Diseño de bases de datos relacionales

El paradigma de la orientación a objetos proporciona una metodología adecuada para el diseño de aplicaciones de base de datos debido a que los modelos orientados a objetos son expresivos, concisos y fáciles de implementar.

Una base de datos relacional es una base de datos en la cual los datos se distribuyen lógicamente en tablas de datos y estructuras asociadas.

Un sistema de gestión de bases de datos relacional tiene tres aspectos fundamentales:

- **Los datos se presentan como tablas bidimensionales**. Las tablas cuentan con un número específico de filas y cada intersección fila-columna representa un valor almacenado. Las columnas representan atributos mientras que las filas representan instancias.
- **Operadores para manipular tablas**. El lenguaje estándar para acceder a los datos es el SQL. Los comandos de SQL operan sobre conjuntos y se aplican a la totalidad de la tabla, en contraste a los sistemas orientados a la manipulación de registros y que operan sobre filas individuales.
- **Reglas de integridad de las tablas**. Los RDBMS's más evolucionados soportan *integridad referencial*, que permite asegurar que los valores referenciados en otras tablas realmente existen. También cuentan con la posibilidad de definir restricciones simples.

Diseño de bases de datos relacionales...

Una característica importante de los RDBMS's es que proporcionan soporte a la independencia de datos:

- **Independencia lógica de los datos**. Esto garantiza que una aplicación sólo deba preocuparse de las porciones de la base de datos que le son relevantes. De este modo es posible añadir nuevas aplicaciones a la base de datos sin interferir con las aplicaciones ya existentes.
- **Independencia física de los datos**. Aisla a las aplicaciones de los mecanismos de optimización y ajuste de la base de datos. Mediante este mecanismo es posible reestructurar la base de datos para aumentar su rendimiento sin afectar a la lógica de la aplicación. Esto se resume en la máxima "El desarrollador decide qué datos son necesarios, el RDBMS decide cómo acceder a esos datos físicos".

Para obetener un rendimiento aceptable, es imprescindible:

- Diseñar cuidadosamente la base de datos relacional.
- Si durante el diseño no se construyen los modelos, no es posible considerar en su totalidad todos los aspectos que se van a solicitar de la base de datos.
- La consecuencia es que el rendimiento será impredecible y la base de datos será vulnerable a errores y difícil de programar.

Implementación del modelo de objetos

El primer paso en la implementación de una aplicación consiste en la proyección del modelo de objetos sobre los elementos constructivos de una base de datos relacional.

Tres son las tareas:

- **Definir la identidad**. La identidad se puede implementar basada en la existencia del objeto o basada en su valor.
- **Construir los dominios** añadidos durante el diseño detallado. En el caso de dominios simples, basta con sustituir por los tipos de datos y tamaños adecuados. El caso de los dominios complejos (enumeración, campos multievaluados, etc...) requieren un mayor esfuerzo.
- **Definir las tablas**. Proyectar el modelo de objetos (clases, asociaciones, generalizaciones, etc...) en tablas. Si el modelo de objetos ha sido refinado y optimizado en la fase de diseño esta tarea consiste en unas simples transformaciones.

Definición de la identidad

- Las claves primarias son el único mecanismo disponible en una base de datos relacional para hacer referencia a una tupla concreta.
- Es aconsejable definir una clave primaria para cada tabla.
- Algunas excepciones: por ejemplo, en ocasiones se utilizan tablas temporales para almacenar resultados intermedios durante la ejecución de un programa, estas tablas efímeras pueden no requerir una clave primaria.

Las asociaciones y las generalizaciones se implementan mediante claves foráneas.

- Cada clave foránea debe hacer referencia sólo a la clave primaria de una tabla
- No debe hacer referencia a cualquier otra clave candidata.
- Esta técnica no presenta desventajas y si múltiples ventajas:
  - Múltiples referencias complican innecesariamente el esquema.
  - Es una buena práctica de estilo mantener la consistencia en las estrategias de implementación.
  - Las búsquedas y comparaciones son más difíciles si se deben considerar múltiples rutas de acceso a un registro.

Identidad basada en la existencia

La implementación de la identidad basada en la existencia es relativamente simple:

- En la implementación de la clase, sólo es necesario añadir a cada clase un atributo identificador del objeto y hacer que éste sea la clave primaria.
- En asociaciones,la clave primaria para la tabla de asociación consiste en los identificadores de una o más clases relacionadas.
- Para la implementación de las relaciones de herencia, en el caso ideal, se debe usar el mismo valor del identificador en toda la jerarquía de herencia de objetos.

No es necesario mostrar los identificadores en el modelo de objetos, pero debe incluirse en el esquema de tablas.

