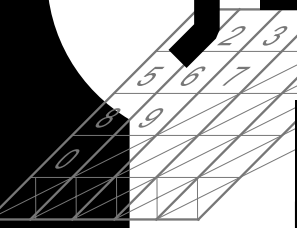
新連載

さわって実感

リレーショナルデータベース設計

1

リレーショナルデータベース入門



日本PostgreSQLユーザ会会長 石井達夫 ISHII Tatsuo
ishii@jp.postgresql.org

はじめに

読者のみなさんこんにちは．今回から『WEB+DB PRESS』で連載を始めることになった石井と申します．

私は今までも各所でPostgreSQLの紹介記事を書いているので，ご存知の方もいらっしゃるかもしれませんが，この連載ではPostgreSQL固有の話ではなくて，もっと広い視野からリレーショナルデータベース[注1]について基礎的な解説をしていきたいと思います．

連載の基本方針

ただ，リレーショナルデータベースの基礎論というと，どうしても「データモデル」「正規化」のような固い話になりがちで，それがためにリレーショナルデータベースの勉強を挫折した方も多いかと思います（かく言う私もそういう経験を何度もしています:-)）．しかしもちろん，リレーショナルデータベースを学ぶ上で，基礎論は避けて通れません．

そこで，本連載ではできるだけわかりやすく，かつ実践的に，実際にPostgreSQLなどを使って手を動かしながら，リレーショナルデータベースの世界を体験できるように解説を進めていきたいと思います．

対象読者

本連載では，以下のような方を対象として想定します．

①これからデータベースについて学びたい方
②すでにデータベースを使っているが，今一つ自分の作ったテーブルやその使い方に自信がない方

最近は技術の進歩は著しく，とりわけ本誌がターゲットにしているWebアプリケーションの分野では，ネットワーク，Apache，PHP，Javaやその他の技術に追いつくのが精一杯で，データベースというと，とりあえずテーブルを作ってアプリケーションから利用することはできる程度，というタイプ②の方が意外と多いのではないのでしょうか．

本連載では，そういう方たちもぜひ初心に帰って「今さら聞けないリレーショナルデータベースの基礎」を楽しく学んでいただければと思います．

これからの連載予定

あくまで「予定」ですが:-)，次のような流れで全6回の連載を行います．

注1）正確には「リレーショナルデータベースマネージメントシステム」（Relational Database Management System, リレーショナルデータベース管理システム）ですが，あまりにも長いので本稿ではリレーショナルデータベースと呼ぶことにします．

1 リレーショナルデータベース入門

●第1回（本号）リレーショナルデータベース入門

リレーショナルデータベースとは何か，リレーショナルデータベースのメリットとは何か，またリレーショナルデータベースの基礎理論（データモデル）などの話題を解説します．

こういった話題は往々にして文章と数式の羅列になりがちですが，それでは退屈すること請け合いですので，実際にリレーショナルデータベースソフトウェアであるPostgreSQLを「体験」しながら直感的に理解できるように話を進めていきます．そのため，厳密な説明はできませんので，必要ならば本稿末の参考文献[1][2]などをご覧ください．

●第2回　データベース設計（1）

連載第2回と第3回では，データベース設計の方法を解説します．第2回ではより上流の工程である「概念データベース設計」について解説します．

●第3回　データベース設計（2）

第3回は第2回の続きで，正規化から実際のデータベース作成（物理設計）までを取り上げます．

●第4回　トランザクション設計

実際にシステムを構築するためには，データベースを作るだけでなく，それをどのように利用するかを設計する必要があります．そのときに核心となるのが**トランザクション設計**です．

トランザクションはリレーショナルデータベースの重要な機能の1つで，データベースに対するデータの出し入れが矛盾なく行われるように管理します．

トランザクション機能自体は，どのリレーショナルデータベースでも共通ですが，実装方法は大きく異なります．そのため，ある程度実際の製品に即した解説にならざるを得ません．

ここでは強力なトランザクション機能を持つPostgreSQLを使い，さらにアプリケーション開発言語としてPHPを用いて，第3回までに作成したテーブルを利用して実際にアプリケーションを構築しながら，解説を進めます．

●第5回　チューニング

できあがったシステムは，要求された機能を満たすだけではなく，想定された範囲の性能で動作しなければなりません．システムが期待したほどの性能を出せない原因はいろいろ考えられますが，ここではデータベースに関連する部分のみ取り上げて性能改善する方法を説明します．このような作業のことを「チューニング」と呼びます．

チューニングのテクニックもリレーショナルデータベース製品に依存する部分が多々ありますが，ここでもPostgreSQLを例にとって解説します．

●第6回　運用管理

第5回までで期待される機能と性能を満たすシステムが作成できましたが，実際にシステムが運用に入ってからもやることはたくさんあります．連載最終回は締めくくりとしてバックアップなどの運用管理の話題を扱います．

