情報システムとWeb技術＃5追補: データウェアハウス

久野　靖*

2015.12.15

はじめに

この資料は進行の関係でここまでに取り上げられなかったデータウェアハウスの話題をまとめている。なお、ネタ本はこれ。

> Ralph Kimball, Margy Ross, Tha Data Warehouse Toolkit, 2nd ed., Willey, 2002. ISBN 0-471-20024-7.

もちろん全部は紹介できないので、興味があったら購入して読むことをお勧めする (もちろん洋書です)。ちなみに「初版の」和訳は出ているようだが、初版からは内容がだいぶ改訂されているらしいのでおすすめしない。

あと、＃5の例題を最後まで完成させたものも掲載しているが、これは希望があれば説明します (長くなるので)。

QUIZ: データウェアハウスって何ですか?
QUIZ: 業務データベースの内容を検索したらデータウェアハウスじゃないの?

1　データウェアハウス入門

1.1　データウェアハウスとは?

企業/組織の最も重要な資産は情報。通常、企業ではこの情報を2つの形態で保持している。

- **業務システム** (operational system) — データが投入されるところ。組織の活動の源泉となっている。ユーザは定常業務を遂行。
- **データウェアハウス** (DWH) — データを取り出すところ。組織が活動する様子を観察できる。ユーザは毎回違った情報を求める。

業務システムのデータをそのままコピーしただけで「データウェアハウス」と称しているものもあるが、それでは有用でない (なぜ?)。

QUIZ: データウェアハウスの目的は何?

1.2　データウェアハウスの目的

ユーザが次のような不満を漏らすことはありがち。

- データは山のようにあるのに、アクセスできない。
- 色々な切り口で分析したいのに、できない。

*経営システム科学専攻

- ビジネスユーザが直接自分でデータを取り出すようになってほしい。
- ズバリ重要なことだけ見たいのに。
- 同じデータのはずなのにプレゼンする人ごとに数字が違う。
- ヤマカンでなく事実に基づいて決定を下すようになって欲しい。

このような不満をなくすのがデータウェアハウスの目的。

- 組織が持つ情報を容易にアクセス可能にする。
- 組織が持つ情報を整合性のある形で提供する。
- 適応性があり、変化に追従できる。
- 重要な情報が安全に取り扱える。
- よりよい判断決定の土台となる。
- ビジネスユーザに受け入れられる。

ちょうど出版社のように、ユーザが必要とするものを分かって企画を立て、情報を集めて編集し、ユーザに提供していく。
QUIZ: で、テータウェアハウスとは何?

1.3 データウェアハウスの構成要素

データウェアハウスは複数の構成要素から成っている。

- (業務システム) — DWH の範囲外。処理性能はあるが、レコード単位の処理に特化していて、検索のことは考えていない (検索性能も低い)。古いデータは残っていない。DWH を構築することで業務システムを検索する必要はなくなる。
- **データステージング** — 業務システムからデータを抽出し、必要な加工を施し、データウェアハウスに投入する (ETL — extract, transfer, load)。DBMS は通常使用されない。普通のファイルの形での加工処理。
- **データプレゼンテーション** — 実際にユーザから参照され利用されるデータを保持し提供する部分。データウェアハウスのデータを蓄積している個々のデータベースを**データマート**と呼ぶことがある。複数のデータマート間でスキーマの共通化を行いデータを組み合わせて活用可能にすることが望ましい (**バスアーキテクチャ**)。
- **データアクセスツール** — ビジネスユーザは直接 SQL を使うことはあまりなさそうだから、ツールが用意されていて、そこにさまざまなパラメタを与えて使うことで 8〜9 割のニーズを満たせるようにすることが重要。

DWH ではプレゼンテーション部分がコアになっている。この部分で結局何がしたいかというと、保持しているデータをあらゆる切り口から分析したいということ。具体的には次のような概念。

- ダイス (さいの目) — 「地域ごとの販売実績」「製品ごとの販売実績」などのように軸を入れ換えた視点から見る。
- スライス (薄切り) — 「地域・製品を固定して月毎推移を見る」のように連続値をさまざまな切り口から見る。
- ドリルダウン (掘削) — 月毎→週毎→日毎→時刻毎のようにより細かい単位まで降りていって分析する。

このため、DWH では「第 3 正規形で正規化された」データベース設計 (**正規化モデル**) は使わない。正規化モデルは業務システムの処理性能向上やデータの整合性管理のためには役立つが、DWH の検索のためには向いていない。このような特性を理解していないと起きがちなこと:

- 高価な機器を入れているのに 1 日に数件の検索で手一杯
- 電算室の人が「DHW の検索を書く専属」になっている
- 簡単な検索のはずなのに SQL が何十行も必要
- マーケティング室が自分達に直接検索をさせないといって不満

これらの問題が起きない、有用な DWH の枠組みとして**次元モデル** (dimensional modeling) がある。これについては以下で詳しく取り上げて行く。

DWH ではサマリー (集算) データでなく、生データをそのまま保持することも重要。そうしておいてはじめて、任意の次元でスライス/ダイス可能になる。

各データマートでデータの次元が**適合** (conform) していることも重要。これにより、各データマートを任意に組み合わせて活用可能 (バスアーキテクチャ)。

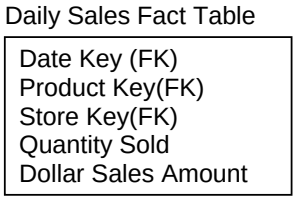
データをオンラインで多次元的に分析することを **OLAP**(On-Line Analytical Processing) と呼ぶ。その実現方法として、多次元データベースを元にした MOLAP(Mutidimensional OLAP) と関係データベースに基づく ROLAP(Relational OLAP) がある。ここでは関係データベースをもとにしているので ROLAP。

1.4 次元モデル

次元モデルとは、収集される情報ごとの**ファクト表**と、ファクト表が結び付ける個々の次元を表す**次元表**の組合せで DWH を設計する方式。

ファクト表

ファクト表は次元モデルの DWH でメインとなる表であり、ビジネスプロセスから発生してくる計測値を格納する。計測値は件数が非常に多くなるので、あちこちに分散せずにコンパクトにまとめて格納したい。ファクト (事実) というのは、ビジネスから実際に発生したデータ、という意味でつけている。ファクト表の例を示す。



ファクト表は次元モデルでは中心となる表であり、1 行につき 1 組の、数値的な計測データが入る。たとえば図 1.2 のような感じ。計測値は全次元の交差点となる (上の例では「日付×製品×店」の組につき 1 つのデータ)。これらの次元の並びがファクト表の**粒度**となる。

最も役に立つファクトは数値で足せるもの。売上額などが典型例。なぜ足せることが重要かというと、DWH では個々のファクトを見ることはまずなく、「○○毎の××」のようにスライス/ダイスするから。そのとき足せるものは足してサマリーにできるから有用。足せないものは件数とか平均とか別の方法を考えるしかない。

ファクト表のデータは「起きたこと」に対応するので、例えば売上がなかったからといって売上 0 のファクトを入れるようなことはしない。

ファクト表はその特性上、複数の外部キーを持ち、そのなかのいくつかを連結したものが主キーとなる (上の例では「日付×製品×店」)。これにより、ファクト表は次元間の多対多の関係を表現。

次元表

次元表はファクト表を補完するもの。ビジネス側の言葉による、テキスト記述を多数含んでいる (図)。

できるだけ多数のテキスト記述を含める (50〜100 とかは普通)。行の数はそう多くならない (百万は希)。

次元表の属性は問い合わせの制約やレポートのラベルとして使われる。たとえば「週毎、ブランド毎の売り上げレポート」が必要なら、週とかブランドとかは属性としてどこかにないといけない。

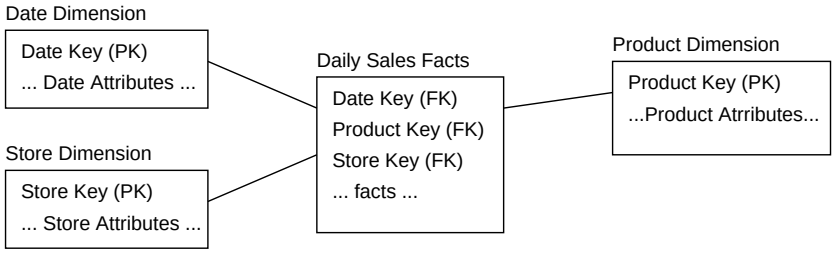
このように多数のテキスト記述があるのは、これによって多様なスライス/ダイス検索が可能であり、またここに載っていることで記述が標準化される。記述は短い (10 文字くらいの) ものと、詳細な (50 文字くらいのものと) 両方あってよい。

実世界で使われるきちんとした語で。謎めいた省略語はダメ。コードも避ける。たとえば「製品コード」みたいな業務コードをユーザが (業務システムで使わされるため) 覚えているとしても、それだけにしてはいけない。どのみち外部に見られるレポートが必要なのだから、きちんとしたテキスト記述が必要。

