

AllFusion® ERwin® Data Modeler

手法ガイド

r7.2



本書及び関連するソフトウェア ヘルプ プログラム(以下「本書」と言います)は、ユーザーへの情報提供のみを目的とし、CA は本書の内容を予告なく変更、撤回することがあります。

CA の事前の書面による承諾を受けずに本書に記載されている内容のすべてまたは一部を複製、譲渡、再生、開示、変更、複製することはできません。本書は、CA が知的財産権を所有する機密および専有の情報であり、アメリカ合衆国及び日本国の著作権法並びに国際条約により保護されています。

前述の例外として、ライセンスを受けるユーザーは、個人使用の目的であれば本書を妥当な部数だけ印刷することや、バックアップや災害時のリカバリ目的など妥当な必要性がある場合に関連ソフトウェアを 1 部複製することができます。ただし、各複製物には CA のすべての著作権表示および説明を添える必要があります。ユーザーの認可を受け、本製品のライセンスに記述されている守秘条項を遵守する従業員、法律顧問、および代理人のみがかかるコピーを利用することを許可されます。

本書を複製する権利および関連ソフトウェアの複製を作成する権利は、本製品の該当するライセンスが完全に有効である期間内に限定されます。いかなる理由であれ、そのライセンスが終了した場合には、ユーザーは本書の複製コピーを含むすべてを CA に返却するかまたは破棄したことを、CA に対し文書で証明する責任を負うものとします。

該当する使用許諾契約書に特に明記されている場合を除き、準拠法により認められる限り、CA は本書を現状有姿のまま提供し、商品性、特定の使用目的に対する適合性、第三者の権利に対する不侵害についての黙示の保証を含むいかなる保証もしません。また、本書の使用が直接または間接に起因し、逸失利益、業務の中断、営業権の喪失、業務情報の損失等いかなる損害が発生しても、CA は使用者または第三者に対し責任を負いません。CA がかかる損害について明示に通告されていた場合も同様とします。

本書に記載された製品は、該当するエンド ユーザー ライセンス契約書に従い使用されるものです。

本書の制作者は CA です。

本書は「制限付権利」のもとで提供されます。アメリカ合衆国政府による使用、複製、または開示は、現行の FAR 第 12.212 条、第 52.227-14 条、および第 52.227-19 条(c)(1) - (2)項および DFARS 第 252.227-7014 条(b)(3)項、またはこれらの後継の条項に規定される制限事項に従います。

本書に記載された全ての製品名、サービス名、商号およびロゴはそれぞれ各社の商標またはサービスマークです。

Copyright © 2006 CA. All rights reserved.

CA 製品のリファレンス

このドキュメントでは、次の CA 製品について説明します。

- AllFusion® ERwin® Data Modeler
- AllFusion® Model Manager

テクニカル サポートへのお問い合わせ

テクニカル サポートの連絡先、および営業時間については、日揮情報システム株式会社のテクニカル サポート ページ (<http://www.jsys-products.com/support/techinfo/index.html>) をご参照ください。

目次

第 1 章 はじめに	9
本書の内容.....	9
データモデリングの利点.....	9
モデリング手法.....	10
モデリング用語の表記.....	10
第 2 章 情報システム、データベース、およびモデル	11
はじめに.....	11
データモデリング.....	11
プロセスモデリング.....	12
データモデリングセッション.....	13
セッションにおける役割.....	13
IDEF1Xモデリング手法のモデルレベル.....	14
モデリングアーキテクチャ.....	15
論理モデル.....	15
エンティティリレーションシップダイアグラム.....	16
キーベースモデル.....	16
全属性モデル.....	16
物理モデル.....	16
変換モデル.....	17
DBMSモデル.....	17
第 3 章 論理モデル	19
論理モデルの構築.....	19
エンティティリレーションシップダイアグラム.....	19
エンティティと属性の定義.....	20
論理リレーションシップ.....	21
多対多リレーションシップ.....	22
論理モデル設計の検証.....	22
データモデルの例.....	23
第 4 章 キーベースデータモデル	25
キーベースデータモデル.....	25
キーの種類.....	26
エンティティと非キー領域.....	26
主キーの選択.....	26
代替キー属性.....	28
逆方向エントリ属性.....	28
リレーションシップと外部キー属性.....	29

依存エンティティと独立エンティティ.....	29
依存型リレーションシップ.....	30
非依存型リレーションシップ.....	30
ロール名.....	31

第 5 章 エンティティと属性の名前付けと定義 33

概要.....	33
エンティティと属性の名前.....	33
同義語、同音異義語、およびエイリアス.....	34
エンティティ定義.....	34
説明.....	35
定義の参照と循環.....	36
業務用語集の作成.....	36
属性定義.....	37
バリデーション ルール.....	37
ロール名.....	38
定義とビジネス ルール.....	39

第 6 章 リレーションシップの作成 41

リレーションシップ.....	41
リレーションシップのカーディナリティ.....	41
非依存型リレーションシップのカーディナリティ.....	43
参照整合性.....	44
参照整合性オプション.....	45
参照整合性、カーディナリティ、および依存型リレーションシップ.....	47
参照整合性、カーディナリティ、および非依存型リレーションシップ.....	48
その他のリレーションシップ タイプ.....	49
多対多リレーションシップ.....	49
多項リレーションシップ.....	51
再帰リレーションシップ.....	52
サブタイプ リレーションシップ.....	54
確定型および未確定型サブタイプ構造.....	56
包括的および排他的リレーションシップ.....	57
IDEF1XおよびIEにおけるサブタイプ表記.....	58
どのような場合にサブタイプ リレーションシップを作成するか.....	59

第 7 章 正規化の問題点と解決法 61

正規化.....	61
正規形の概要.....	62
よくある設計上の問題.....	63
データ グループの繰り返し.....	63
同じ属性の多重使用.....	65
同じ事実が複数存在する.....	66
事実が矛盾する.....	67

導出属性	69
情報の欠落.....	70
一意化	71
必要な正規化のレベル	72
正規化のサポート.....	74
第 1 正規形のサポート	74
第 2 および第 3 正規形のサポート	75
第 8 章 物理モデル	77
目的	77
物理モデルの役割.....	77
論理および物理モデルのコンポーネント	78
非正規化	79
付録A: 依存エンティティの種類	81
依存エンティティの分類	81
用語集	83
索引	87

第1章 はじめに

本章には次のトピックがあります。

[本書の内容](#) (9ページを参照)

[データモデリングの利点](#) (9ページを参照)

[モデリング手法](#) (10ページを参照)

[モデリング用語の表記](#) (10ページを参照)

本書の内容

この『手法ガイド』では、データモデリングの概念とその活用法について説明します。初心者の方でも、データモデリングを実践するための十分な知識を身に付けることができます。

本書の内容は次のとおりです。

- AllFusion ERwin DM で使用されるデータモデリング手法の基本事項を説明します。これらを理解することで、実際にデータベース設計を開始することができます。
- IDEF1X および IE 表記法の記述力と優れた機能について紹介し、その基本事項を説明します。
- IDEF1X または IE の経験豊富なユーザーは、本書を利用して、各表記法の特徴や2つの表記法の対応関係を確認できます。

データモデリングの利点

開発するデータモデルや DBMS の種類によらず、データモデリングに AllFusion ERwin DM を使用すると、さまざまな利点が得られます。

- データベースやアプリケーションの開発スタッフは、システム要件を定義したり、チーム内メンバーおよびエンドユーザーとの情報共有に活用できます。
- 参照整合性制約を簡単に設定および表示できます。リレーショナルモデルでは、参照整合性の管理が不可欠です。
- 特定の RDBMS に依存しない形式で、データベースの論理モデルを作成して、各 RDBMS に固有の物理スキーマを自動生成できます。1つの論理モデルから、さまざまな RDBMS (Oracle、SQL Server、Sybase など) の物理スキーマを生成できます。
- データモデリング作業の成果を1つにまとめたダイアグラムを作成し、そこから対象データベースの物理スキーマを生成できます。

モデリング手法

AllFusion ERwin DM では、次の 2 種類のデータ モデリング手法をサポートします。

IDEF1X

IDEF1X 手法は、米国空軍によって開発されました。現在では、各種政府機関、航空および金融業界をはじめとして、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、およびその他の IE 専門家によって開発されました。さまざまな業界で幅広く使用されています。

いずれの手法も、大規模で厳密さが要求される企業レベルのデータ モデリング環境に適しています。

モデリング用語の表記

本書では、モデリングに関する用語に次のような表記を使用しています。

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」という語を付ける	「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	<レンタル用に利用できる>

第2章 情報システム、データベース、およびモデル

本章には次のトピックがあります。

[はじめに](#) (11ページを参照)

[データモデリングセッション](#) (13ページを参照)

[IDEF1Xモデリング手法のモデルレベル](#) (14ページを参照)

[モデリングアーキテクチャ](#) (15ページを参照)

[論理モデル](#) (15ページを参照)

[物理モデル](#) (16ページを参照)

はじめに

目的に応じて、異なるモデルタイプを使用します。

データモデリング

データモデリングとは、情報の構造を記述してビジネスルールを表し、情報システムの要件を定義するプロセスです。データモデルは、特定の RDBMS 実装プロジェクトに固有の要件と、業務の一般的な要件の間の結び付きを表します。

プロセスモデリング

プロセスモデルを使用すると、データに関連するシステム要件を特定して記述することができます。構造化システム開発の一般的な手法、特にデータ中心設計の手法では、フロントエンド設計と要件分析に重点を置いています。AllFusion® Process Modeler やその他のツールを使用して、データフローダイアグラム、分布モデル、およびイベント/状態モデルなどのプロセスモデルを作成し、プロセス要件を記述することができます。開発の各段階で、さまざまなモデルが使用されます。

データモデリング

データモデルを作成する際には、業務の専門家とシステムの専門家が協力することで、多くの利点が得られます。これらの利点は、次の2種類に分けることができます。

作業

モデル作成プロセスに関連した利点

作業の成果

主としてモデル自体に関連した利点

データ モデルの作成プロセスから得られる利点

- プロジェクトの初期段階でモデル開発セッションが開かれ、さまざまな業務分野の担当者が集まって、業務の要件や方針が検討されます。これらのセッションは、同じ要件に関わる社内部門の担当者が初めて顔を合わせる場となります。
- セッションを通じて業務の共通言語が決定され、各用語について一貫性のある明確な定義が作成されます。これにより、関係者の中で円滑なコミュニケーションが行えるようになります。
- 初期段階のセッションでは、業務関係者の中で多くの情報が交換され、業務に関する情報がシステム開発者に伝えられます。以降のセッションでも引き続き、必要な情報がソリューションの実装スタッフに渡されます。
- セッションの参加者は、それぞれの担当業務が業務全体の中でどのように機能しているのかを、より明確に理解できます。同様に、プロジェクトの各部分もプロジェクト全体に関連付けた形で理解できます。分業ではなく、協力して作業することが重視されます。参加者の意識が徐々に変わり、協調的な視点を持つことの重要性が認識されるようになります。
- 一連のセッションにより、さまざまな合意やチームワークが形成されます。

作成したデータ モデルから得られる利点

- データ モデルは実装から独立しており、特定のデータベースやプログラミング言語に依存しません。
- データ モデルは、要件に対する明確な仕様になります。
- データ モデルはビジネス ユーザー主導のモデルです。モデルの内容と構造は、システム開発者ではなく業務の遂行者によって管理され、制約やソリューションよりも要件そのものが重視されます。
- システム開発部門ではなく、業務部門の用語が使用されています。
- モデルから得られる情報に基づいて、業務において重要な事柄について検討することができます。

業務領域を表すデータ構造を設計することは、システム開発の一部に過ぎません。機能モデリング、つまりプロセス（機能）の分析も重要です。機能モデルは実行方法を記述するもので、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。実際に作業を始めると、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づきます。システムが実行する機能について検討すると、データ要件が明らかになりますし、データを検討することで、追加すべき機能の要件も明らかになります。機能とデータは、それぞれシステム開発というコインの表と裏に相当すると言えます。

プロセス モデリング

AllFusion ERwin DM の関連ツールである AllFusion Process Modeler (AllFusion PM) はプロセス モデリングを直接サポートし、ビジネス プロセスの定義と最適化を支援します。AllFusion PM では、IDEF0、IDEF3 ワークフロー、およびデータ フロー ダイアグラムの各手法をサポートします。AllFusion ERwin DM と AllFusion PM を連携して使用すると、データモデリング プロジェクトと並行して、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

データモデルを作成する際には、モデルを構築するだけでなく、多数の事実調査セッションを実施して、業務データや業務プロセスを明らかにしていきます。優れたセッションを実施するには、通常のミーティングと同様に、十分な準備と議論を円滑にすすめる技量が求められます。通常のモデリングセッションでは、業務および技術の専門家を適切な構成で同席させて、議論を円滑に進めていく必要があります。このためには、事前にスケジュールを練り、すべての議題を検討できるよう十分な計画を立てて、求める結果が得られるように調整します。

可能であれば、機能とデータのモデリング作業を同時に行うことをお勧めします。機能モデルからデータモデルが検証され、新しいデータ要件が明らかになることが多いからです。このアプローチによって、データモデルが機能要件をサポートすることも保証されます。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には、データモデル作成者だけでなく、機能の洗い出しを担当するプロセスモデル作成者もセッションに参加することが重要です。

セッションにおける役割

セッションを決められた手順に従って円滑に進めるには、参加者の役割を設定し、セッションの進め方や規則についてあらかじめ合意を得ることが不可欠です。次のような役割を設定することをお勧めします。

進行役

セッションの進行役を務めます。セッションを手配して会場を予約し、参考資料を用意します。セッション中は必要に応じて、討論の内容が議題から外れないように誘導します。

データモデル作成者

モデルの開発と検証プロセスにおいてグループリーダーの役割を果たします。モデル作成者は、可能であればセッション中にモデルを作成していきます。グループの先頭に立ち、適切な質問を投げかけて重要な情報を明らかにし、得られたモデル構造を記録して参加者全員が参照できるようにします。(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進行役を兼任することも少なくありません。

データアナリスト

セッションの記録係として、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、業務の専門家から提供される情報に基づき、特定のエンティティと属性をサブジェクトエリアにまとめる作業を開始します。サブジェクトエリアとは、データモデル全体を、管理し易くまとめた意味を持つ単位に切り分けたサブセットです。

対象分野の専門家

モデル構築に必要な業務情報を提供します。対象分野の専門家は、必要に応じて何人でも参加させることができます。この役割を担当する者は、システムではなく業務に関する専門家です。

マネージャ

自らに割り当てられた役割(進行役、対象分野の専門家など)でセッションに参加します。必要があれば、議論を進めるために意思決定を行う義務を負います。議論が行き詰ったときに結論を見出す権限もありますが、どうしても必要な場合に限りです。マネージャは、システムまたは業務のいずれのコミュニティから選出しても構いません。

IDEF1X モデリング手法のモデルレベル

AllFusion ERwin DM では、IDEF1X および IE モデリング標準をサポートしています。IDEF1X 手法のさまざまなモデルレベルを使い分けると、システム開発において非常に効果的です。IDEF1X 標準では一般的なモデルレベルが規定されており、本書でもそれに従って説明します。実際には、個別の状況に合わせてモデルレベルの数を増減させるとよいでしょう。

モデルレベルは通常、ある業務の主要なエンティティを広範囲かつ大まかに記述するレベルから、特定の DBMS に固有の用語でデータベース設計を表した詳細レベルまで掘り下げていきます。一番下の詳細レベルでは、モデルは特定の技術に依存したものになります。たとえば、Oracle データベース用のモデルと DB2 データベース用のモデルは大きく異なります。より高レベルのモデルは、特定の技術に依存せず、DBMS には実装されないような情報を表すこともあります。

ここで説明するモデルレベルは、トップダウン方式でシステム開発を行うライフ サイクルアプローチに適しています。プロジェクトの各段階で、各レベルのモデルが作成されます。

最上位レベルのモデルは、次の 2 種類です。

エンティティ リレーションシップ ダイアグラム (ERD)

業務における主要なエンティティとそのリレーションシップを特定します。

キーベースモデル (KB)

業務情報の要件の範囲を設定し(全エンティティを含む)、詳細情報の記述を開始します。

下位レベルのモデルは、次の 3 種類です。

全属性モデル (FA)

第 3 正規形のモデルです。実装に必要なすべての詳細情報が含まれています。

変換モデル (TM)

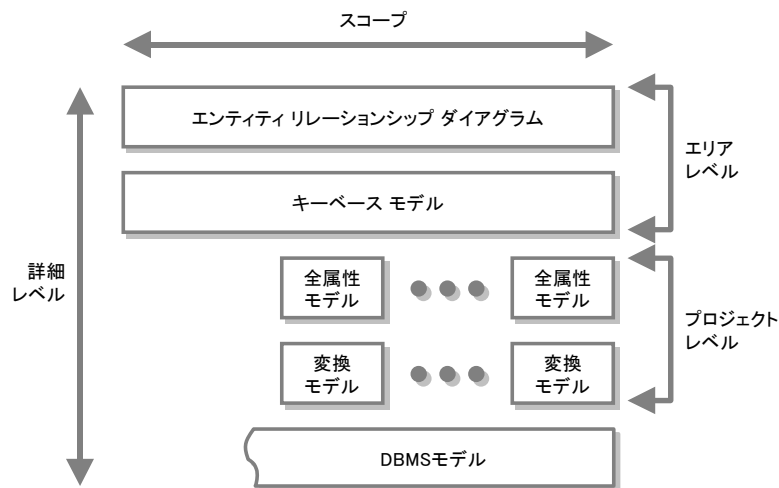
リレーショナル モデルを実装対象の DBMS に適した構造に変換したモデルです。ほとんどの場合、変換モデルは非正規化され、第 3 正規形ではありません。変換モデルの構造は、DBMS の性能、データ量、データに対し予想されるアクセス パターンやアクセス頻度に基づき最適化されています。ある意味で、これは最終的な物理データベース設計に相当します。

