The Data Warehouse Toolkit: 15章

久野　靖*

2004.7.27

1 Insurance

□ ここまでに出て来た概念を総活用→損保の DWH のケーススタディ

□ 背景分析→要求分析→バスマトリックス→次元モデル

□ 扱う概念:

- 要求駆動アプローチによる次元デザイン
- バリューチェーン
- バスマトリックス
- トランザクション、定期スナップショット、累積スナップショット
- 4 ステップのデザインプロセス
- 次元のロールプレイング
- SCD(Slowly Changing Dimension) の扱い
- 大きく、頻繁に変更される次元→ミニ次元
- 多値を持つ次元属性
- 縮退次元 (degenerate dimensions)
- データの流れを追跡するための追跡次元
- ビジネスの場面によって大幅に属性や fact が違うようは非均質な製品
- 次元や fact の conformance
- 複数のビジネスプロセスの尺度を連結する連結ファクトテーブル
- factless fact table
- 次元モデル構築におけるアリガチなミス

2 Insurance Case Study

□ $5,000,000,000 規模の損保 (自動車、家屋、個人財物)

□ 支払、field operations、保険計理、ファイナンス、マーケティング各部門の代表者、役員にインタビュー

- その結果、業界の流動的な状況を把握 (←新規参入 ←ネット損保等)
- 業界の統合整理←グローバル化、規制緩和、相互会社からの転換
- 市場 (顧客) も変化

□ ともかく情報が重要な資産、的確な情報に基づく迅速な応答を各部署とも指向

□ 業務システム/プロセスで必要なデータは既に取られている (保険会社では何トンもの業務データが提供されているのが普通)。

- ただしそれらは統合されていない。
- 政治判断部門とデータ処理部門の間には高い壁。商品、顧客、販売チャネルそれぞれごとにまったく違うデータソース。
- 業務システム上では 1 人の顧客が自動車、家屋、物品の各部門でバラバラに 3 回登録されていたりする
- これまではそれでよかったがこれからはデータを共有しないと…→大量の不統一かつ冗長性を含んだデータを扱う必要

□ ユーザはデータに容易にアクセスできなかったという問題→部門ごとにコンサルタントを頼んでデータ抽出を行っていた→バラバラ

- 経営陣は部門ごとに同じデータが違う値だったりするので問題を認識
- バラバラのデータウェアハウス群では長期的にはやって行けない

□ 新しく CIO を雇用し、従来のデータ群を統合した新世界を作るよう指示

- CIO に効果的な情報統合に必要な権限を付与
- CIO は企業内の大量データを迅速に利用可能とするビジョンを策定
- DWH チームを結成してビジョンに応じたシステムを設計/実装開始

□ 経営陣の主張→業務主導から顧客主導のシステムへの転換

*筑波大学大学院経営システム科学専攻

- CIO もこれに呼応
- 社員も「特定用途向けデータ」から「全社でのデータ共有」に
- 従来のタコツボから離れ、自分の業務も含めた全社データにサマリーレベル、詳細レベルでアクセス可能なシステムへの指向

2.1 Insurance Value Chain

□ 保険業のバリューチェーンは短く単純であるように見える

□ 契約→保険料収入→支払処理

- これらの各プロセスをよりよく理解したい
- 契約決定、支払処理などのトランザクションを詳細に分析したい
- 期間ごと、対象物品ごと、地域ごと、人口構成ごと、販売チャネルごとの収支を分析したい
- 企業全体レベルでも、営業対象ごとの個別にも、分析したい

□ 保険会社は当然、他の関連業務にも関わっている。保険料の運用、HRM、金融、購買、等。ここではコアビジネス部分のみを対象。

□ バリューチェーンの始点である契約について→契約トランザクションを 1 つのビジネスプロセス/ファクトテーブルとして扱いたい

- 場合によってはこれでは単純化しすぎで、ファクトテーブルを分割したいかも→3 章にこのあたりの議論

□ 各契約による保険料収入を月単位で計算したい

- 物品販売とちがって、契約=収入ではない。契約→以後毎月の収入の予定
- さらに顧客は保険料を前払いして割引を受けたりする (雑誌販売などでも定期購読+割引の処理が必要になったりする)
- 均等は保険期間に渡って収入を分配していく必要
- 契約トランザクションンからそのつど毎月の収入を計算するのは遅いだけでなく複雑すぎて困難
- →契約トランザクションファクトと保険料スナップショットファクトの両方が必要

