さわって実感

リレーショナルデータベース設計 3

データベース設計(2)

日本PostgreSQLユーザー会 理事長 石井達夫 ISHII Tatsuo
ishii@postgresql.jp

PostgreSQL関係の出版続報

前回『魚拓本』(台湾版シーラカンス本)をご紹介しましたが，台湾に続いて韓国でもシーラカンス本が出版されました(図1)．筆者は韓国の文字コードは理解していますが，ハングルそのものはまったく理解していません．それでも以下のようなことがわかりました．

- 原著や台湾版がPostgreSQL 6.5ベースなのに対し，韓国版は7.0ベース
- サンプルデータはすべて日本語からハングルに変更済(台湾版はそのまま)
- configureの--enable-multibyteオプションをEUC_JPからEUC_KRにするなど微妙に調整している(台湾版はそのまま)

というわけで，韓国の事情に合わせて細かな調整を行った韓国の翻訳者はなかなかの技術レベルを持っているようです．

PostgreSQL関係の出版と言えば，PostgreSQLのコアメンバーの1人で，来日したこともあるBruce Momjian氏の『PostgreSQL： Introduction and Concepts』の日本語版が出版されました．『はじめてのPostgreSQL』(ピアソンエデュケーション，ISBN4-89471-461-2)です(図2)．コアメンバによる初の書籍ということで，翻訳を待ち望んでいた方も多かったのではないでしょうか．なお，翻訳は日本PostgreSQLユーザー会の石川氏を中心とするボランティアの方々が行いました．ですから，本書はまさにコミュニティから生まれた本と言えるのではないかと思います．

論理データベース設計を完了

さて，前回は論理データベース設計の途中で誌面が

図1 韓国版シーラカンス本

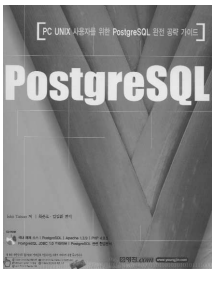


図2 はじめてのPostgreSQL

尽きてしまいましたので，今回は参照整合性制約を追加し，論理データベース設計を完了させるところから始めます．

参照整合性制約とは

前号のER図をもう1度示します（図3）．この図で「出版社」と「雑誌」の間の関係「出版」のように，1対Nになっている関係においては，例外なく参照整合性制約が成立すると考えて差し支えありません[注1]．

出版社を持たない雑誌は存在しないので，「雑誌」テーブルの「出版社名」列の値と同じものが必ず「出版社」テーブルの「出版社名」列に含まれているはずです．このような制約を**参照整合性制約**と呼び，また「雑誌」テーブルの「出版社」列を**外部キー**と呼びます．

以後，本稿では，あるテーブルのある列を「テーブル.列」と記述することにします．たとえば「雑誌.出版社名」などとなります．

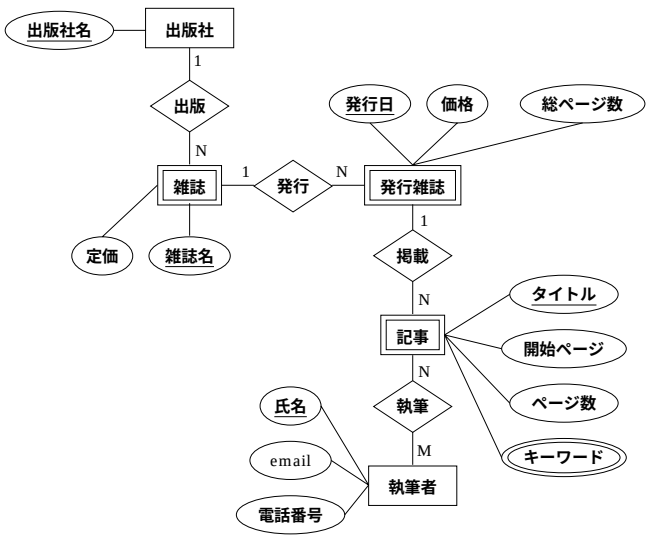
本稿の例で言えば，「出版社」テーブルを**被参照テーブル**，「雑誌」テーブルを**参照テーブル**，出版社.出版社名は**被参照列**，雑誌.出版社名は**参照列**と呼びます．被参照列や参照列は複数の列から構成されていても構いません．さらに被参照列は主キーかユニークキーでなければなりません．また参照列のほうは，同じ値を持つ列を含む行が複数あっても構いませんし，被参照列にはまったく参照されない列を含む行があっても大丈夫です．つまり，被参照列と参照列の関係は1:N（N >= 0）になっています（図4）．

