# AllFusion® ERwin® Data Modeler

## 手法ガイド

**r7**

Computer Associates®

# 目次

## 第 1 章: はじめに



## 第 2 章: 情報システム、データベースおよびモデル



## 第 3 章: 論理モデルの構築

## 第 4 章: キー ベース データ モデルの設計



## 第 5 章: エンティティと属性の名前付けと定義



## 第 6 章: リレーションシップの詳細定義

## 第 7 章: 正規化



## 第 8 章: 物理モデルの作成



## 付録 A: 依存エンティティの種類



## 用語集

## 索引

Chapter

1

# はじめに

AllFusion® ERwin® Data Modeler（以下、AllFusion ERwin DM）によるデータ モデリングにようこそ。これまでデータ モデルについて経験のない方には、データ モデリングとその使い方を理解する上で、この「手法ガイド」が役に立ちます。データおよびデータ モデルについて多少の経験がある方には、業務要件を認識する上で、モデリングがいかに効果的であるかを理解いただけます。情報モデルは、情報システムを新しく設計する場合や既存システムをメンテナンスする場合に、とても効果的です。

## 本書の目的

データ モデリングは複雑な面もありますが、本書を読み終わるまでには、たとえ初心者であっても、AllFusion ERwin DM の手法を利用し始めるまでに理解できるでしょう。本書の全体的な目的は次のとおりです。

- 実際のデータベース設計を行う上で AllFusion ERwin DM に必要なデータ モデリング手法の基本事項の説明。
- AllFusion ERwin DM がサポートする IDEF1X、IE モデリング言語の機能の説明。
- AllFusion ERwin DM のモデリング機能をより良く理解するための追加情報の説明。

本書では、AllFusion ERwin DM がサポートする次のデータ モデリング手法について説明します。

- **IDEF1X** - IDEF1X 手法は、米国空軍によって開発されました。現在では空軍、各種政府機関、航空および金融業界をはじめとして、企業全体を網羅するような大規模で厳密な情報モデリングが不可欠なさまざまな企業で使用されています。
- **IE（Information Engineering）** - IE 手法は、James Martin、Clive Finkelstein、その他の IE 専門家によって開発され、様々な業界で幅広く使用されています。

どちらの手法も、大規模で厳密さが要求される企業規模データ モデリングが不可欠な環境に適しています。

# 対象となるユーザ

本書は、次のユーザを対象としています。

- データベース設計の初心者: 情報モデリングの入門書、および AllFusion ERwin DM 手法のガイドとして利用してください。
- 経験豊富なデータ モデル作成者: AllFusion ERwin DM における IDEF1X および IE データ モデリングおよびアプリケーション開発者の手法ガイドとして利用してください。
- 経験豊富な IDEF1X、IE ユーザ: AllFusion ERwin DM がサポートする IDEF1X、IE の機能ガイドとして、また、両手法のマッピング対応ガイドとして利用してください。

# 表記法

本書では、次の表記法を使用しています。次の表で説明します。

| テキスト項目 | 表記 | 例 |
|---|---|---|
| エンティティ名 | かぎかっこで囲み、後ろに「エンティティ」という語を付加する | 「ビデオテープ」エンティティ |
| 属性名 | かぎかっこで囲む | 「映画名」 |
| カラム名 | かぎかっこで囲む | 「映画名」 |
| テーブル名 | かぎかっこで囲む | 「MOVIE_COPY」 |
| 動詞句 | 山形かっこで囲む | <レンタル用として利用できる> |

Chapter 2

# 情報システム、データベースおよびモデル

データ モデリングとは、情報システムの要件を定義するために、情報の構造を定義し、ビジネス ルールを捉えるプロセスを言います。データ モデルは、特定の RDBMS へ実装するプロジェクトに固有の要件と、その RDBMS を必要とする業務における一般的な要件の間のバランスを表しています。

一般的な構造化システム開発手法、特にデータ中心設計手法では、フロント エンド設計と要件分析に重点が置かれています。これら「トップダウン」方式の設計手法の多くは、データに関わるシステム要件を取り出して表現する手法として、AllFusion ERwin DMのデータ モデリングを使用します。これに対して、データ フロー ダイアグラム、分散モデル、イベント/状態モデルなどのプロセス モデルは、AllFusion® Process Modeler その他のツールで、プロセス要件を記録するために作成されます。開発の各フェーズで、異なるレベルのモデルが使用されます。

## データ モデリングの利点

業務の専門家とシステムの専門家が協力して作成するデータ モデルには、多くの利点があります。データ モデル作成から得られる利点は、一般的に以下の 2 種類に分類されます。

- 主としてモデル（作業の成果）に伴う利点
- モデル作成プロセス（作業）に伴う利点

### 完成したデータ モデルから得る利点

- データ モデルは実装から独立しているため、特定のデータベースやプログラミング言語を必要としません。
- データ モデルは、要求される仕様を明確にします。
- モデルはビジネス ユーザを主体とした手法です。モデルの内容と構造は、システム開発者ではなく、業務遂行者によって制御されます。制約条件や解決方法ではなく、要件そのものが重視されます。
- モデルで使用する用語は業務における用語です。 システム開発における用語は使用しません。
- 業務にとって何が重要かを中心に考えることができます。

データ モデルを作成するプロセスから得る利点

- プロジェクトの初期フェーズで行われる、モデル作成セッションでは、各分野から集まった人が業務要件や方針について検討できます。このセッションは、同じ要件に関わる社内各関連部門の担当者が一堂に会するチャンスともなります。
- 業務で共通して使用する用語の標準化が行われ、一貫性のある正確な定義を作成できます。これにより関係者間のコミュニケーションがスムーズになります。
- 初期フェーズでは、各業務関係者が担当業務の知識と情報を交換し、それをシステム開発者に伝えます。以降のフェーズでは、ソリューションを実現するスタッフに引き続き知識と情報を伝えます。
- 一連のセッションに参加することで、業務全体の中で各アクティビティがどのように機能するのかをより明確に理解できるようになります。また、プロジェクトの各部を全体を通した形で理解できます。重要なことは分業ではなく協調です。時間の経過と共に価値はどんどん変化し、協調的な視点を持つことが重要視されてきます。
- 一連のセッションはコンセンサスやチームワークを促進します。

業務領域をカバーするデータ構造の設計は、システム開発の一部に過ぎません。プロセス（機能）分析も同様に重要です。機能モデルは、実行方法を説明します。機能モデルは、階層分解図、データ フロー ダイアグラム、HIPO ダイアグラムなどで表すことができます。実際に作業を行えば、機能モデルとデータ モデルを同時に作成することの重要性に気が付くでしょう。システムが実行する機能を検討することにより、データ要件が明確になります。また、一般にデータを検討することにより、追加すべき機能要件を明確にすることができます。機能とデータはそれぞれシステム開発の表と裏に相当します。

AllFusion ERwin DM は、プロセス モデリングを直接サポートし、さまざまな技術と一緒に使用できます。たとえば、弊社の機能モデリング ツールである AllFusion Process Modeler を使用すると、IDEF0、IDEF3 ワーク フロー、データ フロー ダイアグラムの各手法を表記できます。AllFusion Process Modeler と AllFusion ERwin DM を併用すると、データ モデリング プロジェクトを実行中にプロセス分析を完了することができます。

## データ モデリング セッション

データ モデルの作成には、モデル構造だけではなく、業務データや業務プロセスを明確にするために行う、事実を検証していく多くのセッションも含まれます。優れた会議を行うのと同様に、優れたセッションを行うには、充分な準備と「リアルタイム」にセッションをまとめて行く技術が必要になります。一般にモデリング セッションは、業務担当者と技術専門家とを適宜に混ぜて同席させ、論議を円滑に進めて行く必要があります。これは、事前にスケジュールが練られ、議論すべき内容をカバーできるように充分に計画し、適切な結果が得られるように組織化する必要があることを意味しています。

可能ならば、機能とデータのモデリング作業は同時に行うべきです。これは、機能モデルがデータ モデルを検証して、新しいデータ要件を明確にできる傾向があるからです。このアプローチにより、データ モデルが機能要件をサポートすることが可能になります。機能モデルとデータ モデルの両方を 1 回のモデリング セッションで作成するためには、データ モデル作成者だけではなく、機能の洗い出しを担当するプロセス モデル作成者も参加することが重要です。

### セッションでの役割

セッションを決められた手順に従って、円滑に進めるためには、参加者の役割を定義し、セッションの進め方や規則について合意を得ていることが、絶対的な条件です。次の役割を設定すると、セッションが円滑に進みます。

- 進行役は、会議に必要となる設備を揃えて会議を開催し、参考資料を準備し、セッション中には必要に応じて議題が外れないように討論を導かなければなりません。
- データ モデル作成者は、モデルの開発および検証プロセスにおいてリーダーの役割を果たします。適切な質問により詳細情報をヒアリングしてグループを先導し、可能ならばヒアリングしながらモデル設計を行い、モデル構造をセッション参加者が参照できるような形で保存していかなければなりません。(若干の困難を伴いますが、)データ モデル作成者がセッションを円滑に進める役割も担うことも少なからずあります。
- データ アナリストは、セッションの書記として、モデルを構成するすべてのエンティティと属性の定義を記録します。また、業務専門家からの情報に基づき、エンティティや属性をサブジェクト エリア単位でグループ化します。サブジェクト エリアとは、データ モデル全体を管理し易く、意味のあるサブセットへとパッケージ化したものです。
- セッションの議題を熟知した専門家は、モデル構築に必要な業務情報を提供します。この役割を担当する者は、システムではなく業務の担当者で、分析されるのは自分たちの業務です。
- マネージャは、システムまたは業務のいずれかの担当者で、自分に割り当てられた役割(進行役、議題を熟知した専門家、その他)でセッションに参加します。しかし議事進行のためには、必要に応じて意思決定を行わなければなりません。また、セッションの結論を見出せない場合、調整する権限も持っています。

## IDEF1X モデリング手法のサンプル

AllFusion ERwin DM は、IDEF1X と IE のモデリング標準をサポートしていますが、手法に関する制限はありません。しかし、IDEF1X 手法でさまざまなモデル レベルを使い分けることは、システム開発の上で極めて効果的です。本マニュアルに記載するモデルは、IDEF1X の標準で表記します。実際には、個別の状況に合わせてより詳細な、または大まかなレベルで表記してください。

モデルのレベルは通常、非常に広範囲にわたるものですが、特定の DBMS で理解できる用語でデータベース デザインを表すのに必要な精度のレベルまで業務に対して掘り下げるというような重要な主エンティティの詳細ビューではありません。つまり、詳細の最も下位のレベルで、モデルは技術に依存します。たとえば IMS データベース用のモデルは、DB2 データベース用のモデルと大きく異なります。一方、論理的なレベルにおけるモデルは特定の DBMS からは独立しています。また、データベース管理システムには実装できない情報を表すこともあります。

以下に説明するモデリング レベルは、トップダウン方式のシステム開発におけるライフ サイクルのアプローチに対応するものです。詳細レベルは引き続き各プロジェクトのフェーズの中で作成されます。

最高位レベルのモデルとしてエンティティ リレーションシップ ダイアグラム（ERD）と、キー ベース（KB）モデルの 2 種類があります。ERD は、主要なビジネス エンティティとリレーションシップを識別します。これに対して KB モデルは第 3 正規形のリレーショナル モデルで、その業務で必要とされる情報（全エンティティを含みます）の範囲を設定し、詳細を表します。

その下のレベルは、全属性（FA : Fully Attribute）モデルと、変換モデル（TM : Transformation Model）の 2 種類です。FA モデルは第 3 正規形のモデルで、アプリケーション プログラムで処理するすべての詳細情報を含みます。一方、TM は、実際の DBMS に適した構造に変換されたモデルです。

TM は多くの場合、非正規化されて第 3 正規形ではなくなります。TM の構造は DBMS の性能、データ量、予想されるアクセス パスやアクセス頻度に基づいて最適化されます。ある意味でこれは、最終的な物理的データベース設計に相当します。

データベース設計には、システムの DBMS モデルが含まれます。業務情報システムを統合する段階では、DBMS モデルは統合システム全体に対するプロジェクト レベルでのモデルか、またはエリア レベルでのモデルとなります。

これらのレベルを以下の図 2-1 に示します。DBMS モデルはエリア レベルでも、プロジェクト レベルでも構いません。なお、ある業務について 1 つの ERD および KB モデルと、各実装環境にそれぞれ対応する複数の DBMS モデルが存在する場合も珍しくはありません。さらにデータベースを共有しないプロジェクトに対しては、その環境の中で新たなエリアやプロジェクト レベルをカバーする DBMS モデルが存在する場合もあります。それぞれの環境に対応したエリア レベルの DBMS モデルがあり、データがその環境内のプロジェクト全体で完全に共有されているのが理想的です。

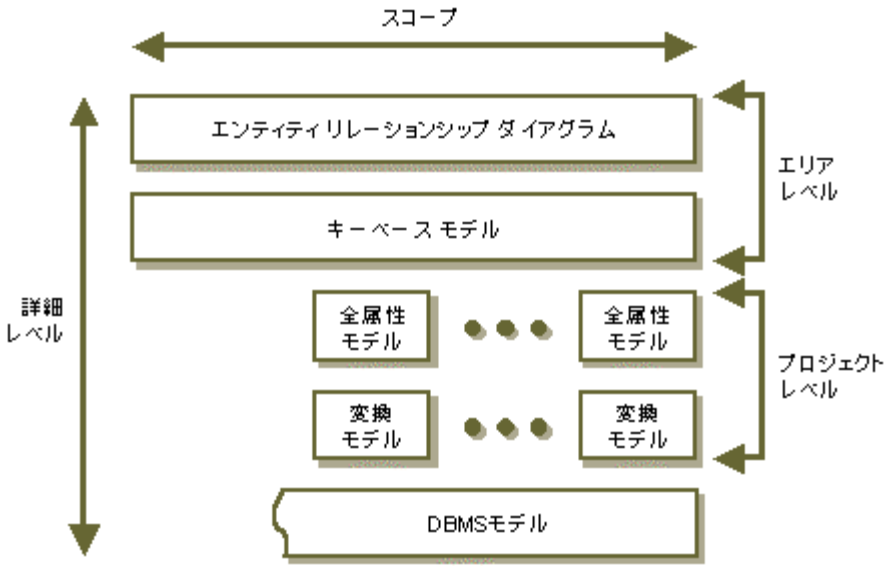



図 2-1 IDEF1X データベースの設計レベル

# 論理モデル

業務情報要件を捕捉するために使用される論理モデルには、エンティティ リレーション ダイアグラム、キー ベース モデル、全属性モデルの 3 つのレベルがあります。エンティティ リレーション ダイアグラムとキー ベース モデルは「エリア データ モデル」とも呼ばれます。これは、しばしば企業が 1 つのプロジェクトで取り組もうとしていることより大きく、より幅広い業務領域をカバーしているからです。これに対して、全属性モデルは「プロジェクト データ モデル」です。これは、通常 1 つのプロジェクトでサポートするデータ構造全体の一部を記述しているからです。

## エンティティ リレーションシップ ダイアグラム

エンティティ リレーションシップ ダイアグラム（ERD）は、幅広い業務領域をサポートする主要なエンティティとリレーションシップを示したハイ レベルなデータ モデルです。おもにプレゼンテーションまたは討論用のモデルです。

ERD は、情報システム開発における幅広い計画で必要となる十分な業務情報要件を表記するためのモデルです。このモデルは、属性情報などの詳細な内容を含まず、主要エンティティだけで構成されています。また、多対多リレーションシップを含んでいても問題なく、通常はキー情報を含みません。

### キー ベース モデル

キー ベース（KB）モデルは、幅広い業務領域をサポートする主要なデータ構造を含んでいます。すべてのエンティティと主キーが、サンプル属性といっしょに含まれます。

KB モデルは、対象業務領域をサポートするために必要なデータ構造とキーに関する幅広い業務ビューを表記するためのモデルです。このモデルには、詳細な実現レベル モデルを構築できる情報が含まれています。モデルは、エリア ERD と同じ範囲をカバーしますが、より多くの詳細情報が定義されています。

### 全属性モデル

全属性（FA）モデルは、1 つのプロジェクトに必要なすべてのエンティティ、属性、リレーションシップを含んだ第 3 正規形のモデルです。このモデルは、エンティティ インスタンスの容量、アクセス パスとアクセス頻度、データ構造に対する予想トランザクション アクセス パターンを含んでいます。

## 物理モデル

実装プロジェクトでは、さらに変換モデルと DBMS モデルという 2 つの物理モデルを必要とします。物理モデルは、システム開発者が論理モデルを理解し、データベース システムとして実装するために必要なすべての情報を捕捉します。変換モデルは、データ構造全体のうち、1 つの自動化処理をサポートするための一部分だけを表記する「プロジェクト データ モデル」でもあります。AllFusion ERwin DM は業務全体の中に存在する個々のプロジェクトをサポートしているので、モデル作成者は 1 つの大きなエリア モデルをサブジェクト エリアと呼ばれる複数のサブモデルに分割できます。サブジェクト エリアは、エリア モデルやモデル内の他のサブジェクト エリアから独立して、開発、レポート、データベースへのスキーマ生成を行えます。

### 変換モデル

変換モデルは、データベース管理者（DBA）に効率的な物理データベースを作成するために必要な情報を提供します。 また、データベースを構成するデータ要素とレコードのデータ ディクショナリ内で、定義と記録のための情報を提供します。 そしてまた、実際のデータにアクセスするプログラムのために物理構造を選択するアプリケーション チームを支援するためモデルです。

開発作業にとって適切な場合、モデルは物理データベース設計を本来の業務情報要件と比較するための基礎も提供します。

- 物理データベース設計がそれらの要件を充分にカバーしていることを例示します。
- 物理設計の選択肢とその影響（たとえば、何が満たされ、何が満たされないか）を記録します。
- データベースの拡張性と制約を明らかにします。

### DBMS モデル

変換モデルは DBMS モデルへと直接変換することが可能です。DBMS モデルは、RDBMS スキーマまたはデータベース システム カタログへと生成する物理データベースのオブジェクト定義を含んでいます。AllFusion ERwin DM は、DBMS モデルを使ってスキーマ生成機能を行います。主キーは一意のインデックスになります。代替キーと逆方向エントリもインデックスにすることができます。カーディナリティは、DBMS の参照整合機能、アプリケーション ロジック、または事後に検出して違反を修正するなどのいずれかの方法で実現できます。

## AllFusion ERwin DM によるデータ モデリングの利点

使用する DBMS または開発するデータ モデルの種類には関係なく、AllFusion ERwin DM を使用したモデリングにはさまざまな利点があります。最大の利点はシステムのドキュメンテーションを作成できることにあります。これは、データベースやアプリケーションの開発スタッフがシステム要件の定義やミーティングの資料に利用したり、またエンド ユーザとコミュニケーションに使用できます。

第 2 の利点は、参照整合性の制約を明示的に表記できることにあります。自動的に参照関係の整合性を保たせることが可能なリレーショナル モデルにおいて、参照整合性を維持することが最も大切なことになります。

第 3 の利点は、特定の RDBMS に特化しない論理モデルを作成し、このモデルから実際に使用する RDBMS に特化した物理スキーマを自動生成できることにあります。これにより、1 つの論理モデルから特定の RDBMS（たとえば、Oracle）へ物理スキーマを生成できるだけでなく、他の RDBMS（たとえば、SQL Server、SYBASE など）への物理スキーマも生成できます。

AllFusion ERwin DM によるデータ モデリングの最大の利点は、データ モデリング作業の結果を集約したダイアグラムを簡単に作成することができ、そのモデルから物理的なデータベース スキーマを生成できることです。