2.2 Draft Insurance Bus Matrix

□ 以上のインタビュー内容にもとづき、バスマトリクスを描いた (図 15.1)

□ 次元はコアとなるものだけに絞って描いた (全部描くと 100 以上になる)

3 Policy Transaction

□ マトリクスの 1 行目→契約の作成/変更

- 契約とは、保証範囲の集まりで、顧客に販売されるもの
- 保証範囲が 1 つの「商品」に相当。
- 家屋→火災、水害、盗難、過失、…。自動車→総合保証、事故、無保険補填、過失、…。
- 保険対象 (家、自動車、…) は契約で厳密に規定。1 つの対象が複数の保険でカバーされることも、1 つの保険が複数の対象をカバーすることも、ともにある。

□ 代理店が保険を販売する祭に、保険計理人がその保証対象についての条件に応じた保険料率を決定し、引き受け人が承認した上で契約

□ オペレーショナルシステムは次のトランザクションを収録

- 契約の新規作成、変更、キャンセル (+理由)
- 対象物に対する保証の新規作成、変更、キャンセル (+理由)
- 対象物の価値見積り、見積り減額 (+理由)
- 引き受け契約、引き受け契約の減額 (+理由)

□ DWH 側の粒度も契約トランザクション単位、できるだけコンテキスト情報も収録して次元モデル分析できるように

- 次元の例→契約日付、発行日付、顧客、担当社員、保証範囲、保証対象、契約番号、トランザクション種別。

3.1 Dimension Details and Techniques

□ このスキーマの次元について、本章までに学んだことの発展も含めて検討。

3.2 Dimension Role-Playing

□ 契約トランザクションには2つの日付→契約が運用システムに投入された日付、契約が発効する日付

□ 契約者 (顧客) は複数の人かも→例:本人+配偶者、企業による契約。

- 契約者次元は非常に大きいかも。5 億の契約で、数百万人が関与とか。

3.3 Slowly Changing Dimension

□ 保険会社→次元の変更履歴の追跡は重視

□ TYPE 1 →上書き更新。たとえば「誕生日」(変更=誤り修正)

□ TYPE 2 →新しい行を新しい代理キーで作成して更新。最も普通。ZIP コードの変更など。

- 古い fact は古い ZIP コードを参照する。新しい fact は新しい代理キーを使うから新しい ZIP コードを参照する。
- 柔軟性があり強力だが、弱点は ETL に負担を掛けること
- また行が増えて行くということも問題になる場合がある→これが問題ならミニ次元を利用 (少しあとでこれについても復習)

□ 例: 契約者のセグメント分類を行っているとする。時とともにセグメント分類は詳細化されたりする

- 以前の分類で分析したいことも、最新の分類で分析したいこともある
- TYPE 3 →「旧分類」を現在の分類と別に残すことで両方に対応
- まとまった分類見直しがあった場合などに有効
- 2 より多くのバージョンを追跡したり、複数属性について適用しようとすると複雑になりすぎる

3.4 Minidimensions for Large or Rapidly Changing Dimensions

□ 契約者次元は百万行くらいとかの巨大な次元になる

□ 契約対象物次元も同様 (契約者が 1 個以上の対象物に対して契約)

□ これらの一部の属性については正確な内容追跡が必要←契約時、契約変更時、支払請求時に必要

□ 6 章でやったように、大きな次元で属性変更を追跡するには、変更が起こる部分をミニ次元として分離するのがよい

- ミニ次元は fact table に別の代理キーを通じて結合
- 属性を参照したりクェリーするときに性能の上では負担になる
- 更新の時も面倒だが、あらかじめ可能な組合せをすべて入れてしまっておけるなら簡単