また，読者の方々が今後自習するのに役立つ指針なども取り上げる予定です．

リレーショナルデータベースとは何か

ではさっそく連載第1回の主題である「リレーショナルデータベースとは何か」というところから入っていきましょう．

リレーショナルデータベースは，データベース管理システム（DBMS，Database Management System）の一種です．これはデータを管理することを目的にしたソフトウェアで，ある程度以上の規模のシステム開発では必ずといってよいほど採用されています．

データモデル

DBMSは独立したソフトウェアであり，他のサブシステムからアクセスするための明確なインタフェースが提供されています．そのようなインタフェースも含めて，どのようなデータの見え方を外部に提供しているかを**データモデル**といいます．データモデルにはさまざまなものがあり，**リレーショナルデータモデル**もその1つです．もちろんリレーショナルデータモデル

以外のデータモデルも存在します（図1）.

DBMSのメリット

データベースを使わなくても，ファイルを利用すれば，データを管理することができます．しかし，データベースをシステムに組み込むことによって，以下のようなメリットを享受することができます.

同時実行制御

複数のユーザが同時に同じデータにアクセスする際に，不整合が起きないように管理します．これを排他制御，あるいはもっと一般的に同時実行制御といいます．データベースは，これをもっと汎用的な枠組みであるトランザクションの形態で提供します.

トランザクションによるエラーリカバリ

トランザクションではいくつかの処理をまとめて，不可分のものとして実行します．これによってアプリケーションのエラー処理が著しく簡単になることがあります.

たとえば，

①処理Aを実行
②処理Bを実行
③処理Cを実行

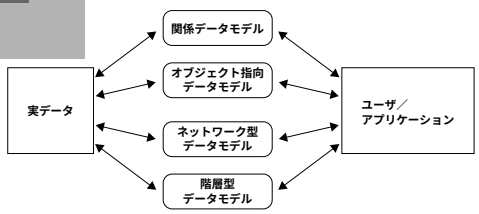
という一連の流れをトランザクションにしておくと，③の処理Cが失敗したときに，処理Aと処理Bを「なかったことにする」ことができます．これを**ロールバック**（Rollback）といいます．DBMSのロールバック機能を利用するアプリケーションは，①処理A，②処理Bを取り消す処理を自分で行う必要がありません.

セキュリティ

データには共有される必要がある一方で，機密を保つことが要求されます.

たとえば，ある社員データベースに対して，社員番号や氏名は一般社員に公開できるが，住所などの個人情報は人事部に所属する社員以外は参照できない，といったある意味相反する要求もあるでしょう．このような要求に応えるために，DBMSはさまざまなレベルのセキュリティ機能を備えています.

図1　データモデル



データの整合性の維持

単にデータを集めるだけでは，運用していくうちにデータの不整合が起きることがあります．そこで，データベース自体が誤ったデータを作り出してしまうことのないようにデータの管理を行います.

高速アクセス

ファイルを使うアプリケーションの場合，データ量が少ないうちはきびきび動いていたのに，データがちょっと増えたら指数関数的に遅くなったという話をよく耳にします.

そのために苦労してチューニングを行うことになってしまうわけですが，そんなことをするくらいなら，最初からデータベースを使ったほうがはるかに楽です．データベースは，何千万件ものデータを高速にアクセスするための各種機能を備えています.

データ独立

単にデータをファイルに蓄積する方法では，データをアクセスする方法がプログラムの中に埋め込まれているため，プログラムを変更するとデータの形式まで影響を受けることがあります.

逆に，性能を改善するためにデータ形式を変更すると，プログラムもそれに合わせて変更せざるを得ないこともあります.

たとえば，先ほどの社員データベースをファイルで実装したとします．同じ社員データをアクセスするプログラムでも，人事部と経理部では，おそらくまったく違った作りになるはずで，プログラムの中にデータ

1 リレーショナルデータベース入門

構造を埋め込む方式では，結局人事部用と経理部用の2つの社員データを用意しなければならなくなります．

一方，**データ独立**の考え方では，社員データベースは社員という現実の存在を反映したものであるべきであると定義します．この考え方の優れた点は，「人事部」「経理部」というデータの利用者の立場から独立した観点でデータベースを設計できることです．

このため，将来人事部や経理部の業務内容が変更されたり，あるいはたとえ社員データベースを利用するまったく新しい部署が設立されても，データベースの変更は最小限で済むはずです．

ちょっと息抜き

さて，普通の教科書では，次にリレーショナルデータモデルの話へと移っていくのですが，このままでは読者が退屈する危険性があるので:-)ちょっと息抜きをかねてコンピュータで操作をしていただきましょう．

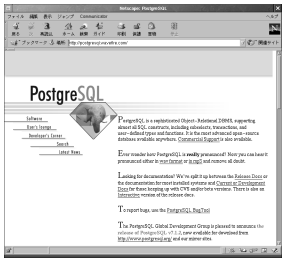
PostgreSQLとは

PostgreSQL（「ポストグレス」あるいは「ポストグレエスキューエル」と呼びます）は，米国カリフォルニア大学バークレー校で開発されたpostgresを元にして，さらに改良が加えられた拡張型リレーショナルデータベースです．