業務コードは「地域-製品種別-番号」みたいに階層化されているかも知れないが、次元表ではそれらの要素は全部バラバラに参照可能にする。次元に階層性があったとしても、平らな一次元の表にする。正規化して枝分かれした表にしたいと思うかもしれないが、DWH ではそれはよくない。

1.4.1 ファクト表と次元表の組み合わせ

ファクト表と次元表の組み合わせ方は、ファクト表が中心となり複数の次元表と結び付く放射状の構造 (**スタースキーマ**) となる (図)。



非常に単純であり、ビジネスユーザにも分かりやすく、すぐに「自分達の仕事に関係している」と認識してもらえる。表の数も少なくて済むので間違いも起きにくい。

このような単純化は、性能面でも有利。ジョイン (結合) の数が少なく、しかも次元表側のキーを予め決定できるので、ファクト表 (こちらが圧倒的に大きい) を順にスキャンしながら条件に合うものを抽出していける。

拡張性も大きく、次元表やファクト表に属性を追加しても既存の検索アプリが失敗することはない。特定の問い合わせパターンに合わせるということをしていないので、柔軟性が高い。ビジネスユーザが違った形の検索をしたいと言って来た時に対応可能。

2　データウェアハウスの設計プロセス

2.1　4 段階の設計プロセス

般に、DWH の設計プロセスは次の 4 ステップを踏むのがよい。

1. モデル化するビジネスプロセスの選択 — どこからどうやってデータが取れるかとかまで含めて。企業データモデルとかを言っているわけではなく、この業務範囲ならこうやってデータを取れ、こういう有用な分析ができるだろう、という範囲決める。
2. ビジネスプロセスの粒度を決める — ファクト表の 1 つの行がどのようなデータを含むかを決める。たとえば:
   - チケット販売であれば、1 枚ずつのチケットをスキャンしたデータ。
   - 病院であれば、医師の請求書の 1 行ずつ。
   - 航空会社であれば、搭乗券の 1 枚ずつ。
   - 流通であれば、毎日の倉庫の製品ごとの在庫数。
   - 銀行であれば、各口座の毎月決まった日の預金額。

   何を粒度として選ぶかは大きな影響を及ぼすので、簡単そうに見えても省略せずきちんと検討する。
3. ファクト表の 1 行ずつに対応した次元の選択。「この項目を特定するのに、ビジネス側の人はどういう言い方で記述するか?」を考える。次元の細かさ (粒度によって決まる) に加え、その言い方ができるだけのテキストデータがないといけない。
4. ファクト表の各行に入る数値ファクトの同定。「我々は何を計測しているのか?」を考える。このとき、ファクトの粒度がステップ 2 で決めたものに合っている必要。

設計において、ソースデータだけを見て決めてはいけない。ビジネス側からの要求と利用可能なデータの双方を考慮して決める。

2.2　ケーススタディ: 小売業

- 5 つの州に渡り 100 店舗を持つ食料雑貨チェーンの本部。
- 各店は食料、冷食、日用品、食肉、ベーカリー、生花、健康食品、化粧品などの各部門がある。
- 各店の商品数は 60,000 くらい。
- 各商品は SKU(Stock Keeping Unit) 単位で配送/保管。
- SKU のうち 55,000 は外部の製造元から。
- SKU にはバーコードがついている。UPC(Universal Product Code)。
- UPC と SKU は同じ粒度。商品パッケージが違えば SKU も違う。
- 残り 5,000SKU は食肉、ベーカリー、生花などの内製品。SKU と SKU 番号はあるが UPC はない。
- データの収集箇所は複数ある。もっとも有用なのはレジの POS データ。製造元からの受領時にもデータが取れる。
- 各店のマネジメントは注文、在庫、販売を管理し利益を最大化。
- 利益は販売数量を上げてコストを下げるところから。最も影響があるのは価格と広告。
- プロモーション (セール) は一時的な値引き、新聞広告/チラシ、ディスプレイ (平積み)、クーポンなど。
- たとえばペーパータオルを 50 ¢ 値引きして広告/ディスプレイを併用すると 10 倍の販売量になることも。ただしコスト割れ
- したがって全てのプロモーションとその効果分析は重要

2.3 設計の 4 ステップ

2.3.1 Step 1: ビジネスプロセスの選択

最初に次元モデルを構築するときは、最もインパクトのあるもの、最もビジネス的に必要な質問に答えるもの、そしてデータ抽出が可能と分かっているものから選ぶ。

食料雑貨店のケースでは、マネジメントは POS が捉える顧客による購入状況とプロモーションの影響をを分析したいと思うはず。そこで POS による販売データをターゲットとする。

2.3.2 Step 2: 粒度の選択

できるだけ、最も原子レベルの情報を捕捉するべき。原子レベルのデータとは最も詳細なデータであり、それ以上分割できないもの。

データを最も下位の原子レベルで扱うことは次の点から好ましい。

- 原子データは次元が最も大きくなる。多くの次元で区分するほど細かいデータになるので。次元モデルによく適合。
- 原子データは柔軟性が最大。どの方向にでも集計できるから。

もちろん、ビジネスプロセスの必要に応じてもっと大きな粒度で保持することもできるが、そうしたとたんに分析の自由度が制約される。ユーザがこういう方向で分析したい、と思ったときにできなくなる。サマリデータはそれ自体重要だが、原子レベルのデータの代替にはならない。

ケーススタディでは、最も小さい粒度のデータは POS データの各ラインなので、それを粒度として採用する。これにより非常に詳細な情報が捕捉できる。ビジネスユーザはこの小さいデータまで見たいとは言わないだろうが、分析で有用。たとえば次のような分析が可能。

- 月曜と日曜で売り上げはどのように違う?
- あるブランドの商品の各サイズを揃えていることの貢献は?
- シャンプーの 50 ¢ 引きセールの効果は?
- ダイエットソーダのプロモーション時に競合商品の影響は?

これらはいずれも特定トランザクションだけでは答えられないが、適切に組み合わせて分析可能。集計データでは無理なこと。

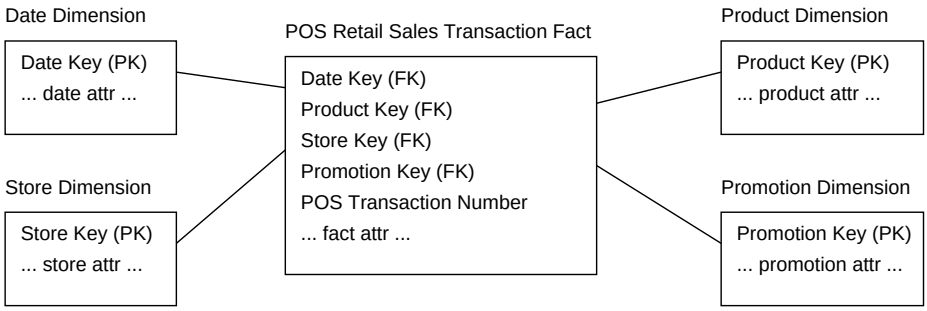
つまり、最も細かい粒度でデータを保持したいのは、その細かさで見たいからではなく、クェリーでどのような方向にでも横断分析できるから。

2.3.3 Step 3: 次元の選択

ファクト表の粒度が決まったら、それに基づいて直ちに以下の次元があることも分かる。

- 日付
- 製品
- 店舗

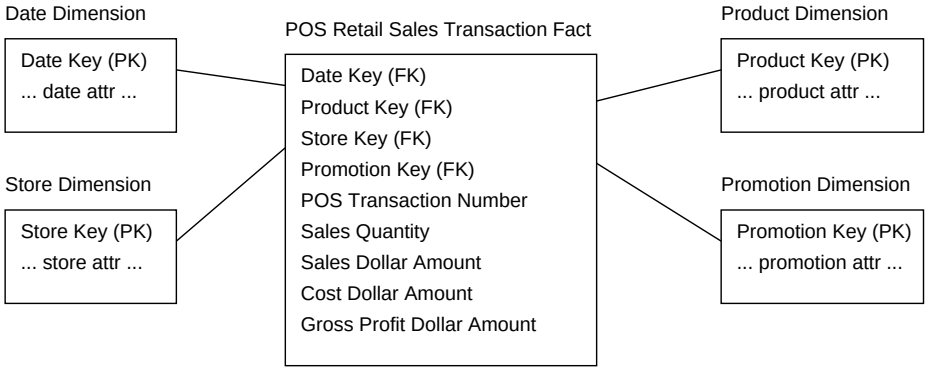
POS システムから得られるデータを考えれば時刻とかも可能性あり (後で考えるかも)。あと、宣伝 (プロモーション) 次元を入れることにする。この次元を入れることで粒度の変更が必要にならないことを確認。そして POS トランザクション番号を入れる (後で検討)。とりあえずの設計を図に示す。



2.3.4 Step 4: ファクトの識別

最後に、ファクト表に現れるファクトがどんなものかを注意深く決める。粒度がちゃんと決まっていればそれに対応したファクトも決まるはず。検討の結果粒度を調整することなどもあるかも。