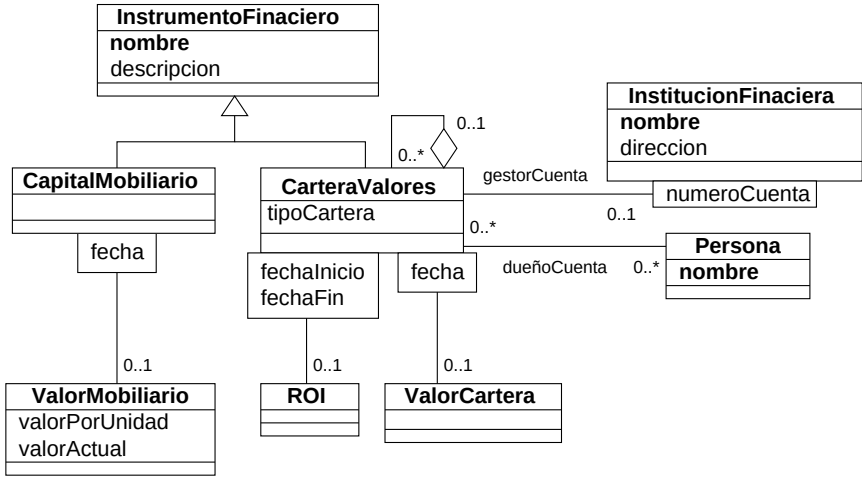
Identidad basada en el valor

Para utilizar identidad basada en el valor, es necesario añadir ciertos elementos notacionales al modelo de objetos para especificar la clave primaria de cada clase.

Se indicará en cada clase si ésta tiene identidad intrínseca o derivada de la identidad de alguna otra clase.

Identidad basada en el valor...

En la figura se muestra la notación de identidad en una parte de una aplicación de gestión de una cartera de valores.



- *InstrumentoFinanciero*, *InstitucionFinanciera* y *Persona* cuentan con la clave primaria *nombre*.
- *CapitalMobiliario* y *CarteraValores* heredan la clave primaria de *InstrumentoFinanciero*.
- La clave primaria de *ValorMobiliario* es la clave primaria de *Capital* combinado con el cualificador *fecha*.
- La clave primaria de *ROI* es la clave primaria de *CarteraValores* y los cualificadores *fechaInicio* y *fechaFin*.
- La clave primaria de *ValorCartera* es la clave primaria de *CarteraValores* combinado con *fecha*.

Identidad basada en el valor...

Se ha utilizado la siguiente notación para indicar el flujo de identidad:

- **Clases independientes**. Los objetos de una clase independiente se distinguen por uno o más atributos intrínsecos a la clase. Los atributos que constituyen la clave primaria se indican en negrita. En la figura anterior las clases *InstrumentoFinanciero*, *InstitucionFinanciera* y *Persona* son clases independientes.
- **Clases dependientes vía asociación**. Los objetos de una clase dependiente derivan su identidad de otros objetos. Un calificador en la clase principal (cuya multiplicidad debe ser siempre 1) indica la identidad de la asociación. En la figura, *ValorMobiliario*, *ROI* y *ValorCartera* son clases dependientes que derivan su identidad a través de una clase fuente vía un cualificador de asociación.
- **Clases dependientes vía generalización**. Normalmente la clave primaria de la superclase se propaga a las subclases. En aquellos casos en que no sea así, se indican en negrita los atributos de la subclase que constituyen la clave primaria.

Construcción de los dominios

- Aunque la mayoría de los RDBMS's carecen de dominios, es necesario utilizarlos en el diseño de bases de datos, ya que constituyen un aspecto importante de la teoría relacional y de la metodología UML.
- El empleo de dominios proporciona un diseño más consistente que el uso directo de los tipos de datos y hace más fácil portar las aplicaciones.
- Es posible definir restricciones en los valores del dominio, por ejemplo, para forzar una enumeración. También es posible utilizar restricciones para definir la multiplicidad.
- Cada atributo de un dominio que está sujeto a una restricción debe contar con la clausula SQL *check* correspondiente.

Identificador del dominio

- Muchos RDBMS's facilitan la identidad basada en la existencia mediante identificadores de dominio: la mayoría de los RDBMS's permiten definir secuencias con nombre

  CREATE SEQUENCE seq_name;

- Cuando se inserta un registro se le asigna el siguiente número de la secuencia.

Dominios enumerados

Un dominio enumerado restringe el valor de un atributo a una lista de valores conocidos.

1. **Cadena de enumeración**. Se almacena la enumeración del atributo como una cadena de caracteres. Si el RDBMS lo permite, es posible imponer la condición de que la enumeración se limite a los valores permitidos. En cualquier otro caso, es necesario forzar la restricción mediante el código de la aplicación.

   Ventajas:

   - El uso de cadenas de caracteres para la enumeración es sencillo.
   - Utiliza un vocabulario controlado, lo que proporciona una gran flexibilidad en las búsquedas.

   Inconvenientes:

   - El impacto en el rendimiento que supone la comprobación de las cadenas puede ser excesivo en el caso de enumeraciones largas.

Dominios enumerados...

2. **Uso de un *flag* por valor de la enumeración**. Consiste en definir un atributo booleano para cada valor de la enumeración. Esta técnica se puede utilizar, por ejemplo, para implementar la enumeración {*rojo,amarillo,azul*} mediante tres valores booleanos.

   Ventajas:

   - Los valores booleanos permiten incorporar el valor en el propio nombre del *flag*, facilitando la legibilidad del código.
   - Permite manejar valores múltiples (por ejemplo, un coche puede ser simultaneamente *rojo* y *azul*).

   Inconvenientes:

   - Es excesivamenta verborréica. Contar con un atributo por valor puede complicar fácilmente el modelo y por extensión la base de datos.
   - No fuerza la exclusividad de los valores, que es frecuentemente un requisito de diseño.

   Recomendación:

   - Los *flags* de enumeración son útiles cuando los valores de la enumeración no son mutuamente excluyentes y se pueden dar múltiples valores simultáneamente.

Dominios enumerados...