3.5 Multivalued Dimension Attirbutes

□ 多値次元→ 9 章で 1 つの口座に複数の顧客を対応させた。13 章で患者に複数の診断を対応させた。

□ ここでも 1 つの契約に複数の契約者を対応させるのは同様にできる

□ 別の例として、企業顧客にその企業の分類を対応させることを考える

- 企業顧客には 1 つ以上の産業分類コード (SIC) が対応
- 13 章の診断グループと同様、SIC グループブリッジテーブルを作ってグループ情報を収録。顧客次元の outrigger に。
- fact table の数値をどの SIC コードについても重みつき集計可能 (または重みを使わない場合 impact report)
- SIC コードがない顧客→ SIC 次元に「Unknown」を入れておく

□ 保証範囲次元→大きな保険会社では 1 つの種類の保証対象について、数十〜数百の保証範囲が選べる

- 保証上限や免責については、fact table の数値とするのがよい。たとえば住宅火災保険では上限は住宅の現在価格だが、このような連続数値は fact であり次元の属性ではない

□ 保証対象次元は 1 つの対象物につき 1 行→契約者次元より大きいのが普通→ミニ次元化を考える

- テキストによる記述は fact table より次元側に入れたい。複数の決まった記述から 1 つ選ぶようにすれば、集計分析しやすい。完全な自由記述だとあまり役に立たない

□ 社員→そのトランザクションを扱った社員。契約作成、保証範囲作成については、代理店が対応。評価については、評価担当者。引き受けにについては引き受け人。

3.6 Degenerate Dimension

□ 契約番号→契約書のヘッダにある必要なものをすべて他の次元に収納してあれば縮退次元でよい

- 契約に関する各種事項 (契約者、日付、保証範囲、…) は契約 fact table に入れ、契約次元に入れたくはない。
- ただし場合によっては契約書に対応する fact が残る場合も。例: 引き受け人が契約書全体を通じてのリスク評価。→縮退次元でなくなることも。

□ 契約トランザクション種別次元は「種別×あり得る理由」→ごく小さい行数 (100 程度まで) で済むはず

3.7 Audit Dimension

□ transaction fact にキーを入れて監査次元にリンクできる。8 章で述べたように、監査次元にはそのデータがどこからいつどのように取り出されたか、用いたソフトウェアのバージョンなどの情報を格納。

□ 図 15.2 に契約トランザクション次元を示す。典型的なトランザクション粒度ファクトテーブル。

- ほとんどすべてがキーから成っている
- トランザクション単位なので詳細に渡る分析ができる
- 数値データは「契約金額」。その解釈は契約種別に応じて変わってくる。
- さまざまな種類の契約が混ざって含まれるため、もっと特定的なラベルをつけることはできない。
- 逆に種別が増えたとしてもスキーマの変更は必要としない

3.8 Heterogeneous Products

□ 企業全体を通じた視点は欠かせないが、ユーザは自分の受け持ち種別固有の事項も見たい

- 保険の場合、対象によってかなり扱う情報が違う (住宅保険と自動車保険、個人所有物、損害保険、など)
- ここまでに述べたようにそれらを統合して扱えるが、固有の事項もやはり追跡したい
- 9 章でやった「非均質製品」の手法を適用→図 15.2

□ 自動車対象の固有情報を扱う部分を図 15.3 に示す

- 各種の対象物ごとに専用の次元テーブル (保証対象、保証範囲ともに) を用意
- アプリケーション側で種別に会った次元テーブルを結合する必要

□ この方法では fact table は専用化しないでよいことに注意。次元だけ新しく追加している。論理的には次元の属性を増やしているものと考えればよい

3.9 Alternative (or Complementary) Policy Accumulating Snapshot

□ トランザクションの累計効果を把握するための累積スナップショットの用法についてあとひとこと。

□ このケースでは fact table の粒度→ 1 つの保証範囲/保証対象が 1 行

- 関連する日付: 見積り日、評価日、引き受け日、発効日、更新日、満了日。
- 他の次元についても同様のことがいえそう (トランザクション種別次元は除いて)

