情報システムとWeb技術＃3: データベース設計; Webアプリケーション言語

久野　靖*

2015.12.1

はじめに

前回はデータベースの原理や機能について扱いましたが、実際に情報システムを作るにはきちんとデータベースを設計する必要があります。そこで今回は、前回の「飲食店データ」みたいに適当でなく、もう少し系統的に考えるやり方について取り上げます。

また後半では、データベース連携 Web アプリケーションを作ために、Web 上のスクリプト言語 PHP を用いて簡単なデータ操作を行う練習から始めてみましょう。

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?

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。

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。

- 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。

*経営システム科学専攻

- 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。

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)。

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。

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…)

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。

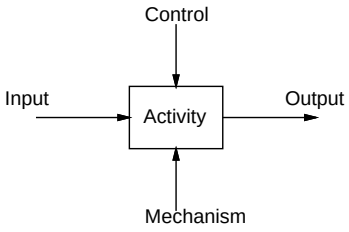

図 1: IDEF0 のアクティビティ

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)。

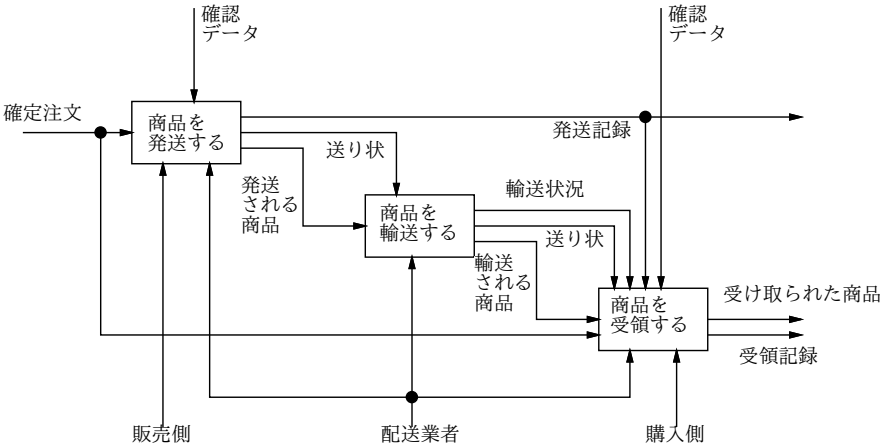

図 2: IDEF0 による記述のサンプル

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

なお、＃1 で取り上げた**データフロー図** (DFD、Data Flow Diagram) も当然、業務モデルの記述に役立てられます。

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)。

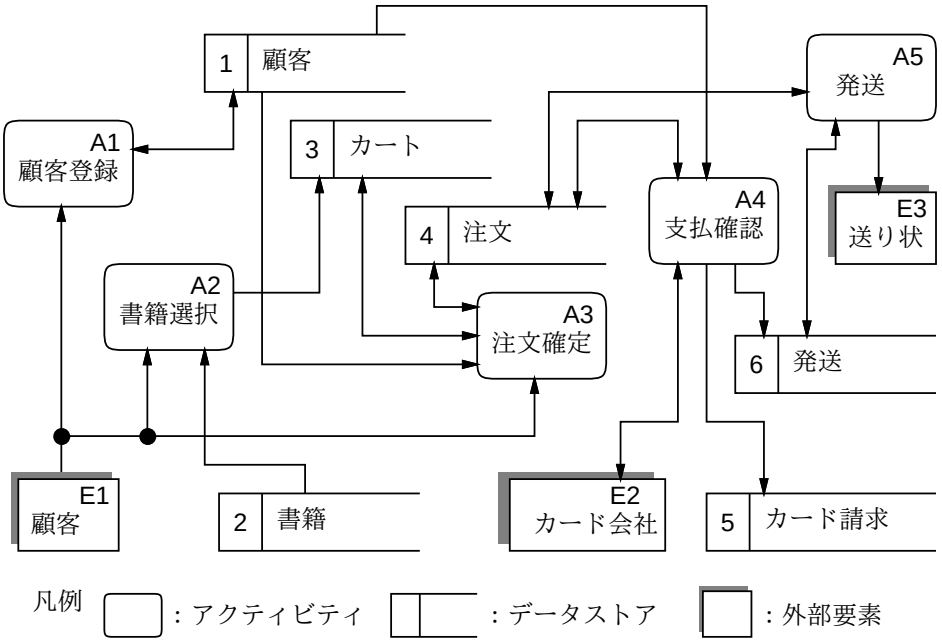

図 3: DFD による記述のサンプル

その概要を以下に説明しておきます。

- 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、クレジットー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。たとえば、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

