第2章

データ

世の中の複雑さを反映して，我々の扱わなければならないデータも多種多様であり，しかも多くの場合，さまざまな形でからみあっている．その共通した扱いのためには，まずデータを単純な数値の並びであるデータベクトルの集まりに分解したうえで，それらをふたたび組み合わせて表現するとよい．そうすれば，相互関係がはっきりし，モデル化もしやすくなる．本章では，まずデータベクトルとそのさまざまな属性を説明した後，それらを組み合わせる代表的な方法である関係形式と配列形式について述べる．もちろん，ここで議論するデータとは，何らかのデータ解析を行うことによってモデルを創り出すことを目的とするものであり，より一般的に言えば，**データ** (data) という，**情報** (information) の具体的な姿からの **知恵** (intelligence) の抽出を目的としたものである．

2.1 データベクトル

どんなに複雑なデータでも，解きほぐしていくと単純な数値の並び，つまり**データベクトル** (data vector) の集まりとなる．このような分解の背後には，意識するしないにかかわらず，常に**変量** (variate) の概念があり，各変量が実際に取った値の並びがデータベクトルである．変量とは，数学で言う**変数** (variable) と表面的には同じであるが，変量には その意味まで考える という

暗黙の了解が含まれていることが多い．好き・嫌いのように数値ではない値を取る変量も考えられるが，その場合でも，以下で述べるようにデータベクトルの一つの属性として整数をそれぞれの値に対応させるコード属性を導入すれば，このような値の並びも数値の並び，つまりデータベクトルで表現できる．いずれにしろ，データベクトルは一つの変量の取る値であり，同じ性質，厳密に言えば同じ**属性** (attribute) を共有する数値の並びでなければならない．言い換えれば，データベクトルに並んだ値の間には**均質性** (homogeneity) が確保されている必要がある．ここで言う均質性とは，第 1 章での均質性よりもより根元的な性質であり，以下で述べる属性がすべて共通であることを意味する．もし均質でなければ，そのデータベクトルをさらに細分化して均質性を確保する必要がある．DandD では次のような形で表現したデータベクトルをまとめて要素 `<DataBody>` に置くことにより，データ本体 (data body) を他から分離している．

```
<DataVector Id = ID 名 属性名 = 属性値　属性名 = 属性値 …
            属性名 = 属性値 />
    数値の並び
</DataVector>
```

ここで，`Id`（個体識別情報, identification）も属性の一つである．現在 DandD の実装手段として用いている XML は，もともと組織立った文章を作成するための言語であるので，必ずしも各要素に `Id` 属性を与える必要はないが，この属性によって各要素を識別することができる． つまり DandD のように複雑に絡み合ったデータを組織的に記述し，さまざまな形で利用するためには，原則として `Id` 属性が必要となる．なお，本体 (body) が空であれば，その値の並びは DandD インスタンスから外部への参照によって得られるはずである．詳細は 2.6 節を参照.

以下では DandD ルールに従って構成された XML ファイルのことを DandD インスタンスとよぶが，国際化を念頭に，このファイルで用いる文字コードは Ascii コードだけであっても UTF-16 を用いる必要がある．また，引用符号の中以外の改行，復帰などの記号は無視されるので，改行したとき，それ

2002-11-12 追加

が区切りを意味する場合には，空白を明示的に入れる必要がある． 2002-12-12 追加

Id 属性と関連して LongName 属性についても触れておこう．ID 名はもっぱら要素を識別するための名前であり，識別できる限りなるべく短い名前のほうが便利である．しかも，アルファベットだけを用いた名前などのほうが後の操作はしやすい．しかし，一方，人間にとってはある程度意味がわかるような説明的な名前のほうが望ましい．このようなとき LongName 属性で説明的な名前をつけたほうがよい．いわば， ID 名は扱いが便利でわかりやすい通称， LongName は正式な名前である．以下の例などでは，ID 名としても分かりやすい名前を用いるが，必ずしもそうする必要はない．識別できれば十分である．しかし，世界的な視野で考えた場合には，日本語の名前では理解できないので他の言語での名前も併記する必要がある． ID 名としてはアルファベットで構成された名前に限定するとしても，LongName としては，Appendix に置いたタグ Name を利用して，それぞれの言語に応じた名前を用意し，それらの ID の並びを引用する必要がある．詳細に関しては章末で述べる．