参照整合性制約が設定されていると，デフォルトではデータベースシステムは以下のような動作をします．

❶出版社.出版社名に“技術評論社”が含まれていないのに雑誌.出版社名に“技術評論社”を含む行をINSERT/UPDATEしようとすると，エラーになります．ですから，まず出版社.出版社名に“技術評論社”を登録してから，次に出版社.出版社名に“技術評論社”を登録するという順序を守る必要があります．

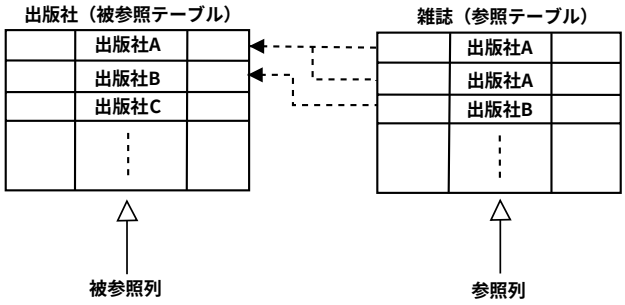
❷雑誌.出版社名に“技術評論社”が含まれているのに，出版社.出版社名から“技術評論社”を含む行をDELETEしようとすると，エラーになります．ですから，まず雑誌.出版社名から“技術評論社”を削除してから，次に出版社.出版社名から“技術評論社”を削除するという順序を守る必要があります．

図3 ER図



注1）「執筆」のようにN対Nの関係では，テーブル設計の段階で生成した「執筆」テーブルに対して参照整合性制約が成り立ちます．

図4 被参照列と参照列の関係



このほか，❶のような場合に整合性のチェックをトランザクションの最後にまとめて行うことによって一時的に不整合の状態を許すとか，❷の場合に出版社.出版社名から“技術評論社”を含む行をDELETEすると，自動的に雑誌.出版社名から“技術評論社”が含まれている行を削除することもできます．

参照整合性制約の追加

テーブルに参照整合性制約を追加するには，参照テーブル側で外部キーを宣言します．被参照テーブルのほうでは特に何もする必要はありません．

外部キーを宣言するには3つの方法があります．

①列宣言のときに外部キーも宣言する
②テーブル制約として外部キーを宣言する
③ALTER TABLE文を使って外部キー宣言を後から追加する

これらを順に説明します．

リスト1 列宣言における外部キー宣言

```
CREATE TABLE t1 (i INTEGER REFERENCES t2(i));
```

リスト2 テーブル制約としての外部キー宣言

```
CREATE TABLE 雑誌 (
  出版社名 TEXT NOT NULL,
  [中略]
  定価 INTEGER NOT NULL,
  CONSTRAINT 雑誌_出版社_外部キー FOREIGN KEY(出版社名)
  REFERENCES 出版社(出版社名)
);
```

●①列宣言のときに外部キーも宣言する

リスト1をご覧ください．この例では，t1テーブルのi列がt2列のi列を外部キーとして参照しています．

●②テーブル制約として外部キーを宣言する

①の方法では，複数列に対して外部キーを定義することができません．このような場合は，テーブル制約として外部キーを宣言します．テーブル制約とは，テーブル定義において，列を全部宣言した後改めて特定の列について定義された制約を指します．例をリスト2に示します．

“CONSTRAINT”から“REFERENCES 出版社(出版社名)”までが外部キーの宣言です．これで雑誌.出版社名が出版社.出版社名を参照していることが定義されます．「雑誌_出版社_外部キー」はこの参照整合性制約の名前で，そのデータベースの中でユニークであれば自由に名前を付けられますが，ここでは「参照テーブル_被参照テーブル_外部キー」という決めで名前を付けています[注2]．これならば名前が重複する心配がありません．

なお，参照整合性制約の名前は，参照整合性制約に違反したときのエラーメッセージに使用されます．例を示します．

```
ERROR: 雑誌_出版社_外部キー referential integrity
violation - key
referenced from 雑誌 not found in 出版社
```

注2）ただし，32バイト以内の文字列でなければなりません．32バイトを超えた分は適当に省略されてしまいます．

●③ ALTER TABLE文を使って外部キー宣言を後から追加する

ALTER TABLE文を使い，定義済のテーブルに後から外部キー宣言を追加することができます（リスト3）．

この方法により，②では不可能な，参照関係が循環している外部キー宣言も可能になります．本連載では使用しませんが，一応例を示しておきましょう．リスト4になります．

ここで，“DEFERRABLE INITIALLY DEFERRED”という宣言が目新しいのですが，これは参照整合性のチェックを遅延することを意味します．これがないとどうなるでしょう？

