CAPÍTULO 5

Normalización

CONTENIDO

Objetivos del capítulo

En este capítulo aprenderá lo siguiente:

- Por qué las relaciones se deben normalizar
- El significado de dependencia funcional y su relación con las claves
- Cómo se pueden usar las reglas de inferencia para dependencias funcionales
- La definición de la primera forma normal y cómo lograrla
- El significado de dependencia funcional completa
- La definición de la segunda forma normal y cómo lograrla
- El significado de la dependencia transitiva
- Definición de la tercera forma normal y cómo lograrla
- Definición de la forma normal Boyce-Codd y cómo lograrla
- El significado de las dependencias multivaluadas
- Definición de la cuarta forma normal y cómo lograrla
- El significado de dependencia de combinación
- Definición de la quinta forma normal (forma normal proyección-combinación)

- Definición de la forma normal dominio-clave
- Cuándo detener el proceso de normalización

5.1 Objetivos de la normalización

El objetivo básico del modelado lógico es desarrollar una "buena" descripción de los datos, sus relaciones y sus restricciones. Para el modelo relacional, esto significa que debe identificar un conjunto adecuado de relaciones. Sin embargo, la tarea de elegir las relaciones es difícil, porque existen muchas opciones para que el diseñador las considere. Este capítulo explica algunos métodos para mejorar el diseño lógico. Las técnicas que aquí se presentan se basan en un gran conjunto de investigación en el proceso de diseño lógico generalmente llamado normalización.

El propósito de la normalización es producir un conjunto estable de relaciones que sea un modelo fiel de las operaciones de la empresa. Al seguir los principios de la normalización, se logra un diseño que es muy flexible, lo que permite al modelo extenderse cuando necesite representar nuevos atributos, conjuntos de entidades y relaciones. La base de datos se diseña en tal forma que se pueden fortalecer con facilidad ciertos tipos de restricciones de integridad. También se puede reducir la redundancia en la base de datos, tanto para ahorrar espacio como para evitar inconsistencias en los datos. También asegura que el diseño esté libre de ciertas anomalías de actualización, inserciones y borrado. Una anomalía es un estado inconsistente, incompleto o contradictorio de la base de datos. Si estas anomalías estuvieran presentes sería incapaz de representar cierta información, podría perder información cuando ciertas actualizaciones se realicen y correría el riesgo de que los datos se vuelvan inconsistentes con el tiempo.

5.2 Anomalías de inserción, actualización y borrado

Considere la siguiente relación

```
NewClass(classNo, stuId, stuLastName, facId, schedule, room, grade)
```

Una instancia de esta relación aparece en la figura 5.1. En este ejemplo se supondrá que solamente existe un miembro del personal docente para cada clase (esto es: no hay enseñanza en equipo). También se supone que cada clase siempre tiene asignado el mismo salón. Esta relación muestra anomalías de actualización, inserción y borrado.

- **Anomalía de actualización.** Suponga que quiere cambiar el horario de ART103A a MWF12. Es posible que pueda actualizar los dos primeros registros de la tabla `NewClass`, pero no el tercero, lo que resulta en un estado inconsistente en la base de datos. Entonces sería imposible decir el verdadero horario para dicha clase. Ésta es una anomalía de actualización.

courseNo	stuId	stuLastName	facId	schedule	room	grade
ART103A	S1001	Smith	F101	MWF9	H221	A
ART103A	S1010	Burns	F101	MWF9	H221	
ART103A	S1006	Lee	F101	MWF9	H221	B
CSC201A	S1003	Jones	F105	TUTHF10	M110	A
CSC201A	S1006	Lee	F105	TUTHF10	M110	G
HST205A	S1001	Smith	F202	MWF11	H221	

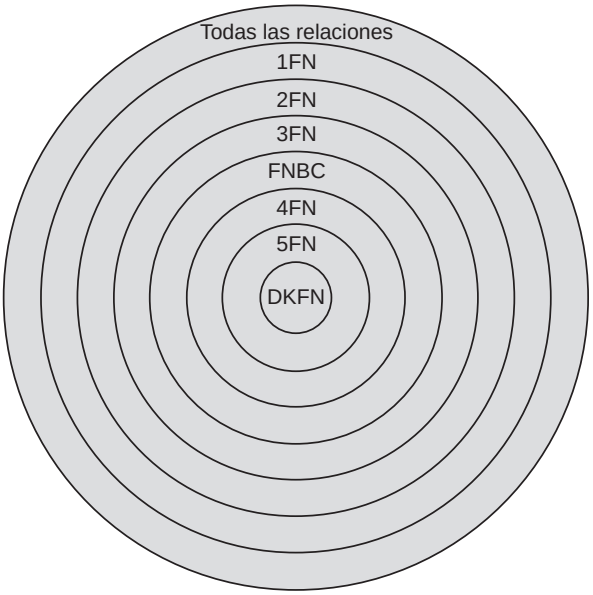
FIGURA 5.1
La tabla NewClass

- **Anomalía de inserción.** Una anomalía de inserción ocurre cuando intenta agregar información acerca de un curso para el cual todavía no hay estudiantes registrados. Por ejemplo, suponga que crea una nueva clase, con valores MTH110A, F110, MTuTh10, H225 para `classNumber`, `facId`, `schedule` y `room`. No es posible registrar la información del curso, aun cuando tenga los valores para estos atributos. Dado que la clave es `{courseNo,stuId}`, no tiene permiso para insertar un registro con un valor nulo para `stuId`. Puesto que no tiene posibilidad de representar esta información de clase, tiene una anomalía de inserción. El mismo problema ocurriría si intenta insertar información acerca de un estudiante que todavía no esté registrado en curso alguno.
- **Anomalía de borrado.** Cuando borra el registro del único estudiante que toma un curso particular, ocurre una anomalía de borrado. Por ejemplo, si el estudiante S1001 se retira de HST205A, perdería toda la información acerca de dicho curso. Sería deseable conservar la información del curso, pero no puede hacerlo sin un `stuId` correspondiente. De igual modo, si un estudiante abandona el único curso que toma, se pierde toda la información acerca de dicho estudiante.

La investigación acerca de estas anomalías la realizó por primera ocasión Codd, quien identificó las causas y definió las primeras tres "formas normales". Una relación es una forma normal específica si satisface el conjunto de requisitos o restricciones para dicha forma. Note que las restricciones que se discuten son restricciones de esquema, propiedades permanentes de la relación, no simplemente de alguna instancia de la relación. Son propiedades de la intensión, no sólo de una extensión particular. Investigación posterior de Boyce y Codd condujo a un refinamiento de la tercera de estas formas. Investigación adicional de Fagin, Zaniolo y Delobel (cada uno de manera independiente) resultó en la definición de tres nuevas formas normales. Todas las formas normales son anidadas (nested), en cuanto a que satisfacen las restricciones de las anteriores, pero es una "mejor" forma porque cada una elimina los fallos que se encuentran en la forma previa. La figura 5.2 muestra cómo se relacionan las siete formas normales. El círculo más grande representa todas las relaciones. Entre el conjunto de todas las relaciones, aquellas que satisfacen ciertas condiciones están en la primera forma normal y se representan mediante el segundo círculo más grande. De las que están en la primera forma normal, existen algunas que tienen condiciones adicionales y que también están en la segunda forma normal; éstas caen en el siguiente círculo, y así por el estilo. El objetivo del diseño debe ser poner el esquema en la forma normal más alta, que es práctica y adecuada para los datos en la base de datos. Normalización significa poner una relación en una forma normal superior. La normalización requiere tener un uso claro de la semántica del modelo. Examinar simplemente una instancia o extensión no es suficiente, porque una instancia no proporciona suficiente información acerca de todos los posibles valores o combinaciones de valores para atributos de relación.

Al intentar se puntualicen las causas de las anomalías de actualización, inserción y borrado, los investigadores identificaron tres tipos de dependencias: **dependencias funcionales**,

FIGURA 5.2
Relaciones de las formas normales



dependencias multivaluadas y **dependencias de combinación**. En la literatura de investigación aparecen dependencias adicionales.

5.3 Dependencia funcional

Una **dependencia funcional (DF)** es un tipo de relación entre atributos, como se describe mediante la siguiente definición:

> **Definición:** Si R es un esquema de relación y A y B son conjuntos de atributos no vacíos en R, se dice que B es **funcionalmente dependiente** en A si y sólo si cada valor de A en R tiene asociado exactamente un valor de B en R.

Esto se escribe como

$$A \rightarrow B$$

que se lee como "A determina funcionalmente a B". La definición dice que si dos tuplas en una extensión de R tienen el mismo valor para A, también deben tener los mismos valores para B. De manera más formal, para cada par de tuplas, t_1 y t_2, en cada instancia de R, se tienen las siguientes reglas:

Si $t_1.A = t_2.A$, entonces $t_1.B = t_2.B$

Aquí, la notación $t_1.A$ significa la proyección de la tupla t_1 sobre los atributos del conjunto A. Esta regla no significa que A **causa a** B o que el valor de B se puede calcular a partir del valor de A mediante una fórmula, aunque a veces éste es el caso. Simplemente significa que, si se conoce el valor de A y se examina la tabla de relación R, se encontrará sólo un valor de B en todas las filas que tienen el valor dado de A en cualquier momento dado. Por tanto, cuando dos filas tienen el mismo valor A, también deben tener el mismo valor B. Sin embargo, para un valor B dado, existen varios valores diferentes de A. Una dependencia funcional es en realidad una **relación muchos a uno** del conjunto de atributos A al conjunto de atributos B. Es una **restricción de integridad** que toda instancia de la base de datos debe obedecer. Note que A y B pueden ser conjuntos que consisten de un solo atributo. Cuando existe una dependencia funcional, el conjunto de atributos en el lado izquierdo de la flecha (A en este ejemplo) se llama **determinante** y el conjunto de atributos en el lado derecho de la flecha se llama **dependiente**.

NewStudent

stuId	LastName	major	credits	status	socSecNo
S1001	Smith	History	90	Senior	100429500
S1003	Jones	Math	95	Senior	010124567
S1006	Lee	CSC	15	Freshman	088520876
S1010	Burns	Art	63	Junior	099320985
S1060	Jones	CSC	25	Freshman	064624738

FIGURA 5.3

Instancia de la tabla NewStudent (suponga que cada estudiante tiene sólo una especialidad)

Para ilustrar la dependencia funcional considere la siguiente relación:

```
NewStudent(stuId, lastName, major, credits, status, socSecNo)
```

La figura 5.3 muestra una instancia de la relación. Esta relación almacena información acerca de los estudiantes en un colegio. Aquí, se supondrá que cada estudiante tiene ID y número de seguridad social únicos, y que cada estudiante tiene cuando mucho una especialidad. Se supondrá que los apellidos no son únicos, que dos estudiantes diferentes pueden tener el mismo apellido. El atributo `credits` significa el número de créditos completados y `status` se refiere al año en el que está el estudiante: freshman (1°), sophomore (2°), junior (3°) y senior (4°). Aunque esta instancia se usa para ayudar a detectar las dependencias funcionales, se intentará determinar características permanentes de la relación, no simplemente de la instancia. Al examinar la tabla, se ve que, si se proporciona un valor específico de `stuId`, sólo hay un valor de `lastName` asociado con dicho `stuId` particular. Por ejemplo, para el `stuId` S1006, el `lastName` asociado es Lee. Puesto que se entiende que cada estudiante tiene una ID única, se sabe que es una característica de la relación, no sólo de esta instancia, de modo que `{lastName}` es funcionalmente dependiente de `{stuId}` y se puede escribir

```
{stuId} → {lastName}
```

Sin embargo, para un valor dado de `lastName`, puede haber más de un `stuId`. Note que, para el `lastName` Jones, existen dos valores `stuId`, S1003 y S1060. Por tanto, no se puede voltear la dependencia funcional y escribir `{lastName}` → `{stuId}`. Para cada uno de los otros atributos, existe sólo un valor asociado con un valor particular de `stuId`, de modo que todos los atributos son funcionalmente dependientes de `stuId`. Se tiene

```
{stuId}→ {stuId}
{stuId}→ {lastName}
{stuId}→ { major}
{stuId}→ {credits}
{stuId}→ {status}
{stuId}→ {socSecNo}
```

que en forma abreviada se escribirá como

```
{stuId }→ {stuId, lastName, major, credits, status, socSecNo}
```

para indicar que cada uno de los atributos a la derecha es funcionalmente dependiente de `stuId`. De igual modo se tiene

```
{socSecNo} → {socSecNo, stuId, lastName, major, credits, status}
```

Note también que `status` es funcionalmente dependiente de `credits`. Por ejemplo, si se sabe que el valor de `credits` es 15, el `status` de manera automática es freshman. Se escribe

```
{credits} → {status}
```

Para conjuntos que consisten en un solo atributo como éste, se escribirá justo el nombre del atributo y se eliminarán las llaves. Para este caso de dependencia funcional, el determinante, `credits`, **no** determina de manera funcional todos los otros atributos de la relación. Note también que el valor de `credits` no necesariamente es único. Muchos estudiantes podrían tener el mismo número de créditos, de modo que la exclusividad no es una característica necesaria de los determinantes. La instancia en la figura 5.3 en realidad no demuestra la falta de exclusividad de `credits`, lo que muestra que debe tener cuidado al hacer juicios sólo a partir de las instancias. Es necesario concentrarse en los significados de los atributos y sus restricciones al identificar dependencias funcionales. Note que no se tiene la DF `status` → `credits`, pues, por ejemplo, dos alumnos de primer año pueden tener un número diferente de créditos, como se ve en los registros de S1006 y S1060.

Ciertas dependencias funcionales se llaman **triviales** porque siempre se satisfacen en cada relación. En las dependencias funcionales triviales, el dependiente es un subconjunto del determinante. Si todos los atributos en el conjunto del lado derecho están incluidos en el conjunto del lado izquierdo de la dependencia, o si los dos lados son el mismo, la DF es trivial. Por ejemplo, las siguientes DF son triviales:

```
{A,B}→ A
{A,B}→ B
{A,B}→ {A,B}
```

5.4 Superclaves, claves candidatas y claves primarias

Recuerde del capítulo 3 que una **superclave** es un atributo o conjunto de atributos que identifica de manera única una entidad. En una tabla, una superclave es cualquier columna o conjunto de columnas cuyos valores se pueden usar para distinguir una fila de otra. En consecuencia, una superclave es cualquier identificador único. Dado que una superclave identifica de manera única a cada entidad, determina funcionalmente a todos los atributos de una relación. Para la tabla `Student` de la figura 5.3, `{stuId}` es una superclave. Al igual que la combinación de `{stuId,lastName}`. De hecho, `{stuId, cualquier otro atributo}` es una superclave para esta relación. Lo mismo es cierto para `{socSecNo, cualquier otro atributo}`. De hecho, cualquier conjunto de atributos que contengan una superclave también es una superclave. Sin embargo, una superclave puede contener atributos adicionales que no son necesarios para exclusividad, y el interés está en encontrar superclaves que no contengan tales atributos adicionales.

Una **clave candidata** es una superclave tal que ningún subconjunto propio de sus atributos sea por sí mismo una superclave. Por tanto, una clave candidata debe ser un identificador **mínimo**. En el ejemplo, la superclave `{stuId, lastName}` no es una clave candidata porque contiene un subconjunto propio, `{stuId}`, que es una superclave. Sin embargo, `stuId` por sí mismo es una clave candidata. De igual modo, `socSecNo` es una clave candidata. Si a ningún par de estudiantes se le permite tener la misma combinación de valores para nombre y especialidad, la combinación `{lastName,major}` también sería una clave candidata. Por tanto, se ve que una relación puede tener varias claves candidatas. Se usará el término clave compuesta para referir una clave que consista en más de un atributo.