この図を見て、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。

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。

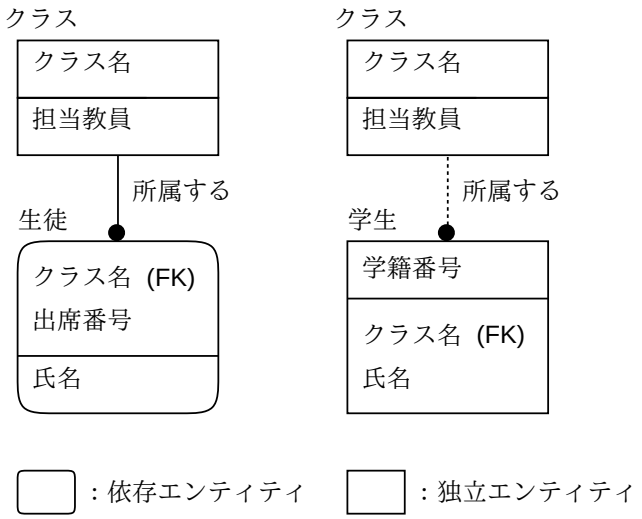

図 5: 依存エンティティと独立エンティティ

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。

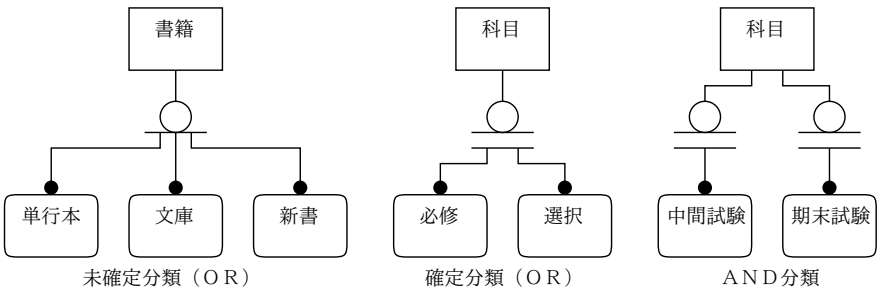

図 6: IDEF1X の分類の記法

- 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)。

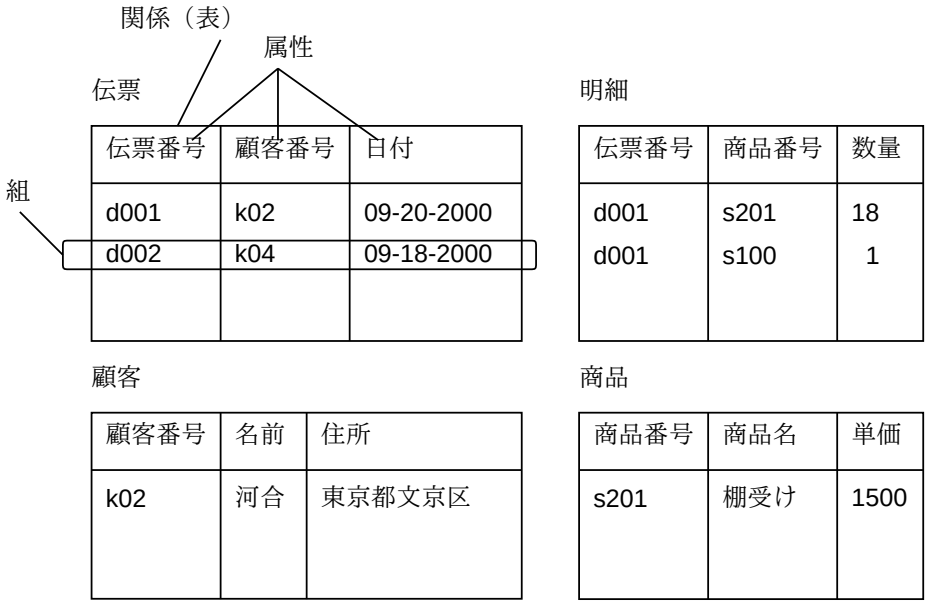

図 7: 部品販売データベース (再掲)