Chapter

3

# 論理モデルの構築

論理モデル構築の第一歩は、幅広い業務領域のハイ レベルなデータ モデルであるエンティティ リレーションシップ ダイアグラム(ERD)を開発することです。ERD は、エンティティ、属性、リレーションシップの主要な 3 つの要素で構成されています。ダイアグラムを業務ルールを表現するグラフィカルな言語とすると、エンティティは名詞、属性は形容詞または修飾句、リレーションシップは動詞と見ることができます。AllFusion ERwin DM によるデータ モデルの構築は、正しい名詞、動詞、形容詞を集め、それらを組み合わせるだけの作業です。

ERD は、情報システム開発における幅広い計画で必要となる十分な業務情報要求を表記するためのモデルです。このモデルは、属性情報などの詳細な内容を含まず、主要エンティティだけで構成されています。また、多対多リレーションシップを含んでいても問題なく、通常はキー情報を含みません。おもにプレゼンテーションまたは討論用のモデルです。

ERD は複数のサブジェクト エリアに分割することができます。サブジェクト エリアは、個々の業務機能を業務単位の観点、または関心のある特定分野という観点から定義できます。そして、大規模モデルを小規模なサブセットに分割することにより、エンティティの定義や管理が楽になる効用もあります。

ERD の開発にはさまざまな方法があります。これには、フォーマルなモデリング セッション(前の章で説明)から、幅広い領域を担当する業務マネージャとの個別インタビューまでが含まれます。

この章では、AllFusion ERwin DM で使用できるデータ モデリング手法を紹介し、業務の情報構造を記述するための豊富な機能の概要について説明します。

## エンティティ リレーションシップ ダイアグラム

リレーショナル データベースの構造を熟知されている方は、テーブルがリレーショナル データベースの基本的な構成要素であることをご存じでしょう。テーブルは、情報を構造化および格納するために使用されます。テーブルはデータのカラム(列)と行からできています。各行には、テーブルのインスタンスと呼ばれるデータ値が格納されています。

リレーショナル データベースでは、すべてのデータ値は原子データまたはアトミックデータ(テーブルの各セルに格納できる事実は 1 つだけで、意味を持つ複数のデータに分割できません)でなければなりません。データベース上のテーブルの間にもリレーションシップがあります。リレーショナル データベースでのリレーションシップとは、2 つのテーブルで 1 つまたは複数のカラムを共有していることを指します。

ERD（または、他の論理データ モデル）は、業務のデータ構造をモデル化するために、リレーショナル データベースにおけるテーブルやカラムと同様のモデル オブジェクトを持っています。リレーショナル データベースにおけるテーブルは論理的にはエンティティ、カラムは属性と呼ばれています。

ERD の中で、エンティティはボックスで表記され、エンティティ名はボックスの中に記述されます。エンティティ名に英語を使用する場合は、その名前は常に単数形となります（例: 「CUSTOMERS」ではなく「CUSTOMER」）。単数形の名詞を使用することによって、一貫した名前付け基準が保たれ、ダイアグラムをエンティティ インスタンスを説明するドキュメントの集合として捉えることができます。

図 3-1 は、あるビデオ ショップの顧客、貸し出しまたは販売可能な映画、ビデオテープの在庫を追跡するために作成されました。

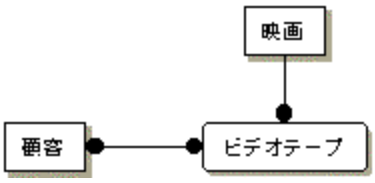



**図 3-1 エンティティ リレーションシップ ダイアグラムのサンプル**

ERD では、リレーションシップをモデル内のエンティティ間を結ぶ線によって表します。2 つのエンティティ間のリレーションシップは、一方のエンティティ内の事実が他方のエンティティ内の事実を参照しているか、その事実と関連付けられていることも暗示します。上の例では、ビデオ ショップが「顧客」と「ビデオテープ」という情報を追跡する必要があります。これら 2 つのエンティティ内の情報は関連付けられており、このリレーションシップは、「「顧客」は 1 つまたは複数の「ビデオテープ」を借りる」というステートメントによって表現できます。

## エンティティと属性の定義

エンティティは、情報の記憶の対象となる任意の人、場所、物、イベント、概念などを表します。より厳密にいうと、エンティティは、インスタンスと呼ばれる個々のオブジェクトの集合と考えることができます。インスタンスは、あるエンティティが 1 回出現することです。また、インスタンスは他のインスタンスから区別できなければなりません。

図 3-1 では、「顧客」エンティティが業務で考えられるすべての顧客の集まりを表します。「顧客」エンティティの各インスタンスは一人一人の顧客です。サンプル インスタンス表のエンティティに関する情報は、次の表のように一覧表示できます。

「顧客」エンティティ

| 顧客番号 | 顧客名 | 顧客住所 |
|---|---|---|
| 10001 | Ed Green | Princeton, NJ |
| 10011 | Margaret Henley | New Brunswick, NJ |
| 10012 | Tomas Perez | Berkeley, CA |
| 17886 | Jonathon Walters | New York, NY |
| 10034 | Greg Smith | Princeton, NJ |

各インスタンスは、関連エンティティに関する事実の集まりを表します。前述の表では、「顧客」エンティティの各インスタンスに「顧客番号」、「顧客名」、「顧客住所」に関する情報が含まれています。論理モデルでは、これらのプロパティをエンティティの属性と呼びます。それぞれの属性は、エンティティに関する個々の情報を取り込みます。

モデル内のエンティティをより詳しく記述するために、以下の図 3-2 に示すようにERD 内に属性を入れることができます。

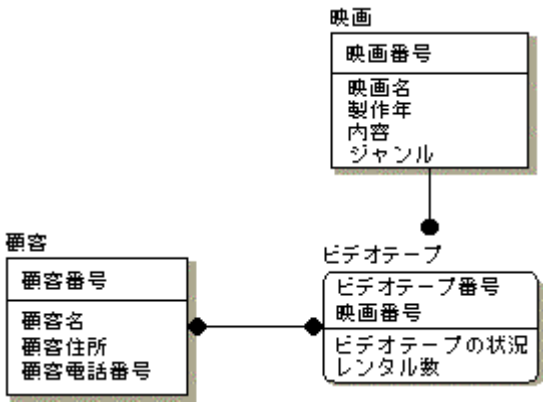



図 3-2 ERD と属性

## 論理リレーションシップ

リレーションシップは、エンティティ間の結び付き、リンク、または関連を表します。これは、エンティティがどのように関連しているかを示す、ダイアグラムの動詞句として表されます。このようにわかり易いルールを使うことにより、業務の専門家はより容易に、データの制約を検証し、リレーションシップのカーディナリティを明らかにすることができます。

いくつかの例を次に示します。

- 「チーム」は多くの「選手」<を持つ>。
- 「航空便」は多くの「乗客」<を輸送する>。
- 「ダブルス テニスマッチ」にはちょうど 4 人の「プレーヤー」<が必要である>。
- 「家」は 1 人または複数の「オーナー」<が所有する>。
- 「販売員」は多くの「製品」<を販売する>。

どの場合も、2 つのエンティティ間の接続は、いわゆる「1 対多」(One－to－Many)の関係になっています。これは、1 番目のエンティティの 1 つのインスタンス(かつ唯一のインスタンス)が、2 番目のエンティティの複数のインスタンスと関連していることを意味しています。1 対多の「1」側のエンティティは親エンティティと呼ばれます。1 対多の「多」側のエンティティは子エンティティと呼ばれます。

リレーションシップは 2 つのエンティティを結ぶ線として表され、一方の端にドットがあり、線の上には動詞句が記述されています。前の例では、山形かっこ内の語(たとえば、<を販売する>)が動詞句です。次の図 3-3 は、そのフライトの「航空便」と「乗客」のリレーションシップを表します。

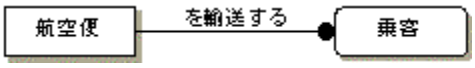



図 3-3 リレーションシップの例

## 多対多(Many-to-Many)リレーションシップ

多対多(Many－to－Many)リレーションシップは非固有(Non－Specific)リレーションシップとも呼ばれ、1 番目のエンティティ内のインスタンスが 2 番目のエンティティ内の 1 つまたは複数のインスタンスと関連付けられ、2 番目のエンティティ内のインスタンスが 1 番目のエンティティ内の 1 つまたは複数のインスタンスと関連付けられるような状況を表します。ビデオ ショップの例では、「顧客」エンティティと「ビデオテープ」エンティティの間に多対多のリレーションシップがあります。概念的に見ると、この多対多関係は「「顧客」が多数の「ビデオテープ」<を借りる>」と「「ビデオテープ」が多数の「顧客」<によって借りられる>」を表しています。

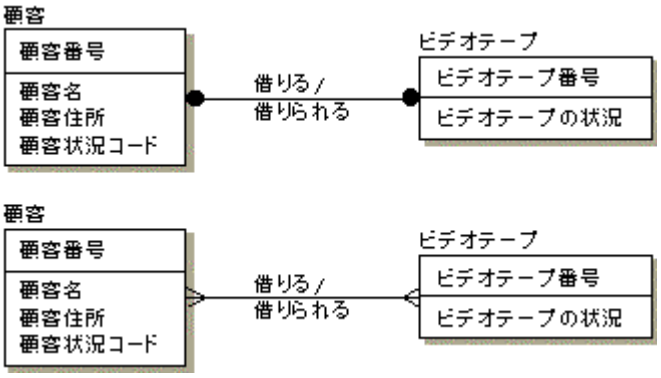



図 3-4 多対多(Many－to－Many)リレーションシップの例－IDEF1X(上)と IE(下)

通常、多対多（Many-to-Many）リレーションシップは、ダイアグラム開発の基本設計段階で ERD などでよく使用されます。IDEF1X では両端にドットのある実線で表されます。

多対多（Many－to－Many）リレーションシップは、他のビジネス ルールや制約を隠すことがあるので、モデリング プロセスのいずれかの時点で、それらを完全に調べる必要があります。たとえば、初期モデリング段階で識別された多対多（Many－to－Many）リレーションシップではラベルが不適切で、実際には関連するエンティティ間の 1 対多（One－to－Many）リレーションシップであることがあります。また、企業は日付またはコメントなどの多対多リレーションシップに関する追加事実を保存しなければならず、その結果、それらの事実を保存するために多対多（Many－to－Many）リレーションシップを別のエンティティに置き換えなければなりません。リレーションシップが正しくモデリングされるためには、すべての多対多（Many-to-Many）リレーションシップが後のモデリング段階で完全に説明されなければなりません。

## 論理モデル設計の検証

動詞句を正しく選ぶと、親から子への関係を「能動態（～するという表現）」の動詞句を使って「読む」ことができます。上の例の 1 つは次のように読むことができます。

「航空便」は多くの「乗客」<を輸送する>。

動詞句は、子エンティティの側から読むこともできます。子エンティティの側から読むには「受動態（～されるという表現）」の動詞句をよく使用します。この例を次に示します。

多くの「乗客」は「航空便」<によって輸送される>。

データ モデルは、モデルの対象となる領域を記述したビジネス ルールを表現しています。したがって、リレーションシップを読めば、その論理モデル設計が正しいか検証できます。動詞句は、リレーションシップによって具体化されるビジネス ルールの概略を示すものです。動詞句は、ビジネス ルールを厳密に記述するものではありませんが、これを書いておくことによってエンティティがどのように結びついているか、モデルを見る人にすぐに理解してもらうことができます。

モデルを読み取って理解した内容が、有効な文になっているかどうかを確認してみるのもよい練習となります。モデルから理解した内容を業務に対応させるのは、ビジネス ルールを適切に実現しているかどうかを確認する基本的な方法の 1 つです。

# データ モデルの例

次のデータベース モデルは、あるビデオ ショップ用に構築されたもので、図 3-5 に出現します。

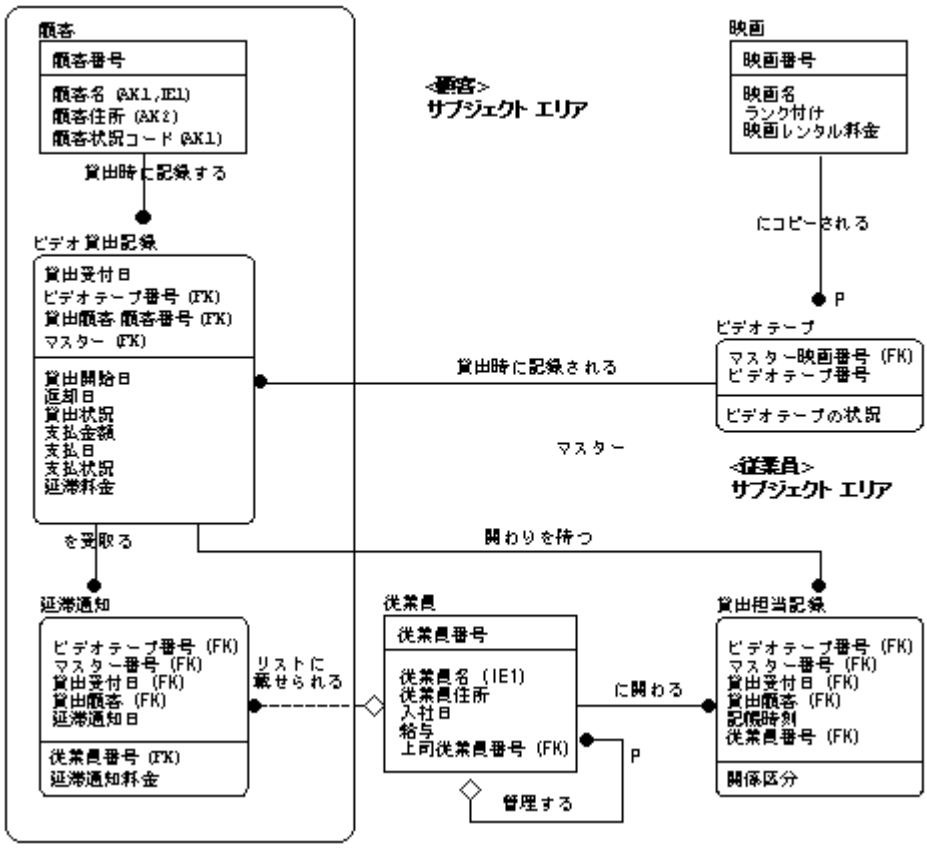



図 3-5　ビデオ ショップのデータ モデル

ビデオ ショップのデータ モデルとそこに示されるオブジェクトの定義から以下のことが言えます。

- 「映画」は 1 つまたは複数の「ビデオテープ」として在庫されています。「映画」に関して記録される情報には、名前、レーティング、および貸し出し料金があります。各「ビデオテープ」の全般的な状態が記録されます。
- 店の「顧客」は「ビデオテープ」を借ります。「ビデオ貸出記録」には、「ビデオテープ」の個々の貸し出しが「顧客」ごとに記録されます。日にちが経つと、1 つの「ビデオテープ」が複数の「顧客」に貸し出されます。
- 各「ビデオ貸出記録」には、ビデオの返却期限と、その期限を過ぎていないかどうかも記録されます。「顧客」にはそれまでの店との関係に応じて、小切手またはクレジット カードによる支払いを受け付けるか、現金のみの支払いかを示すクレジット ステータス コードが割り当てられます。

- 店の「従業員」は、関わりのタイプによって指定されているように、多数の「ビデオ貸出記録」と関わります。各レコードには少なくとも 1 人の「従業員」が関わっていなければなりません。1 人の「従業員」が同じ日に同じ貸出記録に複数回関わることがあるので、関わりはさらにタイム スタンプによって区別されます。
- 「ビデオテープ」の貸し出しに関して延滞料金を徴収する場合があります。テープの返却を「顧客」に求めるために「延滞通知」が必要な場合があります。「従業員」が「延滞通知」にリストされることもあります。
- 店は各「従業員」ごとに給与と住所に関する情報を保管します。「顧客」、「従業員」、「映画」を「番号」ではなく「名前」で検索したい場合があります。

これは比較的小さなモデルですが、ビデオ レンタル ショップに関するさまざまなことを表しています。ここから、業務に関するデータベースがどのようなものかだけではなく、業務自身の概要も把握できます。このダイアグラムにはいくつかのグラフィカル オブジェクトがあります。エンティティ、属性、リレーションシップは、他のシンボルとともにビジネス ルールを説明します。以降の各章では、それぞれのグラフィカル オブジェクトの意味と AllFusion ERwin DM を使って独自の論理データ モデルと物理データ モデルを作成する方法について詳しく学習します。

Chapter

4

# キー ベース データ モデルの設計

キー ベース(KB)モデルは、幅広い業務領域をサポートする主要なすべてのデータ構造を詳しく説明します。KB モデルの目的は、業務の対象となるすべてのエンティティと属性を含めることです。

また、KB モデルは、その名前が示すようにキーを含んでいます。論理モデルでは、キーはあるエンティティ内における一意のインスタンスを示します。キーは、物理モデルに実装されたときに、データに簡単にアクセスできるようにするためのものです。

基本的にキー ベース モデルはエンティティ リレーションシップ ダイアグラム(ERD)と同じスコープをカバーしますが、詳しい実現レベル モデルが構築できるようなコンテキストを含む詳細を明らかにします。

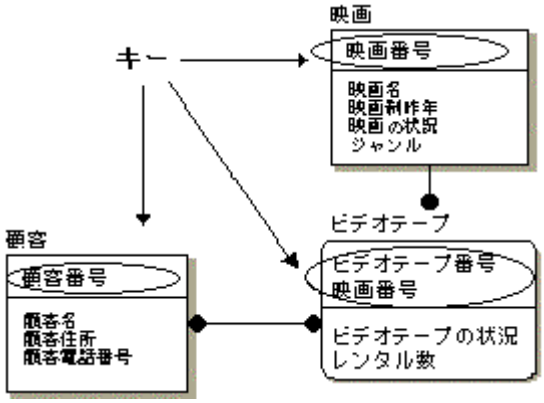



図 4-1 キー ベースの ERD モデル

# キーの種類の識別

データ モデル内のエンティティを作成する場合、最も重要な質問として「一意のインスタンスをどのように識別するか？」があります。正しい論理データ モデルを作成するには、エンティティ内の各インスタンスを一意に識別できなければなりません。

データ モデル内の各エンティティでは、仕切り線によって属性が 2 つのグループ（キー領域と非キー領域）に分割されています。この仕切り線より上はキー領域と呼ばれ、この線より下は非キー領域またはデータ領域と呼ばれます。「顧客」エンティティのキー領域には「顧客番号」があり、データ領域には「顧客名」、「顧客住所」、「顧客電話番号」があります。

エンティティと非キー領域

キー領域には、そのエンティティの主キーが含まれています。主キーは、エンティティの一意のインスタンスを識別するために選ばれた属性のセットです。主キーは 1 つ以上の主キー属性を持つことができます。この場合、選択された属性は、エンティティの各インスタンスに対して一意の識別子を形成していなければなりません。

エンティティは通常、多数の非キー属性を含んでいます。非キー属性は、仕切り線の下に表示されます。非キー属性は、エンティティのインスタンスを一意に識別しません。たとえば、あるデータベースにおいて、同じ顧客名のインスタンスが複数個あるとします。この場合、「顧客名」は一意でないので非キー属性になります。

## 主キーの選択

エンティティの主キーを選択することは重要なステップで、慎重な選択が必要です。場合によっては、実際に主キーを決める前に、いくつかの属性を検討する必要があります。これらの属性は、候補キー属性として参照されるものです。一般に、業務および業務データに詳しい業務ユーザが、候補キー特定作業を支援してくれます。

たとえば、データ モデル（および後続フェーズにおけるデータベース）の中で「従業員」エンティティを正しく使用するには、インスタンスを一意に識別できなければなりません。顧客テーブルでは、従業員名、「従業員」エンティティの各インスタンスに割り当てられた一意の従業員番号、属性のグループ（例: 氏名と生年月日）など、複数のキー属性から選択できます。

すべての候補キーのリストから主キーを選択するのに使用する規則は厳格ですが、あらゆるタイプのデータベースと情報に対して一貫して適用されます。規則では、属性または属性グループが次の条件を満たさなければなりません。