Una **clave primaria** es una clave candidata que en realidad se usa para identificar tuplas en una relación. En el ejemplo, `stuId` puede ser la clave primaria y `socSecNo` una clave alterna. Estos dos atributos se determinan de modo funcional uno a otro, y todos los atributos son funcionalmente dependientes de cada uno de ellos. Cuando se toma una decisión acerca de cuál clave candidata usar como la clave primaria, es importante considerar cuál elección es una mejor representación del mundo real en el que funciona la empresa. Se elige `stuId` en lugar de `socSecNo` porque, dado que la universidad asigna valores `stuId`,

siempre puede estar seguro de tener dicho valor para cada estudiante. Es posible que no se tenga el número de seguridad social de cada estudiante. Por ejemplo, los estudiantes extranjeros pueden no tener números de seguridad social. También existen reglas de privacidad que limitan el uso de los números de seguridad social como identificadores, de modo que es mejor evitar usarlos para la base de datos universidad. Aunque no se establece de manera explícita en la definición, una importante característica de una clave primaria es que ninguno de sus atributos puede tener valores nulos. Si en las claves se permiten valores nulos, sería incapaz de separar los registros, dado que dos registros con valores nulos en el mismo campo de clave pueden ser indistinguibles. Para claves candidatas se especificará que sus valores también son únicos en la base de datos. Esto ayuda a asegurar la calidad de los datos pues se sabe que los valores duplicados serían incorrectos para estos ítems. También es deseable reforzar la regla de "no nulos" para claves candidatas, siempre que pueda asegurar la disponibilidad del valor de la clave candidata.

5.5 El proceso de normalización usando claves primarias

En la práctica, los diseñadores de bases de datos por lo general desarrollan un modelo lógico inicial para una base de datos relacional al mapear un diagrama E-R u otro modelo conceptual a un conjunto de relaciones. Identifican la clave primaria y posiblemente dependencias funcionales durante el proceso de diseño conceptual. Al mapear el diagrama E-R a un modelo relacional, con el método descrito en la sección 4.8, el diseñador crea un modelo relacional que ya está bastante bien normalizado. Para completar el proceso de diseño relacional, el diseñador comprueba cada relación, identifica un conjunto de dependencias funcionales por si existe alguna otra que involucre las claves primarias y las normaliza más si es necesario. Un conjunto de reglas empíricas bien desarrolladas, o heurística, se usa para guiar el proceso de normalización con base en claves primarias. En la siguiente sección se describe este proceso, usando definiciones informales de las formas normales. En secciones posteriores se formalizarán estas nociones.

5.5.1 Primera forma normal

Para describir la primera forma normal se usará un contraejemplo. Si supone que a un estudiante se le permite tener más de una especialidad, y se intenta almacenar especialidades múltiples en el mismo campo del registro del estudiante, la tabla `NewStu` puede parecerse a la de la figura 5.4(a). Este ejemplo viola la definición de la primera forma normal, que es la siguiente.

> **Definición:** Una relación está en la **primera forma normal** (1FN) si y sólo si cada atributo tiene valor sencillo para cada tupla.

Esto significa que cada atributo en cada fila, o cada "celda" de la tabla, contiene sólo un valor. Una forma alternativa de describir la primera forma normal es decir que los dominios de los atributos de la relación son **atómicos**. Esto significa que en el dominio no se permiten conjuntos, listas, campos repetidos o grupos. Los valores en el dominio deben ser valores únicos que no se puedan descomponer más. En la tabla de la figura 5.4(a) se ve la violación de esta regla en los registros de los estudiantes S1006 y S1010, quienes ahora tienen dos valores mencionados para `major`. Éste es el primer ejemplo visto, porque todas las relaciones consideradas hasta ahora estaban en la primera forma normal. De hecho, la mayor parte de la teoría relacional se basa en relaciones en al menos primera forma normal, aunque en la literatura aparecen relaciones que no están en la primera forma normal. Es importante tener primera forma normal de modo que los operadores relacionales, como se les definió, funcionen de manera correcta. Por ejemplo, si realiza la operación del álgebra

relacional SELECT NewStu WHERE major = 'Math' en la tabla de la figura 5.4(a), ¿se debe incluir el registro del estudiante S1006? Si fuera a realizar una combinación natural con alguna otra tabla usando `major` como la columna de combinación, ¿este registro se aparearía sólo con aquellos que tengan tanto CSC como Math como `major`, o con aquellos que tengan o CSC o Math? ¿Qué ocurriría con la combinación si otro registro tuviera una doble especialidad citada como Math CSC? Para evitar estas ambigüedades se insistirá en que toda relación se escriba en la primera forma normal. Si una relación no está ya en 1FN se le puede rescribir y crear una nueva relación que consista en la clave de la relación original, más el atributo multivaluado. Por tanto, la tabla `NewStu` se rescribirá como en la figura 5.4(b) y se creará una nueva tabla `Majors` (especialidades). Note también que la clave de la nueva tabla no es `stuId`, pues cualquier estudiante con más de una especialidad aparece al menos dos veces. Se necesita `{stuId, major}` para identificar de manera única un registro en dicha tabla.

Otro método de normalización a primera forma normal en el caso donde se conoce el número máximo de repeticiones que puede tener un atributo es agregar nuevas columnas

FIGURA 5.4

Ejemplos de la primera forma normal

FIGURA 5.4(a)

Tabla NewStu (suponga que los estudiantes pueden tener doble especialidad)

StuId	lastName	major	credits	status	socSecNo
S1001	Smith	History	90	Senior	100429500
S1003	Jones	Math	95	Senior	010124567
S1006	Lee	CSC Math	15	Freshman	088520876
S1010	Burns	Art English	63	Junior	099320985
S1060	Jones	CSC	25	Freshman	064624738

FIGURA 5.4(b)

Tablas NewStu2 y Majors

NewStu2

stuId	lastName	credits	status	socSecNo
S1001	Smith	90	Senior	100429500
S1003	Jones	95	Senior	010124567
S1006	Lee	15	Freshman	088520876
S1010	Burns	63	Junior	099320985
S1060	Jones	25	Freshman	064624738

Majors

stuId	major
S1001	History
S1003	Math
S1006	CSC
S1006	Math
S1010	Art
S1010	English
S1060	CSC

stuId	lastName	major1	major2	credits	status	socSecNo
S1001	Smith	History		90	Senior	100429500
S1003	Jones	Math		95	Senior	010124567
S1006	Lee	CSC	Math	15	Freshman	088520876
S1010	Burns	Art	English	63	Junior	099320985
S1060	Jones	CSC		25	Freshman	064624738

FIGURA 5.4(c)
Tabla newStu3 con dos atributos para major

stuId	lastName	major	credits	status	socSecNo
S1001	Smith	History	90	Senior	100429500
S1003	Jones	Math	95	Senior	010124567
S1006	Lee	CSC	15	Freshman	088520876
S1006	Lee	Math	15	Freshman	088520876
S1010	Burns	Art	63	Junior	099320985
S1010	Burns	English	63	Junior	099320985
S1060	Jones	CSC	25	Freshman	064624738

FIGURA 5.4(d)
Tabla NewStu2 rescrita en 1FN, con {stuId,major} como clave primaria

para el atributo. Por ejemplo, si sabe que los estudiantes pueden tener cuando mucho dos especialidades, `NewStu` se rescribiría con dos columnas para la especialidad, `major1` y `major2`, como se muestra en la figura 5.4(c). La desventaja de este enfoque es que usted debe conocer el máximo número de repeticiones, y que las consultas se vuelven más complejas. Para formar una selección en álgebra relacional sobre la especialidad, necesitaría probar tanto la columna `major1` como la `major2`.

Un método alternativo para convertir en 1FN la tabla original es hacer al atributo multivaluado parte de la clave. Con este método, la nueva tabla contendría filas múltiples para estudiantes con múltiples especialidades, como se muestra en la figura 5.4(d). Esta solución puede hacer que surjan dificultades cuando el diseñador intente poner la relación en formas normales superiores.

5.5.2 Dependencia funcional completa y segunda forma normal

Para la relación que se muestra en las figuras 5.1 y 5.5(a), se tienen las siguientes dependencias funcionales además de las triviales:

```
{courseNo,stuId} → {lastName}
{courseNo,stuId} → {facId}
{courseNo,stuId} → {schedule}
{courseNo,stuId} → {room}
{courseNo,stuId} → {grade}
```

Dado que no hay otra clave candidata, se elige `{courseNo,stuId}` para la clave primaria. De nuevo, al ignorar las dependencias funcionales triviales, también se tienen las dependencias funcionales

```
courseNo → facId
courseNo → schedule
courseNo → room
stuId → lastName
```

De modo que se encuentran atributos que son funcionalmente dependientes en la combinación `{courseNo,stuId}`, pero también funcionalmente dependientes en un subconjunto de dicha combinación. Se dice que tales atributos no son **por completo** dependientes funcionales de la combinación.

> **Definición:** En una relación R, el atributo A de R es **completamente dependiente funcional** sobre un atributo o conjunto de atributos X de R si A es funcionalmente dependiente sobre X pero no funcionalmente dependiente sobre cualquier subconjunto propio de X.

En el ejemplo, aunque `lastName` es funcionalmente dependiente sobre `{courseNo, stuId}`, también es funcionalmente dependiente sobre un subconjunto propio de dicha combinación, a saber `stuId`. De igual modo, `facId`, `schedule` y `room` son funcionalmente dependientes sobre el subconjunto propio `courseNo`. Note que `grade` es por completo dependiente funcional sobre la combinación `{courseNo,stuId}`.

> **Definición:** Una relación está en **segunda forma normal** (2FN) si y sólo si está en primera forma normal y todos los atributos no clave son completamente dependientes funcionales sobre la clave.

Claro está, si una relación es 1FN y la clave consiste en un solo atributo, la relación es automáticamente 2FN. Tiene que preocuparse por 2FN sólo cuando la clave sea compuesta. A partir de la definición, se ve que la relación `Class` no está en segunda forma pues, por ejemplo, `lastName` no es completamente dependiente funcional sobre la clave `{courseNo, stuId}`. A pesar de que aquí existen otras dependencias no completamente funcionales, una es suficiente para mostrar que la relación no es 2FN.

Una relación 1FN que no es 2FN se puede transformar en un conjunto equivalente de relaciones 2FN. La transformación se efectúa al realizar **proyecciones** sobre la relación original en tal forma que es posible regresar al original al tomar la combinación de las proyecciones.

FIGURA 5.5

Ejemplo de segunda forma normal

FIGURA 5.5(a)

La tabla NewClass no en 2FN

classNo	stuId	stuLastName	facId	schedule	room	grade
ART103A	S1001	Smith	F101	MWF9	H221	A
ART103A	S1010	Burns	F101	MWF9	H221	
ART103A	S1006	Lee	F101	MWF9	H221	B
CSC201A	S1003	Jones	F105	TUTHF10	M110	A
CSC201A	S1006	Lee	F105	TUTHF10	M110	C
HST205A	S1001	Smith	F202	MWF11	H221	

Register

classNo	stuI	grade
ART103A	S1001	A
ART103A	S1010	
ART103A	S1006	B
CSC201A	S1003	A
CSC201A	S1006	C

Stu

stuId	stuLastName
S1001	Smith
S1010	Burns
S1006	Lee
S1003	Jones

Class2

classNo	facId	schedule	room
ART103A	F101	MWF9	H221
CSC201A	F105	TUTHF10	M110
HST205A	F202	MWF11	H221

FIGURA 5.5(b)

Las tablas Register, Stu y Class2 en 2FN

Las proyecciones de este tipo se llaman **proyecciones sin pérdida** (lossless) y se discutirán con más detalle en la sección 5.6.3. En esencia, para hacer una relación 2FN, identifique cada una de las dependencias no completamente funcionales y forme proyecciones al remover los atributos que dependan de cada uno de los determinantes identificados como tales. Estos determinantes se colocan en relaciones separadas junto con sus atributos dependientes. La relación original todavía contendrá la clave compuesta y cualesquier atributos que sean completamente dependientes funcionales sobre ella. Incluso si no existen atributos completamente dependientes funcionales sobre la clave de la relación original, es importante conservar la relación (incluso sólo con los atributos clave) con la finalidad de poder reconstruir la relación original mediante una combinación. Esta "relación de conexión" muestra cómo se relacionan las proyecciones.

Al aplicar este método al ejemplo primero identifique todas las dependencias funcionales de interés. Por brevedad, se combinarán los atributos que aparezcan como dependientes y tengan el mismo determinante, y se eliminarán las llaves en ambos lados. Es importante notar que, cuando se usa esta notación menos formal, los atributos a la derecha de la flecha se pueden "descomponer" y citar como DF separadas, pero los atributos en el lado izquierdo deben permanecer unidos, pues es su combinación la que es determinante. Las dependencias funcionales son

```
courseNo → facId, schedule, room
stuId → lastName
courseNo,stuId → grade (y, desde luego, facId,schedule,room,lastName)
```

Al usar proyección, se descompone la relación `NewClass` en el siguiente conjunto de relaciones:

```
Register (courseNo, stuId, grade)
Class2 (courseNo, facId, schedule, room)
Stu (stuId, stuLastName)
```

Las relaciones resultantes se muestran en la figura 5.5(b). Note que podría reconstruir la relación original al tomar la combinación natural de estas tres relaciones. Incluso si no hubiera atributo `grade`, necesitaría la relación `Register(courseNo,stuId)` con la finalidad de mostrar cuáles estudiantes están inscritos en cuáles cursos. Sin ella, no podría combinar `Class2` y `Stu`, que no tienen atributos comunes. Al usar estas nuevas relaciones se eliminan las anomalías de actualización, inserción y borrado discutidas anteriormente. Puede cambiar el horario del curso de ART103A al actualizar la columna `schedule` en un solo registro en `Class2`. Puede agregar nueva información de curso a `Class2` sin tener algún estudiante inscrito en el curso, y puede insertar nueva información de estudiante al agregar un registro a `Stu`, sin tener que insertar información de curso para dicho estudiante. De igual modo, puede eliminar el registro de matriculación de `Register` y aun así conservar la información acerca del curso en `Class2` y acerca del estudiante en `Stu`.

5.5.3 Dependencia transitiva y tercera forma normal

Aunque las relaciones de la segunda forma normal son mejores que las de la primera forma normal, todavía pueden tener anomalías de actualización, inserción y borrado. Considere la siguiente relación:

```
NewStudent (stuId, lastName, major, credits, status)
```

La figura 5.6(a) muestra una instancia de esta relación. Aquí, la única clave candidata es `stuId` y se usará como la clave primaria. Todo otro atributo de la relación es funcionalmente dependiente de la clave, así que se tiene la siguiente dependencia funcional, entre otras,

```
stuId → credits
```

Sin embargo, dado que el número de créditos determina el `status`, como se discutió en la sección 5.3, también se tiene

$$\texttt{credits} \rightarrow \texttt{status}$$

Por tanto, `stuId` de manera funcional determina `status` en dos formas, directa y transitivamente, a través del atributo no clave `status`. Al usar transitividad se tiene

$$(\texttt{stuId} \rightarrow \texttt{credits}) \wedge (\texttt{credits} \rightarrow \texttt{status}) \Rightarrow (\texttt{stuId} \rightarrow \texttt{status})$$

> **Definición:** Si A, B y C son atributos de la relación R, tales que A → B y B → C, entonces C es **transitivamente dependiente** de A.

