情報システムにおける データベースのモデリングと設計

Josh Jones, Eric Johnson

目次

はじめに

データベース モデリングは、情報技術の分野で長年見過ごされてきたプロセスです。一般に、アプリケーション コードはテストと修正が必要であるのに対し、データは任意のデータベースに投入するだけですべてがうまくいくと思われていますが、これは事実ではありません。適切なデータ モデルを作成すれば、データの読み取りや書き込みを高速化して、アプリケーションのパフォーマンスを改善することができ、さらに将来の開発作業における拡張性と柔軟性を確保することができます。現在のビジネスでは、わずか数秒の差が業務に大きな影響を与えることがあるため、アプリケーションの処理が遅いためにユーザーが待たされるようなことがあれば、別のソリューションが検討されることもあるでしょう。アプリケーションの性能を最大限に発揮するには、データ モデルの作成に時間とリソースを割く必要があります。このホワイトペーパーでは、データ モデリングの概要とその重要性を説明し、さらに基本概念と実践方法を紹介します。

アプリケーション開発におけるデータベースの役割

アプリケーション開発において、データベースは大きな役割を果たします。多くのアプリケーションでは、後で再利用するために特定の形式でデータを保存する必要があります。ほとんどの場合、データの保存にはリレーショナル データベースが使用されます。他にも、フラット ファイル、XML、永続化したレコード セット、または独自のファイル形式などを使用できますが、これらはデータベースに比べて堅牢でも安全でもありません。データベースを使用すると、データ ウェアハウスやレポート作成ツールを通じて収集した情報を分析することができます。さらに、ほとんどのリレーショナル データベース管理システム（RDBMS）では、障害の発生に備えてデータベースをバックアップすることができ、高可用性ソリューションを実装して、障害によるダウンタイムを最小限に抑えることができます。これらの利点を考慮するだけでも、アプリケーション開発において、情報を保存および管理するためにデータベースを使用する価値があると言えるでしょう。

データ モデリングの基本

データ モデリングは、現実の情報をデータの論理表現にマッピングするという、一見非常に単純なプロセスです。つまり、顧客の情報をどのようにデータ モデルに保存するのか? といったことを検討します。このプロセスの大部分は、モデル作成者の判断に基づいて行われますが、データ モデリングにはさまざまな基本概念があり、それらを活用すると正しい判断を下すために役立ちます。また、モデリングを行う際には、論理的な視点からデータを検討する必要がありますが、この段階では、データベース上のテーブルやカラムの配置については考慮しません。このような作業を論理モデリングと呼びます。その目的は、現実のオブジェクトを表すモデルを構築することです。論理モデルが完成すると、物理データベースのモデルについて検討を始めることがで

きます。論理モデルと物理モデルを別々に保持しておくと、すぐれたデータベースを構築する上で役立ちます。

データベースの設計

アプリケーション開発では、通常、プロジェクトの非常に早い段階で、デザイナーやアプリケーション開発者が多くの時間をかけてアプリケーションの要件を収集します。これらの情報は、ユーザーやマネージャーへの聞き取り調査や、既存システム（手動のシステムも含む）の調査を通じて集められます。その結果、使用事例やシステムのダイアグラム、さらにはアプリケーション インターフェイスのモックアップに至るまで、かなり詳細な要件が揃います。この要件がプロジェクトの中心的なメンバーによって承認されると、アプリケーション開発が始まります。ほとんどのアプリケーション開発者は、最初に取り組む作業の一環として、新規アプリケーションで使用されるデータの保存場所、つまりアプリケーションのデータベースを作成します。

多くの場合、データベース開発は、アプリケーション インターフェイスの物理的な要件に基づいてすすめられます。つまり、アプリケーションでデータを取得して表示する仕組みが決まれば、アプリケーション開発者は RDBMS に空のデータベースを作成できます。テーブルを作成して、インターフェイスから入力されるすべてのデータを定義すると、アプリケーション開発者はデータの保存を開始し、新しく構築されたデータベースと連携するためのアプリケーション コードを記述できます。