- インスタンスを一意に識別する。
- NULL 値を含まない。
- 時間とともに変化しない。インスタンスはキーで識別し、取り出します。キーが変化した場合、別のインスタンスになります。
- インデックス付けと検索を容易にするためにできるだけ短くする。他のエンティティのキーの組み合わせであるようなキーを使用する場合、キーの各部は他の規則に従います。

例として、「従業員」エンティティに対する以下の候補キー リストから、どの属性を主キーとして選ぶかについて考えてみます。

- 従業員番号
- 従業員名
- 従業員社会保障番号
- 従業員生年月日
- 従業員賞与額

上にリストした規則を使って「従業員」エンティティの候補キーを探すと、各属性について以下の分析結果が得られるかもしれません。

- 「従業員番号」はすべての「従業員」について一意なので、候補キーである。
- 「従業員名」はおそらく、候補キーとしては望ましくない。なぜなら、同姓同名（例: Mary Jones）の従業員が複数いる可能性があるからである。
- 「従業員社会保障番号」は一意であるが、すべての「従業員」が持っているとは限らない。
- 「従業員名」と「従業員生年月日」の組み合わせが（同じ日に生まれ、しかもこの会社に入社した John Smith という名前の人が 2 人いないかぎり）使えるかもしれない。これを候補キーにすることができる。
- 年間賞与を受け取る人は会社の「従業員」の一部である。したがって、「従業員賞与額」が多くの場合に NULL になる可能性がある。つまり、候補キーとしては使用できない。

分析の結果、2 つの候補キー（1 つは「従業員番号」、もう 1 つは「従業員名」と「従業員生年月日」を含んだ属性からなるグループ）が残りました。そして、最も短く、インスタンスの一意性が保証されることから、「従業員番号」が主キーとして選択されました。

エンティティのための主キーを選択する場合、モデル作成者は代理キー（surrogate key）を割り当てることがよくあります。代理キーは、エンティティ内のインスタンスを一意に識別するためにインスタンスに割り当てられる任意の番号です。「従業員番号」は、代理キーの例です。代理キーは、短かく、しかも短時間でアクセスでき、各インスタンスの一意の識別が保証されるので、多くの場合主キーに対する最善の選択肢となります。さらに代理キーはシステムによって自動的に生成可能なため、番号付けは連続的で、ギャップが生ずることはありません。

論理モデルに関する主キーを選択した結果、物理モデル内のテーブルに効率的にアクセスするのに必要な主キーが異なっていてもかまいません。主キーは、物理モデルやデータベースの要件に応じていつでも変更できます。

## 代替キー属性の指定

候補キーの中から主キーを選択したのち、残りの候補キーの一部またはすべてを代替キーとして指定できます。代替キーは、各種のインデックスを示すために使われることがあります。インデックスは、データにすばやくアクセスするために使われます。データ モデルの中で代替キーを示す場合、「AKn」という記号を使います。この n は、代替キー グループを構成する属性の末尾に付加される番号です。「従業員」エンティティの場合、「従業員名」および「従業員生年月日」は代替キー グループの構成要素です。


図 4-2 代替キー

## 逆方向エントリ属性の指定

逆方向エントリは、エンティティにアクセスする際に一般に使われる属性群ですが、主キーや代替キーと異なり、エンティティの 1 つのインスタンスを正確に検索できない場合があります。データ モデルにおいて逆方向エントリを表す場合、属性の後ろに「IEn」という記号を付加します。

たとえば業務部門は、従業員データベース内の情報を検索する際に、従業員の識別番号を使って検索するだけでなく、従業員名を使って検索したい場合もあるでしょう。名前で検索すると、複数のレコードが検索されることがあります。その場合は、さらに別のステップを実行して正確なレコードを見つける必要があります。属性を逆方向エントリ グループに割り当てると、一意でないインデックスがデータベース内に生成されます。

**注:** 1 つの属性が、代替キー グループと逆方向エントリ グループの両方に属する場合があります。


図 4-3 逆方向エントリ

# リレーションシップと外部キー属性

外部キーとは、親エンティティ内の主キーを定義し、また、リレーションシップを介して親エンティティから子エンティティへ移行する、属性のセットのことです。データ モデルの中で外部キーを表す場合、属性名の後ろに「FK」という記号を付加します。図 4-4 では、「チーム番号」の隣に「FK」という記号が付加されています。

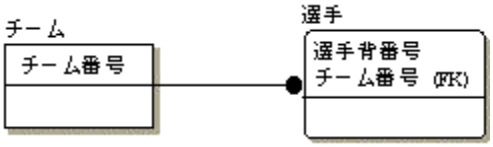



図 4-4 移行された外部キー（FK）

## 依存エンティティと独立エンティティ

データ モデルを開発していると、一意性を持たせるために外部キー属性の値に依存する特定のエンティティが見つかる場合があります。これらのエンティティでは、各エンティティを一意に定義するために、外部キーが子エンティティ（仕切り線より上）の主キーの一部でなければなりません。

リレーショナルの用語では、一意性のための外部キー属性に依存する子エンティティを依存エンティティと呼びます。IDEF1X の表記法では、依存エンティティを角の丸いボックスで表しています。

識別に関してモデル内の他のエンティティに依存しないエンティティを独立エンティティと呼びます。IE と IDEF1X では、独立エンティティを角の四角いボックスで表します。

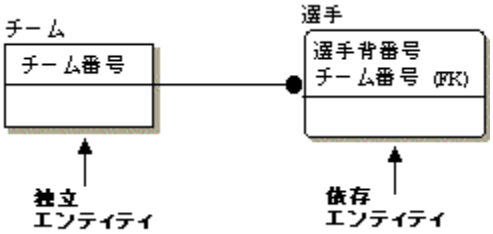



図 4-5 データ モデルにおけるエンティティの種類

依存エンティティは、さらにその親がなければ存在できない存在依存と親のキーを使わずには識別できない識別依存に分けられます。「選手」エンティティは識別依存ですが、存在依存ではありません。これは、「選手」は「チーム」がなくても存在できるからです。

これに対して、あるエンティティが別のエンティティに対して存在依存であるような状況があります。企業が顧客注文の追跡に使用する「注文」、および「注文」の各項目を追跡する「注文明細」という 2 つのエンティティを考えてみます。これら 2 つのエンティティのリレーションシップは、「「注文」は 1 つまたは複数の「注文明細」<を含んでいる>」と表すことができます。この場合、「注文明細」は「注文」に対して存在依存です。これは、業務コンテキストでは、「注文」がないかぎり「注文明細」を追跡しても意味がないからです。

## 依存型リレーションシップ

IDEF1X 表記法では、2 つのエンティティを接続するリレーションシップのタイプによって、依存エンティティと独立エンティティの概念が決まります。外部キーを子エンティティのキー領域に移行したい（結果として依存エンティティを作成したい）場合、親エンティティと子エンティティの間に依存型リレーションシップを作成できます。エンティティを結ぶ実線は、依存型リレーションシップを表します。IDEF1X 表記法では、その実線の子エンティティ側はドットで示されます（次の図 4-6 を参照）。

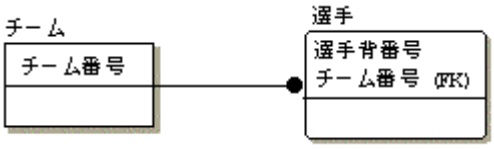



**図 4-6　IDEF1X 表記法における依存型リレーションシップ**

IE 表記法では、実線の子エンティティ側は「カラスの足跡」の形で示されます。

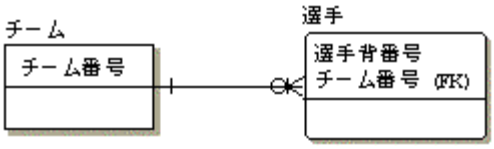



**図 4-7　IE 表記法における依存型リレーションシップ**

**注：** 標準 IE 表記には、エンティティの丸い角はありません。これは、AllFusion ERwin DM での手法間に互換性を持たせるため、IE 表記に加えた IDEF1X のシンボルです。

すでに気がつかれたように、依存型リレーションシップによって子エンティティにキーを移行することには、一部のデータベース クエリが簡単になる利点がありますが多くの欠点もあります。キーの移行はこのようにして生ずるべきではないと暗示している先進的リレーショナル理論があります。各エンティティはその主キーによってばかりではなく、システムのユーザには見えない論理ハンドルまたは代理キーによって識別するべきだという理論です。これについては活発な議論があります。興味のある方は E.F. Codd 氏や C.J. Date 氏のこの分野における著作をご参照ください。

## 非依存型リレーションシップ

非依存型リレーションシップの場合も、親エンティティと子エンティティを線で結びます。ただし、2 つのエンティティが非依存型リレーションシップで結ばれている場合、外部キーは子エンティティの非キー領域（仕切り線の下）に移行します。

AllFusion ERwin DM では、エンティティを結ぶ破線は非依存型リレーションシップを表します。たとえば、「チーム」エンティティと「選手」エンティティを非依存型リレーションシップで接続する場合、「チーム番号」が非キー領域に移行します（図 4-8 を参照）。

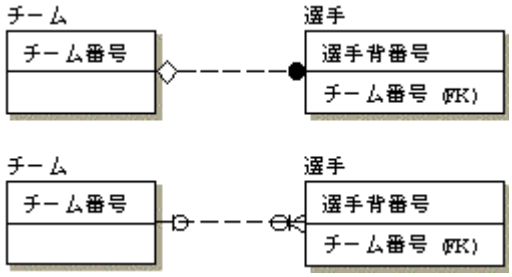



**図 4-8 非依存型リレーションシップ－IDEF1X 表記（上）と IE 表記（下）**

非依存型リレーションシップで移行されるキーは、この主キーの一部ではないので、非依存型リレーションシップから識別依存が生ずることはありません。この場合、「選手」は「チーム」と同様に独立エンティティです。

しかし、リレーションシップに関するビジネス ルールで、外部キーが NULL になれないように指定されている場合、リレーションシップは存在依存を反映できます。外部キーが存在しなければならない場合、子エンティティ内のインスタンスは、関連する親インスタンスも存在する場合のみ存在できることを暗示しています。

**注：** 依存型リレーションシップと非依存型リレーションシップは IE 方法論では使用されません。しかし、AllFusion ERwin DM ダイアグラムでは、IE 手法と IDEF1X 手法の互換性を保つために、この情報を実線と破線のリレーションシップ ラインで示しています。

## ロール名

外部キーがリレーションシップ内の親エンティティから子エンティティへ移行する場合、規定されたビジネス ルールから見てモデル内で二重の役割を果たしているといえます。両方のロールを理解するには、子エンティティでのロールを示すために移行キーの名前を変更するとよい場合があります。このようにして外部キー属性に割り当てられる名前をロール名と呼びます。結果として、ロール名は新しい属性を宣言します。この名前は、ビジネス ステートメントを外部キーに寄与するリレーションシップによって記述することを目的としています。

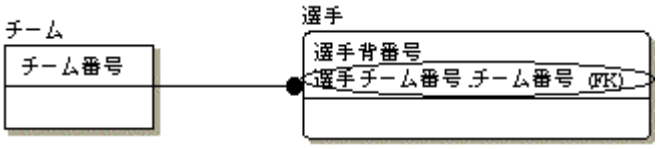



図 4-9 ロール名が割り当てられた外部キー

「選手」エンティティの外部キー属性である「選手チーム番号.チーム番号」は、ロール名を定義および表示するための構文を示しています。前半（ピリオド（.）より前）がロール名です。後半は、外部キーのもとの名前で、基本名とも呼ばれます。

ロール名は、外部キーに割り当てられると、他の外部キーと同様にリレーションシップを介して移行されます。たとえば、シーズンを通じて各「選手」がそれぞれのゲームで取った点がわかるように例を拡張するとします。図 4-10 に示すように「選手チーム番号」ロール名が（親エンティティ内の他の主キー属性とともに「得点プレー」エンティティに移行します。

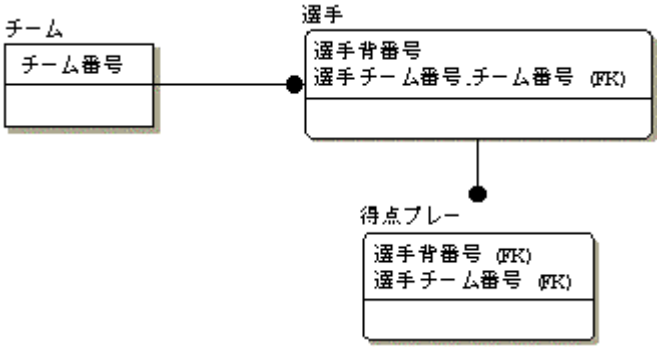



図 4-10 ロール名が付いた外部キー(FK)属性の移行を示すダイアグラム

**注：** ロール名は、従来のデータ モデルとの互換性をモデリングするためにも使用されます。この場合、外部キーには主キーとは異なる名前が付けられます。

Chapter

5

# エンティティと属性の名前付けと定義

一般的に、システム開発のときもそうですが、データ モデリングを行うときは、オブジェクトに対してわかりやすく、よく考えた名前を付けることがきわめて重要です。これによって、業務領域の明確で簡潔な曖昧さのないモデルを作ることができます。

名前付け基準と規則は、エンティティ リレーションシップ ダイアグラム(ERD)やキーベース(KB)ダイアグラムを含むすべての種類の論理モデルについて同じものです。

## エンティティと属性の名前付け

英語の場合、エンティティ名は「A FLIGHT <transports> zero , one or more PASSENGERs.」および「A PASSENGER <is transported by> one FLIGHT.」のように単数形にします。エンティティはインスタンスの集まりですが、各インスタンスを表すような名前をエンティティに付けます。「PASSENGER」エンティティの各インスタンスは一人一人の passenger を指すものであり、passenger 全体を表すものではありません。

属性名も単数形です。たとえば、「person－name」、「employee－SSN」、「employee－bonus－amount」は正しく名前が付けられた属性です。属性に単数形の名前を付けることは、1 つの属性で複数の事実を表現するなどの正規化エラーを避けるために役立ちます。「employee－child－names」や「start－or－end－dates」といった属性は複数形で、属性設計におけるエラーを助長します。

属性に名前を付けるコツとして、エンティティ名を接頭辞として使用する方法があります。次の規則があります。

- 接頭辞は(属性を)修飾する。
- 接尾辞は(属性を)明確化する。

この規則を使用すると、設計を簡単にチェックでき、よく起こる設計上の問題の多くを解消できます。たとえば「顧客」エンティティでは、属性「顧客名」、「顧客番号」、「顧客住所」と名前を付けることができます。属性「顧客請求番号」という名前を付けようとする場合は、規則を使用して、接尾辞「請求番号」は、接頭辞「顧客」よりも意味があることを確認します。実際にはそうならないので、属性をより適切な場所(「請求」エンティティなど)に移動します。

最初に定義を行わないと、エンティティや属性に名前を付けるのが困難な場合があります。エンティティや属性を適切に定義することは、適切な名前を付けることと同様にとても重要です。意味のある名前を付けるには、経験に加えて、モデルが表す内容についての基礎的な理解が必要です。

データ モデルは業務を説明しているので、できるかぎり意味のある業務用語を使用するのが最適です。エンティティに関する業務用語がない場合、モデル内の目的に合った名前をエンティティに付ける必要があります。

### シノニム、ホモニム、およびエイリアス

皆がすべて同じ言葉を話すというわけではありません。また、名前は常に正確に使用されるとは限りません。エンティティと属性は、データ モデル内ではそれらの名前で識別されるため、冗長なデータを表さないことを保証できるように、シノニムが解消される必要があります。次に、モデルを読み取るユーザが、どのエンティティにどの事実がキャプチャされているかを理解できるように、シノニムを正確に定義する必要があります。

また、エンティティや属性が何を表すかが明確に伝わるような名前を選ぶことも重要です。たとえば、「人」、「顧客」、「従業員」には若干の違いがあることは明らかです。これらはすべて個人を表していますが、それぞれ異なる特性または特質を持っています。しかし、「顧客」や「従業員」が異なるか、同じものを表すシノニムかを決めるのは業務ユーザの役割です。

名前の選択には充分注意し、異なるものを同じ名前で呼ばないようにします。たとえば、顧客を「消費者」と呼ぶのが慣習となっている業務領域では、名前に関する強制または主張はしないようにします。別名（同じものを表す別の名前）、ある「もの」とは似ていても、実は異なる新しい「もの」が見つかることがあります。このような場合、「消費者」は、「顧客」の他のカテゴリにはないようなリレーションシップに属する「顧客」のカテゴリです。

AllFusion ERwin DM では、モデリング環境で一意の名前付けを適用することができます。これによって、ホモニム（同名異義語）、曖昧な名前、モデル内のエンティティまたは属性の重複などの誤りを避けることができます。

## エンティティ定義

論理モデル内でエンティティを定義することは、エンティティの目的を詳しく説明し、どの事実をエンティティに含めたいかを明確にするためのよい方法です。 これは、モデルの明確さにとっても重要です。未定義のエンティティまたは属性は、あとのモデリング作業で誤解を招き、その結果、削除または統一されてしまうことがあります。

よい定義を記述することは思ったほど簡単ではありません。「顧客」とは何かを誰でもいえるでしょうか？「顧客」の定義を詳しく書いてみてください。最良の定義は、組織内のさまざまなビジネス ユーザや業務グループの視点に基づいて作成されます。多くの多様なユーザを満足させるような定義には、次のようないくつかの利点があります。

- 企業全体での明確さ
- 1 つの目的を持つ 1 つの事実に関するコンセンサス
- カテゴリ、すなわち一意であっても、同様の目的を持っているか、同様のデータを管理するエンティティ グループの容易な識別

ほとんどの組織と個人は、定義に関する自身の規定または標準を開発します。実際には、長い定義は、読者が定義される「もの」を理解するのを助けるような構造を持つ傾向があります。複数ページに渡る定義（たとえば、「顧客」）もあります。IDEF1X と IE に定義のための標準がなくても、最初に定義の構造について以下の項目を「標準」として採用するとよいでしょう。

- 説明
- 業務の例
- コメント

これらの項目については次の各項で詳しく説明します。

記述

オブジェクトの記述は、明解で簡潔な文章にすることが大切です。一般に、記述は短い方がよいでしょう。また、記述があまりに一般的になりすぎないように、そして、定義されていない用語は使用しないように注意してください。次に、2 つの例を示します。1 つは適切な記述で、もう 1 つは問題を抱えています。1 つめの例は、「「商品」とは、交換できる価値を持つものである。」という記述です。

これはよい記述です。 なぜなら誰かが「商品」と何かと交換したい場合、これを読むと何が「商品」かがわかります。誰かがピーナッツ 3 つとガム 1 つをビー玉と交換したい場合、ビー玉が「商品」であることがわかります。

別の例は、「「顧客」は当社から何かを購入する誰かである。」という記述です。これはよい記述ではありません。同社が他の会社にも製品を販売している場合、「誰か」という語は簡単に誤解されます。また、会社は、すでに自社から何かを購入した人ではなく、潜在的な「顧客」を追跡したいと考えています。さらに製品、サービス、または両者の組み合わせのどれを販売するかを記述するために、「何か」をより完全に定義することもできます。

業務の例

定義するものの典型的な業務例を用意することはよいことです。これは、よい例は読者による定義の理解を助けるのに大きな効果が期待できるからです。ピーナッツとビー玉に関するコメントは、少し幼稚に見えますが、「顧客」の概念を理解するために役立ちます。定義では、値を持っていることを述べています。例は、値が常に金額ではないことを示すのに役立ちます。

コメント

エンティティや属性の定義を担当するのは誰で、誰から情報を引き出し、現在どの状態にあり、定義の中で最後に変更されたのはいつか、などの一般的なコメントを入れることもできます。エンティティによっては、そのエンティティと関連するエンティティまたはエンティティ名がどのように異なるのかも説明する必要があります。たとえば、「顧客」と「見込客」はおそらく区別できます。

## 定義の参照と循環性

辞書を開くと、次のような状況に出くわすことがあります。

- 用語 1 の定義は用語 2 への参照を含んでいるか、用語 2 に基づいている。
- 用語 2 の定義は用語 3 への参照を含んでいるか、用語 3 に基づいている。
- 用語 3 の定義は用語 1 への参照を含んでいるか、用語 1 に基づいている。

個々の定義は問題ないように見えますが、まとめて見ると「循環」しています。エンティティと属性の定義では、注意しないとこのようなことが起こる可能性があります。以下に例を示します。

- 「顧客」: 当社の「製品」を 1 つまたは複数購入する誰か
- 「製品」: 当社が「顧客」に販売する何か

データ モデル内でエンティティや属性を定義するときは、このような循環参照を避けることが重要です。

## 業務用語集の作成

エンティティや属性を定義するときは、共通的な業務用語を使用すると便利な場合がよくあります。たとえば、「「通貨スワップ」とは、ある期間に渡って 2 つの異なる「通貨」のキャッシュ フローを交換することに同意する 2 つの「当事者」間の複数契約である。交換は、スワップの期間が固定されるか、変動する。スワップは、ヘッジ通貨と金利リスクによく使用される。」という例について考えてみます。

この例では、エンティティや属性定義の中で定義された用語を強調表示しています。このようなスタイルを使用することによって、読者は必要に応じて用語を見つけることができるので、用語を使用するたびに定義を行う必要がなくなります。

エンティティ名や属性名でないような用語（たとえば、共通業務用語）を使用する方が便利な場合、その基本定義を用意し、エンティティや属性の定義と同様に、その定義を参照するとよいでしょう。よく使われる用語の用語集（モデルとは別）を使用することができます。このような共通業務用語は、上の例では太字斜体で強調表示されています。.