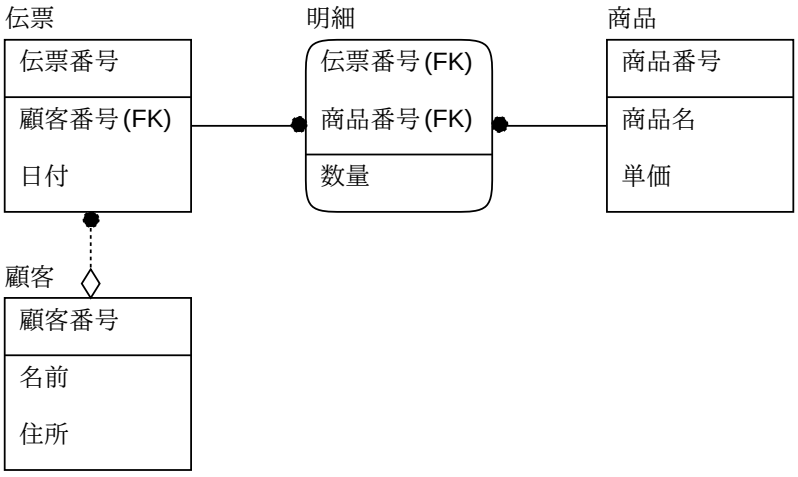

図 8: 部品販売データベースのデータモデル図

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。

2 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。

演習 書籍データに慣れるためと、SQL による検索の演習のため、単一表検索の演習問題を用意しておきます。書籍データは「`\i /u1/kuno/work/book1.psql`」でロードできます。

- 本を出している出版社の一覧を表示。
- 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。
- 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。

演習 上記説明に対応したデータモデルを検討し、IDEF1X で記述したものを作成してください (時間内に済まなければ宿題)。なお、DFD は図 3 を想定して構いません、それとは違う方がよければ変えてもよいです (その場合は DFD も描くこと)。

3 PHP と Web サーバ側プログラミング

3.1 PHP 言語とその由来

前回も取り上げたように、Web アプリケーションは基本的には HTML で記述されたフォームによるユーザインタフェースと、そこから提出されてきたデータを処理する、サーバ側の CGI プログラムとの組み合わせでできています (今日では Ajax などもっと込み入った形態のものもありますが)。

CGI プログラムとは、CGI(Common Gateway Interface) と呼ばれる規約に従って動作するプログラムのことで、この規約が Web サーバとプログラム、Web ページとプログラムの間のやりとり方法を定めています。CGI プログラムは、WWW の初期には C 言語など普通のプログラミング言語で書かれたり、シェルスクリプトを活用して作られたりしてきましたが、その性質上、文字列の処理を多く必要とし、Perl などのスクリプト言語との相性がよいため、現在では Perl をはじめとするスクリプト言語で作られることが多くなっています。

そのような言語のひとつ、**PHP**(Hypertext Processor — どこに最初の P があるんだろうと思うでしょうが、その初期において Personal Home Page tools と呼ばれていたことに由来するらしい) は、Web サーバに埋め込んで実行するスクリプトを書くためのプログラミング言語として、1995 年に Rasmus Lerdorf によって開発されました。その後開発者も実装も変遷して、現在は 2004 年に公開された PHP5 が最新版となっており、以下の説明もこれを対象とします。

PHP の最大のアイデアは、HTML を記述する中に PHP 言語のプログラムが埋め込まれてサーバ上で動くという方式を確立したことで、現在では MS ASP(Active Server Page)、JSP(Java Server Page) など多くのものがそのマネをしています。もう 1 つの特徴は豊富なモジュールによってインターネット上のさまざまな標準をサポートしていることで、そのために PHP のマニュアルは (そして解説本も) ぶ厚いものとなっています。