Para la tercera forma normal se quiere eliminar ciertas dependencias transitivas. Las dependencias transitivas causan anomalías de inserción, borrado y actualización. Por ejemplo, en la tabla `NewStudent` de la figura 5.6(a), no se puede insertar la información de que cualquier estudiante con 30 créditos tenga estatus Soph hasta que se tenga a tal estudiante, puesto que eso requeriría insertar un registro sin un `stuId`, lo que no está permitido. Si se borra el registro del único estudiante con cierto número de créditos, se pierde la información acerca del estatus asociado con dichos créditos. Si tiene varios registros con el mismo valor `credits` y cambia el estatus asociado con dicho valor (por ejemplo, hacer que 24 créditos ahora tenga el estatus de Soph), de manera accidental puede fallar al actualizar todos los registros, lo que deja la base de datos en un estado inconsistente. Debido a estos problemas, es deseable remover las dependencias transitivas y crear un conjunto de relaciones que satisfagan la siguiente definición.

> **Definición:** Una relación está en **tercera forma normal** (3FN) si, siempre que exista una dependencia funcional no trivial X → A, entonces o X es una superclave o A es un miembro de alguna clave candidata.

Parafraseando a Kent (vea las referencias), usted puede recordar las características de la tercera forma normal al decir que cada atributo no clave debe depender de la clave, toda la clave y nada más que la clave.

FIGURA 5.6
Ejemplo de tercera forma normal

FIGURA 5.6(a)
Tabla NewStudent no en 3FN

NewStudent

stuId	lastName	major	credits	status
S1001	Smith	History	90	Senior
S1003	Jones	Math	95	Senior
S1006	Lee	CSC	15	Freshman
S1010	Burns	Art	63	Junior
S1060	Jones	CSC	25	Freshman

FIGURA 5.6(b)
Tablas NewStu2 y Stats en 3FN

NewStu2

stuId	lastName	major	credits
S1001	Smith	History	90
S1003	Jones	Math	95
S1006	Lee	CSC	15
S1010	Burns	Art	63
S1060	Jones	CSC	25

Stats

credits	status
15	Freshman
25	Freshman
63	Junior
90	Senior
95	Senior

Al comprobar la tercera forma normal, se busca si algún atributo no clave candidata (o grupo de atributos) es funcionalmente dependiente de otro atributo no clave (o grupo). Si existe tal dependencia funcional, se remueve de la relación el atributo funcionalmente dependiente, y se le coloca en una nueva relación con su determinante. El determinante puede permanecer en la relación original. Para el ejemplo `NewStudent`, dado que la dependencia indeseable es `credits` → `status`, y `status` no es parte de alguna clave candidata, se forma el conjunto de relaciones:

```
NewStu2 (stuId, lastName, major, credits)
Stats (credits, status)
```

Esta descomposición se muestra en la figura 5.6(b). De hecho, se puede decidir no almacenar `status` en la base de datos, y calcular el `status` para aquellas vistas que la necesiten. En este caso, simplemente se elimina la relación `Status`. Este ejemplo no involucra múltiples claves candidatas. Si en la relación original se tiene una segunda clave candidata, `socialSecurityNumber`, se tendría

```
socialSecurityNumber → status
```

pero esto es permisible dado que `socialSecurityNumber` es una superclave para la relación, de modo que se tendría el número de seguridad social en la relación `NewStu2`, junto con `stuId` y los otros atributos ahí.

La definición de la tercera forma normal es la original desarrollada por Codd. Es suficiente para relaciones que tengan una sola clave candidata, pero se descubrió que es deficiente cuando existen múltiples claves candidatas que son compuestas o se traslapan. Por tanto, para considerar todos los casos se formuló una definición mejorada de la tercera forma normal, que lleva el nombre de sus desarrolladores, Boyce y Codd.

5.5.4 Forma normal de Boyce-Codd

La forma normal Boyce-Codd es ligeramente más estricta que 3FN.

> **Definición:** Una relación está en forma Boyce/Codd (FNBC) si, siempre que existe una dependencia funcional no trivial X → A, entonces X es una superclave.

Por tanto, para comprobar la existencia de FNBC, simplemente identifique todos los determinantes y verifique que son superclaves.

Al regresar a los primeros ejemplos para comprobar directamente FNBC se ve que, para la relación `NewStudent` que se muestra en la figura 5.6(a), los determinantes son `stuId` y `credits`. Dado que `credits` no es una superclave, esta relación no es FNBC. Al realizar las proyecciones como se hizo en la sección anterior, las relaciones resultantes son FNBC. Para la relación `NewClass` que se muestra en la figura 5.5(a) se encuentran los determinantes `courseNo`, que (por sí mismo) no es una superclave, y `stuId`, tampoco una superclave. Por tanto, la relación `Class` no es FNBC. Sin embargo, las relaciones que resultan de las proyecciones son FNBC. Considere un ejemplo en el que se tenga 3FN mas no FNBC.

```
NewFac (facName, dept, office, rank, dateHired)
```

Para este ejemplo, que se muestra en la figura 5.7(a), se supondrá que, aunque los nombres del personal docente no son únicos, dos miembros del personal docente dentro de un solo departamento no tienen el mismo nombre. También se supone que cada miembro del personal docente sólo tiene una oficina, identificada en office. Un departamento puede tener varias oficinas de personal docente, y los docentes del mismo departamento pueden compartir oficinas.

FIGURA 5.7

Ejemplo de forma normal Boyce-Codd

FIGURA 5.7(a)

Tabla NewFac en 3FN, mas no FNBC

Faculty				
facName	**dept**	**office**	**rank**	**dateHired**
Adams	Art	A101	Professor	1975
Byrne	Math	M201	Assistant	2000
Davis	Art	A101	Associate	1992
Gordon	Math	M201	Professor	1982
Hughes	Math	M203	Associate	1990
Smith	CSC	C101	Professor	1980
Smith	History	H102	Associate	1990
Tanaka	CSC	C101	Instructor	2001
Vaughn	CSC	C105	Associate	1995

FIGURA 5.7(b)

Fac1 y Fac2 en FNBC

Fac1	
office	**dept**
A101	Art
C101	CSC
C105	CSC
H102	History
M201	Math
M203	Math

Fac2			
facName	**office**	**rank**	**dateHired**
Adams	A101	Professor	1975
Byrne	M201	Assistant	2000
Davis	A101	Associate	1992
Gordon	M201	Professor	1982
Hughes	M203	Associate	1990
Smith	C101	Professor	1980
Smith	H102	Associate	1990
Tanaka	C101	Instructor	2001
Vaughn	C105	Associate	1995

A partir de estas suposiciones se tienen las siguientes DF. De nuevo, se eliminan las llaves y se combinan los dependientes con el mismo determinante.

```
office → dept
facName,dept → office, rank, dateHired
facName,office → dept, rank, dateHired
```

Se tienen claves candidatas que se traslapan de `{facName,dept}` y `{facNam,office}`. Si elige `{facName,dept}` como clave primaria, queda el determinante, `office`, que no es una superclave. Esto viola la FNBC. Note que la relación es 3FN pues `office` es parte de una clave candidata. Para llegar a FNBC puede descomponer la relación `Faculty` mediante proyección en

```
Fac1 (dept, office)
Fac2 (facName, office, rank, dateHired)
```

La clave de la primera relación es `office`, pues un departamento puede tener varias oficinas, mas cada oficina pertenece sólo a un departamento. Claramente es FNBC, pues el único determinante es la clave. La clave de la segunda es `{facName,office}`. También

es FNBC, dado que su único determinante es la clave. Sin embargo, note que el esquema final no preserva la dependencia funcional `facName,dept` → `office,rank, dateHired (fecha de contratación)`, dado que estos atributos no permanecen en la misma relación.

(Nota: Si se hubiese elegido `facName,office` como la clave primaria de la relación `NewFac` original, se tendría `office` → `dept`. Dado que `office` no es una superclave, la relación no sería FNBC. De hecho, no sería 2FN, pues `dept` no sería completamente dependiente funcional sobre la clave, `facName,office`.)

Cualquier relación que no es FNBC se puede descomponer en relaciones FNBC mediante el método recién ilustrado. No obstante, no siempre puede ser deseable transformar la relación en FNBC. En particular, si existe dependencia funcional que no se preserva cuando se realiza la descomposición, entonces se vuelve difícil fortalecer la dependencia funcional en la base de datos pues dos o más tablas tendrían que unirse para verificar que se refuerza, y se pierde una importante restricción. En este caso es preferible asentar 3FN, que siempre permite preservar dependencias. La relación `NewFac` proporciona un ejemplo en el que se pierde una dependencia funcional al normalizar a FNBC. En las relaciones resultantes, los atributos `facName` y `dept` aparecieron en diferentes relaciones, y no se tenía forma de expresar el hecho de que ellas determinaban todos los otros atributos.

5.5.5 Ejemplo comprensivo de dependencias funcionales

Para resumir las varias formas normales definidas mediante dependencias funcionales, considere la siguiente relación que almacena información acerca de proyectos en una gran empresa:

```
Work (projName, projMgr, empId, hours, empName, budget, startDate, salary, empMgr, empDept,
rating)
```

La figura 5.8(a) muestra una instancia de esta relación.

Se hacen las siguientes suposiciones:

1. Cada proyecto tiene un nombre único.
2. Aunque los nombres de proyecto son únicos, los nombres de los empleados y supervisores no lo son.
3. Cada proyecto tiene un supervisor, cuyo nombre se almacena en `projMgr`.
4. Muchos empleados se pueden asignar para trabajar en cada proyecto, y un empleado se puede asignar a más de un proyecto. El atributo `hours` dice el número de horas por semana que un empleado particular se asigna para trabajar en un proyecto particular.
5. `budget` almacena la cantidad presupuestada para un proyecto, y `startDate` da la fecha de inicio para un proyecto.
6. `salary` proporciona el salario anual de un empleado.
7. `empMgr` proporciona el nombre del supervisor del empleado, que puede no ser el mismo que el supervisor del proyecto.
8. `empDept` proporciona el departamento del empleado. Los nombres de departamento son únicos. El supervisor del empleado es el supervisor del departamento del empleado.
9. `rating` proporciona la calificación del empleado para un proyecto particular. El supervisor del proyecto asigna la calificación al final del trabajo del empleado en dicho proyecto.

FIGURA 5.8

Ejemplo de normalización

projName	projMgr	empId	hours	empName	budget	startDate	salary	empMgr	empDept	rating
Jupiter	Smith	E101	25	Jones	100000	01/15/04	60000	Levine	10	9
Jupiter	Smith	E105	40	Adams	100000	01/15/04	55000	Jones	12	
Jupiter	Smith	E110	10	Rivera	100000	01/15/04	43000	Levine	10	8
Maxima	Lee	E101	15	Jones	200000	03/01/04	60000	Levine	10	
Maxima	Lee	E110	30	Rivera	200000	03/01/04	43000	Levine	10	
Maxima	Lee	E120	15	Tanaka	200000	03/01/04	45000	Jones	15	

FIGURA 5.8(a)

La tabla Work

Proj

projName	projMgr	budget	startDate
Jupiter	Smith	100000	01/15/04
Maxima	Lee	200000	03/01/04

Dept

empDept	empMgr
10	Levine
12	Jones
15	Jones

Emp1

EmpId	empName	salary	empDept
E101	Jones	60000	10
E105	Adams	55000	12
E110	Rivera	43000	10
E120	Tanaka	45000	15

Work1

projName	empId	hours	rating
Jupiter	Jones	25	9
Jupiter	Adams	40	
Jupiter	Rivera	10	8
Maxima	Jones	15	
Maxima	Rivera	30	
Maxima	Tanaka	15	

FIGURA 5.8(b)

Las tablas normalizadas que sustituyen Work

Al usar estas suposiciones se encuentran las siguientes dependencias funcionales para comenzar:

```
projName → projMgr, budget, startDate
empId → empName, salary, empMgr, empDept
projName, empId → hours, rating
```

Dado que se supone que los nombres de las personas no son únicos, `empMgr` no determina funcionalmente `empDept`. [Dos supervisores diferentes pueden tener el mismo nombre y supervisar distintos departamentos, o posiblemente un supervisor puede supervisar varios departamentos, vea a Jones en la figura 5.8(a).] De igual modo, `projMgr` no determina `projName`. Sin embargo, dado que los nombres de departamento son únicos y cada departamento sólo tiene un supervisor, es necesario agregar

```
empDept → empMgr
```

Usted puede preguntarse si `projMgr` → `budget`. Aunque puede ser el caso de que el supervisor determina el presupuesto en el sentido de la palabra, lo cual significa que el supervisor presenta las cifras para el presupuesto, debe recordar que la dependencia funcional no significa causar o calcular. De igual modo, aunque el supervisor del proyecto asigna una calificación al empleado, no hay dependencia funcional entre `projMgr` y `rating`. (Si la hubiera, significaría que cada supervisor siempre da las mismas calificaciones a quienes evalúa. Por ejemplo, significaría que si el nombre del supervisor es Levine, el empleado siempre obtiene una calificación de 9.)

Como se aprecia que cada atributo es funcionalmente dependiente de la combinación `projName`, `empId`, se elegirá dicha combinación como la clave primaria, y vea qué forma normal tiene. Comience por verificar si ya está en FNBC.

FNBC: Observe si existe un determinante que no sea una superclave. Cualesquiera de `empId`, `empDept` o `projName` es suficiente para mostrar que la relación `Work` no es FNBC. Puesto que se sabe que no es FNBC, comience el proceso de normalización por comprobar las formas normales inferiores, y normalice las relaciones conforme avance.

Primera forma normal: Con la clave compuesta, cada celda sería de un solo valor, de modo que `Work` está en 1FN.

Segunda forma normal: Se encuentran dependencias parciales (no completas).

```
projName → projMgr, budget, startDate
empId → empName, salary, empMgr, empDept
```

Puede hacerse cargo de esto, al transformar la relación en un conjunto equivalente de relaciones 2FN mediante proyección, lo que resulta en

```
Proj (projName, projMgr, budget, startDate)
Emp (empId, empName, salary, empMgr, empDept)
Work1 (projName, empId, hours, rating)
```

Tercera forma normal: Con el conjunto de proyecciones, `{Proj,Emp,Work1}`, se prueba cada relación para ver si se tiene 3FN. Al examinar `Proj` se ve que ningún atributo no clave determina funcionalmente otro atributo no clave, así que `Proj` es 3FN. En `Emp` se tiene una dependencia transitiva, pues `empDept` → `empMgr`, como se explicó anteriormente. Como `empDept` no es una superclave, ni `empMgr` es parte de una clave candidata, esto viola 3FN. Por tanto, es necesario rescribir `Emp` como

```
Emp1 (empId, empName, salary, empDept)
Dept (empDept, empMgr)
```

`Work1` no tiene dependencia transitiva que involucre horas o calificación, de modo que la relación ya es 3FN. En consecuencia, el nuevo conjunto de relaciones 3FN es

```
Proj (projName, projMgr, budget, startDate)
Emp1 (empId, empName, salary, empDept)
Dept (empDept, empMgr)
Work1 (projName, empId, hours, rating)
```

Forma normal Boyce-Codd revisada: El nuevo conjunto de relaciones 3FN también es FNBC, pues, en cada relación, el único determinante es la clave primaria.

La figura 5.8(b) muestra las nuevas tablas que sustituyen la tabla `Work` original. Note que se pueden unir para producir exactamente la tabla original.

5.6 Propiedades de las descomposiciones relacionales

Aunque la normalización se puede realizar a partir del uso del enfoque heurístico basado en claves primarias y que se demostró en las secciones anteriores, un enfoque más formal al diseño de bases de datos relacionales se basa estrictamente en dependencias funcionales y otros tipos de restricciones. Este enfoque usa algoritmos de normalización formal para crear esquemas de relación. Para comenzar, todos los atributos en la base de datos se colocan en una sola relación grande llamada **relación universal**. Al usar dependencias funcionales y otras restricciones, la relación universal se descompone en esquemas relacionales más pequeños hasta que el proceso alcanza un punto donde ya no se prefiere más descomposición. Se quiere que los resultados del proceso de descomposición tengan algunas cualidades importantes, si es posible. Es deseable tener cada esquema de relación en FNBC o al menos 3FN. Otras propiedades importantes incluyen preservación de atributo, preservación de dependencia y combinaciones sin pérdida.