このような方法は、一見すると定義間を何度も行き来しかねないように見えることもあります。もう 1 つの方法は、各用語をそれが使用されるたびに完全に定義することです。このような内部定義があちこちにある場合、定義する場所が増え、すべてが同時に変更される可能性が非常に小さくなります。

共通する業務用語集を作ることには、いくつかの意味があります。用語集は、モデリング定義で使用される基礎となり、それ自身が業務における人々の意思疎通にとって大きな役割を果たします。

# 属性定義

エンティティと同様に、すべての属性を明確に定義することが重要です。同様の規則が適用されます。属性を定義と比較することによって、それが合致するかを判断できます。しかし、「「口座」が開かれた日付」として定義される「口座開設日（account－open－date）」のようなものには注意が必要です。明確で完全な定義にするために、「開設された」の意味を詳しく定義する必要があります。

一般に属性定義はエンティティ定義と同じ基本構造を持っており、記述、例、コメントを含んでいます。定義には、必要に応じてどの事実を属性に関する有効値として受け入れるかを指定する規則も含める必要があります。

バリデーション ルールは、属性が取ることができる値の集まりで、受け入れ可能な値の範囲を制約または制限します。これらの値は、抽象的な意味と業務上の意味の両方を持っています。たとえば、「個人名」が「個人」によって選択された住所の好ましい形式として定義された場合、すべての文字列の集まりに制限されます。属性定義として、属性について任意のバリデーション ルールまたは有効値を定義できます。また、ERwin ではドメインを使ってこれらのバリデーション ルールを属性に割り当てます。ERwin は、ドメインとして文字列、数値、日付/時刻、BLOB をサポートしています。

コードや識別子、数量といった属性の定義は、よい業務の例には適さないことがあります。よって通常は、属性のバリデーション ルールまたは有効値の説明を含む必要があります。バリデーション ルールを定義する場合、属性が取りうる値を一覧するだけではなく、もっと詳しく説明するのはよい習慣です。「顧客状況」という属性を定義してみます。

**「顧客状況」:** 「顧客」と当社の業務における関係を記述するためのコード。有効値: *A*、*P*、*F*、*N*

バリデーション ルールの指定は、コードの意味を定義するわけではないので、あまり役には立ちません。以下のような値の表またはリストを使ってバリデーション ルールを使うと、もっとよく説明できます。

| 有効値 | 意味 |
|---|---|
| A: 現在の顧客（Active） | 現在当社と購買関係にある「顧客」。 |
| P: 見込み客（Prospect） | 当社は関係構築に興味を持っているが、現在購買関係にない誰か。 |
| F: 過去の顧客（Former） | 「顧客」リレーションシップは消滅している。つまり、過去24か月間に販売実績がない。 |
| N: 業務リレーションシップを持たない（No Business accepted） | 業務関係を持たないことに決めた「顧客」。 |

# ロール名

外部キーがリレーションシップを通じて子エンティティに移行される場合、子エンティティで外部キー属性がどのように使用されるかを説明するための定義を新たに行うか、または、定義を変更する必要があります。これは、同じ属性が同じエンティティに対して複数回移行される場合です。これらの重複した属性は同じに見えますが、その目的が異なるので、同じ定義を使用すべきではありません。

図 5-1 に示す例を考えてみます。「外国為替売買」エンティティと「通貨」エンティティの間には、2 つのリレーションシップが指定されています。

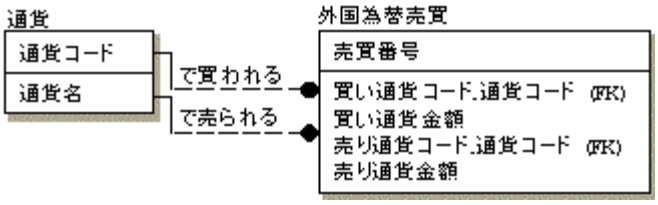



図 5-1

「通貨」エンティティの主キーは「通貨コード」です（これは、対象となる有効な「通貨」を追跡する上での識別子です）。「外国為替売買」によって、ある「通貨」が「買われ」、別の「通貨」が「売られ」ているのがわかります。

「通貨」エンティティの識別子（「通貨コード」）はまた、それぞれの「通貨」を識別するために使用されます。買い通貨の識別子は「買い通貨コード」で、売り通貨の識別子は「売り通貨コード」です。ただし、両方の「通貨コード」は同じものではありません。

同じ「通貨」を売買するのは不自然です。したがって、この「外国為替売買」では、「買い通貨コード」と「売り通貨コード」は異なります。2 つのロール名に異なる定義を指定することで、2 つの通貨コード間の違いを示すことができます。

| 属性/ロール名 | 属性定義 |
|---|---|
| 「通貨コード」 | 「通貨」の一意識別子 |
| 「買い通貨コード」 | 「外国為替売買」によって買われた「通貨」の識別子（「通貨コード」） |
| 「売り通貨コード」 | 「外国為替売買」によって売られた「通貨」の識別子（「通貨コード」） |

購入コードと販売コードの定義とバリデーションは「通貨」コードに基づいています。「通貨コード」はベース属性と呼ばれます。

IDEF1X の標準では、同じ名前を持つ 2 つの属性は同じベース属性からエンティティへ移行する場合、属性を一意化しなければなりません。一意化の結果は、2 つのリレーションシップによって移行された 1 つの属性です。また、IDEF1X の標準によって AllFusion ERwin DM は外部キー属性を自動的に統一します。移行された属性を統一したくない場合、AllFusion ERwin DM のリレーションシップ エディタでリレーションシップに名前を付ける際、属性にロール名を付けることができます。

# 定義とビジネス ルール

ビジネス ルールについては、データ モデル全体の一部としてすでに説明しました。これらのルールは、リレーションシップ、ロール名、候補キー、デフォルト、汎化カテゴリ、参照整合性、カーディナリティを含むまだ調べられていない他のモデリング構造という形式を取ります。また、ビジネス ルールはエンティティや属性の定義とバリデーション ルールでも捕捉されます。

たとえば、図 5-1 の「通貨」エンティティは、世界中どこでも認められているすべての有効な通貨のセットか、会社が毎日の業務で使用することを決めている先の通貨のサブセットとして定義できます。この区別は微妙ですが重要です。後者の場合、ビジネス ルール、すなわちポリシー ステートメントが必要です。

このルールは、「通貨コード」に関するバリデーション ルールの中で明らかになります。これは、「通貨コード」の有効値を業務で使用される値に制限します。ビジネス ルールの保守は、「通貨」の有効値のテーブルを保守する作業になります。「通貨」の交換を許可または禁止するには、有効値テーブルの中のインスタンスを作成するか、または削除するかだけで済みます。

「買い通貨コード」と「売り通貨コード」は同様に制限されます。「買い通貨コード」と「売り通貨コード」は異なる必要がある、というバリデーション ルールでさらに制限されています。そのため実際の使用では、お互いに他方の値に依存しています。AllFusion ERwin DM を使うと、バリデーション ツールを属性の定義で取り扱うことができ、バリデーション ルール、デフォルト値、有効値のリストを使って明示的に定義できます。

Chapter

6

# リレーションシップの詳細定義

リレーションシップは、最初思うよりも少し複雑です。リレーションシップには大量の情報が含まれています。リレーションシップはデータ モデルの心臓部であるという人もいます。なぜなら、リレーションシップはビジネス ルールを説明し、インスタンスの作成、変更、削除を制約しているからです。

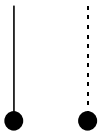
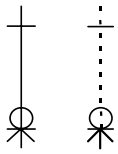
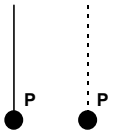
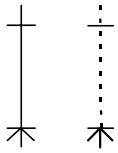
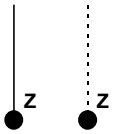
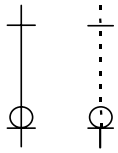
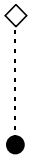
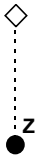
たとえば、カーディナリティを使うと、リレーションシップ内の子エンティティと親エンティティの両方に含まれるインスタンスの数を正確に定義できます。さらに参照整合ルールを使用すると、INSERT、UPDATE、DELETE などのデータベース操作をどのように取り扱うかを詳しく指定できます。

データ モデリングは、業務の専門家とシステムの専門家の両者が理解できるような論理データ モデルの構築を可能にする非常に複雑なリレーションシップもサポートしています。

## リレーションシップ カーディナリティ

これまでは、「多」という語によって意味される情報を取り込むことなく、論理モデルにおける 1 対多(One-to-Many)リレーションシップについて説明してきました。「多」という概念は、与えられた親に接続された子のインスタンスが複数なければならないという意味ではありません。1 対多(One-to-Many)における「多」の本当の意味は、親と対となる子のインスタンスが 0 個、1 個、または 2 個以上あるということです。

カーディナリティは、親テーブル内の対応するインスタンスに対して、子テーブルにあるインスタンスの数を正確に定義するリレーショナル プロパティです。IDEF1X と IE では、カーディナリティの指定に使われる記号が異なります。ただし、どちらの手法も、以下の表で示すように、1 以上、0 以上、0 または 1、または絶対数 N 個を示す記号があります。

| カーディナリティ記述 | IDEF1X 表記 | | IE 表記 | |
|---|---|---|---|---|
| | 依存型 | 非依存型 | 依存型 | 非依存型 |
| 「1」対「0、または 1 以上」 |  | |  | |
| 「1」対「1 以上」 |  | |  | |
| 「1」対「0 または 1」 |  | |  | |
| 「0 または 1」対「0、または 1 以上」<br>(非依存型のみ) | |  | |  |
| 「0 または 1」対「0 または 1」<br>(非依存型のみ) | |  | |  |

カーディナリティを使うと、リレーションシップに適用される追加のビジネス ルールを指定できます。図 6-1 では、外部キー「映画番号」と代理キー「ビデオテープ番号」の両方に基づいて各「ビデオテープ」を識別することに決めました。さらに、それぞれの「映画」は 1 本以上の「ビデオテープ」としてコピーできます。また、対応する「映画」がないかぎり「ビデオテープ」は存在できないことを示すリレーションシップも記述されています。

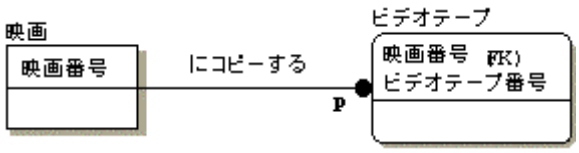



**図 6-1 1 対多（One-to-Many）依存型リレーションシップにおけるカーディナリティ**

「映画」－「ビデオテープ」モデルでは、リレーションシップのカーディナリティも指定しています。リレーションシップ ラインは、1 つの「映画」だけがリレーションシップに参加していることを示しています。これは、「映画」がリレーションシップにおける親だからです。

「ビデオテープ」をリレーションシップにおける子（IDEF1X ではドットで示します）にすることによって、「ビデオテープ」はいくつかある貸出用映画タイトルの 1 つとして定義されます。また、「映画」をデータベースに入れるには、「ビデオテープ」が少なくとも 1 本なければなりません。これによって、<としてレンタル可能である>リレーションシップのカーディナリティが「1 対多（One－to－One or More）」になります。ドットの横の「P」記号は「多（One or More）」というカーディナリティを表しています。ビデオテープとして存在しない「映画」は、このデータベースでは許されません。

反対に、このビデオ ショップでは、世界中のすべての「映画」についての情報を登録しています。ビデオテープを持たない「映画」についても同様です。このビデオ ショップのビジネス ルールは、「「映画」が存在する（情報システムに記録する）には、ビデオテープが 0、1 またはそれ以上なければならない」です。このビジネス ルールを記録するには、「P」を取り除きます。ダイアグラムにカーディナリティが明示的に示されていない場合、カーディナリティは「1 対多（One－to－Zero, One or More）」になります。

## 非依存型リレーションシップでのカーディナリティ

非依存型リレーションシップの場合も、親エンティティと子エンティティを線で結びます。ただし定義では、一部（またはすべて）のキーが、子のキーの一部になりません。これは、子が親の識別依存にならないことを意味します。また、リレーションシップの「多」側に親のないエンティティが存在するような（存在依存ではない）状況が生じます。

リレーションシップが子からみて必須の場合、子は親に対して存在依存です。リレーションシップがオプションの場合、子は、そのリレーションシップについて存在依存でも識別依存でもありません（他のリレーションシップではおそらく存在します）。IDEF1X では、オプションを示すのにダイアモンド記号を使用します。IE では、リレーションシップ ラインの親側に丸をつけます。

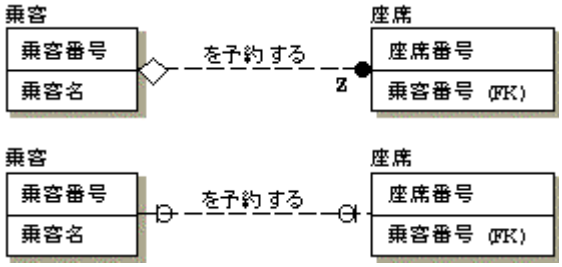



**図 6-2 1 対多（One－to－Zero, One or More）非依存型リレーションシップのカーディナリティ－IDEF1X（上）と IE（下）**

図 6-2 では、「乗客番号」属性は「座席」の外部キー属性となります。「乗客番号」は「座席」を識別するのではなく、「座席」を占める「乗客」を識別するので、リレーションシップは非依存型になります。「乗客」がいなくても「座席」は存在できるので、リレーションシップはオプションであることもわかります。リレーションシップがオプションの場合、ダイアグラムでは IDEF1X 表記ではダイアモンド、IE 表記では丸が使われます。オプションでない場合、非依存型リレーションシップのカーディナリティは識別リレーションシップと同じになります。

リレーションシップのカーディナリティは、IDEF1X では「Z」、IE では線で表します。これは、「乗客」が 0 または 1 つの「座席」を<予約できる>ことを示します。「座席」が予約されている場合、座席を予約している「乗客」は「乗客番号」によって識別されます。「座席」が予約されていない場合、「乗客番号」属性は空（NULL）になります。

# 参照整合性

リレーショナル データベースは、データ値を使ってリレーションシップを実現しているので、キー フィールドにあるデータの一貫性が非常に重要です。たとえば、親テーブルの主キー カラムにある値を変更する場合、そのカラムが外部キーとして存在するすべての子テーブルで同じ変更が行われることを覚悟しなければなりません。外部キーに適用される操作は業務で定義されたルールによって異なります。

たとえば、複数のプロジェクトを管理する業務では、図 6-3 のようなモデルで従業員とプロジェクトを追跡するかもしれません。「プロジェクト」と「プロジェクト メンバ」の間のリレーションシップがすでに識別になっており、「プロジェクト」の主キーが「プロジェクト メンバ」の主キーの一部となります。

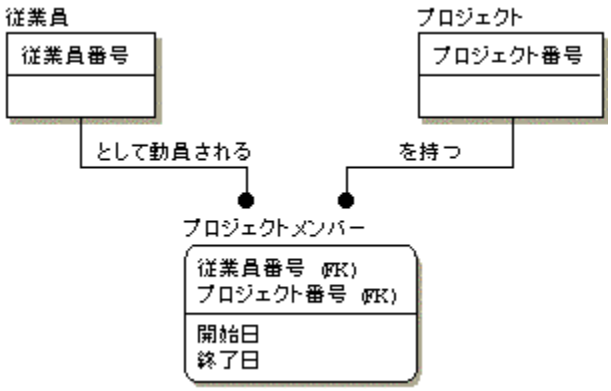



図 6-3 プロジェクトー従業員モデル

さらに「プロジェクト メンバ」の各インスタンスについて、「プロジェクト」インスタンスが 1 つだけあります。これは、「プロジェクト メンバ」が「プロジェクト」に対して存在依存であることを意味します。

「プロジェクト」インスタンスを削除したらどうなるでしょうか？

「プロジェクト」が削除されたときに「プロジェクト メンバ」のインスタンスを追跡しない場合、「プロジェクト」から一部のキーを継承する「プロジェクト メンバ」のすべてのインスタンスも削除する必要があります。

親キーが削除されたときに行われる操作を指定するルールを参照整合性と呼びます。リレーションシップの中でこの操作について選択された参照整合性オプションが CASCADE です。「プロジェクト」インスタンスが削除されるたびに、「プロジェクト メンバ」テーブルに対する CASCADE が削除され、「プロジェクト メンバ」内の関連するすべてのインスタンスも削除されます。

参照整合性の操作オプションには CASCADE のほかに、RESTRICT、SET NULL、SET DEFAULT があります。各オプションを以下のとおり説明します。

- **CASCADE** - 親エンティティのインスタンスを削除するたびに、子エンティティ内の関連するインスタンスも削除しなければなりません。
- **RESTRICT** - 子エンティティに関連するインスタンスが 1 つまたは複数ある場合、親エンティティ内のインスタンスの削除が禁止されるか、親エンティティに関連するインスタンスが子エンティティ内にある場合、子エンティティ内のそのインスタンスの削除が禁止されます。
- **SET NULL** - 親エンティティ内のインスタンスが削除されるたびに、子エンティティ内の関連するインスタンスにあるキー属性が NULL に設定されます。
- **SET DEFAULT** - 親エンティティ内のインスタンスが削除されるたびに、子エンティティ内の関連するインスタンスにある外部キー属性が指定されたデフォルト値に設定されます。

- **<None>** - 参照整合性の操作は必要ありません。すべての操作に参照整合ルールが関連付けられている必要はありません。たとえば、子エンティティ内のインスタンスを削除するときに、参照整合性が必要ない場合があります。これは、カーディナリティが 1 対多（Zero、One－to－Zero、One or More）の場合に有効なビジネス ルールです。これは、親エンティティに関連するインスタンスがなくても子カーディナリティにインスタンスが存在できるからです。

参照整合性は IDEF1X や IE では正式に定義されていませんが、実際に完成したデータベースの動作を示すビジネス ルールを捕捉するため、データ モデリングの重要な要素の一部であると言えます。このため AllFusion ERwin DM には、参照整合ルールを取り込んだり、表示したりする両方の手法があります。

参照整合性が定義されたら、進行役またはアナリストは、業務上の決定事項の結果を示す各種プランについて質問したり、作成したりして、ビジネス ユーザが定義した参照整合性ルールを試してみる必要があります。要件を定義し、完全に理解したうえで、ファシリテータまたはアナリストは、RESTRICT や CASCADE など、具体的な参照整合性アクションを推奨することができます。