なお， XML の規約では，属性値は必ず 2 重引用符 " で囲まなければならないが，以下の説明ではこれを省略していることもあるので注意されたい．

2.1.1 値

データベクトルの要素，つまりそこに並ぶ値としては数値だけを考えるが，それでも整数，実数などさまざまな数値が考えられ，その表記法もさまざまである．しかし， DandD のデータベクトルには §2.1.2 で述べるような，さまざまな属性が付随しているので，次のような 10 進浮動小数点表示だけに限っても問題はない．

```
[+-]nnn.mmm[ekkk]
```

この形式的な定義は， nnn，mmm が 0 〜 9 の文字 (digit) の並び，それを小数点で区切ったあとに必要に応じて指数部 (exponent)，つまり， e のあとに正負の 10 進桁数 kkk を置く慣用に従っている．たとえば， 2.5e3 は 2500 のことであり -30.2e-2 は -0.302 のことである．

また，特殊な数値として欠損 (Not Available) を表す NA を許すことが必要である．しばしば，このような特別な値を導入しないために，99999，あるいはありえないと考えられる数値を用いたり，固定長の記録形式の場合には，単に空白を置くことにより欠損であることを示したりするが，これは混乱のもとである．また，後で述べるように，欠損の原因をきちんと記述することも後の解析には重要となるが，その属性との対応をはっきりさせるためにも，欠損であることの一定の表示形式の導入は欠かせない．

ファイルにおける数値の並べ方としては，大きく分けて**固定欄形式** (fixed column format) と**自由欄形式** (free column format) がある．固定欄形式は，1 行を 1 文字単位の欄 (field) に分け，何番目の欄から何番目の欄までが一つの数値，つまり一つの**記録** (record) を表すというように，あらかじめ区切りを定めていく方法である．この形式は特別な区切り記号が必要ないため，スペースの節約になり，またこの形式の固定長レコードから成るファイルへのアクセスは能率的でもあるため，大量の高速な処理を必要とする場面では好んで用いられる．しかし反面，フレキシブルでないことも確かである．また，人間には読みにくく，勘違いを起こしやすい．これに対し，自由欄形式はあらかじめ欄を区分したりせず，区切り記号 (separator) を導入して記録同士を区分するものである．可変長レコードとみなせないこともないが，可変長レコードの場合は各記録の先頭にその記録の占める欄数が書かれているのに対し，自由欄形式の場合はそのような情報は特になく，区切り記号だけで記録を区分する．**CSV** (Comma Separated Values) の名前でよく用いられる記録形式も区切り記号をコンマに固定した自由欄形式にほかならない．もちろん，自由欄形式の問題点は，区切り記号として用いた記号をレコードの一部としてそのまま用いることができないことであるが，ここではデータベクトルに並ぶ値を数値だけに限っているので，空白やタブなどを区切り記号としてだけの目的に用いることができる．

2.1.2 属性

情報科学では，伝統的にコンピュータ内部での扱いが異なるという理由から，整数，浮動小数，複素数，文字，文字列などの基本的な「型」を区別してきた．最近のオブジェクト指向言語では，このような区別をなるべく意識しないですませる方向，つまり，文脈で型を判断する方向に進んできている．それでも内部的には区別しており，また場合によっては，型を明示的に指定しなければ混乱してしまう状況もしばしばある．その意味ではいまだにこのようなデータの型も重要であることに変わりない．しかし，少しデータの意味するところにまで踏み込むと，上記のような表面的な扱いの違いによる型よりも，もっと広い範囲にわたる「属性」が必要になる．以降，そのようなデータベクトルのおもな属性を紹介していくことにする．

