データベースの超初級

Hukanzen

平成 30 年 11 月 21 日

目 次

第1章　はじめに

本書は若造も良いところの，未熟ポンコツ学生が教科書[1]の内容を削りつつ，自分なりにわかりやすく記述する．そのため，当然間違いや，歯抜けもあるだろうが，その時は，指摘してくれると有難い[2]．特に，SQL における概念も合わせて，説明する．

本書の用途としては，自分の振り返りや後輩等への教育，データベースに対するイメージ付け程度である．そのため，本書を理解したからと言って，データベースが完璧なわけではないし，本書がさっぱりわからないからと言って，データベースから遠ざかったりしないで欲しい．特に，リレーショナルデータベースなどは，数学の集合論に基づいて作成されている．そのため，当たり前のことだが，集合論に対する理解や知識が，最低限必要となってくる．しかし，高専 3 年や大学 1 年生といった，若造は，集合論の概念はなかなかハードルが高く，存在自体知らない場合もある．そこで，本書では極力数学的な知識を必要とせず，表や図を用いて説明を行い，補助的に数学を用いる程度に留める．もし，数学などを通じた理論を必要として，本書を見つけたのなら，即刻引き返して欲しい．おそらく，数学とデータベースの関連に気付いている人にとっては，まったくの無価値であろう．

そのような中身のないものであっても，これを用いて，データベースを周知させることは大切であると考える．世の中には，馬鹿げた話であるが，中規模で大量のデータのやり取りを行う場合でも，頑なにファイル保存から離れない開発者も存在する．データというものは，組織体にとって，共有資源であり，管理し，様々な場所に使用することに価値がある．1 つのプログラムに束縛されたデータなど，それは只の呪縛でしかないのだ．たまに，Microsoft Excel で全ての名簿や情報を取得し，管理した気になっている長がいる．何故かそういうファイルには既に消えた人の情報が根強く残っており，ふと他人が管理しようとすると,「この人まだいますか」などという会話を繰り広げることになる．また，新規追加や編集を行う場合でも，謎の VBA や参照で固めており，周りのデータに合わせてユーザが調整しなければならない．表計算ソフトは表計算ソフトであり，データベースではないのだ．数件ならば，まだいいが，本格的に数十件単位になると，非常に不便である．だが，そもそも何故このような状況が発生するのだろうか．それは，そもそもデータベースの存在や動作を知らないことにある．たしかに，パイプラインなどで一瞬でデータを保存できるのに対して，データベースはそもそも DBMS のインストールや使い方の勉強から始まる．

しかし，よく考えて欲しい，データベースに保存した場合の利便性，拡張性を．もし，複数のプログラムでデータを共有するときに，いちいちディレクトリ構造やシンボリックリンクを考えて，ファイルを置かなくてもよい．不十分なファイル移動や管理によって，データの矛盾に悩まされなくても良い．大量の Excel から，指定の場所をみつけて，行や列の挿入をしなくても良い．恐らくだが，人生単位で考えると，数日ぐらいは浮くだろう．これらの利点を新たな人材，人財に周知することで，人類は，不要なデータの管理から，解放されることを望む．そこで，まずは非常に初歩的な部分のみを，周知することで，データベースに対する不安や恐怖を，信頼や好奇心に変換したい．本書はその部分，いわゆるデータベースに対する「取っ掛かり」部分を補助することを目標とする[3]．

繰り返すが，本書は，あくまで，データベースに対する印象づけや，感覚を養う程度であり，理論に関しては，全く役に立たないことを御理解頂きたい．

[1] 問題があれば，教えてください (震え)．
[2] 当て字をよく使うので，お見苦しければ，指摘してください．
[3] 深夜に書いているため，統一性がないかも．

第2章　データベースとは

データベース[1]とは，コンピュータ内に構築された「実世界の写し絵」である．実世界のものを，データモデル[2]を用いて，データモデリング[3]することで，データベースを作成することができる．そして，データベース管理システムを用いて，データを管理[4]する．