PostgreSQLは現在世界中のボランティアの手によって維持／改良されており，誰でも無償で自由に利用できます．ソースコードが完全に公開されており，機能的にも商用データベースにひけを取りません．

また本連載にとって都合が良いことに，PostgreSQLはSQL規格にかなり忠実に実装されていて，変な癖がないので，ここで学んだことは他のデータベースにも応用しやすいというメリットがあります．

図2 PostgreSQLの公式サイト



PostgreSQLの情報

PostgreSQLに関する公式の情報はhttp://www.postgresql.orgから手に入ります（図2）．

なお，トップページはいきなりミラーサイトの選択画面になっているので，適当なサイトを選択する必要があります．

日本語の情報がよいという方は，筆者が運営するサイトと，日本PostgreSQLユーザ会のサイトがおすすめです．

- 日本語PostgreSQL MLサポートページ
http://www.sra.co.jp/people/t-ishii/PostgreSQL/
- 日本PostgreSQLユーザ会のページ
http://www.jp.postgresql.org

また以下のURLで，PostgreSQLの日本語のドキュメントも公開されてます．ぜひダウンロードしておきましょう．

http://osb.sra.co.jp/PostgreSQL/Manual/

書籍なら，参考文献[3]をご覧ください．

PostgreSQLのインストール

PostgreSQLはLinuxとほとんどのUNIX系OSに移植されています．ここでは，Vine Linux 2.1（http://vinelinux.org/）を前提に，ソースからのインストール方法を説明します（他のOSでもほとんどインストール方法は同じです）．

もちろんバイナリパッケージも各種用意されていますが，

- 最新版のPostgreSQLのほうが，SQLサポートがより完全
- ソースからのインストールも比較的簡単

ですので，今回はソースからインストールしてみます．

詳しいインストール方法は，

http://osb.sra.co.jp/PostgreSQL/Manual/PostgreSQL-7.1-ja/installation.html

か，ソースコードの中のINSTALLを見てください．ここでは最低限の説明をします．以下，#で始まる行はrootでの実行，$で始まる行はroot以外での実行を表します．

●必要なもの

PostgreSQLのインストールには，コンパイラ（gcc）とGNU makeが必要です．

●ディスク容量

できるだけ一杯用意しましょう．200Mバイト以上は欲しいところです．

●古いRPMの削除

すでにPostgreSQLがRPMでインストールされていると干渉しますので，まず，ばっさりと削除します．

```
$ rpm -qa|grep postgres
```

と入力すると，postgresqlという名前の付いたパッケージの一覧表を得ることができますので，これを消去します．

```
# rpm -e パッケージ名…
```

●アカウント

PostgreSQLは，rootではインストールできません．PosstgreSQL専用のアカウントを1個，新たに用意しましょう．UIDが0以外なら名前は何でもいいのですが，ここではPostgreSQLの伝統に従って，“postgres”とします．

●インストールの実際

今回インストールするのは，本稿執筆時点で最新のPostgreSQL 7.1.2です．ソースコードは，

ftp://ftp.ring.gr.jp/pub/misc/db/postgresql/v7.1.2/postgresql-7.1.2.tar.gz

から入手するのがおすすめです．容量は大体8Mバイトくらいあります．これを/tmpに置いておくものとします．

以下，手順を示します（#で始まる行はroot権限で実行することをお忘れなく）．

```
# mkdir /usr/local/pgsql
# chown postgres /usr/local/pgsql
# su - postgres
$ tar xfz /tmp/postgresql-7.1.2.tar.gz
$ cd postgresql-7.1.2
$ ./configure --enable-multibyte=EUC_JP
$ make
$ make install
```

これで/usr/local/pgsql以下にPostgreSQLがインストールされましたが，PostgreSQLを使うためには，この後，コマンドサーチパスなどの設定，データベース領域の初期化，およびデータベースサーバの立ち上げが必要です．

●postgresユーザの.bashrcの編集

以下を$HOME/.bashrcに追加します．

```
PG=/usr/local/pgsql
PATH="$PATH":$PG/bin
export PGLIB=$PG/lib
export PGDATA=$PG/data
```

そして，postgresでログインし直します．

●データベース領域の初期化

initdbをインストール後，1度だけ実行します．

```
$ initdb
```

●データベースサーバの立ち上げ

データベースサーバ（postmaster）を立ち上げます．

```
$ postmaster &
```

これでPostgreSQLを使う準備ができました．次にデータベースユーザを登録します．いつも自分が使う

1 リレーショナルデータベース入門

アカウント名をwebdbとします.

```
$ createuser webdb
Shall the new user be allowed to create
databases? (y/n) y
y
Shall the new user be allowed to create
more new users? (y/n) n
n
CREATE USER
```

postgresユーザをログアウトし，いつものアカウントでログインします．そして，以下を$HOME/.bashrcに追加します.

```
PG=/usr/local/pgsql
PATH="$PATH":$PG/bin
```

そしてログインし直します．最後に自分用のデータベースを作ります.

```
$ createdb foo
CREATE DATABASE
```

これでPostgreSQLが使えるようになりました.