DBMS モデル

DBMS モデルには、データベース設計の内容が反映されています。業務の情報システムがどの程度統合されているかによって、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のモデルか、またはエリアレベルのモデルとなります。

モデリング アーキテクチャ

上に説明した5つのモデルレベルを次の図に示します。DBMSモデルの範囲は、エリアレベル、またはプロジェクトレベルのどちらでも構いません。たとえば、ある業務について1つのERDおよびKBモデルがあり、各実装環境に対応した複数のDBMSモデルが作成されます。さらに、データベースを共有しないプロジェクトに対応した別のDBMSモデルが作成される場合もあります。理想的な状況では、各実装環境に対応したエリアレベル範囲のDBMSモデルが作成され、その環境内のすべてのプロジェクトでデータが完全に共有されます。



これらのモデルは、次の2つのカテゴリに分類されます。

- 論理
- 物理

論理モデル

論理モデルは、業務情報の要件を記述するために使用されます。エンティティリレーションシップダイアグラム、キーベースモデル、および全属性モデルの3つのレベルがあります。エンティティリレーションシップ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」と呼ぶこともあります。これらのモデルは、1つの実装プロジェクトの対象業務よりも広い業務領域をカバーしていることが多いためです。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」と呼びます。通常、1つの実装プロジェクトの対象業務に相当する、一部の業務領域のみが記述されるためです。

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) は高レベルのデータ モデルです。主要なエンティティとリレーションシップを表し、幅広い業務領域をサポートします。主にプレゼンテーションや検討用のモデルとして使用されます。

ERD の目的は、情報システム開発を計画するときに最低限必要となる業務情報の要件を表すことです。このモデルは主要なエンティティのみで構成されます。属性情報がある場合でも、その詳細は記述されません。多対多(不特定)リレーションシップを使用することができ、通常、キー情報は含まれません。

キーベース モデル

キーベース (KB) モデルは主要なデータ構造を記述し、幅広い業務領域をサポートします。すべてのエンティティと主キーが、サンプル属性と共に含まれています。

KB モデルの目的は、対象の業務領域で必要となるデータ構造とキーを記述して、幅広い業務情報を表すことです。このモデルには、詳細な実装レベルのモデルを構築できるだけの情報が含まれています。KB モデルは、エリア データ モデルである ERD と同じ範囲をカバーしますが、より詳細な情報が定義されています。

全属性モデル

全属性 (FA) モデルは第 3 正規形のデータ モデルです。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、およびリレーションシップが含まれています。このモデルには、エンティティ インスタンスの容量、アクセスのパスと頻度、データ構造に対して予想されるトランザクションのアクセス パターンが含まれています。

物理モデル

実装プロジェクト用に 2 レベルの物理モデルがあります。変換モデルと DBMS モデルです。物理モデルには、システム開発者が論理モデルを理解してデータベース システムとして実装するために必要なすべての情報が記述されます。変換モデルは、プロジェクト単位の「データ モデル」と呼ぶこともあります。1 つの実装プロジェクトでサポートされるデータ構造のみが記述されるためです。AllFusion ERwin DM では、1 つの業務領域内で個別のプロジェクトを扱うことができ、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 エリアと呼ぶ複数のサブモデルに分割できます。各サブジェクト エリアごとに開発、レポート作成、およびデータベースのスキーマ生成を実行でき、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処理されます。

変換モデル

変換モデルの目的は、データベース管理者 (DBA) が効率的な物理データベースを作成するのに必要な情報を提供することです。また、アプリケーション チームが、データにアクセスするプログラムの物理構造を選択するのに役立つ情報も提供します。

変換モデルは、物理データベース設計とオリジナルの業務情報の要件を比較するための基盤にもなります。

- 物理データベース設計が業務情報の要件を満たしていることを示します。
- 物理設計の決定事項とその影響 (各要件を満たしているかなど) を記述します。
- 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

変換モデルは直接、DBMS モデルに変換され、物理データベースのオブジェクト定義を RDBMS スキーマまたはデータベース カタログに記述します。AllFusion ERwin DM のスキーマ生成機能では、DBMS モデルを直接サポートしています。主キーはユニーク 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。カーディナリティを実装するには、DBMS の参照整合性機能、アプリケーション ロジック、または事後検出して違反を修復する方法のいずれかを使用します。

第3章 論理モデル

本章には次のトピックがあります。

[論理モデルの構築](#) (19ページを参照)

[エンティティリレーションシップ ダイアグラム](#) (19ページを参照)

[論理モデル設計の検証](#) (22ページを参照)

[データモデルの例](#) (23ページを参照)

論理モデルの構築

論理モデルを構築するには、はじめにエンティティリレーションシップ ダイアグラム(ERD)を作成します。ERDは幅広い業務領域をカバーする高レベルのデータモデルであり、主要な3つの要素(エンティティ、属性、およびリレーションシップ)から構成されます。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して考えてみると、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になります。データモデルの構築は、正しい名詞、動詞、および形容詞を集めて、それらを組み合わせて配置する作業だと言えます。

ERDの目的は、業務情報の要件を大まかに表して、業務情報システムの開発計画を立てるのに必要な情報を提供することです。このモデルは主要なエンティティのみで構成されており、属性情報がある場合でも、その詳細は記述されません。多対多(不特定)リレーションシップを使用することができ、通常、キー情報は含まれません。主にプレゼンテーションや検討用のモデルとして使用されます。

ERDは複数のサブジェクト エリアに分割することができ、さまざまな業務上の視点や、各業務機能に関わる領域を個別に定義することができます。サブジェクト エリアは、大きなモデルを小さく管理しやすいサブセットに分割して、エンティティの定義や管理を容易にします。

ERDを作成するには、組織的なモデリング セッションの実施から、幅広い業務領域を担当するマネージャとの個別面談まで、さまざまな方法があります。

エンティティ リレーションシップ ダイアグラム

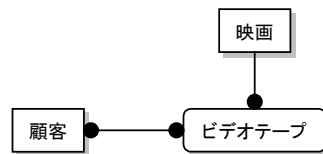
リレーショナル データベースの構造に詳しい方は、リレーショナル データベースで最も基本的な構成要素がテーブルであることをご存じでしょう。テーブルを使用すると、情報を編成して格納することができます。テーブルはデータのカラム(列)と行から構成されます。各行には一連の事実(データ値)が格納されており、これをテーブルのインスタンスと呼びます。

リレーショナル データベースでは、すべてのデータ値はアトミックでなければなりません。つまり、テーブルの各セルには1つの事実のみを格納できます。データベースのテーブル間にはリレーションシップを作成します。RDBMSにおけるリレーションシップは、2つのテーブル間で1つ以上のカラムを共有していることを表します。

リレーショナル データベースの物理モデルがテーブルやカラムから構成されるのと同様に、ERD(および他の論理データ モデル)も、DBMS ではなく業務に関するデータ構造をモデル化するためのコンポーネントから構成されます。論理データ モデルではテーブルをエンティティ、カラムを属性と呼びます。

ERD では、エンティティはボックスで表記され、その中にエンティティ名が含まれます。エンティティ名に英語を使用する場合、その名前は常に単数形となります(例:「CUSTOMERS」ではなく「CUSTOMER」)。常に単数形の名詞を使用することで、名前付け基準の一貫性を保つことができ、ダイアグラムを、エンティティ インスタンスの宣言文の集まりとして捉えることができます。

次の図は、あるビデオ ショップの顧客、レンタル/販売できる映画、およびビデオテープの在庫を追跡するために作成されました。



ERD でリレーションシップを表すには、モデル内のエンティティ間を線で結びます。2つのエンティティ間のリレーションシップが意味するのは、一方のエンティティ内の事実がもう一方のエンティティ内の事実を参照しているか、またはその事実と関連付けられているということです。上の例では、ビデオ ショップは「顧客」と「ビデオテープ」に関する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つ以上の「ビデオテープ」をレンタルする」というステートメントで表すことができます。

エンティティと属性の定義

エンティティは、人、場所、物、出来事、または概念など、管理する必要がある情報を表します。より厳密に言えば、インスタンスと呼ばれる個々のオブジェクトの集まりがエンティティになります。あるエンティティで、1つのインスタンス(行)は1つだけ存在します。各インスタンスは、他のすべてのインスタンスと区別できなければなりません。

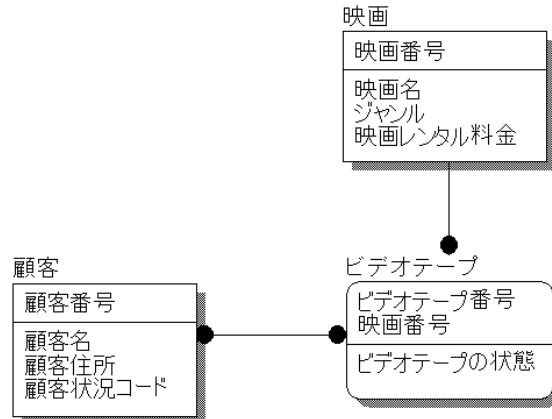
前の図では、「顧客」エンティティは、業務で考えられるすべての顧客の集まりを表します。「顧客」エンティティの各インスタンスは、一人の顧客です。エンティティに関する情報は、次のようなサンプル インスタンス表に一覧表示できます。

「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、エンティティに関する一連の事実を表します。上の表では、「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。各属性は、エンティティに関する個々の情報を表します。

ERD に属性を含めると、次の図に示すように、モデル内のエンティティをより詳しく説明することができます。



論理リレーションシップ

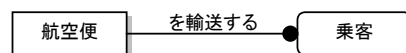
リレーションシップは、エンティティ間の接続、リンク、または関連を表します。また、リレーションシップに動詞句を設定すると、エンティティが互いにどのように関連しているかを示せます。分かりやすい動詞句を指定することで、業務の専門家がデータの制約を検証して、最終的にリレーションシップのカーディナリティを特定する作業が容易になります。

1 対多リレーションシップの例

- 「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 人以上の「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「1 対多」の関係です。つまり、1 番目のエンティティのただ 1 つのインスタンスが、2 番目のエンティティの複数のインスタンスと関連付けられます。「1 対多」の「1」側のエンティティを親エンティティ、「多」側のエンティティを子エンティティと呼びます。

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、線上に動詞句が表示されます。上の例で、山形かっこ内の語が動詞句です(<を輸送する>など)。次の図は、フライトの「航空便」と「乗客」のリレーションシップを表しています。

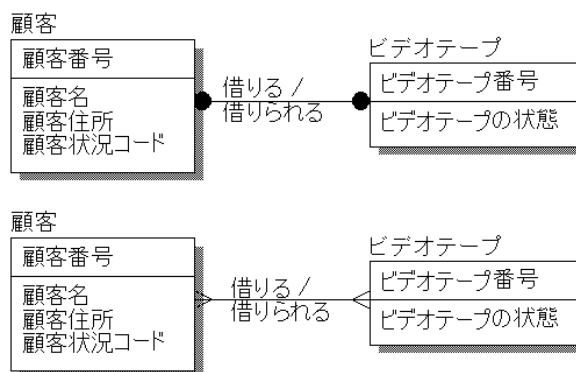


多対多リレーションシップ

多対多リレーションシップは、不特定リレーションシップとも呼びます。これは、1番目のエンティティのインスタンスが、2番目のエンティティの1つ以上のインスタンスと関連付けられており、逆に2番目のエンティティのインスタンスも、1番目のエンティティの1つ以上のインスタンスと関連付けられている状況を示します。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、この多対多リレーションシップは次の事実を示しています。

- 「顧客」が多数の「ビデオテープ」<を借りる>
- 「ビデオテープ」が多数の「顧客」<によって借りられる>

通常、多対多リレーションシップが使用されるのは、ERDなど、基本設計段階のみです。IDEF1X表記法では、両端にドット(●)のある実線で表されます。



多対多リレーションシップは、他のビジネスルールや制約を隠蔽してしまうことがあるため、モデリングプロセスの後の段階で十分に検討する必要があります。たとえば、モデリングの初期段階で指定した多対多リレーションシップが間違っており、実際には、関連するエンティティ間に2つの1対多リレーションシップを指定するのが正しいという場合があります。また、日付やコメントなど、多対多リレーションシップについて追加で情報を保存する必要が生じたときに、多対多リレーションシップを別のエンティティに置き換えて、それらの情報を保存する場合もあります。すべての多対多リレーションシップは、モデリングプロセスの後の段階で十分に検討して、正しくモデル化されているかを確認する必要があります。

論理モデル設計の検証

データモデルが表すビジネスルールは、モデリングの対象となる領域を記述しているため、リレーションシップを読み取ると、その論理モデル設計が正しいかどうかを検証することができます。動詞句が示す内容は、リレーションシップによって表現されたビジネスルールを要約したものとと言えます。動詞句は、ビジネスルールの厳密な記述ではありませんが、エンティティがどのように結びついているかを簡単に理解することができます。

適切な動詞句を指定すると、親から子へのリレーションシップを、能動態(～するという表現)の動詞句で読み取ることができます。

例:

「航空便」は多くの「乗客」<を輸送する>。

リレーションシップは、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)の動詞句を使用します。

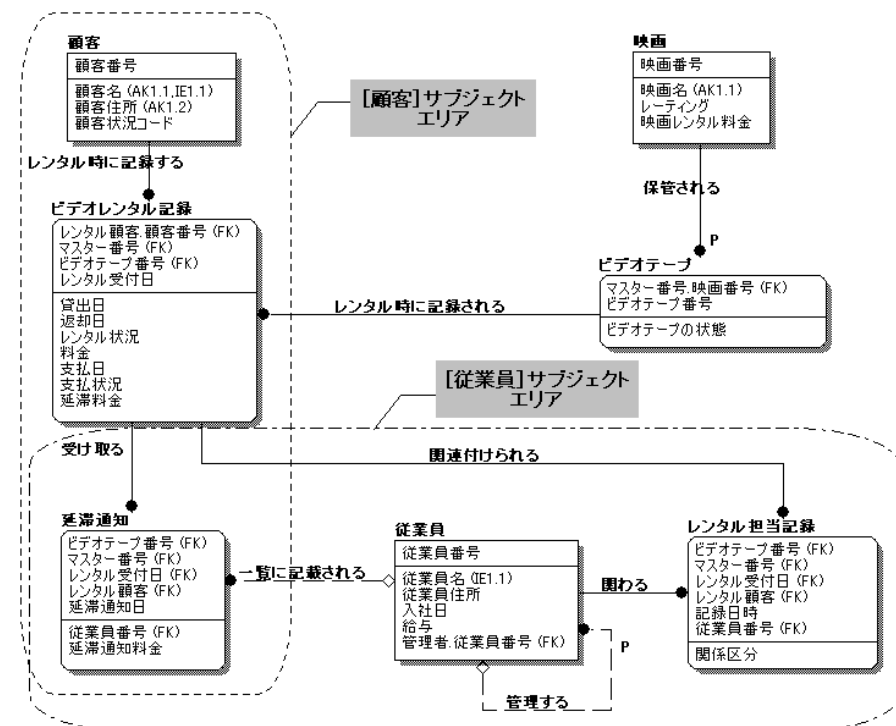
例:

多くの「乗客」は「航空便」<によって輸送される>。

この方法を練習するには、動詞句を使用してモデル内の各リレーションシップを読み取り、有効な文になっているかどうかを確認してみるとよいでしょう。作成したモデルの内容を、ビジネスアナリストや対象分野の専門家に読み取ってもらうことは、ビジネスルールが適切に表現されているかを確認するためによく使われる手法です。

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す。



ビデオショップのデータモデルと、そこに示されたオブジェクト定義から、次の事柄がわかります。

- 「映画」は1つ以上の「ビデオテープ」として保管されます。「映画」ごとに、名前、レーティング、およびレンタル料金についての情報が記録されます。各「ビデオテープ」の状態も記録されます。
- ショップの「顧客」は「ビデオテープ」をレンタルします。「ビデオレンタル記録」には、「ビデオテープ」のレンタル情報が「顧客」ごとに記録されます。日にちが経つと、同一の「ビデオテープ」は複数の「顧客」にレンタルされます。

- 「ビデオレンタル記録」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。これまでの履歴に応じて、「顧客」にクレジットステータスを表すコードが割り当てられます。このコードは、ビデオショップが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、現金のみの支払いに応じるかを示すものです。
- ショップの「従業員」は多数の「ビデオレンタル記録」に関わります。これは「関係区分」によって指定されます。各記録には少なくとも1人の「従業員」が関わっていないければなりません。1人の「従業員」が1日に何度も同じレンタル記録に関わることもあるので、「レンタル担当記録」はさらにタイムスタンプで区別されます。
- 「ビデオテープ」のレンタル時に、延滞料金を徴収する場合があります。ビデオテープの返却を「顧客」に求めるために、「延滞通知」が必要になることがあります。「延滞通知」には「従業員」が記載されることもあります。
- ショップは各「従業員」の給与と住所に関する情報を保管します。「顧客」、「従業員」、および「映画」を、番号ではなく名前を検索する場合があります。

