データベース 2009 #2

久野　靖*

2009.10.15

はじめに

さて今回の内容ですが、「データベース設計」の話題を取り上げます。その後、設計課題を説明しますが、時間が余れば前回ほとんどやらなかった Join の話も少し用意しました。

1　データベース設計の概要

1.1　データ設計の必要性とアプローチ

データベース (前回から言っているように、現在の情報システムで最も多く使われているのは関係データベースなので、以後ずっと関係データベースを想定して頂いて構わない) の概要や機能や SQL による操作については一応分かったものとしましょう。

それはいいとして、実際の情報システムにおいては、「どういう表を作り、それぞれの表にはどのような属性を持たせ、それらの間にはどのような関連づけを持たせるのか」を決めなければならないのは当然です。それには具体的にはどうすればいいのでしょうか? つまり、データベース設計はどうやって行うべきなのでしょうか?

QUIZ: データベースのテーブルや属性などはどうやって決める?

1 つの明らかな方法は、もともとデータベースの用途は「アプリケーションから見てデータを格納する場所」だということに基づくものです。つまり、アプリケーションを開発するに当たって、それが必要とする (記録したり参照する) データを列挙し、それをもとに表を作る、というものです。

これでもまだ抽象的だと思うのなら、実際にアプリケーションが扱う業務の伝票や請求書などを収集してきて、そこにあるデータに基づいてデータを列挙することもできます。このように、現実の業務やアプリケーションの仕様などに基づいてデータ設計を行うことを**ボトムアップアプローチ**と言います。

QUIZ: ボトムアップアプローチの利点は何か? また弱点は何か?

もちろん、請求書そのままを記録しようとするとうまく表の形にできないことがありますし、表の形になったとしても、もっと分解した方が良い場合もあります (正規化の問題…これについては後でまた説明します)。

しかし、ボトムアップアプローチの最大の問題は、それが「現実にあるもののつぎはぎ」であり、次のような弱点を抱えていることです。

*経営システム科学専攻

- 複数のサブシステム間できちんと整合性が取れているかどうか疑問である。もともとデータベースは「あらゆるデータを 1 個所に置いて共有する」ものなので、その中に個別アプリむけにバラバラに表を用意していたのでは共有の利点がなく無意味。
- さらに問題なのは、データベースは「将来の業務の進化や、将来になってから作られるアプリケーションまでをサポートした」ものだということ。これら「まだない」ものはボトムアップではどこからも情報が来ないため、対応できない。

そこで、これと対照をなすアプローチとして**トップダウンアプローチ**が存在しています。トップダウンアプローチでは、まず対象とする業務の動き方のモデルを考え、そのモデルが動くために必要なデータは何であるかを分析し列挙していくことを通じて、データ設計を行うものです。

QUIZ: トップダウンアプローチの利点は何か? また弱点は何か?

トップダウンアプローチでは、「あるべき理想的な業務のあり方」を考えてそれに基づくデータ設計が行えますから、各種の業務がうまくその中にあてはまる、統一的なビジョンの設計を提供可能です。ただし、それには業務の動き方のモデルがうまく現実 (ないしその改善されたもの) と整合している必要があるわけですが。

そして実際には、現存する伝票などのデータも無視することはできませんし、想像だけできちんと細かい詳細を詰めることも難しいですから、トップダウンとボトムアップの双方を併用したシステム設計を行うのが通常の姿だと言えるでしょう (ボトムアップオンリーはあるかも知れませんが、上記の問題点のためあまり望ましくないと言えます)。

QUIZ: 業務モデルって何ですか? あなたは業務モデルを持ってますか?

1.2 業務モデルの必要性

しかし、業務の動き方のモデル (縮めて**業務モデル**と呼びます) とは何でしょう? たとえば営業担当者であれば、上司による客先の割り当てや売り上げ目標、重点商品などの指針に基づき、客先を回って売り込み、引き合いがあれば在庫や配送状況を調べて価格交渉を行い、契約を取りますが、それらの活動それぞれについて状況をデータとして蓄積したり、必要な参照データ (在庫、配送、価格現況)、生成データ (契約内容、商品の発送先や発送日時) を読み書きする必要があります。

営業なら大まかな枠組みは上のようかも知れませんが、細かい情報の動き方は商品や業界慣行、企業文化によってすべて違っています。そして、実際に行っている仕事に対応したデータがサポートされなければ困るわけです。これらをきちんと同定して「どこで何が起こってどんなデータが出入りする」ことを明確化したモデルが業務モデルです。

しかも、実際の仕事は営業だけでは済まず、製造、出荷、経理など多種の業務が相互に組み合わさって会社の仕事が達成されているわけですから、企業全体の業務モデルは極めて複雑になります。いちばん望ましいのは、企業活動全体についての業務モデルがまず存在していて、それを持って来て必要な範囲の分析を行うことです。しかし現実には、システム設計者が必要な範囲の業務モデルをそのつど分析/構築してそれに基づいてデータ設計を行うのが普通、といったところでしょうか。

QUIZ: ERPって何ですか?