5.6.1 Preservación de atributo

Cuando la relación universal se construye contiene, por definición, todo atributo en la base de datos. En el proceso de descomponer la relación universal en relaciones más pequeñas y mover atributos hacia ellas se quiere garantizar que cada atributo aparezca en al menos una de las relaciones, de modo que no se pierden ítem de datos. Como se vio en los ejemplos, el esquema de base de datos usualmente contiene alguna repetición de atributos con la finalidad de representar relaciones entre las tablas. Sin embargo, tienen que colocarse en relaciones en una forma que preserve toda la información, no sólo todos los atributos.

5.6.2 Preservación de dependencia

Una dependencia funcional representa una restricción que se debe reforzar en la base de datos. Siempre que se realice una actualización, el DBMS debe comprobar que no se viola la restricción. Es mucho más fácil verificar las restricciones dentro de una tabla que verificar una que involucre múltiples tablas, lo que requeriría realizar primero una combinación. Para evitar que deban hacerse tales combinaciones se quiere asegurar que, en una descomposición, las dependencias funcionales involucren atributos que estén todos en la misma tabla, si es posible. Dada una descomposición de una relación R, con un conjunto de dependencias funcionales en ella, en un conjunto de relaciones individuales $\{R_1, R_2, \ldots, R_n\}$, para cada dependencia funcional $X \rightarrow Y$ es deseable que todos los atributos en $X \cup Y$ aparezcan en la misma relación, R_i. Esta propiedad se llama **preservación de dependencia**. Siempre es posible encontrar una dependencia que preserve la descomposición que sea 3FN, pero no siempre es posible encontrar una que sea FNBC, como se ilustra en el ejemplo de la figura 5.7 y que se discutió en la sección 5.5.4.

5.6.3 Descomposición sin pérdida

Al dividir las relaciones mediante proyección en los ejemplos anteriores se fue muy explícito acerca del método de descomposición a usar. En particular, se tuvo cuidado al usar proyecciones que pudieran deshacerse al combinar las tablas resultantes, de manera que resultaría la tabla original. Por el término tabla original no se entiende simplemente la estructura de la tabla, esto es, los nombres de columna, sino también las tuplas reales. Tal descomposición se llama **descomposición sin pérdida**, porque conserva toda la información en la relación original. Aunque se usó la palabra descomposición, no se escribió una definición formal, y ahora se hace.

> **Definición:** Una **descomposición** de una relación R es un conjunto de relaciones $\{R_1, R_2, \ldots, R_n\}$ tal que cada R_i es un subconjunto de R y la unión de todas las R_i es R.

Ahora está listo para la definición de descomposición sin pérdida.

> **Definición:** Una descomposición $\{R_1, R_2, \ldots, R_n\}$ de una relación R se llama **descomposición sin pérdida** de R si la combinación natural de $R_1, R_2, \ldots, R_n$ produce exactamente la relación R.

No todas las descomposiciones son sin pérdida, porque existen proyecciones cuya combinación no da de vuelta la relación original. Como ejemplo de una proyección con pérdida, considere la relación `EmpRoleProj (empName,role,projName)` que se muestra en la figura 5.9(a). La tabla muestra cuáles empleados juegan qué papel para cuál proyecto. Se puede descomponer la tabla mediante proyección en las dos tablas, `Table 1` y `Table 2` que se muestran en la figura 5.9(b). Sin embargo, cuando se combinan estas dos tablas, en la figura 5.9(c), se obtiene una tupla adicional que no aparece en la tabla original. Ésta es una tupla **espuria** (falsa), creada por los procesos de proyección y combinación. Puesto que, sin la tabla original, no habría forma de identificar cuáles tuplas son genuinas y cuáles son espurias, en realidad se perdería información (aun cuando se tengan más tuplas) si las proyecciones se sustituyen por la relación original.

Se puede garantizar la descomposición sin pérdida al asegurarse de que, por cada par de relaciones que se combinarán, el conjunto de atributos comunes es una superclave de una de las relaciones. Esto se puede hacer al colocar atributos funcionalmente dependientes en una relación con sus determinantes y conservar los determinantes ellos mismos en la relación original.

De manera más formal, para descomposiciones binarias, si R se descompone en dos relaciones $\{R_1,R_2\}$, entonces la combinación no tiene pérdida si y sólo si alguno de los siguientes se mantiene en el conjunto de DF para R, o se implica mediante las DF en R:

$$R1 \cap R2 \rightarrow R1 - R2$$

o

$$R1 \cap R2 \rightarrow R2 - R1$$

En el ejemplo que se muestra en la figura 5.9, `role` no fue un determinante para `projName` o `empName`, de modo que la intersección de las dos proyecciones, `role`, no determina funcionalmente alguna proyección. Cuando se normalizaron relaciones se vieron muchos ejemplos de proyección sin pérdida.

Para una descomposición que involucra más de dos relaciones, la prueba anterior no se puede usar, así que se presenta un algoritmo para probar el caso general.

Algoritmo para probar la combinación sin pérdida

Dada una relación R(A1, A2, . . . , An), un conjunto de dependencias funcionales, F, y una descomposición de R en las relaciones R1, R2, . . . , Rm, para determinar si la descomposición tiene una combinación sin pérdida

1. *Construya una tabla m por n, S, con una columna para cada uno de los n atributos en R y una fila por cada una de las m relaciones en la descomposición.*
2. *Para cada celda S(i,j) de S,*

   si el atributo para la columna, Aj, está en la relación para la fila, Ri,

   entonces coloque el símbolo a(j) en la celda

   de otro modo coloque ahí el símbolo b(i,j)

FIGURA 5.9

Ejemplo de proyección con pérdida

FIGURA 5.9(a)

Tabla original EmpRoleProj

EmpRoleProj

empName	role	projName
Smith	diseñador	Nilo
Smith	programador	Amazonas
Smith	diseñador	Amazonas
Jones	diseñador	Amazonas

FIGURA 5.9(b)

Proyecciones de EmpRoleProj

Table 1

empName	role
Smith	diseñador
Smith	programador
Jones	diseñador

Table 2

role	projName
diseñador	Nilo
programador	Amazonas
diseñador	Amazonas

FIGURA 5.9(c)

Combinación de Table 1 y Table 2

empName	role	projName	
Smith	diseñador	Nilo	
Smith	diseñador	Amazonas	
Smith	programador	Amazonas	
Jones	diseñador	Nilo	←tupla espuria
Jones	diseñador	Amazonas	

3. *Repita el siguiente proceso hasta que ya no se hagan más cambios a S:*

   para cada DF X → Y en F

   para todas las filas en S que tengan los mismos símbolos en las columnas correspondientes a los atributos de X, iguale los símbolos para las columnas que representan atributos de Y mediante la regla siguiente:

   si alguna fila tiene un valor a, a(j), entonces establezca el valor de dicha columna en todas las otras filas igual a a(j)

   si ninguna fila tiene un valor a, entonces escoja cualquiera de los valores b, por decir b(i,j), y establezca todas las otras filas iguales a b(i,j)

4. *si, después de que se han hecho todos los posibles cambios sobre S, una fila está constituida completamente con símbolos a, a(1), a(2), . . . , a(n), entonces la combinación es sin pérdida. Si no hay tal fila, la combinación es con pérdida.*

Ejemplo:

Considere la relación *R(A,B,C,D,E)* que tiene descomposición que consiste en *R1(A,C)*, *R2(A,B,D)* y *R3(D,E)* con DF $A \rightarrow C$, $AB \rightarrow D$ y $D \rightarrow E$. La figura 5.10 ilustra el algoritmo. Con referencia a la figura 5.10(a) construya una fila por cada relación en la descomposición y una columna por cada uno de los cinco atributos de *R*. Para cada fila, coloque el valor *a* con el subíndice de columna en alguna columna cuyo encabezado represente un atributo en dicha relación, y el valor *b* con el subíndice usual de fila y columna en la columna para cualquier atributo que no esté en dicha relación. Por ejemplo, en la primera fila, para la relación

FIGURA 5.10

Prueba para combinación sin pérdida

R(A, B, C, D, E)
Descomposición: R1(A, C), R2(A, B, D), R3(D, E)
DF: A → C, AB → D, D → E

	A	B	C	D	E
R1(A, C)	a(1)	b(1, 2)	a(3)	b(1, 4)	b(1, 5)
R2(A, B, D)	a(1)	a(2)	b(2, 3)	a(4)	b(2, 5)
R3(D, E)	b(3, 1)	b(3, 2)	b(3, 3)	a(4)	a(5)

FIGURA 5.10(a)
Colocación inicial de valores

	A	B	C	D	E
R1(A, C)	a(1)	b(1, 2)	a(3)	b(1, 4)	b(1, 5)
R2(A, B, D)	a(1)	a(2)	a(3)	a(4)	a(5)
R3(D, E)	b(3,1)	b(3, 2)	b(3, 3)	a(4)	a(5)

FIGURA 5.10(b)
Tabla después de considerar todas las DF

R1(A,C), se coloca *a(1)* en la primera columna, para *A*, y *a(3)* en la tercera columna, para *C*. Dado que *B* no aparece en *R1*, se coloca *b(1,2)* en su columna. De igual modo, se coloca *b(1,4)* en la columna *D* y *b(1,5)* en la columna *E*, pues estos atributos no aparecen en *R1*. Ahora considere la DF $A \rightarrow C$, y busque las filas que concuerden con el valor del lado izquierdo, *A*. Se encuentra que las filas 1 y 2 concuerdan con el valor *a(1)*. Por tanto, se pueden igualar los valores *C*. Se encuentra que la fila 1 tiene un valor *a*, *a(3)*, en la columna *C*, de modo que en la figura 5.10(b) el valor de la columna *C* de la fila 2 se iguala a *a(3)*. Al considerar la segunda DF, $AB \rightarrow D$, no se pueden encontrar dos filas que concuerden en sus valores *A* y *B*, de modo que no hay posibilidad de realizar cambios. Ahora, al considerar la DF $D \rightarrow E$ se encuentra que las filas 2 y 3 concuerdan en sus valores *D*, *a(4)*, de modo que se pueden igualar sus valores *E*. Puesto que la fila 3 tiene un valor *E* de *a(5)*, el valor *E* de la fila 2 se cambia a *a(5)*, así como en la figura 5.10(b). Ahora se encuentra que la segunda fila de la figura 5.10(b) tiene toda valores *a*, y se concluye que la proyección tiene la propiedad de combinación sin pérdida.

La prueba para la propiedad de combinación sin pérdida para el caso general es mucho más compleja que la del caso de descomposición binaria. Sin embargo, si las proyecciones se limitan a proyecciones binarias sucesivas, cada una de las cuales es sin pérdida, el resultado final también será sin pérdida. Si se tiene una descomposición D que consiste en el conjunto $\{R_1,R_2\}$, que tiene la propiedad de combinación sin pérdida (confirmada fácilmente mediante la prueba del caso binario), y R_2 a su vez tiene una proyección sin pérdida $\{T_1,T_2\}$ (también probada fácilmente pues es binaria), entonces la descomposición D_2 que consiste en $\{R_1,T_1,T_2\}$ es sin pérdida.

Siempre es posible encontrar una descomposición FNBC que sea sin pérdida. En la sección 5.7.8 se presentará tal algoritmo.

5.7 Diseño relacional formal

5.7.1 Reglas de inferencia: axiomas de Armstrong

Para comenzar con el abordaje más formal de la normalización es necesario un conjunto de axiomas que proporcionen reglas para trabajar con las dependencias funcionales. Las reglas

de inferencia para dependencias funcionales, llamados **axiomas de inferencia** o **axiomas de Armstrong**, en honor a su desarrollador, se pueden usar para encontrar todas las DF lógicamente implicadas por un conjunto de DF. Estas reglas son **sonoras**, lo que significa que son una consecuencia inmediata de la definición de dependencia funcional y que cualquier dependencia funcional que se pueda derivar a partir de un conjunto dado de DF, usándolas, es verdadera. También son **completas**, lo que significa que se pueden usar para derivar toda inferencia válida acerca de las dependencias, de modo que, si una DF particular no se puede derivar a partir de un conjunto dado de DF usando estas reglas, entonces el conjunto dado de DF no implica dicha DF particular.

Sean A, B, C y D subconjuntos de atributos de una relación R. Los siguientes axiomas se sostienen (note que aquí AC significa la unión del conjunto A y el conjunto C):

- **Reflexividad.** Si B es un subconjunto de A, entonces $A \rightarrow B$. Esto también implica que $A \rightarrow A$ siempre se sostiene. Las dependencias funcionales de este tipo se llaman **dependencias funcionales triviales**.
- **Aumento.** Si $A \rightarrow B$, entonces $AC \rightarrow BC$.
- **Transitividad.** Si $A \rightarrow B$ y $B \rightarrow C$, entonces $A \rightarrow C$.

Las siguientes reglas se pueden derivar a partir de las tres anteriores:

- **Aditividad** o unión. Si $A \rightarrow B$ y $A \rightarrow C$, entonces $A \rightarrow BC$.
- **Proyectividad** o descomposición. Si $A \rightarrow BC$, entonces $A \rightarrow B$ y $A \rightarrow C$.
- **Pseudotransitividad.** Si $A \rightarrow B$ y $CB \rightarrow D$, entonces $AC \rightarrow D$.

Estas reglas se pueden usar para desarrollar una teoría formal de las dependencias funcionales, pero en vez de ello el texto se concentrará en sus aplicaciones prácticas.

5.7.2 Clausura de un conjunto de dependencias funcionales

Para la normalización es necesario identificar superclaves, claves candidatas y otros determinantes, lo que se puede hacer si se identifican todas las dependencias funcionales en una relación. También es necesario poder razonar acerca de todas las dependencias funcionales implicadas por un conjunto dado de dependencias funcionales. Hacer esto requiere la noción de la clausura de un conjunto de DF. Si F es un conjunto de dependencias funcionales para una relación R, entonces el conjunto de todas las dependencias funcionales que se pueden derivar de F, F^+, se llama **clausura de F**. Los axiomas de Armstrong son suficientes para calcular todas las F^+; esto es, si se fueran a aplicar estas reglas repetidamente, se encontrarían todas las dependencias funcionales en F^+. Sin embargo, la tarea obviamente sería muy compleja y tomaría mucho tiempo. Las cosas se simplificarían si se pudiera encontrar un conjunto más pequeño de DF que se pudiera usar en lugar de todas las F^+.

5.7.3 Clausura de un atributo

Dado un conjunto de dependencias funcionales F de una relación R, con frecuencia se tiene interés en encontrar todos los atributos en R que son funcionalmente dependientes de cierto atributo o conjunto de atributos, A, en R. A este conjunto de atributos se le llama **clausura de A** o A^+. Claro es, si A^+ es todo de R, entonces A es una superclave para R. A^+ se podría encontrar al calcular todos los F^+ y luego elegir sólo aquellas dependencias funcionales donde A es el determinante, pero hay un atajo. Se puede usar el siguiente algoritmo para encontrar A^+, dado un conjunto F de dependencias funcionales:

Algoritmo de clausura para conjunto de atributos A

resultado ← A;
while (resultado cambia) do
  for each dependencia funcional B → C en F
    if B está contenido en resultado then resultado ← resultado ∪ C;
end;
A+ ← resultado;

Por ejemplo, sea R una relación con atributos W, X, Y, Z y dependencias funcionales

W → Z
{Y,Z} → X
{W,Z} → Y

Calcule {W∪Z}+. Asigne {W∪Z} a `resultado` e ingrese el while por primera vez. Busque una DF cuyo determinante esté contenido en `resultado`, que al momento sólo tiene W∪Z. La DF {W,Z} → Y satisface el requisito, así que se asigna {W∪Z} ∪ Y a `resultado`. Dado que se tiene un cambio en resultado, se ingresa de nuevo el while. Ahora busque una DF donde W, Z, Y o cualquier combinación de estas tres sea un determinante. Puede usar {Y,Z} → X y ahora asigne {{W∪Z} ∪Y)} ∪ X a `resultado`. Puesto que se encontró que todo atributo de R está en {W∪Z}+, W∪Z es una superclave.