リレーショナルモデルとは

息抜き（？）が終わったので，またリレーショナルモデルの話に戻りましょう.

リレーショナルモデルは，1970年にEdger F.Codd博士が提案したデータモデルで，すべてのデータをリレーション（関係）の集まりとして表現します.

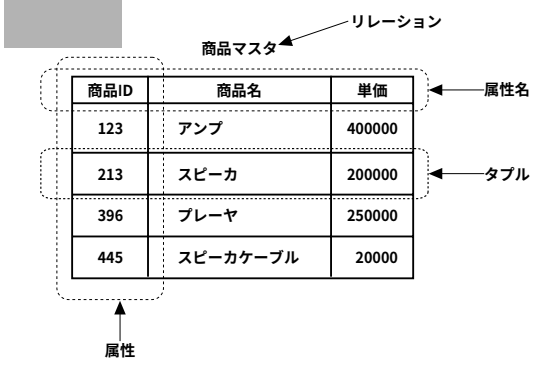
リレーションは図3のように「表」として表現することができます．この「商品マスタ表」はリレーションを表しているとも言えます.

リレーションは一般に，「商品ID」や「年月日」などの列を持ちますが，これは属性あるいはアトリビュート（attribute）と呼ばれ，名前（属性名）と決まった型を持ちます.

商品IDの型は整数ですが，商品名の型は文字列型です．型は基本的に，あらかじめ用意されたものの中から選ばれ，原則として後で変更されることはありません.

属性の数は次数と呼ばれます．このようにして決まるリレーションの構造をリレーションスキーマと呼びます．実際の表は，各属性の具体的な値が組になったもの（タプル）から構成されます．図3の中では，行として表現されています.

図3 リレーションの例



リレーショナルモデル≠表

このようにリレーショナルモデルを表を使って説明することがよくありますが，厳密には「リレーショナルモデル=表」ではありません．たまたま表の形で表現するとわかりやすいから，表を使っただけです.

リレーショナルモデルの表は，現実世界の表と比較して，次のような違いがあります.

- リレーショナルモデルの表では，タプル（行）や列の並んでいる順は無関係
- リレーショナルモデルの表では，重複した行は許されない
- リレーショナルモデルでは1つも行がない空の表も許される

これらの性質は，リレーショナルモデルの表が，数学でいう集合を表現しているからに他なりません．従って，ほかのデータモデルにない特徴があります.

- データの操作は，個々のデータを指定するのではなく，集合演算で行います．集合を直接操作するので，最初のデータを取り出して操作し，次に2番目

のデータを取り出して…といった手続き的な操作が必要がありません．逆に，個別のデータを取り出すためには，「商品名がXXXで…」のように値で指定します．

- 個々の表に対する集合演算だけではなく，複数の表に対して和／差／積などの集合演算を行うことができます．この結果もまた表になるので，さらに集合演算を施すことができます．

図3の「商品マスタ」はあるリレーション（関係）を表しますが，その属性である商品ID，商品名，単価は一定の制限のもとに値が設定されることが期待されます．

たとえば，商品IDは整数であり，商品分類がしやすいような何らかの規則性をもつ値を取るでしょう．商品名は文字列で，たぶん文字列長の制限があるでしょう．単価は明らかに整数で，しかも負の値は取らないはずです．

このように，各属性には一定のデータ型があり，しかもある範囲の値，もしくは決まった値のどれかを取ります．これを，その属性の**ドメイン**（domain，定義域）といいます．商品マスタというリレーションは，これらの各属性のドメインの取りうる値の範囲の集合を，順列組み合わせ的に掛け合わせたもの（直積）の部分集合になっています．

リレーションの意味

ところで，これまで使ってきた「リレーション」とは，どのような意味なのでしょうか．実は，リレーションは，「属性間の関係」を表現しているのです．

先の例で説明すると，「Xという商品の商品IDはYであり，単価はZである」を表現しているリレーションは，まさに「商品マスタ」という名前にふさわしい，と言えます．

しかし，ここで注意しなければならないことがあります．商品マスタという名前が付いているからといって，このリレーションに登録されるデータが必ず商品マスタとしてふさわしいもののみである，という保証はどこにもないことです．

たとえば，「1980年に生まれたA氏の年齢は現在21才である」という事実を，商品ID=1980，商品名=A氏，単価=21として商品マスタに登録することは可能です．ドメイン制約に違反していないからです．

このようなナンセンスなデータを排除し，商品マスタが商品マスタという意味を維持し続けるようにすることは，あくまでデータベースの利用者（ないしはアプリケーション）の仕事です．

リレーションに対する操作

さて，以上でごく簡単にリレーションの構造を説明したわけですが，これだけではまだ何も利用者にとって有用なものはありません．リレーショナルモデルの威力は，モデルに対する操作が集合操作であり，かつ簡潔に表現できる点にあります．