たとえば，まず最初にt1.iに1をINSERTしたいとします．しかし，参照整合性制約により，その前にt2.iに1が登録済であることが要求されます．しかし，そのためにはt1.iに1が登録済であることが要求されます．しかし，t1.iに1をINSERTするためには，その前にt2.iに1が登録済であることが要求されます…これでは堂々巡りですね．このような矛盾を防ぐために，DEFERRABLE INITIALLY DEFERREDによる参照整合性制約の一時的な「棚上げ」が必要になります．棚上げになった参照整合性制約は，トランザクションがコミットしたときにチェックされます．すなわち，最後につじつまが合っていれば良いわけです．

例を示します．

```
BEGIN;
INSERT INTO t1 VALUES(1);
INSERT INTO t2 VALUES(1);
COMMIT;
```

なお，コミット前にあえて整合性をチェックしたい場合は，SET CONSTRAINT文を使います．

```
SET CONSTRAINT t1_t2_外部キー IMMEDIATE;
SET CONSTRAINT t2_t1_外部キー IMMEDIATE;
```

最後に注意ですが，今のところPostgreSQLはADD CONSTRAINTで追加した外部キー宣言を後から変更したり，削除することができません．テーブルを一旦削除し，テーブルを新たに作り直すことから始めなければなりません．

例題に外部キー追加してみる

では，前回作ったテーブル定義に外部キーを追加してみましょう．今回は，③の方法を使います．

リスト5（次ページ）が完成したスキーマ定義文です[注3]．本当は完成したスキーマ定義を再びER図にしておいたほうが良いのですが，今回使用しているER図の記法はややかさばってしまい何度も誌面で紹介するには不便なので，もっと簡便な方法として，連載第1回で取り上げたpgaccessというツールで図を書いて代用することにします．

図5がそれです．pgaccessの「クエリ」→「新規」→「ビジュアルデザイナ」と選んでいくと「ビジュアルクエリデザイナ」という画面になるので，そこで左上の「テーブル追加」からテーブルを選択するとテーブル名と列名が記入された箱が表示されます．今回は，作成したテーブルについてすべて箱を表示した後，参照整合性制約で結ばれている列名をマウスでドラッ

リスト3　ALTER TABLEによる外部キー宣言

```
ALTER TABLE 雑誌 ADD CONSTRAINT 雑誌_出版社名_外部キー
FOREIGN KEY(出版社名) REFERENCES 出版社;
```

リスト4　参照関係が循環している外部キー宣言

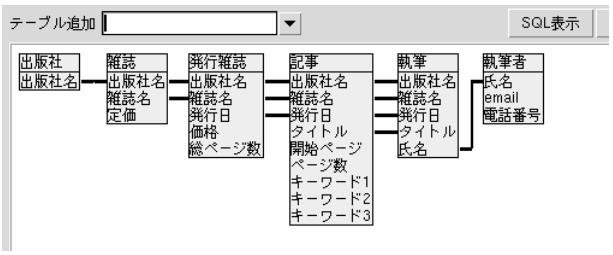
```
CREATE TABLE t1(i INTEGER PRIMARY KEY);
DROP TABLE t2;
CREATE TABLE t2(i INTEGER PRIMARY KEY);
ALTER TABLE t1 ADD CONSTRAINT t1_t2_外部キー FOREIGN KEY(i) REFERENCES t2
DEFERRABLE INITIALLY DEFERRED;
ALTER TABLE t2 ADD CONSTRAINT t2_t1_外部キー FOREIGN KEY(i) REFERENCES t1
DEFERRABLE INITIALLY DEFERRED;
```