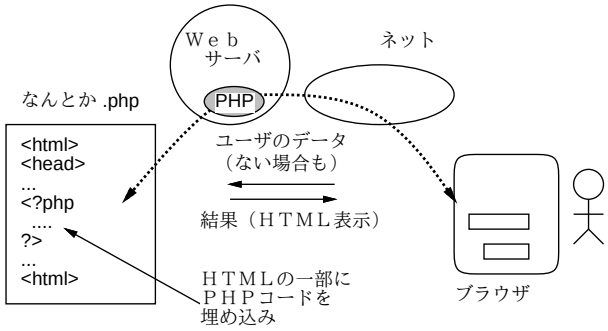

図 9: PHP によるデータの受け取りと処理

さらに、現在標準の Web サーバとして広く使われている Apache ではモジュールとして PHP を組み込むことができ、「`.php`」で終わるファイルに対しては自動的に PHP 処理系が解釈実行するように設定できます。この場合、「`.php`」で終わるファイルは自動的に PHP 処理系が実行し、これによって (PHP 言語で記述された) ユーザデータの処理やページ内容の処理による生成が可能になります (図 9)。我々の内部サーバもこのように設定してあります。

一般に、ユーザ側から PHP プログラムを参照する形は次のどちらかの方法になります。

- 「なんとか`.php`」の URL を自分で開いたりリンクを通じて開く — データは送信しないので、PHP 側では「最初に開かれた」ものとして処理を行う。
- フォーム要素の `action` 属性として「なんとか`.php`」の URL が指定されていてそこに送信する — データが送られるので、PHP 側では「送られたデータを処理し、新たな (結果の) ページを返送する」ものとして処理を行う。

以下では、PHP 言語を使うとどのように簡単に CGI プログラム、つまり Web アプリケーションのサーバ側プログラムが構成できるか、ということを見物して行きたいと思います。

3.2 PHP 言語の基本的な構成

最初に知っておくべきことは、Web で使用する場合の PHP プログラムは HTML ファイルの中に次のような形 (処理命令と呼ばれるものの一種) として埋め込む、ということです。

```
<?php …PHP プログラム… ?>
```

なお、PHP プログラムの部分は何行に渡っても構いません。PHP プログラムの中では C や C++や Java と同じコメント (/* ... */」で囲むコメント、または「//」(「#」でもよい) から行末までのコメントが使えます。

もう 1 つ重要なことですが、PHP プログラムの中で HTTP ヘッダを出力する機能 (header() 関数) を使うときは、一切の通常出力を行う前に行う必要があることです。たとえば、PHP で日本語が正しく表示できるように文字エンコーディングを指定するときにも header() が必要です。なので、PHP プログラムをサーバ上に置く時は次の形のものを使うとよいでしょう (文字化けの問題がないなら、最初のヘッダ生成などは省略してもよいです):

```
<?php
  header("Content-type: text/html; charset=euc-jp");
  /* その他ヘッダ出力はここで */
?>
<!DOCTYPE html>
<html><head><title>title...</title>
<style type="text/css">
/* CSS 記述はここに */
</style><head><body>
<?php
  /* ページ本体出力を含む動作 */
?>
</body></html>
```

ファイル中に日本語を含める場合は、ファイルの文字コードは EUC にしてください。Emacs であれば「Ctrl-X [RET] f euc-jp [RET]」を実行することで編集中のファイルの文字コードを EUC にできます。ステータス表示の最後が「…E」となっていれば EUC ですので、そうなっていない時だけ上記を実行すればよいでしょう。

ではまず最初に、PHP で単純に HTML を生成するという簡単なサンプルを示します (図 10)。

```
<php? header("Content-type: text/html; charset=euc-jp"); ?>
<!DOCTYPE html>
<html><head><title>php sample</title>
<style type="text/css">
div.num { margin: 2px 20px; background: rgb(200,215,255) }
</style></head><body>
<h1>数を並べる</h1>
<?php
  for($i = 0; $i < 10; ++$i) {
    echo "<div class='num'>番号{$i}です。</div>";
  }
?>
</body></html>
```