3. **Tabla de enumeración**. Almancenar las definiciones de la enumeración en una tabla. La aplicación deberá inspeccionar la tabla de enumeración para encontrar los valores aplicables.

   Ventajas:

   - Puede manejar de forma eficiente enumeraciones grandes.
   - Los valores posibles de la enumeración son visibles en la base de datos. Esto es consistente con la tendencia a almacenar las reglas de negocio en la propia base de datos en lugar de embeberlas en el código de la aplicación.
   - Permite al administrador añadir nuevos valores a la enumeración sin necesidad de modificar el código.

   Inconvenientes:

   - Es incómoda de utilizar en casos triviales. En estos casos es más fácil utilizar una cadena de enumeración y las restricciones que proporciona SQL.
   - Es necesario escribir código genérico especial para leer la tabla de enumeración e imponer los valores.

   Recomendación:

   - Las tablas de enumeración son útiles cuando el número de valores de la enumeración es grande o cuando no se conocen todos los posibles valores durante la fase de desarrollo.

Dominios enumerados...

4. **Codificación de la enumeración**. Consiste en codificar los valores de la enumeración mediante un entero. La aplicación deberá codificar y descodificar automáticamente los valores de la enumeración en las operaciones de lectura y escritura. Es posible utilizar una tabla de enumeración para almacenar ambos valores: la enumeración y el código.

   Ventajas:

   - Ahorra espacio en disco: un entero consume menos espacio que una cadena de caracteres.
   - La aplicación puede mostrar los valores de la enumeración, ya que es sencillo codificar y descodificar la enumeración de forma transparente y con un impacto modesto.
   - Permite imponer los valores de la enumeración mediante el uso de claves foráneas.

   Inconvenientes:

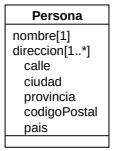
   - Complica el mantenimiento y la depuración del código.
   - No es fácil inspeccionar la base de datos, ya que es necesario consultar la metatabla que contiene la enumeración para descodificar cada valor

   Recomendación:

   - Esta técnica es útil en los mismos casos descritos en el punto anterior.
   - Permite la generación de tablas más compactas y en general mejora el rendimiento.

Dominios estructurados

Un dominio estructurado se puede expandir en dominios más pequeños que son a su vez simples o estructurados.



1. **Cadena concatenada (a)**. Consiste en renunciar a la subestructura y almacenar el dominio como una cadena.
2. **Múltiples columnas (b)**. Se reemplaza el domino estructurado por los elementos constituyentes.
3. **Mediante una tabla adicional (c)**. Se define una tabla para implementar el dominio y se hace uso de una clave foránea para hacer referencia al valor.

Tabla Persona

idPersona	nombre	direccion:String

(a)

Tabla Persona

idPersona	nombre	calle	ciudad	provincia	codigoPostal	pais

(b)

Tabla Persona

idPersona	nombre	**idDireccion** (referencia Direccion)

Tabla Direccion

idDireccion	calle	ciudad	provincia	codigoPostal	pais

(c)

Dominios multievaluados

- Los atributos opcionales (multiplicidad [0..1]) se implementan directamente añadiendo un atributo adicional y permitiendo que tome el valor **NULL**.
- Los RDBMS's no pueden expresar directamente los dominios multievaluados, por lo que es necesario utilizar las técnicas descritas para implementar dominios estructurados (concatenación, múltiples columnas o empleando una tabla adicional).

Implementación de las clases

Cada clase se proyecta en una tabla separada en la que cada atributo es una columna.

También puede ser necesario añadir columnas adicionales para:

- Los identificadores cuando se utiliza identidad basada en la existencia.
- Asociaciones embebidas.
- Discriminadores de generalización que implementan las relaciones de herencia.

Implementación de asociaciones simples

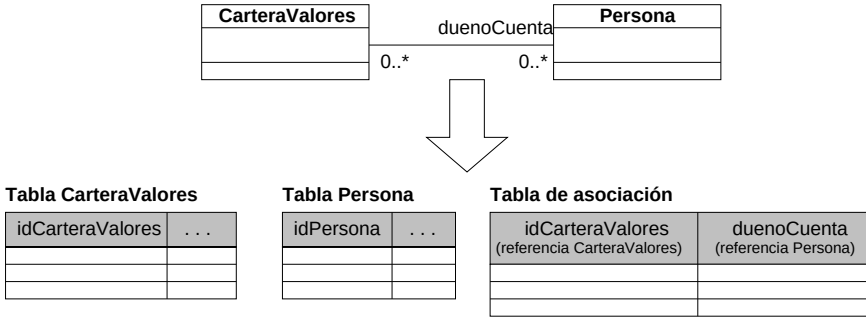
Es posible utilizar varias estrategias para implementar las asociaciones.

Puesto que algunas son mejores que otras, se han clasificado en proyecciones recomendadas, altenativas y desaconsejadas.

- Las proyecciones recomendadas son suficientes para implementar cualquier modelo de objetos y deben ser consideradas en primer lugar.
- En ocasiones y por motivos de rendimiento, extensibilidad o estilo es posible usar una proyección alternativa.
- En ningún caso se deberá utilizar una proyección desaconsejada.