これらの属性は，それぞれのデータベクトルに対して固有なものであり，基本的に他のデータベクトルには依存せずに定まる．複数のデータベクトルが構成する属性は，§2.2 で説明するように，要素 `<Data>` で特別な構造を定義することによって表現する．例外は変域属性で，変域を表すのに他のデータベクトルを参照する．

まず，もっとも簡単で基本的な属性から始めよう．それは，精度と確度である．たとえば，`3.0e5` は 300000 のことであるが，この二つの表現はしばしば意味が異なる．そこには重要な属性として**精度** (precision) と**確度** (accuracy) がからんでくる． 精度とは，数値を数字の列で表現したときの精確さのことであり，たとえば，もとが整数なら `3.0` のように表されていても精度は小数点以上 1 桁である．一方，確度とはその値を測定する段階，調べる段階，さらには処理の段階で確保できる正確さのことである．この二つの属性はよく似ていて区別されないことも多いが，独立な属性であるので区別する必要がある．

精度

精度 (precision) は，主に計算機の表現できる 10 進表現の限界や，扱えるヴォリュームの制約などが原因で生ずる，本来の数値との乖離を示すための属

性であり，この属性によって示される有効桁数以上は無意味であることを示している．実際的な扱いとしては，切り下げ，切り上げ，四捨五入など，さまざまな扱い方があるが，計算機での内部表現に起因するものは切り上げとして扱うとよい．この精度という属性を常に意識するかぎり，実は実数と整数の区別も必要ないことになる．整数は `Precision = 0` の数値だからである．

`Precision = n`

`n > 0`	小数点以下の有効桁数
`n = 0`	整数を意味する
`n < 0`	小数点以上の有効桁数

例：

	Precision
3000.01	2
1	0
3200	-2
1.3e10	-9

確度

確度 (accuracy) は実際に現れた数字の示す「精度」とは，一応無関係である．この値より小さな絶対値の違いは意味がないことを示す属性である．実験器具の精度，収集段階での値の切り捨て，丸めなどによって生ずる精度限界である．つまり，精度が数値の表現上の精確さであるのに対し，確度は表現とは無関係に，データそれ自身に内在する正確さの限界である．この確度をさらに，**偏り** (bias) と**ばらつき** (deviation) に分けることもできる（本シリーズの『医学データ』を参照）が，この属性ではそれらをまとめた，測定時にあらかじめわかる絶対的な精度といったものを表現する．

`Accuracy =` 数値

さて，精度や確度の次に必要となるのは，取りうる値とその背後にある変量に関する記述である．

変域

データベクトルの値として出現可能な値の範囲をこの**変域** (domain) 属性で示す．それだけでなく，異なるデータベクトルに同一の変域属性を与えることによって，同一の変量が対応していることを示す役割も果たす．たとえば，このデータベクトルが調査などで回答を寄せた人を識別するベクトルの場合，回答しなかった人も含めた調査対象となったすべての人のリスト，あるいはその人数がこの属性となる．また，§2.3 で説明する関係形式を構成するデータベクトルの変域属性と配列形式の軸に対応するデータベクトルの変域属性を共通にすることによって，この二つの形式の間の関係を明示するのに利用できる．関係形式の間で共通な変域属性をもつデータベクトルは外部キー (foreign keys) となる．この変域属性のついたデータベクトルすべてでその関係形式の候補キー (candidate keys) を構成する．つまり，記録を一意的に定める．一つの関係形式にこの属性を持つデータベクトルが一つしかなければ，それは他の関係形式に対して外部キー (foreign key) となりうる．