注3）ここでは被参照テーブルの列指定がありませんが，被参照列が主キーの場合は，このように列指定を省くことができます．

リスト5　参照整合性を追加したスキーマ定義文

```
DROP TABLE 出版社;
CREATE TABLE 出版社 (
       出版社名 TEXT PRIMARY KEY,
       CONSTRAINT 出版社_出版社名_check CHECK(character_length(出版社名) > 0)
);

DROP TABLE 雑誌;
CREATE TABLE 雑誌 (
       出版社名 TEXT NOT NULL,
       雑誌名 TEXT NOT NULL,
       定価 INTEGER NOT NULL,
       PRIMARY KEY (出版社名,雑誌名),
       CONSTRAINT 雑誌_出版社名_check CHECK(character_length(出版社名) > 0),
       CONSTRAINT 雑誌_雑誌名_check CHECK(character_length(雑誌名) > 0),
       CONSTRAINT 雑誌_定価_check CHECK(定価 > 0)
);

ALTER TABLE 雑誌 ADD CONSTRAINT 雑誌_出版社名_外部キー
      FOREIGN KEY(出版社名) REFERENCES 出版社;

DROP TABLE 発行雑誌;
CREATE TABLE 発行雑誌 (
       出版社名 TEXT NOT NULL,
       雑誌名 TEXT NOT NULL,
       発行日 DATE NOT NULL,
       価格 INTEGER NOT NULL,
       総ページ数 INTEGER NOT NULL,
       PRIMARY KEY(出版社名,雑誌名,発行日),
       CONSTRAINT 発行雑誌_出版社名_check CHECK(character_length(出版社名) > 0),
       CONSTRAINT 発行雑誌_雑誌名_check CHECK(character_length(雑誌名) > 0),
       CONSTRAINT 発行雑誌_価格_check CHECK(価格 > 0),
       CONSTRAINT 発行雑誌_総ページ数_check CHECK(総ページ数 > 0)
);

ALTER TABLE 発行雑誌 ADD CONSTRAINT 発行雑誌_雑誌_外部キー
      FOREIGN KEY(出版社名,雑誌名) REFERENCES 雑誌;

DROP TABLE 記事;
CREATE TABLE 記事 (
       出版社名 TEXT NOT NULL,
       雑誌名 TEXT NOT NULL,
       発行日 DATE NOT NULL,
       タイトル TEXT NOT NULL,
       開始ページ INTEGER NOT NULL,
       ページ数 INTEGER NOT NULL,
       キーワード1 TEXT,
       キーワード2 TEXT,
       キーワード3 TEXT,
       PRIMARY KEY(出版社名,雑誌名,発行日,タイトル),
       CONSTRAINT 記事_出版社名_check CHECK(character_length(出版社名) > 0),
       CONSTRAINT 記事_雑誌名_check CHECK(character_length(雑誌名) > 0),
       CONSTRAINT 記事_タイトル_check CHECK(character_length(タイトル) > 0),
       CONSTRAINT 記事_開始ページ_check CHECK(開始ページ > 0),
       CONSTRAINT 記事_ページ数_check CHECK(ページ数 > 0)
);

ALTER TABLE 記事 ADD CONSTRAINT 記事_発行雑誌_外部キー
      FOREIGN KEY(出版社名,雑誌名,発行日) REFERENCES 発行雑誌;

DROP TABLE 執筆;
CREATE TABLE 執筆 (
       出版社名 TEXT NOT NULL,
       雑誌名 TEXT NOT NULL,
       発行日 DATE NOT NULL,
       タイトル TEXT NOT NULL,
       氏名 TEXT NOT NULL,
       PRIMARY KEY(出版社名,雑誌名,発行日,タイトル,氏名),
       CONSTRAINT 執筆_出版社名_check CHECK(character_length(出版社名) > 0),
       CONSTRAINT 執筆_雑誌名_check CHECK(character_length(雑誌名) > 0),
       CONSTRAINT 執筆_タイトル_check CHECK(character_length(タイトル) > 0),
       CONSTRAINT 執筆_氏名_check CHECK(character_length(氏名) > 0)
);

ALTER TABLE 執筆 ADD CONSTRAINT 執筆_記事_外部キー
      FOREIGN KEY(出版社名,雑誌名,発行日,タイトル) REFERENCES 記事;
ALTER TABLE 執筆 ADD CONSTRAINT 執筆_執筆者_外部キー
      FOREIGN KEY(氏名) REFERENCES 執筆者;

DROP TABLE 執筆者;
CREATE TABLE 執筆者 (
       氏名 TEXT PRIMARY KEY,
       email TEXT UNIQUE,
       電話番号 TEXT UNIQUE,
       CONSTRAINT 執筆者_氏名_check CHECK(character_length(氏名) > 0)
);
```

図5 pgaccessでER図もどきを書いてみる



グ&ドロップして重ね，線で結んで外部キーを表現してみました．本来ですとこれはテーブルの結合を行うための操作で，このようにER図もどきを描くために使うのは邪道なのですが，使えるものは何でも使うというのが筆者のモットーなので，ありがたく利用させていただきます[注4]．

正規化

これで一応スキーマ定義ができましたが，この段階ではリレーショナルモデルの観点から見て好ましくない性質を持ったものが含まれる可能性があるため，正規化という作業を通じて修正を行います．一般に正規化を行うと，テーブルの数が増えるため，性能の低下を嫌って正規化のフェーズを省く方がいますが，これはよくありません．正規化されていないテーブルは，誤ったデータを入力できてしまったり，整合性を維持するために余計な負担がかかることがあるからです．まず正規化を行い，その結果どうしても目的の性能が達成できない場合にのみ正規化を崩す作業（非正規化と呼びます）を行うべきです．

正規化のレベル

正規化にはその厳密さによっていくつかのレベルがあります．

●第一正規形（1NF：first normal form）

このレベルでは，以下が要求されます．

①列の値が原子的，すなわちそれ以上分解できない値であることです．原子的でない例としては，配列，リスト，あるいはテーブルの中にさらにテーブルを含んでいるようなものが挙げられます．