□ →累積スナップショットは拡張された fact set に対応することになるだろう

- 累積スナップショットは契約トランザクションに関する主要なマイルストンの情報を収集するのに有効
- 契約、保証対象、保証品目などのライフスパンにわたる追跡
- ただしすべてのトランザクションの情報を収録はしない←例外的なイベントや通常のプロセスから外れるものは入れない
- その代わり、プロセス間の遅延時間などがはっきり分かる

4 Policy Periodic Snapshot

□ 契約トランザクションスキーマ→広い範囲の問い合わせに回答可能

- しかし、大量のトランザクションが流れているため「ある時点での資産価値」のようなものは把握しづらい
- 情報自体はトランザクションにあるとしても、それを最初から累計して来ないと現時点の値にならない
- KPM をこのようにして毎回計算してくるのは論外

□ トランザクションテーブルと対で累計テーブルも作る。この場合月毎の保険金スナップショット。粒度は各月ごと、契約範囲、契約対象ごとに 1 行。

4.1 Conformed Dimensions

□ もちろん、新しいテーブルでも次元はできるだけ再利用(共通化)する

- 新しいテーブルの次元はトランザクションテーブルと同一か、または部分集合に
- 契約者、契約対象、契約範囲などの次元は同一
- 日付の代わりに月次元に(conformance はある)
- 担当社員については月累計では追跡しなくてよい
- 代理店については入金に関わるので残したい
- トランザクション種別次元は不要(月単位の粒度で関係ない)
- 代わりに status 次元を作って「新規」「キャンセル」等を収録

4.2 Conformed Facts

□ fact についても conform する必要

- 1 つの fact が複数の fact table に現れるなら、整合性のある定義やラベルを持つ必要(後で例)
- 逆に同じでないものは違う名前にすることも重要

4.3 Pay-in-Advance Metrics

□ マネジメントは「いくら分収入があった」だけでなく「いくら分契約がとれたか」も関心

- この場合、トランザクションを累計すれば分かるわけではない
- 顧客が契約を結んで契約金を払ったとしても、収入はサービス提供とひきかえ
- 保険の場合は収入は(キャンセルされない限り)毎月保険料として入る
- 正確な金額はオペレーションシステムと同じ計算規則で計算しなければ分からないがひどく複雑→オペレーションシステムに計算してもらえば済むが

□ 図 15.4 のように、保険料収入については「契約額」と「入金額」の 2 つを含む

- たとえばある年契約を 1/1 に$600 で取ったとすると、1 月は契約額は$600 だが入金額は$50(つまり 1 月ぶん)。2 月は契約額$0、入金額は$50。3 月に契約キャンセルしたら、入金額は$50、契約額は-$450。

□ 一括入金が入って来るとトランザクション粒度と毎月のスナップショットの組合せが必要。どちらか片方だけで済ませられるようには普通できない。

4.4 Heterogeneous Products Again

□ スナップショットに特定業務むけデータを含めることも必要

- その特定業務のものでない行についてはその属性は null → null だらけに
- その解決策としては、スナップショットを業務ごとに分ける
- →コア(共通)スナップショット+各業務のカスタムスナップショット群
- カスタムスナップショットにはコアのデータもコピーで入れるか、または 9 章のようにキーを使ってコアテーブルに結びつける

4.5 Multivalued Dimensions Again

□ 自動車だと 1 契約に複数の契約運転者があるかも→多値次元

- 図 15.5 のようにブリッジテーブルで結びつけて重みつけ計算
- 変更に対応するためブリッジテーブルには開始日/終了日を入れる

5 More Insurance Case Study Background

□ 保険業には暗〜い側面もある→支払請求と損失

□ 支払請求者(契約者でも新規の第 3 者でもあり得る)が支払請求→当然、契約内容に基づいて

- 請求があった場合、調査を(請求者、契約者、その他に対して)行う→トランザクション群の発生。複雑なものでは複数の外部専門家に判断を求めたりもする
- だいたいは調査後、支払を行う→支払先は第 3 者のことが多い。医者、弁護士、修理工場。請求者に直接支払うこともある
- 大きな保険会社では 1,000 人以上が請求に対する支払いを行える→どの社員がその請求を担当したかもそれぞれ記録