ここの事例では POS トランザクションの 1 行ずつがファクトになる。POS から来るデータには、販売数量 (缶詰なら 1 個、2 個、…)、売り上げ額。頭のいい POS システムだと店舗販売時点までの製品毎積算コストが提供されるので、ここではそれもあることにする。もしこれがないと、コストをどう割り当てるかは面倒な政治的問題になるが。結果は図で。



数量、粗売上、粗コストはすべての次元について加算可能であり、どんな次元でスライス/ダイスしても正しい数値が求まる。

粗利益 = 粗売上 − 粗コスト

これもすべての次元について加算可能。

計算で求まるものを表に格納するかどうかは議論のあるところだが、格納してしまった方がユーザの間違いがなく、すべてのレポートで同じ名称が使われ整合性が増すので、これでよいと考える。ビューで済むのではという意見については、すべてのユーザがそのビューだけを見るのなら、そうしてもよい。

粗利益率 = 粗利益 / 粗売上

こちらはどの次元についても加算可能ではない。割合ないし比率は加算したらめちゃくちゃになる。スライス/ダイスした対象についての粗利益、粗売上を計算し (これらは加算可能だから)、最後に割る。

単価も加算できないファクト。異る製品の単価を足しても無意味。各店における平均単価の違いを分析するのなら、粗売上と数量をそれぞれ合計して最後に割る。

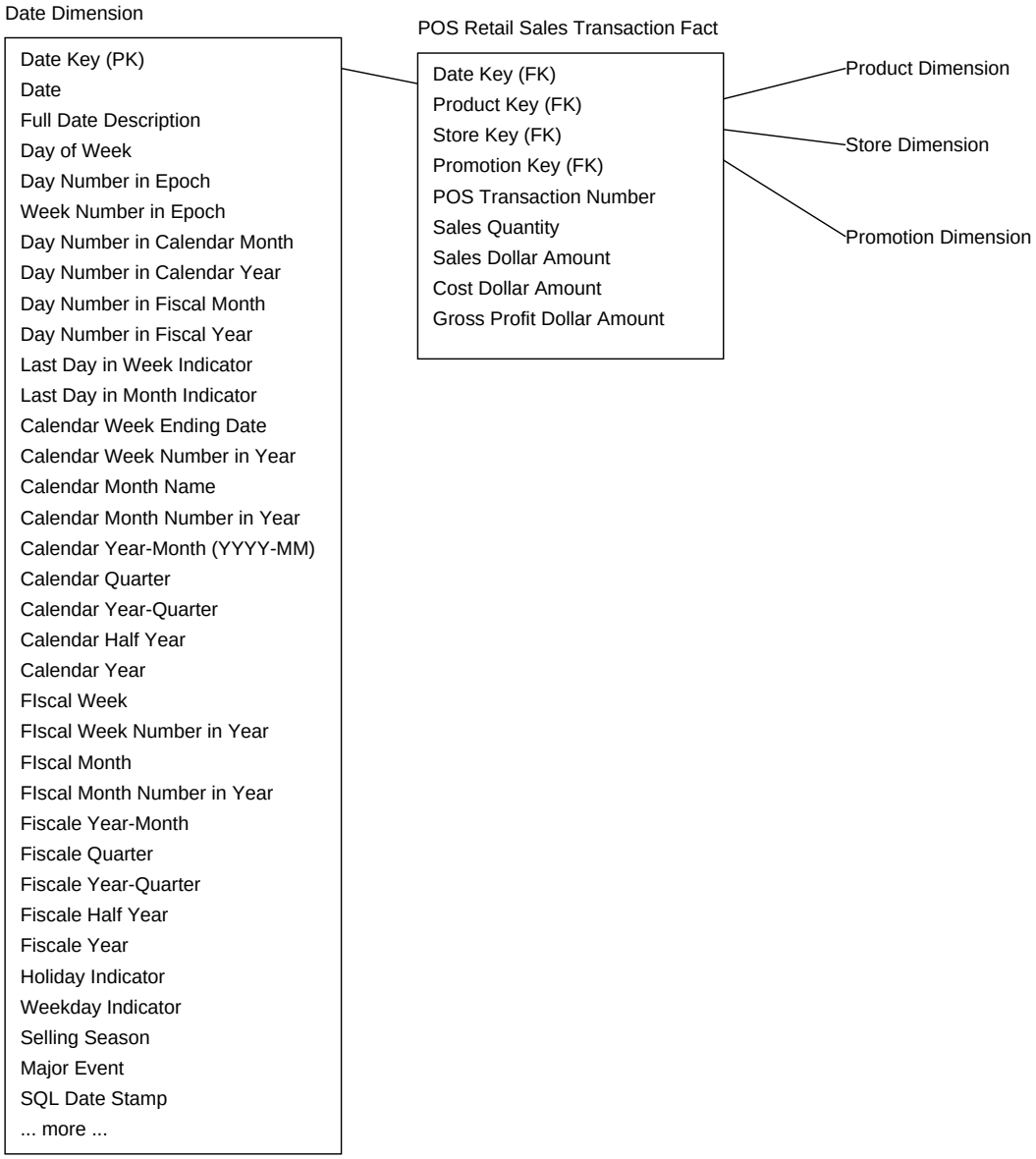
この段階で最も大きな表のデータ量を見積もっておくことは有効。この事例では当然、ファクト表。で、どれくらいかを見積もるには… この企業の粗売上合計が$4,000,000,000/年、商品の平均単価が$2 だとすると、年あたり 1,000,000,000 行。1 行 20 バイトとすれば、20GB。そんなのパソコンのディスクでも入る。このように常にデータ量の感覚は持っておくべき。

2.4 次元表

2.4.1 日付次元

どんな DWH にでも日付次元はあるのが普通。また検索時にも最初のキーになる (データは日毎に DWH に追加されてくるので、物理的にその順番に並んでいる)。

日付次元は予め作っておいてしまうことができる (例外的)。最初に 10 年ぶんの日付情報を用意したとしてもたかが 3650 行ていど。たとえば図のような感じ。



各カラムはその日の特定の特性を表す。Day of Week だったら「Monday」とか「Sunday」とか。Day Number は 1 月なら 1〜31、2 月なら 1〜28 ないし 29、という具合。毎月の同じ日を比較するのに便利。同様にして、月も月名「January」と数字「1」の両方で持つのがよい。Year-Month とかは「2005-11」などの形の文字列になるのでレポートに便利。Quarter は「Q1」「Q2」など。実日付と会計日付が違うこともあるので両方持つ。

Holiday Indicator は「Holiday」「Nonholiday」などを入れる。暗号みたいな「Y」「N」はよくない (レポート生成にプログラムが必要だったり、表現がレポートごとに違ったりするから)。Weekday Indicator も「Weekday」「Weekend」を入れる。

Selling Season は「Christmas」「Thanksgiving」「Easter」「Valentine's Day」または「None」など。Major Event は「Super Bowl Sunday」「Labor Strike」とか。日常のセールはここではなく宣伝次元の方で扱う。

しかしそもそも、日付だったら SQL の日付型を使っておけばいいのでは、と思うかも? しかしそれは次の点でよくない。

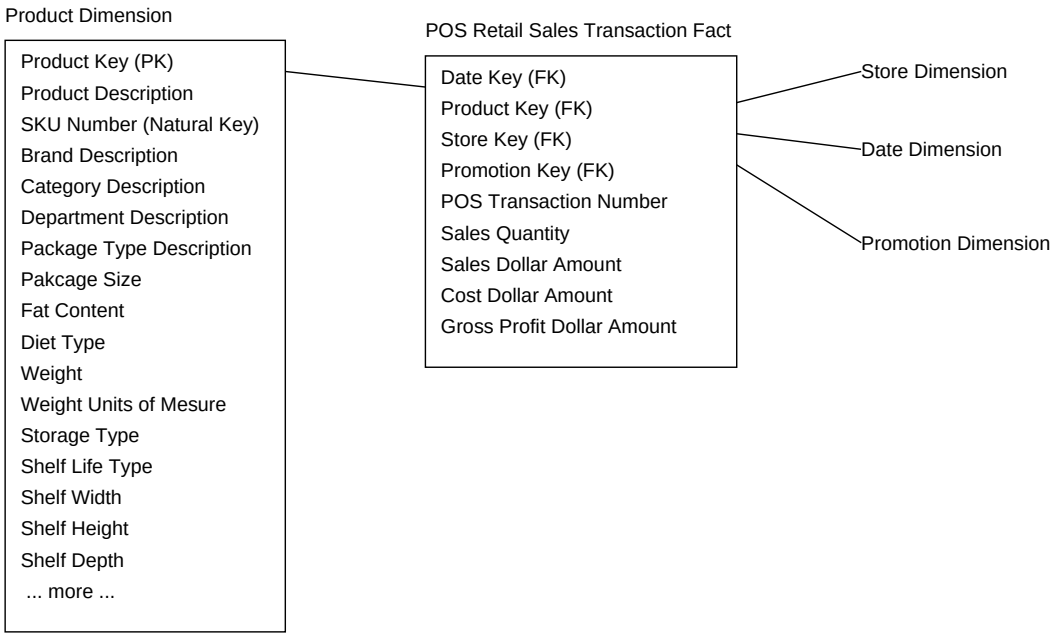
- 多くの DBMS では日付は結合の効率が悪く、索引もつけにくい。
- ビジネスユーザは SQL の日付型に慣れていない。
- SQL の日付型の機能では情報が不十分。
- 日付に関する論理はアプリケーションではなく次元表にあるべき。