図 10: PHP による簡単なページ生成

資料では見やすいように下げてありますが、実際のファイルでは最初の「<?php」より前に 1 文字の空白もあってはいけません (あるとページ内容が出力されてしまい、header() が失敗します)。

この例は、単に「番号 i です。」という div 要素が 10 個出力されるだけです。ソースと対比すればプログラムが動いて出力していることに納得が行くと思います。echo 命令は文字列を出力する機能で、文字列中に変数を埋め込む場合は (Perl とはちょっと違って) 「{$変数名}」という形を使います。また、CSS を使って div 要素のマージンと背景色を指定することで、ページの見ためをそれらしくしています。

演習 「10 未満の数を表示する」PHP プログラムを自分のページとして設置し、動作を確認しなさい。動いたら次の課題から 1 つ以上選んでやってみなさい。

a. 10000 未満の数を表示するように修正する。できれば、縦に並ぶと長いので見やすいように工夫するとなおよい。

b. 掛け算の九九の表を表示するようにする。できれば、HTML の表を使って縦横にきれいに並んだものにできるとなおよい。

c. Web ページで使えるさまざまな色の「色見本」のページを表示するようにする。RGB 値は 0〜255 段階なので、たとえば 16 飛びに変化させて全部の組み合わせを見せるなどが考えられる。[1] できれば、ぱっと見て好きな色が見つけやすいように工夫するとなおよい。

3.3 変数とデータ型

PHP では (Perl などと同様)、変数は先頭に「$」をつけた名前で表します。PHP の変数には型がなく、任意のデータを格納できます。データの種類としては次のものがあります。

- 論理値 (boolean) — 真偽値であり TRUE と FALSE という定数も用意されている。
- 整数 (integer) — 整数値。
- 浮動小数点数 (float, double) — 小数点以下の部分を含む値を表現できる形式の数。

[1] ヒント: 任意の HTML 要素についてタグ内に「style="background:rgb(赤, 緑, 青)"」という属性をつければその指定した色に塗ることができます。

- 文字列 (string) — 文字の並び。
- 配列 (array) — 複数の値が並んだもの
- オブジェクト (object) — PHP5 で加わったオブジェクト指向機能によって作成されたオブジェクト
- リソース (resource) — ファイルやデータベース検索結果など外部とのやりとりに使うデータ
- NULL — 「何もない」ことを表す特別な値。NULL という定数もある。

なお、PHP の中核部分では変数や定数の大文字小文字は区別されません。

文字列リテラルは「'...'」でも「"..."」でも囲めますが、ダブルクォートでは変数を書いておくとその値が埋め込まれます。なお、先の例題では埋め込みでもよかったのですが、「.」(文字列連結演算子) の方を使いました (埋め込みを使う場合は、「…{$i}…」のように「{}」で囲んでどこまでが変数の範囲なのか指定する必要がある場合が多いので、例では常にそう書くようにしています。

配列は関数 array() を使って「array(値, 値, ...)」で作り出すことができますし、上記で値を 1 個も指定しないで空の配列を作り、直接添字を指定して「$a[添字] = 値」により値を入れることもできます (もちろん添字を指定して特定の要素を読み出すこともできます)。要素の個数に制限はなく、また添字には文字列などを使うこともできます (連想配列)。

3.4 演算子

PHP の演算子は C などと共通のものが多いです。主要なものを強さの (かっこ無しで書いた時に結び付きの強い) 順に挙げておきます。

- !、=~、++、--、(型名)、@ — 論理否定、ビット反転、増加、減少、キャスト、エラー無視。[2]
- *、/、% — 乗算、除算、剰余
- +、-、. — 加算、減算、文字列連結
- <、<=、>、>= — 大小比較
- ==、!=、===、!== — 等しい/等しくない。後の 2 つは型変換しないで比較する。たとえば「1 == TRUE」だが「1 === TRUE」ではない
- && — 「A かつ B」(論理積)
- || — 「A または B」(論理和)
- =、+=、-=、*=、/=、%= — 代入および代入演算 (足し込む等)