2002-07-11 2003-06-16 追加 2003-07-25 修正

`Domain =` とりうる値の集合を規定するデータベクトルの `ID`

とりうる値が有限個ならば，`ID` で引用されるデータベクトルの本体としてその可能な値を与えてもよいが，そうでなければ，データベクトルの内容は空にして，`Explanation` 属性によって説明する．このように文章による説明しかできない典型的な例としては，正の値しか取らない変量，0 と 1 の間しか取らない変量などの場合がある．このような記述はモデルを絞り込むときの大きな助けとなる．また，場合によっては，データベクトルに現れた値が制約を持っていることもある．たとえば，比率なら，正で和が 1 という制約を持ち，いくつかの項目から答えを一つ選ぶ調査ならば，度数の和は解答者数に一致しなければならない．このような制約は多様すぎて，現在のところ文章によって補足するしかない．

欠損

実際のデータを扱うとき，しばしば遭遇するのが**欠損値** (missing value) である．これをデータベクトルの値としては NA で統一的に表現するとしても，値が欠損する原因にはさまざまなものがある．一つは最初から値が得られなかったいわゆる**欠測値** (unobservable) である．この欠測値にも何種類かあり，もともと値がありえない条件での測定なり調査なりであった場合の欠測値 (never happen)，測定器の測定限界を越えたための欠測値 (out of scale)，たまたま観測し忘れたり，記入し忘れたことによる欠測値 (happening)，調査における回答拒否による欠測値 (refused to answer) などさまざまである．

値が得られないことには変わりがないので，これらの区別はどうでもよいようにも思えるが，実は，後で欠損値の補間を行う必要が生じたときや，モデル構築の段階で，これらの区別が重要になる．たとえば，もともと値がありえなければ補間することに意味はなく，測定限界を越えただけならば，補間せずに範囲を越えた場合として別扱いするモデルのほうが適切である．

また，欠測以外にも欠損の生ずる場合がある．データの不適切な取扱いやミスで欠損が生じたり，以下で述べるような不適切な回答 (invalid answer)，明らかに誤りの回答や値 (sic value) を欠損として扱うことにした場合などに生ずる．このような「欠損」を記述するためには，データベクトルの各要素ごとに欠損の理由が異なることもあるので，上記のような欠損のタイプ (missing type) をデータベクトルの要素に現れた欠損値それぞれに対応して記述する必要がある．

欠損値の扱い追加; 2002-07-22

`Missing =` 欠損値の添字番号からなるデータベクトルの ID
`MissingType =` 対応する欠損のタイプ番号からなるデータベクトルの ID

具体的な欠損理由つまり各欠損タイプの説明は，MissingType に与えられたデータベクトルのコード属性で記述する．理由としては

```
"unobservable", "never happen", "out of scale", "happening",
"refused to answer", "invalid answer", "sic value" "not recorded"
"multiple unrecorded elements"
```

”not recorded” 追加, 2002-07-01, 2003-06-16

などが考えられる． すべて同じタイプの欠損値ならば，欠損値を NA (外部データの場合は 同じ要素数の NULL でもよく， DandD サーバは NA と見なす）で表し，Missing を省略し， MissingType だけを与えてもよい． そうでなければ，この 2 つの属性ともに与える必要があるが，その場合は欠損値の表現は必ずしも NA である必要はない．これは，コーディングの必要な場合も同様で，Missing を省略した場合の欠損値は 2 文字 NA (外部データの場合は NULL でもよい）で表す.

NULL 追加; 2002-09-17
multiple unrecorded elements 追加; 2002-07-04

ただし，場合によっては，ある一定期間観測なり，記録が失敗して欠損していることはわかっているものの，何記録欠損しているか不明のこともある．この場合は， 該当する位置に 1 つの欠損値を置き，タイプを "multiple unrecorded elements" としておく.

無効

すでに欠損という属性の説明で触れたことであるが，値はあるものの，それが信用できないということもある．信用できないならば，その値は捨てて NA にしてしまえばよいとも考えられる．しかし，まったく信用できないわけでなければ，後の解析なりモデル化の段階で何らかの形で活用できる可能性も残っており，あながち捨て去ることが賢明とは限らない．このような場合，**無効** (invalid) 属性を用いることがある．欠損と同様，無効にもさまざまな種類がある．すでに現れた，不適切な回答，明らかに誤りの回答や値以外にも，計測器が狂っていたため信用できない (inaccurate) 場合もある．これは，確度がわからないため，おかしな値の可能性があるという場合も含む.