Para encontrar W^+ asigne W a `resultado` e ingrese el while por primera vez. La DF W → Z tiene W como determinante, así que asigne W∪Z a `resultado`. Dado que se tiene un cambio a `resultado`, ingrese el while de nuevo. Esta vez use {W,Z} → Y, así que {W∪Z} ∪ Y se asigna a `resultado`. Ahora se busca una DF donde cualquier combinación de los atributos W, Z, Y aparezca como el determinante. Esta vez puede usar {Y,Z} → X, y ahora asigne {{W∪Z}∪Y} ∪ X a `resultado`. Dado que se encontró que todo atributo de R está en W^+, W es una superclave para esta relación. Puesto que no tiene subconjunto propio que sea también una superclave, W también es una clave candidata. Note que ahora se sabe que W∪Z no es una clave candidata.

Es fácil verificar que {Y∪Z} no es una superclave. Comience con `resultado` ←{Y∪Z}. Al usar {Y,Z} → X, `resultado` se convierte en {Y∪Z}∪X. Ahora se busca una DF donde alguna combinación de Y, Z, X es el determinante. Dado que no existe alguna, no se puede agregar algún atributo nuevo a `resultado`. Esto significa que {Y∪Z}+ sólo es {{Y∪Z}∪X}, de modo que W no es funcionalmente dependiente de {Y∪Z}, lo cual significa que {Y∪Z} no es una superclave.

El algoritmo de clausura también permite determinar si existe una dependencia funcional particular en R. Para los conjuntos de atributos A y B, si quiere determinar si A → B, puede calcular A^+ y ver si incluye a B.

5.7.4 Identificación de dependencias funcionales redundantes

Dado un conjunto de dependencias funcionales se desea poder sustituirlas por un conjunto de DF equivalente pero más pequeño. Una forma de hacerlo es determinar si alguna de ellas es **redundante**, lo que significa que se puede derivar de las otras. Para hacerlo, puede usar el siguiente algoritmo:

Algoritmo para determinación de redundancia en un conjunto de DF

1. Elija una DF candidata, por decir X → Y, y remuévala del conjunto de DF.
2. resultado ← X;

```
while (resultado cambia y Y no está contenida en resultado) do
    for each DF, A → B, que permanece en el conjunto reducido de DF
        if A es un subconjunto de resultado, then resultado ← ∪ B
end
```

3. si Y es un subconjunto de resultado, entonces la DF X → Y es redundante.

Entonces se podría remover la DF X → Y del conjunto, pues se puede derivar de las otras DF. Al probar cada DF en el conjunto en turno y remover cualquiera que se pueda derivar de las otras, entonces repetir este proceso con las restantes DF puede encontrar un conjunto no redundante de DF equivalente al conjunto original.

Por ejemplo, suponga que se tienen los siguientes conjuntos de DF:

(1) W → Z
(2) W → Y
(3) {Y,Z} → X
(4) {W,Z} → Y

Comience por probar (1), W → Z. Asigne W a `resultado`. Ahora busque una DF (distinta a (1)) en la que W sea el determinante. Se encuentra una en (2), W → Y. Ahora asigne W ∪ Y a `resultado`. Al buscar una DF que tenga un determinante que esté contenido en W ∪ Y, no se encuentra alguna. Por tanto, no hay posibilidad de demostrar que Z está contenida en `resultado` y se concluye que (1) es no redundante.

A continuación pruebe (2), W → Y. Asigne W a `resultado`. Al buscar una DF distinta a (2) cuyo determinante sea W, se encuentra una en (1), así que se puede asignar W∪Z a `resultado`. Ahora busque una DF cuyo determinante esté contenido en W∪Z. Se encuentra (4), {W,Z} → Y, y ahora se puede asignar {W∪Z}∪Y a `resultado`. Dado que Y ahora está contenida en `resultado`, se puede salir del while y concluir que (2) es redundante. Ahora se elimina (2) del conjunto de DF.

Al probar (3) se asigna Y∪Z a `resultado`. Busque una DF distinta a (3) o (2) (que se eliminó en el paso anterior) cuyo determinante esté contenido en Y∪Z. Al no encontrar alguno se concluye que (3) no es redundante.

Al probar (4) se asigna W∪Z a `resultado`. En (1) se ve que el determinante, W, está contenido en `resultado`, así que se podría agregar el lado derecho de (1), Z, a `resultado`, pero ya está ahí. No hay otra DF cuyo determinante esté contenido en W∪Z, excepto para (2), que se eliminó. Por tanto, se concluye que (4) no es redundante en el conjunto reducido de DF pues no es posible obtener Y en `resultado`. El conjunto final de DF es

(1) W → Z
(3) {Y,Z} → X
(4) {W,Z} → Y

5.7.5 Cubiertas (covers) y conjuntos equivalentes de DF

Si F y G son dos conjuntos de DF para alguna relación R, entonces F es una **cubierta** (cover) para G si toda DF en G también está en F^+. Esto significa que toda DF en G se puede derivar a partir de las DF en F, o que G^+ es un subconjunto de F^+. Para probar que F es una cubierta para G, examine cada DF X → Y en G. Luego calcule X^+ en F y demuestre que X^+ contiene los atributos de Y. Si eso se mantiene verdadero para todas las DF en G, entonces F es una cubierta para G. Para una relación R, dos conjuntos de DF, F y G, se dice que son **equivalentes** si y sólo si F es una cubierta para G y G también es una cubierta para F (es decir, $F^+ = G^+$). Para probar equivalencia, se prueba que F y G son cubiertas una de otra.

Si G es un conjunto de DF en R, y G es mayor, se desea tener la capacidad de encontrar un conjunto más pequeño de DF tal que todas las DF en G también están implicadas por dicho conjunto más pequeño, esto es, el conjunto más pequeño es una cubierta para G.

5.7.6 Conjunto mínimo de dependencias funcionales

Se dice que un conjunto de DF, F, es **mínimo** si satisface estas condiciones:

- El lado derecho de cada DF en F tiene un solo atributo. Esta forma se conoce como estándar o **canónica** para DF.
- Ningún atributo en el lado izquierdo de cualquier DF en F es **extraña**. Esto significa que, si $X \rightarrow Y$ es una DF en F, entonces no hay subconjunto propio S de X tal que $S \rightarrow Y$ se pueda usar en lugar de $X \rightarrow Y$ y el conjunto resultante es equivalente a F.
- F no tiene DF redundantes.

5.7.7 Cómo encontrar una cubierta mínima para un conjunto de DF

Se dice que una cubierta F para G es una **cubierta mínima** (también llamada cubierta **no redundante**) si F es una cubierta para G mas ningún subconjunto propio de F es una cubierta para G. Un conjunto de DF puede tener muchas cubiertas mínimas, pero siempre se puede encontrar una de ellas. Para hacerlo, comience con el conjunto de DF, G. Cada DF en G se expresa en forma canónica, esto es, con un atributo en el lado derecho. Luego se examina el lado izquierdo de cada DF, y se verifica cada atributo A en el lado izquierdo para ver si borrarlo no afecta a G^+. Si el borrado de A no tiene efecto, se borra del lado izquierdo. Esto elimina atributos extraños de todas las DF. A continuación se examina cada DF restante y se verifica para ver si es redundante, esto es, si borrarla no tiene efecto sobre G^+. Si es redundante, se elimina. El conjunto final de DF, que se llamará F, es irreductible y equivalente al conjunto original. A continuación se presenta el algoritmo.

Algoritmo para encontrar una cubierta mínima, F, para un conjunto dado de DF, G

1. *Sea* F $\leftarrow$ *G*;
2. *Para cada DF en* F *que no esté en forma canónica, es decir, de la forma* $X \rightarrow \{Y1, Y2, \ldots Yn\}$, *sustitúyala por las n DF* $X \rightarrow Y1$, $X \rightarrow Y2, \ldots, X \rightarrow Yn$;
3. Elimina *atributos extraños:*

   for each FD $X \rightarrow Y$ *en* F

   for each atributo A *que es un elemento de* X

   if $((F-\{X \rightarrow Y\}) \cup \{(X - \{A\}) \rightarrow Y\}$ *es equivalente a* F

   then sustituya $X \rightarrow Y$ *con* $X \rightarrow Y$ *con* $\{X - \{A\}) \rightarrow Y$ *en* F;
4. Elimine DF *redundantes:*

   for each DF restante $X \rightarrow Y$ *en* F

   si $(F-\{X \rightarrow Y\})$ *es equivalente a* F

   then *remueva* $X \rightarrow Y$ *de* F

5.7.8 Algoritmo de descomposición para la forma normal Boyce-Codd con combinación sin pérdida

Siempre es posible encontrar una descomposición, D, que sea forma normal Boyce-Codd y que tenga la propiedad de combinación sin pérdida. El proceso involucra encontrar cada

violación de FNBC y removerla al descomponer la relación que la contiene en dos relaciones. El proceso se repite hasta que todas tales violaciones se remueven. El algoritmo es

Dada una relación universal R y un conjunto de dependencias funcionales en los atributos de R:

1. D ← R;
2. while existe algún esquema de relación S en D que no sea ya FNBC
   {
   a. Encuentre una dependencia funcional X → Y en S que viole FNBC
   b. Sustituya S con dos esquemas de relación (S-Y) y (X,Y)
   }

5.7.9 Algoritmo de síntesis para descomposición de tercera forma normal

Siempre se puede encontrar una descomposición de tercera forma normal que sea sin pérdida y que preserve dependencias. El algoritmo para la descomposición de la tercera forma normal es:

Dada una relación universal R y un conjunto de dependencias funcionales, G, en R,

1. Encuentre una cubierta mínima F para G, usando el algoritmo dado en la sección 5.7.7.
2. Examine los lados izquierdos de todas las dependencias funcionales. If hay más de una dependencia funcional en F con el mismo lado izquierdo; por ejemplo, existen algún atributo o conjunto de atributos X, y uno o más atributos Ai tales que

   X→ A1, X→ A2, . . ., X→ An;

   then combine X con los atributos en el lado derecho para formar una relación que tenga el esquema R1(X, A1, A2,...,An) en el que X es la clave.

   Repita esto para todos tales determinantes en F.
3. if ninguno de los esquemas de relación resultantes contiene una clave para la relación universal R, then cree una nueva relación que contenga atributos que formen una clave para R.

5.8 Dependencias multivaluadas y cuarta forma normal

Aunque la forma normal Boyce-Codd es suficiente para remover cualquier anomalía debida a dependencias funcionales, mayor investigación por parte de Fagin condujo a la identificación de otro tipo de dependencia que puede causar similares problemas de diseño. Éstas son dependencias multivaluadas. Para ilustrar el concepto de dependencias multivaluadas, considere la siguiente relación:

```
JointAppoint(facId, dept, comité)
```

Aquí se supondrá que un miembro del personal docente puede pertenecer a más de un departamento. Por ejemplo, un profesor puede contratarse conjuntamente por los departamentos CSC y matemáticas. Un docente puede pertenecer a varios comités en toda la universidad, cada uno identificado por el nombre del comité. No existe relación entre departamento y comité. La figura 5.11(a) muestra una versión no normalizada de esta relación. Con la finalidad de hacer la relación 1FN se le debe rescribir de modo que cada celda tenga sólo un valor. Aunque el método preferido es crear una relación separada para cada atributo multivaluado, otro método para crear 1FN es mediante "aplanamiento" de la tabla, como se muestra en la figura 5.11(b). Note que uno está forzado a escribir todas las combinaciones de valores `dept` con valores `comité` para cada miembro docente, o de otro modo parecería que hay alguna relación entre `dept` y `comité`. Por ejemplo, sin la segunda fila, parecería que F101 está en el comité de presupuesto sólo como miembro del departa-

mento CSC, mas no como miembro del departamento Mat. Note también que la clave de la relación es ahora `{facId,dept,comité}`. La relación resultante es FNBC, pues el único determinante es la clave. Aunque se atendieron todas las dependencias funcionales, todavía hay anomalías de actualización, inserción y borrado. Si se quiere actualizar un `comité` al que pertenezca F101 de Presupuesto a Promoción, se necesita hacer dos cambios. Si quiere insertar el registro de un miembro del personal docente que no esté en algún comité, no podrá hacerlo, pues comité es parte de la clave, de modo que en dicha columna no se permiten valores nulos. Ésta es una anomalía de inserción. De igual modo, si F221 deja la membresía en el comité Biblioteca, se pierde el resto de la información almacenada para él, pues no está permitido tener un valor nulo para un atributo de la clave. Anteriormente se encontró que problemas similares eran causados por dependencias funcionales, pero no hay ninguna en este ejemplo, así que es necesario identificar una nueva causa. Aunque un miembro del personal docente no está asociado sólo con un departamento, ciertamente está asociado con un conjunto particular de departamentos. De igual modo, un docente está asociado con un conjunto específico de comités en algún momento dado. El conjunto de departamentos para una `facId` particular es independiente del conjunto de comités para dicho miembro del personal docente. Esta independencia es la causa de los problemas. Para ver cómo se pueden corregir, es necesaria otra definición.

> **Definición:** Sea R una relación que tiene atributos o conjuntos de atributos A, B y C. Existe una **dependencia multivaluada** del atributo B sobre el atributo A si y sólo si el conjunto de valores B asociados con un valor A dado es independiente de los valores C.

Esto se escribe como A —>> B y se lee como A multidetermina B. Si R tiene al menos tres atributos, A, B y C, entonces en R(A,B,C), si A —>> B, entonces A —>> C también.

A diferencia de las reglas para dependencias funcionales, que hacen ilegales ciertas tuplas en las relaciones, las dependencias multivaluadas hacen esenciales ciertas tuplas en una relación. En la tabla `JointAppoint1` normalizada que se muestra en la figura 5.11(b), uno está forzado a escribir ciertas tuplas porque se incluyeron otras. Por ejemplo, cuando se escribió la combinación de F101 con valores de departamento CSC y Mat, tuvieron que escribirse dos tuplas por cada uno de los valores de comité, Presupuesto y Currículum, y colocar cada valor de departamento en una tupla con cada valor de comité. Una definición exacta de la dependencia multivaluada describe las tuplas que deben aparecer.

Definición alternativa de dependencia multivaluada

Más generalmente, si R es una relación con dependencia multivaluada,

```
A —>> B
```

entonces, en cualquier tabla para R, si dos tuplas, t1 y t2, tienen el mismo valor A, entonces deben existir otras dos tuplas t3 y t4 que obedezcan las siguientes reglas:

1. t3 y t4 tienen el mismo valor A que t1 y t2
2. t3 tiene el mismo valor B que t1
3. t4 tiene el mismo valor B que t2
4. si R – B representa los atributos de R que no están en B, entonces t2 y t3 tienen los mismos valores para R – B y
5. t1 y t4 tienen los mismos valores para R – B

La dependencia A —>> B se llama **dependencia multivaluada trivial** si B es un subconjunto de A o A ∪ B es toda de R. Ahora está listo para considerar la cuarta forma normal.

