Superclases con claves diferentes - como una categoría.

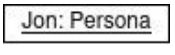
Modelo UML - bloques de construcción de Unified Modeling Language.

1. **Elementos** - abstracciones. Los elementos estructurales son las partes estáticas de un modelo y representan cosas que son conceptuales o materiales.

1.1. Clase - descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica.



1.2. Objeto - manifestación concreta de una clase.

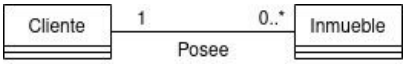


1.3. Variable/atributo - propiedad de una clase identificada con un nombre. representa alguna propiedad del elemento que se está modelando que es compartida por todos los objetos de esa clase.

2. **Relaciones** - conexiones entre elementos.

2.1. Multiplicidad/cardinalidad - número de relaciones entre objetos (1, 0..1, *, 0..*, 1..*, n, m..n, n..*) conectados a través de una instancia de una asociación.

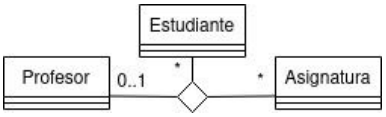
2.2. Relación asociación - relación estructural que describe un conjunto de enlaces, los cuales son conexiones entre objetos. Se le puede dar nombre (ej: un cliente posee 0 a N inmuebles, y un inmueble pertenece a un cliente).



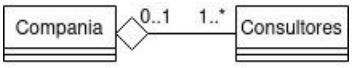
2.2.1. Grado:

2.2.1.1. Binario - ejemplo de la imagen anterior.

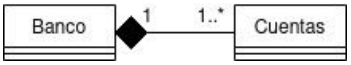
2.2.1.2. N-ario (ej: par<est, asig> conocidos con 0 o 1 prof. Un estudiante se puede matricular en una asignatura con uno de los profesores que la imparte, o no hacerlo):



2.2.2. Agregación simple - modela relación todo/parte en la cual una clase representa una cosa grande que consta de elementos más pequeños.

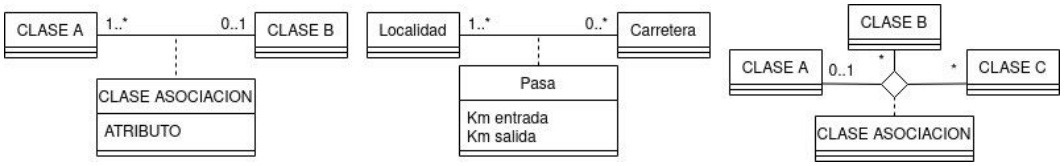


2.2.3. Composición - variación de la agregación simple. Las partes una vez creadas, viven y mueren con la parte compuesta. La única cardinalidad con sentido es 1.

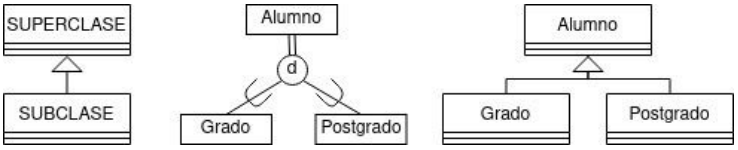


2.2.4. Cualificadas - semejante a la notación de tipo entidad débil.

2.2.5. Clases asociación - asociación entre dos clases donde la propia asociación puede tener propiedades. Puede ser binaria o N-aria.



2.2.6. Generalización/especialización - relación entre elemento general (superclase o padre) y un tipo más específico de ese elemento (subclase o hijo). El hijo hereda la estructura y comportamiento del padre, y el hijo es un sustituto del padre (las instancias del hijo pueden usarse donde quiera que se puedan usar las instancias del padre). Hay cuatro restricciones estándar que se aplican a las relaciones de generalización (complete, incomplete, disjoint y overlapping). El hijo puede añadir nueva estructura y comportamiento o modificar el comportamiento del padre. Existe la noción de herencia múltiple.



3. **Diagrama** - representación gráfica de un conjunto de elementos, visualizado casi siempre como grafo conexo de nodos (elementos) y arcos (relaciones).

NORMALIZACIÓN

¿Que es un buen diseño?¿Que implica?