用語としての「データベース」は主に，2 つの意味を持つ．

- コンテンツとしてのデータの格納庫
- Oracle，MySQL，PostgreSQL などの，データベース管理システム

しかし，それをいうと，Google 等で検索できるインターネットの情報は，地球規模のデータベースになってしまう．原義を考えると「データの基地」なので，ケースバイケースで読み取ろう．

2.1　データモデル

現在，データモデルには，大きく分けると 3 つ存在する．これらは世代ごとに順になっている．

第 1 世代 ネットワーク・データモデル，ハイアラキカル・データモデル

第 2 世代 リレーショナル・データモデル[5]

第 3 世代 オブジェクト指向・データモデル

一般的には，リレーショナル・データモデルに基づいた，リレーショナル・データベース[6]が広く使われおり，講義で行う内容もリレーショナル・データベース中心であることが多い．

第 1 世代は，1960 年代当時のコンピュータ事情に即して，考案された．親子集合とレコード型の要素を用いており，レコードとレコードは，ポインタで関連付けられる．そのため，細かなデータを得る場合は，繰り返し親から子に辿る必要があり，非効率である．

第 2 世代は，徹底的にフォーマルなデータモデルであり，数学の集合論に基づいて作成されてる．全てのデータを表で管理し，リレーショナル代数やリレーショナル論理を用いて，データ集合を得る．従って，ポインタやレコードの実装技術などとは，全く無関係である．なお，用語として，行を示すことを「レコード」と呼ぶことがあるが，恐らく間違った使い方である．

第 3 世代は，CAD 等のエンジニアリングデータやマルチメディアデータ，文書や図版が入り混じった文書データなどの部品展開構造を持つ，複合オブジェクトを表現するのに向いている．

以上より，実世界では同じ内容でも，データモデルの違いにより，異なって表現される．本来は，データベースに格納するデータに合わせて，それに適したデータモデルを使うべきである．しかしながら，一般的には，リレーショナル・データモデルで全て解決させてしまうことが多い．

2.2　データベース管理システム

データベース管理システム[7]とは，データベースに存在するデータを「管理」するシステムである．商用，オープンソースでは，Oracle，SQL Server，PostgreSQL，MySQL，Maria DB などを指す．本書では，主に PostgreSQL，補助的に MySQL について述べる．通常，OS[8]の力を借りて，その上で稼働するミドルウェアである．基本的に，次に示す 3 つの機能を有する．

[1] DataBase
[2] 記号系
[3] 実世界のデータ化する過程
[4] 取得，挿入，更新など
[5] 関係モデル，RDB
[6] 関係データベース
[7] DataBase Management System
[8] Operating System

- メタデータ管理機能
- 質問処理機能
- トランザクション管理機能

メタデータ管理機能とは,「データのデータ」としてのメタデータを管理する．データベースのデータ構造などを管理するデータでありそのデータの管理を行う．ユーザと DBMS そのものに対して用いられる．ユーザに対して中にどのようなデータがどのように格納されているかを示す．DBMS に対しては，自分が管理しているデータの種類，サイズ，インデックス，アクセス件など，基本的な情報提供に不可欠なものである．

質問処理機能とは，文字通り，ユーザやアプリケーションが要求する「質問」に対し，処理を行う．特にリレーショナル・データベースでは，SQL に代表される，質問言語を用いる．

トランザクションとは，DBMS に対するアプリケーションレベルの仕事の単位である．障害時回復，同時実行制御の 2 つの機能を有しており，一貫性の保持や，複数人の同時実行を行う．

2.3 ファイルとの相違

データベースに比較して，プログラムごとにデータをファイル保存する手法がある．C 言語ならば，FILE 構造体を用いて，fprintf()，fscanf() 関数などで，ファイルに対して入出力を行える．しかし，この場合だと，データはプログラムに従属する形になり，無秩序になりがちである．また，データ形式の違いから，同じデータを複数保存したり，プログラムによるアクセスのタイミングから矛盾が発生することがある．また，近年は，Microsoft Excel などの高機能な表計算ソフトの登場により，表計算ソフトをデータ保存に用いている場合も有る．この場合，一見，完璧に一元管理が可能なように見えるが，異なるファイルに対しては，同様に矛盾が発生しやすく，一元管理がなされないことが多い．