なお、ERP(Enterprise Resource Planning) とは、企業全体の業務モデルを「パッケージソフトウェア (SAP が有名) に一定範囲のカスタマイズを施したしたデキアイのものとして」システム側から提供してし

まい、それに企業活動の方を合わせることで企業全体の業務のコンピュータサポートを行うことだ、と考えることができます。(むちゃくちゃ大変そうですが…)

QUIZ: DAって何ですか?

全社的な業務モデルと関連して必要なのは、各所で使われているデータの呼び名を標準化し統一することです。なぜなら、同じものを A 部門では「顧客レコード」、B 部門では「顧客情報」と呼んでいたら、それが同じものであることがよく分からないからです。そこで、このような名前の統一を行い、さらにそのデータの内容についても同じになるように調整する作業が必要になります。このような仕事を専門に行う人を、**DA**(Data Administrator、ないし Data Architect) と呼びます。DA は企業内の各部署に渡って使用する名前やデータ構造の内容を制定し従わせるという、強い権限を持つことになります。既に学んだ **DBA**(Data Base Administrator、データベース管理者) よりさらに上位の仕事と言えるでしょう。

1.3 業務モデルの図法 IDEF0

情報システムの世界でモデルといった場合、さまざまなモノやそのつながり方を表すため、特定の図法を用いてあらわす、ということが多く行われています。たとえばプログラムの実行の流れを表すのには、過去においては**フローチャート**が多く使われて来ました (が、今では下火になっています。なぜか分かりますか?)。

業務モデルなど各種のモデル化のアプローチは、プロセス指向アプローチ、すなわち「どのような処理/プログラミングを行うか」に焦点を当てるものと、**データ主導アプローチ**(DOA, Data Oriented Approach) に大別できます。プロセス指向アプローチに使われる図法としては、今日では開発に広く使われているオブジェクト指向言語と親和性のよい **UML**(Universal Modeling Language) が一般的になっています。

データ主導アプローチでも UML は適用可能ですが、ここでは DOA で多く使われている業務モデルの図法として、米国空軍が情報システムの仕様書を標準化するために実施したプロジェクト (ICAM、Integrated Computer Aided Manufacturing program) のための定義用図法 **IDEF**(ICAM Definition) の中の **IDEF0** を紹介します。

IDEF0 では、業務の中の個々の活動を**アクティビティ**(activity) と呼び、長方形で表します。そして、アクティビティに関連する情報や制御やモノの流れを次の 4 種類に分類し、4 辺のそれぞれに矢線で記述します (図 1)。ちなみに、アクティビティを行う主体や、矢線上を流れる情報/モノなどをまとめて**エンティティ**(entity、「もの」という意味) と呼びます。

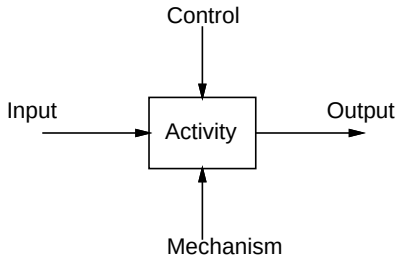

図 1: IDEF0 のアクティビティ

IDEF0 におけるアクティビティでは、上下左右に接続されるものはそれぞれ次のように決まっています。

- Input — 左。そのアクティビティの入力となる (主に処理される、また加工され得る) もの。「何を、どこから入力」を表現。
- Output — 右。そのアクティビティの出力となる (結果として出て来る) もの。「何を、どこへ出力」を表現。

- Control — 上。そのアクティビティが参照する (書き換えたり加工はしない) もの。「何を、どこから入力」を表現。
- Mechanism — 下。「誰が」「何を使って」「どういう方法で」「どういう要件」などの情報を記述。