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の業務について多くの事実を表しています。ここから、業務用データベースをどのような構成にすればよいかということが分かります。また、業務自体の概要も把握することができます。このダイアグラムには、さまざまな種類のグラフィックオブジェクトがあります。エンティティ、属性、およびリレーションシップは、他のシンボルと共にビジネスルールを記述します。以降の各章では、それぞれのグラフィックオブジェクトの意味と、AllFusion ERwin DMを使用して論理および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。

第4章 キーベース データ モデル

本章には次のトピックがあります。

[キーベース データ モデル](#) (25ページを参照)

[キーの種類](#) (26ページを参照)

[主キーの選択](#) (26ページを参照)

[代替キー属性](#) (28ページを参照)

[逆方向エントリ属性](#) (28ページを参照)

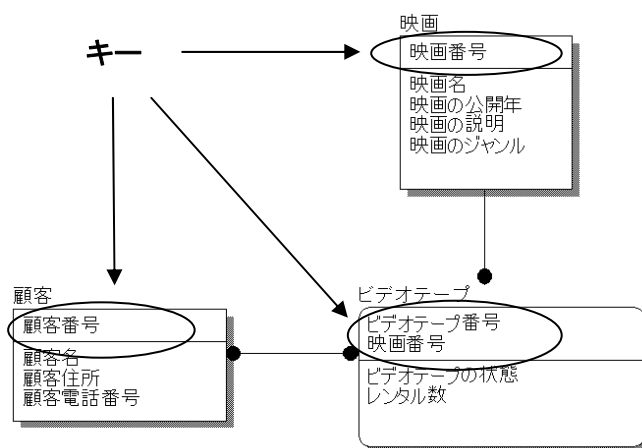
[リレーションシップと外部キー属性](#) (29ページを参照)

キーベース データ モデル

キーベース(KB)モデルは、幅広い業務領域の主なデータ構造を完全に記述したデータモデルです。KBモデルの目的は、すべてのエンティティと業務の対象となる属性を含めることです。

KBモデルは、その名前が示すようにキーを含みます。論理モデルでは、キーはエンティティ内の一意なインスタンスを識別します。キーが物理モデルに実装されると、格納されたデータに簡単にアクセスすることができます。

基本的に、キーベースモデルで対象となる範囲は、エンティティリレーションシップダイアグラム(ERD)と同じですが、より詳しい内容が記述されており、詳細な実装レベルのモデルを構築できるだけの情報が含まれています。



キーの種類

データモデル内にエンティティを作成する場合、最も重要な質問の一つに「一意なインスタンスをどのように識別するか?」というものがあります。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する必要があります。

データモデル内の各エンティティは、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)に分かれています。仕切り線より上のグループがキー領域、下のグループが非キー領域またはデータ領域です。「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。主キーとは、エンティティの一意なインスタンスを識別するための属性セットです。主キーは1つ以上の主キー属性から構成されます。主キー属性として選択された属性は、エンティティの各インスタンスに対して一意な識別子である必要があります。

通常、エンティティには多数の非キー属性が含まれています。これらは、仕切り線の下に表示されます。非キー属性は、エンティティ内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同じ顧客名のインスタンスが複数ある場合、「顧客名」は一意ではないため、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は、慎重に検討した上で選択する必要があります。実際に主キーを選択するには、あらかじめ候補キー属性と呼ばれるいくつかの属性を検討することが必要になる場合があります。通常は、業務および業務データに詳しいユーザーが、候補キーの特定作業を支援してくれます。

たとえば、データモデル(後の段階ではデータベース)内の「従業員」エンティティを正しく使用するには、そのインスタンスを一意に識別できなければなりません。「従業員」エンティティでは、複数の属性から主キーを選択することができます。候補キーとなるのは、従業員名、一意の従業員番号(「従業員」エンティティの各インスタンスに割り当てられる)、または属性の組み合わせ(たとえば、氏名と生年月日)などです。

候補キーの一覧から主キーを選択する場合は、厳密な規則に従います。これはあらゆるタイプのデータベースに適用されます。この規則では、属性または属性グループが次の条件を満たす必要があります。

- 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。インスタンスの識別にはキーを使用するため、キーが変化すると、別のインスタンスになります。
- できるだけ短いものにする。これはインデックス作成と検索の速度を向上させるためです。使用するキーが、他のエンティティから移行されたキーとの組み合わせである場合、構成要素の各キーが他の条件を満たしていることを確認してください。

例:

「従業員」エンティティの候補キーの一覧から、どの属性を主キーに選択するか考えてみましょう。

- 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

先に挙げた規則に従って「従業員」エンティティの候補キーを探すと、各属性について次のような結果が得られます。

- 「従業員番号」はすべての「従業員」に対して一意であるため、候補キーである。
- 同姓同名の従業員がいる可能性があるため、「従業員名」はおそらく候補キーには望ましくない。
- 「従業員社会保障番号」は、ほとんどのインスタンスで一意であるが、すべての「従業員」が社会保障番号を持っているとは限らない。
- 「従業員名」と「従業員生年月日」の組み合わせは、候補キーに使用できる。ただし、同姓同名で生年月日が同じ従業員がいないことが条件となる。
- 賞与の受給資格があるのは、会社の「従業員」の一部のみであり、「従業員賞与額」は多くの場合、NULL となることが予想される。つまり、候補キーとしては使用できない。

検討の結果、2つの候補キー（「従業員番号」と、「従業員名」および「従業員生年月日」の属性グループ）が残りました。「従業員番号」の方が短く、インスタンスの一意性が保証されるため、「従業員番号」を主キーとして選択します。

エンティティの主キーを選択する場合、代理キーを割り当てることがあります。代理キーとは、インスタンスに割り当てられる任意の番号で、エンティティ内のインスタンスを一意に識別します。「従業員番号」は、代理キーの一例です。代理キーは短かく、高速にアクセスでき、各インスタンスの一意性が保証されるため、しばしば主キーとして最適です。また、代理キーはシステムによって自動生成できるため、抜けのない連番を振ることができます。

論理モデルで選択した主キーが、物理モデルのテーブルに効率的にアクセスできるという要件を満たしているとは限りません。物理モデルやデータベースの要件に応じて、主キーはいつでも変更できます。

代替キー属性

候補キーの一覧から主キーを選択すると、残りの候補キーの一部またはすべてを代替キーに指定できます。多くの場合、代替キーは、別のインデックスを指定して高速なデータアクセスを実現するために使用されます。データモデルでは、代替キーは「AKn」という記号で指定されます。nは、代替キーグループを構成する属性の末尾に付けます。次に示す「従業員」エンティティでは、「従業員名」および「従業員生年月日」は、代替キーグループのメンバーです。

従業員番号
従業員名 (AK1.1)
従業員性別
入社日
従業員社会保障番号
従業員生年月日 (AK1.2)
従業員賞与額

逆方向エントリ属性

逆方向エントリは、エンティティへのアクセスに使用されることが多い属性（または属性のセット）ですが、主キーや代替キーとは違い、エンティティ内のただ1つのインスタンスを検索できないことがあります。データモデルでは、逆方向エントリに指定した属性の末尾に「IE」という記号が付きます。

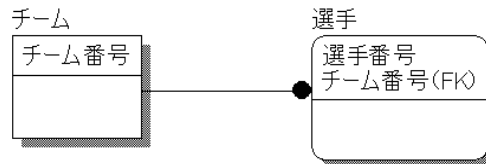
たとえば、従業員データベースの情報を検索する際に、従業員番号の代わりに従業員名で検索したい場合もあるでしょう。名前で検索すると複数のレコードが見つかることがあります。その場合、さらに別のステップを実行して目的のレコードを見つける必要があります。属性を逆方向エントリグループに割り当てると、データベースに非ユニークインデックスが作成されます。

注: 1つの属性は、代替キーグループと逆方向エントリグループの両方に含めることができます。

従業員番号
従業員名 (AK1.1,IE1.1)
従業員性別
入社日
従業員社会保障番号
従業員生年月日 (AK1.2)
従業員賞与額

リレーションシップと外部キー属性

外部キーは、リレーションシップを通じて、親エンティティの主キーとして定義された属性のセットを、親エンティティから子エンティティへ移行したものです。データモデルでは、外部キーに指定した属性名の末尾に「(FK)」という記号が付きまます。次の図では、「チーム番号」の末尾に「(FK)」が付いています。

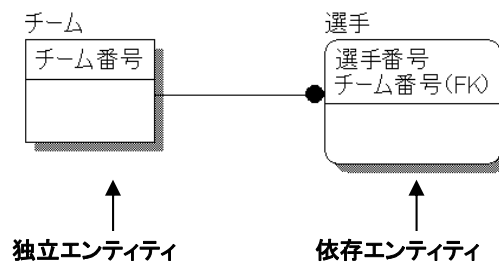


依存エンティティと独立エンティティ

データモデルの作成中に、一意性を保つために外部キー属性値が必要となるエンティティが見つかることがあります。これらのエンティティを一意に定義するには、外部キーが子エンティティの主キーの一部(仕切り線より上のグループ)でなければなりません。

外部キー属性によって一意性を確保する子エンティティを、依存エンティティと呼びます。IDEF1X 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ます。

モデル内の他のエンティティに依存せずに識別できるエンティティを、独立エンティティと呼びます。IE および IDEF1X 表記法では、独立エンティティを角の四角いボックスで表します。

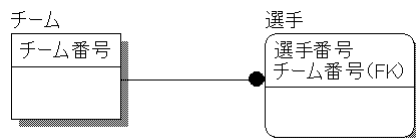


依存エンティティはさらに2種類に分類されます。親がなければ存在できない存在依存と、親のキーを使用しないと識別できない識別依存です。「選手」エンティティは識別依存ですが、存在依存ではありません。「選手」は「チーム」に所属しなくても存在できるためです。

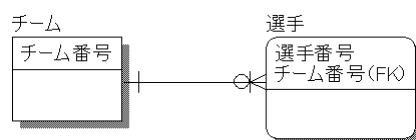
一方、あるエンティティが別のエンティティに対して存在依存である場合もあります。顧客注文の追跡に使用する「注文」、および「注文」の各項目を追跡する「注文明細」という2つのエンティティを考えてみます。2つのエンティティ間のリレーションシップは、「注文」は1つ以上の「注文明細」<を含む>と表すことができます。この場合、「注文明細」は「注文」に対して存在依存です。「注文」がないのに「注文明細」を追跡しても業務上意味がないからです。

依存型リレーションシップ

IDEF1X 表記法では、2つのエンティティ間を接続するリレーションシップの種類によって、依存エンティティと独立エンティティの概念を表します。外部キーを子エンティティのキー領域に移行する(結果として依存エンティティを作成する)には、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します。エンティティ間の実線は、依存型リレーションシップを表します。IDEF1X 表記法では、実線の子エンティティ側にドット(●)が付きます。



IE 表記法では、実線の子エンティティ側に「鳥の足」記号が付きます。



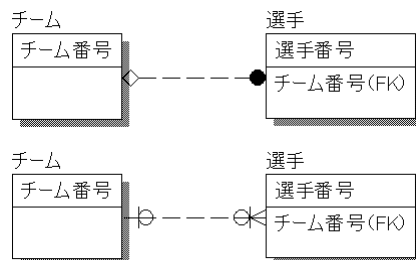
注: 標準の IE 表記法では、エンティティの角は丸くありません。丸い角は IDEF1X 表記法で使用されますが、All Fusion ERwin では 2つの手法間の互換性を確保するために IE 表記法でも使用しています。

依存型リレーションシップを通じて子エンティティにキーを移行すると、一部の DBMS でクエリが簡単になるという利点がある一方で、多くの欠点もあります。先進的なリレーショナル理論の中には、このような方法によるキーの移行を勧めていないものもあります。各エンティティは、その主キーだけでなく、システムのユーザーに不可視の論理ハンドルまたは代理キーによって識別されるべきだという理論です。これについては活発な議論が行われています。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参照してください。

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も、親エンティティと子エンティティを接続します。ただし、2つの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と、外部キーは子エンティティの非キー領域(仕切り線の下側)に移行されます。

エンティティ間の点線は、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と、次のように「チーム番号」が非キー領域に移行します。



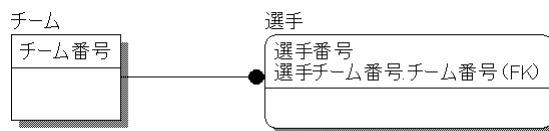
非依存型リレーションシップによって移行されたキーは、子エンティティの主キーには含まれないため、非依存型リレーションシップによって識別依存が生じることはありません。この場合、「選手」は「チーム」と同じく独立エンティティです。

ただし、リレーションシップに関するビジネス ルールで、外部キーに NULL を許可しないように指定している場合は、リレーションシップによって存在依存が反映されることがあります。外部キーの存在が必須である場合、子エンティティのインスタンスが存在できるのは、関連する親インスタンスが存在する場合のみになるからです。

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 手法では使用されません。しかし、AllFusion ERwin DM では IE 手法と IDEF1X 手法の互換性を確保するために、IE 手法でも、これらのリレーションシップを実線と破線で区別して表示します。

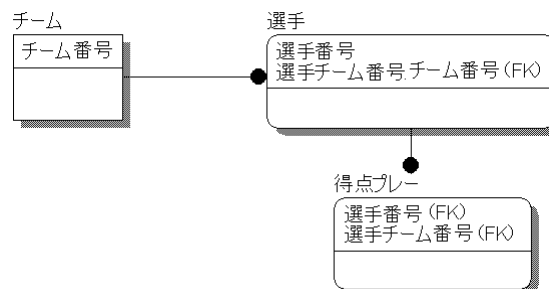
ロール名

外部キーが、リレーションシップの親エンティティから子エンティティへ移行される場合、ビジネス ルールの観点から見ると、その外部キーはモデル内で2つの役割(ロール)を果たしていると言えます。両方の役割を区別して理解できるように、移行キーの名前を変更して、子エンティティでの役割を表すことができます。このようにして、外部キー属性に新たに割り当てられた名前をロール名と呼びます。実際には、ロール名を設定すると新しい属性が宣言されます。ロール名で説明するビジネス ステートメントは、外部キーを移行したリレーションシップによって付加されたものです。



「選手」エンティティの外部キー属性「選手チーム番号.チーム番号」は、定義したロール名が表示されるとき構文を示しています。名前前半(ピリオドより前)がロール名です。名前の後半(ピリオドより後)は外部キーの元の名前です。これをベース名と呼びます。

外部キーに割り当てたロール名は、他の外部キーと同じくリレーションシップを通じて移行されます。たとえば、上の例を拡張して、シーズンを通して各ゲームでどの「選手」が得点したか分かるようにしてみましょう。次の図のように、「選手チーム番号」ロール名が(他の主キー属性と共に)「得点プレー」エンティティに移行します。



注: ロール名は、外部キーと主キーの名前が異なるような古いデータ モデルとの互換性を確保するために使用されることもあります。

第5章 エンティティと属性の名前付けと定義

本章には次のトピックがあります。

[概要](#) (33ページを参照)

[エンティティと属性の名前](#) (33ページを参照)

[エンティティ定義](#) (34ページを参照)

[属性定義](#) (37ページを参照)

[ロール名](#) (38ページを参照)

[定義とビジネス ルール](#) (39ページを参照)

概要

データ モデリングやシステム開発の際には、分かりやすく、よく考えられた名前をオブジェクトに付けることがきわめて重要です。これを徹底することで、業務領域のモデルは明確かつ簡潔で曖昧さのないものになります。

名前付けの基準と規則は、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) ダイアグラムなど、すべての種類の論理モデルで共通です。

エンティティと属性の名前

英語の場合、エンティティ名は常に単数形にします。たとえば、「A FLIGHT <transports> zero , one or more PASSENGERs.」および「A PASSENGER <is transported by> one FLIGHT.」のようになります (大文字がエンティティ名)。エンティティに付ける名前は、各インスタンスの名前にもなります。たとえば、「PASSENGER」エンティティの各インスタンスは個々の passenger です。

属性名も単数形にします。たとえば、「person-name」、「employee-SSN」、および「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という原則に従うと、1つの属性で複数の事実を表しているような正規化エラーを避けることができます。「employee-child-names」や「start-or-end-dates」などの属性は複数形であるため、属性設計における間違いであることが分かります。

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。次のような規則があります。

- 接頭辞は(属性を)修飾する。
- 接尾辞は(属性を)明確にする。

この規則を使用すると、設計を簡単に検証することができ、設計上の問題の多くを解消できます。たとえば、「顧客」エンティティの属性には、「顧客名」、「顧客番号」、「顧客住所」などの名前を付けることができます。ある属性に「顧客請求書番号」という名前を付けようとする場合、接尾辞「請求書番号」は、接頭辞「顧客」をより詳しく説明する情報であるかを確認してみます。すると、規則に従っていないことがわかるため、この属性は、より適切な場所(「請求書」エンティティなど)に移動する必要があることがわかります。

先にエンティティや属性の定義を作成しておく、名前を付けるのが容易になります。一般に、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のと同じく重要な作業です。適切な意味を持つ名前を付けるには、経験に加えて、モデルが表す内容を理解している必要があるからです。

データモデルには業務の内容が記述されるので、業務で使用されている用語から名前を選択するのが最適です。エンティティに対応する業務上の用語がない場合、モデル内での役割に合った名前をエンティティに付ける必要があります。

同義語、同音異義語、およびエイリアス

常に共通の用語が使用されるとは限りませんし、常に的確な名前が使用されるとは限りません。データモデルでは、エンティティと属性を名前で識別するので、同義語(シノニム)があればそれらを解決して、重複したデータを表すことがないようにする必要があります。重複したエンティティや属性を正しく定義し直して、モデルを読み取るユーザーが、各エンティティに記述された事実を正しく理解できるようにします。