このアプローチは決して理想的なものではありませんが、少なくとも短期間であればうまくいきます。データベースを更新する必要がなく、保存されるデータが少ない場合に限りますが、注意深く作業すれば、構築されたデータベースは正しく動作するでしょう。ただし、このような方法でデータベースを設計すると、拡張性に関する深刻な問題をかかえることになり、時間が経つにつれて修正作業が非常に困難になります。アプリケーションは、新機能を追加するか古い機能を削除するために、いずれ修正を加えるか書き直す必要が生じますが、古いデータは通常そのまま保持することが求められます。そこで、古いデータベースを部分的に再設計して、データ損失のリスクを避けつつ新機能を追加することになります。このような修正作業では、新しい構造（テーブルおよびビュー）や、新旧のデータを同じインターフェイスに表示するための複雑な SQL ロジックを既存のデータベースに追加することが多いため、パフォーマンスが低下してしまいます。データベースを開発する前に、適切な論理データ モデルを構築しておけば、これらの問題を避けることができ、さらにデータ層の要件で見逃していた点があっても、コードを記述する前に見つけ出すことができます。

データ モデルとデータベース

データ モデルを作成するタイミングは、設計の初期段階です。通常はプロジェクトの要件収集がほぼ完了する時点になります。ユーザーへの聞き取り調査や既存システムの調査が完了すると、データ モデルを作成して、アプリケーションで使用されるデー

タを記述していきます。モデルには、アプリケーションで必要となるすべての情報が論理的に表され、各データの関連性が記載されます。このモデルは、ビジネス ユーザーがデータを確認する際にも用いられ、関係者からアプリケーション設計の承認を得るために役立ちます。

厳密に言えば、リレーショナル データベースは、データを保存するテーブル一式を編成したものです。具体的には、データの保存とアクセスに使用するテーブル、ビュー、およびストアド プロシージャの集まりです（RDBMS の種類によって異なります）。これらの構造は、RDBMS の言語（通常は SQL の一種）によって定義されます。RDBMS では、オペレーティング システム内のファイルにデータが保存され、ファイル管理やセキュリティ管理の機能、さらにはデータ操作に用いるクエリーのパフォーマンス チューニング機能を備えています。アプリケーションの側から見ると、データベースはデータを入出力するための場所であると言えます。

データ モデルはデータベースとは違い、データの物理ストレージを表しません。データベースでは、データの保存方法や、そのデータの操作時に適用されるリレーションシップを定義し、プログラムを通じてデータにアクセスできます。一方、データ モデルでは、どのようなデータが存在し、各データがどのように関連しているかを記述するだけです。適切に設計されたデータ モデルは、開発する物理データベースの論理的な設計図になります。このような関係から、しばしばデータ モデルはプラットフォームに依存しないことが求められるため、Oracle 10g、SQL Server 2005、または MySQL など、さまざまな物理データベースを作成するために使用できます。ただし、データ モデリングと、最終的に使用される RDBMS の種類がまったく関係ないという訳ではありません。場合によっては、使用される RDBMS の種類がデータ モデリングのプロセスに影響することもあります。

データ モデリングの重要性

データ モデリングが重要となるのはなぜでしょうか? 第一に、すぐれたデータ モデルがあれば、パフォーマンスを向上させることができます。これは特に、オンライン トランザクション処理システム（OLTP）で顕著です。第二に、信頼できるデータ モデルがあれば、十分な情報に基づいて拡張性の高いデータベースを構築できます。適切なデータ モデルを作成できなければ、モデルに新しいデータを追加するときに問題が生じ、最終的には物理データベースにも悪影響を与えます。

パフォーマンス

適切なデータ モデルを作成すれば、RDBMS はすぐれたパフォーマンスを発揮できます。標準的なデータ モデリングの規則に従って作業すると、データの不整合（データの重複など）が生じることがなく、後からそのような問題に対処するためにアプリケーションに余分なロジックを追加する必要もありません。

さらに、データを構造化された形式で保存すると、フラット ファイルやその他の不適切な形式で保存した場合よりも、クエリー エンジンはデータを高速に検索および取得することができます。これによって、アプリケーションやレポート作成のパフォーマンスが向上します。

データのアクセシビリティ

すぐれたデータ モデルがあれば、データベース内のデータを理解するのに役立ちます。エンティティとテーブルを厳密に定義して、作業対象のデータを分類しておけば、レポート作成やデータ ウェアハウス構築によるデータ分析がはるかに容易になります。必要なデータが見つからなければ、利用することはできません。