1. **Pautas:**
   - 1.1. Esquema de una relación fácil de explicar.
   - 1.2. Evitar combinar información sobre más de una entidad en la misma tabla. Descomposición.
   - 1.3. Minimizar el número de anomalías.
   - 1.4. De-normalización para ganar eficiencia, admitiendo redundancia e indicándose claramente con las razones que llevan a ello.
   - 1.5. Evitar atributos con valores nulos. Si muchos de los atributos no se aplican a todas las tuplas de la relación, acabaremos con un gran número de nulos en esas tuplas. Puede originar desperdicio de espacio de almacenamiento. Esto incluye valores NULL, N/A, Unknown, ausente. Evitar incluir en una relación base atributos cuyos valores pueden ser nulos.
   - 1.6. Diseñar relaciones que puedan ser agrupadas mediante joins basados en atributos que son clave primaria o clave extranjera a fin de garantizar que no se formarán tuplas espurias. Las tuplas espurias representan información no válida.

Dependencias:

1. **Funcionales** (DF) - son propiedades de la semántica o del significado de los atributos. Una DF es una propiedad de la intensión de R, no de la extensión de R (no puede inferirse automáticamente). Los diseñadores de bases de datos especifican primero el conjunto de dependencias funcionales F que se pueden determinar sin dificultad a partir de la semántica de los atributos. Luego se usan las reglas para inferir las dependencias funcionales adicionales.
   - 1.1. Explicación - dados dos conjuntos de atributos A y B, subconjuntos de un esquema R, diremos que B depende funcionalmente de A (A→B), si para cualquier extensión r(R) se verifica que para todo par de tuplas t1, t2 de r t1.A=t2.A → t1.B=t2.B. Es decir, conocido el valor de A, se identifica unívocamente al valor de B (determinante → atributo).
   - 1.2. Clave candidata - si X es clave candidata de R, entonces X→Y para todo el subconjunto Y de R.
   - 1.3. Reglas de Armstrong - se pueden deducir DF a partir de otras, utilizando las siguientes reglas correctas y completas:
     - 1.3.1. Reflexividad - si Y⊆X, entonces X→Y. Dependencia trivial. Ejemplo: nombre ⊆ {nombre,dni}, entonces nombre, dni → nombre (un conjunto de atributos siempre se determina a sí mismo).
     - 1.3.2. Aumentación - si X→Y, entonces XW→YW. Ejemplo: laAsig → elProf, entonces laAsig, grupo → elProf, grupo.
     - 1.3.3. Transitividad - si X→Y y Y→Z, entonces X→Z. Ejemplo: laAsig → elProf y elProf → dpto, entonces laAsig → dpto.
     - 1.3.4. Union - si X→Y y X→Z, entonces X→YZ.
     - 1.3.5. Pseudotransitividad - si X→Y y WY→Z, entonces XW→Z.
     - 1.3.6. Descomposición - si X→Y y Z⊆Y , entonces X→Z.
   - 1.4. Cierre de un conjunto de dependencias funcionales F - el cierre F^+ es el conjunto de dependencias funcionales que F implica lógicamente, es decir, que se pueden deducir utilizando los axiomas de Amstrong. Dos conjuntos de DF, E y F son equivalentes si $E^+=F^+$.
   - 1.5. Superclave - K es super-clave en una relación R si para cualquier extensión de R, no puede haber dos tuplas con el mismo valor para KhaberdostuplasconelmismovalorparaK. K es clave si K^+ incluye todos los atributos de R. Un conjunto de atributos A de la relación R es superclave si A →R.
   - 1.6. Clave candidata - un conjunto de atributos A de la relación R es superclave mínima si A es superclave y ∀B | B⊂A, A-B no es superclave.
   - 1.7. Clave primaria - es una de las claves candidatas.
   - 1.8. Atributo primo - es aquel que forma parte de de una clave candidata.
   - 1.9. Atributo no primo - aquel que no figura en ninguna clave candidata.
   - 1.10. Recubrimiento mínimo - dado el conjunto de DF F, se trata de buscar otro conjunto F_c, tal que equivalga a F, pero el número de dependencias a comprobar sea mínimo. F es un recubrimiento mínimo de G,si F es mínimo y equivalente a G (ej: {X→Z, X→Y} es un recubrimiento mínimo para {XY→Z, X→Y}). Siempre se puede hallar por lo menos un recubrimiento mínimo. DF es mínimo si:
     - 1.10.1. Toda dependencia en F tiene un solo atributo en la parte derecha. Esto es para garantizar que todas las DF tienen la misma forma estándar de cara a ser procesadas por los algoritmos. Mediante la regla de la descomposición se puede convertir toda DF a la forma estándar.
     - 1.10.2. No podemos quitar ninguna dependencia de F, y seguir teniendo un conjunto de dependencias equivalente a F. Esto es con motivo de quitar las dependencias redundancias.
     - 1.10.3. No podemos reemplazar ninguna dependencia X→A por una dependencia Y→A donde Y⊆X y seguir teniendo un conjunto de DF equivalente a F.
2. **Transitivas** - sean A, B y C conjuntos de atributos de un esquema R. C es transitivamente dependiente de A bajo R si A→B, B↛A y B→C.
3. **Multivaluadas** (DMV) - A→→B es una dependencia multivaluada trivial en cualquiera de los dos casos: A∪B→R y B⊆A. Generalización de dependencia funcional. Los valores que toman los atributos de B son independientes de los tomados por los de R-A-B. Si A→→B, entonces A→→R-B. Generalmente las relaciones que contienen DMV no triviales suelen ser aquellas en las que la clave está formada por todos los atributos.