リレーショナルモデルの操作は**リレーショナル代数**など[注2]を使って数学的に厳密に定義されますが，少々わかりにくいので，ここではSQLを使って実際にデータベースを操作しながら説明することにします．

SQLとは

SQL（Structured Query Language）は，リレーショナルデータベースの標準言語で，ANSI（米国規格協会），ISO（国際標準化機構）によって規格化されています．

SQLは何度か改訂されており，最新の規格は通称「SQL99」です．もっとも，SQL99を完全に実装している製品はまだありませんし，ほとんどの製品は部分的にすら実装できていないのが実情です．ここでは，SQL99の1つ前の規格であるSQL92にもとづいてお話しすることにします[注3]．

タプル vs. 行？

実は，リレーショナルデータモデルとSQLには若干

注2）このほかにリレーショナル論理という一階述語論理にもとづいた体系があります．リレーショナル代数とリレーショナル論理は異なった体系ですが，数学的な記述力は同じです．
注3）SQL92に関する定評ある解説書としては，参考文献[4]などがあります．

1 リレーショナルデータベース入門

の違いがあります．

- リレーションには重複したタプルがありませんが，SQLは重複を許します．
- リレーショナルモデルにはデータの登録，変更，削除などの操作は定義されていませんが，SQLにはあります．
- リレーショナルモデルには集約関数がありませんが，SQLにはあります．

つまり，リレーショナルデータモデルを厳密に表現したものがSQLではないのです．そこで，リレーショナルデータモデルの用語と区別するため，以後，本稿では次の用語を用います．

- 「タプル」ではなくて「行」
- 「属性」ではなくて「列」
- 「リレーション」ではなくて「テーブル」

リレーショナル代数の操作とSQL文

リレーショナル代数には，いくつかの集合演算が含まれます．この集合演算操作は，SQLではすべてSELECTという単一のコマンドで実行できます．

ここでは，SELECT文を実際に操作しながらそれらを見ていきますが，その前に，まずテーブルを作り，行を登録しなければなりません．こういった操作はリレーショナル代数には含まれていませんが，SQLでは，CREATE TABLE，INSERT文で可能です．

CREATE TABLE文

テーブルを作成します（逆にテーブルを削除するのは，DROP TABLE文です）．

基本的な構文は以下です．

```
CREATE TABLE テーブル名 (
    列名 データ型,
    列名 データ型,
      :
);
```

SQL文ではほとんどの場合，大文字と小文字は区別されませんが，本稿ではわかりやすくするために，SQLのキーワードである“CREATE”や“TABLE”などを大文字で，それ以外は小文字で記述します[注4]．

後述するテストデータでは，データ型として数値にINTEGER（整数型），日付にはDATE型，文字列にはVARCHAR（可変長文字列型）を使用しています．他にSQLで定義されているデータ型はいろいろありますが，これは連載第2回以降で詳しく説明します．

INSERT文

作成したテーブルに行を追加するには，INSERT文を使います．INSERT文の基本的な構文は次です．

```
INSERT INTO テーブル名 VALUES (値, 値,…);
```

値は，CREATE TABLE文で定義した列の順に書きます．値の指定は，数値の場合は数字をそのまま，日付／文字列はシングルクォーテーション（'）で囲みます．

作ってみよう

では早速テストデータを作りましょう．**リスト1**を適当なテキストエディタで“create.sql”という名前で打ち込んでおきます[注5]．文字コードは，必ず日本語EUCにしておいてください．

次に，PostgreSQLの付属コマンドであるpsqlを使って以下のSQL文を入力してください．

```
$ psql -e -f create.sql
```

入力が完了したら，テーブルの内容を確認しましょう．

まずpsqlを引数なしで起動します．すると，インタラクティブモードになり，プロンプト“webdb=>”の後に続けてSQL文を入力できるようになります．1つ

注4）本稿では，わかりやすさのために，テーブル名や列名に日本語を使用していますが，決して日本語の使用を推奨しているわけではありません．むしろ，実際のプロジェクトでは日本語を使用しないことをおすすめします．

注5）打ち込むのが面倒な人は，本誌のWebページ（http://www.gihyo.co.jp/wdpress）からダウンロードして使ってください．

のSQL文のおわりはセミコロン（；）で，；が出てくるまでは改行しても構いません．

たとえば商品マスタ1の内容を見るには，図4のようにします．　ここで，“SELECT”の次の“*”は，全列の表示を意味します．

他のテーブルも確認しておきましょう（図5，図6）．

キー

今回の例題の中の商品マスタ1と2を見ると，商品IDという列があります．商品IDは1つのテーブルの中で重複しない数字が割り当てられており，商品IDを指定すれば必ず特定の行を選ぶことができるようになっています．このような列を候補キーと呼びます．