Invalid = 無効と考えられる値の添字番号からなるデータベクトルの ID
InvalidType = 対応する無効のタイプ番号からなるデータベクトルの ID

欠損と同様に，無効の理由は InvalidType のコード属性で説明を与える．

名前変更 InvalidCode を InvalidType に; 2002-07-01

切断

計測器の設計によっては，計測範囲を越えた場合にその範囲の上限あるいは下限のみを表示・記録するものがある．何らかの反応が飽和状態の場合に

もこのようなことが起こる．このような場合，データベクトルに並んだ値だけでは，果たしてその値がちょうど下限あるいは上限だったのか，それを越えたためにそう記録されたのか判断できない．これを明示するのがこの**切断** (truncation) 属性である．

`Truncation` = 切断が起きたと考えられる値の添字番号から成るデータベクトルの `ID`

`TruncationType` = 対応する切断タイプ番号から成るデータベクトルの `ID`

`TruncationLeft` = 下限

`TruncationRight` = 上限

TruncationReason を TruncationType に変更; 2002-07-01

打切り

故障の計測実験など，すべての機器が故障するまで待っているわけにはいかない場合，あらかじめ定めた時間の経過後に観測を中止することになる．また，経済的な理由で早期に観測を中止することもある．このような**打切り** (censoring) は**時間打切り** (time censoring)，あるいは**タイプ 1 のセンサリング** (type 1 censoring) と呼ばれる． データベクトルが，§2.3.4 で扱うような，事象が生起した時点のデータ，つまり点過程データの場合には，切断属性で記述することもできるが，切断とは理由が異なるのでこの属性で明示する．また，ベクトルが起こった故障の回数などの場合には切断属性では記述できない．いずれにしてもこの属性で統一的に記述する．

`CensoringTime` = 打ち切るまでの経過時間

この属性を持つデータベクトルの値は同一の打切り時間を共有している必要がある．打切り時間が異なれば，データベクトルの要素の均質性を確保する観点からも，打切り時間ごとにデータベクトルを細分する必要がある．

もう一つのタイプの「打切り」属性が個数打切りである．あらかじめ定めた回数以上故障などが起こったらそこで実験なり調査なりを中止する，いわゆる**タイプ 2 のセンサリング** (type 2 censoring) である．この場合のデータ

ベクトルは，その事象の生起時間の系列や，調査個体を識別する番号の系列が典型である．タイプ 1 と対照的に，データベクトルが故障の回数などの場合には切断属性で記述することもできるが，時間打切りの場合と同様，この属性で統一的に記述する．

CensoringNumber = 打ち切るまでの観測個数

この打切り属性に与えられた数値は後述の Length 属性と一致するのが普通であるが，欠損値などがあれば大きくなることもある．

以上は意図した打切りであるが，意図しない打切りもある．患者が通院しなくなったりしたため，それ以降記録が取れなくなったような場合や，データを採取している途中で，意図に反して観測を打ち切らざるを得なくなった場合がその例である．このような打切りは**ランダムセンサリング** (random censoring) と呼ばれ，特に寿命データの場合には重要な属性である．多くの場合，データベクトルの要素ごとに打切りが起こったかどうかが変わるので，Missing と同じような記述をする．

RandomCensoring=ランダムセンサリングが起きた値の添字番号からなるデータベクトルの ID

RandomCensoringType=対応するランダムセンサリングのタイプ番号からなるデータベクトルの ID

具体的なランダムセンサリングの理由は, RandomCensoringType のコード属性で与える．たとえば，手術後の生存時間の場合，RandomCensoring には，死亡によって観測が打ち切られたら 1, 追跡不能なら 2, データをまとめるため現時点で打ち切ったのなら 3 としたベクトルの ID を与え，そのコード属性には ”死亡”, ”追跡不能”, ”生存中” のコードを与えればよい．ただし，この例の場合，ある程度以上生存すれば，完治したものとして扱うことが多いので，その場合は CensoringTime も与える必要がある．