たとえば、販売側が確定注文に応じて商品を配送業者に依頼して購入側に送るという業務フローを IDEF0 で記述すると図 2 のような感じになります。1)。

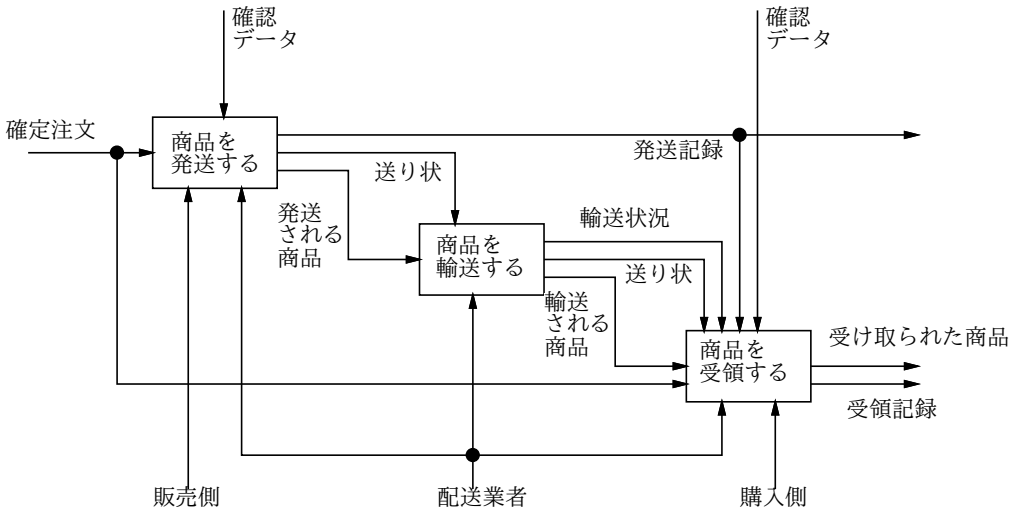

図 2: IDEF0 による記述のサンプル

具体的な各アクティビティの説明は次のとおりです。

- 「商品を発送する」では、販売側が、確定注文を確認データと突き合わせた上で、商品を送り状とともに配送業者に引き渡し、発送を記録する。
- 「商品を輸送する」では、配送業者が、発送される商品と送り状を輸送して受け取り側に届けますが、問い合わせがあれば輸送状況もそのつど返答する。
- 「商品を受領する」では、購入業者が配送業者から、商品と送り状を受け取り、確定注文の控え、および確認データと照合し、受領を記録する。

実際には、さらに「発送」「受領」「輸送」の中も複数のアクティビティから成っていると考えられ、それぞれの四角の中をさらに複数の四角に分解したモデル図も必要ならそれぞれ作るかも知れません。かなり面倒そうですが、このようにして業務の流れを**すべての**出入りする情報と一緒に記述できれば、それらの情報のどれとどれをデータベースに格納し、その中には何が含まれているかを検討する役に立ちそうだということはお分かり頂けると思います。

1.4 データフロー図

データフロー図(DFD、Data Flow Diagram) とは、アクティビティ、データストア (データベースその他)、外部要素 (顧客などシステム外のパーティや伝票など外部とやりとりするもの) 相互間のデータの流れを記述する図であり、業務モデルを詳細化してデータを抽出するために描かれます。(これ以外に、システム全体の処理や構成を概観するためにも DFD を使うことがありますが、それはシステム開発系の話題なのでここでは触れません。)

図 3 に、例題として皆様に構築して頂く予定の、ネット書店 (Web 経由で顧客から書籍の注文を受け付け、クレジットカードで支払いを受け、宅配便で発送する) の DEF を描いてみたものを示します (分析する人によってこれと違う形になることもあるかも知れませんので、あくまでも例のつもりです)。