## 参照整合性オプションの読み取り

参照整合ルールは、以下の要因によって変化します。

- エンティティがリレーションシップにおける親と子のどちらであるか
- 実現されるデータベース操作

このため各リレーションシップには、参照整合性が定義可能な次の 6 つの操作が考えられます。

- PARENT INSERT
- PARENT UPDATE
- PARENT DELETE
- CHILD INSERT
- CHILD UPDATE
- CHILD DELETE

図 6-4 は、プロジェクトー従業員モデルの参照整合ルールを示しています。

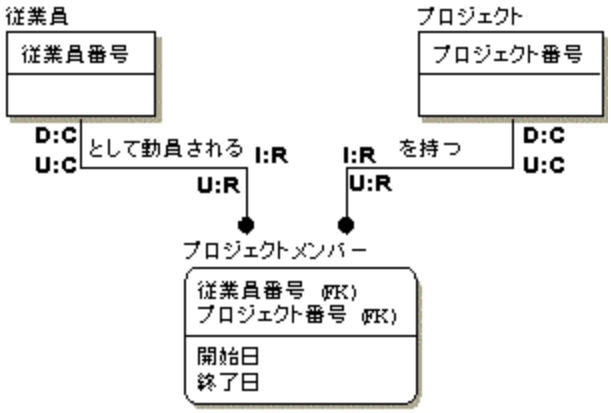



**図 6-4 参照整合性の例**

図 6-4 で捕捉された参照整合ルールは、「プロジェクト」エンティティ内のすべての削除を「プロジェクト メンバ」エンティティにカスケード（連鎖）するという業務意思決定を示しています。このルールは PARENT DELETE CASCADE と呼ばれ、図 6-4 では、指定されたリレーションシップの親側にある「D:C」という文字によって示されます。参照整合性記号の 1 番目の文字は常にデータベース操作（I－挿入、U－更新、または D－削除）を示します。2 番目の文字は参照整合性オプション（C－CASCADE、R－RESTRICT、SN－SET NULL、SD－SET DEFAULT）を示します。

図 6-4 では、PARENT INSERT の参照整合性オプションは指定されていないため、挿入（I:）の参照整合性は図にはありません。

## 参照整合性、カーディナリティ、依存型リレーションシップ

図 6-4 では、「プロジェクト」と「プロジェクト メンバ」の間のリレーションシップが依存型です。したがって、リレーションシップの親エンティティである「プロジェクト」の参照整合性について有効なオプションは CASCADE と RESTRICT です。

CASCADE では、「プロジェクト」インスタンスの削除によって影響を受ける「プロジェクト メンバ」のすべてのインスタンスも削除されます。RESTRICT では、「プロジェクト」のキーを継承した「プロジェクト メンバ」のすべてのインスタンスが削除されるまで、「プロジェクト」を削除できません。すべてが削除されていない場合、「削除」は制限されます。

削除を禁止（RESTRICT）するのはなぜでしょうか？これには、「プロジェクトー従業員」に関する他の事実（プロジェクトの「開始日」など）を知りたい場合などがあります。削除をカスケード（CASCADE）した場合、この補助情報は失われます。

親エンティティのインスタンスを更新する場合、更新された情報は子エンティティの関連するインスタンスもカスケードされます。

例を見るとわかるように、子エンティティ内でインスタンスを挿入、更新、または削除するときに適用されるルールはそれぞれ異なります。たとえば、インスタンスを挿入する場合、操作は RESTRICT に設定されます。この操作は、子エンティティの横の「I:R」によって示されます。これは、参照される外部キーが親エンティティに存在するインスタンスと一致する場合のみ子エンティティにインスタンスを追加できることを意味します。したがって、キー フィールドの値が「プロジェクト」エンティティのキー値と一致する場合のみ、「プロジェクト メンバ」に新しいインスタンスを挿入できます。

## 参照整合性、カーディナリティ、非依存型リレーションシップ

「プロジェクト メンバ」が「プロジェクト」に対して存在依存または識別依存の場合、「プロジェクト」と「プロジェクト メンバ」の間のリレーションシップをオプションの非依存型に変更することができます。このようなリレーションシップでは、参照整合性のオプションが大きく異なります。次の図 6-5 を参照してください。

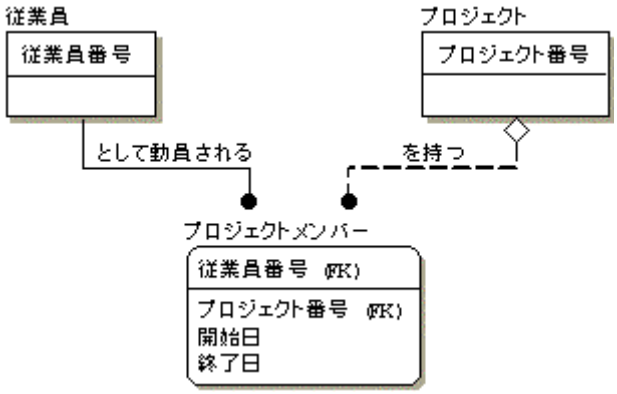



図 6-5 非依存型リレーションシップの参照整合性

非依存型リレーションシップによって移行される外部キーは NULL を設定することが可能なので、PARENT DELETE について指定できる参照整合性オプションに「SET NULL」があります。SET NULL は、「プロジェクト」のインスタンスが削除された場合、「プロジェクト メンバ」の関連するインスタンスで「プロジェクト」から継承されたすべての外部キーを NULL に設定する必要があります。この「削除」は前の例のようにカスケードされず、(RESTRICT のように) 禁止されません。このアプローチの利点は、「プロジェクト メンバ」に関する情報を失うことなく、「プロジェクト メンバ」と「プロジェクト」の間の接続を効果的に切断できることです。

CASCADE または SET NULL の使用は、外部キーによって表されるリレーションシップの履歴的な知識を保つという業務意思決定を反映しています。

# その他のリレーションシップ タイプ

リレーションシップは、子エンティティが親エンティティに依存するかどうか、親エンティティと子エンティティにインスタンスがいくつあるかを定義します。論理モデルを作成する際、これまでに説明した標準の 1 対多(One－to－Many)リレーションシップに当てはまらないリレーションシップが見つかることがあります。これには、次のようなリレーションシップがあります。

- **多対多(Many-to-many)リレーションシップ** - 「1 番目のエンティティが 2 番目のエンティティの複数インスタンスを<所有する>、かつ 2 番目のエンティティが 1 番目のエンティティの複数インスタンスを<所有する>」というリレーションシップ。たとえば、「「従業員」は 1 つまたは複数の「役職」を<持ち>、かつ「役職」には 1 人または複数の「従業員」が<割り当てられる>」といったリレーションシップ。
- **多項リレーションシップ** - 2 つのエンティティ間の単純な 1 対多(One－to－Many)リレーションシップは 2 項リレーションシップと呼びます。複数の親と 1 つの子エンティティ間の 1 対多(One－to－Many)リレーションシップを多項リレーションシップと呼びます。
- **再帰リレーションシップ** - エンティティが自分自身へのリレーションシップを持っている場合、そのエンティティは再帰リレーションシップとなります。たとえば、「従業員」エンティティの場合、「1 人の「従業員」が 1 人または複数の「従業員」を<管理する>」ことを示すリレーションシップが考えられます。このようなリレーションシップは、部品表などで部品間のリレーションシップを示すためにも使用されます。
- **サブタイプ リレーションシップ** - 関連するエンティティをまとめて、共通する属性は 1 つのエンティティ、共通しない属性は別の関連エンティティに分類します。たとえば、「従業員」エンティティは「正社員」と「パートタイム」というサブタイプに分類できます。

これらのリレーションシップ タイプについては、次の各項で説明します。

# 多対多(Many-to-Many)リレーションシップ

キー ベース モデルおよび全属性モデルでは、親エンティティにある 0 または 1 つのインスタンスが子エンティティにある特定のインスタンス セットと関連付けられていなければなりません。このルールによって、ERD または初期のモデリング フェーズで発見され、表現された多対多(Many－to－Many)リレーションシップは、1 対多(One－to－Many)リレーションシップに分解しなければなりません。

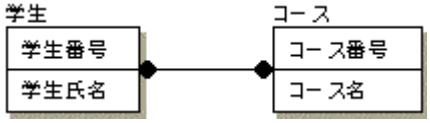



図 6-6 多対多(Many-to-Many)リレーションシップ

図 6-6 は、「学生」と「コース」の間の多対多（Many－to－Many）リレーションシップを示しています。「コース」と「学生」の間の多対多リレーションシップを削除しないと、たとえば「コース」のキーが「学生」のキーに含まれ、「学生」のキーが「コース」のキーに含まれてしまいます。しかし、「コース」は自分自身のキーによって識別され、「学生」も自分自身のキーによって識別され、エンドレス ループになってしまいます。

多対多（Many－to－Many）リレーションシップは、関連エンティティを作成することによって解消できます。次の図 6-7 では、「コース登録簿」エンティティを追加することで、「学生」と「コース」の間の多対多リレーションシップが解決されています。

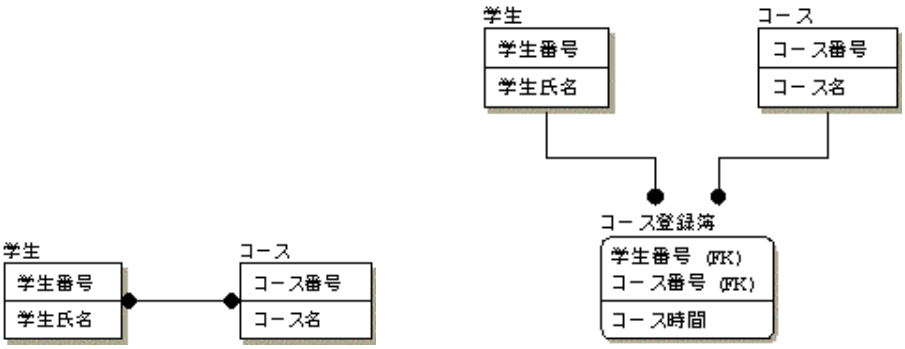



**図 6-7 関連エンティティを使った「学生－コース」リレーションシップの確定**

「コース登録簿」は関連エンティティです。これは、2 つの関連するエンティティ間の関連性の定義に使用されていることを示します。

多対多（Many－to－Many）リレーションシップでは、しばしば意味が隠れてしまう場合があります。多対多（Many－to－Many）リレーションシップを含んだダイアグラムでは、「学生」が複数の「コース」を登録しますが、どのように登録したかに関する情報は含まれていません。多対多（Many－to－Many）リレーションシップを確定すると、エンティティの関係だけではなく、リレーションシップに関する事実も説明する「コース時間」などのその他の情報も明らかになります。

多対多（Many－to－Many）リレーションシップを確定すると、構造をチェックするリレーションシップ動詞句が必要であることに気付きます。動詞句を含める方法は 2 つあります。つまり、新しい動詞句を作成する方法と、多対多（Many－to－Many）リレーションシップに対して存在する動詞句を使用する方法です。最も簡単な方法は、関連エンティティによって多対多（Many－to－Many）リレーションシップを読み取ることです。つまり、「1 人の「学生」が複数の「コース」に<登録する>、かつ 1 つの「コース」が複数の「学生」によって<履修される>」と読み取ることができます。モデル作成者は、このスタイルを使ってモデルの構築と読み取りを行います。

もう 1 つ正しく、しかし厄介なスタイルがあります。モデルの構造はまったく同じですが、動詞句は異なり、モデルを読み取る方法が少し異なります。図 6-8 では、「「学生」が 1 つまたは複数の「コース登録簿」<に記録されたコースを登録する>、かつ「コース」は 1 つまたは複数の「コース登録簿」<に記録された学生によって履修される>」と読み取れます。

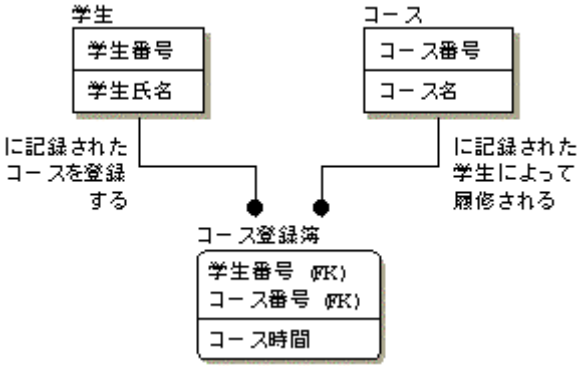



図 6-8 関連エンティティによるリレーションシップを読み取る

動詞句が長くなりますが、親エンティティから子への直接的な標準パターン読み取りになります。

どちらのスタイルを選ぶ場合も一貫性が重要です。多対多(Many－to－Many)リレーションシップのための動詞句を記録する方法を決めることは、上の例のようにシンプルな構造ではあまり難しくありません。しかし、関連エンティティのいずれかの側がそれ自身関連エンティティである(他の多対多(Many－to－Many)リレーションシップを表すためにある)場合など、複雑な構造の場合は難しくなることがあります。

# 多項リレーションシップ

1 つの親子リレーションシップがある場合、そのリレーションシップは 2 項リレーションシップと呼ばれます。これまでのリレーションシップに関するすべての例は 2 項リレーションシップです。しかし、データ モデルを作成する際、複数の親エンティティと 1 つの子テーブル間のリレーションシップに関するモデリング名である多項リレーションシップが見つかることは珍しくありません。多項リレーションシップの例を次の図 6-9 に示します。

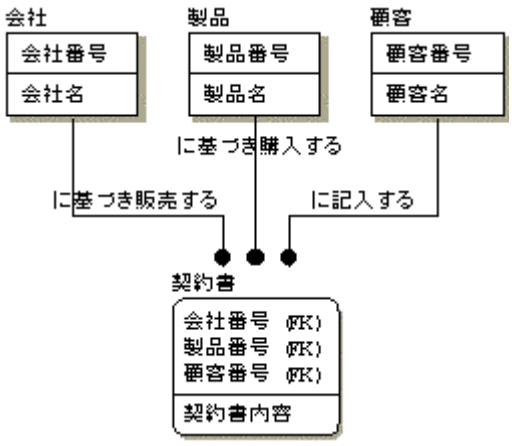



図 6-9 多項リレーションシップ

多対多（Many－to－Many）リレーションシップと同様に 3 項、4 項、または N 項のリレーションシップは、エンティティ リレーションシップ ダイアグラムの中の有効な構造体です。また、多対多（Many－to－Many）リレーションシップと同様に、多項リレーションシップは、あとのモデルで 2 項リレーションシップのセットを使って関連エンティティに確定されます。

図 6-9 で示したビジネス ルールを見ると、「契約書」は「会社」、「製品」、「顧客」の間の 3 方向のリレーションシップを表していることがわかります。この構造は、複数の「会社」が複数の「製品」を複数の「顧客」に販売していることを示しています。しかし、このようなリレーションシップを見ると、業務上の疑問が生じてきます。たとえば「製品を販売する前にこれを会社が買い付ける必要はあるか？」「顧客は複数企業からの製品が含まれた単一の契約を結ぶことができるか？」「どの顧客がどの企業に『所属している』かについて追跡する必要があるか？」これらの問いに対する答えによって、構造が変わります。

仮に、「製品を販売する前にこれを会社が買い付ける必要はあるか？」に対する答えが「はい」である場合、次の図 6-10 に示すような構造になります。

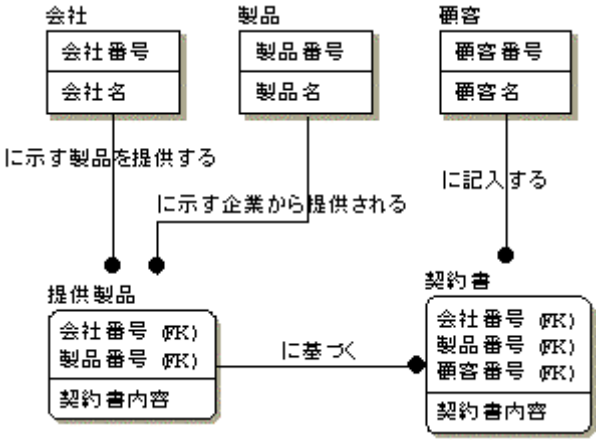



図 6-10 多項リレーションシップの確定

「製品」は「会社」が買い付けなければならないので、このリレーションシップを取り込むには関連エンティティを作成できます。「契約書」に対するもとの 3 項リレーションシップは 2 項リレーションシップに置き換えられます。

さまざまな業務上の問いをすることによって、ほとんどの多項リレーションシップが関連エンティティに対する一連のリレーションシップに分割できることがわかります。

# 再帰リレーションシップ

エンティティが親と子の両方を兼ねる場合、このエンティティは再帰的リレーションシップとなります。このリレーションシップは、部品表を実現するために再帰リレーションシップを使用する IMS や IDMS などの従来の DBMS にもともと格納されていたデータをモデリングするときに非常に重要です。

たとえば、「会社」は他の「会社」の親になることができます。すべての非依存リレーションシップと同様に、親エンティティのキーは子エンティティのデータ領域に現れます。次の図 6-11 を参照してください。

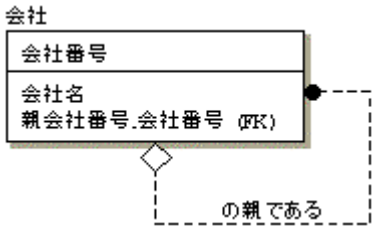



**図 6-11 再帰リレーションシップの例**

「会社」に関する再帰リレーションシップには、「会社」に親がない場合など、外部キーが NULL になれることを示すダイアモンド記号があります。再帰リレーションシップは、オプション(ダイアモンド)で非依存型を指定します。

「会社番号」属性は再帰リレーションシップによって移行し、例ではロール名「親会社番号」で表示されています。これには 2 つの理由があります。第 1 に一般的設計ルールとして、同じエンティティ内に同じ名前の属性は複数存在できません。したがって、再帰リレーションシップを完成させるには、移行された属性のロール名を用意しなければなりません。

第 2 に、キーにある「会社番号」属性(「会社」の各インスタンスを識別する)は、リレーションシップによって移行した「会社番号」と同じではなく、親の「会社」を識別します。両方の属性に同じ定義を使用できないので、移行属性にはロール名を付けなければなりません。定義の例を次に示します。

- **会社番号** - 「会社」を一意に識別する。
- **親会社番号** - 親「会社」の「会社番号」。すべての「会社」が親「会社」を持つわけではない。

以下のようなサンプル インスタンス表を作成すると、妥当性を検証するためにリレーションシップにおけるルールをテストすることができます。

**会社**

| 会社番号 | 親会社番号 | 会社名 |
|---|---|---|
| C1 | NULL | Big Monster Company |
| C2 | C1 | Smaller Monster Company |
| C3 | C1 | Other Smaller Company |
| C4 | C2 | Big Subsidiary |
| C5 | C2 | Small Subsidiary |
| C6 | NULL | Independent Company |

上記のサンプル インスタンス表から、「Big Monster Company」は、「Smaller Monster Company」と「Other Smaller Company」の親であることがわかります。そして「Smaller Monster Company」は、「Big Subsidiary」と「Small Subsidiary」の親です。「Independent Company」は、これ自身が親ではなく、また親を持つこともありません。「Big Monster Company」も親を持ちません。この情報階層を図示してみれば、図 6-12 のようにサンプル インスタンス表の情報を検証することができます。