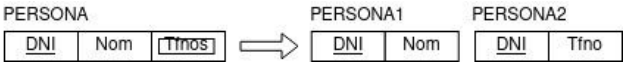
Descomposición de un esquema R - esquema equivalente a R mejorado {R1, R2, … , Rn}, conservando los atributos, contenido y dependencias y minimizando la redundancia.

1. **Propiedades**:
   - 1.1. Primera - Ri es un resultado de una proyección sobre R.
   - 1.2. Segunda - todo atributo de R se encuentra en alguna de las proyecciones Ri (R=R1∪R2∪...∪Rn).
   - 1.3. Tercera - para cualquier extensión r de R se cumple r⊂r1⋈r2⋈...⋈rn. Lo contrario no es siempre cierto, ya que el join de las proyecciones puede generar tuplas espurias.

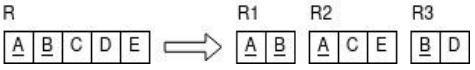
2. **Descomposición sin pérdida de un esquema** - un conjunto de relaciones {R1, R2, … , Rn} es una descomposición sin pérdida de R si para cualquier extensión r de R se cumple que r=r1⋈r2⋈...⋈rn, donde ri es la proyección de r sobre los atributos Ri, es decir, cuando no se generan tuplas espurias (ej: sea F = {R1, R2} conjunto de dependencias funcionales en R. F es una descomposición sin pérdida de R si por lo menos una de estas dependencias están en F^+. R1∩R2→R1, es decir R1∩R2 es clave candidata de R1 y R1∩R2→R2, es decir R1∩R2 es clave candidata de R2).

Normalización - metodología de diseño que minimiza el número de posibles problemas. Es un proceso de análisis de los esquemas de relación basado en las dependencias funcionales y claves primarias para alcanzar las propiedades deseables de minimizar redundancias y con ello las anomalías. Un esquema se somete a una serie de pruebas para certificar si pertenece a cierta forma normal. La teoría de la normalización establece guías sobre cuándo descomponer una relación y cómo.

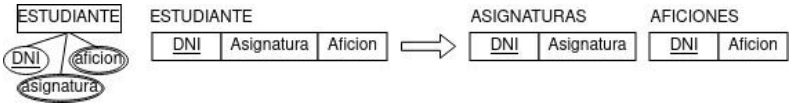
1. **Objetivo** - mejorar y validar el diseño lógico, evitar redundancias, relaciones bien estructuradas y asegurar que cada tabla trata de un único concepto. Lograr diseños estructuralmente consistentes, siguiendo el criterio de calidad "mínima redundancia, cero anomalias".
2. **Motivación** - los esquemas no normalizados tienen redundancias que producen desperdicio de espacio y anomalías de inserción, borrado y/o modificación (inconsistencias). Estas anomalías se producen por relaciones que contienen más de un tipo de entidad.
3. **Solución** - descomponer las relaciones para aislar las entidades, pero preservando la información y dependencias.
4. **Formas Normales** - (1FN), dependencias funcionales (2FN, FNBC), dependencias multivaluadas (4FN) y dependencias de join (5FN). Buen diseño no implica llegar a la FN más alta.
   - 4.1. 1FN - una relación está en primera forma normal si y sólo si ninguna de sus tuplas tiene elementos que sean conjuntos.
     - 4.1.1. Solución - descomposición en tablas para eliminar los atributos multivaluados. Ejemplo:



