The Data Warehouse Toolkit: 8章

久野　靖*

2004.6.8

1 Human Resource Management(HRM)

□ HR情報→7章の会計/財務情報同様、組織内に遍在

- 一方、HR情報の技術に強いアナリストがいるわけではあまりない
- 今日の競合的環境→社員の特性を把握し活かすことは重要

2 Time-Stamped Transaction Tracking in a Dimension

□ ここまでに例示した次元モデルは互いに類似

- fact table に KPM(key performance metrics)が含まれ縦横に足せる
- 次元モデラ→加法性に安住しがち。普通はそれでいいがHRデータは足せない。数値ですらない。でも絶えず変化している

□ 100,000社員の大企業の人事部にいるものとしよう

- 各社員には100以上の項目を持つプロフィールが付属。雇用日、職位、給与、考課日、考課内容、休暇状況、部署、教育、住所、保険プラン、その他。
- このデータに関する変更トランザクションの流れ→雇用、移動、昇進、…

□ これらの事象を正確に追跡し分析したい。トランザクションヒストリにはすべての情報が含まれている。

- 複雑な(予想もできない)問い合わせもあるかも知れないが、答えは必ずここから得られる。

□ 図8.2のスキーマ。

- 社員トランザクションをトランザクション単位の粒度で記録するfact table。1行1トランザクション。factless(数値データはなし)
- 付随するデータ→社員プロフィールへの変更内容。

□ 次元→トランザクションの日付/時刻、種別、社員。

- 種別次元→変更の種別によってさまざまなデータを保持
- 社員次元→多数の属性を持つ幅の広いテーブル。社員IDはここに保持

□ 社員情報の変化は社員次元をTYPE2SCDとして扱うとしよう

- トランザクションごとに社員次元に新しいTYPE2の行を追加

□ …とすると、社員次元とファクトテーブルの行数が同じに?そりゃヘン

□ そこで社員次元を強化してファクトテーブルはなくす→図8.2

- トランザクション種別等もここに含まれる
- 数値データはない(あったとしても単に前のものを置き換え)
- c.f.　サマリーが取れるような(足せるような)ものであればファクトテーブルにする必要

□ 社員トランザクションキー(代理キー)が主キー

- 社員ID+トランザクションコード+日付を組み合わせたスマートキーを作るのはよくない。これらはいずれもカラムデータ

□ 重要なのは2番目の日付/時刻である「終了日時」:「次の」トランザクションの日時

- つまり2つの日時の「間」(1秒〜数か月)がこのプロフィールの有効範囲
- 現在有効のプロフィールでは終了日時は未来の適当な日時。nullはクエリーがしにくくなるのでよくない。

*筑波大学大学院経営システム科学専攻

□ 最終更新フラグ→現在有効なものをすぐ取れるために使用

- 更新があったら新しい行を追加し、これまでのはフラグを降ろす

□ この方法で大企業でもディスク量は大したことはない

- 100,000 人× 10 変更/年× 2,000 バイト/行× 5 年 =10GB

□ このトランザクション次元だけで多くの情報が取り出せる

- 特定社員に関する全経歴
- 過去の特定時点における全社員のプロフィール
- 取り出すだけでスナップショットになっているのは便利

□ ☆トランザクションに開始/終了日を付加することで、TYPE2SCD は過去時点の正確な断面を提供可能になる

□ トランザクション次元の詳細についてきちんと考える必要

- HR 運用システムは細かいデータを保持していて、全部データウェアハウスに取り込みたくはないかも
- たとえば「昇進」多数の個別変更 (マイクロトランザクション) の累積として記録されているかも→我々は 1 つの「昇進」として扱いたい
- このような「複合トランザクション」をうまく取得するのは簡単ではないかも。HR 運用システム側が対応する情報を持っているのがベストだが。

2.1 Time-Stamped Dimension with Periodic Snapshot Facts

□ 社員トランザクション次元はタイムスタンプ付きなのである種の fact table なんじゃないかと思うかも知れない。

- 技術的にはそうかも知れないが、実際にはその内容はテキスト文字列が中心で、クェリー条件や出力ラベルとして参照→次元テーブルと考えるのがよい
- 社員次元を必要とするようなすべての fact table と組み合わせて使うこともできるし (社員代理キーの代わりに社員トランザクション代理キーを使う)

□ 毎月のサマリーのようなものを作るのにも使いたい→社員数、給与総額、休暇取得/残余日数、新入社員数、昇進数、など

- すべての断面において調べたい→当該範囲にトランザクションがなくても

□ HR 定期スナップショット fact table(図 8.3) を用意。

- 月次元は普通に月をあらわす。社員トランザクションキーはその月の最後のトランザクションをあらわす。組織次元は企業の各組織をあらわす。

□ fact はその月ぶんの数値サマリーで、トランザクションから直接計算するのが難しいもの。マネージャからの検索要求を満たす。

- すべての値は (balance を除いて) すべての次元に対して加算可能。balance は準加算可能、他の次元について足した後、時間平均を取る必要がある。

3 Audit Dimension

□ fact table は「真だと分かっているすべてのこと」を記録。この延長として、メタデータのキー情報を入れておくことも考える

- fact を供給したソースシステム
- 行を挿入したソフトウェア/バージョン
- 行挿入時の allocation logic のバージョン
- Not Applicable の詳細…未知、不可能、破損、未到着
- その fact が最初の投入後変更されたか、されたのならなぜか
- 値が標準偏差の 2(3、4) 倍以上外れているか