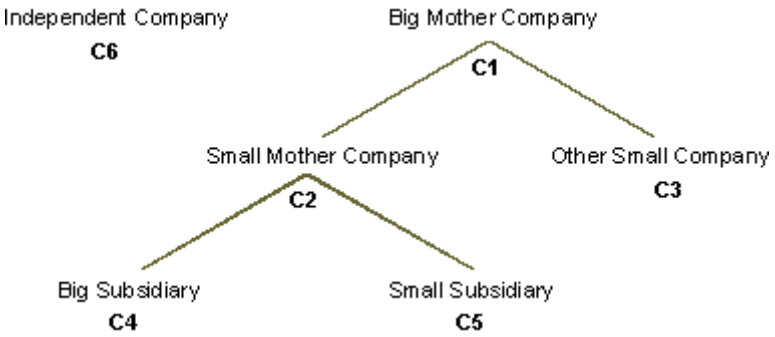



図 6-12 「会社」階層

# サブタイプ リレーションシップ

サブタイプ リレーションシップ（汎化カテゴリ、汎化階層、または継承階層とも呼ばれる）は、共通の特性を持つエンティティをグループ化するための方法です。たとえば、図 6-13 に示すようにモデリング作業で銀行に複数の「口座」（当座預金、普通預金、借入など）があることがわかったとします。

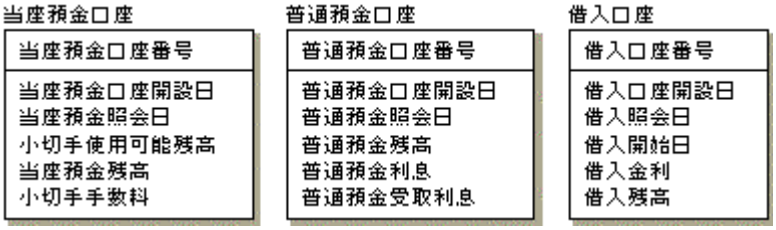



図 6-13　「口座」エンティティの例

別々の独立エンティティの間に類似性が認められる場合、3 種類の口座に共通の属性をまとめて階層構造にすることができます。

これらの共通の属性は、スーパータイプ エンティティ（または汎化エンティティ）と呼ばれるより高いレベルのエンティティに移動します。各口座固有の属性は、サブタイプ エンティティのままです。例では、3 種類の口座に共通の情報を表す「口座」というスーパータイプ エンティティを作成できます。サブタイプ「口座」には「口座番号」という主キーがあります。

サブタイプ リレーションシップを使って、「当座預金口座」、「普通預金口座」、「借入口座」の 3 つのサブタイプ エンティティが「口座」に関連付けられた依存エンティティとして追加されます。

これによって図 6-14 のような構造になります。

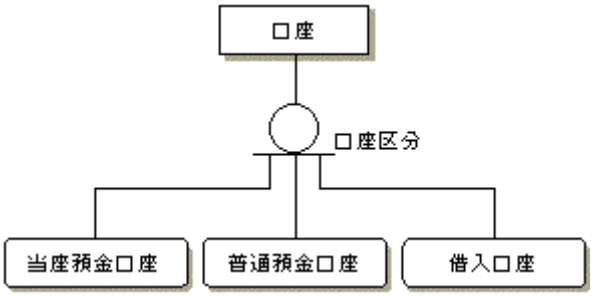



図 6-14　サブタイプ リレーションシップの例

図 6-14 で、「口座」は「当座預金口座」、「普通預金口座」、「借入口座」のいずれかです。各サブタイプ エンティティは「口座」の種類で、「口座」のプロパティを継承します。「口座」の 3 つのサブタイプは互いに排他的です。

「口座」の種類を区別するために、「口座区分」という属性をサブタイプ識別子として追加します。サブタイプ識別子は、スーパータイプ（「口座」）の属性で、その値によってどの「口座」かがわかります。

サブタイプ リレーションシップを確立したら、もとのモデルの各属性を順番に調べ、サブタイプ エンティティに残すか、スーパータイプに移動するかを判断できます。たとえば、各サブタイプ エンティティは「口座開設日」を持っています。これら 3 種類の「口座開設日」の定義が同じ場合、これらのスーパータイプに移動し、サブタイプ エンティティから取り除くことができます。

各属性は、サブタイプ エンティティに残すか、スーパータイプに移動するかを判断するために順番に分析する必要があります。1 つの属性がすべてではないがほとんどのサブタイプ エンティティにある場合、判断が難しくなります。属性をサブタイプ エンティティに残すことも、スーパータイプに移動することもできます。この属性がスーパータイプにある場合、スーパータイプにある属性が対応するサブタイプ エンティティにはないものの場合、値が NULL になります。

どちらの方法を選択するかは、一般に、いくつのサブタイプ エンティティが共通属性を共有しているかによります。ほとんどのサブタイプ エンティティで共有している場合は、より高レベルのモデルで属性をスーパータイプに移動することがよいでしょう。属性を共有しているサブタイプが少ない場合は、そのままにしておくのが最適です。なお、物理的なレベルのモデルでは、その目的にもよりますが、属性をサブタイプ エンティティにそのまま残す方が一般的です。

分析の結果、モデルは図 6-15 のようになりました。

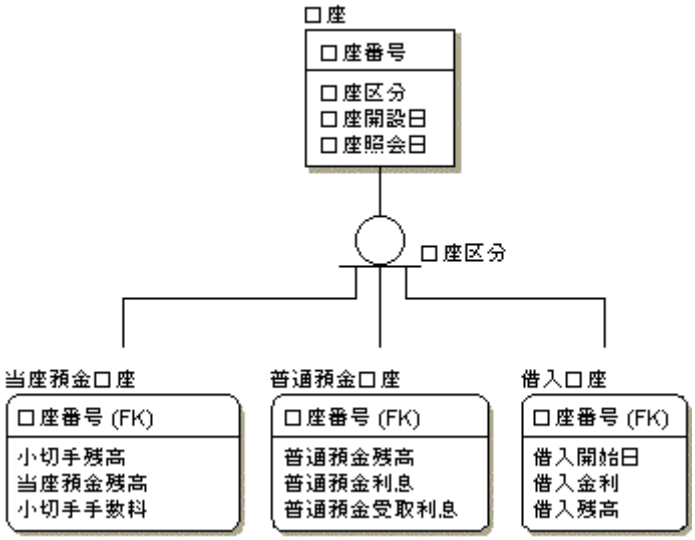



**図 6-15　「口座」サブタイプの例**

サブタイプ リレーションシップを作成する場合、スーパータイプの他のサブタイプには属さないサブタイプ レベルで適用する必要のあるようなビジネス ルールにも注意しなければなりません。たとえば、「借入口座」は支払い後に削除されます。同じ条件で「当座預金口座」や「普通口座」を削除することはめったにありません。

このリレーションシップは、階層のあるサブタイプに対しては意味があっても、別のサブタイプに対しては無意味です。たとえば、「借入口座」エンティティについては、顧客の支払いまたは資産の記録との前のリレーションシップが組織構造の違いによって失われていないかを確認する必要があります。

## 確定サブタイプ構造と未確定サブタイプ構造の比較

IDEF1X では、サブタイプ リレーションシップ内のサブタイプ エンティティ セットが確定しているかどうかを示すために各種の記号が使用されます。未確定サブタイプは、まだ見つかっていないサブタイプ エンティティがあるかもしれないとモデル作成者が考えていることを意味します。図 6-16 に示すように、未確定サブタイプはサブタイプ記号の下の 1 本線によって示されます。

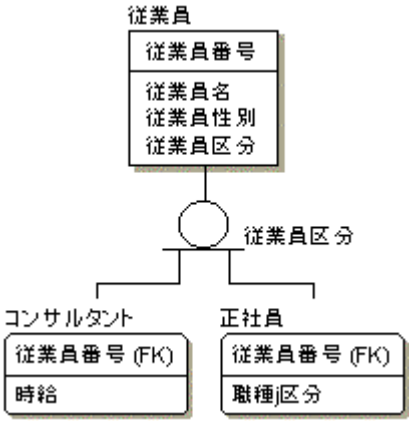



図 6-16　未確定サブタイプ

確定サブタイプは、サブタイプ構造が考えられるすべてのサブタイプ エンティティを含んでいるとモデル作成者が確信していることを意味します。たとえば、図 6-17 に示すように、確定サブタイプは男性従業員と女性従業員に固有の情報を取り込むことができます。確定サブタイプは、サブタイプ記号の下の 2 本線によって示されます。

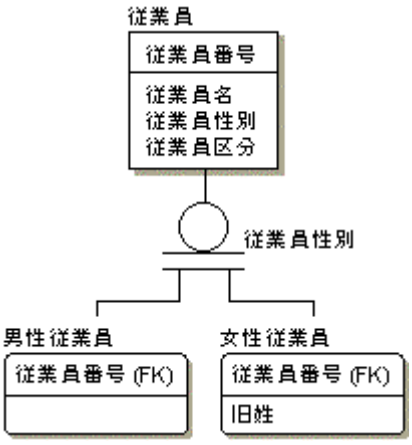



図 6-17 確定サブタイプ

サブタイプ リレーションシップを作成する場合、識別子のバリデーション ルールも作っておくとよいでしょう。これは、サブタイプの見落としを避けるのに役立ちます。たとえば、「口座区分」のバリデーション ルールは、C（当座預金口座）、S（普通預金口座）、L（借入口座）とします。口座区分「0」の古いデータがある場合、バリデーション ルールによって、表示されていない区分が見つかり、「0」が古いシステムの設計上の問題から生じたものか、見落としていた口座区分かが判断できます。

## 包括的および排他的リレーションシップ

IDEF1X とは異なり、IE の表記では確定サブタイプと未確定サブタイプのリレーションシップを区別しません。そのかわり IE では、リレーションシップが包括的か排他的かを表示します。

排他的サブタイプ リレーションシップでは、サブタイプの各インスタンスが 1 つのサブタイプとのみ関連付けられます。たとえば、従業員は正社員またはパートタイムのどちらかで、両方ではないというビジネス ルールをモデリングできます。モデルを作成するには、「正社員」と「パートタイム」のサブタイプ エンティティからなる「従業員」スーパータイプと、「従業員区分」という識別子属性を含めます。さらに識別子の有効値を、正社員を示す「F」とパートタイムを示す「P」に制限します。

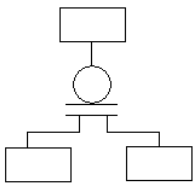
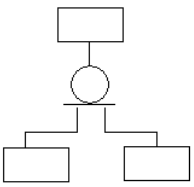
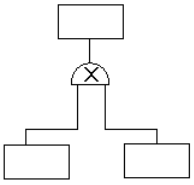
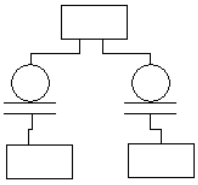
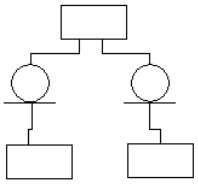
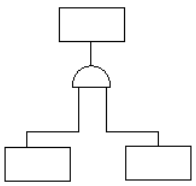
包括的サブタイプ リレーションシップでは、スーパータイプの各インスタンスが 1 つまたは複数のサブタイプと関連付けられます。上の例では、従業員は、正社員、パートタイム、またはその両方とビジネス ルールに記述します。この例では、識別子の有効値を「正社員」を示す「F」、パートタイムを示す「P」、両方を示す「B」に制限します。

**注:** IDEF1X の表記では、スーパータイプ エンティティと各サブタイプ エンティティの間のリレーションシップを別々に表示することによって、包含的サブタイプを表すことができます。

### IDEF1X と IE でのサブタイプ表記

以下の表は、IDEF1X と IE でのサブタイプ表記です。

**IDEF1X サブタイプ表記**

| | 確定型 | 未確定型 | IE サブタイプ表記 |
|---|---|---|---|
| 排他的 (Exclusive) |  |  |  |
| 包括的 (Inclusive) |  |  |  |

## どんなときにサブタイプ リレーションシップを作成するか

以上をまとめると、サブタイプ リレーションシップを作成するには次の 3 つの理由があります。

- エンティティが共通の属性セットを共有する。これは上の例の場合です。
- エンティティが共通のリレーションシップ セットを共有する。これについてはまだ説明していませんが、口座の構造を見ると、サブタイプ エンティティを持つ共通のリレーションシップの汎化親を基とする 1 つのリレーションシップにまとめることができます。たとえば、各口座区分が複数の「顧客」と関連付けられている場合、「口座」レベルに 1 つのリレーションシップを作り、個々のサブタイプ エントリにある別々のリレーションシップを取り除くことができます。
- 業務上要求されている場合、たとえサブタイプ エンティティが異なる属性を持っていなかったり、サブタイプ エンティティが他のサブタイプ エンティティとは異なるリレーションシップに参加していなくても、(意志疎通や理解の目的で)モデル内でサブタイプ エンティティを明示する必要があります。モデルの主要な目的の 1 つは、情報構造の意志疎通の支援であることを思い出してください。これにサブタイプ エンティティが役立つ場合、それを示します。

Chapter

7

# 正規化

正規化とは、リレーショナル データベースのために E. F. Codd によって概略が定義された設計ルールに準拠するデータベース設計を作成するプロセスです。正規化のルールに従うと、同じ事実を得る手段を複数提供するようなすべてのモデル構造を取り除くことによって、データの冗長性をコントロールおよび解消することができます。

正規化の目的は、事実を得る手段を 1 つだけにすることです。これは次の一言で表されます。

**「1 つの事実は 1 つの場所に！」**

正規化の原理を理解するために、この章では設計上共通する問題と正規化による解決のさまざまな例を示します。

## 正規形の概要

最も一般的な正規形の定義を以下に示します。

### 機能依存（FD: functional dependence）

エンティティ「E」内の属性グループ「A」の各値に「E」内の属性グループ「B」の一部でも依存する場合、「E」の属性「B」は「E」の属性「A」に機能依存するといいます。言い換えれば、「A」によって「B」が一意に識別されます。

### 完全機能依存（full functional dependence）

エンティティ「E」内の属性グループ「B」が、属性グループ「A」全体に依存する場合、「E」内の属性グループ「B」は「E」内の属性グループ「A」に完全機能依存するといいます。

### 第 1 正規形（1NF）

属性の値がもうこれ以上分割できない最小単位値だけを含む場合、エンティティ「E」は第 1 正規形です。繰り返し集団（COBOL データ構造などで見られる）は取り除かれていなければなりません。

### 第 2 正規形（2NF）

エンティティ「E」が第 1 正規で、かつ、すべての非キー属性が主キーに完全機能依存する場合、エンティティ「E」は第 2 正規です。つまり、部分的なキー依存はありません（依存は「E」のキー「K」全体に対してで、「K」の一部に対してではありません）。.

### 第 3 正規形（3NF）

エンティティ「E」が第 2 正規で、かつ、すべての「E」の非キー属性が他の非キー属性に依存しない場合、エンティティ「E」は第 3 正規です。第 3 正規を表現する方法は他にもあります。別の方法としては、エンティティ「E」が第 2 正規形で、かつ、すべての非キー属性が全面的に主キーに依存する場合、エンティティ「E」は第 3 正規形です。最後の定義としては、エンティティ「E」内のすべての属性が、エンティティ「E」（第 2 正規形）のすべてを表し、なおかつエンティティ「E」だけの情報を表す場合（エンティティすべてのキーとそのキーのみによって表される時）、エンティティ「E」は第 3 正規形です。第 3 正規形の実装方法を忘れないようにする手段の一つは、「各属性はキーにのみ依存している。したがって、Codd が定義したルールに従うべきである」という警句を尊重することです。

第 3 正規形に続いて、Boyce－Codd、第 4 正規形、第 5 正規形という 3 種類の正規形があります。実際には、第 3 正規形が標準です。なお、物理データベース設計では、トランザクションのパフォーマンスを優先して、通常は構造の非正規化が行われます。非正規化することによって構造に冗長性が生じますが、効果的にパフォーマンスを向上させることができます。

## 設計上共通する問題

設計上の多くの共通する問題は、正規形に違反した結果生じます。共通する問題には次のようなものがあります。

- データ グループの繰り返し
- 同じ属性の複数使用
- 同じ事実の複数出現
- 矛盾する事実
- 導出属性
- 欠落情報

以下の各項では、これらの問題を個々に見て行き、モデルやサンプル インスタンス表を使って説明します。設計問題を解消する際、サンプル インスタンス表の使用は、多くの正規化エラーを見つける上で不可欠です。

## データ グループの繰り返し

データ グループの繰り返しは、属性内のリスト、繰り返し要素、内部構造として定義できます。このような構造は、古いデータ構造ではよく使われていたとしても、第 1 正規形に違反し、RDBMS モデルでは取り除かなければなりません。これは、RDBMS では可変長の繰り返しフィールドを取り扱えないからです。RDBMS では、このような配列に添字を付けることができません。図 7-1 のエンティティには、「子供の名前」というデータ グループの繰り返しがあります。繰り返しデータ グループは、「エンティティの各属性がそれぞれのインスタンスに対して意味を 1 つだけ持っている場合、そのエンティティは正規形である」という第 1 正規形に違反しています。

図 7-1 のような繰り返しデータ グループは、実際のデータが入ったデータベースを定義するときに問題を起こします。たとえば「従業員」エンティティの設計後に、次のような疑問に直面します。「子供の名前は何人分を記録する必要があるか？」「名前用に、データベースの各行にどのくらいのスペースを残しておくべきか？」「確保してあるスペースよりも名前が多い場合は、どうしたらいいだろうか？」

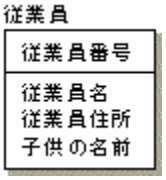



図 7-1 「従業員」エンティティ

次のサンプル インスタンス表によって問題が明らかになることがあります。

**従業員**

| 従業員番号 | 従業員名 | 従業員住所 | 子供の名前 |
|---|---|---|---|
| E1 | Tom | Berkeley | Jane |
| E2 | Don | Berkeley | Tom, Dick, Donna |
| E3 | Bob | Princeton | — |
| E4 | John | New York | Lisa |
| E5 | Carol | Berkeley | — |

設計を修正するには、「従業員」エンティティから子供の名前のリストを何らかの方法で取り除かなければなりません。1 つの方法は、図 7-2 に示すように、従業員の子供に関する情報を含む「子供」テーブルを追加することです。この場合、子供の名前を「子供」テーブルの 1 つのエントリとして表すことができます。従業員の物理レコード構造から見ると、領域割り当てに関する問題がある程度解決され、従業員に子供がいない場合のレコード構造の領域の無駄を防ぐことができ、家族がいる従業員のためにどの程度領域を割り当てるべきか迷うこともありません。

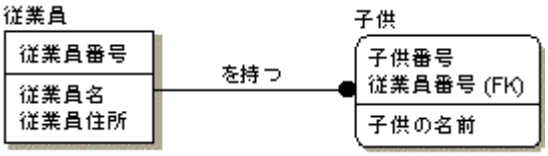



図 7-2 「子供」テーブルの追加

「従業員－子供」モデルのサンプル インスタンス表を次に示します。

**従業員**

| 従業員番号 | 従業員名 | 従業員住所 |
|---|---|---|
| E1 | Tom | Berkeley |
| E2 | Don | Berkeley |
| E3 | Bob | Princeton |
| E4 | Carol | Berkeley |

**子供**

| 従業員番号 | 子供番号 | 子供の名前 |
|---|---|---|
| E2 | C1 | Tom |
| E2 | C2 | Dick |
| E2 | C3 | Donna |
| E4 | C1 | Lisa |

この変更は正規化モデルへの第一歩（第 1 正規形への変換）です。両方のエンティティともフィールドが固定長になり、わかりやすく、プログラミングも容易になります。

## 同じ属性の複数使用

1 つの属性が 2 つの事実の一方を表し、どちらの事実を表しているのかがわからない場合も問題となります。たとえば、図 7-3 の「従業員」エンティティには「入社日または退職日」という属性があります。ここには、従業員についてこの情報を記録できます。