3.5 制御構造

PHP の制御構造はほぼ C と同じなので、C にない foreach と echo 以外は説明は略します。

```
if(条件) 文 [ else 文 ] --- if 文
while(条件) 文 --- while 文
for(式; 式; 式) 文 --- for 文
foreach(配列 as 変数) 文 --- foreach 文その 1
foreach(配列 as 変数 => 変数) 文 --- foreach 文その 2
switch(式) 文 --- switch 文
{ 文… } --- ブロック
break; --- ループから抜け出す
continue; --- ループの次の周回へ
echo 文字列; --- 文字列の出力。式; --- 代入や関数呼び出し
```

[2] @は関数呼び出しなどの前につけることでエラーメッセージを抑制させられる。

foreach ですが、これは配列の全要素を順番に処理するのに使います。その 1 では値を順番に指定した変数に入れながらループします。その 2 は添字と値をそれぞれ 1 番目と 2 番目に指定した変数に入れながらループします。

echo については、普通の関数呼び出しでも済みそうに思えますが、PHP では特別扱いの命令になっています。というのは、PHP では文字列を出力するのが非常に重要な機能だからです (`<?php ... ?>`の外側にある HTML なども実はすべて echo によって出力されていることに注意)。

なお、制御構造より外側のプログラム構造ですが、PHP では先の例題にもあるように、C などと違い、一番外側レベルに書かれた文を順番に実行するのが `main` に相当します。関数やクラスは次の形により定義します。

```
function 関数名 ( 引数名,... ) { 文… }
class クラス名 { クラス本体 }
```

関数定義の箇所を通ると関数が定義されるので、通る前に関数を呼ぶことはできません。if 文などを使って枝別れの中で関数を定義することもできます。クラスとオブジェクト指向機能については話が長くなるのでここでは説明しません。

3.6 フォームを用いた対話的ページ

WWW における**対話的ページ**とは、ユーザからの入力を受け取ってそれに応答するようなもの全般を指します。対話的ページが動くためにはそれを動かすプログラムが必要ですが、それらはプログラムが動く場所により次の 2 通りに分類されます。[3]

- クライアント側プログラミング — ブラウザ上で動くプログラムによる対話的ページ。応答が速い、細かい画面操作やブラウザ操作ができるなどの特徴がある。JavaScirpt、Flash、Java Applet など。
- サーバ側プログラミング — サーバ上で動くプログラムによる対話的ページ。サーバ上のリソース (典型的にはデータベースやファイル) にアクセスできるという利点がある。CGI(言語は何でもよいが Perl が多い)、ASP、Java Servlet、JSP、PHP などが代表的。

以下ではもちろん、PHP を用いたサーバ側プログラミングを前提に説明します。サーバ側プログラミングの場合、ブラウザ画面からサーバにデータを送信するのには HTML の**フォーム機能**を使うので、まずフォームを作るための HTML 要素について説明しましょう。まずは全体を囲む form 要素から。

- `<form method="post" action="#"> ... </form>` — フォーム全体は form 要素で囲まれる必要がある。form タグでは、データの送信方法や送信先 URI(データを処理するサーバ側プログラムのありか) を指定する。ここでは PHP を使うので、フォームを生成する URI とデータを処理する URI は同一 (同じプログラム) になるので「`action="#"`」を指定しておけばよい。送信手法はデータが URI にくっついて見える「`"get"`」と別途送られる「`"post"`」があるが、通常はデータが見えない方がいいので後者を選ぶ。

form 要素の内側には、さまざまな入力部品 (HTML 用語ではコントロールと言います) を入れます。入力部品は通常のテキストや HTML 要素と混ぜて入れられるので、通常の HTML の機能を使って説明文やラベルをつけたり、見やすく配置することができます。

入力部品は以下で説明するようにさまざまな種別がありますが、すべてに共通するのは `name` 属性 (名前) と `value` 属性 (値) です。これらは、サーバ側にフォームが送られる時に対になって送られます。たとえば「`name="age" value="30"`」という属性だったら、「age:30」という対が送信されるわけです。なお、値は部品によってはユーザが入力した文字列になります。