2.4.2 製品次元

製品次元は食料雑貨店に並ぶすべての SKU に対する記述を持つ。我々のチェーンでは各店舗に 60,000SKU、店舗ごとの違う扱い品 (内製品を含む) や歴史的変化 (廃番など) を含めると 150,000SKU くらい。実際のデータは業務システムのマスターファイルから来るのが普通。SKU 番号の割り当てなどはそちら側で済んでいる。

マスターファイルには各製品の記述情報も入っているので、それを製品次元に取り込む。通常、「製品→ブランド→カテゴリ→部門」という階層関係がある。このため、上位の階層の値 (部門名など) は、多くの重複を持つことになるが、DWH ではそれを別の表に分離したりする必要はない。次元表のデータサイズはファクト表に比べれば重大ではない。

階層情報のほか、主さとか大きさなどの多くの属性が合わさって製品次元ができる (図)。



典型的な製品次元では 50 以上の記述的な属性があり、これらに基づいて「どういう製品」という絞り込みを含んだ検索ができる。そういう意味では、ドリルダウンとはより多くの属性を指定することに対応。

たとえば単純な次のレポートを考える。

```
Department     Sales Dollar
Description    Amount        Sales Quantity
-----------------------------------------
Bakery         $12,331            5,088
Frozen Foods   $31,776           15,565
```

ドリルダウンしたければ、どんな属性でも追加すればよい。たとえばブランド毎の違いを見たければブランドを追加する。

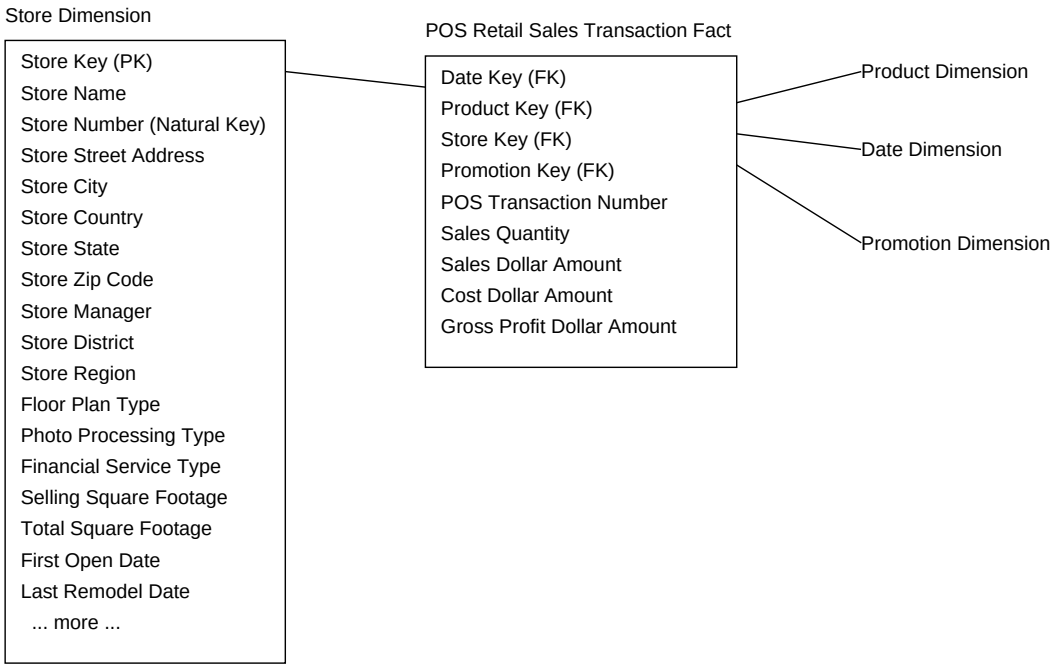
```
Department      Brand         Sales Dollar
Description     Description   Amount         Sales Quantity
-----------------------------------------------------------
Bakery          Baked Well     $3,009           1,138
Bakery          Fluffy         $3,024           1,476
Bakery          Light          $6,298           2,474
Frozen Foods    Coldpack       $5,321           2,640
Frozen Foods    Freshlike     $10,476           5,234
Frozen Foods    Icy            $2,184           1,437
Frozen Foods    QuickFreeze    $6,467           3,162
```

また別の属性を追加してもよい。製品次元は多くのデータマートに存在するものの 1 つ。

2.4.3 店舗次元

店舗次元はチェーンの各店舗に関する情報を保持。製品マスターファイルと違って、店舗マスターファイルはないかも知れない。そのため必要なデータを別途作ることも。

店舗次元は第一には地域次元。ZIP コード、州などでロールアップする。または、地域 (郊外とかダウンタウンとか) でロールアップするかも。それらも含めてまとめて地域次元表に格納 (図)。



Floor Plan Type、Photo Processing Type、 Finance Service Type などはどれも短いテキスト。短いといっても 10〜20 文字くらいはあって、ただし標準化されているべき。床面積は数値だがファクトに入れるのはやりすぎ (売上ごとに面積が変わるわけではない)。

開店日と最後のリニューアル日はいずれも日付次元と同じものだが、同じ表を 2 回参照したくないので、次のようにして「写し」のビューを作る。

```
CREATE VIEW FIRST_OPEN_DATE
  (FIRST_OPEN_DAY_NUMBER, FIRST_OPEN_MANTH, ... )
    AS SELECT DAY_NUMBER, MANTH, ...
      FROM DATE
```

属性名を全部名前つけ替えしているのは、ファクト表と結合して検索するときに混同が起きなくするため。

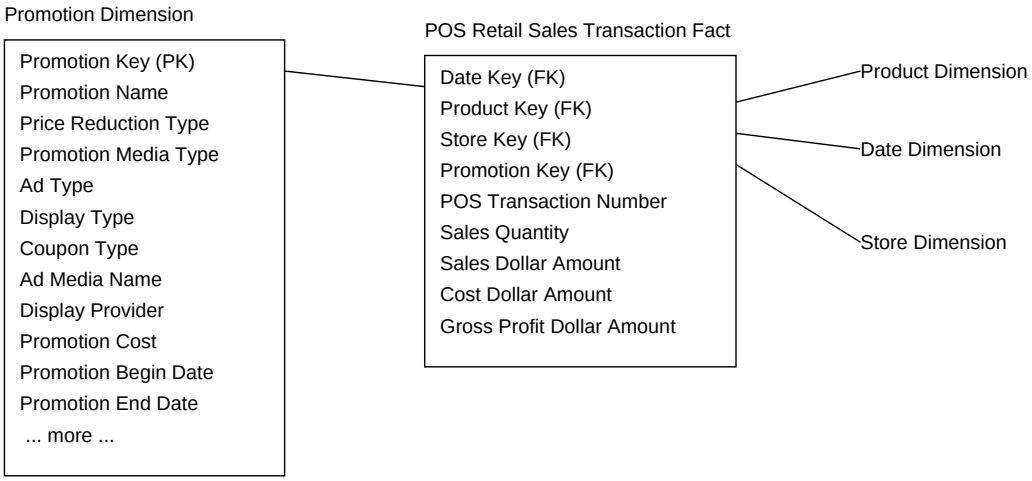
2.4.4 宣伝次元

宣伝次元はこの事例で最も面白いところ。販売時のさまざまな宣伝条件はここに入る。たとえば、一時的な値引き、平積み展示、新聞広告、クーポンなど。このような次元はファクトの計測値に影響を及ぼす原因を集めていることから**原因次元** (causal dimension) と呼ばれることも。

ビジネスの人はある宣伝が効果的かどうかに関心。たとえば:

- 宣伝により、販売数が増加するかどうか。これが判断できるためには、基準販売数が分からないといけない。以前の販売量から計算するか、別のモデルで推測計算するとか。
- 宣伝の前後に、販売数の落ち込みがあるかどうか。その結果、宣伝による増加が相殺されたかどうか。
- 宣伝による販売増加と同時に、別の商品が落ち込んだかどうか。
- 宣伝に伴い、そのカテゴリの商品群が全体として売上増加したかどうか。つまりマーケットの拡大。
- 宣伝は全体として利益となったか。上記の事項全体を加味した利益から宣伝費用を差し引いてプラスかどうか。

原因次元の一部は POS で捕捉できる (値引きは販売金額で分かるし、クーポン使用も POS で記録される) が、捕捉されないものもある。そこで原因次元として集める。実際に行われる宣伝の組み合わせがそれぞれ個別の行となる (図)。



主要な宣伝方式ごとに別の次元を作ることもできる (設計者の選択)。ここでは次の理由から 1 つにまとめた。

- 4 つの方式は互いに関連しているので、組み合わせてもさほど大きくならない。
- 1 つにまとまっている方が見て関連が分かりやすい。

一方、次のような場合は分けた方がいいかも。

- 4 つの方式が互いに独立している場合は分けた方が分かりやすい。
- くっつけた 1 つの次元を管理するより分けた方が管理しやすい。

ほとんどの商品販売は「宣伝なし」なので、宣伝次元に「宣伝なし」の行を 1 つ作り、ファクト表の宣伝なしの販売はすべてこれに結びつける。ファクト表に NULL を入れてはいけない。

2.5 設計プロセスのまとめ

DWH の設計は 4 ステップよる全体枠組みの設計＋次元表の設計がおすすめ。ただし実際には多数細かいことがある。以下では全部は取り上げ切れないので、次節で興味深いトピック単位で取り上げる。

QUIZ: この小売行 DWH から「どのような情報」を引き出したい?