□ 保証対象物は支払い後に保険会社の所有物となる場合がある

- たいていはジャンクの価値→入金したらその請求の会計に入る

□ 支払い完了後、請求は完了→トランザクション群は完結

- もめた場合はさらに請求があったり裁判になって請求がまた変わる→収支もまた計算し直しに→どれくらい、どんな場合にこれが起こるかなども重要なデータ

□ 係争中の裁判や反訴が決着したらいくら戻るかなども計算

- これらの権利をまとめて専門家に売却して換金する場合も (subrogation) →そういうトランザクションとして記録

□ 個別の記録に加えて請求の全体像も知りたい。請求があってから支払いまでに何日か→請求処理の効率が分かる

5.1 Updated Insurance Bus Matrix

□ 請求についても分かったので図 15.1 のマトリクスを更新→図 15.6

- 多くの次元は再利用。請求、請求者、第 3 者を追加。

□ マトリクスの詳細度で苦しむかも→計画段階では抽象レベルの高いところに留まるのがよい

- 1 つのビジネスプロセスの行から複数の fact table が作られるかも

□ 実装設計段階まで来たらマトリクスの一部を取ってもっと詳しく堀り下げる→図 15.7

6 Claim Transactions

□ 請求処理から次のようなトランザクションが発生

- クレーム開始、再開、終了
- 留保設定、留保再設定、留保終了
- ジャンク見積り、ジャンク歳入
- 調査員による調査、インタビュー
- 裁判開始、裁判終了
- 支払い実行、支払い受領
- 請求の下取り

□ 図 15.6 のマトリクスを更新してみると、契約について設計していたときの次元が多くそのまま利用。日付 (2role)、社員、トランザクション種別。

□ あたらしい次元ももちろんある (図 15.8)。

- 請求次元→請求内容をコード化。契約範囲、契約品目に対応つけられる必要。
- 請求者は通常は個人。第 3 者は目撃者、専門家、支払い受領者など。これらは「汚れた次元」。きれいに識別し追跡するのは難しい。悪意ある受領者は他の請求に関係していることを同定されないように行動する。

□ 請求データについても非均質な製品と同様の手法が有効

7 Claims Accumulating Snapshot

□ トランザクションをきっちり押えていても回答するのが難しい問い合わせがいろいろあった

- 請求の現時点での収支などもこの類
- 毎日、その日までの収支を累計スナップショットとして計算
- 粒度は契約×保証範囲×保証対象×請求の異る組合せごとに 1 行
- 請求が開始されると行ができて決着するまで順次更新されていく

□ 図 15.9 のように多くの次元は互換性があり再利用。ここではマイルストン間の進捗が分かるように細かく記録。

- status もすぐ分かるように記録。
- トランザクションレベルの次元である社員、請求者、第 3 者、トランザクション種別などは除外。

□ 長期間 (何年も?) 掛かる請求の場合は累計スナップショットでなく月毎定期スナップショットのようにするかも。

- 粒度は現在開いている請求 1 つにき 1 行
- fact は数値で加算可能なものでその月に属するもの。請求金額、支払い金額、留保の変更額。

□ 場合によっては 3 種類 (トランザクション、累積スナップショット、定期スナップショット) すべて作ることになるかも。

8 Policy/Claims Consolidated Snapshot

□ ここまでで契約と請求について（スナップショットまで含めて）はっきりしたが、利用者は収支が知りたい

- それぞれの集計をそのつど差引することもできるが…
- 保険料収入と請求による支出を合わせたfact tableを作れる（図15.10）。月単位スナップショットなので次元も少しすくなく→ consolidated fact table（7章）
- 基本的なモデルがそれぞれ出来たあとで作るのがよい