その概要を以下に説明しておきます。

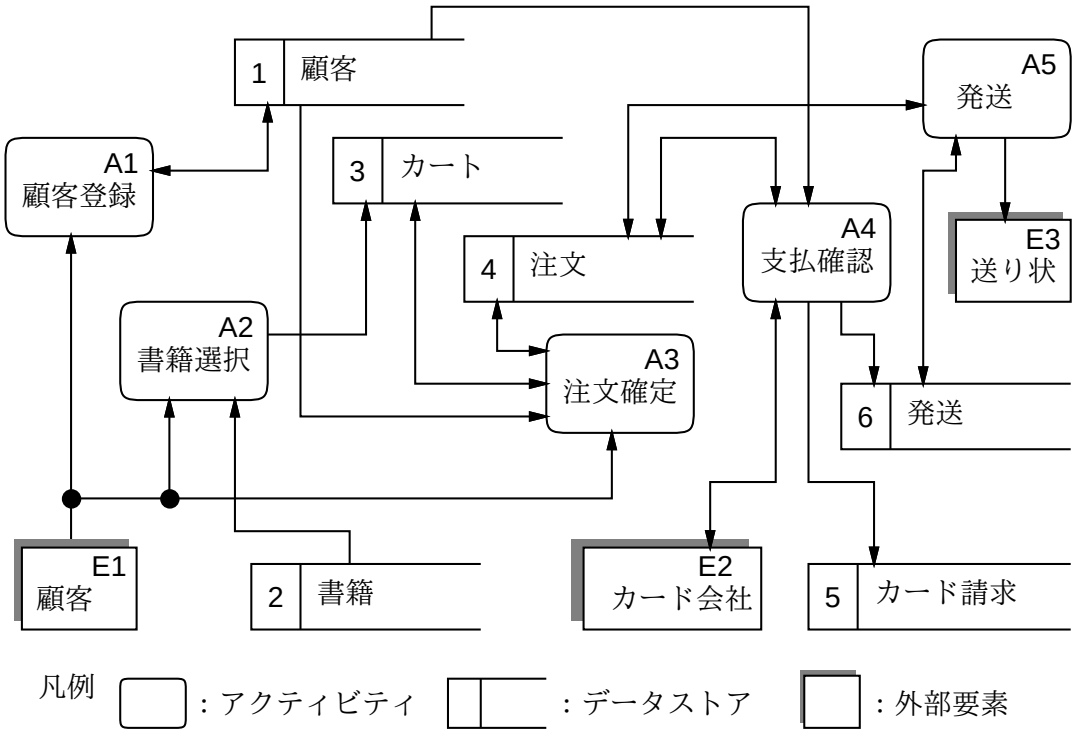

図 3: DFD による記述のサンプル

- A1:顧客登録 — 「E1 顧客」は最初に顧客登録を行う。ここで氏名などに加え、配送先、クレジットカード情報なども入力してもらい、すべて「1 顧客」として記録する。登録は最初の 1 回だけでよい。また、いちど登録した情報を修正することもある。
- A2:書籍選択 — 「E1 顧客」は「2 書籍」にある書籍をブラウズし、購入したいものを選択し「3 カート」に加えていく。
- A3:注文確定 — 「E1 顧客」は「3 カート」にある商品群と使用するカードや配送先などの情報を確認し、OK ならこれらのデータをもとに「4 注文」を生成する。カートの内容は空になる。
- A4:支払確認 — 「4 注文」の情報を参照し、クレジットカード会社に照会の上、支払い確認する。OK であれば「5 カード請求」に支払い内容を記録し、また「6 発送」に発送すべき注文の情報を書き込み、「4 注文」は発送可能状態に更新する k。
- A5:発送 — 「6 発送」にある未発送の注文について、「5 注文」から商品、配送先の情報を取り出し商品を梱包して、作成した「E3 送り状」とともに宅配業者に渡して発送する。「6 発送」の対応するデータを発送ずみに更新する。

ここでは簡単化のため、顧客のからまない A4、A5 を中心にかなり省略があります。現実にはもっと色々なアクティビティやエンティティが必要になるでしょう。

QUIZ: 図 3 で現実には必要と思えるのに省略されている事柄としてどんなものが思いつきますか。

1.5 CRUD 分析

CRUD 分析とは、個々のデータについて「生成 (Creation)」「参照 (Read)」、「更新 (Update)」、「削除 (Delete)」がどこで起きるかを表の形に記入してチェックすることを言います。具体的には、縦の列が各ア

クティビティ(生起順)、横の行が各データであるような表を作成し、各データについてそれを生成/参照/更新/削除するアクティビティとの交点に C/R/U/D の文字を記入して行きます。たとえば、図 3 のデータとアクティビティについてこれを行った例を図 4 に示します。

	A1 顧客登録	A2 書籍選択	A3 注文確定	A4 支払確認	A5 発送
1 顧客	C/U		R	R	R
2 書籍		R	R	R	R
3 カート		C/U	R/U		
4 注文			C	R/U	R/U
5 カード請求				C	
6 発送				C	R/U

図 4: DFD に対応する CRUD 分析

QUIZ: これを見て何かおかしいと思ったところはないでしょうか?

一般には、CRUD では次のような点をチェックします。

- C のないデータがないか。どこでも生成されないデータというのはおかしい。
- C だけで R/U のないデータがないか。使われないデータを生成するというのはおかしい。
- R/U/D は C よりも「後の時点」になっているか。作る前に使うというのはおかしい。
- 1 箇所だけに C/R/U/D が集中していないか。その他偏りはないか。

特に C については、そのデータに主として責任を持つアクティビティを意味することになるので、十分に検討する価値があります。

なお、C とは対照的に、D については、現れないことは多くあります。というのは、一度生成したデータは削除してしまうのではなく、ずっと残しておいて分析に使うのが普通だからです。もちろん、無限に蓄積していてはデータベースがパンクしますから、定期的に古いデータを吸い上げて保管してから削除するといった業務を入れますが、それはこの分析の外ということです。

このような視点で図 4 を見ると、次のような点がおかしいと分かります。

- 「2 書籍」はどこにも C がない。
- 「5 カード請求」は C だけで他の使用がない。

これは、前者についてはこの DFD の範囲外のどこかで書籍データを作ってくれているという前提で練習問題を作っているので、これでよいことにします。また、後者については、請求記録をチェックするといった業務があるのが正しいことを示唆していますが、この DFD の範囲ではそこは簡略化してあるというのでいいことにしましょう。