QUIZ: そのような情報を引き出すにはどういう検索をすればいい?

3 個別の興味深い話題

3.1 ファクトレスファクト表

宣伝次元に関連して「宣伝を掛けたのに売れなかった商品は?」という質問があり得る。それに答えるには、これまでに出て来たファクト表では無理 (ファクト表には「売れたもの」の情報しかない)。

そこで、「宣伝範囲表」を追加する。属性は「日付、商品、店舗、宣伝」つまり通常のファクト表と同じだが、粒度が違う。具体的には、宣伝の行われている各日について、宣伝されているすべての商品について 1 行ずつデータを入れる。実データがないので**ファクトレスファクト表** (factless fact table) と呼ばれる。

これを用いて上記の質問に答えるには、宣伝範囲表から当該期間に含まれる商品を取り出し、ファクト表から売れた商品を取り出し、差集合を取ればよい。MOLAP ツールならもっと簡単にこういうことができる。

3.2 縮退次元

ファクト表の POS トランザクション番号の役割は? 業務システム側では、この番号がレシートのヘッダ情報に紐ついていろいろなヘッダ情報を格納しているだろうが、ここで設計した DWH では必要な情報はファクト表に入れてしまったので何もない。このような、一意的なキーになっている (つまり 1 つの次元) だけれど対応する次元表がないものを**縮退次元** (degenerate dimension) と呼ぶ。

トランザクション番号のような、業務側からくる**自然キー** (それ自体で一意になっているもの) は縮退次元となりやすい。たとえば、クレジットカードによる支払情報を DWH に入れるとすると、それはレシートに紐着くので POS トランザクション次元を新たに作ってそこに入れることになるだろう。その場合はトランザクション番号は縮退次元ではなくなる。

3.3 次元表の正規化

普通のデータベース屋が DWH を設計すると、次元表が正規化されていないことが気持ち悪く、正規化して複数の表に分けてしまうことがある。このようにしたものを (枝分かれした雪の結晶になぞらえて)「雪の結晶化」とよぶ。次元を正規化しても複雑で遅くなるだけで、DWH としてはいいことがない。DWH はデータを更新することはなく (変更があれば再ロード)、検索専用であることに注意。

3.4 ファクト表の非正規化/多数の次元

次元表とは対照的に、DWH ではファクト表は正規化された形が普通。なぜか「よく使う属性」を次元表からコピーしてきて非正規化しようとする設計者がいるが、よくない。次元表はデータ量が多くなるので、属性数は減らしたい。

同様な理由から、次元が極めて多いのも困る。次元どうしにいくらかでも依存性があれば、それらをまとめた「複合次元」を作成してまとめることでファクト表に紐着く次元を減らすのがよい。

3.5 代理キー

代理キー (surrogate key) とは、整数を連番で生成して一意なキーを生成したものをいう。DWH のファクト表と次元表の紐付きには必ず代理キーを使うようにするべき。

たとえば製品番号のような業務側でのキー (**自然なキー**、ナチュラルキー) が利用可能であったとしても、代理キーを使用し、自然なキーは次元表側に入れるべき。

QUIZ: なぜそんな面倒なことをするのか?

- 整数の代理キーは使用領域が小さく、結合なども高速に行える。

- DWH と業務システムの緩衝地帯となる。業務キーが突然振り直されたらどうするか? 代理キーなら次元表だけ対応すればよい。
- 古い製品のキーを廃止してしばらくして新しい製品のキーとして再利用するこもある。業務システムでは構わないが DWH では困る。
- 「宣伝なし」とか「日付なし」のような特別な値も問題なく入れられる。(データベースに NULL を入れるのは避けたい。)

代理キーは通常、値順は問題にしないが、日付のようなものは、後の日付ほど大きい値にするのが便利。

縮退次元については、わざわざ代理キーを割り当てないのが普通だが、たとえば店ごとの POS システムが出力するトランザクション番号に重なりがあったりすると、そのままではまずいので代理キーを用いる。また、重複がなくても「英数字まざった 24 バイト」とかだと領域が沢山必要であり操作も遅くなって不便なので代理キーに変換した方がよい。

3.6 3 つのファクト表のタイプ

ファクト表には大きくわけて 3 種類のタイプがある。

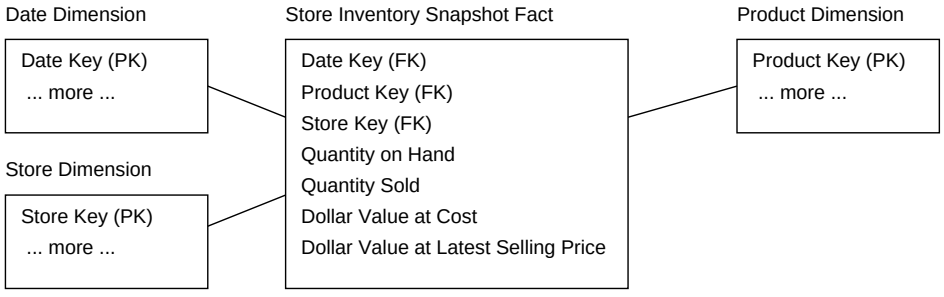
- トランザクション
- 定期的スナップショット
- 累積スナップショット

トランザクションとは、先に出て来た食料雑貨の POS データ (販売トランザクション) のように、取り引きなどイベントがあるごとに 1 件ずつのデータ。これが基本。

しかし、たとえば店舗在庫を管理するとしたらどうするか? たとえば、入庫トランザクション、出庫トランザクションが当然あるはずなので、店が開店してから現在までの全入庫から全出庫を差し引けば在庫が分かるが… 毎回そんなことをしてはいられない。

3.6.1 定期的スナップショット

このため、毎日 1 回ずつ、全製品の在庫量を抽出したファクト表を持ち、これを分析に使用する。このような、一定期間毎の現状値を**定期的スナップショット** (periodic snapshot) という (図)。



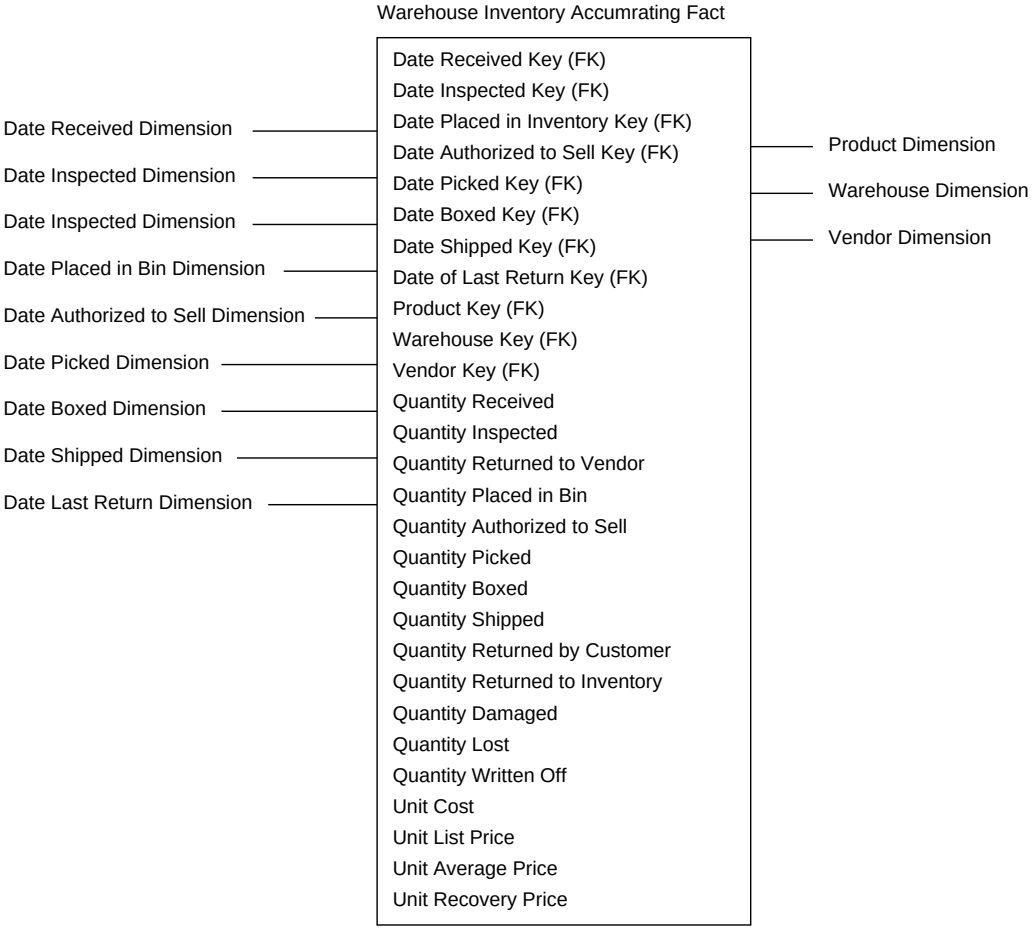
単純には在庫量だけでいいが、それに加えて販売量、製品コスト、製品価値を入れておくことで粗平均 ROI が計算できる。

$$GMROI = \frac{\text{販売量} \times (\text{製品価格} - \text{製品コスト})}{\text{平均在庫量} \times \text{製品価格}}$$