十分に検討された信頼できるデータ モデルがあれば、物理データベースの開発時にその利点を実感できます。特に、複数のデータベース開発者が参加するような大規模プロジェクトでは、包括的なデータ モデルを作成しておくと、各開発者はそれぞれの担当するデータベースの一部を作業しつつ、データベース全体の構造を把握することができます。作業の重複を避けることができ、より柔軟に開発の人員を割り当てることができます。さらに、少なくともデータベースが関係する個所では、迅速かつ密接に開発作業をすすめることができます。

データ モデルの構築

モデルの構築にとりかかるには、あらかじめエンティティ、属性、およびリレーションシップについて理解しておく必要があります。これらの概念は、データ モデルの構築要素です。以下に、各要素の詳細と、これらの要素を使用した論理モデルの構築方法を説明します。

エンティティ

エンティティは、データ モデルでもっともよく使用されるオブジェクトです。エンティティは、あるデータ型のグループを表します。たとえば、「顧客」、「注文」、「売上」、および「製品」は、すべてエンティティの例になります。論理的には、各エンティティは特定のデータ型を持つインスタンスの集まりです。インスタンスは、そのデータに属する 1 つの事象を表します。物理データベースの行とテーブルの関係と比較すると、テーブルの行が、エンティティのインスタンスに対応します。

テーブル/行とエンティティ/インスタンスの間には似ている点がありますが、エンティティとテーブルは同じものではありません。エンティティはデータ型を論理的に表したものであり、テーブルはデータの物理ストレージ構造です。最終的に、1 つのエンティティが複数のテーブルとして物理的に実装されたり、逆に、1 つの物理テーブルが複数のエンティティに対応することもあります（ただし、後者は稀なケースです）。

データ モデルの要件を検討してエンティティとなる情報を決定するには、顧客、ベンダー、製品などの特定のキーワードを見つけ出します。通常、これらのキーワードは、ユーザーへの聞き取り調査や事前調査のレポートで名詞として表されます。基本的には、名詞がエンティティになると考えてもよいでしょう。項目やオブジェクトとして参照されるデータ型があれば、通常はそれらもエンティティになります。

属性

属性は、エンティティ内のインスタンスに関する記述子です。通常、エンティティには複数の属性が含まれています。たとえば、「犬の首輪」という名前のエンティティを作成したとします。これはペットショップで販売する犬の首輪を表しています。各首輪の色、長さ、幅など、一連の物理的な属性は、［犬の首輪］エンティティに含まれる［色］、［長さ］、［幅］、［ブランド］などの属性で表されます。データ モデルを構築するときは、各エンティティに必要な属性のセットを定義して、エンティティに保存されるデータを記述します。

属性を検討する際に問題となるのは、ある属性がどのエンティティに属しているのかを決定することでしょう。この点を間違いやすいのは、これまでアプリケーションが関わっていなかった物理プロセスに基づいて、新規アプリケーション（およびそのデータ モデル/データベース）を設計するときです。よく取り上げられる小売店の例では、電話番号や住所のような顧客情報は、注文を受けるたびに頻繁に保存されます。データ モデルを構築する際には、スプレッドシートなどにあらかじめ記載された情報からエンティティを作成することがありますが、そのような場合、たとえば［注文］エンティティに顧客の属性を含めてしまうことがあります。これは論理的には誤りです。注文はデータの論理的な単位、つまり論理オブジェクトであり、顧客も同じく論理オブジェクトです。したがって、顧客の電話番号や住所を表す属性は、［注文］エンティティではなく［顧客］エンティティに含める必要があります。

リレーションシップ

リレーショナル データベースの基本概念は、同じデータベース上で、あるデータが別のデータに関連付けられているというものです。リレーションシップを定義せずにデータベースを使用するのは、領収書や給与小切手、預金残高証明などを大きな袋にまとめて保管するようなものです。収入や支出を管理したり、納税手続きをするには、これらの書類がどのように関連付けられているかを把握しなければなりません。リレーションシップが存在しないデータ モデルも、これと同じような状況です。各エンティティとその属性を他のエンティティと属性に関連付けて、これらのリレーションシップを記述する必要があります。

データ モデルの論理リレーションシップには、「1 対 1」、「1 対多」、および「多対多」という 3 つの基本タイプがあります。各リレーションシップは、2 つのデータが互いに関連付けられる方法を表します。