Proyecciones recomendadas

1. **Una tabla propia para las asociaciones muchos-a-muchos**. Cada asociación muchos-a-muchos debe proyectarse en una tabla propia, como se muestra en la figura. La clave primaria de la asociación es la combinación de las claves primarias de cada tabla. Es interesante notar que se ha utilizado el nombre del *rol dueñoCuenta* en la tabla de asociación. Cuando se implementan asociaciones, los nombre de *rol* se deben convertir en el nombre de la clave foránea.



   - El orden de las clases carece de importancia; es posible definir la clave primaria de la asociación de dos formas:
     - *idCarteraValores* + *dueñoCuenta*
     - *dueñoCuenta* + *idCarteraValores*.
   - Por motivos de velocidad en las búsquedas, es necesario definir un índice secundario sobre la clase de menor prioridad en la clave primaria.

     Es decir, si la clave primaria de la asociación es *idCarteraValores* + *dueñoCuenta*, es necesario definir un índice secundario sobre *dueñoCuenta*.

Proyecciones recomendadas...

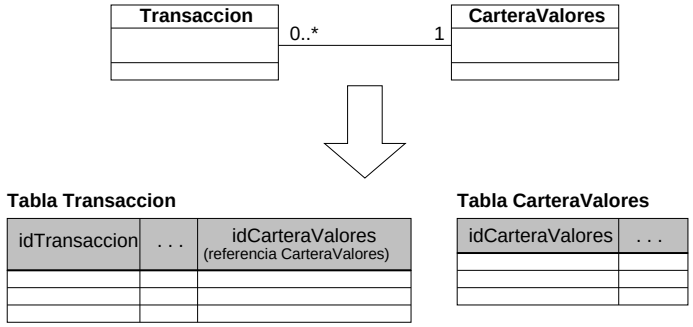
```
CREATE TABLE CarteraValores (
  idCarteraValores NUMBER PRIMARY KEY,
  ...
);

CREATE TABLE Persona (
  idPersona NUMBER PRIMARY KEY,
  ...
);

CREATE TABLE Asociacion (
  idCarteraValores NUMBER,
  duenoCuenta      NUMBER,
  FOREIGN KEY (idCartera) REFERENCES CarteraValores,
  FOREIGN KEY (duenoCuenta) REFERENCES Persona,
  PRIMARY KEY (idCartera,duenoCuenta)
);

CREATE INDEX Asociacion_duenoCuenta_indx
  ON Asociacion (duenoCuenta);
```

- Mediante una tabla de asociación no es posible imponer una multiplicidad mínima de 1. Para este caso es necesario escribir código adicional.
- En el ejemplo anterior este no es el problema ya que la multiplicidad mínima en ambos extremos de la asociación es 0.

Proyecciones recomendadas...

2. **Asociaciones uno-a-muchos embebidas**. Una asociación uno-a-muchos se puede implementar mediante una clave foránea embebida como se muestra en la figura. Para ello, se embebe la clave foránea en la clase que contiene el extremo *muchos* de la asociación.

   Cada transacción debe estar asociada a una cartera de valores: la clave foránea *idCarteraValores* en la tabla *Transaccion* no puede ser **NULL**.




```
CREATE TABLE CarteraValores (
  idCarteraValores NUMBER PRIMARY KEY,
  ...
);

CREATE TABLE Transaccion (
  idTransaccion NUMBER PRIMARY KEY,
  ...
  idCarteraValores NUMBER NOT NULL,
  FOREIGN KEY (idCarteraValores)
    REFERENCES CarteraValores
);
```

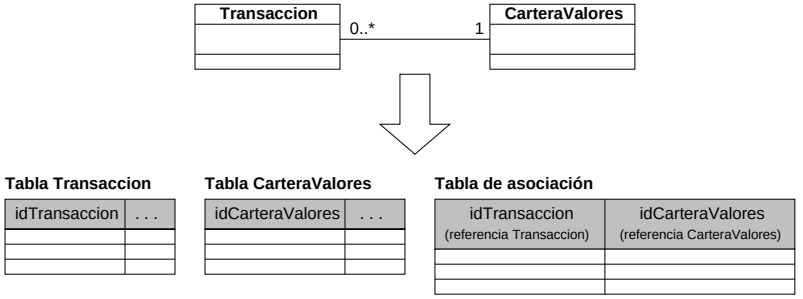
Si la multiplicidad del extremo *CarteraValores* hubiese sido cero-o-uno, la clave foránea *idCarteraValores* debería poder tomar el valor **NULL**.

Proyecciones recomendadas...

3. **Asociaciones cero-o-uno-a-exactamente-uno embebidas**. Este tipo de asociaciones se puede implementar embebiendo la clave foránea en la clase con multiplicidad cero-o-uno. La clave foránea debe definirse con la restricción **NOT NULL**.
4. **Otras asociaciones uno-a-uno embebidas**. Se implementan embebiendo la clave foránea en cualquiera de las dos clases (pero no en ambas). Dependiendo de la mínima multiplicidad en el *rol* elegido será necesario o no definir la restricción **NOT NULL**.