「いいことにした」のでは意味がないと思うかも知れませんが、おかしい可能性のあるところを拾い出して検討するプロセスを提供するという点で CRUD 分析に意味があるわけです。なお、ここでは説明の都合上、データについて細かい検討をする前に CRUD 分析をしていましたが、実際にはもっとデータについて詰めた後にやるのが普通かと思います (データについて詰めながら繰り返し実施してもよい)。

また、データ設計が進んだ時点では、全体としてのデータでなくその個々の項目、たとえば「顧客の中のメールアドレス欄」「注文の中の請求完了欄」などについて同様に分析をすることもできます。その場合は、そのデータが「生成 (Create)」ではなく「挿入 (Insert)」された時点、また「削除 (Delete)」でなく「NULL 値設定」された時点を見るため **IRUN 分析**と呼びます。

1.6 データモデルの図法 IDEF1X

ではいよいよ、データモデルを検討する段階に進みましょう。ここでは図法として IDEF ファミリーの図法である **IDEF1X** を用います。データモデルは一般に、エンティティ(さまざまな対象データ) と、それらに間のリレーションシップ (関係) とを記述するので、その図法は **ER 図**(Entity-Relasinship Diagram) と呼ばれることもあります。なお、ER 図にもさまざまな流儀のものがあるので (UML のクラス図を ER 図のように使うこともできます)、その 1 つが IDEF1X ということです。

IDEF1X では、エンティティには次の 2 種類の形があります。

- **独立エンティティ** — 長方形で表し、それ自体単独で存在し得るようなデータに対応。
- **依存エンティティ** — 角の丸い長方形で表し、いずれかの独立エンティティと関連づけられてのみ存在するようなデータに対応。

たとえば、学校 (小中高校ないし大学) の生徒/学生とクラスの関係を考えます。高校までは、生徒にはクラスごとの「出席番号」だけがあることが多く、その場合はまずクラスを決めて、それのクラス内での出席番号を指定することではじめて、生徒が特定できます。つまり生徒はクラスなしには存在できない依存エントリです。

一方、大学では学生は学籍番号を持っていて、クラス (語学のクラスなど) に学生が所属しているという関係自体はありますが、クラスに入っていなくても特に困りません。つまり、学生は独立エンティティになります。この両者を図示したものを図 5 に示します。

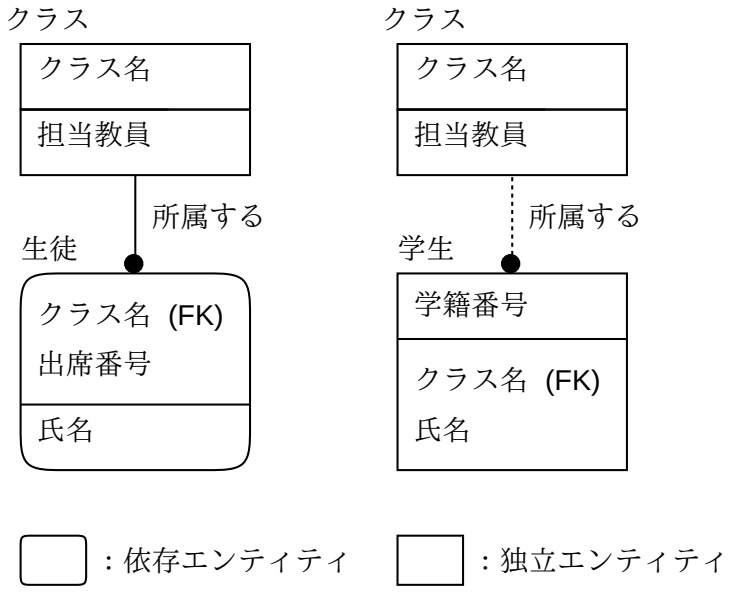

図 5: 依存エンティティと独立エンティティ

ここでもう少し記法の説明をしましょう。エンティティの上側に名前、箱の中にその要素名の並びを書きますが、横線が引いてある場合は線の上側にある要素 (列) がそのエンティティを一意に識別します。つまり、RDB 的に言えば主キーとなります。たとえばクラス名はクラスを識別する主キーです。また、生徒はクラス名と出席番号を連結したもので一意に識別できるので、これらが主キーです。

主キー以外の要素の性質はかっこ書きで記述しますが、次のものがあります。

- FK (Foreing Key) — 他のデータの主キーを格納している。
- AK (Alternate Key) — 代替キー、つまり主キーとはしていないが、このデータを用いてエンティティを一意に識別することもある。
- IE (Inversion Entry) — エンティティを一意に識別するという性質はないが、このデータを用いてエンティティを検索することがある。

エンティティ間の関係 (リレーションシップ) は線で示していますが、その線が「必ずある」ときは実線、「ない場合もある」ときは点線で示します。図 5 の場合は、生徒は必ずクラスに所属しているが、学生はクラスに所属していない場合もあるわけです (依存エンティティは必ず親に相当するエンティティと実線で結ばれます)。線についている黒丸は、その黒丸がある側のエンティティが複数対応することを表します。生徒も学生もクラスに複数名が所属するのでこうなるわけです。そして、「どういう関係か」を示す動詞句を書いておくとその関係の意味が分かりやすくなります。