1 対 1 リレーションシップ

おそらく最も理解しやすいリレーションシップですが、残念ながら使用頻度が一番高いというわけではありません。このリレーションシップでは、親エンティティの各インスタンスが、子エンティティの各インスタンスに 1 対 1 で関連付けられます。例としてキャッチボールを考えてみましょう。ボールを投げる側もボールを受ける側も、ただ 1 人に限られます。

1 対多リレーションシップ

最も一般的なリレーションシップであり、ほとんどの人にとってよく知られたタイプです。1 対多リレーションシップでは、親エンティティの 1 つのインスタンスが、子エンティティの複数のインスタンスに関連付けられます。実際には、このタイプのリレーションシップは「1 対 1 以上」という名前で呼ばれることがあり、子エンティティの関連インスタンスが 1 つであっても許可されます。ただし、子の関連インスタンスは複数あるのが普通です。これは物理データベースをモデル化する際に重要な意味を持ちます。機能定義では、親テーブルと子テーブルのレコード数をそれぞれ明確に指定する必要があるためです。1 対多リレーションシップの簡単な例は、［注文］エンティティと［注文明細］エンティティからなるモデルです。［注文］エンティティ（親）の注文インスタンスには複数の品目を指定できますが、各品目はすべて［注文明細］エンティティ（子）のインスタンスです。

多対多リレーションシップ

3 つの基本タイプのうち、理解するのが最も難しいリレーションシップであり、モデルで一番複雑な構造だと言えます。多対多リレーションシップでは、親エンティティの 1 つ（または複数）のインスタンスを、子エンティティの 1 つ（または複数）のインスタンスと関連付けることができます。逆も同様です。例として、自動車部品のデータ モデル、特にセダン型の自動車の座席について考えてみましょう。セダンの座席には複数の種類があり、運転席と助手席にはバケット シート、後部座席にはベンチシートが使用されます。データ モデルには、セダンに対し［車種］エンティティ、座席に対し［座席］エンティティがあります。一見すると、［車種］エンティティから［座席］エンティティへの 1 対多リレーションシップがあるように思えますが、多くの自動車メーカーでは、異なる車種で同じ座席を使用しています。つまり、［車種］の複数インスタンスと［座席］の複数インスタンスの間にリレーションシップがあります。これが基本的な多対多リレーションシップです。

ここで説明しているのは論理モデル用のリレーションシップであることに注意してください。データベース内で多対多リレーションシップを物理的に実装するには、3 つ目のオブジェクト（テーブル）を使用して、このリレーションシップを保持するための規則を適用する必要があります。はじめから 3 つ目のエンティティを使用して、このリレーションシップをモデル化するのも良いでしょう。そうすれば、多対多リレーションシップを完全に記述しつつ、物理データベースも容易に実装できます。

ドメイン

データ モデルの作成中に各エンティティに属性を定義していくと、ほとんどの場合、複数のエンティティ間で共通の属性が見つかります。それらの属性は一般的なデータ型であるためです。たとえば、顧客、ベンダー、および従業員の住所が必要な場合は、それぞれのエンティティに住所を保存します。このような共通の属性に対してドメインを作成すると、データの一貫性を確保することができます。ドメインとは、特定のエンティティに定義するものではなく、論理モデル内に保存する属性定義です。あらかじめドメインを作成してから、関連するエンティティに追加していきます。エンティティに追加されたドメインは属性として表示されます。ドメインから分離しない限り、その属性をエンティティ内で編集することはできません。そのため、共通の属性を含むエンティティ間で、データの一貫性を確保するのに役立ちます。上記の例では、住所ドメインを作成して、住所フィールドのデータ型とデータ長を定義することができます。ドメインを利用すると、どのエンティティでも住所データの入力形式が同じになるため、アプリケーション コードの記述がいくらか容易になるという利点もあります。

正規化

正規化とは、論理的な手法でデータをグループ化し、冗長性を減らしてデータ構造を単純化するプロセスです。正規化には複数のレベル（正規形）があり、レベルが進むほど高度なものになります。以下に説明する各正規形についての知識を身につけると、実際に正規化を行う際の注意点を理解することができるでしょう。