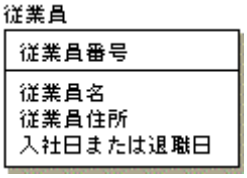



図 7-3 「入社日または退職日」という属性を持つ「従業員」エンティティ

「入社日または退職日」を示すサンプル インスタンス表を次に示します。

**従業員**

| 従業員番号 | 従業員名 | 従業員住所 | 入社日または退職日 |
|---|---|---|---|
| E1 | Tom | Berkeley | 1 月 10, 2004 |
| E2 | Don | Berkeley | 5 月 22, 2002 |
| E3 | Bob | Princeton | 3 月 15, 2003 |
| E4 | John | New York | 9 月 30, 2003 |
| E5 | Carol | Berkeley | 4 月 22, 2000 |
| E6 | George | Pittsburgh | 10 月 15, 2002 |

この設計の問題は、入社日（「従業員」が会社に入った日付）と退職日（「従業員」が会社をやめた日付）がわかっている場合、両方を記録できないことです。これは、1 つの属性で 2 つの異なる事実を表しているからです。これも COBOL ステートメントではよく使われる構造ですが、しばしば保守上の問題や情報の誤った解釈の原因となります。

この問題を解決するには、それぞれの事象を別々の属性で表すようにします。次の図 7-4 は、問題を解決する試行を表します。しかし、これでもまだ完全ではありません。たとえば、従業員の入社日を知りたい場合、「日付区分」属性から日付の種類を導かなければなりません。これは物理データベース領域を節約するうえでは効果的ですが、検索ロジックに混乱が生じます。

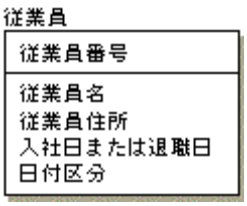



**図 7-4 「日付区分」という属性を持つ「従業員」エンティティ**

実際、この変更によって新たな正規化違反が生じてしまいました。それは、「日付区分」は「従業員番号」に依存していないことです。この設計は、技術的問題を解決していますが、もととなった業務上の問題（従業員に関する 2 つの事実をどのようにして格納するか）は解決されていません。

データをよく見ると、図 7-5 のように、各属性が別々の事実を表すようにするのがもっとよい解決法であることがすぐにわかります。


図 7-5 「入社日」属性と「退職日」属性を持つ「従業員」エンティティ

「入社日」属性と「退職日」属性を示すサンプル インスタンス表を次に示します。

**従業員**

| 従業員番号 | 従業員名 | 従業員住所 | 入社日 | 退職日 |
|---|---|---|---|---|
| E1 | Tom | Berkeley | 1 月 10, 2004 | — |
| E2 | Don | Berkeley | 5 月 22, 2002 | — |
| E3 | Bob | Princeton | 3 月 15, 2003 | — |
| E4 | John | New York | 9 月 30, 2003 | — |
| E5 | Carol | Berkeley | 4 月 22, 2000 | — |
| E6 | George | Pittsburgh | 10 月 15, 2002 | Nov 30, 2003 |

これまでに示した 2 つの例は第 1 正規形違反に相当します。この違反を避けるには、エンティティの構造を変更して、各属性がエンティティ内に 1 回だけ出現し、かつ 1 つの事実だけを表すようにします。すべてのエンティティと属性の名前を単数形にし、複数の事実を表す属性がないようにすると、第 1 正規形モデルの確立に向けて大きな一歩を踏み出したことになります。

## 同一事象の複数オカレンス

リレーショナル データベースの目的の 1 つはデータの一貫性を最大限にすることです。これによって、データベースの情報が常に正確で、データベース内の事象が互いに矛盾しないようになります。データの一貫性を最大限にするには、データベース内の各事実を 1 回だけ表すことが重要です。ある事実が複数の場所で出現する場合、データにエラーが入り込む可能性があります。このルール(1 つの事実は 1 ヶ所に)の唯一の例外として、キー属性の場合、データベース内に複数回現われます。しかし、キーの一貫性は「リレーションシップの詳細定義」の章で述べた参照整合性を使って管理されます。

同じ事実が複数回発生する場合、もとのデータ設計に欠陥があることがしばしばあります。図 7-6 では、「子供」エンティティに「従業員住所」をいれたことによって、データベース設計にエラーが生じたことがわかります。従業員に複数の子供がいる場合、それぞれの子供について住所を管理しなければなりません。

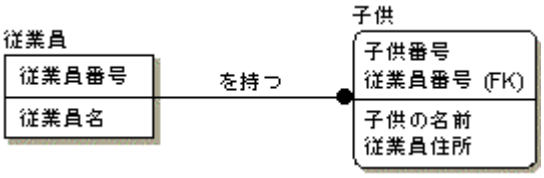



図 7-6 同じ事実の複数出現

「従業員住所」は、「子供」に関する情報ではなく「従業員」に関する情報です。実際、このモデルは「各事実がエンティティに属するには、エンティティのキー全体に依存していなければならない」という第 2 正規形に違反しています。上の例は、「従業員住所」が「子供」のキー全体に依存していないので、第 2 正規形ではありません。「従業員住所」を「従業員」に戻すと、モデルは少なくとも第 2 正規形になります。

## 矛盾する事実

事実の矛盾は、第 1、第 2、または第 3 正規形の違反を含むさまざまな理由で起こります。第 2 正規形の違反によって発生する矛盾する事実の例を図 7-7 に示します。

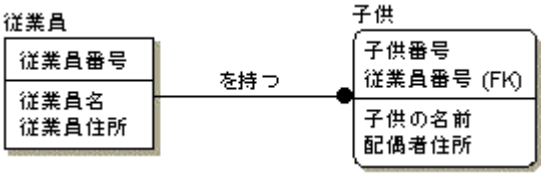



図 7-7 事実の矛盾する「従業員－子供」モデル

「従業員-配偶者-住所」のサンプル インスタンス表を次に示します。

**従業員**

| 従業員番号 | 従業員名 | 従業員住所 |
|---|---|---|
| E1 | Tom | Berkeley |
| E2 | Don | Berkeley |
| E3 | Bob | Princeton |
| E4 | Carol | Berkeley |

子供

| 従業員番号 | 子供番号 | 子供の名前 | 従業員配偶者住所 |
|---|---|---|---|
| E1 | C1 | Jane | Berkeley |
| E2 | C1 | Tom | Berkeley |
| E2 | C2 | Dick | Berkeley |
| E2 | C3 | Donna | Cleveland |
| E4 | C1 | Lisa | New York |

「従業員配偶者住所」という属性が「子供」にありますが、この設計は第 2 正規形違反です。この違反は、インスタンス データを見るとよくわかります。「Don」は「Tom」、「Dick」、「Donna」の親であることがわかります。しかし、「Don」の配偶者に 2 つの異なる住所が登録されています。おそらく「Don」には「Berkeley」と「Cleveland」に 2 人の配偶者がいるか、「Donna」は「Tom」や「Dick」と母が異なるのでしょう。あるいは、「Don」の配偶者は「Berkeley」と「Cleveland」の両方に住所があることが考えられます。正しい答えはどれでしょうか。このような意味上の問題を解決するにはビジネスユーザに聞くしかありません。アナリストは、正しい設計を見つけるために業務に関する適切な質問をする必要があります。

この例で問題なのは、「従業員配偶者住所」が、「子供」ではなく「従業員」の「配偶者」についての情報である点です。モデルの構造をこのままの状態にしておくと、「Don」の配偶者が(「Don」と共に)住所を変えるたびに、複数の情報を更新しなければなりません。 複数の情報とは、「Don」の「子供」のインスタンスのことです。複数の更新が必要な場合、必ずしもすべての更新が正しく行われるという保証はありません。これは、あまり好ましいことではありません。

「従業員配偶者住所」は配偶者ではなく子供に関する事実であることがわかると、問題を解決できます。この情報を取り込むには、図 7-8 に示すように、「配偶者」エンティティをモデルに追加します。

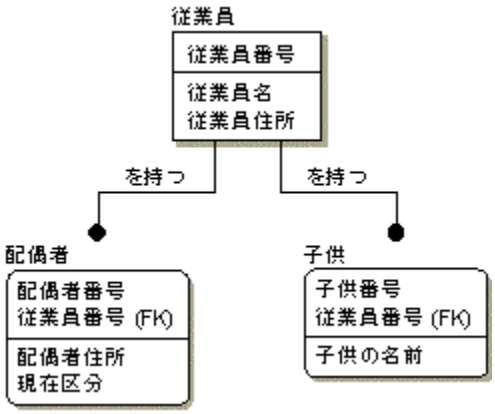



図 7-8 「従業員－子供」モデルに「配偶者」エンティティを追加したモデル

「配偶者」エンティティを反映した 3 つのサンプル インスタンス表を次に示します。

**従業員**

| 従業員番号 | 従業員名 | 従業員住所 |
|---|---|---|
| E1 | Tom | Berkeley |
| E2 | Don | Berkeley |
| E3 | Bob | Princeton |
| E4 | Carol | Berkeley |

**子供**

| 従業員番号 | 子供番号 | 子供の名前 |
|---|---|---|
| E1 | C1 | Jane |
| E2 | C1 | Tom |
| E2 | C2 | Dick |
| E2 | C3 | Donna |
| E4 | C1 | Lisa |

**配偶者**

| 従業員番号 | 配偶者番号 | 配偶者住所 | 現在区分 |
|---|---|---|---|
| E2 | S1 | Berkeley | Y |
| E2 | S2 | Cleveland | N |
| E3 | S1 | Princeton | Y |
| E4 | S1 | New York | Y |
| E5 | S1 | Berkeley | Y |

「配偶者」を別のエンティティに分割することによって、「Don」の配偶者の住所データに間違いがないことが確認できました。現在の配偶者だけでなく前妻も登録されています。

一般にエンティティ内の各属性が、そのエンティティについてただ 1 つの事実を表すようにすることで、少なくとも、そのモデルは第 2 正規形であると見なすことができます。さらに第 3 正規形に変換すると、重複情報を減らし、必須情報の欠落などを防ぐことができ、より良いデータベースにすることができます。

## 導出属性

矛盾する事実のもう 1 つの例は、第 3 正規形に違反したときに起こります。たとえば、「生年月日」と「年齢」の両方を非キー属性として「子供」エンティティにいれた場合、第 3 正規形に違反します。これは、「年齢」が「生年月日」に依存するからです。「生年月日」と今日の日付がわかれば、「子供」の「年齢」を導出することができます。

導出属性は、他の属性から計算可能な属性（たとえば、合計値）で、直接格納する必要はありません。より厳密にいうと、導出属性は、その導出元が変化するたびに更新する必要があります。これは、バッチ ロードやバッチ更新を行うアプリケーションなどで大量のオーバーヘッドを起こします。また、導出した事象の更新を必ず実行するという責任をアプリケーション設計者やプログラマに負わせることになります。

正規化の目的は、データベースに記録された各事象を知るための方法を 1 つにすることです。導出事象の値、それを求めるためのアルゴリズムおよびアルゴリズムが使用する属性の値を知っている場合、事象を知るための方法が 2 つあります（導出属性の値を見る方法と計算で求める方法）。2 つの方法で答えが得られる場合、2 つの答えが異なっている可能性があります。

たとえば、「子供」について「生年月日」と「年齢」の両方が記録できるとします。さらに「年齢」属性は毎月の保守作業の終わりにだけ変更されるとします。ここで「この「子供」は何才か？」という質問があるとき、直接「年齢」を聞くか、「今日の日付」から「生年月日」を減算することによって答えが得られます。減算を行った場合、常に正しい答えが得られます。「年齢」が更新されてから日数が経っている場合、誤った答えとなり、2 つの答えが矛盾する可能性が常にあります。

データの計算に時間がかかる場合などには、導出データをモデル内に記録することに意味があります。また、モデルを業務の中で議論する場合も非常に便利です。モデリング理論では、導出データを使用しないように述べていますが（このマニュアルでも通常は使わないように推奨していますが）、本当に必要なときはルールを破ることも大切です。しかし、少なくとも属性が導出した事象と導出アルゴリズムの状態は記録してください。

## 欠落情報

モデル内で欠落した情報は、データの正規化作業がもとで起こることがあります。次の例では、「従業員－子供」モデルに「配偶者」エンティティを追加することによって設計が改善されていますが、「子供」エンティティと「配偶者」住所の間の暗示的なリレーションシップが壊れてしまっています。最初に「従業員配偶者住所」を「子供」エンティティに格納したのは、子供にとってもう一方の親（配偶者と仮定した）の住所を表すためでした。それぞれの子供のもう一方の親を知る必要がある場合は、この情報を「子供」エンティティに追加する必要があります。

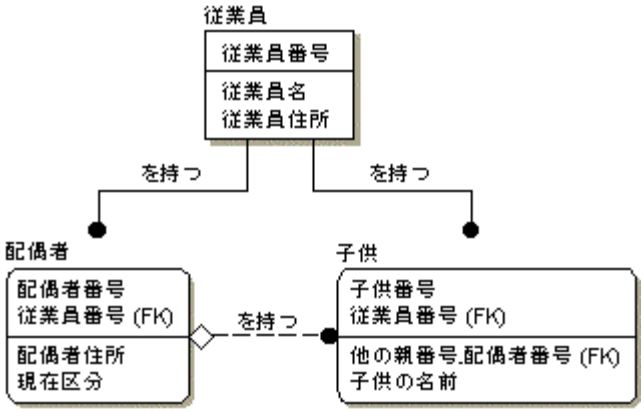



図 7-9　新しいリレーションシップを使用して欠如情報を置き換える

「従業員」、「子供」、および「配偶者」の 3 つのサンプル インスタンス表を次に示します。

**従業員**

| 従業員番号 | 従業員名 | 従業員住所 |
|---|---|---|
| E1 | Tom | Berkeley |
| E2 | Don | Berkeley |
| E3 | Bob | Princeton |
| E4 | Carol | Berkeley |

**子供**

| 従業員番号 | 子供番号 | 子供の名前 | 他の親番号 |
|---|---|---|---|
| E1 | C1 | Jane | — |
| E2 | C1 | Tom | S1 |
| E2 | C2 | Dick | S1 |
| E2 | C3 | Donna | S2 |
| E4 | C1 | Lisa | S1 |

**配偶者**

| 従業員番号 | 配偶者番号 | 配偶者住所 | 現在区分 |
|---|---|---|---|
| E2 | S1 | Berkeley | Y |
| E2 | S2 | Cleveland | N |
| E3 | S1 | Princeton | Y |
| E4 | S1 | New York | Y |
| E5 | S1 | Berkeley | Y |

しかし、このモデルの正規化は完成していません。完成するには、両親とも従業員である場合を含む、従業員と子供の間のすべてのリレーションシップを表すことができなければなりません。

# 一意化

以下の例では、「従業員」に関するリレーションシップと「子供」に関するリレーションシップの 2 つによって、「従業員番号」属性が「子供」エンティティに移行します。この結果、「子供」エンティティに外部キー属性が 2 回現れることが考えられます。しかし、「従業員番号」属性は「子供」のキー領域にすでにあるので、「配偶者」のキーの一部であっても「子供」の中で繰り返されません。

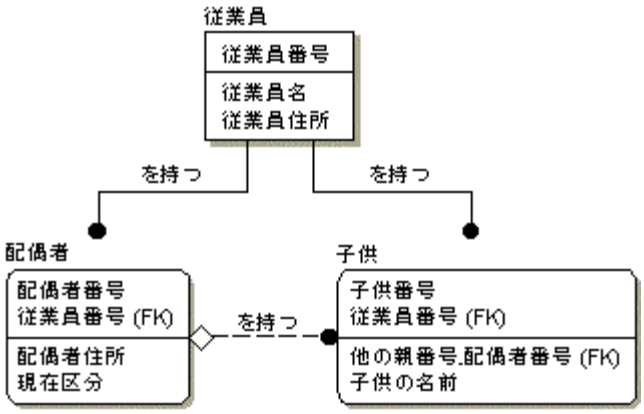



図 7-10　「従業員番号」外部キー属性の一意化

複数のリレーションシップによって同じ基本属性から移行された 2 つの同じ外部キー属性を組み合わせることを一意化（unification）と呼びます。上の例では、「従業員番号」が「子供」の主キー（「従業員」からリレーションシップ<を持つ>によって移行される）で、「子供」の非キー属性（「配偶者」からリレーションシップ<を持つ>によって移行される）にもなっています。両方の外部キー属性とも同じ「従業員」の識別子なので、属性は 1 つだけにした方がよいのです。このような場合、AllFusion ERwin DM は自動的に一意化を適用します。

一意化のルールを次に示します。

- 1 つのエンティティから移行した同じ外部キーは一意化されます（ただし、ロール名が設定されている場合は別）。
- 外部キーに異なるロール名が指定されている場合、一意化は行われません。
- 異なる外部キーに同一のロール名が定義されていて、これらの外部キーが同一基本属性に結び付く場合、一意化が行われます。一方、同一基本属性に結び付かない場合、ダイアグラムの作成エラーとなります。
- 一意化の対象となる外部キーが主キーの一部である場合、一意化された属性は主キーの一部として残ります（識別するために必要だからです）。
- 一意化の対象となる外部キーが主キーでない場合、一意化された属性は主キーの一部にはなりません。

したがって、必要であればロール名を割り当てることによって外部キーの一意化を無視できます。同じ外部キーが子エンティティ内で複数化現れるようにするには、それぞれの外部キー属性にロール名を追加します。

## どの程度の正規化で充分か？

形式的な正規化（エンティティや属性の意味を考えることなく、アルゴリズムがモデルの形だけから見つけ出した正規化）という観点では、「従業員－配偶者」モデルに何ら問題はありません。ただし、正規化されたからといって、そのモデルが完全または正しいとは限りません。すべての必要な情報を格納できなかったり、格納が非効率的な場合があります。経験を積むと、純粋な正規化が終わったあとも設計上の問題を見つけ、削除できることがあります。

前の例の「従業員－子供－配偶者」モデルでは、両親とも「従業員」であるような「子供」を記録する方法がないことについてすでに述べました。したがって、このようなデータに対応するための変更の余地があります。

「従業員」、「配偶者」、「子供」がすべて人というインスタンスを表していることに気がつくと、これらの情報を人に関する事実を表すテーブルと、リレーションシップに関する事実を表すテーブルにまとめることができます。モデルを修正するには、「子供」と「配偶者」を削除し、「人」と「人関係」に置き換えます。これによって、「人関係」エンティティで取り込まれた 2 人の「人」のリレーションシップによって、親子関係や婚姻関係を記録できます。

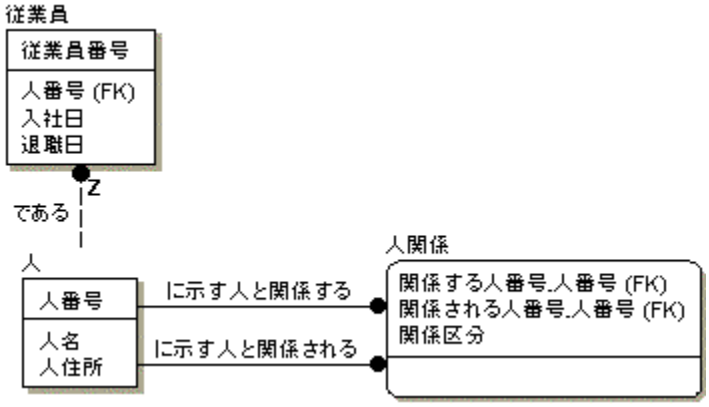



図 7-11 「従業員」、「人」、「人関係」エンティティ

この構造では、2 つの「個人」の関係をいくつでも記録できるようになるとともに、それまで記録できなかったいくつかの関係も記録できるようになりました。たとえば、前のモデルでは養子縁組を考慮していませんでした。一方この新しい構造では、養子縁組が自動的にカバーされます。「養子縁組」を表すには、養子の親であることを表す新しい値を「人関係区分」のバリデーション ルールに追加します。さらに必要なら「後見人」、「内縁」、その他のリレーションシップを 2 人の「人」の間に追加できます。

「従業員」は「人」と区別されるので、独立したエンティティのままです。ただし「人」に対する「である(is a)」リレーションシップにより、「EMPLOYEE」は「人」のプロパティを継承しています。リレーションシップに「Z」が付いていることと、ダイアモンドがないことに注意してください。これは、1 対 1(One－to－Zero or One)関係で、サブタイプエンティティに別のキーが必要なときにサブタイプの代わりに使用されます。上の例では、「人」が「従業員」<である>か、ないかです。