エンティティや属性が表すものがはっきりとわかるような名前を選択することも重要です。たとえば、「人」、「顧客」、および「従業員」には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、それぞれ異なる特徴や性質を持っています。ただし、「人」と「従業員」が異なるものであるか、または同じものに対する同義語であるかを決定するのはビジネスユーザーの役割です。

名前の選択には十分注意して、異なるものに同じ名前を付けないようにしてください。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に対して、顧客の名前に関する強制はしないようにします。エイリアス(同じものを表す別名)や、他の「もの」と似ているが実は新しい「もの」が見つかることがあります。この場合、おそらく「消費者」は「顧客」カテゴリに属しており、他の「顧客」カテゴリには存在しないリレーションシップに関与しています。

モデリング環境では、一意な名前のみを許可するように設定することができます。これによって、同音異義語(ホモニム)、曖昧な名前、またはモデル内で重複したエンティティや属性を誤って使用することを回避できます。

エンティティ定義

明確なモデルを作成するには、論理モデル内のエンティティを定義することが不可欠です。また、エンティティの目的を詳しく説明したり、エンティティに含めたい事実を明確にするためにも定義を活用できます。未定義のエンティティや属性は、後のモデリング作業で間違っ​​て解釈され、結果として削除されたり、他のエンティティまたは属性と統一されてしまうことがあります。

優れた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」とは何かということや、皆が説明できるでしょうか？ 試しに、精査に耐えるような「顧客」の定義を書いてみてください。優れた定義を作成するには、組織内のさまざまなビジネスユーザーや業務グループの視点が必要になります。さまざまなユーザーの精査に耐えるような定義を作成すると、次のような利点が得られます。

- 企業全体にわたって業務内容が明確になる
- 各事実の意味について合意が得られる
- データの種類を容易に識別できる

ほとんどの組織や個人は、定義を作成する上で独自の決まりや基準を作成します。実際には、定義される「もの」を読み手が理解しやすいように、長い定義が作成されていることが多いでしょう。たとえば、複数ページに渡るような定義（「顧客」に関する定義など）もあります。IDEF1X および IE 手法では、定義に関する基準を定めていないため、定義を作成するときの基本的な基準として次の項目を採用するとよいでしょう。

- 説明
- 業務の例
- コメント

説明

説明は、分かりやすく簡潔な文章で、定義しようとしているオブジェクトについて記述します。一般に、説明は短い方がよいでしょう。一般的な内容になりすぎないように、また、定義されていない用語を使用しないように注意してください。次に、適切な例と不適切な例を示します。

適切な説明の例

「商品」とは、交換できる価値を持つ何かである。

これは良い説明です。何かを「商品」と交換したい人は、これを読むと何が「商品」かわかるからです。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。

不適切な説明の例

「顧客」とは、当社から何かを購入する誰かである。

これは良い説明ではありません。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てしまいます。また、すでに自社から何かを購入した人だけでなく、見込みの「顧客」を追跡したい場合もあるかもしれません。さらに、「何か」をより詳しく定義して、製品、サービス、または両者の組み合わせのうちどれを販売するのかを説明することもできます。

業務の例

定義の対象について、典型的な業務上の例を挙げることは良い考えです。良い例は、定義を理解する上で大いに役立つからです。ピーナッツやビー玉、または業務に関係するものを例に挙げて説明することで、読み手が「商品」の概念を理解しやすくなります。定義では、商品に価値があることを述べています。具体的な例を挙げると、価値が常に金額で表されるとは限らないことを示すことができます。

コメント

定義には、さまざまなコメントを入れることもできます。たとえば、定義の責任者、作成者、現在の状態、および最終更新日時などです。定義するエンティティによっては、関連するエンティティとの相違点について説明する必要があります。たとえば、「顧客」と「見込客」は区別されることを記述します。

定義の参照と循環

個々の定義は問題ないように見えても、まとめて確認すると定義が「循環」していることがあります。エンティティと属性の定義では、注意を怠るとこのような問題が起こることがあります。

例:

- 「顧客」: 当社の「製品」を1つ以上購入する人
- 「製品」: 当社が「顧客」に販売するもの

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際は、共通の業務用語を使用すると便利です。たとえば、「通貨スワップ」とは、ある期間にわたり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」間の契約である。為替手数料は、スワップの**期間**を通して固定されるか、または**変動**することがある。スワップは、しばしば**通貨と金利のリスク**をヘッジするために使用される。」という例について考えてみます。

この定義例では、定義済みの用語が強調表示されています。このようなスタイルにすると、用語が使用されるたびに定義する必要がなくなります。用語の定義をすぐに見つけることができるからです。

エンティティや属性の名前以外で、共通の業務用語を使用すると便利な場合は、それらの用語の基本定義をあらかじめ作成しておき、必要に応じて参照するとよいでしょう。一般的な用語についての用語集であれば、特定のモデルに限らず使用することができます。このような業務用語は、上の例で示したように太字で強調表示します。

この方法は、一見するとさまざまな定義の間を何度も行き来することになると思われるかもしれませんが、もう1つの方法は、それぞれの用語が使用されるたびに定義することです。しかし、このような内部定義がたくさん場所に作成されると、そのすべてを管理する必要があります。ある定義が変更されたときに、その定義が記述されたすべての場所を同時に修正するのは非常に困難になります。

共通の業務用語集を作成すると、さまざまな目的に利用できます。モデリング定義の基盤として使用できるだけでなく、実際の業務における意思疎通を支援するという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。エンティティの定義と同じ基準を使用して、属性を正しく説明する定義を作成してください。不完全な定義とならないように注意が必要です。

例:

「口座開設日」

「口座」が開かれた日付です。より完全な定義にするには、「開かれた」の意味をさらに定義する必要があります。

一般に、属性定義の基本構造はエンティティ定義と同じであり、説明、例、およびコメントを含みます。可能であれば、定義にバリデーション ルールも含めて、その属性の有効値としてどのような事実を受け入れるかを指定します。

バリデーション ルール

バリデーション ルールは、属性が取ることができる値のセットで、受け入れ可能な値の範囲を制限します。これらの値には、抽象的な意味と業務上の意味があります。たとえば、「個人名」という属性が、「個人」によって選択された敬称として定義された場合、受け入れ可能な値は文字列のみに制限されます。属性定義の一部として、属性のバリデーション ルールまたは有効値を定義できます。ドメインを使用して、これらのバリデーション ルールを属性に割り当てることができます。

コード、識別子、または数量といった属性を定義する際に、業務の例として適切なものが見当たらないことがあります。その場合は、属性のバリデーション ルールや有効値の説明を定義に含めるのが良いでしょう。バリデーション ルールを定義する際には、属性が取りうる値の一覧を並べるだけでなく、詳しい説明を付けることをお勧めします。例として、「顧客状況」という属性を定義してみましょう。

「顧客状況」: 「顧客」と当社の業務との関係を記述するコード。有効値: A、P、F、N

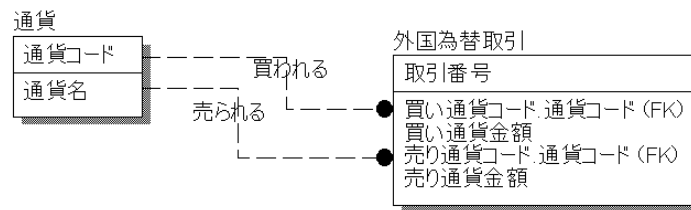
バリデーション ルールを指定するだけでは、コードの意味が定義されないため、その意味を理解するのに役に立ちません。次のような値の一覧を使用すると、バリデーション ルールの内容を分かりやすく説明できます。

有効値	意味
A: 現在の顧客 (Active)	現在当社と取引関係を持つ「顧客」。
P: 見込み客 (Prospect)	当社は関係構築に興味があるが、まだ取引関係にない人物。
F: 過去の顧客 (Former)	取引関係が失われた「顧客」。つまり、過去 24 か月間に販売実績がない。
N: 取引関係を持たない (No Business accepted)	取引関係を持たないことに決めた「顧客」。

ロール名

ある外部キーが、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たものである場合、その外部キー属性の定義を新しく作成したり、拡張しなければならないことがあります。外部キー属性が子エンティティでどのように使用されるのかを説明する必要があるためです。そのような場合、ロール名を割り当てることができます。同じ属性が同じエンティティに複数回移行されるような場合も同様です。これらの重複した属性は同一に見えますが、2つの役割を持つため、同じ定義を使用すべきではありません。

次の図に示す例を考えてみましょう。「外国為替取引」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あります。



「通貨」エンティティの主キーは「通貨コード」です（これは、追跡対象となる有効な「通貨」の識別子です）。リレーションシップを確認すると、「外国為替取引」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」）を使用して、2つの「通貨」それぞれが識別されます。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。これらのロール名は、両方の属性が「通貨コード」と同一ではないことを示します。

ある「通貨」を同じ「通貨」と売買するのは不自然です。したがって、取引（「外国為替取引」のインスタンス）では、「買い通貨コード」と「売り通貨コード」が異なっている必要があります。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」の一意的識別子
「買い通貨コード」	「外国為替取引」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」	「外国為替取引」によって売られた「通貨」の識別子（「通貨コード」）

買い通貨コードと売り通貨コードの定義とバリデーション ルールは、「通貨コード」に基づいて作成されます。「通貨コード」はベース属性と呼ばれます。

IDEF1X 標準では、同じ名前を持つ 2つの属性が同じベース属性から 1つのエンティティへ移行する場合、それらの属性は一意化しなければなりません。一意化の結果、2つのリレーションシップを通じて移行された 1つの属性が残ります。IDEF1X 標準により、外部キー属性も自動的に一意化されます。移行属性を一意化したくない場合、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に、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 ルールは、データ モデルに不可欠な要素の一つです。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、および他のモデリング構造(汎化カテゴリ、参照整合性、およびカーディナリティなど)を使用して表されます。また、ビジネス ルールは、エンティティと属性の定義、およびバリデーション ルールで表されることもあります。

たとえば、「通貨」エンティティは、世界中で認められているすべての有効な通貨セットか、または会社が業務で使用することを定めている通貨のサブセットとして定義できます。この違いはわずかですが、重要なものです。後者の場合は、関連するビジネス ルール、またはポリシー ステートメントがあるでしょう。

このルールは、「通貨コード」のバリデーション ルールに明示され、「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。したがって、このビジネス ルールの管理は、「通貨」の有効値テーブルを管理する作業になります。「通貨」の取引を許可または禁止するには、有効値テーブル内のインスタンスを作成または削除するだけです。

「買い通貨コード」と「売り通貨コード」という属性も同様に制限されます。これらの属性は、「買い通貨コード」と「売り通貨コード」は異なっている必要がある、というバリデーション ルールでさらに制限されます。そのため、実際の使用時に、一方の値はもう一方の値に依存します。バリデーション ルールは、属性の定義に記述することができますが、バリデーション ルール、デフォルト値、および有効値の一覧を使用して、明示的に定義することもできます。

第6章 リレーションシップの作成

本章には次のトピックがあります。

[リレーションシップ](#) (41ページを参照)

[リレーションシップのカーディナリティ](#) (41ページを参照)

[参照整合性](#) (44ページを参照)

[その他のリレーションシップ タイプ](#) (49ページを参照)

リレーションシップ

リレーションシップは、見た目よりもやや複雑に思えるかもしれませんが。リレーションシップにはさまざまな情報が含まれており、データモデルの中核であるという人もいます。ビジネスルールの記述から、インスタンスの作成、変更、および削除時における制約の記述までカバーするからです。たとえば、カーディナリティを使用して、リレーションシップの子エンティティと親エンティティ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、参照整合性ルールを使用して、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複雑なリレーションシップの種類もサポートしているため、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を構築することができます。

リレーションシップのカーディナリティ

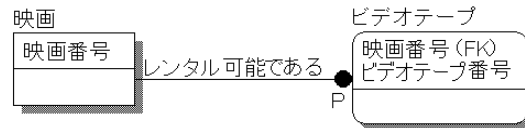
1 対多リレーションシップにおける「多」という概念は、親に接続された子のインスタンスが複数でなければならないという意味ではありません。親と対となる子のインスタンスが 0 個、1 個、またはそれ以上あるという意味です。

カーディナリティは、親テーブルの各インスタンスに対応する、子テーブルのインスタンス数を明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、カーディナリティ指定に異なる記号が使用されます。ただし、どちらの手法でも、次の表に示すように、1 以上、0 以上、0 または 1、または定数 N を示す記号があります。

カーディナリティの説明	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」				

カーディナリティの説明	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」				
「0 または 1」対「0 または 1 以上」 (非依存型のみ)				
「0 または 1」対「0 または 1」 (非依存型のみ)				

カーディナリティを指定すると、リレーションシップに追加のビジネス ルールを適用することができます。次の図では、各「ビデオテープ」を識別するのに、外部キー「映画番号」と代理キー「ビデオテープ番号」の両方が使用されます。また、それぞれの「映画」は 1 本以上の「ビデオテープ」として利用できます。さらに、依存型リレーションシップによって、「ビデオテープ」が存在するには、対応する「映画」が必要であることを表しています。



この「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の線の種類から、リレーションシップに関与するのが 1 つの「映画」だけであることが分かります。「映画」は、このリレーションシップの親であるからです。

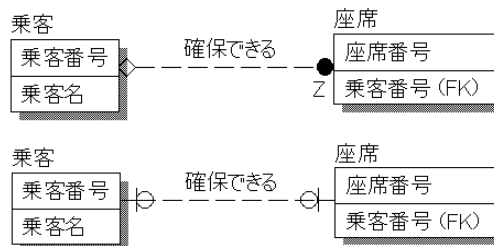
「ビデオテープ」をリレーションシップの子 (IDEF1X ではラインの端にドット (●) が付く) に指定したため、「ビデオテープ」は、ある映画タイトルの複数ある貸出用テープの 1 つとして定義されました。「映画」には少なくとも 1 本の「ビデオテープ」が必要であるという情報も含まれています。したがって、<レンタル可能である>リレーションシップのカーディナリティは、「1」対「1 以上」になります。ドット (●) の横の「P」記号は「1 以上」というカーディナリティを表します。ビデオテープが存在しない「映画」は、このデータベースのインスタンスとして許可されることがわかります。

このビデオ ショップでは、ビデオテープが存在しない「映画」も含め、世界中の「映画」についての情報を登録したいかもしれません。そのようなビジネス ルールは、「映画」が存在する (情報システムに記録される) には、0 または 1 つ以上のビデオテープが必要である」となります。「P」記号を取り除くことで、このビジネス ルールを表すことができます。依存型リレーションシップの場合、カーディナリティ記号 (P または Z) が表記されない場合、カーディナリティは「1」対「0 または 1 以上」になります。

非依存型リレーションシップのカーディナリティ

非依存型リレーションシップでも、親エンティティから子エンティティにキーが移行されます。ただし定義により、親のキーの一部(またはすべて)は、子のキーの一部になることはありません。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(存在依存ではない)状況が生じることがあります。

このリレーションシップが子から見て必須の場合、子は親に対して存在依存となります。このリレーションシップが必須ではない場合、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に依存している可能性はあります)。任意であることを示すために、IDEF1Xではリレーションシップの親側の端にひし形(◇)が付き、IEでは丸(O)が付きます。



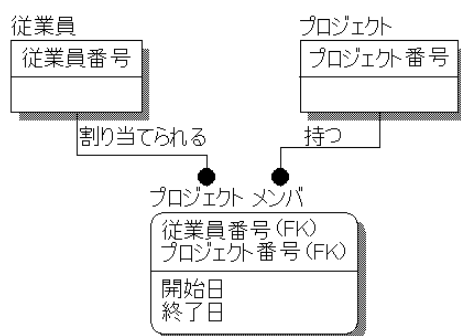
この図では、「乗客番号」属性は「座席」の外部キー属性となります。「乗客番号」は「座席」を識別するのではなく、「座席」を占める「乗客」を識別するので、リレーションシップは非依存型になります。「乗客」がいなくても「座席」は存在できるので、このリレーションシップは任意であることもわかります。リレーションシップが任意の場合、IDEF1X表記ではひし形(◇)、IE表記では丸(O)がダイアグラムに表示されます。リレーションシップが必須の場合、非依存型リレーションシップのカーディナリティ表示は依存型リレーションシップと同じになります。

リレーションシップのカーディナリティが、IDEF1Xでは「Z」、IEでは縦線(|)で表示されている場合、「乗客」が0または1つの「座席」を<確保できる>ことを示します。「座席」が確保されている場合、座席を確保した「乗客」は「乗客番号」によって識別されます。「座席」が確保されていない場合、「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 データベースでは、データ値に基づいてリレーションシップが実装されるため、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 カラムの値を変更する場合、そのカラムが外部キーであるすべての子テーブルにも同じ変更が反映されることを知っておく必要があります。外部キーに適用されるアクションは、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、次の図のようなモデルで従業員とプロジェクトの情報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は依存型であり、「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っています。



さらに「プロジェクト メンバ」の各インスタンスに対して、「プロジェクト」のインスタンスが 1 つだけ存在します。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」の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」のインスタンスが不要になる場合には、削除された「プロジェクト」から一部のキーを継承している「プロジェクト メンバ」のインスタンスも、すべて削除する必要があります。

親キーが削除されたときのアクションを指定するルールを、参照整合性と呼びます。上記のリレーションシップで、このアクションに選択された参照整合性オプションは CASCADE です。「プロジェクト」のインスタンスが削除されるたびに、「プロジェクト メンバ」テーブルに対してこの削除アクションが CASCADE され、「プロジェクト メンバ」のすべての関連インスタンスも削除されます。