正規化の規則は、IBM の研究者であった E. F. Codd によって提唱されました。これらの規則をデータ モデルに順次適用していくと、モデルはより厳密に正規化されていきます。もともとは 3 つの形式がありましたが、研究がすすむにつれて更新時異常という問題が明らかになり、その問題を解消するため、さらに 2 つの形式が定義されました。これらの形式を正規形と呼び、1 つの例外を除いて、適用する順序に基づいた簡単な名前が付いています。

第 1 正規形（1NF）

データ モデルが第 1 正規形となるには、すべてのエンティティに主キーが必要です。エンティティの主キーは 1 つ以上の属性で構成され、そのエンティティのインスタンスを一意に識別します。主キーはすべてのインスタンスに対して定義されるため、NULL 値を取ることはできません。さらに、これらの主キーに対応するインスタンスは同一の値を取ることはできません。

また、第 1 正規形には繰り返しグループがありません。繰り返しグループとは、同じインスタンス内で、ある属性に含まれる複数の値が、別の属性に含まれる 1 つの値に関連付けられていることです。たとえば、［アーティスト］エンティティにアーティスト情報を保存する場合、リリースされているアルバム名も保存する必要があるでし

ょう。アーティストが複数のアルバムをリリースしている場合は、［アルバム名］属性を作成すれば、各アーティストのすべてのアルバム名を保存できますが、同じインスタンス内で、複数の値（［アルバム名］属性）と 1 つの値（［アーティスト名］属性）が関連付けられることになります。この繰り返しグループを解消するには、アルバム名を表すエンティティを作成して、［アーティスト］エンティティと［アルバム］エンティティの間にリレーションシップを作成します。

第 2 正規形（2NF）

第 2 正規形では、すべての非キー属性が主キーに完全従属します。エンティティの主キーが 1 つの属性から構成される場合、これは自明であり、問題にはなりません。しかし、エンティティに複合主キー（複数の属性から構成される主キー）がある場合、第 2 正規形では、エンティティ内のすべての非キー属性が、主キーを構成するすべての属性と関連している必要があります。主キーの一部のみに関連している非キー属性がある場合、その属性を別のエンティティに含めて、リレーションシップの外部キーを通じて元のエンティティに関連付けます。外部キーとは、子エンティティの非キー属性のうち、親エンティティの主キー属性と関連している属性です。

簡単な例として、製品情報を保存するエンティティを考えてみましょう。たとえば、［製品］エンティティの主キーが［製品 ID］、［製品名］、および［倉庫 ID］（製品の保管場所を表す ID）であるとします。このエンティティの非キー属性は、［倉庫の住所］、［製品の説明］、［ベンダー名］、および［ベンダーID］です。倉庫の住所は、そこに保管される製品とは無関係なので、［倉庫の住所］属性と［製品名］属性が関連していないのは明らかです。そこで、［倉庫］エンティティを作成し、［製品］エンティティとの間にリレーションシップを作成して、各製品が保管される場所を定義します。

第 3 正規形（3NF）

ほとんどのデータ モデルとデータベースは、このレベルまで正規化されます。第 3 正規形は、第 2 正規形のデータ モデルから推移従属性を取り除いたものです。推移従属性とは、エンティティ内のある非キー属性が、別の非キー属性に従属している状態です。つまり、ある属性が、主キーを構成しない別の属性に従属している場合、その属性には推移従属性があります。

推移従属性を解消するには、第 1 および第 2 正規形と同じように、新しいエンティティを作成して、正規化の規則に違反する属性を、そのエンティティに移動します。先に挙げた例では、［倉庫］エンティティを作成すると、倉庫に関するすべての情報をそのエンティティに移動できます。しかし、ベンダーの情報は［製品］エンティティに残っています。製品を販売するベンダーの名前を知ることは、論理的には関連性がありますが、このままでは、ベンダーの情報を効率的に保存することはできません。この場合、［ベンダー名］は［ベンダーID］という非キー属性に従属しています。

この問題を解決するには、［ベンダー］エンティティを作成して、ベンダーに関するすべての情報をそのエンティティに移動します。［製品］エンティティの［ベンダー ID］属性は残しておき、2 つのエンティティ間にリレーションシップを作成します。