なお、在庫量は店や製品にまたがっては加算可能だが、日付次元にそっては加算可能でない。このような量を**順加算可能ファクト** (semiadditive fact) という。

3.6.2 累積スナップショット

3 番目のファクト種別として、製造業のように、ある製品を受注し、順次製造加工し、最終的に出荷するといったパイプラインになっている場合、その節目ごとの日付とその他の情報を記録していくことが必要な場合が多い。これを**累積スナップショット**という。累積スナップショットでは、1 つのレコードを節目が来るたびに何回も訪問して書き換えて行くという特徴がある (図)。



3.7 次元の変化

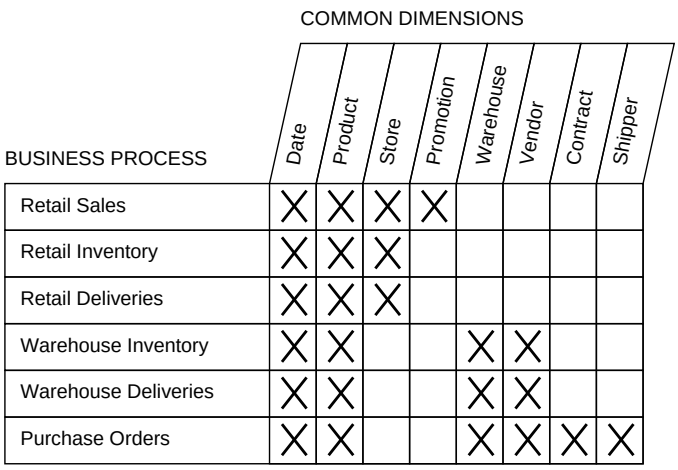
ここまで次元は変化しないものとして扱って来たが、実際には変化するに決まっている。たとえば製品が追加されたり廃番になったり、地域の分類見直しがあったりなど。そのような場合、DWH で変化に対応するのには基本的に 3 つの方法がある。

- Type 1: 上書き — 変化があった場合、そのデータを書き換えてしまう。たとえば地域分類が変わったら地域分類名を変更。古いデータがなくなってしまうという問題。
- Type 2: 行追加 — 変化があった場合、その新しいデータに対応する次元表の行を追加して、以後そちらに紐付ける。古いデータはそのままなので以前の情報を参照。新旧の情報がそのまま保持可能という利点がある。これが基本。
- Type 3: 列追加 — 「地域分類」に加えて「旧地域分類」という属性を増やすなど。一連のデータを旧分類と新分類とそれぞれで分析できるという利点がある。

3.8 DWH バスアーキテクチャ

企業全体の DWH を一気に開発するのはまず無理だが、順次開発するとして最終的にはそれらを組み合わせて分析できるようにしたい。そのためには、次元が共通である必要。どのデータマートはどの次元を

持つかをチェックするためのマトリクスを **DWH バスマトリクス**と呼ぶ (図)。



共通する次元は同一であるか、**互換性を持つ次元** (conformed dimension) である必要。たとえば、製品次元のなかのブランドに係わる部分だけ抽出したブランド次元は製品次元と互換性を持つ。製品次元を持ったファクト表をブランド単位でまとめたデータはブランド次元のファクト表と組み合わせて分析できるというふうに、互換性は重要。

企業全体で互換性を最大限持ったデータマート群を作成していくことでより価値ある分析ができる。ただし複数のビジネスプロセスにまたがったデータマートは単一プロセスに比較して ETL とかも大変。

4 まとめ

今回は「DWH とはどんなものか」「どういうことが問題になるか」が分かる程度のイントロでしたが、実際には「こういうケースではどういう設計の工夫が必要」という話が沢山あって、実務でやるならそういうことまで知る必要がある。とりあえず、ネタ本はいろいろなサンプル事例が出てきて面白いので、興味を持った人は一読をおすすめする。

A　オンライン書店の PHP アプリ

前回やったオンライン書店の PHP アプリでは「カートに入れるところ」までだったので、カートの内容を確認して注文発行するところと、発行した注文から伝票を作るところも加えてみた。またデータを増やし、問題点 (ユーザ情報更新画面でいきなり完了するとまずい) を手直ししてみた。いちおう全部示すことにする。

A.1　sam06a.php — login 画面

ログイン画面については、前回から変更していない。


```
<?php
session_start();
if(!($conn = pg_connect("dbname=kuno"))) {
  $mesg = "データベース接続不能; 管理者に連絡を。";
} else if($_POST['cmd'] == 'newuser' && $_POST['newid'] != '') {
  $newid = pg_escape_string($_POST['newid']);
  $res = pg_query($conn, "begin");
  $res = pg_query($conn, "select id from ユーザ where id = '{$newid}'");
  if(pg_num_rows($res) == 0) {
    $res = pg_query($conn, "insert into ユーザ values('{$newid}',NULL,NULL)");
    if(pg_query($conn, "commit")) $_SESSION['userid'] = $newid;
    header("Location: sam06b.php"); exit;
  } else {
    $res = pg_query($conn, "commit");
    $mesg = "その ID は取得されています; 別の ID を選んでください。";
  }
} else if($_POST['cmd'] == 'login' && $_POST['userid'] != '') {
  $userid = pg_escape_string($_POST['userid']);
  $res = pg_query($conn, "select id,pass from ユーザ where id='{$userid}'");
  $a = pg_fetch_row($res);
  if(pg_num_rows($res) == 0) {
    $mesg = "ユーザ ID かパスワードが違います。";
  } else if($a[0] == $userid && $a[1] == $_POST['pass']) {
    session_destroy(); session_start();
    $_SESSION['userid'] = $userid;
    header("Location: sam06c.php"); exit;
  } else {
    $mesg = "ユーザ ID かパスワードが違います。";
  }
} else {
```

```
  $mesg = "ユーザ ID とパスワードを入れてください。";
}
header("Content-type: text/html; charset=euc-jp");
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>php sample</title>
<style type="text/css">
div.test { margin: 2px 20px; padding: 4px; background: rgb(200,255,225) }
</style></head><body>
<form method="post" action="#"><div class="test">
<p>ユーザ ID とパスワードを入力してログインしてください。</p>
 ユーザ ID:<input type="text" name="userid">
 パスワード:<input type="password" name="pass">
 <button name="cmd" value="login">ログイン</button><br>
<p>はじめての方は使用されるユーザ ID を選んでください。</p>
 新規ユーザ ID:<input type="text" name="newid">
 <button name="cmd" value="newuser">新規 ID 登録</button><br>
<?php if($mesg) echo "<p>{$mesg}</p>"; ?>
</div></form></body></html>
```

A.2 sam06b.php — ユーザ情報更新画面

ユーザ情報更新画面については、郵便番号、都道府県、住所も追加した。これがないと伝票が恰好つかないので。動かす場合はユーザテーブルを次のように一旦削除してから再作成してください。

```
drop table ユーザ;
create table ユーザ (id varchar(20), pass varchar(20),
  name varchar(20), tel varchar(20),
  zip varchar(10), perf varchar(6), addr varchar(40));
grant all on ユーザ to nobody;
```