Proyecciones alternativas

1. **Asociaciones en una tabla separada**. La opción recomendada es embeber las asociaciones uno-a-muchos y uno-a-uno, pero también es posible implementarlas mediante una tabla separada:



   - Para asociaciones uno-a-muchos, se toma como clave primaria de la tabla de asociación la clave foránea de la clase con el *rol* muchos.
   - En el caso de asociaciones uno-a-uno, se puede definir como clave primaria para la tabla de asociación cualquiera de las dos claves.

```
CREATE TABLE CarteraValores (
  idCarteraValores NUMBER PRIMARY KEY,
  ...
);
CREATE TABLE Transaccion (
  idTransaccion NUMBER PRIMARY KEY,
  ...
);
CREATE TABLE Asociacion (
  idTransaccion NUMBER PRIMARY KEY,
  idCarteraValores NUMBER NOT NULL,
  FOREIGN KEY (idTransaccion)
    REFERENCES Transaccion,
  FOREIGN KEY (idCarteraValores)
    REFERENCES CarteraValores
);
```

Proyecciones alternativas...

Ventajas:

- Implementar las asociaciones mediante tablas separadas mejora la uniformidad del diseño y proporciona una implementación más extensible.
- Si posteriormente se modifica la multiplicidad de la asociación sólo es necesario modificar las restricciones, pero no la estructura de la tabla.

Inconvenientes:

- Fragmenta la base de datos.
- Complica la "navegación": es necesario recurrir a *joins* sobre las tablas adicionales. Esto complica el código SQL y se reduce el rendimiento.
- Mediante esta técnica no se puede imponer la multiplicidad exactamente-uno, por lo que es necesario escribir código adicional para comprobar la restricción.

Proyecciones desaconsejadas

1. **Combinación**. En la figura se combinan dos clases y la asociación entre éstas en una tabla simple. La combinación de clases presenta desventajas sustanciales:
   - La tabla resultante puede violar la tercera forma normal dependiendo del modelo.
   - Es contrario a la filosofía orientada a objetos combinar dos o más clases en una clase única. Las clases son unidades atómicas dentro del modelo orientado a objetos. La implementación es más comprensible y extensible si se sigue esta semántica.

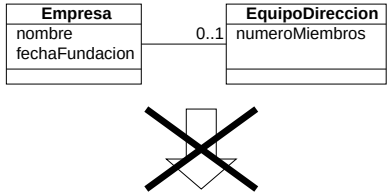


Tabla combinada

nombre	fechaFundacion	numeroMiembros

2. **Asociaciones uno-a-uno doblemente embebidas**. Es posible implementar una asociación de modo redundante embebiendo dos claves foráneas (cada una en la clase relacionada). Esta proyección está desacoonsejada por varios motivos:
   - En la mayoría de las consultas que implican a ambas tablas es necesario recurrir a un *join* en cualquier caso.
   - Los RDBMS's no pueden mantener la consistencia de las asociaciones redundantes. Como consecuencia, es necesario escribir código adicional para mantener la integridad de la asociación.

Implementación de asociaciones avanzadas

Proyecciones recomendadas

1. **Atributos de enlace**. Como regla general, siempre se utilizan tablas separadas para implementar asociaciones con atributos de enlace.
2. **Clases de asociación**. La norma general es implementar las clases de asociación mediante tablas separadas. La identidad de las instancias de una clase de asociación deriva de la identidad de las clases relacionadas. En la figura la clave primaria de *Autorizacion* es la combinación de las claves primarias de las clases *Tabla* y *Usuario*.

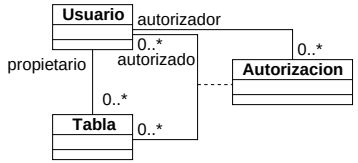


Tabla Usuario

idUsuario	. . .

Tabla Tabla

idTabla	. . .	propietario (referencia Usuario)

Tabla Autorizacion

autorizado (referencia Usuario)	idTabla (referencia Tabla)	autorizador (referencia Usuario)

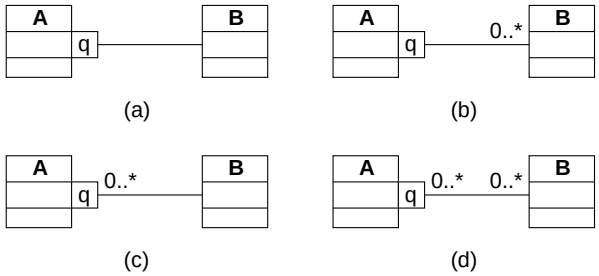
Proyecciones recomendadas...

```
CREATE TABLE Usuario (
  idUsuario NUMBER PRIMARY KEY,
  ...
);
CREATE TABLE Tabla (
  idTabla NUMBER PRIMARY KEY,
  ...
  propietario NUMBER NOT NULL,
  FOREIGN KEY (propietario) REFERENCES Usuario
);
CREATE TABLE Autorizacion (
  autorizado NUMBER NOT NULL,
  idTabla NUMBER NOT NULL,
  autorizador NUMBER NOT NULL,
  PRIMARY KEY (autorizado,idTabla),
  FOREIGN KEY (autorizado) REFERENCES Usuario,
  FOREIGN KEY (idTabla) REFERENCES Tabla,
  FOREIGN KEY (autorizador) REFERENCES Usuario
);
CREATE INDEX Autorizacion_idTabla_indx
  ON Autorizacion (idTabla);
```

2. **Asociaciones ternarias**. Como en los dos casos anteriores las asociaciones ternarias se implementan mediante una tabla separada.

Proyecciones recomendadas...