参照整合性で利用できるアクションには次のようなものがあります。

CASCADE

親エンティティのインスタンスが削除されるたびに、子エンティティの関連インスタンスも削除されます。

RESTRICT

子エンティティに関連インスタンスが1つ以上ある場合、親エンティティのインスタンスの削除が禁止されます。または、親エンティティに関連インスタンスがある場合、子エンティティのインスタンスの削除が禁止されます。

SET NULL

親エンティティのインスタンスが削除されるたびに、子エンティティの各関連インスタンスにある外部キー属性が NULL に設定されます。

SET DEFAULT

親エンティティのインスタンスが削除されるたびに、子エンティティの各関連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>

参照整合性のアクションは必要ありません。必ずしもすべてのアクションに参照整合性ルールを関連付ける必要はありません。たとえば、子エンティティのインスタンスを削除するときに、参照整合性が不要な場合があります。これが有効なビジネスルールとなるのは、カーディナリティが「0 または 1」対「0 または 1 以上」の場合です。子エンティティのインスタンスが、親エンティティの関連インスタンスがなくても存在できるからです。

参照整合性は、IDEF1X や IE 表記法で正式に定義されているわけではありませんが、完成したデータベースの動作を表すビジネスルールを記述するため、データモデリングの重要な要素の一つです。このため、AllFusion ERwin DM では、参照整合性ルールを記述および表示できるようになっています。

参照整合性が定義されたら、進行役またはアナリストは、ビジネスユーザーが定義した参照整合性ルールをテストする必要があります。このために、さまざまな質問をしたり、業務上の意思決定をもたらす各種シナリオを通して検証します。要件を定義し、完全に理解した上で、進行役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提案することができます。

参照整合性オプション

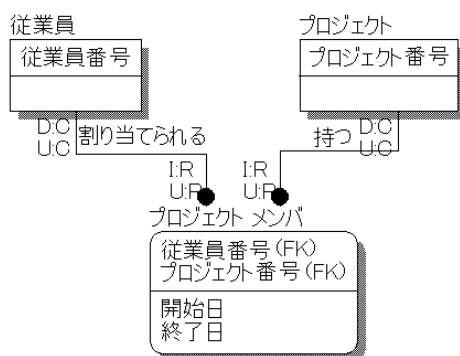
参照整合性ルールは、次のような要因によって変化します。

- エンティティがリレーションシップの親と子のどちらであるか
- 実装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

次の図に、「従業員」-「プロジェクト」モデルの参照整合性ルールを示します。

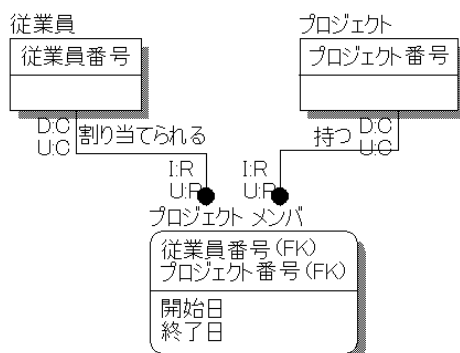


図に表された参照整合性ルールは、「プロジェクト」エンティティ内のすべての削除操作を「プロジェクトメンバ」エンティティにカスケードするという業務上の意思決定を示しています。このルールは PARENT DELETE CASCADE と呼ばれ、この図では、指定されたリレーションシップの親側にある「D:C」という文字で示されます。参照整合性を表す記号の1番目の文字は、常にデータベース操作 (I - INSERT、U - UPDATE、または D - DELETE) を示します。2番目の文字は、参照整合性オプション (C - CASCADE、R - RESTRICT、SN - SET NULL、および SD - SET DEFAULT) を示します。

この図では、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:) の参照整合性は親側には表示されていません。

参照整合性、カーディナリティ、および依存型リレーションシップ

次の図では、「プロジェクト」と「プロジェクトメンバ」間のリレーションシップは依存型です。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。削除されていないインスタンスがある場合、DELETE 操作が制限されます。

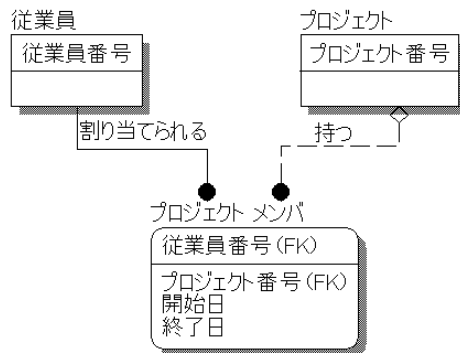
削除を制限する理由の1つとして、「プロジェクトメンバ」に関する他の事実（プロジェクト参加の開始日など）を知りたい場合などがあります。DELETE 操作をカスケード (CASCADE) した場合、このような補足情報は失われます。

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インスタンスにもカスケードされるよう指定されています。

この例を見ると分かるように、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は、異なるルールが適用されます。たとえば、インスタンスを挿入する場合、アクションは RESTRICT に設定されます。このルールは、リレーションシップの子エンティティ側にある「I:R」で表されます。これは、子エンティティにインスタンスを追加できるのは、外部キーが親エンティティの既存インスタンスと一致する場合のみであることを意味します。したがって、「プロジェクトメンバ」に新しいインスタンスを挿入できるのは、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です。

参照整合性、カーディナリティ、および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」に対して存在依存でも識別依存でもない場合、「プロジェクト」と「プロジェクトメンバ」間のリレーションシップを任意で非依存型に変更することができます。非依存型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。



非依存型リレーションシップを通じて移行される外部キーには NULL を設定可能であるため、PARENT DELETE に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」の関連インスタンスで「プロジェクト」から継承された外部キーが NULL に設定されることを示します。DELETE 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、事実上「プロジェクトメンバ」と「プロジェクト」間の接続を解除できることです。

CASCADE と SET NULL のどちらを使用するかは、外部キーによって表されたリレーションシップの履歴情報をどのように管理するかという業務上の意思決定に基づきます。

その他のリレーションシップ タイプ

論理モデルを作成する際に、標準の 1 対多リレーションシップに当てはまらない親/子のリレーションシップが見つかることがあります。このような例外的なリレーションシップには、次のようなものがあります。

多対多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>」リレーションシップです。たとえば、「従業員」は 1 つ以上の「役職」を<持ち>、かつ「役職」には 1 人以上の「従業員」が<割り当てられる>といったリレーションシップです。

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 対多リレーションシップを、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を持ちます。たとえば「従業員」エンティティに、「1 人の「従業員」が 1 人以上の「従業員」を<管理する>」ことを示すリレーションシップを含めることができます。このタイプのリレーションシップは、部品表の構成時に部品間のリレーションシップを表すために使用されることもあります。

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グループ化して、すべての共通属性を 1 つのエンティティに表示し、共通しない属性を個別の関連するエンティティに表示します。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。

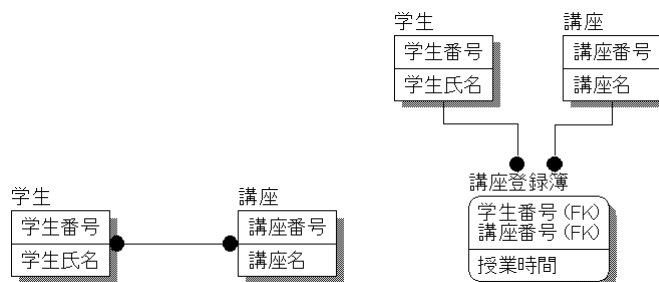
多対多リレーションシップ

キーベース モデルおよび全属性モデルでは、リレーションシップによって親エンティティの 0 または 1 つのインスタンスを、子エンティティの特定のインスタンス セットに関連付ける必要があります。このルールによって、エンティティリレーションシップ ダイアグラムや初期のモデリング段階で表された多対多リレーションシップは、一組の 1 対多リレーションシップに分解する必要があります。



この図は、「学生」と「講座」の間の多対多リレーションシップを示しています。「講座」と「学生」の間の多対多リレーションシップを解消しないと、「講座」のキーが「学生」のキーに含まれ、「学生」のキーが「講座」のキーに含まれてしまいます。「講座」および「学生」は共に自分自身のキーによって識別され、無限ループになります。

多対多リレーションシップを解消するには、関連エンティティを作成します。次の図では、「講座登録簿」エンティティを追加して、「学生」と「講座」間の多対多リレーションシップが解決されています。

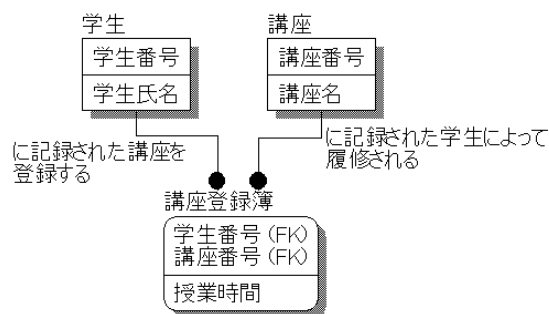


「講座登録簿」は関連エンティティです。これは、2つの関連するエンティティ間のつながりを定義するために使用されます。

多対多リレーションシップでは、しばしば2つのエンティティがどのように関連しているかという意味を隠蔽してしまうことがあります。多対多リレーションシップを含むダイアグラムでは、「学生」が複数の「講座」に登録することがわかっていても、どのように登録するかという情報は含まれていません。多対多リレーションシップを解決すると、エンティティ間の関係だけでなく、リレーションシップについての事実を説明する「授業時間」などの情報も明らかになります。

多対多リレーションシップが解決されたら、リレーションシップに動詞句を含めて構造を検証する必要があります。動詞句を含める方法は2つあります。新しい動詞句を作成する方法と、多対多リレーションシップで指定した動詞句を使用する方法です。最も簡単な方法は、関連エンティティを介して多対多リレーションシップの動詞句を読み取ることです。つまり、「1人の「学生」が複数の「講座」に<登録する>、かつ1つの「講座」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。モデル作成者の多くは、このスタイルでモデルの構築と読み取りを行います。

もう1つの方法はやや面倒かもしれませんが、モデルの構造はまったく同じですが、動詞句が異なり、モデルを読み取る方法も少し異なります。

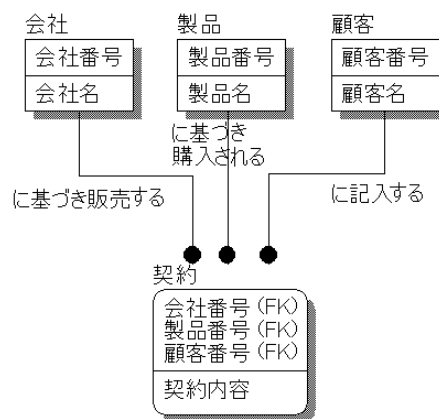


「学生」が1つ以上の「講座登録簿」<に記録された講座を登録する>、かつ「講座」は1つ以上の「講座登録簿」<に記録された学生によって履修される>と読み取ります。動詞句は長くなりますが、この方法は親エンティティから子へと直接読み取る標準パターンに従っています。

どちらのスタイルを選択する場合も、一貫性が重要です。多対多リレーションシップに動詞句を含める方法を決定することは、このように比較的単純な構造ではあまり難しくありません。しかし、関連エンティティに接続されたいずれかのエンティティも関連エンティティ(他の多対多リレーションシップを表すために存在する)である場合など、より複雑な構造では難しくなることがあります。

多項リレーションシップ

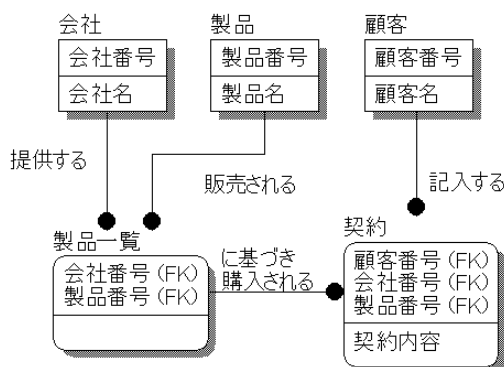
1つの親/子リレーションシップがある場合、そのリレーションシップは2項リレーションシップと呼びます。これまで取り上げたリレーションシップに関するすべての例は、2項リレーションシップです。しかし、データモデルを作成する際、多項リレーションシップが見つかることは珍しくありません。これは、複数の親エンティティと1つの子テーブル間のリレーションシップに対するモデリング上の名前です。多項リレーションシップの例を次の図に示します。



多対多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において有効な構造です。多項リレーションシップも、多対多リレーションシップと同じく、後のモデリング段階で解決する必要があります。これには、関連エンティティと2項リレーションシップのセットを使用します。

この図に示されたビジネスルールを検討すると、「契約」は「会社」、「製品」、および「顧客」の間の3方向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」に販売す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、解決すべき業務上の疑問が生じてきます。たとえば、「会社は販売前に製品を提供する必要はあるか?」「顧客は複数企業の製品を含む単一の契約を結ぶことができるか?」「各顧客がどの会社に『属している』かを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、モデルの構造が変わります。

たとえば、会社が販売前に製品を提供する必要がある場合、次の図に示すような構造に変更する必要があります。



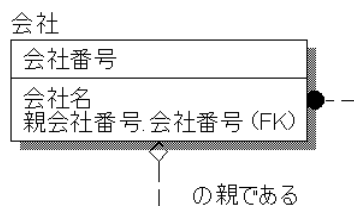
「製品」は「会社」が提供するため、関連エンティティを作成して、このリレーションシップを表すことができます。「契約」に対する元の3方向リレーションシップは、2つの2方向リレーションシップに置き換えられます。

さまざまな業務上の疑問を検討することによって、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と一連の2項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

エンティティがリレーションシップの親と子の両方を兼ねる場合、このエンティティは再帰リレーションシップを持ちます。このリレーションシップが重要となるのは、IMS や IDMS などの古い DBMS に格納されていたデータをモデリングする場合です。たとえば、部品表の構造を実装するために再帰リレーションシップが使用されます。

下の図の例では、「会社」は他の「会社」の親になることができます。非依存型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表示されます。



「会社」に対する再帰リレーションシップには、「会社」に親会社がない場合など、外部キーが NULL 値を取れることを示すひし形(◇)の記号が付きます。再帰リレーションシップは、任意 (NULL を許可) および非依存型である必要があります。

「会社番号」属性は再帰リレーションシップを通じて移行されます。上の例ではロール名「親会社番号」で表示されていますが、これには2つの理由があります。第1に、一般的な設計ルールとして、同じ名前の属性は同じエンティティ内に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確立するには、移行された属性のロール名を用意する必要があります。

第2に、キー領域にある「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を通じて移行された「会社番号」(親「会社」を識別する)と同じではありません。これらの属性に同じ定義は使用できないので、移行属性にはロール名を付ける必要があります。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。

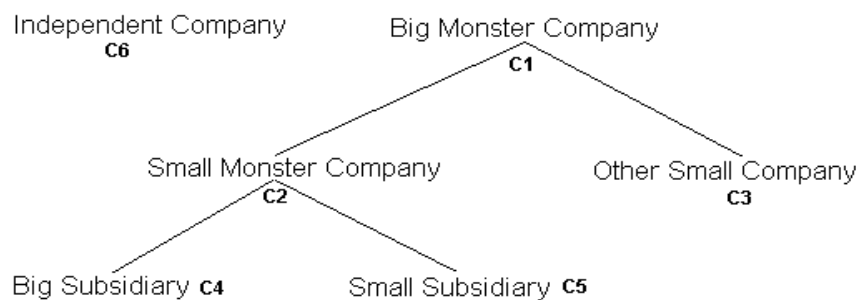
親会社番号

親「会社」の「会社番号」。ただし、すべての「会社」が親「会社」を持つわけではない。

次のようなサンプル インスタンス表を作成すると、リレーションシップのルールをテストして、その妥当性を確認することができます。

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

このサンプル インスタンス表は、「Big Monster Company」が「Smaller Monster Company」と「Other Smaller Company」の親であることを示しています。同様に、「Smaller Monster Company」は、「Big Subsidiary」と「Small Subsidiary」の親です。「Independent Company」は、他の会社の親ではなく、また自身の親を持ちません。「Big Monster Company」も親を持ちません。この情報を次のように階層的に図示してみると、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。



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。たとえば次の図に示すように、モデリング作業中に、銀行に複数の「口座」(当座、普通、借入など)があることがわかったとします。

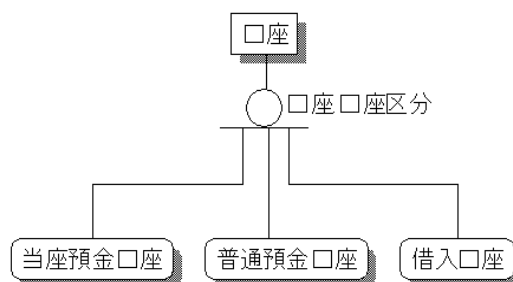
当座預金口座	普通預金口座	借入口座
当座預金口座番号	普通預金口座番号	借入口座番号
当座預金口座開設日	普通預金口座開設日	借入口座開設日
当座預金照会日	普通預金照会日	借入照会日
当座預金残高	普通預金残高	借入総額
利用可能残高	普通預金利息	借入金利
小切手手数料	普通預金受取利息	借入残高

それぞれ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、より高レベルのエンティティに移動することができます。各口座に固有の属性は、サブタイプ エンティティに残ります。この例では、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。スーパータイプ「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用して、3つのサブタイプ エンティティである「当座預金口座」、「普通預金口座」、および「借入口座」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、次の図のような構造になります。