また、入力欄を変更したまま「更新」を押さずに「完了」を押すと「変更したけど更新していない」という警告が出るようにした。このようなブラウザ上でのチェックと警告は JavaScript を使用する。

```
<?php
session_start();
$userid = $_SESSION['userid'];
if(!($conn = pg_connect("dbname=kuno"))) {
  $mesg = "データベース接続不能; 管理者に連絡を。";
} else {
```

```
  $res = pg_query($conn, "select id, pass, name, tel, zip, perf, addr".
                         " from ユーザ where id = '{$userid}'");
  if(pg_num_rows($res) == 0) { header("Location: sam06a.php"); exit; }
  $a = pg_fetch_row($res);
  $name = $a[2]; $tel = $a[3]; $zip = $a[4]; $perf = $a[5]; $addr = $a[6];
  if($_POST['cmd'] == 'ok') {
    $mesg = "";
    $f = array('ユーザ ID','パスワード','名前','電話','〒','県名','住所');
    foreach($a as $field => $value) {
      if($value == NULL) $mesg = $mesg . "「{$f[$field]}」が空です。<br>";
    }
    if($mesg == "") { header("Location: sam06c.php"); exit; }
  } else if($_POST['cmd'] == 'update') {
    $res = pg_query($conn, "begin");
    $mesg = "";
    if($_POST['pass1'] != '') {
      if($_POST['pass1'] != $_POST['pass2']) {
        $mesg = $mesg."2 つのパスワード入力が一致しません。<br>";
      } else {
        $pass = pg_escape_string($_POST['pass1']);
        $res = pg_query($conn, "update ユーザ set pass='{$pass}' ".
          "where id='{$userid}'");
      }
    }
    if($_POST['name'] != '') {
      $name = pg_escape_string($_POST['name']);
      $res = pg_query($conn, "update ユーザ set name='{$name}' ".
        "where id='{$userid}'");
    }
    if($_POST['tel'] != '') {
      $tel = pg_escape_string($_POST['tel']);
      $res = pg_query($conn, "update ユーザ set tel='{$tel}' ".
        "where id='{$userid}'");
    }
    if($_POST['zip'] != '') {
      $zip = pg_escape_string($_POST['zip']);
      $res = pg_query($conn, "update ユーザ set zip='{$zip}' ".
        "where id='{$userid}'");
    }
    if($_POST['perf'] != '') {
      $perf = pg_escape_string($_POST['perf']);
      $res = pg_query($conn, "update ユーザ set perf='{$perf}' ".
        "where id='{$userid}'");
    }
    if($_POST['addr'] != '') {
      $addr = pg_escape_string($_POST['addr']);
      $res = pg_query($conn, "update ユーザ set addr='{$addr}' ".
        "where id='{$userid}'");
    }
    if(pg_query($conn, "commit")) {
      $mesg = $mesg."更新完了。さらに変更して「更新」; 変更不要なら「完了」";
    } else {
      $mesg = $mesg."更新エラー;「更新」を再試行してください。";
    }
  } else { // 最初に入力ページを表示した時はここに来る
    $mesg = "フィールドに記入して「更新」; 変更不要なら「完了」";
  }
}
header("Content-type: text/html; charset=euc-jp");
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
```

```
<html><head><title>php sample</title>
<style type="text/css">
div.test { margin: 2px 20px; padding: 4px; background: rgb(200,255,225) }
</style><script type="text/javascript">
var changed = false;
function cg() { changed = true; }
function ck() {
  if(!changed) return true;
  if(confirm('変更された欄がありますが「更新」せず完了?')) return true;
  return false;
}
</script></head><body>
<form method="post" action="#"><div class="test">
<?php
echo "<p>ユーザ「{$userid}」のデータ登録/更新</p><table>".
 "<tr><td>パスワード</td>".
 "<td><input type='password' name='pass1' onchange='cg()'>".
 "</td></tr><tr><td>確認用入力</td>".
 "<td><input type='password' name='pass2' onchange='cg()'>".
 "</td></tr><tr><td>名前</td>".
 "<td><input type='text' name='name' value='{$name}' onchange='cg()'>".
 "</td></tr><tr><td>電話</td>".
 "<td><input type='text' name='tel' value='{$tel}' onchange='cg()'>".
 "</td></tr><tr><td>郵便番号</td>".
 "<td><input type='text' name='zip' value='{$zip}' onchange='cg()'>".
 "</td></tr><tr><td>都道府県</td>".
 "<td><input type='text' name='perf' value='{$perf}' onchange='cg()'>".
 "</td></tr><tr><td>住所</td>".
 "<td><input type='text' size=40 name='addr' value='{$addr}' onchange='cg()'>".
 "</td></tr></table><p>{$mesg}</p>";
?>
<button name="cmd" value="update">更新</button>
<button name="cmd" value="ok" onclick="return ck()">完了</button>
</div></form></body></html>
```

A.3 sam06c.php — 書籍検索/選定画面

書籍検索を検索してカートに入れる画面については、前回とほとんど変えていないが、ただし「購入に進む」というのを追加した。


```
<?php
  session_start();
```

```
  if(!isset($_SESSION['userid'])) { header("Location: sam06a.php"); exit; }
  $userid = $_SESSION['userid'];
  $cmd = $_POST['cmd'];
  if($cmd == 'logout') {
    session_destroy(); header("Location: sam06a.php"); exit;
  } else if($cmd == 'update') {
    header("Location: sam06b.php"); exit;
  } else if($cmd == 'purchase') {
    header("Location: sam06d.php"); exit;
  }
  header("Content-type: text/html; charset=euc-jp");
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>php sample</title>
<style type="text/css">
div.test { padding: 4px; background: rgb(255,215,200) }
td { margin: 2px 4px; padding: 4px; background: rgb(200,255,225) }
</style></head><body>
<form method="post" action="#"><div class="test">
<p>検索語:<input type='text' name='word'>
<button name="cmd" value="lookup">検索</button>
<button name="cmd" value="purchase">購入へ進む</button>
<button name="cmd" value="update">ユーザ情報更新</button>
<button name="cmd" value="logout">ログアウト</button>
</div></form>
<?php
  echo "<p>こんにちは、${userid}さん。</p>";
  if($_POST['word'] == '') {
    echo "<p>検索キーワードを入れてください。</p>";
  } else if(!($conn = pg_connect("dbname=kuno"))) {
    echo "<p>データベース接続できません。管理者に連絡を。</p>";
  } else if($cmd == 'lookup' || $cmd == 'order' || $cmd == 'remove') {
    if($cmd == 'order') {
      $isbn = $_POST['isbn'];
      if(is_numeric($_POST['num'])) $num = $_POST['num']; else $num = 1;
      $res = pg_query($conn, "delete from カート"
       ." where id = '{$userid}' and isbn = '{$isbn}'");
      $res = pg_query($conn, "insert into カート"
       ." values('{$userid}', '{$isbn}', {$num})");
    } else if($cmd == 'remove') {
      $isbn = $_POST['isbn'];
      $res = pg_query($conn, "delete from カート"
       ." where id = '{$userid}' and isbn = '{$isbn}'");
    }
    $word = pg_escape_string($_POST['word']);
    $res = pg_query($conn, "select 書籍.isbn, 題名, 著者, 出版社, 価格, 数量"
      ." from 書籍 left join カート on ( カート.id = '{$userid}' and"
      ." カート.isbn = 書籍.isbn ) where 題名 like '%{$word}%'"
      ." order by 書籍.isbn");
    if(pg_num_rows($res) > 0) {
      echo "<table><tbody><tr><th>ISBN</th><th>題名</th><th>著者</th>"
          ."<th>出版社</th><th>価格</th><th>数量</th></tr>\n";
      while($a = pg_fetch_row($res)) {
        echo "<form method='post' action='#'>";
        echo "<input type='hidden' name='isbn' value='{$a[0]}'>";
        echo "<input type='hidden' name='word' value='{$word}'>";
        echo "<tr><td>{$a[0]}</td><td>{$a[1]}</td><td>{$a[2]}</td>";
        echo "<td>{$a[3]}</td><td align='right'>{$a[4]}</td>";
        echo "<td><input type='text' size='3' name='num' value='{$a[5]}'></td>"
            ."<td><button name='cmd' value='order'>注文</button>"
            ."<button name='cmd' value='remove'>削除</button>"
```

```
            ."</td></tr></form>\n";
      }
      echo "</tbody></table>";
    } else {
      echo "<p>検索結果が空でした。</p>";
    }
  }
?>
</body></html>
```