   - 4.2. 2FN - una relación está en segunda forma normal si y sólo si está en 1FN y todo atributo no primo es totalmente dependiente de las claves candidatas. El Atributo primo es aquel que pertenece a alguna clave candidata.
     - 4.2.1. Solución - las dependencias parciales se llevan a nuevas tablas, mientras en la tabla original queda la clave y los atributos que dependen totalmente de ella. Si hay dependencias parciales con una clave candidata en lugar de con una clave primaria, se resuelve de la misma forma que si la clave candidata fuera clave primaria. Ejemplo con AB→CDE, A→CE, B→D:



   - 4.3. 3FN - una relación está en tercera forma normal si y sólo si está en 2FN y ningún atributo no primo tiene dependencias transitivas respecto a las claves candidatas. Todo atributo no clave depende sólo completamente de la clave.
     - 4.3.1. Solución - para toda dependencia X→Y, X es superclave o Y es atributo primo.
   - 4.4. BCFN - una relación está en forma normal de Boyce-Codd si y sólo si para toda dependencia X→Y, X es superclave o clave. Toda relación BCFN está en 3FN, pero no viceversa. Todo atributo depende sólo completamente de la clave.
     - 4.4.1. Solución - descomponer la tabla basándose en las dependencias, sabiendo que se puede recomponer la tabla inicial realizando el join de las proyecciones.
     - 4.4.2. Problema - no se garantiza la conservación de dependencias funcionales. Barajar las diferentes descomposiciones posibles que conserven las dependencias. También aparecen redundancias (varias tuplas con la misma clave) y valores nulos.
   - 4.5. 4FN - una relación está en cuarta forma normal si y sólo si por cada dependencia multivaluada no trivial X→→Y, X es superclave para R.
     - 4.5.1. Solución - descomponerse en varias relaciones que contienen DMV triviales. Ejemplo (DNI→→asignatura, DNI→→afición; asignaturas y aficiones son hechos independientes):



   - 4.6. 5FN - no explicada.
5. **Resumen**:
   - 5.1. De relación a 1FN - eliminar atributos multivaluados y compuestos.
   - 5.2. De 1FN a 2FN - eliminar dependencias parciales.
   - 5.3. De 2FN a 3FN - eliminar dependencias transitivas.
   - 5.4. De 3FN a BCFN - eliminar dependencias de claves no candidatas.
   - 5.5. De BCFN a 4FN - eliminar dependencias multivaluadas.
   - 5.6. De 4FN a 5FN - eliminar dependencias de join.

De-normalización - proceso de introducción de cierta redundancia (juntar tablas), utilizando formas normales más bajas. Esto hace que la implementación sea más compleja, sacrifica flexibilidad y ralentiza las actualizaciones.

1. **Objetivo** - mejorar la eficiencia de preguntas frecuentes.
2. **Motivación** - una BD normalizada no implica máxima eficiencia. La descomposición conlleva a usar muchos joins a la hora de hacer consultas en la BD.
3. **Solución** - juntar tablas R1 y R2 en R si la relación resultado tiene pocas actualizaciones (pocas posibilidades de anomalías), muchas consultas y las consultas que implican R1 join R2 son poco eficientes.
4. **De-normalizaciones**:
   - 4.1. Combinación de tablas con asociación 1:1 - para atributos accedidos frecuentemente juntos. Si la asociación es parcial, el número de null es pequeño.

4.2. Duplicidad de atributos que no forman parte de la clave en asociaciones 1:N - donde hay un atributo duplicado que se actualiza raramente, pero aparece frecuentemente en costosos joins.

4.3. Duplicidad de atributos en asociaciones M:N - identico al punto anterior 4.2.

4.4. Atributos multivaluados - donde el número máximo de valores es conocido y pequeño (<=12), y se fija el número máximo de valores. Aparecen muchos null a veces.

Nota final: el autor de este documento no se hace responsable de los errores, variaciones o incompletitudes que puede contener.
Apuntes obtenidos a partir de las diapositivas de Diseño de Bases de Datos facilitadas por Maria Aranzazu Illaramendi Echave
Curso 2020 Facultad de Informática 3er curso
Ingeniería del Software