この図では、「口座」は「当座預金口座」、「普通預金口座」、または「借入口座」のいずれかになります。各サブタイプ エンティティは「口座」の種類を表し、「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 エンティティは互いに排他的です。

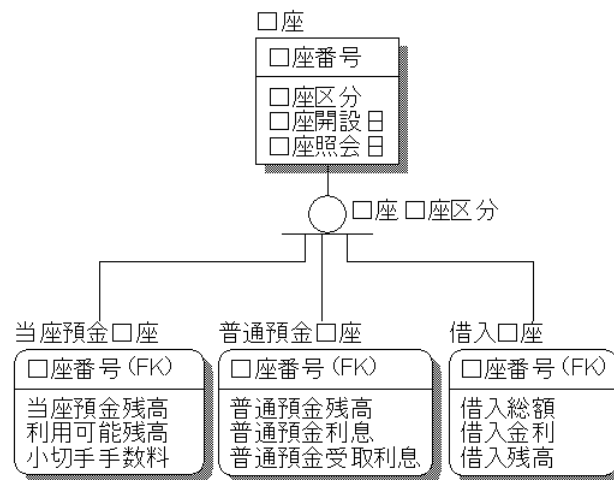
「口座」の種類を区別するために、「口座区分」という属性をサブタイプ識別子として追加します。サブタイプ識別子は、スーパータイプ(「口座」)の属性です。その値から、どの「口座」であるかがわかります。

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て、サブタイプ エンティティに残るか、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 エンティティには「口座開設日」があります。3種類の「口座開設日」の定義が同じ場合、これらをスーパータイプに移動して、サブタイプ エンティティから取り除くことができます。

各属性は順番に分析して、サブタイプ エンティティに残すか、スーパータイプ エンティティに移動するかを判断する必要があります。1つの属性が一部のサブタイプ エンティティのみにある場合、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。このような属性をスーパータイプに移動すると、スーパータイプの属性が対応するサブタイプ エンティティに含まれていない場合、その値は NULL になります。

どちらの方法を選択するかは、いくつかのサブタイプ 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、高レベルのモデルでは、その属性をスーパータイプに移動するのがよいでしょう。属性を共有しているサブタイプ エンティティが少ない場合は、そのままにしておくのが最適です。物理寄りのモデルでは、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、次のようなモデルになりました。

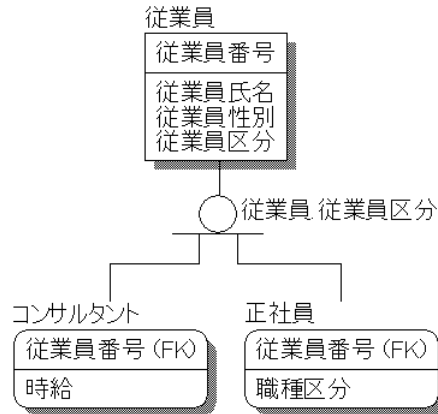


サブタイプ リレーションシップを作成する場合、スーパータイプの下の特定のサブタイプ エンティティのみに適用されるビジネス ルールにも注意する必要があります。たとえば、「借入口座」は借入残高がゼロになった時点で閉じられます。しかし、同じ条件で「当座預金口座」や「普通預金口座」を閉じることはないでしょう。

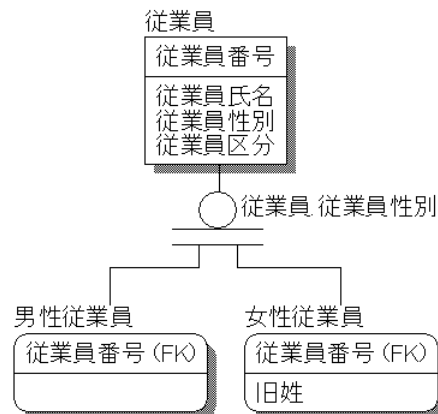
階層内の1つのサブタイプにのみ関与するリレーションシップが存在する場合があります。たとえば、「借入口座」エンティティには、顧客の支払いや資産の記録との間に作成されたリレーションシップが存在する可能性があります。

確定型および未確定型サブタイプ構造

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、異なる記号を使用します。未確定型サブタイプが使用されている場合、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。次の図に示すように、未確定型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。



確定型サブタイプは、サブタイプ構造において考えられるすべてのサブタイプエンティティを洗い出しているとモデル作成者が確信していることを意味します。次の図に示すように、確定型サブタイプは、サブタイプ記号の下の2本線によって示されます。



サブタイプリレーションシップを作成する場合、サブタイプ識別子のバリデーションルールも作成しておくといでしょう。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、サブタイプ識別子「口座区分」のバリデーションルールに、C(当座預金口座)、S(普通預金口座)、L(借入口座)を含めます。口座区分「O」の古いデータも存在する場合、バリデーションルールによって記述されていない区分が見つかり、「O」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であるかを判断できます。

包括的および排他的リレーションシップ

IDEF1Xと違い、IE 表記法では確定型と未確定型のサブタイプ リレーションシップを区別しません。代わりに IE 表記法では、リレーションシップが包括的か排他的であるかを表します。ただし、IDEF1X 表記法でも排他的か包括的であるかを表せます。

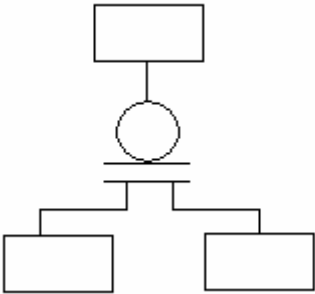
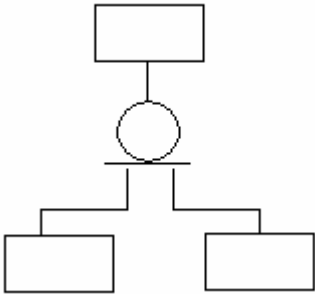
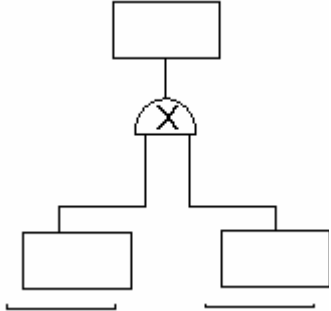
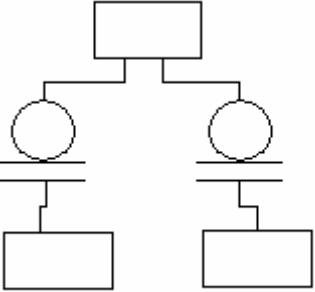
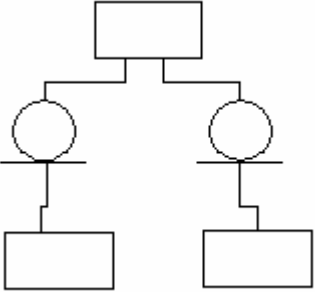
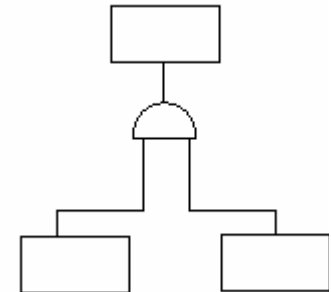
排他的サブタイプ リレーションシップでは、スーパータイプの各インスタンスが1つのサブタイプのみに関連付けられます。たとえば、従業員は正社員またはパートタイムのいずれかに属し、その両方に属することはないというビジネス ルールをモデル化することができます。このモデルを作成するには、「従業員」スーパータイプ エンティティに「正社員」および「パートタイム」というサブタイプ エンティティと、「従業員区分」という識別子属性を含めます。さらに、識別子の有効値を、正社員を示す「F」とパートタイムを示す「P」に制限します。

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を1つ以上のサブタイプに関連付けることができます。この場合、従業員は正社員、パートタイム、またはその両方に属するというビジネス ルールが表されます。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」、その両方を示す「B」に制限することができます。

注: IDEF1X 表記法で包括的サブタイプを表すには、スーパータイプ エンティティと各サブタイプ エンティティの間に個別のリレーションシップを作成します(次ページの図を参照)。

IDEF1XおよびIEにおけるサブタイプ表記

次の表に、IDEF1X および IE でのサブタイプ表記を示します。

	IDEF1X サブタイプ表記		IE サブタイプ表記
	確定型	未確定型	
排他的サブタイプ			
包括的サブタイプ			

どのような場合にサブタイプ リレーションシップを作成するか

次のような場合にサブタイプ リレーションシップを作成します。

- エンティティに共通の属性セットがある。これはすでに説明しました。
- エンティティに共通のリレーションシップ セットがある。これについては説明していませんでしたが、すでに例として挙げた銀行口座の構造のように、すべてのサブタイプ エンティティに共通のリレーションシップを、サブタイプの親からの 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、それぞれの種類の口座が「顧客」エンティティに関連付けられている場合、「口座」エンティティを親とする 1 つのリレーションシップを作成して、各サブタイプ エンティティからのリレーションシップをまとめることができます。
- 業務モデルでは、必要に応じてサブタイプ エンティティを使用すると、意思疎通やモデルの理解が行いやすくなります。サブタイプ エンティティに共通の属性のみが存在する場合や、特定のサブタイプ エンティティに固有のリレーションシップが存在しない場合でも同様です。モデルの主な目的の 1 つは、情報構造の共有を支援することであるということをお忘れなく。サブタイプ エンティティの使用が構造の理解に役立つ場合は、そのようなモデルを作成します。

第7章 正規化の問題点と解決法

本章には次のトピックがあります。

[正規化](#) (61ページを参照)

[正規形の概要](#) (62ページを参照)

[よくある設計上の問題](#) (63ページを参照)

[一意化](#) (71ページを参照)

[必要な正規化のレベル](#) (72ページを参照)

[正規化のサポート](#) (74ページを参照)

正規化

正規化とは、リレーショナル データベース設計において、リレーショナル構造のデータを編成し、冗長性と非リレーショナル構造を最小限にするプロセスです。正規化のルールに従って、同じ事実を得るために複数の手段があるようなモデル構造をすべて取り除くことで、データの冗長性を管理および解消することができます。

正規化の目的は、ある事実を得る手段を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！」 (ONE FACT IN ONE PLACE!)

正規形の概要

最も一般的な正規形の定義を次に示します。

関数従属(FD: Functional Dependence)

エンティティ「E」で、属性「A」の各値が常に属性「B」のただ 1 つの値に関連付けられている場合、属性「B」は属性「A」に関数従属します。言い換えると、「A」は一意に「B」を識別します。

完全関数従属(Full Functional Dependence)

エンティティ「E」で、属性「B」が複合属性「A」のすべての要素に関数従属しているが、「A」の一部には関数従属していない場合、属性「B」は属性「A」全体に完全関数従属します。

第 1 正規形(1NF)

属性値が、これ以上分割できない最小単位の値のみを含む場合、エンティティ「E」は第 1 正規形です。(COBOL データ構造などで見られる) 繰り返しグループは取り除く必要があります。

第 2 正規形(2NF)

エンティティ「E」が第 1 正規形であり、かつ、すべての非キー属性が主キーに完全関数従属する場合、エンティティ「E」は第 2 正規形です。つまり、キーの一部には従属しません(エンティティ「E」のキー全体「K」には従属するが、「K」の一部には従属しません)。

第 3 正規形(3NF)

エンティティ「E」が第 2 正規形であり、かつ、「E」の非キー属性が他の非キー属性に従属しない場合、エンティティ「E」は第 3 正規形です。第 3 正規形を表現する方法は他にもあります。別の定義として、エンティティ「E」が第 2 正規形であり、かつ、すべての非キー属性が非推移的に主キーに従属する場合、エンティティ「E」は第 3 正規形です。また、エンティティ「E」のいかなる属性も、エンティティ「E」(第 2 正規形)のすべての事実を表し、かつ、エンティティ「E」の事実のみを表す場合も、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、「各属性はキーに依存している。すべてのキーに依存している。そしてキーのみに依存している。だから助けてください、Codd ! 」(Each attribute relies on the key, the whole key, and nothing but the key, so help me Codd!)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 正規形、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、通常は構造を非正規化します。非正規化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。

よくある設計上の問題

設計上の問題の多くは、正規形に違反した結果生じます。よくあ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
- 同じ属性の多重使用
- 同じ事実が複数存在する
- 事実が矛盾する
- 導出属性
- 情報の欠落

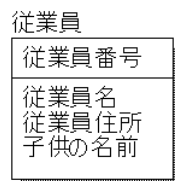
設計上の問題を解消する際は、サンプル インスタンス表を使用すると、多くの正規化エラーを見つけるのに役立ちます。

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、属性内にあるリスト、繰り返し要素、または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ましたが、第 1 正規形に違反しているため RDBMS モデルでは取り除く必要があります。RDBMS では可変長の繰り返しフィールドを取り扱うことができません。このような配列に添字を付けることができないためです。

下図のエンティティには、「子供の名前」という繰り返しデータグループがあります。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 1 つだけの意味と値を持つ場合、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。

以下に示すような繰り返しデータグループは、データベースを定義して実際のデータを含める際に問題を引き起こします。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多くの名前がある場合は、どうしたらよいか?」



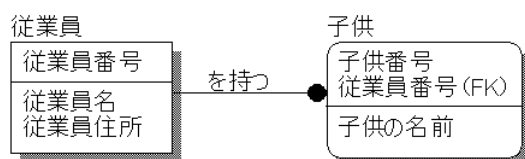
次のようなサンプル インスタンス表を使用すると、問題が明らかになります。

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

従業員番号	従業員名	従業員住所	子供の名前
E3	Bob	Princeton	-
E4	John	New York	Lisa
E5	Carol	Berkeley	-

設計上の問題点を修正するには、「従業員」エンティティから子供の名前の一覧を何らかの方法で取り除く必要があります。1つの方法は、下図に示すように、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エン트리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、これによって領域割り当てに関する問題がある程度解決されます。子供がいない従業員に対してレコード構造の領域の無駄な消費を防ぐことができ、家族がいる従業員に対して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。

「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段階(第1正規形への変換)です。両方のエンティティには固定長のフィールドのみが含まれており、構造の理解とプログラミングが容易になります。

同じ属性の多重使用

1つの属性が2つの事実の一方を表すことができ、どちらの事実を表しているのかわからない場合も問題となります。たとえば、「従業員」エンティティに「入社日または退職日」という属性があるとして、次のように、各従業員に対してこの情報を記録できます。

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	

「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	2004年1月10日
E2	Don	Berkeley	2002年5月22日
E3	Bob	Princeton	2003年3月15日
E4	John	New York	2003年9月30日
E5	Carol	Berkeley	2000年4月22日
E6	George	Pittsburgh	2002年10月15日

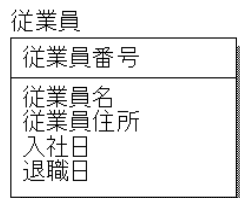
この構造の設計上の問題は、入社日（「従業員」が会社に入った日付）と退職日（「従業員」が会社を辞めた日付）がわかっている場合に、その両方を記録できないことです。これは、1つの属性で2つの異なる事実を表しているためです。COBOL システムでもよく使われる構造ですが、しばしば保守上の問題を引き起こしたり、情報を誤って解釈する原因となります。

この問題を解決するには、それぞれの事実を個別の属性で表します。次の図は、問題を解決する試みの一つです。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るには、「日付区分」属性から日付の種類を取得する必要があります。これは物理データベースの領域を節約するには効果的かもしれませんが、クエリのロジックに混乱が生じます。

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	
日付区分	

実際、この変更によって別の正規化違反が生じてしまいます。「日付区分」の存在が「従業員番号」に依存していないためです。さらに設計上の問題もあります。技術的問題は解決されていますが、根本的な業務上の問題(従業員に関する2つの事実をどのように格納するか)が解決されていないためです。

データを検討すると、より優れた解決策は、次の図のように各属性が別々の事実を表すようにすることだとすぐにわかります。



「入社日」および「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

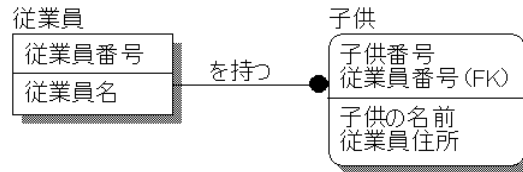
従業員番号	従業員名	従業員住所	入社日	退職日
E1	Tom	Berkeley	2004年1月10日	-
E2	Don	Berkeley	2002年5月22日	-
E3	Bob	Princeton	2003年3月15日	-
E4	John	New York	2003年9月30日	-
E5	Carol	Berkeley	2000年4月22日	-
E6	George	Pittsburgh	2002年10月15日	2003年11月30日

これまでに示した2つの例には、第1正規形の違反が含まれていました。構造を変更して、各属性がエンティティ内に一度だけ出現し、1つの事実だけを表すようになりました。すべてのエンティティと属性の名前を単数形にして、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となります。

同じ事実が複数存在する

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高めることです。そのためには、データベース内で各事実を1回だけ表すことが重要です。そうしないと、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)の唯一の例外はキー属性で、データベース内に複数回現われます。しかし、キーの一貫性は参照整合性によって管理されます。

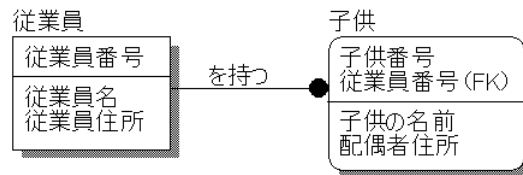
同じ事実が複数存在する場合、しばしば元のデータベース設計に欠陥があります。次の図では、「子供」エンティティに「従業員住所」を含めたことで、データベース設計にエラーが生じたことがわかります。従業員に複数の子供がいる場合、住所は子供とは別に管理する必要があります。