FIGURA 5.11

Ejemplo de cuarta forma normal

FIGURA 5.11(a)

Tabla JointAppoint no en 1FN

JointAppoint

facId	dept	comité
F101	CSC Mat	Currículum
F221	Biología	Biblioteca
F330	Inglés	Admisiones

FIGURA 5.11(b)

Tabla JointAppoint1 en 1FN que muestra dependencias multivaluadas

JointAppoint1

facId	dept	comité
F101	CSC	Presupuesto
F101	Mat	Presupuesto
F101	CSC	Currículum
F101	Mat	Currículum
F221	Biología	Biblioteca
F330	Inglés	Presupuesto
F330	Inglés	Admisiones

FIGURA 5.11(c)

Tablas Appoint1 y Appoint2 en 4FN

Appoint1

facId	dept
F101	CSC
F101	Mat
F221	Biología
F330	Inglés

Appoint2

facId	Comité
F101	Presupuesto
F101	Currículum
F221	Biblioteca
F330	Admisiones
F330	Presupuesto

> **Definición:** Una relación está en **cuarta forma normal** (4FN) si y sólo si es una forma normal Boyce-Codd y no hay dependencias multivaluadas no triviales.

La relación `JointAppoint1` que se muestra en la figura 5.11(b) no está en cuarta forma normal debido a las dos dependencias multivaluadas no triviales,

```
facId —>> dept
facId —>> comité
```

Cuando una relación es FNBC mas no 4FN, se le puede transformar en un conjunto equivalente de relaciones 4FN mediante proyección. Se forman dos relaciones separadas, y en cada una se coloca el atributo que multidetermina a las otras, junto con uno de los atributos multideterminados.

Para la relación `JointAppoint1`, se forman las dos proyecciones,

```
Appoint1 (facId, dept)
Appoint2 (facId, comité)
```

Estas dos relaciones están en 4FN. Se muestran en la figura 5.11(c).

5.9 Descomposición sin pérdida y quinta forma normal

Como se discutió en la sección 5.6.3, no todas las descomposiciones son sin pérdida, porque existen proyecciones cuya combinación no da de vuelta la relación original. Como ejemplo de una proyección con pérdida se usó la relación `EmpRoleProj (empName, role, projName)` que se muestra de nuevo en la figura 5.12(a). La tabla muestra cuáles empleados juegan cuáles papeles para cuáles proyectos. La tabla se puede descomponer mediante proyección en las dos tablas, `Table 1` y `Table 2`, que se muestran en la figura 5.12(b). (Por el momento, ignore la `Table 3`.) Sin embargo, cuando se combinan estas dos tablas, en la figura 5.12(c), se obtiene una tupla espuria adicional que no aparecía en la tabla original y por tanto se pierde información. La tabla original se puede recrear sólo mediante la combinación del resultado con `Table 3`, como se muestra en la figura 5.12(d). La recrea-

FIGURA 5.12
Ejemplo de dependencia de combinación

EmpRoleProj

empName	role	projName
Smith	diseñador	Nilo
Smith	programador	Amazonas
Smith	diseñador	Amazonas
Jones	diseñador	Amazonas

FIGURA 5.12(a)
Tabla original EmpRoleProj

Table 1

empName	role
Smith	diseñador
Smith	programador
Jones	diseñador

Table 2

role	proName
diseñador	Nilo
programador	Amazonas
diseñador	Amazonas

Table 3

empName	projName
Smith	Nilo
Smith	Amazonas
Jones	Amazonas

FIGURA 5.12(b)
Proyecciones de EmpRoleProj

empName	role	projName	
Smith	diseñador	Nilo	
Smith	diseñador	Amazonas	
Smith	programador	Amazonas	
Jones	diseñador	Nile	←tupla espuria
Jones	diseñador	Amazonas	

FIGURA 5.12(c)
Primera combinación usando Table 1 y Table 2

empName	role	projName
Smith	diseñador	Nilo
Smith	diseñador	Amazonas
Smith	programador	Amazonas
Jones	diseñador	Amazonas

FIGURA 5.12(d)
Combinación de primera combinación con Table3 sobre EmpName, projName

ción depende de esta combinación; esto es: la combinación final es necesaria para obtener de vuelta la tabla. Una **dependencia de combinación** existe cuando, para una relación R con subconjuntos de sus atributos A, B, . . . , Z, R es igual a la combinación de sus proyecciones sobre A, B, . . . , Z.

> **Definición:** Una relación está en **quinta forma normal** si toda dependencia de combinación es implicada por las claves candidatas.

En esencia, esto significa que las únicas descomposiciones válidas son aquellas que involucran claves candidatas. Las dependencias de combinación se relacionan con dependencias multivaluadas, pero pueden ser muy difíciles de identificar porque son sutiles. Si un diseño consiste en relaciones que son todas 5FN, están en su forma útil más simple de modo que no hay nada a ganar al descomponerlas aún más, pues esto resultaría en una pérdida de información. Por desgracia, no hay una prueba simple para 5FN. Se cree que las dependencias de combinación son relativamente raras, de modo que los diseñadores con frecuencia detienen el proceso de normalización en 4FN, FNBC o 3FN (para preservar dependencias funcionales).

5.10 Forma normal dominio-clave

La forma normal final a definir por Fagin involucra los conceptos de dominio, clave y restricción. Fagin demostró que una relación en esta forma no puede tener anomalías de actualización, inserción y borrado. Por tanto, esta forma representa la forma normal final con respecto a estos defectos.

> **Definición:** Una relación está en **forma normal dominio-clave** (FNDC) si toda restricción es una consecuencia lógica de las restricciones de dominio o restricciones de clave.

La definición usa los términos dominio, clave y restricción. Como siempre, el dominio de un atributo es el conjunto de valores permisibles para dicho atributo. Fagin usa la palabra clave para dar a entender lo que se describió como superclave, un identificador único para cada entidad. La restricción es un término general que significa una regla o restricción que se puede verificar al examinar estados estáticos de la base de datos. Entonces, para que exista una restricción, debe ser capaz de establecerla como un predicado lógico, así como determinar si dicho predicado es verdadero o falso al examinar instancias de la relación. Aunque las dependencias funcionales, dependencias multivaluadas y dependencias de combinación son restricciones, también existen otros tipos, llamados restricciones generales. Es posible tener reglas acerca de relaciones entre atributos de una relación (restricciones intrarrelación) que no se expresan como dependencias, o puede haber reglas acerca de relaciones entre relaciones (restricciones interrelación). Por ejemplo, considere la relación Student (`stuId, ..., credits`). Suponga que se tiene una regla de que `stuId` tiene un prefijo que cambia conforme el estudiante avanza; por ejemplo, todo alumno de primer año (freshman) tiene un 1, todos los de segundo año (sophomores) un 2, etc., al comienzo de sus ID. Esto se podría expresar como la restricción general

- Si el primer dígito de `stuId` es 1, entonces `credits` debe estar entre 0 y 30.
- Si el primer dígito de `stuId` es 2, . . .
- . . .

Un ejemplo familiar de una restricción interrelación resulta una restricción de integridad referencial. Considere `Student (stuId, lastName, ...)` y `Enroll (stuId, courseNo, grade)`. Aquí, una restricción interrelación es

- Para cada tupla en Enroll debe haber una tupla en `Student` con el mismo `stuId`.

Para que una relación sea FNDC, las restricciones intrarrelación deben ser expresables como restricciones de dominio o restricciones de clave. La restricción sobre `Student` se podría expresar al dividir `Student` en cuatro relaciones diferentes. Por ejemplo, para alumnos de primer año, se puede tener

`Stu1(stuId, . . ., credits)` con las restricciones de dominio

- `stuId` debe comenzar con un 1
- `credits` debe estar entre 0 y 30.

Entonces se tendría `STU2` con restricciones de dominio similares para estudiantes de segundo año, `STU3` para los de tercer año y `STU4` para los de cuarto año. Para la restricción interrelación, tendría que restringir el dominio de `stuId` en `Student` a los valores realmente representados en `Enroll`. Sin embargo, la definición de Fagin no se extiende a restricciones interrelación, pues su objetivo fue definir una forma que permitiera la comprobación de restricciones generales dentro de una relación al verificar sólo las restricciones de dominio y clave de dicha relación. Puesto que una restricción interrelación como una restricción de integridad referencial involucra dos relaciones, no se considera una FNDC.

Desafortunadamente, aunque el concepto de forma normal dominio-clave es simple, no existe un método probado de convertir un diseño a esta forma.

5.11 El proceso de normalización

Como se afirmó al comienzo del capítulo, el objetivo de la normalización es encontrar un conjunto estable de relaciones que sea un modelo fiel de la empresa. Se encontró que la normalización elimina algunos problemas de la representación de datos y resulta en un buen esquema para la base de datos. Existen dos procesos diferentes que se podrían usar para desarrollar el conjunto de relaciones normalizadas, llamado análisis y síntesis. La mayoría de los diseñadores en vez de ello eligen comenzar con un diagrama entidad-relación y trabajar desde ahí.

5.11.1 Análisis

El abordaje de análisis o descomposición comienza con una lista de todos los atributos a representar en la base de datos y supone que todos ellos están en una relación sencilla llamada relación universal para la base de datos. Entonces el diseñador identifica dependencias funcionales entre los atributos y usa las técnicas de descomposición explicadas en este capítulo para dividir la relación universal en un conjunto de relaciones normalizadas. Los ejemplos de normalización en la sección 5.5 fueron todos ejemplos de descomposición. En particular, en la sección 5.5.5 se consideró un ejemplo en el que se tenía una relación sencilla llamada `Work`. En realidad, `Work` es una relación universal, pues todos los atributos a almacenar en la base de datos estaban en ella. Luego se escribieron las suposiciones y se identificaron cuatro dependencias funcionales. Las dependencias funcionales permitieron la realización de proyecciones sin pérdida, de modo que se desarrolló un conjunto de relaciones en FNBC que preservó todas las dependencias funcionales. Dado que no existían dependencias multivaluadas que estuvieran también en 4FN, y dado que no había proyecciones sin pérdida no triviales restantes, están en 5FN. Todas las restricciones expresadas en la lista de suposiciones y todas las dependencias identificadas en la discusión se consideraron en el desarrollo del conjunto de relaciones final. Las restricciones se representan como restricciones de clave en las relaciones resultantes, de modo que en realidad se tiene un conjunto de relaciones FNDC. Este ejemplo ilustra el proceso de análisis a partir de una relación universal. El algoritmo dado en la sección 5.7.8 para encontrar una descomposición FNBC es un algoritmo de análisis.

5.11.2 Síntesis

La síntesis es, en un sentido, el opuesto de análisis. En el análisis se comienza con una sola relación grande y se descompone hasta que alcanza un conjunto de relaciones normalizadas más pequeñas. En la síntesis se comienza con atributos y se les combina en grupos relacionados, usando dependencias funcionales para desarrollar un conjunto de relaciones normalizadas. Si el análisis es un proceso arriba-abajo, entonces la síntesis es abajo-arriba. El algoritmo que se presentó en la sección 5.7.9 para crear un conjunto de relaciones 3FN es un algoritmo de síntesis.

5.11.3 Normalización desde un diagrama entidad-relación

En la práctica, cuando existe un diagrama E-R completo, el proceso de normalización es significativamente más eficiente que o el análisis puro o la síntesis pura. Al usar las técnicas de mapeo descritas en la sección 4.8, el conjunto de tablas resultante por lo general está bastante bien normalizado. El diseñador puede comenzar por comprobar cada relación para ver si es FNBC. Si no lo es, se pueden examinar los determinantes que causan los problemas y la relación normalizada. Este proceso se ilustra en el proyecto de muestra para la Galería de Arte al final de este capítulo.

5.12 Cuándo detener la normalización

Sin importar el proceso utilizado, el resultado final debe ser un conjunto de relaciones normalizadas que preserven las dependencias y formen combinaciones sin pérdida sobre atributos comunes. Una pregunta importante es cuán lejos ir en el proceso de normalización. Idealmente, se intenta llegar a FNDC. Sin embargo, si eso resulta en una descomposición que no conserva dependencias, entonces se conforma con pérdidas. De igual modo, si intenta 5FN, 4FN o FNBC, es posible que no sea capaz de obtener una descomposición que conserve dependencias, así que uno se conformaría con 3FN en este caso. Siempre es posible encontrar una dependencia que preserve la descomposición para 3FN. Incluso así, puede haber razones válidas para no elegir también implementar 3FN. Por ejemplo, si tiene atributos que casi siempre se usen juntos en aplicaciones y terminen en diferentes relaciones, entonces siempre tendrá que hacer una operación de combinación cuando se les recupere. Un ejemplo familiar de esto ocurre al almacenar direcciones. Suponga que almacena nombre y dirección de un empleado en la relación

```
Emp(empId, apellido, nombre, calle, ciudad, estado, cp)
```

Como siempre, se supone que los nombres no son únicos. Se tiene la dependencia funcional

```
cp → ciudad,estado
```

que significa que la relación no es 3FN. Se le podría normalizar mediante descomposición en

```
Empl(empId,nombre,calle,cp)
Códigos(cp,ciudad,estado)
```

Sin embargo, esto significa que tendría que hacer una combinación siempre que quiera una dirección completa para una persona. En este caso, puede conformarse con 2FN e implementar la relación Emp original. En general, se deben tomar en cuenta los requisitos de desempeño para decidir cuál será la forma final. El proyecto de muestra de la Galería de Arte ilustra las negociaciones en la normalización.

5.13 Resumen del capítulo

Este capítulo trata con un método para desarrollar un conjunto adecuado de relaciones para el modelo lógico de una base de datos relacional. Se encontraron tres tipos de dependencias, **dependencias funcionales**, **dependencias multivaluadas** y **dependencias de combinación** para producir problemas en la representación de información en la base de datos. Muchos problemas involucraban **anomalías de actualización**, **inserción** y **borrado**. Se dice que un conjunto de atributo B es **funcionalmente dependiente** sobre un conjunto de atributos A en una relación R si cada valor A tiene exactamente un valor B asociado con él.

La normalización con frecuencia se realiza comenzando con un conjunto de relaciones diseñadas a partir del mapeo de un diagrama E-R y el uso de métodos heurísticos para crear un conjunto equivalente de relaciones en una forma normal superior. La **primera forma normal** significa que una relación no tiene atributos de valor múltiple. Una relación que no esté ya en 1FN se puede normalizar al crear una nueva tabla que consiste en la clave y el atributo multivaluado, al agregar columnas adicionales para el número esperado de repeticiones, o mediante "aplanamiento", al hacer el atributo multivaluado parte de la clave. Las relaciones en 1FN pueden tener anomalías de actualización, inserción y borrado, debido a **dependencias parciales** sobre la clave. Si un atributo no clave no es **completamente dependiente funcional** sobre toda la clave, la relación no está en **segunda forma normal**. En este caso se le normaliza mediante proyección al colocar cada determinante que sea un subconjunto propio de la clave en una nueva relación con sus atributos dependientes. La clave compuesta original también debe mantenerse en otra relación. Las relaciones en 2FN todavía pueden tener anomalías de actualización, inserción y borrado. Las **dependencias transitivas**, en las que uno o más atributos no clave determinan funcionalmente otro atributo no clave, causan tales anomalías. La **tercera forma normal** se logra al eliminar las dependencias transitivas usando proyección. El determinante que causa la dependencia transitiva se coloca en una relación separada con los atributos que determina. También se mantiene en la relación original. En la tercera forma normal, cada atributo no clave es funcionalmente dependiente de la clave, toda la clave y nada más que la clave. La **forma normal Boyce-Codd** requiere que cada determinante sea una superclave. Esta forma también se logra mediante proyección, pero en algunos casos la proyección separará los determinantes de sus atributos funcionalmente dependientes, lo que resulta en la pérdida de una restricción importante. Si esto ocurre, es preferible 3FN.