データベースに関しては，DBMS を用いることで，決まった形にデータを保存することが可能である．また，全てのプログラムから 1 つのデータベースにアクセスすることで，重複や矛盾を排除し，データを一元管理することができる．データは共有資源にすることに意味があり，様々なプログラムからのアクセスに，対応させることは必須である．

第3章　リレーショナルデータモデル

3.1　構造記述

リレーションとはなんであるか．実世界のデータ構造記述の拠り所であり，リレーションは，様々な概念で構成される．SQL においては，テーブルが相当する．リレーションの定義について，表 3.1 表 3.2 を用いて説明する．

表 3.1: リレーションのサンプル

数字	英字	平仮名
1	a	あ
1	a	い
1	b	あ
1	b	い
1	c	あ
1	c	い
2	a	あ
2	a	い
2	b	あ
2	b	い
2	c	あ
2	c	い

表 3.2: 友人

Name	Age
太郎	25
一郎	30
花子	26
桃子	22

3.1.1　ドメイン

ドメインの別名は，定義域と言われる．プログラミング言語には，INTEGER，CHARACTER，FLOAT などのデータ型が存在する．それを元に，定義された集合である．例として，$D_1 = INTEGER$ ，$D_2 = VARCHAR(1)$ などと定義できる．

3.1.2　直積

次に，ドメインからなる直積を求める．ドメイン

$$D_1 = \{x|x = 1, 2\}, D_2 = \{c|c = a, b, c\}, D_3 = \{s|s = \text{あ}, \text{い}\} \tag{3.1}$$

とする．このとき，$D_1 \times D_2 \times D_3$ を求めると，次のような 12 個の要素が得られる．

$$\{(1, a, \text{あ}), (1, a, \text{い}), (1, b, \text{あ}), ..., (2, c, \text{あ}), (2, c, \text{い})\} \tag{3.2}$$

この直積集合の各要素を組[1]という．

そして，ドメイン D_1，D_2，D_3 を用いた部分集合，リレーション

$$R \subseteq [\{D_1 \times D_2 \times D_3\} = \{(1, a, \text{あ}), (1, a, \text{い}), ..., (2, c, \text{い})\}] \tag{3.3}$$

[1] タプル，タップル，tuple

を定義できる．また，リレーションの組の総数を R の基数[2]，定義されているドメインの個数を次数[3]をという．この場合は，$cardinality(R) = 12$，$degree(R) = 3$ である．ドメインに対する名前を属性名[4]という．ドメイン関数

$$R \subseteq \{D_i = dom(A_i)\}(i = 1, 2, ..., n) \tag{3.4}$$

と定義できる．表 3.1 でいうと，式 (3.3) より，

$$R \subseteq \{D_1 \times D_2 \times D_3 = dom(A_1) \times dom(A_2) \times dom(A_3) = dom(\text{数字}) \times dom(\text{英字}) \times dom(\text{平仮名})\} \tag{3.5}$$

となる．

3.1.3 行と列

リレーション R を用いて，テーブル[5](表 3.1) を表すことができる．テーブルの横を行[6]，縦を列[7]という．ここで，行の順番には何の情報もない．例として，$(2, b, \text{い}), (2, c, \text{あ}), (1, a, \text{あ})$ となっていても何の問題もない．だが，列の順番は，直積の演算の結果であるため意味があり，この場合，$(1, a, \text{あ})$ が，$(\text{あ}, 1, a)$ になるのは，間違いである．

また，行数は特に規定が無い限り，無制限に入れられる．リレーションの中身は人間が手動で検索するものでは無く，積極的に閲覧しないものである．検索を行う場合は，必ず質問言語を用いて必要な結果を得る．ここが，MS Excel などの表計算ソフト等との大きな違いである．