一方，売り上げテーブルでは，同じ商品ID，たとえば123が2回表れています．これは，同じ商品が別の日にも売れることがあるからで，そのため商品IDだけでは候補キーになりません．しかし，商品IDと年月日を併せたものなら必ずユニークになるので，これを候補キーとすることができます．このようなタイプのキーを連結キーまたは複合キーと呼びます．

NULL

現実世界では，テーブルの中のある値が未定，不明，あるいはそもそも値を適用すること自体が不適切[注6]という場合があります．

そのような状況のために，リレーショナルデータベースでは，NULL（「ヌル」あるいは「ナル」と読みます）という特別な属性値を用います．

主キー

候補キーのうち，NULLを含まず，かつ運用上もっとも適切なものを主キーと呼びます．

リスト1　create.sql

```
DROP TABLE 商品マスタ1;
CREATE TABLE 商品マスタ1 (
       商品ID INTEGER,
       商品名 VARCHAR(256),
       単価 INTEGER
);
DROP TABLE 商品マスタ2;
CREATE TABLE 商品マスタ2 (
       商品ID INTEGER,
       商品名 VARCHAR(256),
       単価 INTEGER
);
DROP TABLE 売り上げ;
CREATE TABLE 売り上げ (
       年月日 DATE,
       商品ID INTEGER,
       数量 INTEGER
);
INSERT INTO 商品マスタ1 VALUES(123, 'アンプ', 400000);
INSERT INTO 商品マスタ1 VALUES(213, 'スピーカ', 200000);
INSERT INTO 商品マスタ1 VALUES(396, 'CDプレーヤ', 250000);
INSERT INTO 商品マスタ1 VALUES(445, 'スピーカケーブル', 2000);
INSERT INTO 商品マスタ2 VALUES(558, 'テレビ', 50000);
INSERT INTO 商品マスタ2 VALUES(691, '冷蔵庫', 40000);
INSERT INTO 商品マスタ2 VALUES(756, '洗濯機', 60000);
INSERT INTO 商品マスタ2 VALUES(445, 'スピーカケーブル', 2000);
INSERT INTO 売り上げ VALUES('2001/7/1', 123, 1);
INSERT INTO 売り上げ VALUES('2001/7/1', 213, 2);
INSERT INTO 売り上げ VALUES('2001/7/1', 396, 1);
INSERT INTO 売り上げ VALUES('2001/7/1', 445, 2);
INSERT INTO 売り上げ VALUES('2001/7/2', 123, 2);
INSERT INTO 売り上げ VALUES('2001/7/2', 396, 1);
INSERT INTO 売り上げ VALUES('2001/7/2', 445, 2);
```

図4　商品マスタ1

```
webdb=> SELECT * FROM 商品マスタ1;
 商品id |       商品名       |  単価
--------+--------------------+--------
    123 | アンプ             | 400000
    213 | スピーカ           | 200000
    396 | CDプレーヤ         | 250000
    445 | スピーカケーブル   |   2000
(4 rows)
```

図5　商品マスタ2

```
webdb=> SELECT * FROM 商品マスタ2;
 商品id |       商品名       | 単価
--------+--------------------+------
    558 | テレビ             | 50000
    691 | 冷蔵庫             | 40000
    756 | 洗濯機             | 60000
    445 | スピーカケーブル   |  2000
(4 rows)
```

図6　売り上げ

```
webdb=> SELECT * FROM 売り上げ;
   年月日    | 商品id | 数量
-------------+--------+------
 2001-07-01 |    123 |    1
 2001-07-01 |    213 |    2
 2001-07-01 |    396 |    1
 2001-07-01 |    445 |    2
 2001-07-02 |    123 |    2
 2001-07-02 |    396 |    1
 2001-07-02 |    445 |    2
(7 rows)
```

注6）たとえば，「妊娠経験の有無」などという列に対して男性の場合には値の設定のしようがありません．

外部キー

売り上げテーブルには商品名の列がありませんが，商品マスタテーブルと結合することによって商品名を知ることができます．たとえば7/1の売り上げには商品ID=123というデータがあります．一方商品マスタ1を見ると，商品ID123の商品名列には「アンプ」とあります．ですから，7/1に1個売れた商品は実はアンプであることがわかりました．

このとき，売り上げテーブルの商品ID列は，外部のテーブルである商品マスタと結び付いているので，外部キーと呼ばれます[注7]．

基本演算

以上で準備ができたので，リレーショナル代数で定義されている基本演算を見ていきましょう．これらはそれほど多くありませんが，リレーショナルデータベースに関する操作のほとんどを行うことができます．