A.4 sam06d.php — 注文確定画面

注文確定画面を新たに追加した。ここではカートにあるものを一覧表示し、その内容で確定していいか問い合わせている。

確定した場合は、カートの内容を (前に設計したように)「注文」「注文明細」テーブルに書き込み、カートをクリアする。また購入せずにカートだけクリアすることもできるようにしている。両テーブルを作成する場合は次のように。

```
create table 注文 (orderid int, id varchar(20), date varchar(20));
grant all on 注文 to nobody;
create table 注文明細 (orderid int, isbn varchar(20), 数量 int);
grant all on 注文明細 to nobody;
```

```
<?php
  session_start();
  if(!isset($_SESSION['userid'])) { header("Location: sam06a.php"); exit; }
  $userid = $_SESSION['userid'];
  $cmd = $_POST['cmd'];
  if($cmd == 'logout') {
    session_destroy(); header("Location: sam06a.php"); exit;
  } else if($cmd == 'update') {
    header("Location: sam06b.php"); exit;
  } else if($cmd == 'lookup') {
    header("Location: sam06c.php"); exit;
  }
  header("Content-type: text/html; charset=euc-jp");
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>php sample</title>
```

```
<style type="text/css">
div.test { padding: 4px; background: rgb(255,215,200) }
td { margin: 2px 4px; padding: 4px; background: rgb(200,255,225) }
</style></head><body>
<?php
  if(!($conn = pg_connect("dbname=kuno"))) {
    echo "<p>データベース接続できません。管理者に連絡を。</p>";
  } else if($_POST['cmd'] == 'confirm') {
    $res = pg_query($conn, "begin");
    $res = pg_query($conn, "select max(orderid) from 注文");
    $a = pg_fetch_row($res);
    $orderid = $a[0] + 1;
    $today = date("Y.m.d");
    $res = pg_query($conn, "select isbn, 数量 from カート".
                           " where id='{$userid}'");
    if(pg_num_rows($res) == 0) {
      $res = pg_query($conn, "abort");
      echo "<p>{$userid}さんのカートは空なので注文できません。</p>";
    } else {
      while($a = pg_fetch_row($res)) {
        $res1 = pg_query($conn, "insert into 注文明細 values(".
         "{$orderid}, '{$a[0]}', {$a[1]})");
      }
      $res = pg_query($conn, "insert into 注文 values(".
        "{$orderid}, '{$userid}', '{$today}')");
      $res = pg_query($conn, "delete from カート where id='{$userid}'");
      $res = pg_query($conn, "commit");
      echo "<p>{$userid}さんの注文を確定しました。注文番号={$orderid}</p>";
    }
  } else if($_POST['cmd'] == 'purge') {
    $res = pg_query($conn, "delete from カート where id='{$userid}'");
    echo "<p>{$userid}さんのカートをクリアしました。</p>";
  } else {
    $total = 0;
    $res = pg_query($conn, "select 書籍.isbn, 題名, 著者, 出版社, 価格, 数量".
      " from 書籍 natural join カート".
      " where id='{$userid}' order by 書籍.isbn");
    if(pg_num_rows($res) > 0) {
      echo "<p>こんにちは、${userid}さん。あなたの選定商品です。</p>";
      echo "<table><tbody><tr><th>ISBN</th><th>題名</th><th>著者</th>"
          ."<th>出版社</th><th>単価</th><th>数量</th><th>価格</th></tr>\n";
      while($a = pg_fetch_row($res)) {
        $p = $a[5]*$a[4]; $total += $p;
        echo "<tr><td>{$a[0]}</td><td>{$a[1]}</td><td>{$a[2]}</td>";
        echo "<td>{$a[3]}</td><td align='right'>{$a[4]}</td>";
        echo "<td align='right'>{$a[5]}</td><td align='right'>{$p}</td></tr>";
      }
      echo "</tbody></table>";
      echo "<p>価格合計は {$total}円です。</p>";
    } else {
      echo "<p>{$userid}さん、あなたのカートは空です。</p>";
    }
  }
?>
<form method="post" action="#"><div class="test">
<p><button name="cmd" value="lookup">検索へ戻る</button>
<button name="cmd" value="confirm">購入を確定</button>
<button name="cmd" value="purge">カートをクリアする</button>
<button name="cmd" value="update">ユーザ情報更新</button>
<button name="cmd" value="logout">ログアウト</button>
</div></form>
```

```
</body></html>
```

A.5 sam06e.php — 伝票の表示

最後に一応それっぽく伝票を画面に表示してみた。

この画面については入力フォームは使わず、URI の末尾に「?orderid=注文番号」を指定すると直接その伝票が表示されるというふうにしている。そのため、PHP 側では変数$GET['orderid'] から注文番号を取るようにしている。

```
<?php
  session_start();
  if(!isset($_SESSION['userid'])) { header("Location: sam06a.php"); exit; }
  $userid = $_SESSION['userid'];
  $cmd = $_POST['cmd'];
  if($cmd == 'logout') {
    session_destroy(); header("Location: sam06a.php"); exit;
  } else if($cmd == 'update') {
    header("Location: sam06b.php"); exit;
  } else if($cmd == 'lookup') {
    header("Location: sam06c.php"); exit;
  }
  header("Content-type: text/html; charset=euc-jp");
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>php sample</title>
<style type="text/css">
div.test { padding: 4px; background: rgb(255,215,200) }
td { margin: 2px 4px; padding: 4px; background: rgb(200,255,225) }
</style></head><body>
<?php
  if(!($conn = pg_connect("dbname=kuno"))) {
    echo "<p>データベース接続できません。管理者に連絡を。</p>";
  } else if($_POST['cmd'] == 'confirm') {
    $res = pg_query($conn, "begin");
    $res = pg_query($conn, "select max(orderid) from 注文");
    $a = pg_fetch_row($res);
    $orderid = $a[0] + 1;
    $today = date("Y.m.d");
    $res = pg_query($conn, "select isbn, 数量 from カート".
                           " where id='{$userid}'");
```

```
    if(pg_num_rows($res) == 0) {
      $res = pg_query($conn, "abort");
      echo "<p>{$userid}さんのカートは空なので注文できません。</p>";
    } else {
      while($a = pg_fetch_row($res)) {
        $res1 = pg_query($conn, "insert into 注文明細 values(".
         "{$orderid}, '{$a[0]}', {$a[1]})");
      }
      $res = pg_query($conn, "insert into 注文 values(".
        "{$orderid}, '{$userid}', '{$today}')");
      $res = pg_query($conn, "delete from カート where id='{$userid}'");
      $res = pg_query($conn, "commit");
      echo "<p>{$userid}さんの注文を確定しました。注文番号={$orderid}</p>";
    }
  } else if($_POST['cmd'] == 'purge') {
    $res = pg_query($conn, "delete from カート where id='{$userid}'");
    echo "<p>{$userid}さんのカートをクリアしました。</p>";
  } else {
    $total = 0;
    $res = pg_query($conn, "select 書籍.isbn, 題名, 著者, 出版社, 価格, 数量".
      " from 書籍 natural join カート".
      " where id='{$userid}' order by 書籍.isbn");
    if(pg_num_rows($res) > 0) {
      echo "<p>こんにちは、${userid}さん。あなたの選定商品です。</p>";
      echo "<table><tbody><tr><th>ISBN</th><th>題名</th><th>著者</th>"
          ."<th>出版社</th><th>単価</th><th>数量</th><th>価格</th></tr>\n";
      while($a = pg_fetch_row($res)) {
        $p = $a[5]*$a[4]; $total += $p;
        echo "<tr><td>{$a[0]}</td><td>{$a[1]}</td><td>{$a[2]}</td>";
        echo "<td>{$a[3]}</td><td align='right'>{$a[4]}</td>";
        echo "<td align='right'>{$a[5]}</td><td align='right'>{$p}</td></tr>";
      }
      echo "</tbody></table>";
      echo "<p>価格合計は {$total}円です。</p>";
    } else {
      echo "<p>{$userid}さん、あなたのカートは空です。</p>";
    }
  }
?>
<form method="post" action="#"><div class="test">
<p><button name="cmd" value="lookup">検索へ戻る</button>
<button name="cmd" value="confirm">購入を確定</button>
<button name="cmd" value="purge">カートをクリアする</button>
<button name="cmd" value="update">ユーザ情報更新</button>
<button name="cmd" value="logout">ログアウト</button>
</div></form>
</body></html>

<?php
  header("Content-type: text/html; charset=euc-jp");
?>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
<html><head><title>php sample</title>
<style type="text/css">
h2 { text-aligh: center; text-decoration: underline }
div.test { padding: 4px; background: rgb(255,215,200); padding: 2ex }
td { margin: 2px 4px; padding: 4px; background: rgb(200,255,225) }
</style></head><body><div class="test">
<?php
  if(!($conn = pg_connect("dbname=kuno"))) {
    echo "<p>データベース接続できません。管理者に連絡を。</p>"; return;
```

```
  }
  $orderid = $_GET['orderid'];
  $res = pg_query($conn, "select id,date,name,tel,zip,perf,addr ".
    "from 注文 join ユーザ using(id) where 注文.orderid=${orderid}");
  if(pg_num_rows($res) == 0) {
    echo "<p>注文番号{$orderid}は存在しません。</p>"; return;
  }
  $a = pg_fetch_row($res);
  $id = $a[0]; $date = $a[1]; $name = $a[2]; $tel = $a[3];
  $zip = $a[4]; $perf = $a[5]; $addr = $a[6];
  echo "<h2>注文伝票</h2>".
    "<div align='right'>id={$id}, no={$orderid}</div>".
    "<p>{$zip} {$perf} {$addr} (TEL: {$tel})</p>".
    "<p>{$name} 様:</p>";
  $res = pg_query($conn, "select isbn, 著者, 題名, 出版社, 価格, 数量 ".
    " from 書籍 join 注文明細 using(isbn) ".
    " where 注文明細. orderid = {$orderid}");
  $total = 0;
  echo "<table>";
  while($a = pg_fetch_row($res)) {
    $isbn = $a[0]; $author = $a[1]; $title = $a[2]; $pub = $a[3];
    $uprice = $a[4]; $quant = $a[5];
    $price = $uprice * $quant; $total += $price;
    echo "<tr><td>{$isbn}</td><td>{$author}</td><td>{$title}</td>".
      "<td>{$pub}</td><td align='right'>{$uprice}</td>".
      "<td align='right'>{$quant}</td><td align='right'>{$price}</td></tr>";
  }
  echo "</table><p>合計金額 = {$total}円</p>";
?>
</div></body></html>
```