黒丸の横に記号や数を書いて「何個対応している」という情報をさらに表すことができます。具体的には次のものがあります。

- ● — 0 個以上
- ● P — 1 個以上
- ● Z — 0 個または 1 個
- ●数 — その数以上
- ◇ — 0 個または 1 個 (点線にのみ使う)

また、IDEF1X ではあるデータを複数通りのどれかに分類する場合には図 6 のような書き方を使います。ここで 3 つの記法の意味はそれぞれ次の通りです。

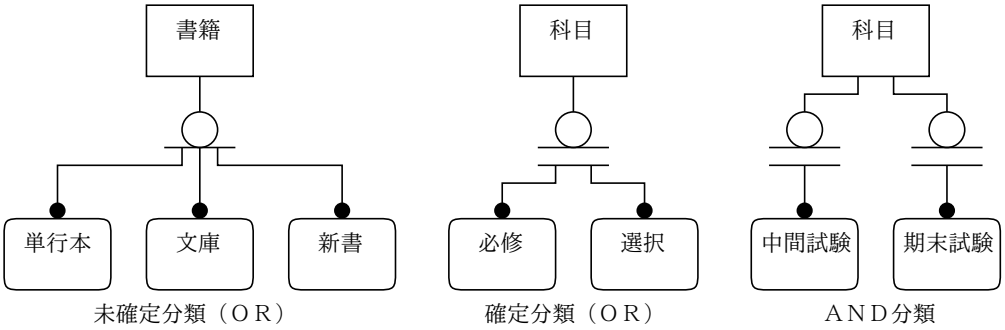

図 6: IDEF1X の分類の記法

- 未確定分類 — 親要素が複数の場合 (下に描かれた依存エントリ) のどれかに分類 (=対応づけ) されるが、ここに描かれたものが全部ではなく、これ以外の場合もあり得る。
- 確定分類 — 上と同様だが、ただしここに描かれたものが全て。つまり、親要素は必ず複数の場合のどれかちょうど 1 つと対応する。
- AND 分類 — 親要素が下に描かれた依存エントリのそれぞれと対応することもしないこともある。つまり、2 つ以上と対応することもある。

ではもっと実際のデータベースに近い例として、前回やった部品販売データベース (図 7 に再掲) の 4 つのリレーションを記述してみましょう (図 8)。

ここにはデータモデルでよく使われるパターンが 2 つ現れています。1 つは、「明細」は伝票と商品の両方に依存したエンティティで、1 つの伝票に複数の商品を対応づけ、さらに余分のデータ (数量) を付属させています。もう 1 つは、「顧客」が伝票が必要とする顧客のデータを保持していますが、伝票自体は独立エンティティで、顧客から見て対応する伝票が 1 つもないこともあるという形になっている点です。

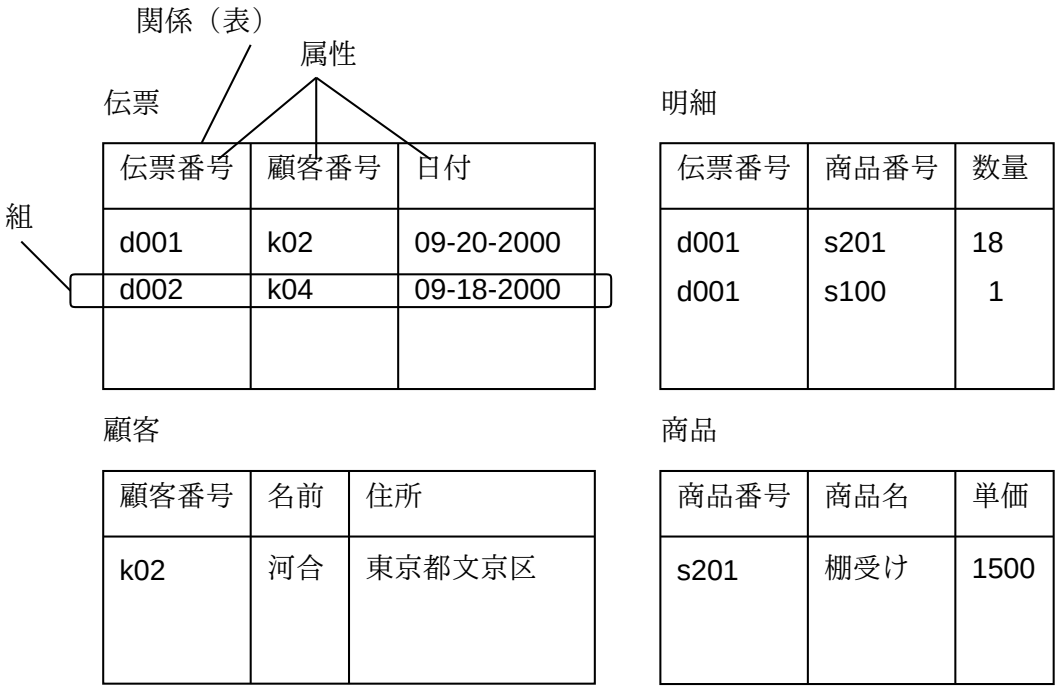

図 7: 部品販売データベース (再掲)