このような単純さがリレーショナルモデルの特徴の1つと言えましょう．

和（union）

2つのリレーションの和集合です．SQLでは，“UNION”という述語で指定します（図7）．

UNIONの対象となるリレーションは，列の数（次数）が同じで，かつ各属性のドメインも一致していなければなりません．

また，商品マスタ1，商品マスタ2には同じ

```
445 | スピーカケーブル |   2000
```

という行が含まれていましたが，UNIONの結果では重複が省かれていることに注意してください[注8]．

差（difference）

2つのリレーションの差集合です．SQLでは，“EXCEPT”という述語で指定します（図8）．

直積（cartesian product）

2つのリレーションの行，列を「総当たり」で結合します（図9）．

図7　和集合の例

```
webdb=> SELECT * FROM 商品マスタ1 UNION SELECT * FROM 商品マスタ2;
 商品id |      商品名       |  単価
--------+-------------------+--------
    123 | アンプ            | 400000
    213 | スピーカ          | 200000
    396 | CDプレーヤ        | 250000
    445 | スピーカケーブル  |   2000
    558 | テレビ            |  50000
    691 | 冷蔵庫            |  40000
    756 | 洗濯機            |  60000
(7 rows)
```

図8　差集合の例

```
SELECT * FROM 商品マスタ1 EXCEPT SELECT * FROM 商品マスタ2;
 商品id |   商品名    |  単価
--------+-------------+--------
    123 | アンプ      | 400000
    213 | スピーカ    | 200000
    396 | CDプレーヤ  | 250000
(3 rows)
```

図9　直積の例

```
webdb=> SELECT * FROM 商品マスタ1,売り上げ;
 商品id  |      商品名       |  単価  |   年月日    | 商品id  | 数量
---------+-------------------+--------+-------------+---------+------
    123  | アンプ            | 400000 | 2001-07-01  |    123  |    1
    123  | アンプ            | 400000 | 2001-07-01  |    213  |    2
    123  | アンプ            | 400000 | 2001-07-01  |    396  |    1
    123  | アンプ            | 400000 | 2001-07-01  |    445  |    2
    123  | アンプ            | 400000 | 2001-07-02  |    123  |    2
    123  | アンプ            | 400000 | 2001-07-02  |    396  |    1
    123  | アンプ            | 400000 | 2001-07-02  |    445  |    2
(中略)
    445  | スピーカケーブル  |   2000 | 2001-07-02  |    445  |    2
(28 rows)
```

図10　射影の例

```
webdb=> SELECT 商品名,単価 FROM 商品マスタ1;
      商品名       |  単価
-------------------+--------
 アンプ            | 400000
 スピーカ          | 200000
 CDプレーヤ        | 250000
 スピーカケーブル  |   2000
(4 rows)
```

注7）このように外部キーの列名と，それが指している他のテーブルの列名は同じにすることが多いのですが，これは外部キーを使う上での必要条件ではありません．

注8）あえて重複を許したい場合には，UNIONの後に“ALL”をつけ，“UNION ALL”とします．

射影（projection）

選択した列のみを出力します（図10）.

選択（selection）

選択した行のみを出力します（図11）.

選択条件としてはこの=（等しい）のほかにも，<>（等しくない），<（小さい），>（大きい），<=（以下），>=（以上）が使えます.

さらに，図12のように複数の選択条件のANDやORを指定することもできます.

また，SQLでは列に対する任意の四則演算，あるいは関数を適用した結果を用いて選択条件を指定することができます.

- 単価の1/2が20万以上であるもの（図13）
- 商品名が4文字より長いもの（図14）

そのほかの演算子

リレーショナルモデルにおける基本操作は以上の5つですが，比較的頻繁に出現する操作があるので，これを加えてリレーショナル代数の演算子とします.

●結合

直積と選択を組み合わせた演算で，2つのリレーシ

図11　選択の例

```
webdb=> SELECT * FROM 商品マスタ1 WHERE 商品ID=123;
 商品id | 商品名 |  単価
--------+--------+--------
    123 | アンプ | 400000
(1 row)
```

図12　選択の例（複数選択条件）

```
webdb=> SELECT * FROM 商品マスタ1 WHERE 商品ID=123 OR 商品ID=445;
 商品id |       商品名       |  単価
--------+--------------------+--------
    123 | アンプ             | 400000
    445 | スピーカケーブル |   2000
(2 rows)
```

図13　選択の例（単価の1/2が20万以上）

```
webdb=> SELECT * FROM 商品マスタ1 WHERE 単価/2 >= 200000;
 商品id | 商品名 |  単価
--------+--------+--------
    123 | アンプ | 400000
(1 row)
```

図14　選択の例（商品名が4文字より長い）

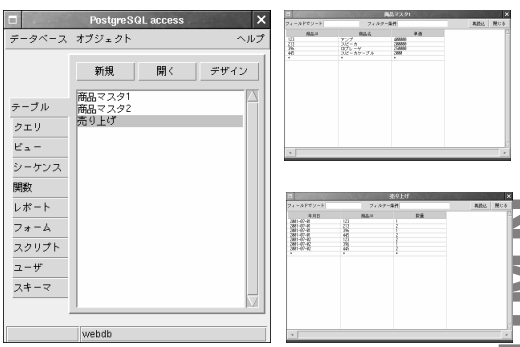
```
webdb=> SELECT * FROM 商品マスタ1 WHERE char_length(商品名) > 4;
 商品id |       商品名       |  単価
--------+--------------------+--------
    396 | CDプレーヤ         | 250000
    445 | スピーカケーブル |   2000
(2 rows)
```

コラム：PostgreSQLのGUI管理ツールpgaccess

PostgreSQLにはGUIで操作できるpgaccessというツールが標準で付属しています．右図のようにテーブルを選択してその内容を表示できます．もちろん，テーブルの内容を直接編集することもできます.