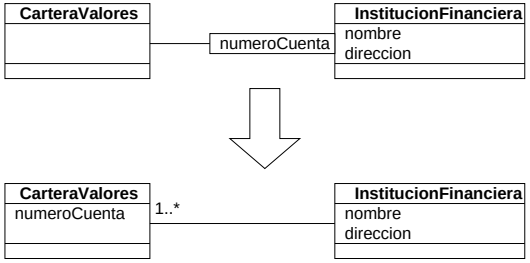
3. **Asociaciones cualificadas**. Las asociaciones cualificadas siguen el mismo esquema de proyección que las asociaciones simples correspondientes sin cualificador. La recomendación es:
   - Las asociaciones cualificadas representadas en la figuras (a) y (b) se implementan embebiendo el cualificador en la clase B.
   - Los casos (c) y (d) requieren una tabla separada ya que la asociación subyacente sin cualificador es muchos-a-muchos. La clave primaria de la asociación (d) es la combinación de las claves primarias de A, B y el atributo (cualificador) q.



Proyecciones recomendadas...

3. ...En la figura se muestra un ejemplo de la implementación de una asociación cualificada.

   El esquema de proyección es:
   - El primer paso consiste en transformar la asociación cualificada a la forma correspondiente sin cualificar.
   - La transfomación pone de manifiesto que la asociación subyacente es uno-a-muchos: una institución financiera dispone de muchas cuentas, pero una cuenta determinada sólo pertenece a una institución financiera.
   - El último paso consiste en implementar la asociación como se ha explicado.



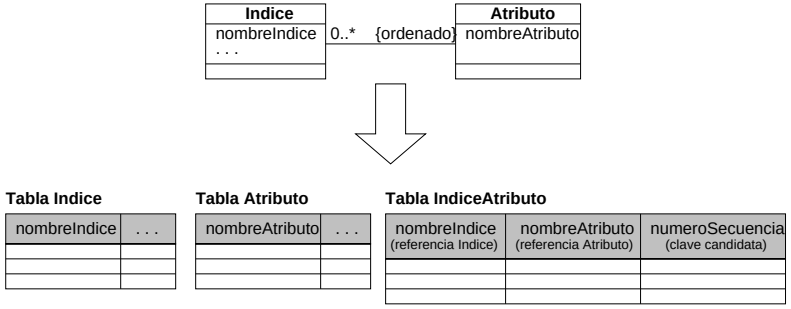
Proyecciones recomendadas...

4. **Asociaciones ordenadas**. Las asociaciones ordenadas se pueden implementar mediante un atributo adicional que introduce un número de secuencia.

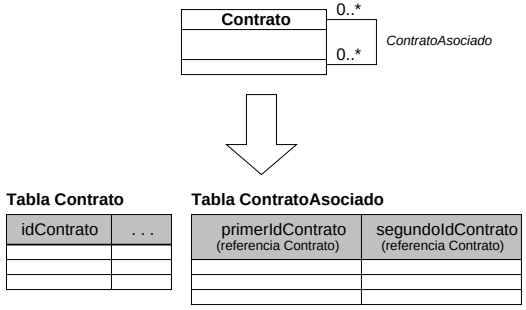
   La tabla *IndiceAtributo* cuenta con dos claves candidatas:

   - *nombreIndice* + *nombreAtributo*
   - *nombreIndice* + *numeroSecuencia*.

   Se ha escogido arbitrariamente la clave primaria *nombreIndice* + *nombreAtributo*.



5. **Asociaciones simétricas**. Una asociación simétrica es una asociación entre objetos de la misma clase que presentan *roles* intercambiables.



Proyecciones recomendadas...

5. ...

```
CREATE TABLE Contrato (
  idContrato NUMBER PRIMARY KEY,
  ...
);

CREATE TABLE ContratoAsociado (
  primerIdContrato  NUMBER,
  segundoIdContrato NUMBER,
  PRIMARY KEY (primerIdContrato,segundoIdContrato),
  FOREIGN KEY (primerIdContrato) REFERENCES Contrato,
  FOREIGN KEY (segundoIdContrato) REFERENCES Contrato
);
CREATE INDEX ContratoAsociado_segundoIdContrato_indx
  ON ContratoAsociado (segundoIdContrato);
```

Las asociaciones simétricas son poco frecuentes, y suelen dar problemas de implementación. En el caso de la tabla *ContratoAsociado*, se pueden adoptar dos aproximaciones:

- Para no romper la simetría, se almacena cada registro dos veces. Si el contrato A está asociado al contrato B, sería necesario almacenar (A,B) y (B,A).
- Si se rompe la simetría, para encontrar todos los contratos relacionados con A es necesario buscar primero todas las tuplas con $primerIdContrato = A$ y después todas las tuplas con $segundoIdContrato = A$.

En la medida de lo posible, es recomendable evitar las relaciones simétricas.

6. **Agregación**. La agregación es un tipo de asociación, por lo que se implementan como éstas.

Proyecciones alternativas

1. **Atributos de enlace**. Las asociaciones uno-a-uno y uno-a-muchos con atributos de enlace también se pueden implementar empleando una clave foránea embebida, pero será necesario embeber también el atributo de enlace junto con la clave foránea.

   Ventajas:

   - La clave foránea embebidas permite reducir el número de tablas que es necesario utilizar en una consulta.
   - Este esquema de implementación permite imponer la multiplicidad mínima 1 en el esquema de la base de datos.

   Inconvenientes:

   - Embeber los atributos de enlace viola la segunda forma normal en el caso de asociaciones uno-a-muchos y la tercera forma normal en el caso de asociaciones uno-a-uno.