Tres propiedades deseables para las descomposiciones relacionales son **preservación de atributo**, **preservación de dependencia** y **descomposición sin pérdida**. Las proyecciones con pérdida pueden perder información porque, cuando se deshacen mediante una combinación, pueden resultar tuplas espurias. Para proyecciones binarias, puede garantizar que una descomposición es sin pérdida si la intersección de las dos proyecciones es un determinante para una de ellas. Para descomposiciones binarias, existe una prueba simple para la descomposición sin pérdida. También existe un algoritmo para probar la descomposición sin pérdida en el caso general. Sin embargo, si la descomposición se hace mediante repetidas descomposiciones binarias, la prueba más simple se puede aplicar a cada descomposición sucesiva. Las dependencias funcionales se pueden manipular formalmente con el uso de los **axiomas de Armstrong** o reglas de inferencia. El conjunto F^+ de todas las dependencias funcionales implicadas lógicamente por un conjunto dado F de dependencias funcionales se llama su **clausura**. La clausura se puede construir mediante la aplicación repetida de los axiomas de Armstrong, pero es un procedimiento largo. De igual modo, la clausura A^+ de un atributo A en un conjunto de dependencias funcionales es el conjunto de todos los atributos que A determina funcionalmente. Se puede calcular A^+ mediante un algoritmo simple. El algoritmo para clausura de atributo se puede usar para responder algunas preguntas muy importantes, como si A es una superclave, si A es una clave candidata y si una

dependencia funcional aparece en F^+. Para un conjunto de DF, F, existe un algoritmo simple para determinar si una DF particular es **redundante** en el conjunto, lo que significa que se puede derivar a partir de las otras DF en F. Si F y G son dos conjuntos de DF para alguna relación R, entonces F es una **cubierta** para G si cada DF en G también está en F^+. Si F es una cubierta para G y G también es una cubierta para F, entonces se dice que F y G son **equivalentes**, y $F^+ = G^+$. Un conjunto de dependencias funcionales es **mínimo** si se escribe en forma canónica (es decir, toda DF tiene sólo un atributo en el lado derecho), ningún atributo de un determinante es extraño y no hay DF redundantes. Se dice que una cubierta F para G es una **cubierta mínima** (cubierta **no redundante**) si F es una cubierta para G mas ningún subconjunto propio de F es una cubierta para G. Un conjunto de DF puede tener varias cubiertas mínimas, pero siempre es posible encontrar una de ellas mediante un algoritmo simple. Si una relación no es ya FNBC, se puede usar un algoritmo de descomposición para encontrar un conjunto equivalente de relaciones FNBC que forman una proyección sin pérdida. No siempre es posible encontrar una descomposición FNBC que también preserve dependencias. Sin embargo, siempre es posible encontrar una descomposición 3FN que sea tanto sin pérdida como conservadora de la dependencia. Para hacerlo existe un algoritmo de síntesis que comienza con una cubierta mínima para el conjunto de DF.

Las **dependencias multivaluadas** pueden ocurrir cuando existen tres atributos en una clave de relación, y dos de ellas tienen valores múltiples independientes para cada valor distinto del tercero. La cuarta forma normal requiere que una relación sea FNBC y no tenga dependencias multivaluadas. Se puede lograr 4FN mediante proyección, pero usualmente se hace sólo si las proyecciones resultantes preservan dependencias funcionales. Una relación está en quinta forma normal si no hay proyecciones sin pérdida no triviales restantes. De manera alternativa, una relación está en quinta forma normal si cada dependencia de combinación es implicada por las claves candidatas. No hay un método probado para lograr la quinta forma normal. La **forma normal dominio-clave** requiere que toda restricción sea consecuencia de restricciones de dominio o restricciones de clave. No existe método probado para producir forma normal dominio-clave.

En análisis se comienza con una relación, se identifican dependencias y se usa proyección para lograr una forma normal superior. El enfoque opuesto, llamado síntesis, comienza con atributos, encuentra dependencias funcionales y agrupa dependencias funcionales con el mismo determinante, y forma relaciones.

Al decidir cuál forma normal elegir para la implementación, considere la preservación de atributo, la proyección sin pérdida y la preservación de dependencia. Por lo general se elige la forma normal superior que permite todo esto. En la implementación también debe equilibrar desempeño contra normalización, así que a veces se acepta un diseño que está en una forma normal inferior por razones de desempeño.

Ejercicios

5.1 Considere la siguiente relación universal que conserva información acerca de libros en una librería:

```
Libros(titulo, isbn, autor, nombreEditorial, direcEditorial, totalCopiasOrdenadas,
copiasEnStock, fechaPublicacion, categoria, precioVenta, costo)
```

Suponga:

- El `isbn` identifica de manera única a un libro. (Sin embargo, no identifica cada copia del libro.)
- Un libro puede tener más de un autor.

- Un autor puede tener más de un libro.
- Cada nombre de editorial es único. Cada editorial tiene una dirección única: la dirección de las oficinas centrales de la firma.
- Los títulos no son únicos.
- `totalCopiasOrdenadas` es el número de copias de un libro particular que la librería solicitó alguna vez, mientras que `copiasEnStock` es el número que todavía no se vende en la librería.
- Cada libro tiene sólo una fecha de publicación. A la revisión de un libro se le da un nuevo ISBN.
- La categoría puede ser biografía, ciencia ficción, poesía, etc. El título solo no es suficiente para determinar la categoría. El `precioVenta`, que es la cantidad que carga la librería por un libro siempre está 20% arriba del costo, que es la cantidad que la librería paga a la editorial o distribuidor por el libro.

  a. Con estas suposiciones, y el establecimiento de las que considere necesarias, mencione todas las dependencias funcionales no triviales para esta relación.

  b. ¿Cuáles son las claves candidatas para esta relación? Identifique la clave primaria.

  c. ¿La relación está en tercera forma normal? Si no, encuentre una 3FN mediante descomposición de combinación sin pérdida de Libros que preserve dependencias.

  d. ¿La relación o conjunto resultante de relaciones está en forma normal Boyce-Codd? Si no, encuentre una descomposición por combinación sin pérdida que esté en FNBC. Identifique cualesquiera dependencias funcionales que no se conserven.

5.2 Considere la siguiente relación que almacena información acerca de estudiantes que viven en dormitorios en una universidad:

```
Universidad(apellido, stuId, dirCasa, telCasa, habitacNum, nombreCompañero, dirDorm,
status, planComida, cargoHabitac, cargoPlanComida)
```

Suponga:

- A cada estudiante se le asigna un número de habitación y puede tener varios compañeros de habitación.
- Los nombres de los estudiantes no son únicos.
- La universidad tiene varios dormitorios. `habitacNum` contiene un código para el dormitorio y el número de la habitación particular asignada al estudiante. Por ejemplo, A221 significa Adams Hall, habitación 221. Los nombres de los dormitorios son únicos.
- `dirDorm` es la dirección del edificio de dormitorios. Cada edificio tiene su propia dirección única. Por ejemplo, Adams Hall puede estar en 123 Main Street, Anytown, NY 10001.
- `status` menciona el estatus del estudiante: freshman (primer año), sophomore (segundo año), junior (tercer año), senior (cuarto año) o graduado.
- `planComida` menciona cuántas comidas por semana eligió el estudiante como parte de su plan alimenticio. Cada plan alimenticio tiene un solo `cargoPlanComida` asociado con él.
- `cargoHabitac` es diferente para distintos dormitorios, pero todos los estudiantes en el mismo dormitorio pagan la misma cantidad.

Para este ejemplo, responda las preguntas (a)-(d) como en el ejercicio 5.1.

5.3 Considere los siguientes atributos para tablas en un modelo relacional diseñado para seguir la información de una compañía de mudanza que mueve clientes residenciales, usualmente de una casa o departamento a otro:

`clienteId, clienteNombre, clienteDirActual, clienteTelActual, clienteNuevaDir, clienteNuevoTel, ubicacionLevanta, ubicacionDeja, fechaDeMudanza, horaInicio, pesoEstimado, costoEstimado, camion#Asignado, conductorNombre, conductorLicNo, costoReal, cantidadDeDaños, capacidadCamion, costoCasetas, impuestos, cantidadFinal, facturaNumero, cantidadPagada, fechaPago, chequeNumero, saldo`

Suponga:

- Aunque en la mayoría de los casos la `ubicacionLevanta` es la antigua dirección del cliente y la `ubicacionDeja` es la nueva dirección, existen excepciones, como cuando el mobiliario se mueve hacia o desde un almacén.
- Antes de la mudanza se proporciona una estimación usando una factura preimpresa que contiene un número de factura único. El costo real se registra en la misma forma una vez que la mudanza está completa. El costo real puede diferir del costo estimado. La cantidad final incluye el costo real de la mudanza, más impuestos y casetas.
- En la mayoría de los casos, el cliente paga la cantidad final de manera inmediata, pero en ocasiones paga sólo parte de la cuenta, en especial si la cantidad de daños es elevada. En dichos casos, existe una cantidad saldo, que es la diferencia entre la cantidad final y la cantidad pagada.

(a)-(d) Para este ejemplo, responda las preguntas (a)-(d) como en el ejercicio 5.1.

(e) ¿Cuáles tablas implementaría realmente? Explique cualquier desnormalización, omisión o adición de atributos.

5.4 El propietario de una compañía de alimentos y bebidas quiere una base de datos para seguir la huella de varios aspectos del negocio. Los clientes pueden ser individuos o empresas que hagan arreglos para eventos como bodas, bailes, cenas corporativas, recaudación de fondos, etc. La compañía es propietaria de un salón en el que sólo puede desarrollarse un evento a la vez. Además, los clientes pueden usar los servicios de la compañía para realizar eventos en sus hogares, centros de trabajo o lugares que el cliente renta, como mansiones históricas. Muchos de estos eventos fuera del salón tienen lugar al mismo tiempo. Los clientes que rentan el salón deben garantizar un mínimo de 150 invitados, pero el salón puede acomodar hasta a 200 personas. Las reservaciones para el salón se aceptan hasta con dos años de anticipación. Los clientes que proporcionen su propio espacio pueden tener cualquier número de invitados. Las reservaciones para los eventos externos por lo general se realizan con muchos meses de anticipación. La firma proporciona la comida, vajilla (platos, cubertería y vasos), mantelería, meseros y barman para el evento, sin importar si es en el salón o en alguna otra parte. Los clientes pueden elegir el color de la mantelería. La firma tiene menúes preparados que se identifican por número. Para un número de menú dado, existe un aperitivo, una ensalada, un plato principal y un postre. (Por ejemplo, el menú número 10 puede incluir coctel de camarones, ensalada César, prime rib y *mousse* de chocolate.) La firma cita un precio total por sus servicios, con base en el número de invitados, el menú, la ubicación y factores intangibles como cuán ocupados están en dicha fecha. La firma también puede contratar arreglos florales, músicos, animadores y fotógrafos, si el cliente lo solicita. El cliente paga a la compañía, que entonces paga al contratista. El precio que se carga al cliente por cada uno de éstos siempre está 10% arriba del costo a la compañía. Suponga que los nombres son

únicos. Suponga también que los músicos y animadores proporcionan sólo un tipo de música o entretenimiento.

Suponga que se identificaron los siguientes atributos:

clienteApellido, cNombre, cTel, cCalle, cCiudad, cEstado, cCp, eventoFecha, eventoHoraInicio, eventoDuracion, eventoTipo, numeroInvitados, ubicacionNombre, ubicacionCalle, ubicacionCiudad, ubicacionEstado, ubicacionCp, mantelColorSolicita, numeroMeseros, numeroBarman, precioTotal, floristaNombre, floristaTel, floristaCosto, floristaPrecio, musicaContacto, musicaTelContacto, musicaTipo, musicaCosto, musicaPrecio, animadorNombre, animadorTel, animadorTipo, animadorCosto, animadorPrecio, fotografoNombre, fotografoTel, fotografoCosto, fotografoPrecio, menuNumeroElegido, menuAperitivo, menuEnsalada, menuPrincipal, menuPostre.

(a)-(d) Para este ejemplo, responda las preguntas (a)-(d) como en el ejercicio 5.1.

(e) ¿Cuáles tablas implementaría realmente? Explique cualquier desnormalización, omisión o adición de atributos.

5.5 Considere las siguientes relaciones e identifique la forma normal más alta para cada una, como se proporcionan, y enuncie cualquier suposición que necesite realizar.

a. Work1 (empId, empName, dateHired, jobTitle, jobLevel)

b. Work2 (empId, empName, jobTitle, ratingDate, raterName, rating)

c. Work3 (empId, empName, projectNo, projectName, projBudget, empManager, hoursAssigned)

d. Work4 (empId, empName, schoolAttended, degreeAwarded, graduationDate)

e. Work5 (empId, empName, socialSecurityNumber, dependentName, dependentAddress, relationToEmp)

5.6 Para cada una de las relaciones del ejercicio 5.5 identifique una clave primaria y,

a. si la relación no está en tercera forma normal, encuentre una 3FN por descomposición de combinación sin pérdida que preserve dependencias.

b. si la relación o conjunto de relaciones resultante no está en forma normal Boyce-Codd, encuentre una descomposición por combinación sin pérdida que esté en FNBC. Identifique cualquier dependencia funcional que no se conserve.

5.7 Considere instancias de relación R(A, B, C, D) como se muestra en la figura 5.13. Para cada uno de los siguientes conjuntos de DF, determine si cada una de las instancias es consistente con las DF dadas:

a. $A \rightarrow B,C$
   $B \rightarrow D$

b. $AB \rightarrow C$
   $B \rightarrow D$

c. $AB \rightarrow CD$
   $AC \rightarrow B$

5.8 Dado el siguiente conjunto S de DF:

$S = \{A \rightarrow B,\ B \rightarrow C,\ AC \rightarrow D\}$

a. Encuentre la clausura de A, A^+.

b. ¿A es una superclave? Explique.

c. ¿B es una superclave? ¿Cómo lo sabe?

d. La DF $B \rightarrow D$, ¿está en S^+? ¿Cómo lo sabe?

e. Encuentre la clausura del conjunto de DF, S^+.

FIGURA 5.13
Instancias de R(A,B,C,D)

Instancia 1

A	B	C	D
a1	b1	c1	d1
a2	b2	c1	d2
a3	b1	c2	d3
a4	b3	c2	d3

Instancia 2

A	B	C	D
a1	b1	c1	d1
a2	b1	c2	d1
a3	b2	c1	d2
a4	b2	c2	d2

Instancia 3

A	B	C	D
a1	b1	c1	d1
a2	b1	c2	d1
a1	b2	c3	d2
a2	b2	c4	d2

5.9 Examine cada uno de los siguientes conjuntos de DF y encuentre cualquier DF redundante en cada uno. Proporcione una cobertura mínima para cada uno.

a. B → D
E → C
AC → D
CD → A
BE → A

b. A → CDE
B → CE
AD → E
CD → F
BD → A
CED → ABD

c. D → C
AB → C
AD → B
BD → A
AC → B

5.10 Considere la relación

R (A,B,C,D,E)

con DF

A → B
BC → D
D → BC
C → A

a. Identifique las claves candidatas de esta relación.

b. Suponga que la relación se descompone en

R1 (A, B)

R2 (B, C, D)

¿Esta descomposición tiene una combinación sin pérdida?