このほか，マウス操作だけでSELECTを実行するビジュアルクエリデザイナ，簡易アプリケーションビルダなどの機能もあります.

pgaccessを使うためには，Tcl/Tkというスクリプト言語が必要です．Vine Linuxでは，tcl-8.0.5_jp-9とtk-8.0.5_jp-9というパッケージがインストールされていればOKです．そしてPostgreSQLをconfigureするときに，--with-tclというオプションを追加，コンパイル，インストールすればpgaccessが使えるようになります.



なお，文字コードの関係でTcl 8.3では日本語の扱いがうまくいかないかもしれませんので，ご注意ください.

COLUMN

ョンのうち必要な部分を取り出します．外部キーを使った結合が典型的な例となります（図15）．

SQLでは，結果列に演算が書けますので，単価と数量を出力するかわりに，売り上げ高を自動的に計算させることもできます（図16）．

“単価*数量 AS 売り上げ”は，単価*数量を計算した結果を「売り上げ」という新しい列名で出力することを指示します．

共通部分（intersection）

2つの表の共通部分です．SQLではINTERSECTという述語を使います（図17）．

また，INTERSECTはEXCEPTの組み合わせでも表現できます（図18）．

◇ ◇ ◇

リレーショナル代数には，このほか「除算」という操作もありますが，あまり使われないのでここでは省略します．

まとめ

連載第1回目はデータベース管理システムの要件，リレーショナルデータモデルの概要についてお話ししました．また，実際に手を動かしてデータベースを体験するということでPostgreSQLのインストール，簡単な操作方法についてもお話ししました．次回はデータベース設計の話をしたいと思います．

◇ ◇ ◇

最後に，参考文献を掲載しておきます．

リレーショナルデータベースを含めて，データベースを学ぶための教科書としてはいろいろなものがありますが，私は主に以下のものを参考にしました．

[1]『データベースシステム』／北川博之／昭晃堂, 1996

[2]『データベースとデータモデル』／三浦孝夫／サイエンス社, 1997

拙著で恐縮ですが，PostgreSQLに関する総合的な解説書です．

[3]『改訂第3版 PostgreSQL完全攻略ガイド』／石井達夫／技術評論社, 2001

SQLを学ぶための本もいろいろありますが，以下は定評ある解説書です．

[4]『A Guide to THE SQL STANDARD, 4th edition』／C.J.Date／Addison-Wesley, 1997（訳書:『標準SQLガイド（改訂第4版）』／アスキー出版局, 1999）

図15　結合の例

```
webdb=> SELECT 年月日,商品名,単価,数量 FROM 商品マスタ1,売り上げ WHERE
商品マスタ1.商品ID=売り上げ.商品ID;
   年月日     |       商品名       |  単価  | 数量
------------+------------------+--------+------
 2001-07-01 | アンプ             | 400000 |    1
 2001-07-02 | アンプ             | 400000 |    2
 2001-07-01 | スピーカ           | 200000 |    2
 2001-07-01 | CDプレーヤ         | 250000 |    1
 2001-07-02 | CDプレーヤ         | 250000 |    1
 2001-07-01 | スピーカケーブル   |   2000 |    2
 2001-07-02 | スピーカケーブル   |   2000 |    2
(7 rows)
```

図16　結合の例（演算処理）

```
webdb=> SELECT 年月日,商品名,単価*数量 AS 売り上げ FROM 商品マスタ1,売
り上げ WHERE 商品マスタ1.商品ID=売り上げ.商品ID;
   年月日     |       商品名       | 売り上げ
------------+------------------+----------
 2001-07-01 | アンプ             |   400000
 2001-07-02 | アンプ             |   800000
 2001-07-01 | スピーカ           |   400000
 2001-07-01 | CDプレーヤ         |   250000
 2001-07-02 | CDプレーヤ         |   250000
 2001-07-01 | スピーカケーブル   |     4000
 2001-07-02 | スピーカケーブル   |     4000
(7 rows)
```

図17　intersection

```
webdb=> SELECT * FROM 商品マスタ1 INTERSECT SELECT * FROM 商品マ
スタ2;
 商品id |       商品名       | 単価
--------+------------------+------
    445 | スピーカケーブル   | 2000
(1 row)
```

図18　EXCEPTの組み合わせ

```
webdb=> SELECT * FROM 商品マスタ1 EXCEPT (SELECT * FROM 商品マス
タ1 EXCEPT SELECT * FROM 商品マスタ2);
 商品id |       商品名       | 単価
--------+------------------+------
    445 | スピーカケーブル   | 2000
(1 row)
```