   Recomendación:

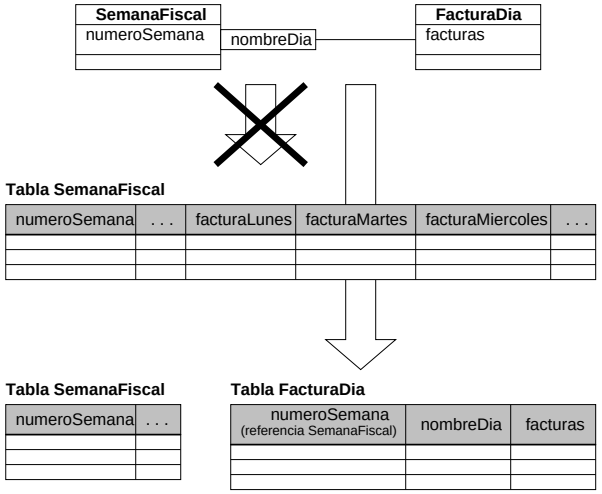
   - La violación de las formas normales comentadas en el punto anterior hace que sea recomendable implementar las asociaciones con atributos de enlace mediante una tabla separada.

2. **Asociaciones cualificadas**. Es posible utilizar una tabla separada para implementar una asociación cualificada con multiplicidad uno-a-muchos. Esto puede ser útil para imponer la unicidad del extremo 1. En el ejemplo de implementación de una asociación cualificada, se podría haber creando una tabla con los campos *idCarteraValores*, *idInstitucionFinanciera* y *numeroCuenta*. La clave primaria de esa tabla debe ser *idInstitucionFinanciera* + *idCarteraValores*.

Proyecciones desaconsejadas

1. **Asociaciones cualificadas**. No es aconsejable implementar el papel "muchos" de una asociación cualificada embebiendo atributos en paralelo. Los atributos paralelos aparecen frecuentemente cuando el cualificador es un atributo enumerado.

   En la figura, es preferible separar las tablas *SemanaFiscal* y *FacturaDia* mediante una tabla con un atributo para cada día de la semana.



   - La única ventaja del uso de atributos paralelos es que el número de tablas es menor.
   - Las búsquedas puede ser muy compleja. Por ejemplo: determinar cual es el día de la semana que presenta el mayor número de facturas.
   - La implementación es menos extensible: si se añade un nuevo atributo a la enumeración es necesario añadir una nueva columna a la tabla y modificar todo el código asociado.
   - En contraste, no es necesario modificar la tabla *FacturaDia* cuando se introduce o elimina un valor de la enumeración.

Implementación de la herencia simple

Proyección recomendada

- La generalización se implementa generalmente empleado tablas separadas para la superclase y las subclases (figura).
- En el caso ideal, un objeto debe tener la misma clave primaria a lo largo de toda la jerarquía de herencia. El discriminador (*tipoInstrumentoFinanciero*) indica la subclase apropiada que implementa la totalidad del objeto registrado en la superclase.
- Para implementar sucesivos niveles de generalización se aplica el procedimiento de proyección a un nivel cada vez.

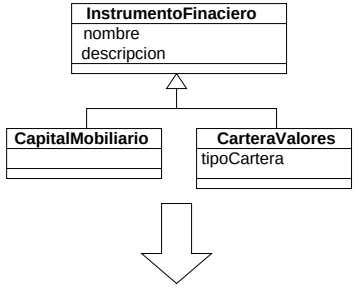


Tabla InstrumentoFinanciero

idInstrumentoFinanciero	nombre	descripcion	tipoInstrumentoFinanciero

Tabla CapitalMobiliario

idCapitalMobiliario (referencia InstrumentoFinanciero)

Tabla CarteraValores

idCarteraValores (referencia InstrumentoFinanciero)	tipoCartera

Proyección recomendada...

La integridad referencial puede asegurar que cada registro de la subclase corresponde a algún registro de la superclase. Sin embargo, ningún RDBMS puede asegurar:

- Que cada registro de la superclase corresponde a un registro en alguna de las subclases.
- Que cada registro de la superclase puede ser descrito en su totalidad por la tabla que indica el discriminador.

Por tanto, es necesario suplir estas deficiencias mediante el código de la aplicación.

Adicionalmente, es posible definir una vista para cada subclase para consolidar los datos heredados y hacer más fácil el acceso al objeto.

```
CREATE VIEW view_CarteraValores AS
  SELECT c.idCarteraValores, i.nombre,
         i.descripcion, c.tipoCartera
    FROM InstrumentoFinanciero i, CarteraValores c
   WHERE i.idInstrumentoFinanciero = c.idCarteraValores;
```

Proyecciones alternativas

1. **Eliminación**. Es posible eliminar aquellas subclases que no cuentan con más atributos que la clave primaria. En la figura, el discriminador *tipoCapitalMobiliario* indica si el objeto *CapitalMobiliario* pertenece a la clase *MetalPrecioso* o *ValorMobiliario*. La clase *MetalPrecioso* carece de atributos adicionales, asociaciones o discriminadores, por lo que puede eliminarse la tabla.