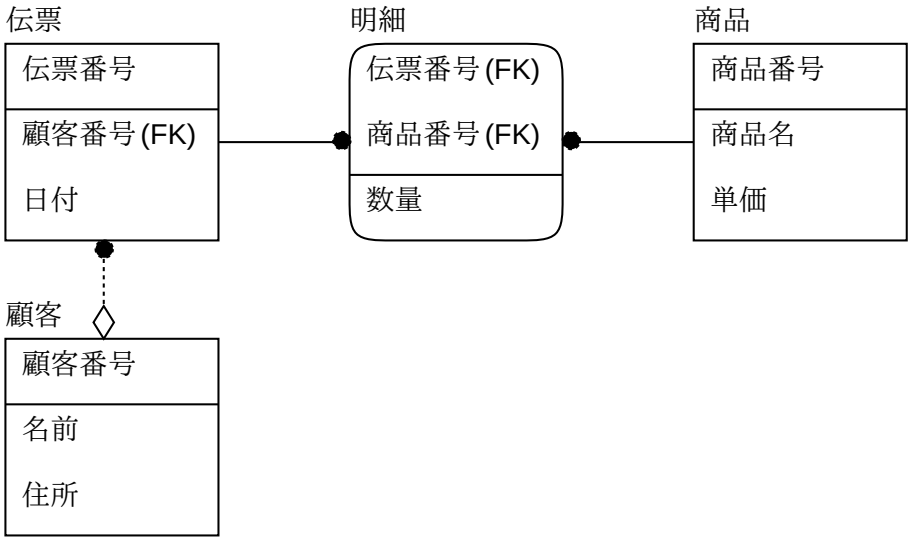

図 8: 部品販売データベースのデータモデル図

A　演習例題: ネット書店のモデル

あなたはこれから、とあるネット書店のシステムを構築することになっている。発注元から次のような簡単な指示が来ているので、これをもとにデータモデル、データベース設計、およびサイトの画面設計を行わなければならない。指示が曖昧で苦しいところだが、ベストを尽くして頂きたい。

- 販売する書籍のデータは、定期的に供給されるので、そのデータに含まれている書籍のみを販売する。データの本 1 冊ぶんの内容は以下の通り。「isbn 10 文字、題名 50 文字、著者 30 文字、出版者 20 文字、発行年月 5 文字、価格 整数、種別『単行本』『文庫』『新書』のいずれかの文字列、順位 整数 (それぞれの種別ごとの人気ランクを表す数字で)」。1 冊ぶんが 1 行の CSV 形式で提供される。同じ本でも価格は改訂されることがあるので注意が必要。(`/u1/kuno/work/book.data`)
- 顧客はサイトに接続したらまず、ID とパスワードを打ち込む。新規顧客の場合は ID とパスワードは自由に選んでもらってよい (ただし既にある ID は除く)。メールアドレスは連絡用に必須。そのほかの入力情報はあなたが決めてよいが、配送やクレジットカード請求に必要な情報もここで入れてもらうこと。再訪問の顧客がこれらの情報を変更することもできなければいけない。
- 新規でも再訪問でも、顧客は本の情報をブラウズできる。ブラウズのさせ方はあなたが決めてよい。検索もさせてもよいしさせなくてもよい。ともかく、画面で顧客が本を選ぶと選んだ本はカートに入る。カートには同じ本が複数入ってもよい。
- 顧客はそのままやめてもいいし、注文確定に進んでもよい。注文確定のところでは、カートにある本の情報を表示させ、また支払合計額も表示させ、それで注文確定してよいかを確認する (送料無料がウリなので金額は本の価格合計のみで決まる)。どの本の注文もこの段階で取り消せる (=カートから取り除ける) ようにしなければならない。注文はカート全体に対して行うので、注文確定するとカートは一旦空になる。
- 注文を確定した時は、クレジットカード会社に請求を出すための「カード請求」データを用意する。そこには「注文番号、日付、カード会社、カード番号、有効期限 (MM/YY)、カード所有者のカード上の名前」が含まれている必要がある (これ以外にあった方がいいものを入れてもいい)。また、それと同時に発送担当が参照する「発送」データも用意する。そこには「注文番号、日付、〒番号、住所、TEL、注文状態」および発送すべきすべての書籍がそれと分かる情報が含まれている必要がある。発送担当は ISBN と冊数だけ分かれば出荷してくれるが、それ以外の情報も必要ならつけてよい。注文状態は、決裁待ち/発送待ち/発送ずみのいずれかが分かるような情報であれば形式等はあなたが決めてよい。
- 画面は顧客が接するすべての画面だけでよい。支払確認業務や発送業務で使う画面は別注の予定。