以下に主要な (以下の例題などで使用する) 入力部品とその属性指定を説明します (`name` と `value` は上記の通りなので特に注記することがない場合は説明していません)。

[3] 最近では両者を複合させた **Ajax**(ブラウザ上の JavaScript プログラムがサーバ上のプログラムと通信して必要なデータを取り寄せ画面に反映する技術全般をいう) なども現れています。

- `<button name="名前" value="値">...</button>` — 送信送信ボタン。要素の内側部分がボタンの表示内容になる。ボタンが押されるとフォームのデータがサーバに送信される。送信ボタンは複数あってよいが、送信のために押されたボタンの名前と値だけがサーバに送られる。
- `<input type="text" name="名前" [size="長さ"] [value="値"]>` — テキスト入力欄。送信時に欄に入っている文字列が値として送られる。`size` で欄の幅 (文字数単位) を指定できる。`value` で最初に入っている文字列を指定可能。
- `<input type="password" name="名前" [size="長さ"] [value="値"]>` — 上と同じだが、打ち込んだものが見られないように「*」で表示される。ただし、データそのものは暗号化されるわけではないので、本当に盗まれるとまずい情報は暗号化通信を使ったページから送信すること。
- `<textarea [rows="行数"] [cols="文字数"]> ... </textarea>` — 複数行入力欄。機能は複数行入れられること以外は上と同じ。要素の内側部分が最初の表示内容になる。`rows` と `cols` で大きさを指定できる。
- `<input type="checkbox" name="名前">` — チェックボックス。画面上でチェックを ON/OFF でき、送信時にチェックされているものだけが「`"on"`」という文字列を送信します。
- `<input type="radio" name="名前" value="値" [checked]>` — ラジオボタン。同じ名前のものが複数あってよく、その中でどれか 1 つだけが ON になる。最初に ON であって欲しいものがあれば、それ 1 つだけに `checked` を指定する。送信時には ON になっているものの値が送信される。
- `<input type="hidden" name="名前" value="値">` — 特別な部品で、画面に表示されない (従ってユーザが値を変更することもない)。常に指定された名前と値を送信する。
- `<select name="名前">...</select>` — 選択メニュー。内側には次に示す option 要素を複数入れられ、これらを項目とするメニューができる。
- `<option [value="値"] [selected]> ... </option>` — 選択メニューの 1 項目。この項目が選択されている時は選択メニューの値としてこの項目の `value` が送られる。ただし `value` を省略しているときは要素の内側の内容 (項目として画面に表示される) が送られる。最初に選択された状態であってほしい項目には `selected` を指定する。

色々書きましたが、一番良く使うのは「入力欄」と「送信ボタン」なので、とりあえずこれらだけ覚えましょう。次に、2 つの数値を足すという簡単なページの例を示します。

```
<php? header("Content-type: text/html; charset=euc-jp"); ?>
<!DOCTYPE html>
<html><head><title>php sample</title>
<style type="text/css">
div.main { padding: 5px 20px; background: rgb(200,215,255) }
</style></head><body>
<h1>2 つの数を加える</h1>
<div class=main>
<form method=get>
データ 1: <input type=text size=5 name=data1><br>
データ 2: <input type=text size=5 name=data2>
<button name=calc>計算</button>
</form>
<?php
  if($_GET['data1'] && $_GET['data2']) {
    $v = $_GET['data1']+$_GET['data2'];
    echo "<p>合計は $v です。</p>";
```

```
  }
?>
</body></html>
```