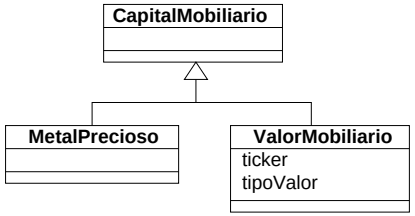


Tabla CapitalMobiliario

idCapitalMobiliario	tipoCapitalMobiliario

Tabla ValorMobiliario

idValorMobiliario (referencia CapitalMobiliario)	tiker	tipoValor

   - La ventaja de esta aproximación es que la base de datos cuenta con menos tablas.
   - La mayor desventaja es que la implementación se hace de forma irregular.
   - Además, la implementación es menos extensible, ya que sería necesario añadir la tabla si se modifica la subclase eliminada.

Proyecciones alternativas...

2. **Mover los atributos de la superclase a las subclases**. También es posible implementar la generalización eliminando la tabla de la superclase y replicando sus atributos en cada subclase.

   Los objetos se describen en una única tabla en lugar de distribuir su valor entre tablas para cada nivel de la jerarquía de herencia.

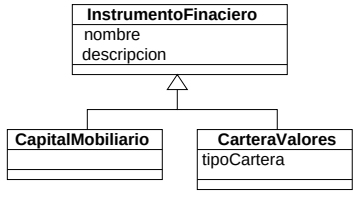


Tabla CapitalMobiliario

idCapitalMobiliario	nombre	descripcion

Tabla CarteraValores

idCarteraValores	nombre	descripcion	tipoCartera

Ventajas:

- El beneficio más claro es que cada objeto es descrito en su propia tabla, eliminando la necesidad de recurrir a *joins* para reconstruir el objeto.

Inconvenientes:

- Aunque esta proyección satisface las formas normales, introduce redundancia.
- Es necesario buscar en múltiples tablas para encontrar un objeto.
- La implementación pierde la estructura descriptiva que es intrínseca a la generalización.

Proyecciones alternativas...

3. **Mover los atributos de las subclases a la superclase**. Otra posibilidad es embeber los atributos de las subclases en la superclase y eliminar las tablas asociadas a las subclases.

   Aquellos atributos que no son aplicables a un objeto dado deben tener asignado el valor **NULL**.

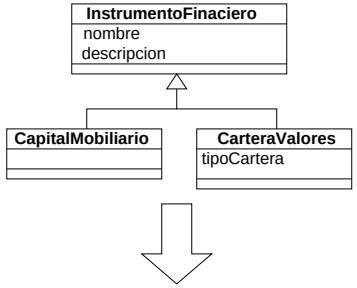


Tabla InstrumentoFinanciero

idInstrumentoFinanciero	nombre	descripcion	tipoInstrumentoFinanciero	tipoCartera

Ventajas:

- Permite describir cada objeto utilizando sólo una tabla.

Inconvenientes:

- Viola la segunda forma normal ya que algunos atributos no dependen completamente de la clave primaria.
- La implementación pierde, como en el punto anterior, la estructura descriptiva del problema.

Proyecciones alternativas...

4. **Proyección híbrida**. Existe una proyección híbrida que es útil en ocasiones. Consiste en embeber los atributos de la superclase en las subclases y mantener una tabla para la superclase para facilitar la "navegación" de las subclases.

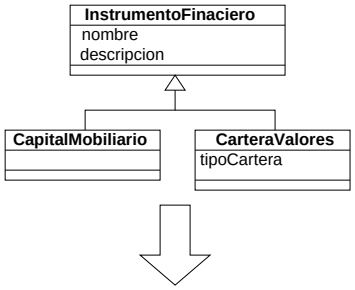


Tabla InstrumentoFinanciero

idInstrumentoFinanciero	tipoInstrumentoFinanciero

Tabla CapitalMobiliario

idCapitalMobiliario (referencia InstrumentoFinanciero)	nombre	descripcion

Tabla CarteraValores

idCarteraValores (referencia InstrumentoFinanciero)	nombre	descripcion	tipoCartera

Ventajas:

- Cada objeto es descrito en su propia tabla, eliminando la necesidad de recurrir a *joins* para reconstruir el objeto.
- Sólo es necesario buscar en una tabla para encontrar un objeto.

Inconvenientes:

- Introduce redundancia en el esquema.
- La implementación pierde la estructura descriptiva que es intrínseca a la generalización.

Sumario de las reglas de proyección del modelo de objetos

Concepto	Modelo de objetos	Proyección RDBMS
Dominio	Dominio simple	Utilizar un dominio del RDBMS
	Identificador	Emplear las características específicas de los RDBMSs.
	Enumeracion	Almacenar el atributo de enumeración como una cadena o en una tabla separada.
	Estructurado	Utilizar cadenas concatenadas, columnas múltiples o tablas adicionales.
	Multievaluado	
Clase	Clase	Proyectar cada clase en una tabla.
Asociación	Muchos-a-muchos	Utilizar una tabla separada.
	Uno-a-muchos	Embeber la clave foránea.
	Uno-a-uno	
	Atributo de enlace	Emplear una tabla separada.
	Clase de asociación	
	Asociación ternaria	
	Asociación cualificada	Seguir las recomendaciones para las asociaciones sin cualificar.
	Asociación ordenada	Introducir atributos con números de secuencia y seguir las recomendaciones para asociaciones cualificadas.
	Asociación simétrica	Intentar romper la simetría.
	Agregación	Seguir las recomendaciones para las asociaciones.
Generalización	Herencia simple	Usar tablas separadas para la superclase y las subclases.
	Herencia múltiple	