②繰り返しが含まれていないこと．たとえば，売り上げのテーブルがあったときに，1つのテーブルの中に，1日の売り上げ，2日の売り上げ…のように日毎の売り上げが繰り返し表れるような場合です．

このような観点でもう1度リスト5のスキーマ定義を見てみると，「記事」テーブルでは「キーワード1」「キーワード2」「キーワード3」というように，「キーワード」が繰り返し表れているので，②を満たしていません．すなわち，第一正規形になっていないことがわかります．第一正規形を満たしていないといろいろと不都合があります．たとえばこのテーブルでは，

- キーワードが3つまでしか登録できない
- キーワードが1つもないことを表現できない

このような場合，繰り返し表れている列を別テーブルに移すのが解決法です．

注4）最近のpgaccessには「スキーマ」という機能があり，似たような図を書くことができます．というか，たぶん「スキーマ」という名前からして本稿の目的にはこちらが適しているような感じがするのですが，まったくドキュメントがなく，どうもまだ未完成の機能のようなので今回は使用しませんでした．

3 データベース設計(2)

リスト6　キーワード格納テーブル

```
CREATE TABLE キーワード (
       出版社名 TEXT NOT NULL,
       雑誌名 TEXT NOT NULL,
       発行日 DATE NOT NULL,
       タイトル TEXT NOT NULL,
       キーワード TEXT NOT NULL,
       keyword TEXT NOT NULL,
       PRIMARY KEY(出版社名,雑誌名,発行日,タイトル,キーワード),
       CONSTRAINT キーワード_出版社名_check CHECK(character_length(出版社名) > 0),
       CONSTRAINT キーワード_雑誌名_check CHECK(character_length(雑誌名) > 0),
       CONSTRAINT キーワード_タイトル_check CHECK(character_length(タイトル) > 0),
       CONSTRAINT キーワード_キーワード_check CHECK(character_length(キーワード) > 0)
);
```

リスト7　参照整合性制約の追加

```
ALTER TABLE キーワード ADD CONSTRAINT キーワード_記事_外部キー
      FOREIGN KEY(出版社名,雑誌名,発行日,タイトル) REFERENCES 記事;
```

リスト8　記事テーブルの作成

```
CREATE TABLE 記事 (
       出版社名 TEXT NOT NULL,
       雑誌名 TEXT NOT NULL,
       発行日 DATE NOT NULL,
       タイトル TEXT NOT NULL,
       開始ページ INTEGER NOT NULL,
       ページ数 INTEGER NOT NULL,
       PRIMARY KEY(出版社名,雑誌名,発行日,タイトル),
       CONSTRAINT 記事_出版社名_check CHECK(character_length(出版社名) > 0),
       CONSTRAINT 記事_雑誌名_check CHECK(character_length(雑誌名) > 0),
       CONSTRAINT 記事_タイトル_check CHECK(character_length(タイトル) > 0),
       CONSTRAINT 記事_開始ページ_check CHECK(開始ページ > 0),
       CONSTRAINT 記事_ページ数_check CHECK(ページ数 > 0)
);
```

●第一正規形への分解

まず，**リスト6**のようなキーワードを格納するテーブルを作ります．ついでに英文のキーワードを格納する「keyword」という列を作り，英語のキーワードで検索できるようにしてみました．また**リスト7**のように参照整合性制約も追加します．

一方「記事」テーブルのほうは，単に「キーワード1」「キーワード2」「キーワード3」を削除するだけです．この結果，**リスト8**のようになります．

pgaccessを使って図にしたものを**図6**に示します．

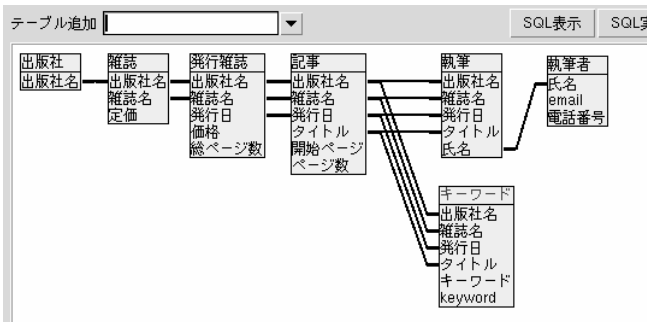
第一正規形での問題点

「キーワード」テーブルを設けたことにより，第一正規形になりましたが，よく見るとまだ不都合があります．

❶同じキーワードが複数回「キーワード」テーブルに現れることがあり，無駄です．たとえば，「データーベース」とか「PostgreSQL」というキーワードを持つ記事はたぶん複数あるでしょうから，その分だけ「キーワード」テーブルに行を追加しなければなりません．それだけでなく，「データーベース」という表記を仮に「データベース」という表記に変更したいとすると，「キーワード」テーブルにある複数のレコードの更新作業を行わなければならなくなります．このような問題を**修正不整合**と呼びます．