PROYECTO DE MUESTRA: NORMALIZACIÓN DEL MODELO RELACIONAL PARA LA GALERÍA DE ARTE

En la sección de proyecto de muestra al final del capítulo 3 se creó un diagrama E-R para la Galería de Arte. Al final del capítulo 4 se vio cómo mapear dicho diagrama a tablas. En la presente sección se quiere normalizar las relaciones. Aunque el proceso de normalización podría comenzar por elaborar una sola relación universal que mencione todos los atributos

y luego intentar aislar las dependencias funcionales entre los atributos, el trabajo se puede hacer más sencillo al usar las entidades y relaciones identificadas en el capítulo 3 y las tablas a las que se mapearon al final del capítulo 4. En la práctica, las tablas que resultan de tales mapeos por lo general ya casi están normalizadas. Por tanto, use las tablas mapeadas como punto de partida.

- Paso 5.1. Comience con la lista de las tablas a las que las entidades y relaciones del diagrama E-R se mapearon de manera natural, a partir de la sección del proyecto de muestra al final del capítulo 4. Para cada tabla en la lista, identifique las dependencias funcionales y normalice la relación a FNBC. Luego decida si las tablas resultantes se deben implementar en dicha forma. Si no, explique por qué.

Del mapeo resultaron las siguientes tablas:

Artist(firstName, lastName, street, city, state, zip, interviewDate, interviewerName, areaCode, telephoneNumber, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

PotentialCustomer(firstName, lastName, areaCode, telephoneNumber, street, city, state, zip, dateFilledIn, *preferredArtistLastName, preferredArtistFirstName,* preferredMedium, preferredStyle, preferredType)

Artwork(*artistLastName, artistFirstName,* workTitle, askingPrice, dateListed, dateReturned, dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted, *collectorSocialSecurityNumber*)

ShownIn(*artistLastName, artistFirstName, workTitle, showTitle*)

Collector(socialSecurityNumber, firstName, lastName, street, city, state, zip, interviewDate, interviewerName, areaCode, telephoneNumber, salesLastYear, salesYearToDate, *collectionArtistFirstName, collectionArtistLastName,* collectionMedium, collectionStyle, collectionType, salesLastYear, salesYearToDate)

Show(showTitle, *showFeaturedArtistLastName, showFeaturedArtistFirstName*, showClosingDate, showTheme, showOpeningDate)

Sale(invoiceNumber, *artistLastName, artistFirstName, workTitle,* amountRemittedToOwner, saleDate, salePrice, saleSalesPersonCommission, saleTax, saleTotal, *buyerLastName, buyerFirstName, salespersonSocialSecurityNumber*)

Buyer(firstName, lastName, street, city, state, zip, areaCode, telephoneNumber, purchasesLastYear, purchasesYearToDate)

Salesperson(socialSecurityNumber, firstName, lastName, street, city, state, zip)

Para la tabla `Artist`, identifique las DF.

firstName + lastName → todos los atributos

Parecería que

socialSecurityNumber → todos los atributos

Recuerde que FNBC permite a los determinantes que son claves candidatas permanecer en la tabla, así que no hay problema con dejar `socialSecurityNumber` en la tabla. Sin embargo, recuerde que se supuso que no siempre se tendría el registro federal de contribuyentes de los artistas cuyas obras fueran propiedad de coleccionistas, así que el valor de este atributo puede ser nulo para algunos artistas. No obstante, cuando aparece, es único en la tabla.

zip → city, state

Es posible que quiera considerar

areaCode → ? city, state

Se concluye que, con teléfonos celulares, el código de área no necesariamente está asociado con la ciudad y el estado de residencia o estudio del artista, de modo que ésta no se tiene como dependencia funcional. También considere si el número telefónico completo determina la dirección.

```
areaCode + telephoneNumber → ? street, city, state, zip
```

¿Es posible que dos artistas registrados con el mismo número telefónico tengan dos direcciones diferentes? Si el teléfono es de una casa o estudio, las direcciones deben ser las mismas (la dirección de la ubicación del teléfono). Si es un teléfono celular, los dos artistas tendrían que compartir el mismo número para aparecer en dos registros diferentes, de modo que es probable que también compartan la misma dirección en dicho caso. Esto se tendrá como una dependencia funcional. Entonces se puede preguntar

```
areaCode + telephoneNumber → ? todos los atributos
```

Se decide que éste no es el caso, pues se considera la posibilidad de que dos artistas puedan compartir el mismo hogar o estudio y el mismo número telefónico ahí, pero aún así tener diferentes nombres, estilos, etc. La tabla está en 1FN y 2FN, mas no en 3FN o FNBC, debido a las dependencias mencionadas. Por tanto, la tabla Artist se descompone del modo siguiente:

```
Artist1(firstName, lastName, interviewDate, interviewerName, areaCode,
telephoneNumber, salesLastYear, salesYearToDate, socialSecurityNumber,
usualMedium, usualStyle, usualType)

Phones(areaCode, telephoneNumber, street, zip)

Zips(zip, city, state)
```

Sin embargo, este diseño requeriría usar el número telefónico con la finalidad de obtener la calle y código postal de un artista, y que se hagan dos combinaciones siempre que se quiera obtener la dirección completa de un artista. Por cuestiones de eficiencia se llegará al acuerdo y la calle y código postal se pondrán de vuelta en la tabla Artist1. Se elige dejar la tabla Zips como está, y se nota que las tablas de códigos postales completos están disponibles para compras en forma electrónica. Ahora la forma para las tablas Artist es

```
Artist2 (firstName, lastName, interviewDate, interviewerName, areaCode, telephoneNumber,
street, zip, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle,
usualType)

Zips(zip, city, state)
```

Note que la clave primaria de Artist2 es compuesta, y que se convertirá en clave externa compuesta en otras tablas. Las claves externas compuestas requieren verificar múltiples campos en las combinaciones. En la práctica, es mejor usar un campo numérico para un valor de clave que una cadena de caracteres, que está sujeta a diferencias en pronunciación, mayúsculas y puntuación al ingresar datos. Los errores o variaciones de entrada de datos al ingresar datos cadena pueden causar errores cuando se intenta comparar valores; en particular, cuando los campos se usan como claves externas. Por tanto, se creará un identificador numérico único para cada artista, que se llamará `artistId`, y ésa será la clave primaria de la primera tabla. Todos los atributos de la primera tabla serán funcionalmente dependientes sobre ella. Este tipo de clave se llama **clave subrogada** y la mayoría de los sistemas de gestión de base de datos tienen un mecanismo para generar valores y seguir la pista de los valores para claves subrogadas. Oracle usa un concepto llamado **secuencia** para este propósito. Microsoft Access usa un tipo de dato **autonúmero** para el mismo propósito. De hecho, Access conmina al usuario a permitirle generar un campo clave de este tipo si el usuario rechaza especificar uno. En consecuencia, como tablas finales de artista se tiene

(1) Artist3(artistId, firstName, lastName, interviewDate, interviewerName, areaCode, telephoneNumber, street, *zip*, salesLastYear, salesYearToDate, socialSecurityNumber, usualMedium, usualStyle, usualType)

(2) Zips(zip, city, state)

Para `PotentialCustomer` se tienen muchas de las mismas DF que se vieron para Artist.

firstName + lastName + areaCode + telephoneNumber → todos los atributos

areaCode + telephoneNumber → street, city, state, zip

zip → city, state

Al usar el mismo razonamiento como se hizo para `Artist` se agregará un identificador único, `potentialCustomerId`, para usar como la clave primaria. Puede preguntar si existe una dependencia funcional entre el artista preferido y otras preferencias. Aunque puede haber alguna conexión, no es una verdadera dependencia funcional pues, por ejemplo, dos personas que admiren al mismo artista pueden preferir obras que estén en diferentes medios o estilos, acaso incluso producidas por el mismo artista.

Para FNBC estricta `PotentialCustomer` se descompondría del modo siguiente:

Customer1(potentialCustomerId, firstName, lastName, *areaCode, telephoneNumber,* dateFilledIn, *preferredArtistLastName, preferredArtistFirstName,* preferredMedium, preferredStyle, preferredType)

Phones(areaCode, telephoneNumber, street, *zip*)

Zips(zip, city, state)

Al usar la misma lógica que se utilizó para la información del artista, se elige crear una tabla que conserve la calle y código postal con los otros datos del cliente, y usar la tabla `Zips` que ya existe para determinar ciudad y estado. También se quiere usar `preferredArtistId` en lugar del apellido y el nombre. En consecuencia, al diseño de la tabla se agrega

(3) PotentialCustomer2 (potentialCustomerId, firstName, lastName, areaCode, telephoneNumber, street, *zip,* dateFilledIn, *preferredArtistId,* preferredMedium, preferredStyle, preferredType)

Para `Artwork`, se tienen las siguientes DF:

artistLastName + artistFirstName + workTitle → todos los atributos

Aquí se usa el nombre del artista como una clave externa. Puesto que se cambió la clave primaria de `Artist` a `artistId`, también se cambiará la clave externa, lo que sustituye el nombre por la Id en la tabla `Artwork`. Aunque existe alguna conexión lógica entre las fechas no hay dependencia funcional entre ellas. También se creará un identificador único para cada obra, `artworkId`,

(4) Artwork (artworkId, *artistId,* workTitle, askingPrice, dateListed, dateReturned, dateShown, status, workMedium, workSize, workStyle, workType, workYearCompleted, *collectorSocialSecurityNumber*)

Para la tabla `ShowIn` no existen dependencias funcionales no triviales, así que la tabla se podría conservar en su forma actual. Sin embargo, se desea usar artworkId para identificar la obra de arte.

(5) ShownIn (*artworkId, showTitle*)

En la tabla `Collector` se tienen las DF

socialSecurityNumber → todos los atributos

así como las DF vistas anteriormente que involucran números telefónicos y códigos postales. También se quiere usar `artistId` en lugar del nombre y apellido del artista. Por tanto, se elige crear la tabla que se muestra a continuación y usar la tabla `Zips` existente

(6) Collector1 (socialSecurityNumber, firstName, lastName, street, *zip*, interviewDate, interviewerName, areaCode, telephoneNumber, salesLastYear, salesYearToDate, *collectionArtistId*, collectionMedium, collectionStyle, collectionType, salesLastYear, salesYearToDate)

Para la tabla `Show` se tiene la DF

showTitle → todos los atributos

Sin embargo, se quiere sustituir el `artistId` por el nombre, como se hizo para las tablas anteriores. Puede existir alguna conexión entre el artista presentado y el título de la exposición, pero no necesariamente son dependientes funcionales. Por ejemplo, dos diferentes exposiciones pueden ambas presentar al mismo artista, pero sus títulos serán distintos. Lo mismo es cierto del tema y el título. También existe alguna conexión entre la fecha de apertura y la fecha de cierre de la exposición. Si se supone que sólo una exposición puede estar abierta en una fecha dada, entonces sólo habrá una fecha de cierre asociada con dicha fecha de apertura, y se tendría una dependencia transitiva que necesitaría remover. Agregue a las suposiciones que más de una exposición puede abrir al mismo tiempo. Ésta es una suposición razonable si la galería es suficientemente grande. En este acaso, puede haber exposiciones que tengan la misma fecha de apertura pero diferentes fechas de cierre. Dadas estas suposiciones, la tabla `Show` ya está normalizada.

(7) Show (showTitle, *showFeaturedArtistId*, showClosingDate, showTheme, showOpeningDate)

Para `Buyer`, se tiene la DF

firstName + lastName + areaCode + telephoneNumber → todos los atributos

así como las DF que involucran números telefónicos y códigos postales, como se vio anteriormente para `Artist` y para `Collector`. Como se hizo para `Artist` se creará una clave primaria numérica, `buyerId`. Al usar el mismo patrón para dichas tablas se diseña una nueva tabla `Buyer` y se usa la tabla `Zips` diseñada anteriormente.

(8) Buyer (buyerId, firstName, lastName, street, *zip*, areaCode, telephoneNumber, purchasesLastYear, purchasesYearToDate)

Para la tabla `Sale` se tiene la DF

invoiceNumber → todos los atributos

Dado que cada obra de arte se vende cuando mucho una vez, también se tiene

artworkId → todos los atributos

Dado que es permisible conservar una clave candidata en la relación, esto no presenta un problema. De nuevo, se sustituirá `artworkId` por el nombre del artista y el título, y la `buyerId` por el nombre del comprador como claves externas. Si la comisión es un porcentaje constante del precio de venta, entonces se tiene

salePrice → saleSalesPersonCommission

No se supone que el precio de venta determina el impuesto, pues algunos compradores, como las organizaciones no lucrativas, pueden estar exentos de impuestos. Sin embargo, la venta total es sólo la suma del precio de venta e impuesto, así que se tiene

salePrice + tax → saleTotal

En realidad, cualquiera de estos dos determina al otro. Puede pensar que se tiene la DF entre `salePrice` y la `amountRemittedToOwner`. Sin embargo, es posible que pueda

haber una demora entre el tiempo cuando una obra se vende y el tiempo en que se le paga al propietario, así que se podrían tener dos ventas con el mismo precio de venta y diferentes cantidades remitidas al propietario, puesto que una todavía no se paga. Al remover las DF identificadas aquí, se forma la nueva tabla

(9) `Sale1 (InvoiceNumber, artworkId, amountRemittedToOwner, saleDate, salePrice, saleTax, buyerId, salesPersonSocialSecurityNumber)`

También podría tener la tabla

`Commissions(salePrice, saleSalesPersonCommission)`

pero no se necesitaría almacenar ésta, pues la comisión se calcula fácilmente. De igual modo, debido a la relación aritmética entre los atributos, no necesita almacenar la tabla.

`Totals(salePrice, saleTax, saleTotal)`

Para `Salesperson` se tiene

`socialSecurityNumber` → todos los atributos
`zip` → `city, state`

Al remover la dependencia transitiva y usar la tabla `Zips` existente, se tiene

(10) `Salesperson (socialSecurityNumber, firstName, lastName, street, zip)`

Las tablas numeradas 1-10 se usarán como el conjunto final de tablas para el diseño relacional de esta base de datos.

- Paso 5.2. Actualice el diccionario de datos y cite las suposiciones según se requiera.

Es necesario agregar al diccionario de datos los nuevos identificadores creados, del modo siguiente:

artistId Identificador numérico único creado para cada artista.

artworkId Identificador numérico único creado para cada obra de arte.

buyerId Identificador numérico único creado por cada comprador.

potentialCustomerId Identificador numérico único creado por cada cliente potencial.

Se hace una nota de los ítems calculados que no se almacenan.

saleSalesPersonCommission Cantidad en dólares de la comisión para una vendedor por la venta de una obra de arte. *Ítem calculado*.

saleTotal Cantidad total, en dólares, de una venta, incluido precio e impuesto, para una obra de arte; *calculado a partir de salePrice y saleTax. Ítem calculado.*

Se ve que `Artist`, `Buyer`, `Collector`, `PotentialCustomer` y `Salesperson` comparten atributos que tienen el mismo significado, así que se eliminan las distinciones con base en las fuentes para los siguientes atributos: `name`, `firstName`, `lastName`, `address`, `street`, `city`, `state`, `zip`, `phone`, `areaCode`, `telephoneNumber` y `socialSecurityNumber`.

No hay cambios a la lista de suposiciones.

PROYECTOS ESTUDIANTILES: NORMALIZACIÓN DEL MODELO RELACIONAL PARA LOS PROYECTOS ESTUDIANTILES

- Paso 5.1. Comience con la lista de las tablas a las que las entidades y relaciones del diagrama E-R se mapearon de manera natural, a partir de la sección del proyecto de muestra al final del capítulo 4. Para cada tabla en la lista, identifique las dependen-

cias funcionales y normalice la relación a FNBC. Luego decida si las tablas resultantes se deben implementar en dicha forma. Si no, explique por qué. Proporcione el esquema final para el modelo relacional.

- Paso 5.2. Actualice el diccionario de datos y la lista de suposiciones según se requiera.