課題　上記説明に対応したデータモデルを検討し、IDEF1X で記述したもの、および必要と思われるすべての Web 画面のスケッチ (ラフでよい) を作成すること (時間内に済まなければ宿題)。なお、DFD は図 3 を想定してよいが、それとは違う方がよければ変えてもよい (その場合は DFD も描いてくること)。

B　単一表検索の演習

書籍データに慣れるためと、単一表の検索の演習があまりできなかった場合に備えて、単一表検索の演習問題を用意しておきます。

- 関係「書籍」を作成し、書籍データをロードせよ。
- 著者名が「山」で始まる人が書いた本、および著者名の中に「田」の入る人が書いた本の情報を表示。
- 本を出している出版社の一覧を表示。
- 出版社ごとの、および著者ごとの、本の件数を多い順に表示。

- 全著者を通じての平均出版冊数を表示。
- 最も出版冊数の多い出版社が出している本の一覧を表示。

C 補講: 表の結合について

- 表の結合の基本は、実はクロスジョイン (交差結合) にある。
- たとえば、「曜日表」(曜日番号と曜日名の表)、「英曜日表」(曜日番号と英曜日名の表) を考えてみる。
- 次の検索はどういうものを返すか?

```
select * from 曜日表, 英曜日表;
```

- この中から、2 つの曜日番号が同じものを選ぶとどうか?

```
select 曜日表.曜日番号, 曜日名, 英曜日名 from 曜日表, 英曜日表
  where 曜日表.曜日番号 = 英曜日表.曜日番号;
```

- 別にこれが「>」でも何でもいい。しかし「=」を使うのが一番自然で多い。
- こういうの (同じ項目…この場合は曜日番号…で結びつける) のが自然結合。これは JOIN 句を使えば次のように書ける。

```
select 曜日表.曜日番号, 曜日名, 英曜日名 from 曜日表 join 英曜日表
  on 曜日表.曜日番号 = 英曜日表.曜日番号
    where 曜日表.曜日番号 > 3;
```

- この方が WHERE が分けて書けるので and とか言わないで済むという利点がある。しかし自然結合で結合する属性名が一緒ならこう書ける。

```
select 曜日表.曜日番号, 曜日名, 英曜日名 from 曜日表 join 英曜日表
  using(曜日番号)
    where 曜日表.曜日番号 > 3;
```

- この方が結合に使う属性が増えて来た時に楽。さらに、using で結合するものは「どっちの表」と言わなくてよい。

```
select 曜日番号, 曜日名, 英曜日名 from 曜日表 join 英曜日表
  using(曜日番号)
    where 曜日番号 > 3;
```

- しかしさらに、結合に使う属性が「名前が同じもの全部」だったらこれでいい

```
select 曜日番号, 曜日名, 英曜日名 from 曜日表 natural join 英曜日表
  where 曜日番号 > 3;
```

- ところで、同じ表を 2 回結合することもある。その場合は曖昧さを除くために、表の名前のつけ変え機能を使う必要がある。

```
select * from 曜日表 as t1 join 曜日表 as t2
  on t1.曜日番号 = t2.曜日番号 + 1;
```

- ここまではすべてナチュラルジョインだったが、この記法でクロスジョインを書くこともできる。

```
select * from 曜日表 cross join 英曜日表;
```

- 実際には外ジョインの方が多く使われる。たとえば月は 1〜12、週は 0〜6 だから、番号でナチュラルジョインすると共通部分である 1〜6 しか出てこない。

```
select * from 月表 join 曜日表
  on 月表.月番号 = 曜日表.曜日番号;
```

- ここで月表にあるデータは対応する曜日番号が無くても活かしたい場合は、左外結合を使う。

```
select * from 月表 left join 曜日表
  on 月表.月番号 = 曜日表.曜日番号;
```

- ここで逆に曜日表にあるデータは対応する月番号が無くても活かしたい場合は、右外結合を使う。

```
select * from 月表 right join 曜日表
  on 月表.月番号 = 曜日表.曜日番号;
```

- そして両方とも活かしたい場合はフルジョインする。

```
select * from 月表 full join 曜日表
  on 月表.月番号 = 曜日表.曜日番号;
```

- 以上で結合の原理は分かったとして、結合結果から様々なデータを抽出するには、これまで通り WHERE、GROUP BY などを活用すればよい。