□ 前半 3 つはデータの起源、後半 3 つはデータの信頼性にかかわる

- このようなデータを入れるには図 8.4 のように audit 次元を作りそのキーを入れる
- これによりメタデータが普通のデータ同様分析可能になる

□ audit 次元の内容はテキスト。「Employee extract versin 5 using ETL vendor ABC release 6.4」「Altered」「Not Altered」など

- ELT(extract-transoformation-load) プロセスでこれらの内容を調整して投入。トランザクションが多ければほとんどは同じ値のはず。

4 Keyword Outirigger Dimension

□ たとえば、IT 部門が社員次元に「IT スキル情報」を追加したいと考えたとする。

- 1 人の社員が非常にさまざまな IT スキルを持つことがあり得る。
- スキル自体もさまざまだが、ある程度構造はありそう。プログラミング言語 (Cobol、C++、Pascal)、OS(Unix、Windows、Linux)、DBMS、…
- スキルを指定して誰がそのスキルを持つか調べたりもしたい

□ 各社員が可変個のスキルを持つ→スキルは多値次元の候補

- キーワードは本質的に open ended…絶えず新しいものが追加
- 図 8.5 →スキルを outrigger として設計。場合によっては fact table に混ぜることもあるが (13 章参照)

□ Skill Group はある集合 (例:Oracle+Unix+SQL) に対して 1 つ。この 3 つをスキルとして持つ社員は同じ Skill Group Key を持つ。

- スキルごとのデータが多くなれば分けるがここでは 2 つしかないので混ぜてある

4.1 AND/OR Dilemma

□ 図 8.5 のスキーマでの問題点

- OR クエリー (例: Oracle または Linux のスキルを持つ社員) は簡単
- AND クエリー (例: 〃かつ〃) は outrigger で 2 つの行の分かれるので難しい
- 解決策: SQL で union/intersection を使ったクエリーを用意してユーザからは隠す

```
(SELECT EMPLOYEE_ID, EMPLOYEE_NAME
 FROM EMPLOYEE, SKILLS
 WHERE EMPLOYEE.SKILLGROUP = SKILLS.SKILLGROUP
  AND SKILL = "UNIX")
 UNION / INTERSECTION
(SELECT EMPLOYEE_ID, EMPLOYEE_NAME
 FROM EMPLOYEE, SKILLS
 WHERE EMPLOYEE.SKILLGROUP = SKILLS.SKILLGROUP
  AND SKILL = "LINUX")
```

4.2 Searching for Substrings

□ 図 8.6 のように単純な形にして and/or を避けることもできる

- スキル記述は 1 つの長い文字列にして決まった区切り文字で区切る (例: /UNIX/C++/)
- ただし多数の社員が同じ行を共有する場合。そうでないなら文字列を各社員ごとに入れてしまう。
- 文字列探索は曖昧だったりして難しい。UNIX? Unix? unix? (たとえば大文字に統一したりケース無視探索を使ったり…)

□ AND/OR 問題は次のように解決できる (%は多くの SQL で「任意文字列にマッチ」機能)

```
SKILL_LIST LIKE '%/UNIX/%' OR
  SKILL_LIST LIKE '%/LINUX/%'

SKILL_LIST LIKE '%/UNIX/%' AND
  SKILL_LIST LIKE '%/LINUX/%'
```

□ 標準 SQL の機能なのでどこでも動くという利点

□ ただし「%」が先頭にある探索はのろいという問題

- Skill List outrigger をメモリ常駐させると速くできる
- DBMS によっては特別なパターンインデックスを構築可能かも

5 Survey Questionnaire Data

□ 人事部はしばしば社員に対してサーベイを実施 (ピアレビュー、マネージャによる考課などでとくに)。

- これらのデータを分析したい (各社員の平均点数、部署の平均点数など)

□ このためには 1 つの質問ごとに 1 行を持つ fact table を作成 (図 8.7)

- 社員次元は「評価社員」「被評価社員」の 2 つの役割次元
- サーベイ次元はどういうサーベイかを記録
- 質問次元は質問内容とその分類カテゴリを記録 (たぶん同じ質問が複数のサーベイで用いられるし)
- サーベイ次元や質問次元を探索することで「こういう事項の評価」という形で探せる
- 応答次元は応答内容とカテゴリ (よい、悪い)

□ 図 8.7 のような単純なスキーマでも多くの分析が可能。さまざまなサーベイに応用可能。

6 Summary

- ☐ HR データの文脈でいくつかの概念を検討
- ☐ 次元テーブルをうまく修飾することで主要情報を格納するだけでなくトランザクションを追跡することまで可能
  - HR データの場合、これだけで多数のクェリーに対応可能
- ☐ `audit dimension` でデータの由来やメタデータ情報の格納
- ☐ キーワードグループの導入→ `outrigger` または文字列のリスト
- ☐ サーベイ/アンケートデータの取り扱い