9 Factless Accident Events

□ 自動車事故の場合、関係する車両、人すべての関連をfactless fact tableとして表せる→図15.11

□ 被害者→事故に遭遇した他の人（同乗者、歩行者など）

- 車両→自動車と関わりのない人であれば「No Vehicle」
- 被害関係→関わり方を表す
- 事故カウント→「いつも1」集計に便利

□ factless fact tableを使うことで複雑な事故でも収録できる

- 請求者や被害者が複数の場合、多値次元で増やすことも→1事故1レコードに保てるという利点

10 Common Dimensional Modering Mistakes to Avoid

□ ここまでで次元モデリングの技法に関する章はおしまい→デザイナはどこから先は行くべきで*ない*かについて。

- ここまでは「何をやるのが*いい*」というpositiveな話だった
- なのでここでは「*やらない方がいい*」話
- 例によって重要でない順（でも最初のでも失敗したらダメージあるよ）

□ 間違い10: 制約/グループ化のためのテキスト属性をfact tableに入れる

- 数値はfact tableに
- 記述テキストはdimension tableに
- そこから先は個々に。測定ぽいものはfact、記述ぽいものはdimension
- コメントをfact tableに残さないように

□ 誤り9: 領域節約のため次元テーブルに入れるテキスト量を制限する

- 次元テーブルの量はたかが知れている
- 使いやすいDWHのためにちゃんと読めるテキストを使おう
- 次元テーブル中のテキストはユーザインタフェースのため重要

□ 誤り8: 階層/階層レベルを複数の次元として分割する

- 階層→多対1関係の連続したもの
- 最下位レベルを指定すれば全部決まる
- ユーザは階層構造については分かっている
- バラバラにするのではなく素直に見せることが大切（DWHはプレゼンテーションのためのもの!）
- 複数の階層が独立にあってもいっこうに構わない

□ 誤り7: 次元属性の変更を追跡することを考えない

- ユーザは次元属性の変更による影響を知りたいと思う（意外かも知れないが）
- 最新版だけ見せておけば満足だろうというのは大間違い
- SCDの3つのタイプを学んだ。タイプ1のみに頼らないこと
- 一群の属性が激しく変化→ためらわずにミニ次元化すること
- どれくらい変化するかは最初からは分からない

□ 誤り6: 性能問題には常にハードウェア増強で対処する

- 集合化、サマリーテーブルの2つが性能改善には極めて有効
- 多くのクェリーツールは集合化の機能を組み込んでいる
- 索引つけ、DBMSソフトの増強、メモリ増強、CPU増強、並列度の増強などバランスをとって進めるべき

□ 誤り5: 実務システムのキーやスマートキーをfact tableに入れてjoinに使う

- 初心者は全部次元テーブルに入れて代理キーで連結とかいう考えに抵抗がある

- そうすると実務システムのキーや日付文字列をキーに使ってしまう
- そうしうると必ず後でいろいろ醜い問題に遭遇する。代理キーを使おう

□ 誤り 4: `fact table` の粒度を最初に定義せず、後で選択を守らない

- 次元モデル→ビジネスプロセスが生成する基本データから始めるべき
- 粒度をまず決める。最も原始的な細かいレベルを選ぶことで抵抗を排除
- 測定値をその粒度にあった次元で囲む
- 粒度に忠実であることが次元モデルの心臓
- 小計とかのように粒度に合わないものを `fact table` に入れるとまずい (一見よさそうでも集計すると多重カウントになるしいいことがない)

□ 誤り 3: 特定のレポートに基づいて次元モデルを設計する

- 次元モデルはそのレポートとは独立なはず。
- あくまでも計測プロセスに基づいたものが次元モデルであるはず。
- 各レポート向けに数百ものテーブルを作ってしまった設計チームの悲劇

□ 誤り 2: ユーザが最下位レベルの原始データを正規形で検索すると思うこと

- 最下位レベルのデータは基本だが、ユーザはそれをそのまま見るわけではない。

□ 誤り 1: 複数の `fact table` 間で `fact` や次元の互換性を保たないこと

- ここまで読んで来て孤立したテーブルを作るようではしょうもない
- 複数のソースから「収入」が取れて来るならそれは別の名前にして次元的に適合させること→ `fact` の互換性
- 次元が適合しいればこそ複数の `fact table` を結合して分析ができる。これがとにかく基本。

11 Summary

□ これで次元モデリングの用語もツールも分かりましたよね?