「従業員住所」は「従業員」に関する情報です。「子供」に関する情報ではありません。実際、このモデルは第2正規形に違反しています。「エンティティ内の各事実は、エンティティのキー全体に従属していなければならない」という定義に従っていないからです。上の例は、「従業員住所」が「子供」のキー全体に従属していないので（キーの一部である「従業員番号」にのみ従属しています）、第2正規形ではありません。「従業員住所」を「従業員」エンティティに戻すと、モデルは少なくとも第2正規形になります。

事実が矛盾する

事実の矛盾は、第1、第2、または第3正規形の違反を含む、さまざまな理由で発生します。次の図に、第2正規形の違反によって発生する、事実の矛盾した例を示します。



「配偶者住所」を示す2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

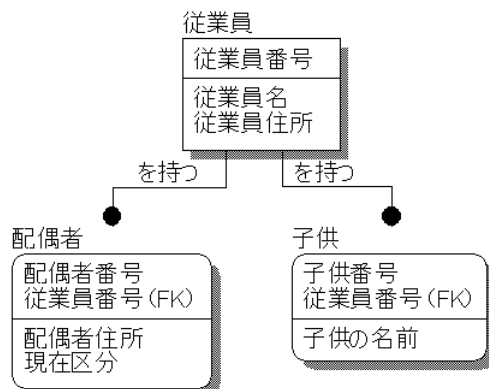
子供

従業員番号	子供番号	子供の名前	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

「配偶者住所」という属性が「子供」エンティティに含まれていますが、この設計は第2正規形に違反します。インスタンスデータを見ると、この違反がわかります。「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2人の配偶者(それぞれ「Berkeley」と「Cleveland」に住む)がいるか、または「Donna」の母は「Tom」と「Dick」とは異なるのでしょうか。あるいは、「Don」の配偶者は1人ですが「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。この場合、モデルから答えを知ることはできません。このような意味上の問題を解決するには、ビジネスユーザーに聞くしかありません。そのため、アナリストは業務に関する適切な質問をして、正しい設計を見つける必要があります。

この例で問題となるのは、「配偶者住所」が「従業員」の「配偶者」についての事実であり、「子供」についての事実ではないという点です。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(おそらく「Don」と共に)住所を変えるたびに、複数の場所に格納された情報(「Don」が親である「子供」の各インスタンス)を更新する必要があります。複数の場所を更新する必要があると、見落としによりエラーが発生する可能性があります。

「配偶者住所」は、子供ではなく配偶者に関する事実であることを認識すると、問題を解決できます。この情報を表すには、次の図に示すように「配偶者」エンティティをモデルに追加します。



「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

「配偶者」を別のエンティティに分割することで、「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。「Don」の2人の配偶者のうち、1人は前妻です。

エンティティ内の各属性が、そのエンティティに関するただ1つの事実を表すことを確認すると、モデルは少なくとも第2正規形であると見なすことができます。さらに、モデルを第3正規形に変換すると、データベースに問題が発生する可能性が低くなり、矛盾する情報が含まれたり、必要な情報が欠落することを回避できます。

導出属性

事実が矛盾するもう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」という属性の両方を非キー属性として「子供」エンティティに含めた場合、第3正規形に違反します。これは、「年齢」が「生年月日」に関数従属するからです。「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)であるため、データベースに直接格納する必要はありません。厳密に言えば、導出元が更新されるたびに導出属性も更新する必要があります。これは、たとえば読み込みや更新を一括で行うアプリケーションで大きなオーバーヘッドを発生させます。アプリケーション設計者やプログラマは、導出した事実が必要に応じて更新されるようにする必要があります。

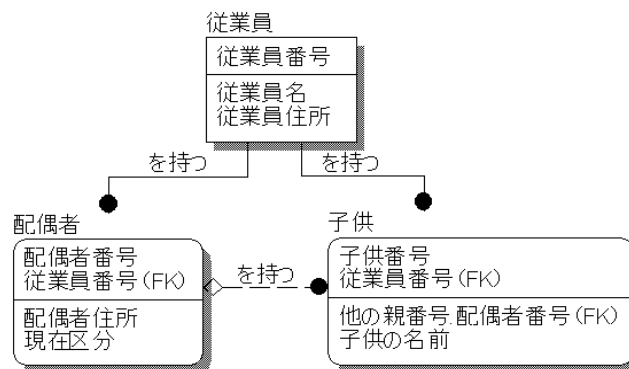
正規化の目的は、データベースに記録された各事実を知る方法をただ1つにすることです。導出属性の値、導出アルゴリズム、およびアルゴリズムで使用する属性値がわかっている場合、事実を知る方法は2つあることになります。導出属性の値を単に確認するか、または手動計算で求める方法です。答えを知るために2つの方法がある場合、それぞれの方法で得られた答えが異なっている可能性があります。

たとえば、「子供」の「生年月日」および「年齢」属性の両方を記録できるとします。さらに、「年齢」属性がデータベース内で変更されるのは、毎月末の保守作業時のみとします。ここで、「この「子供」は何才か?」という質問の答えを得るには、直接「年齢」にアクセスするか、または「今日の日付」から「生年月日」を減算することができます。後者の方法で減算を行った場合、常に正しい答えが得られます。一方、前者の方法であると「年齢」が最近更新されていない場合は誤った答えとなるため、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、導出データをモデル内に記録しておく役割のような状況もあります。これは、業務の担当者と共にモデルを検討するような場合にも役立ちます。モデリング理論では、導出データをまったく含めないか含めるとしても控え目にすべきだと述べていますが、本当に必要な場合はこのルールに従う必要はありません。ただし、少なくともその属性は導出される値であるという事を記録して、導出アルゴリズムを記載しておきましょう。

情報の欠落

モデル情報の欠落は、データの正規化作業によって発生することがあります。前に挙げた例では、「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で設計を改善しましたが、「子供」エンティティと「配偶者住所」の間の暗黙的なリレーションシップが壊れてしまっていました。最初に「配偶者住所」属性が「子供」エンティティに格納されたのは、子供にとってもう一方の親(配偶者と仮定して)の住所を表すためだったと考えられます。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。



「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

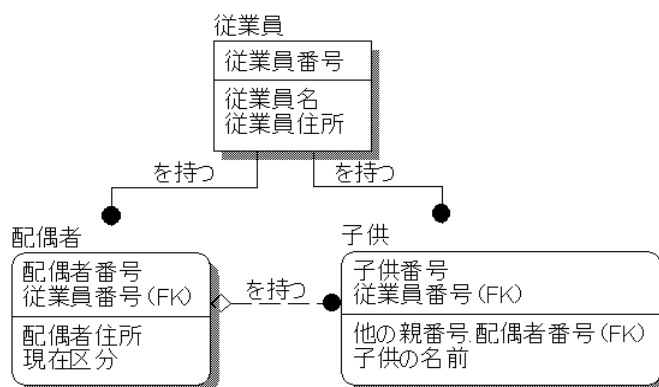
配偶者

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

このモデルの正規化はまだ完了していません。正規化を完了するには、両親とも従業員である場合を含む、従業員と子供の間ですべてのリレーションシップを表すことができればなりません。

一意化

次の例では、「従業員」および「配偶者」エンティティと「子供」エンティティとの間に作成された2つのリレーションシップを通じて、「従業員番号」属性が「子供」エンティティに移行されています。この結果、「子供」エンティティに外部キー属性が2回追加されると思われるかもしれませんが、「従業員番号」属性は「子供」エンティティのキー領域にすでに存在するため、「配偶者」エンティティのキーに「従業員番号」が含まれていても、「子供」エンティティの中で繰り返し追加されることはありません。



このように、複数のリレーションシップを通じて同じベース属性から移行した同一の外部キー属性2つを結合することができます。この操作を一意化と呼びます。この例では、「従業員番号」は「子供」エンティティの主キーの一部であり（「従業員」エンティティから<持つ>リレーションシップを通じて移行）、「子供」エンティティの非キー属性（「配偶者」エンティティから<持つ>リレーションシップを通じて移行）にもなります。両方の外部キー属性は共に「従業員」エンティティの識別子であるため、この属性は1つにまとめたほうが良いと考えられます。このような場合、自動的に一意化が実行されます。

一意化の際に適用される規則を次に示します。

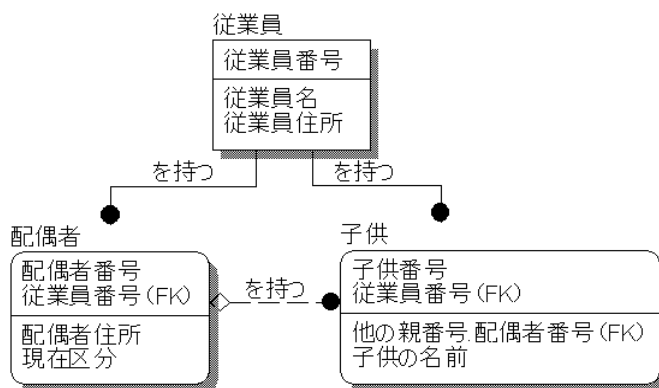
- 同じ外部キーがあるエンティティに対して繰り返し移行され、ロール名の割り当てがない場合、その外部キーは一意化されます。
- それぞれの外部キーに異なるロール名が割り当てられている場合、一意化は行われません。
- 異なる外部キーに同じロール名が割り当てられており、外部キーのベース属性が同じ場合、一意化が行われます。ロール名が同じで、外部キーのベース属性が異なる場合、エラーになります。
- 一意化する外部キーのうち、いずれかの外部キーがエンティティの主キーの一部である場合、一意化された属性は主キーの一部として残ります。
- 一意化する外部キーのうち、いずれの外部キーも主キーの一部ではない場合、一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効にすることができます。同じ外部キーが子エンティティ内で複数回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。

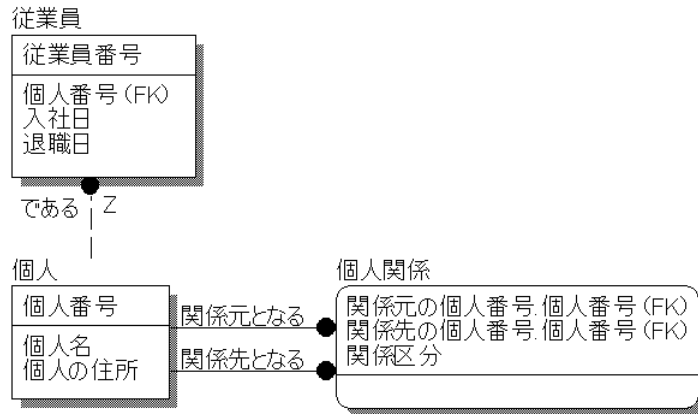
必要な正規化のレベル

形式的な正規化(アルゴリズムによってモデルの構造から見つけ出され、エンティティや属性の意味を考慮することがない正規化)の観点からは、「従業員」-「子供」-「配偶者」モデルに問題はありません。しかし、正規化されたからといって、そのモデルが完全に正しいとは限りません。必要な情報の一部を格納できなかったり、効率的に格納されない場合があります。経験を積むと、純粋な正規化が完了した後にも、その他の設計上の問題を見つけて解消できるようになります。

次の「従業員」-「子供」-「配偶者」モデルの例を使用すると、両親が共に「従業員」であるような「子供」を記録する方法がないことがわかります。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。



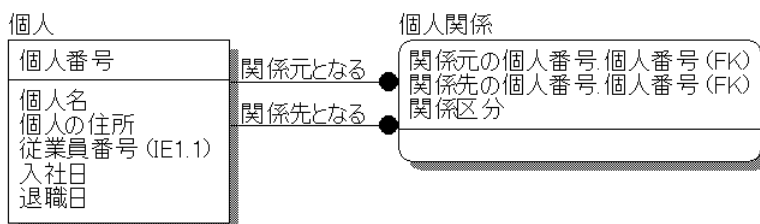
「従業員」、「配偶者」、および「子供」がすべて、個人のインスタンスを表していることに気がつく、これらの情報を個人についての事実を表すテーブルと、その関係についての事実を表すテーブルにまとめることができます。モデルを修正するには、「子供」と「配偶者」を削除して、「個人」と「個人関係」に置き換えます。「個人関係」エンティティに表された「個人」同士の関係を通じて、親子関係や婚姻関係を記録できます。



この構造では、「個人」同士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関係も記録できるようになります。たとえば、この新しい構造では養子縁組を取り扱うことができます。養子縁組を表すには、「個人」エンティティの「関係区分」属性のバリデーションルールに新しい値を追加して、養子の親であることを表します。必要に応じて、「後見人」、「内縁」、またはその他の「個人」同士の関係を追加できます。

「従業員」は独立エンティティのままです。業務において「個人」とは区別されるためです。ただし、「個人」に対する「である (is a)」リレーションシップにより、「従業員」は「個人」のプロパティを継承します。そのリレーションシップに「Z」が付いていることと、ひし形(◇)記号がないことに注意してください。これは、「1」対「0 または 1」のリレーションシップです。サブタイプエンティティに別のキーが必要な場合に、サブタイプの代わりに使用されます。上の例では、「個人」は「従業員」<である>か、または<ではない>かのいずれかとなります。

「個人」と「従業員」の両方で同じキーを使用したい場合、「従業員」エンティティを「個人」に組み込み、「個人」が従業員でないときは従業員の属性を NULL に指定できるようにします。別の識別子(「従業員番号」)で従業員を検索するように指定することもできます。このような構造を次の図に示します。



上の例から、モデルを正規化できても、業務が正しく表現されない場合があることがわかります。形式的な正規化は重要ですが、この章で行ったように、サンプル インスタンス表を使用してモデルの内容を検証することも重要になります。

正規化のサポート

AllFusion ERwin DM ではデータ モデルの正規化をサポートしていますが、現在のところ、正規化アルゴリズムを完全にはサポートしていません。これまで対話型のモデリング ツールを使用したことがない方には、標準のモデリング機能が非常に役立つことに気がつくでしょう。これらの機能を活用することで、多くの正規化違反を避けることができます。

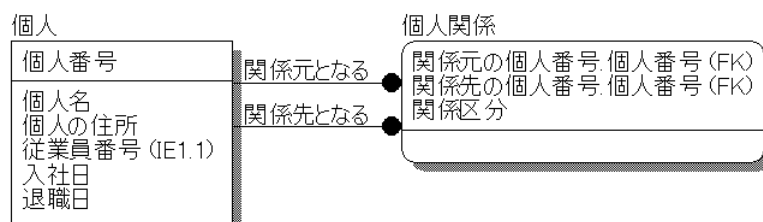
第 1 正規形のサポート

モデルの各エンティティや属性は、名前で識別されます。オブジェクトには任意の名前を指定できますが、以下の場合は例外となります。

- 同じエンティティ名が 2 度使用されるとメッセージが表示されます (重複名の基本設定によって動作が変わります)。
- 同じ属性名が 2 度使用されるとメッセージが表示されます (重複名の基本設定によって動作が変わります)。ただしロール名の場合は除きます。ロール名を割り当てると、同じ名前の属性を異なるエンティティで使用できます。
- あるエンティティに同じ外部キーを何度も設定するには、同一属性を一意化する必要があります。

重複名を許可しないように設定すると、各事実を 1 つの場所でのみ使用するようにできます。それでも、属性を正しく配置しない場合は、第 2 正規形に違反することがあります。どのように属性名を設定すべきかは、対象となる業務を詳しく理解した上で決定する必要があります。

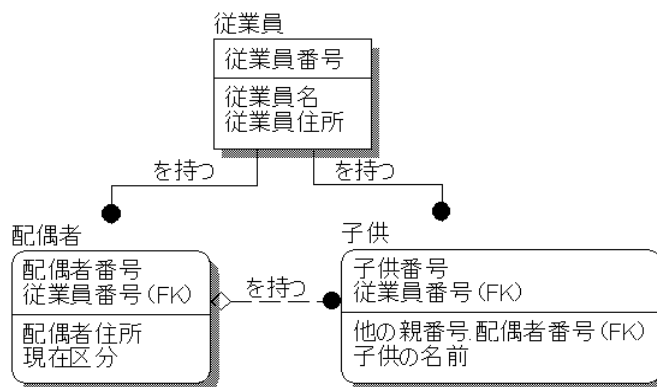
AllFusion ERwin DM では、データ モデル内の属性に割り当てられた名前がリスト データを表しているかを判断することはできません。そのため、AllFusion ERwin DM では次の例の場合でも、「子供達の名前」を属性名として受け入れます。このような場合、AllFusion ERwin DM では、すべてのモデルが第 1 正規形であることは保証されません。



しかしながら、リスト データ型は DBMS スキーマでもサポートされません。スキーマは、物理リレーショナル システムのデータベースを表しているため、このレベルでも第 1 正規形の違反は回避できません。

第2および第3正規形のサポート

現在のところ、AllFusion ERwin DMでは関数従属性を管理していません。しかし、第2および第3正規形の違反を回避するために役立つ機能があります。たとえば、次の例を再構成する場合に、「配偶者」の属性として定義した「配偶者住所」を、「子供」の属性として定義することはできません。(ただし、重複名の基本設定によって動作は変わります。)



ロール名を持たない外部キーが繰り返しモデルに表れないように、モデルの構造をよく検討してください。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、業務の面から確認すべき事柄があります。「2つの異なるインスタンスのキーを表しているのか、それとも両方のキーが同じインスタンスを表しているのか」ということです。