❷ある雑誌では，その分野で使われそうなキーワードがわかっているので，あらかじめ「キーワード」テーブルに登録することを考えました．しかし発行日やタイトルが不明な状態では「キーワード」テーブ

図6　第一正規形:「キーワード」を別テーブルにする



ルにキーワードを登録することはできません．これを**挿入不整合**と呼びます．

❸ある記事に「シーラカンス」というキーワードと対応する英語のキーワード“coelacanth”が登場したので「キーワード」テーブルに登録されましたが，さる事情によりこの記事はボツになり，その結果キーワードテーブルからもこの行が削除されました．ところが「シーラカンス」というキーワードは後にも先にもここで使われたきりでしたので，「シーラカンス」という「キーワード」テーブルの行は永久に消えてなくなり，その結果「シーラカンス」に対応する英語が“coelacanth”である，という事実もなくなってしまいました．このような問題を**削除不整合**と呼びます．

以上の3点をまとめて**更新不整合**と呼びます．

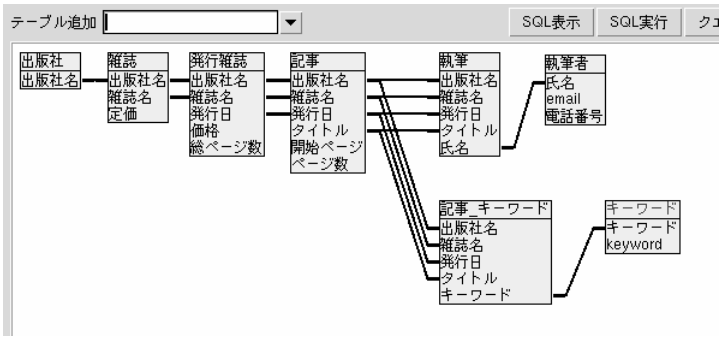
第三正規形への分解

更新不整合を解消するためには，第一正規形より

リスト9　記事_キーワードテーブルの作成

```
CREATE TABLE 記事_キーワード (
       出版社名 TEXT NOT NULL,
       雑誌名 TEXT NOT NULL,
       発行日 DATE NOT NULL,
       タイトル TEXT NOT NULL,
       キーワード TEXT NOT NULL,
       PRIMARY KEY(出版社名,雑誌名,発行日,タイトル,キーワード),
       CONSTRAINT 記事_キーワード_出版社名_check CHECK(character_length(出版社名) > 0),
       CONSTRAINT 記事_キーワード_雑誌名_check CHECK(character_length(雑誌名) > 0),
       CONSTRAINT 記事_キーワード_タイトル_check CHECK(character_length(タイトル) > 0),
       CONSTRAINT 記事_キーワード_キーワード_check CHECK(character_length(キーワード) > 0)
);

ALTER TABLE 記事_キーワード ADD CONSTRAINT 記事_キーワード_記事_外部キー
      FOREIGN KEY(出版社名,雑誌名,発行日,タイトル) REFERENCES 記事;

ALTER TABLE 記事_キーワード ADD CONSTRAINT 記事_キーワード_キーワード_外部キー
      FOREIGN KEY(キーワード) REFERENCES キーワード;

CREATE TABLE キーワード (
       キーワード TEXT NOT NULL,
       keyword TEXT NOT NULL,
       PRIMARY KEY(キーワード),
       CONSTRAINT キーワード_キーワード_check CHECK(character_length (キーワード) > 0),
       CONSTRAINT キーワード_keyword_check CHECK(character_length(keyword) > 0)
);
```

図7　図6を第三正規形にしたもの



もレベルの高い正規形である第三正規形（3NF：third normal form)を導入します．

今回の場合は，キーワード（和文，英文）を格納する独立したテーブルを作り（「キーワード」という名前にします」)，「記事」と「キーワード」の間の関係を表現する「記事_キーワード」というテーブルを新たに作ることによって，第三正規形とすることができます（リスト9)．

pgaccessを使って図にしたものを図7に示します．

●第一正規形の詳細

第三正規形は，第一正規形と違って自明なものではありませんので，少し系統立てて考えてみましょう．

図6の「キーワード」テーブルの主キーは{出版社名,雑誌名,発行日,タイトル,キーワード}です．主キーですから，{出版社名,雑誌名,発行日,タイトル,キーワード}の組が決まれば，行もただ1つだけ決まります．すると，対応するkeywordも決まります．つまり，{出版社名,雑誌名,発行日,タイトル,キーワード}が決まればkeywordも必ず決まるわけで，このことを