「人」と「従業員」の両方に同じキーを使用したい場合、「従業員」エンティティを「人」に組み込み、「人」が「従業員」ではない場合に備えて属性が NULL を取れるようにします。別の識別子によって「従業員」を検索しようとしていると指定できますが、ビジネス ステートメントは若干異なります。この構造を図 7-12 に示します。

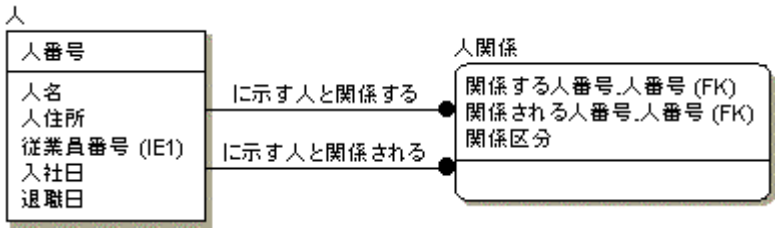



図 7-12 一般化された従業員モデル構造体

つまり、モデルは正規化できますが、それでも業務を正しく表現していない場合があるということです。形式的正規化は重要です。この章で行ったようにいくつかのサンプル インスタンス表によって、モデルに何らかの意味を表していることをチェックするのはやはり大切なことです。

# 正規化サポート

AllFusion ERwin DM はデータ モデルの正規化をある程度までサポートしますが、現在のところ、完全に正規化アルゴリズムをサポートしているわけではありません。これまでに「対話型」モデリング ツールを使用したことがない方には、AllFusion ERwin DM の標準モデリングの機能は大変効果的です。これによって、初期の段階で正規化違反の発生を防ぐことができます。

## 第 1 正規形のサポート

モデルの中で各エンティティまたは属性は、その名前で識別されます。AllFusion ERwin DM では、オブジェクトの名前として任意の名前を指定できますが、以下の場合は例外です。

- AllFusion ERwin DM は、同一のエンティティ名が 2 度使用されるとフラグを立てます（[ユニーク名]オプションの指定により変わります）。
- AllFusion ERwin DM は、同一の属性名が 2 度使用されるとフラグを立てます。ただしロール名の場合は除きます。ロール名を指定すると、同一名の属性を異なるエンティティの中で使用できます。
- AllFusion ERwin DM は、ロール名を指定せずに、エンティティの中に同一外部キーを複数設定することはできません。

AllFusion ERwin DM は、同一名を何度も使用しないように、また、事実は 1 つの場所に限って設定するようにします。この原則に従わない場合には、第 2 正規形違反になることもあります。どのように属性名を設定すべきかは、その業務を詳しく理解した上で決定する必要があります。

データ モデルでは、AllFusion ERwin DM は属性に設定された名前がリスト データを表すかどうかを判断することはできません。たとえば、前例の中で、「子供達の名前」という属性名がそのまま設定されてしまったような場合です。このような時には、すべてのモデルが第 1 正規形であるという保証ができません。

AllFusion ERwin DM の DBMS スキーマ関数は、リスト データというデータ型をサポートしません。スキーマは、物理的なリレーショナル データベースに相当するため、このことからも第 1 正規形違反を防止しています。

## 第 2 および第 3 正規形のサポート

AllFusion ERwin DM は現在のところ、機能依存を認識していません。ただし、第 2 および第 3 正規形違反の発生を防ぐ手助けをすることはできます。たとえば、この章のはじめに示した例を再構成する場合、「配偶者」の属性として指定した「配偶者住所」を、「子供」の属性として設定することはできません。(ただし、[ユニーク名]オプションの設定によります。)

ロール名を持たない外部キーが複数回繰り返しモデル上に表れないように、構造をよく考慮する必要があります。同一のエンティティに同じ外部キーが 2 回出現する場合は、確認する業務上の問いがあります。つまり、「2 つの個別のインスタンスを記録しているのか、それとも両方のキーが同一のインスタンスを表しているのか」ということです。

外部キーが異なるインスタンスを表す場合、別のロール名が必要です。2 つの外部キーが同一インスタンスを表す場合、正規化違反の可能性が高まります。ロール名を指定せずにエンティティ内で同一外部キーが 2 回以上使用される場合、モデル内のリレーションシップ構造は冗長です。なお、2 つの外部キーに同一ロール名を設定した場合にも、一意化が行われます。

Chapter

8

# 物理モデルの作成

物理モデルの目的は、データベース管理者が効率的な物理データベースを作成するために必要な情報を提供することです。さらにこのモデルは、データ ディクショナリの中でデータベースを形成するデータ要素を定義および記録するためのコンテキストを提供し、アプリケーション チームがデータベースにアクセスするプログラムのための物理構造を選択するのを支援します。システム側のすべてのニーズが満足されるように、物理モデルはしばしばデータ管理、データベース管理、アプリケーション開発の各領域を代表するチームと共同で開発されます。

開発作業にとって適切な場合、モデルは物理データベース設計を本来の業務情報要件と比較するための基礎も提供します。

- 物理データベース設計がそれらの要件を充分にカバーしていることを例示します。
- 物理設計の選択肢と意味（たとえば、何が満足され、何が満足されないか）を文書化します。
- データベースの拡張性と制約を明らかにします。

## 物理モデルの役割のサポート

AllFusion ERwin DM は、物理モデルの両方の役割、つまり、物理データベースの生成と業務要件に対する物理設計の文書化をサポートしています。たとえば、論理/物理モデルでは、モデルの表示を論理モデルから物理モデルに変更するだけで、ERD モデル、キー ベース モデル、または全属性モデルから物理モデルを作成できます。論理モデルの各オプションは、物理モデルの各オプションと対応しています。つまり、エンティティがテーブル、属性がカラム、キーがインデックスになります。

物理モデルが完成すると、AllFusion ERwin DM では選択した対象サーバに対応した正しいシンタックスで、すべてのモデル オブジェクトを（対象 DBMS のシステムカタログに対して直接的か、DDL スクリプト ファイルとして間接的に）作成できます。

## 論理モデルと物理モデルの構成要素

論理モデルのオブジェクトと物理モデルのオブジェクトの関係を以下の表に示します。

| 論理モデル | 物理モデル |
|---|---|
| エンティティ | テーブル |
| 依存エンティティ | FK は子テーブルの PK の一部です。 |
| 独立エンティティ | 親テーブルまたは子テーブル。子テーブルの場合、FK は子テーブルの PK ではありません。 |
| 属性 | カラム |
| **論理データ型**（TEXT、NUMBER、DATETIME、BLOB） | 物理データ型<br>（有効な例は選択された対象サーバによって異なる） |
| ドメイン（論理） | ドメイン（物理） |
| 主キー | 主キー、PK インデックス |
| 外部キー | 外部キー、FK インデックス |
| 代替キー（AK） | AK インデックス（ユニークな非主キー インデックス） |
| 逆方向エントリ（IE） | IE インデックス（顧客の名字のような、ユニークでない値によってテーブル情報を検索するために作成される非ユニーク インデックス） |
| キー グループ | インデックス |
| ビジネス ルール | トリガまたはストアド プロシージャ |
| バリデーション ルール | 制約 |
| リレーションシップ | FK を使用して実装されたリレーションシップ |
| 依存型リレーションシップ | FK は子テーブルの PK の一部です（仕切り線の上側）。 |
| 非依存型リレーションシップ | FK は子テーブルの PK の一部ではありません（仕切り線の下側）。 |
| サブタイプ リレーションシップ | 非正規化テーブル |
| 多対多リレーションシップ | 関連テーブル |
| 参照整合性リレーションシップ（CASCADE、RESTRICT、SET NULL、SET DEFAULT） | INSERT トリガ、UPDATE トリガ、DELETE トリガ |
| カーディナリティ リレーションシップ | INSERT トリガ、UPDATE トリガ、DELETE トリガ |
| なし | ビューまたはビュー リレーションシップ |
| なし | プリスクリプトまたはポストスクリプト |

**注:** 参照整合性は、論理モデルの一部として記述されます。 これは、リレーションシップをどのように保持するかは業務上の意思決定によるからです。また、参照整合性は物理モデル構成要素でもあります。 これは、トリガや宣言ステートメントがスキーマに現れるからです。AllFusion ERwin DM は、論理モデルと物理モデルの両方についてリレーショナル参照整合性をサポートしています。

## 非正規化

AllFusion ERwin DM では、論理モデルの構造を非正規化することもできます。 これによって、対象 RDBMS にとって効率よく設計された物理モデルを作ることができます。非正規化をサポートしている機能には次のものがあります。

- エンティティ、属性、キー グループ、およびドメインの論理のみプロパティ。論理モデルのどの項目にも、「論理のみ」とマークを付けることができます。これにより、その項目は論理モデルには表示されますが、物理モデルには表示されません。たとえば、「論理のみ」の設定を使って、サブタイプ リレーションシップを非正規化したり、物理モデル内の特定のキーの移行をサポートしたりできます。
- テーブル、カラム、インデックス、およびドメインの[物理のみ]プロパティ。物理モデルのどの項目にも、「物理のみ」とマークを付けることができます。これにより、その項目は物理モデルにのみ表示されます。この設定は、物理モデルの非正規化もサポートしています。これは、モデル作成者がテーブル、カラム、インデックスを物理への実装要件を直接サポートする物理モデルに含めることができるからです。
- 物理モデルでの多対多リレーションシップの解決。AllFusion ERwin DM は、論理モデルと物理モデルの両方について多対多リレーションシップの解決をサポートしています。論理モデルで多対多リレーションシップを確定すると、関連エンティティが作成され、追加属性を追加できます。論理モデルで多対多リレーションシップを維持する場合でも、物理モデルでリレーションシップを確定できます。AllFusion ERwin DM では、元の論理設計と新しい物理設計の間のリンクを維持します。そのため、関連テーブルの起源はモデル内で文書化されます。

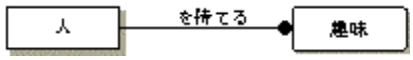
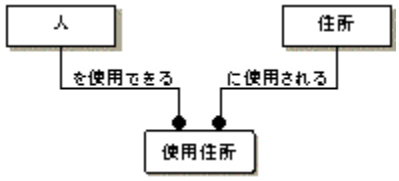
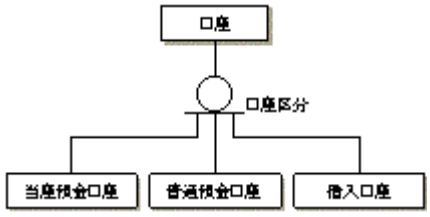
Appendix

A

# 依存エンティティの種類

## 依存エンティティの分類

IDEF1X のダイアグラムに表示される依存エンティティの種類を以下の表に示します。

| 依存エンティティの種類 | 説明 | 例 |
|---|---|---|
| 特性（Characteristic） | 特性エンティティは、1 つのエンティティに対して複数回出現する属性からなり、他のエンティティによって直接識別されないようなグループを表します。例では、「趣味」が「人」の特性です。 |  |
| 関連（Associative）<br>または<br>指定（Designative） | 関連エンティティと指定エンティティは、複数のエンティティ間の複数のリレーションシップを記録します。リレーションシップの情報しかないエンティティを指定エンティティと呼びます。リレーションシップの情報に加え、リレーションシップについて説明する属性も持っている場合、関連エンティティと呼びます。例では、「使用住所」が関連エンティティまたは指定エンティティです。 |  |
| サブタイプ | サブタイプ エンティティは、サブタイプ リレーションシップ内の依存エンティティです。例では、「当座預金口座」、「普通預金口座」、「借入口座」は、サブタイプ エンティティです。 |  |

# 用語集

### 2 項リレーションシップ
親エンティティの 1 つのインスタンスが正確に、0、1、または 1 以上の子エンティティのインスタンスに関連付けられているリレーションシップ。IDEF1X 表記法では、依存型、非依存型、サブタイプの各リレーションシップをすべて 2 項リレーションシップといいます。

### BLOB
テーブル カラムに保管される 2 進ラージ オブジェクト、つまり、BLOB を構成するバイトおよびテキスト データのストレージのために予約されている DB 領域。BLOB DB 領域には、イメージ、オーディオ、ビデオ、長いテキスト ブロック、またはその他のデジタル化された情報を保持することができます。

### 依存エンティティ
別のエンティティまたは複数のエンティティとのリレーションシップを決定しない限り、そのエンティティのインスタンスを独自に識別できないエンティティ。

### 依存型リレーションシップ
子エンティティのインスタンスが、親エンティティとの関連から識別されるリレーションシップ。親エンティティの主キー属性が、子エンティティの主キー属性になります。

### エンティティ
エンティティとは、共通の属性または特性を持つ具体的または抽象的なもの（人、場所、事象など）を意味します。エンティティの種類には、独立エンティティと依存エンティティがあります。

### 外部キー
リレーションシップを通じて子エンティティに移行された、親エンティティの主キー属性のことをいいます。外部キーは、1 つの値セットに対する二次参照です（一次参照は実属性です）。

### カーディナリティ
親と子がそれぞれ何対何の関係にあるかを定義します。IDEF1X 表記法では、2 項リレーションシップのカーディナリティは 1:n で、n は以下のいずれかがあてはまります。

- 0、1、または 1 以上
  黒いドットが表示されます。
- 1 以上
  黒いドットとともにPのマークが表示されます。
- 0 または 1
  黒いドットとともにZのマークが表示されます。
- N
  黒いドットとともにn（指定した数）が表示されます。

### 確定型サブタイプ エンティティ
サブタイプ エンティティの中に、想定されるすべてのサブタイプが含まれている場合（汎化親のすべてのインスタンスが 1 つのサブタイプに関連付けられている場合）、そのサブタイプ エンティティは確定型サブタイプ エンティティです。たとえば各「口座」は当座預金口座、普通預金口座、または借入口座のいずれかであり、「当座預金口座」、「普通預金口座」、または「借入口座」のサブタイプ エンティティは、確定型サブタイプ エンティティです。

### 基本名
ロール名が付けられた外部キーの本来の名前。

### 逆方向エントリ
エンティティのインスタンスを一意に識別しないが、エンティティのインスタンスにアクセスする際によく使われる属性。AllFusion ERwin DM では、各逆方向エントリの非ユニーク インデックスが生成されます。

### サブタイプ エンティティ
他のエンティティの特定のタイプであるエンティティ、というものが存在します。たとえば、「正社員」は「従業員」の特定のタイプです。サブタイプ エンティティは、特定のサブタイプにのみ適用される情報を保管する場合に役立ちます。また、その特定のサブタイプに対してのみ有効なリレーションシップを表現する際にも有用です。たとえば、「「正社員」は福利厚生などの手当てをもらう資格があるが、「パートタイマー」にはその資格がない」というケースです。IDEF1X では、サブタイプ エンティティ内のサブタイプは相互に排他的です。

### サブタイプ リレーションシップ
サブタイプ リレーションシップは、スーパータイプとサブタイプ エンティティとの間のリレーションシップのことをいいます。サブタイプ リレーションシップでは、常にスーパータイプとサブタイプのインスタンスとが 1:0 または 1 で結び付きます。

### 参照整合性
「子エンティティのインスタンス内の外部キー値に対応する値を、親エンティティ内に持っている」ということ。

### 識別子
汎化親のインスタンスにおける属性の値によって、そのインスタンスが属するサブタイプが決まります。この属性を識別子と呼びます。たとえば、「口座」のインスタンスにおける「口座区分」属性の値によって、そのインスタンスが属するサブタイプ(「当座預金口座」、「普通預金口座」、または「借入口座」)が決まります。

### 主キー
エンティティのインスタンスを一意に識別する属性。1 つ以上の属性または属性グループが各インスタンスを一意に識別できる場合、主キーは、業務における識別子として認められた価値を基準にして、この候補キーの中から選択されます。主キーは、時間が経過しても変化せず、またできるだけ短いことが理想的です。主キーに指定された属性には、ユニーク インデックスが生成されます。

### スキーマ
データベースの構造。通常、DDL(データ定義言語)スクリプト ファイルのことです。DDL は CREATE TABLE、CREATE INDEX、およびその他のステートメントで構成されます。

### 正規化
冗長性と非リレーショナル構造を最小限にするために、リレーショナル構造のデータを編成するプロセス。

### 属性
具体的または抽象的な物(人、場所、イベントなど)のセットに関連付けられた特性やプロパティの種類を表します。

### 代替キー
1. エンティティのインスタンスを一意に識別する属性。
2. **1 つ以上の属性または属性のグループがエンティティのインスタンスをユニーク(一意)に識別する場合、代替キーとは主キーとして選択されない属性または属性のグループのことです。**代替キーに指定された属性には、ユニーク インデックスが生成されます。

### 特定リレーションシップ
特定リレーションシップとは、「親エンティティの各インスタンスが子エンティティの 0 個、1 個、または複数個のインスタンスと関連しており、かつ、子エンティティの各インスタンスが親エンティティの 0 個または 1 個のインスタンスと関連している、2 つのエンティティ間の関連」です。

### 独立エンティティ
他のエンティティとのリレーションシップが決まっていなくてもそのインスタンスを一意に識別できるエンティティ。

### ドメイン
事前に定義された論理および物理プロパティの特定のグループのことで、保存後、それを選択して属性とカラムにアタッチできます。

### 非正規化
テーブル内のデータの冗長性を許可して、クエリの性能を向上させること。

### 非キー属性
エンティティの主キーの一部になっていない属性。非キー属性は、逆方向エントリまたは代替キーの一部になっていることがあります。また、外部キーになっていることもあります。

### 非依存型リレーションシップ
子エンティティのインスタンスが、親エンティティとの関連を通して識別されないリレーションシップ。親エンティティの主キー属性が、子エンティティの非キー属性になります。

### 物理モデル
AllFusion ERwin DM のデータ モデリング レベル。これには、データベースおよびデータベース管理システム(DBMS)特定のモデリング情報(テーブル、カラム、データタイプなど)を追加できます。

### 不特定リレーションシップ
親－子接続とサブタイプ リレーションシップは両方とも、特定リレーションシップであると見なされますがその理由は、あるエンティティのインスタンスと別のエンティティのインスタンスとの関係が正確に定義されるからです。ただし、モデルの開発の初期段階では、2 つのエンティティ間を不特定リレーションシップと識別すると便利な場合があります。不特定リレーションシップは、多対多(Many－to－Many)リレーションシップとも呼ばれます。不特定リレーションシップは、「第 1 エンティティの各インスタンスが第 2 エンティティの 0 個、1 個、または複数個のインスタンスと関連しており、かつ、第 2 エンティティの各インスタンスが第 1 エンティティの 0 個、1 個、または複数個のインスタンスと関連している、2 つのエンティティ間の関連」です。

### 未確定型サブタイプ エンティティ

サブタイプ エンティティの中に、想定されるすべてのサブタイプが含まれていない場合（汎化親のすべてのインスタンスが 1 つのサブタイプに関連付けられているわけではない場合）、そのサブタイプ エンティティは未確定型サブタイプ エンティティです。たとえば、一部の従業員に特定の資格が与えられている場合、「正社員」と「パートタイマー」のサブタイプ エンティティは未確定型サブタイプ エンティティです。

### ロール名

外部キー属性に対して新たな名前（別名）を定義したものです。ロール名は、業務内容をよりわかりやすく表現することを目的として、そのキーの果たす役割に応じて付けられる名前です。

### 論理モデル

概念モデルを作成するための AllFusion ERwin DM のデータ モデリング レベルで、これにはエンティティ、属性、キー グループなどのオブジェクトが含まれます。

### 論理/物理モデル

論理モデルおよび物理モデルが自動的にリンクされる AllFusion ERwin DM で作成されるモデルの種類。

# 索引

## 英数字



## あ

## か



## さ

## た



## な



## は

## ま



## ら