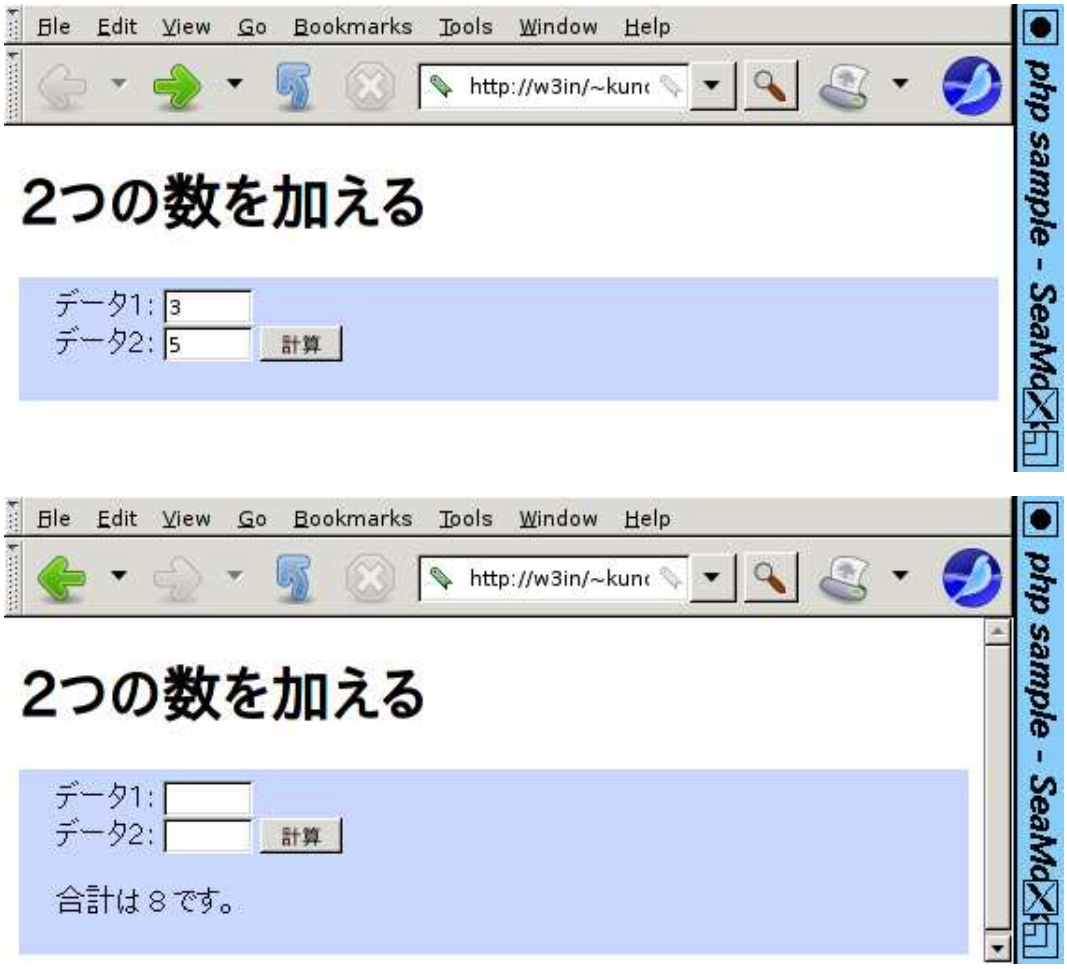

図 11: 2 つの数の和を計算する

このプログラムでは、method=get の form 要素を使っています。この方法だと、フォームの表示とデータ送信後の計算とを 1 つのページで行うので、1 つのページだけで有用な処理がおこなえます。form 要素の中は上述のように、2 つの入力欄 (名前は data1 と data2) と送信ボタンだけです。さて、その後に PHP のコードがありますが、その冒頭で GET メソッドによる送信データ data1、data2 が存在しているかを調べます。ない場合は何もしないのでここで終わりですが、両方ある場合は 2 つのデータの和を計算し、echo 命令で (1 つの p 要素として) 表示しています。

演習 このページをそのまま入力し、動かしてみなさい。動いたら、次のようなページを作成してみなさい。まずページの形を自分でスケッチして設計し、それから作成すること。

a. 2 つの数値を入力し、その積 (掛け算の結果) を表示する。
b. 3 つの数値 (RGB 値) を入力し、その色を表示する。(ヒント: 「`style="bagkground:rgb(○,○,○)"`」というスタイル指定をつけた要素はその指定した色の背景になります。)
c. ローンの借り入れ金額と返済年額を受け取り、完済までの毎年のローン残額を計算する。

演習 ネット書店の Web アプリで必要とする画面群を考え、そのスケッチを作成しなさい (完成しなければ宿題)