{出版社名，雑誌名，発行日，タイトル，キーワード}⇨{keyword}

と表現することにします．このような関係を関数従属性と呼び，このとき{keyword}は{出版社名,雑誌名,発行日,タイトル,キーワード}に関数従属すると言います．「キーワード」テーブルには，他にも関数従属性が存在します．

{キーワード}⇨{keyword}[注5]

あるテーブルが第三正規形であるためには，すべての関数従属性X⇨Aが以下の条件のうちのどちらかを満たしている必要があります．

①Xが候補キーまたは候補キーに候補キー以外の列を組み合わせたもの[注6]である
②Aが候補キーの一部である

図6の「キーワードテーブル」で，{出版社名,雑誌名,発行日,タイトル,キーワード}⇨{keyword}は①を満たしていますが，{キーワード}⇨{keyword}は①も②も満たしていないので，第三正規形の条件からは外れています．

もう1度図7の「キーワードテーブル」を見てみましょう．{出版社名,雑誌名,発行日,タイトル,キーワード}⇨{keyword}は①を満たしており，かつこれ以外に関数従属性がないので「記事_キーワード」テーブルは第三正規形です．「キーワード」テーブルも唯一の関数従属性{キーワード}⇨{keyword}が①を満たしているので第三正規形です．

テーブルが第三正規形の条件を満たすことにより，

注5）1つの日本語キーワードに対して複数の英語のキーワードが対応するようなケースはここでは考えません．
注6）このようなキーを超キーと呼びます．

第一正規形の条件のみを満たすテーブルで見られたような不都合はほとんど解消されます．正規形にはさらに高次のものもありますが，実用的には第三正規形止まりで十分な場合が多いので，本稿でもここまでとします[注7]．

第三正規形への正しい分解

第一正規形から第三正規形にテーブルを分解するときに注意しなければいけないのは，適当に分解したのでは正しい第三正規形のテーブルが得られない可能性があることです．つまり，テーブルを分解したことにより情報が失われたり，もともとなかった余計な情報が追加されてしまうかもしれないということです．

例を挙げましょう．あるレースに出場するチーム（T），各チームに所属する複数のドライバー（D），ドライバーが操縦する車（C）の関係を記述したテーブル{D,C,T}があったとします．あるドライバーが複数のチームに所属することはありませんし，また車も複数のチームに所属することはありません．このレースは24時間レースなので，ある車を担当するドライバーは複数いて，随時交替するものとします．ただし，あるドライバーが複数の車に乗ることはありません．このとき以下のような関数従属性が成り立ちます[注8]．

- D⇨C

ドライバーが決まれば，乗る車がただ1つに決まる

- C⇨T

車が決まれば，所属するチームがただ1つに決まる

テーブル{D,C,T}は明らかに第三正規形ではありません．そこで複数のテーブルへと分解するのですが，その方法は3通りあります．

{D,T}と{T,C}

{D,C}と{T,C}

{D,C}と{T,D}

ところが，この中で{D,T}と{T,C}の組合わせは，元の{D,C,T}にはないデータを産み出してしまいます．そのことを確かめてみましょう．

たとえば，図8のように{D,C,T}にデータが格納されているとします．すると，{D,T}と{T,C}のデータは図9のようになります．

ところが，この2つのテーブルを自然結合（同じ列名で結合するJOIN），すなわち

```
SELECT d,c,t FROM TD NATURAL JOIN TC ORDER BY d,c,t;
```

または

図8　{D,C,T}のデータ

d	c	t
D1	C1	T1
D2	C1	T1
D3	C2	T1
D4	C2	T1
D5	C3	T2
D6	C3	T2

図9　{D,T}と{T,C}のデータ

d	t
D1	T1
D2	T1
D3	T1
D4	T1
D5	T2
D6	T2

t	c
T1	C1
T1	C2
T2	C3

図10　自然結合の結果

d	c	t
D1	C1	T1
D1	C2	T1
D2	C1	T1
D2	C2	T1
D3	C1	T1
D3	C2	T1
D4	C1	T1
D4	C2	T1
D5	C3	T2
D6	C3	T2

注7）第一正規形からいきなり第三正規形へと飛んでしまいましたが，実はこの間に第二正規形というものが存在します．ただ，これは第三正規形を導き出すために理論的に導入されたものであり，実用的な意味はほとんどないので本稿では説明は省略します．

注8）このほか，D⇨T（ドライバーが決まれば，所属するチームがただ1つに決まる）という関数従属性も考えられますが，これはD⇨C，C⇨Tから当然の帰結として推論できる内容なので，ここではあえて挙げていません．また，このような論理的に推論できる関数従属性を除いた最小限の関数従属性の集合を極小被覆と呼びます．

```
SELECT d,c,TC.t FROM TD,TC WHERE TD.t = TC.t
ORDER BY d,c,TC.t;
```