2つの外部キーが異なるインスタンスを表す場合、個別のロール名が必要になります。2つの外部キーが同じインスタンスを表す場合、正規化に違反している可能性が非常に高いと言えます。あるエンティティで同じ外部キーが2回使用されており、ロール名が指定されていない場合、モデルのリレーションシップ構造に冗長性があります。2つの外部キーに同じロール名を割り当てた場合、一意化が実行されます。

第8章 物理モデル

本章には次のトピックがあります。

[目的](#) (77ページを参照)

[物理モデルの役割](#) (77ページを参照)

[非正規化](#) (79ページを参照)

目的

物理モデルの目的は、データベース管理者に必要な情報を提供して、効率的な物理データベースを作成することです。物理モデルは、データベースを構成するデータ要素を(データディクショナリに)定義および記録するために必要な情報を提供します。また、アプリケーションチームがデータにアクセスするプログラムの物理構造を選択するのを支援します。情報システムのすべてのニーズを満たすために、多くの場合、物理モデルはデータ管理、データベース管理、およびアプリケーション開発の各領域を担当するチームで共同開発されます。

開発時には必要に応じて、物理モデルに基づいて物理データベース設計と業務情報の要件を比較することで、次のような情報を得ることができます。

- 物理データベース設計が必要な要件を十分に満たしていることを示します。
- 物理設計上の選択事項とその影響(たとえば、何が満たされ、何が満たされないか)を記述します。
- データベースの拡張性と制約を特定します。

物理モデルの役割

物理モデルでは、次の役割がサポートされます。

- 物理データベースの生成
- 業務上の要件に対応した物理設計の文書化

たとえば、論理/物理モデルでは、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簡単に作成できます。AllFusion ERwin DM では、モデル表示を[論理]から[物理]に変更するだけで物理モデル設計を開始できます。論理モデルの各オプションに対応するオプションが、物理モデルに用意されています。つまり、エンティティはテーブル、属性はカラム、キーはインデックスになります。

物理モデルが作成されると、すべてのモデル オブジェクトを、選択した対象サーバーに対応した正しい構文で生成することができます。対象サーバーのカタログに直接生成するか、または DDL スクリプト ファイルとして間接的に生成できます。

論理および物理モデルのコンポーネント

論理モデルと物理モデルのオブジェクトの関係を次の表に示します。

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	外部キーは子テーブルの主キーに含まれる。
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、外部キーは子テーブルの主キーには含まれない。
属性	カラム
論理データタイプ(文字列、数値、日付/時刻、BLOB)	物理データタイプ(有効なデータタイプは選択した対象サーバーごとに異なる)
ドメイン(論理)	ドメイン(物理)
主キー	主キー、主キー インデックス
外部キー	外部キー、外部キー インデックス
代替キー(AK)	代替キー インデックス(主キー以外のユニークインデックス)
逆方向エントリ(IE)	逆方向エントリ インデックス(非ユニーク インデックス。顧客の名字など、一意でない値でテーブル情報を検索するために作成される)
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	リレーションシップ(外部キーを使用して実装)
依存型リレーションシップ	外部キーは子テーブルの主キーに含まれる(仕切り線の上側)。
非依存型リレーションシップ	外部キーは子テーブルの主キーに含まれない(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(CASCADE、RESTRICT、SET NULL、SET DEFAULT)	INSERT、UPDATE、および DELETE トリガ
カーディナリティ	INSERT、UPDATE、および 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립トまたはポストスク립ト

参照整合性は論理モデルに記述されます。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の構成要素でもあります。トリガや宣言ステートメントがスキーマ内に記述されるからです。したがって、参照整合性は論理モデルと物理モデルの両方でサポートされます。

非正規化

論理モデルの構造を非正規化したり、テーブル内のデータを冗長化して、クエリのパフォーマンスを改善することができます。これによって、対象の RDBMS に対して効率よく設計された物理モデルを構築することができます。次のような機能で非正規化がサポートされます。

- 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。論理モデルの任意の項目で[論理のみ]チェックボックスをオンにすると、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]設定を使用して、物理モデルでサブタイプリレーションシップを非正規化したり、部分キーの移行を実行できます。
- 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。物理モデルの任意の項目で[物理のみ]チェックボックスをオンにすると、その項目は物理モデルにのみ表示されます。この設定を使用して、物理モデルを非正規化することもできます。論理設計では必要ではないものの、物理実装の要件を直接サポートするテーブル、カラム、およびインデックスを物理モデルに含めることができます。
- 物理モデルでの多対多リレーションシップの解決。論理モデルと物理モデルの両方で、多対多リレーションシップの解決がサポートされます。論理モデルで多対多リレーションシップを解決すると、関連エンティティが作成され、必要な属性を追加することができます。論理モデルでは多対多リレーションシップを保持しておき、物理モデルで解決することもできます。その際は、元の論理設計と新しい物理設計の間のリンクは維持され、関連テーブルの生成元の情報がモデルに記述されます。

付録A: 依存エンティティの種類

本章には次のトピックがあります。

[依存エンティティの分類](#) (81ページを参照)

依存エンティティの分類

次の表に、IDEF1X ダイアグラムに表示される依存エンティティの種類を示します。

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複数回出現する属性グループを表します。他のエンティティによって直接的に特定されることはありません。例では、「趣味」が「人」の特性です。	
関連 (Associative) または 指定 (Designative)	関連エンティティおよび指定エンティティは、2つ以上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情報のみを含む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情報に加え、リレーションシップを説明する属性も含む場合、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」、および「借入口座」がサブタイプ エンティティです。	

用語集

BLOB

バイト データとテキスト データを保存するために予約された DB 領域。バイナリ ラージ オブジェクト(BLOB)を構成し、テーブル カラムに格納されます。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情報を保存することができます。

依存エンティティ

インスタンスを一意に識別するために、別のエンティティとのリレーションシップが必要となるエンティティ。

依存型リレーションシップ

親エンティティとの関連から、子エンティティのインスタンスが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

共通の属性や特性を持つ具体的または抽象的なもの(人、場所、出来事など)の集まりを表す。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親エンティティから子エンティティに移行された属性。外部キーは、ある 1 つの値セットに対する二次参照を表します(一次参照は実属性)。

カーディナリティ

親と子のインスタンスが何対何の関係にあるかを定義するもの。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 です。n は次のいずれかになります。

- 0 または 1 以上 (黒いドットが表示されます)
- 1 以上 (黒いドットと共に P のマークが表示されます)
- 0 または 1 (黒いドットと共に Z のマークが表示されます)
- 定数 (黒いドットと共に指定した数 n が表示されます)

確定型サブタイプ クラス

サブタイプ クラスに、考えられるすべてのサブタイプが含まれている場合(スーパータイプ エンティティのすべてのインスタンスが、いずれかのサブタイプに関連付けられる場合)、そのサブタイプ クラスは確定型となります。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になります。したがって、「当座預金口座」、「普通預金口座」、および「借入口座」のサブタイプ クラスは確定型です。

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属性または属性のセット。エンティティのインスタンスにアクセスする際に使用することがあります。各逆方向エントリには、1 つの非ユニーク 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を表すエンティティ。たとえば、「正社員」は「従業員」の特定のタイプを表します。サブタイプ エンティティは、特定のサブタイプのみ適用される情報を保管することができます。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すことができます。たとえば、「正社員」は「福利厚生」を受けられるが、「パートタイム従業員」は受けられない」というケースです。IDEF1X では、サブタイプ クラス内のサブタイプは互いに排他的です。

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(カテゴリ リレーションシップとも呼びます)は、サブタイプ エンティティとそのスーパータイプ エンティティとの間のリレーションシップです。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 エンティティのインスタンスとサブタイプのインスタンスが「1」対「0 または 1」で関連付けられます。

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が、親エンティティ内に対応する値を持っている」ということ。

識別子

スーパータイプ エンティティのインスタンスに含まれた属性値から、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」のインスタンスの「口座区分」属性値から、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または属性のセット。各インスタンスを一意に識別できる属性または属性グループが複数ある場合、これらの候補キーの中から主キーが選択されます。業務における識別子としての重要性が選択基準となります。時間が経過しても変化せず、できるだけ短いものが主キーとして理想的です。各主キーには、1 つのユニーク 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

リレーショナル構造のデータを編成して、冗長性と非リレーショナル構造を最小限にするプロセス。

属性

具体的または抽象的なもの(人、場所、出来事など)の集まりに関連付けられた、特徴や性質の種類を表す。

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または属性のセット。
 2. エンティティのインスタンスを一意に識別できる属性または属性グループが複数ある場合、主キーとして選択されなかった属性または属性グループ。
- 各代替キーには、1 つのユニーク インデックスが生成されます。

特定リレーションシップ

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 または 1 つ以上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 または 1 つのインスタンスと関連しているリレーションシップ。

独立エンティティ

インスタンスを一意に識別するために、他のエンティティとのリレーションシップが不要なエンティティ。

ドメイン

定義済みの論理および物理プロパティのグループ。あらかじめ作成しておく、属性やカラムに簡単にアタッチできます。

2 項リレーションシップ

親エンティティと子エンティティのインスタンスが、「1」対「0 または 1 以上」で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はすべて 2 項リレーションシップです。

非正規化

テーブル内に冗長性のあるデータを配置して、クエリのパフォーマンスを向上させること。

非キー属性

エンティティの主キーに含まれていない属性。非キー属性は、逆方向エントリまたは代替キーの一部となったり、または外部キーとなる場合があります。

非依存型リレーションシップ

親エンティティとの関連から、子エンティティのインスタンスが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は、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。データベースおよびデータベース管理システム (DBMS) に固有のモデリング情報 (テーブル、カラム、データタイプなど) を追加することができます。

不特定リレーションシップ

親-子接続されたリレーションシップとサブタイプ リレーションシップは、いずれも特定リレーションシップであると見なされます。各エンティティのインスタンス間の関係が正確に定義されるからです。一方で、モデリングの初期段階に、2 つのエンティティ間を不特定リレーションシップで結ぶと便利な場合があります。不特定リレーションシップは、多対多リレーションシップとも呼びます。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 または 1 つ以上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も、親エンティティの 0 または 1 つ以上のインスタンスと関連しているリレーションシップです。

ベース名

ロール名が付けられた外部キーの元の名前。

未確定型サブタイプ クラス

サブタイプ クラスに、考えられるすべてのサブタイプが含まれていない場合 (スーパータイプ エンティティのすべてのインスタンスが、いずれか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 クラスは未確定型となります。たとえば、ある会社で従業員の勤務形態として正社員、パートタイム、および派遣の 3 種類が認められている場合、サブタイプとして「正社員」と「パートタイム従業員」のみを含むサブタイプ クラスは未確定型です。

ロール名

外部キーに割り当てる新しい名前。ロール名を使用して、外部キーは親エンティティに含まれる属性のサブセットであり、子エンティティで特定の機能 (または役割) を果たすことを表します。

論理モデル

AllFusion ERwin DM のデータ モデリング レベル。概念モデルの作成に使用され、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

AllFusion ERwin DM で作成できるモデル タイプ。論理モデルと物理モデルが自動的にリンクされます。

索引

A

AllFusion ERwin DM
ダイアグラム コンポーネント • 20
モデルの利点 • 9

C

CASCADE
定義 • 47
例 • 50

E

ERD
エンティティ リレーションシップ ダイアグラムを参照
• 16

I

IDEF1X, 開発者 • 9
IE, 開発者 • 9

R

RESTRICT
定義 • 47
例 • 50

S

SET DEFAULT
定義 • 47
SET NULL
定義 • 47
例 • 51

い

移行, ロール名 • 33
依存
識別 • 31
存在 • 31
依存エンティティ • 31
種類 • 85
依存型リレーションシップ • 32
カーディナリティ • 44
一意化
外部キーにロール名を付ける • 41
正規化における問題の回避 • 75
1 対多 • 22
インスタンス, 定義 • 21

え

エイリアス, エンティティ名 • 36
エンティティ
ERD 内 • 20
依存 • 31
親 • 22
関連 • 53, 85
業務用語による定義 • 39
子 • 22
サブタイプ • 57, 85
指定 • 85
循環定義の回避 • 38
スーパータイプ • 57
定義 • 21
定義の基準 • 37
定義の説明 • 37
定義の割り当て • 37
同義語と同音異義語の回避 • 36
特性 • 85
独立 • 31
名前 • 35
エンティティ リレーションシップ ダイアグラム
作成 • 20
サブジェクト エリア • 19
サンプル • 20
定義 • 17
目的 • 19

お

親エンティティ • 22

か

カーディナリティ
IDEF1X および IE での表記 • 44
依存型リレーションシップ • 44
定義 • 44
非依存型リレーションシップ • 46
外部キー
一意化 • 41
参照整合性の割り当て • 47
外部キー属性, ロール名 • 33
確定サブタイプ リレーションシップ • 60
関連エンティティ • 53
定義 • 85

き

キー
逆方向エントリ • 30

- 主キー・28
 - 選択の例・28
 - 代替キー・30
 - 代理・28
- キー属性・28
- キーベース モデル
 - 定義・17, 27
 - 目的・27
- 逆方向エントリ・30
- 業務
 - 用語集の作成・39
 - 用語の整理・39

け

- 継承階層
 - 定義・57

こ

- 候補キー, 定義・28
- 子エンティティ・22
- コンポーネント, ERD 内・20

さ

- 再帰リレーションシップ・52
 - 定義・56
- サブタイプ エンティティ, 定義・85
- サブタイプ リレーションシップ・52
 - 確定・60
 - 作成・63
 - 識別子・57
 - スーパータイプ・57
 - 定義・57
 - 排他的・61
 - 表記法・62
 - 包括的・61
 - 未確定・60
- 参照整合性・47
 - AllFusion ERwin DM ダイアグラムでの表記・49
 - CASCADE・47
 - RESTRICT・47
 - SET DEFAULT・47
 - SET NULL・47
 - 定義・47
 - 例・50, 51

し

- 識別依存・31
- 識別子, サブタイプ リレーションシップ・57
- 指定エンティティ, 定義・85
- 主キー・28
 - 選択・28
- 進行役, 役割・14

す

- スーパータイプ・57

せ

- 正規化
 - AllFusion ERwin DM のサポート・78
 - 完了・76
 - 設計上の問題の回避・67, 69, 70, 71, 73
 - 第1正規形・67, 69
 - 第2正規形・70
 - 第3正規形・71, 73
 - 物理モデルでの非正規化・83
- 正規形
 - 6形式の概要・66
- セッション
 - 計画・13
 - 役割・14
- 全属性モデル・15
 - 定義・17

そ

- 属性
 - ERD 内・20
 - 値のドメインの指定・40
 - 業務用語による定義・39
 - 定義・21, 40
 - 定義内のバリデーション ルール・40
 - 同義語と同音異義語の回避・36
 - 導出・73
 - 名前・35
 - 複数使用の回避・69
 - 複数存在の回避・70
 - ロール名・33
 - ロール名の指定・41
- 存在依存・31

た

- 第1正規形・67, 69
- 第2正規形・70
- 第3正規形・71, 73
 - キーベース モデル・17
 - 全属性モデル・17
- 対象分野の専門家, 役割・14
- 代替キー・30
- 代理キー, 割り当て・28
- 多項リレーションシップ・52
 - 定義・54
- 多対多・23, 52, 53
 - 削除・53

て

- 定義

エンティティ・37
属性・40
ビジネス ルールの記述・42
ロール名・41
データ アナリスト, 役割・14
データ グループの繰り返し・67
データ モデリング
IDEF1X 手法のサンプル・15
作成例・25
手法・11
進行役の役割・14
セッション・13
対象分野の専門家の役割・14
定義・11
データ アナリストの役割・14
データ モデル作成者の役割・14
動詞句の使用・24
マネージャの役割・14
利点・9, 11
データ モデル作成者, 役割・14

と

動詞句・22
データ モデル内・24
例・22
導出属性
使用するタイミング・73
定義・73
特性エンティティ, 定義・85
独立エンティティ・31
ドメイン, 有効な属性値の指定・40

な

名前付け
エンティティ・35
属性・35

に

2 項リレーションシップ, 定義・54

は

排他的サブタイプ リレーションシップ・61
バリデーション ルール, 属性定義内・40
汎化
階層の定義・57
カテゴリの定義・57

ひ

非依存型リレーションシップ・33
カーディナリティ・46
非キー属性・28
ビジネス ルール

定義に記述・42
非正規化
物理モデル・83

ふ

物理のみプロパティ・83
物理モデル
作成・81
定義・17

へ

ベース属性, 定義・41
変換モデル・15
作成・81
定義・18

ほ

包括的サブタイプ リレーションシップ・61

ま

マネージャ, 役割・14

み

未確定サブタイプ リレーションシップ・60

り

リレーションシップ
ERD 内・20
依存エンティティ・31
依存型・32
1 対多・22
親から子へ読み取る・24
カーディナリティの適用・44
確定サブタイプ・60
子から親へ読み取る・24
再帰・52, 56
サブタイプ・52
サブタイプ(カテゴリ)・57
サブタイプ表記・62
参照整合性・47
多項・52, 54
多対多・23, 52, 53
定義・22
動詞句・22
独立エンティティ・31
排他的サブタイプ・61
非依存型・33
必須および任意・46
包括的サブタイプ・61
未確定サブタイプ・60

ろ

ロール名

移行 • 33

定義 • 33

定義の割り当て • 41

論理のみプロパティ • 83

論理モデル, 定義 • 16