だが，リレーションにおいて，同じ行が存在してはならない．リレーション上では，属性の各要素の組み合わせからなる行は，重複しない組み合わせで有る必要がある．例として，$(1, a, \text{あ}), (1, b, \text{い}), (1, a, \text{あ})$ となってはならず，$(1, a, \text{あ})$ は唯一の $(1, a, \text{あ})$ である．2 行目以上のの同じ行があった場合は，どちらかが間違っているため，リレーションにおいける整合性はなくなってしまう．

3.1.4 リレーションスキーマ

リレーションが持つ，普遍的な構造や意味を抽出したものであり，時間的に常に不変である．例として，表 3.2 でいうと，太郎，一郎，花子，桃子が友人であるのは，この時だけで，翌日には，友人ではなくなる可能性や，新たな友人ができる可能性もある．つまり，リレーションとは時間とともに変化するものである．しかし，このリレーションが友人の Name と Age を表していることは，変化しない．この，リレーションの時間に関係なく，不変な性質をリレーションスキーマという．

それに対して，一般に時間の移り変わり共に変化するリレーションそのものをリレーションスキーマのインスタンスと呼ぶ．この例では，リレーションスキーマ友人 (Name,Age) が定義される．よって，表 3.2 はリレーション友人でのある時刻での友人のインスタンスである．

従って，リレーションスキーマに対して成立しないと行けない事象は全てのインスタンスに成立する．また，その逆も言える．

3.2 意味記述

意味とは，リレーションという形だけでは捕えられない実世界の制約のことだ．例として，表 3.3 の場合，社員の平均給与は 30 以上であるという制約は，構造では捕えられない．また，キー制約を用いることで，リレーションを常に一貫性[8]のある状態に保つことが出来る．

3.2.1 キー制約

RDB では，データベースの検索や更新に，どの組にアクセスしているのかを一意に指定しなければならない．例として，表 3.3 の場合，1508 の山田二郎の給与を 10 減らしたくても，名前や所属では 2 人の山田二郎に共通で当てはまってしまう．そのため，正常な操作を行うことができない．

---

[2]濃度,cardinality
[3]degree
[4]attribute name，属性
[5]表,table
[6]row．タプルに対応する．
[7]カラム，column．属性に対応する．
[8]実世界と矛盾のない状態

表 3.3: リレーション社員

社員番号	社員名	給与	所属	郵便番号
650	鈴木一郎	50	K55	805-96
1508	山田二郎	40	K41	814-03
231	田中三郎	60	K41	802-01
2034	佐藤四郎	40	K55	819-98
3019	山田二郎	50	K41	825-43
401	堂本花子	60	K55	814-03

表 3.4: リレーション納品

商品番号	顧客番号	納品数量
G1	C1	3
G1	G2	10
G2	G2	5
G2	G3	10

スーパーキー

一般に，いかなるインスタンスにおいても，2 つ以上の組が空値[9]となる場合を除いて，同一の属性値を持つことがないような，属性あるいは属性の集合である．主キーや候補キーとの違いは，一意に識別できる全ての組み合わせを示す点である．つまり，候補キーに属性をつけて冗長にしたものは，候補キーではないが，スーパーキーではある．

表 3.3 の場合，スーパーキーは以下のとおりとなる．

$$\{ \text{社員番号} \}, \{ \text{社員番号}, \text{社員名} \}, \{ \text{社員番号}, \text{給与} \}, ..., \{ \text{社員番号}, \text{社員名}, \text{給与} \}, \tag{3.6}$$

$$\{ \text{社員番号}, \text{社員名}, \text{所属} \}, ..., \{ \text{社員番号}, \text{社員名}, \text{給与}, \text{所属}, \text{郵便番号} \} \tag{3.7}$$

候補キー

全体集合のうち，組の一意識別能力を持つ部分集合の極小組を候補キー[10]という．候補キーは極小であるため，属性を 1 つでも取り除くと組の一意識別能力を失う．

例として表 3.3 を考えると，{ 社員番号 } が候補キーとなる．また，別の例として，表 3.4 を考える．この場合は，{ 商品番号，顧客番号 } が候補キーとなる．同じ商品を異なる顧客に納品する場合や，同じ顧客に異なる商品を納品することが考えられるからである．

[9] そらね，Null Value
[10] candidate key