通常、データベースを第 2 正規形にすると、自然に第 3 正規形になっていることがよくあります。この理由は、エンティティの作成時に、各エンティティに関連する属性のみを含めて、さらにリレーションシップの作成時に、他のエンティティに関連する属性のみを子エンティティに追加して、親エンティティのみに関わる属性は追加しないためです。ただし、物理データベースで更新時の問題が発生しないように、モデルに推移従属性がないことを必ず確認するようにしてください。

ボイス-コッド正規形（BCNF）

ボイス-コッド正規形は、第 3 正規形のデータ モデルで発生する可能性がある問題を解決するために定義されました。基本的に、第 3 正規形は 1 つのエンティティに複数の候補キー（単一または複合）があってはならないことを明示的に定めていません。エンティティに複数の主キーがある場合、そのエンティティを分割する必要があることは明らかですが、ボイス-コッド正規形は、この規則を明文化したものです。

その他の正規形

正式には、さらに 2 つの正規形（第 4 正規形と第 5 正規形）があります。第 4 正規形では、主キーは多値従属性を持つことはできません。多値従属性とは、ある複合主キーを構成する 1 つの属性が、その主キーを構成する別の属性の複数値と関連している状態です。他のほとんどの正規形と同じく、この問題を解決するには新しいエンティティを作成します。

第 5 正規形は、しばしば 3 項リレーションシップと呼ばれる、3 つ以上のエンティティ間のリレーションシップを扱います。第 5 正規形では、リレーションシップが指定されたエンティティは、その他のリレーションシップに従属せず個々のエンティティとして独立していなければなりません。ただし、これらのエンティティは互いに関連しているため、第 5 正規形では通常、他のエンティティを互いに関連付けて従属性を解消するための物理エンティティが必要です。この特別なエンティティには 3 つ以上の外部キーがあり（リレーションシップに関わるエンティティ数に基づきます）、エンティティ間の関連付けを指定します。多対多リレーションシップは、このようにして実装されます。そのため、多対多リレーションシップが正しく実装されると、データベースは第 5 正規形になります。

時空間データベースのような分野では、さらに正規形に関する作業が必要になります。ただし、そのようなケースでは決まった手法が確立されていないため、ここでは扱いません。ほとんどのリレーショナル データ モデルおよびデータベースでは、ここで説明した正規形が確実で有効な解決策になります。

非正規化

データベースを適切に正規化すると、受注システムなどのオンライン トランザクション処理（OLTP）データベースで高いパフォーマンスと柔軟性を実現できますが、特定の状況では、パフォーマンスを向上させるためにエンティティ/テーブルを非正規化する必要があります。レポートの作成やデータ ウェアハウスの構築では、クエリーの実行時に大量のデータが結合および集計されます。完全に正規化（第 3 または第 4 正規形）されたデータベースに対して、このようなクエリーを実行すると、多数のテーブルを結合しなければならないため、処理のオーバーヘッドが増加してパフォーマンスが低下します。この問題に対しては、複数のエンティティ間で、それぞれの属性が論理的にゆるやかな関連性を持つ場合、それらのエンティティを 1 つのテーブルにまとめると効果的です。通常は、テーブルが非正規化された別のデータベースを作成した後に、データの読み込みプロセスを用意して、データを正規化データベースから非正規化データベース（レポート作成やデータ ウェアハウス用）に移動します。このような場合に非正規化をためらう必要はありませんが、作業を行う際は、元のオブジェクトとは別に新しいオブジェクトを作成するようにしてください。OLTP システムで使用中のオブジェクトを非正規化すると、パフォーマンスに悪影響を与えてしまいます。

まとめ

データ モデリングは、データベースを正しく設計するために重要な役割を果たします。開発作業を始める前に、要件を収集してデータ モデルを設計すると、データベースとアプリケーション コードの両方で、厳密なデータ定義を行うことができます。これによって、曖昧さを減らして作業効率を高めることができます。さらに、データベースを適切に正規化することで、オンライン トランザクション処理（OLTP）データベースの読み取り/書き込みのパフォーマンスが向上します。

Josh Jones と Eric Johnson は、Consortio Services (www.consortioservices.com)に所属しており、それぞれオペレーティング システムとデータベース システムのコンサルタントです。Eric と Josh は『Architecting Database Models for SQL Server』を含む 3 冊の書籍を執筆しています。