を実行すると図10のようになり，元のテーブルにはない，

```
D1 | C2 | T1
D2 | C2 | T1
D3 | C1 | T1
D4 | C1 | T1
```

というデータを産み出してしまいます．したがって，これは正しい第三正規形の分解ではありません．

第三正規形への無損失結合分解

それでは，第一正規形から第三正規形への正しい分解（**無損失結合分解**）を行うためにはどうしたらよいのでしょう？ これには，関数従属性が鍵になります．関数従属性を使えば，以下のような方法で第三正規形への無損失結合分解を行うことができます．先の{D,C,T}を例に使って説明しましょう．

❶まずD⇨Cの右辺と左辺を列として持つ{D,C}というテーブルを作ります．
❷同様に，C⇨Tから{C,T}を作ります．
❸最後に{D,C,T}の候補キー(D)が生成されたテーブルに含まれているかどうか調べます．今回は{D,C}に含まれているのでOKですが，もし含まれていなければ，それを列として持つテーブルを新たに追加します．
❹結果として{D,C}と{C,T}を得ます．

{C,T}の列名を左右入れ替えれば，先ほど例として引き合いに出した{D,C}および{T,C}の組み合わせと一致することがわかります．

では{D,C}と{T,D}はどうでしょう．前ページの注8により，D->CとC->Tという関数従属性の組からD->CとD->Tの組が導き出されます．D⇨CとD⇨Tに対して上記ステップと同じ処理を行えば，{D,C}と{T,D}の組合わせが導き出されます．

関数従属性と正規化の関係

ここまで説明したように，正規化にあたっては関数従属性が重要な役割を果たします．というか，テーブルのスキーマ定義を見ただけでは正規化は不可能ですし，関数従属性が変化すれば正規化の結果得られるテーブルも変化します．したがって，事前に関数従属性をしっかり定義しておくことが重要です．

また，今回のような簡単な例では簡単に正規化が行えましたが，複雑で関数従属性がたくさんあるようなテーブルでは，正規化は簡単にはいきません．参考文献[1]では，第2章で航空路線に関するそのような例が出てきます．そこでは全部で10個もの関数従属性が登場します．ただ，いくら複雑なものでも，きちんと手順を追っていけば必ず正しい解が求まります．この例では，10個の関数従属性を整理して極小被覆を求めて正規化を行っています．

リレーショナルデータベースはしっかりした数学的基礎に基づいているので，このようなことが可能になっています．他のデータモデルと比べ，リレーショナルモデルが持つ大きな利点の1つと言えるでしょう．

関数従属性の保存

さて，無損失結合分解により，元のテーブルと同等の情報を持つ第三正規形のテーブルを得られることがわかりましたが，元のテーブルが持っていた関数従属性に関してはどうなのでしょうか．実は，前述の手順で得たテーブルは，元の関数従属性も保存することが知られていますので，その点はご安心ください．

ボイス・コッド正規形（BCNF）

第三正規形の条件として，関数従属性(X⇨A)が，

①Xが候補キーまたは候補キーに候補キー以外の列を組み合わせたものである
②Aが候補キーの一部である

を満たす必要があると述べました．実は，第三正規形でも更新不整合を起こすことがあり，この点に関しては，条件②を外してより厳しい制約を課せばよいことがわかっており，この条件を満たす正規形をボイス・

コッド正規形と呼びます．であれば，すべてボイス・コッド正規形にすればよいような気がしますが，残念ながらボイス・コッド正規形では関数従属性が必ずしも保存されないという問題があります．

また，第四正規形や第五正規形といった，第三正規形よりも更に高度な正規形もありますが，これらについてもボイス・コッド正規形同様，関数従属性が必ずしも保存されません．

まとめ

前回に引き続き，データベースの論理設計を行いました．またさらに，参照整合性を追加し，続いて正規化を行いました．予定ではこの後物理設計まで行うつもりでしたが，誌面が尽きてしまったので，次回に回します．

テーブルの正規化はなかなか解説が難しいところで，できるだけ数式を使わずに直感的に理解できるようにしたつもりですが，その分記述の厳密さは甘くなっていますし，アルゴリズムの解説も省略されている部分があります．本稿を読んでもっと深く知りたくなった方は，参考文献[2][3]などを見てみることをお勧めします．本稿を執筆するにあたっても大変参考にしました．WD

参考文献：

[1]『プログラマのためのSQL 第2版』／Joe Celco／ピアソンエデュケーション／ISBN4-89471-480-9，2001

[2]『データベースシステム』／北川博之／昭晃堂／ISBN4-7856-2046-3，1996

[3]『データベース・システムの原理』／Jeffrey D. Ullman／日本コンピュータ協会／ISBN0-7167-8158-1，1988