ランダムセンサリングを追加, 2002-07-01

変換

注目する変量の値そのものではなく，簡単な変換が施された値しか記録されないこともある．たとえば，デシベルに変換されて観測された音量や，常

に値が大小順に整列 (sort) されてしまう場合などである．また，データを扱う段階や解析の段階で便利のため変換を施す場合もある．このような**変換** (transform) 属性は多くの場合，後述の単位属性に反映されるが，それではわかりにくいことも多いので，この属性も必要である．

`Transform =` 施された変換の記述の `ID` の並び

ここで，複数の ID が与えられるようになっているのは，多言語対応のためである．変換については §4.4 で述べる．

コード

好き・嫌いのように，もともと数値でない値，つまり分類指標の値は数値とは異なる扱いが必要である．このような値は**カテゴリカルな値** (categorical value) あるいは**類別値** (classification index) と呼ばれることも多い． これらをそのまま “好き”，“嫌い” といった文字列として扱うことも考えられ，実際，そう扱っているソフトウェアも多いが，データの流れの途中で数値と文字の混在は問題を引き起こす．それだけではなく，文字列での表現には多様性があり，同じ `"好き"` を `"すき"` と表したいこともあるだろうし，`"prefer"` といった英語で表したいこともある．さらに，もっと説明的な長い表現が必要なとき，コンパクトな表現が望ましいときなどもある．このような多様な要求に応えるためには，やはり一度，数値（通常は自然数）に直し，その属性として，文字列表現との対応を記述しておくのがよい．この属性が**コード** (code) 属性（マッピング，コーディング (coding)）である．通常，各分類指標の取りうる値のことを**水準** (level) と呼ぶ．

`Code = Appendix` に置かれた `Code` タグで囲まれた（自然数に対応する）水準の並びの `ID` の並び

なお，2 重引用符で囲んだ文字列の並びである「水準の並び」は要素 `<Appendix>` に置かれ，それを引用する形をとっているので，一つの「水準の並び」を複数の `<DataVector>` で共有できる．複数の `ID` を与えられるようにしてあるのは，多言語対応だけでなく，短縮したコードなど，一つのデータベクトル

に対して，さまざまな形式のコードも与えられるようにという配慮からである． ID の間は半角スペースで区切る．なお， 現段階では Code の実体は外部データとはしない．また，分類指標としての意味がなく，後述の DataType 属性が Category あるいは OrderedCategory で無くても，数字以外の値を扱う場合にはこの属性を用いてマッピングする必要がある．DatabaseCode も参照．

2002-08-05, 2003-07-25 修正

要素数

データベクトルの要素数，つまりデータベクトルに並んだ数値（記録）の個数．これはデータベクトルの要素数を数えればわかるはずの値であるが，何らかの事故でデータが失われた恐れのあるときのチェックとしても役立つ属性であるし，大量のデータの場合にあらかじめ作業領域を確保したり，表示形式を変えたりするためにも大いに役立つ属性である．しかし，必須とはしない． データベースで更新があったりしたときの障害とならないようにである．

2002-07-12

Length = そのデータベクトルの要素数

ここまで説明してきた属性は客観的に定まるものであるが，後のデータの処理や解析，モデル化の段階で重要になる属性には，意味 (semantics) に関わるものもある．その一例が統計学の教科書の最初でよく強調される**比尺度** (ratio scale)，**間隔尺度** (interval scale) ，**順序尺度** (ordinal scale) ，**名義尺度** (nominal scale) の区別である．

比尺度は値の比それ自体に意味のある尺度であり，数学で言えば単位元を 1 とする乗除算に意味がある尺度である．一方，間隔尺度は値の差自体に意味のある尺度である．つまり，単位元を 0 とする加減算に意味のある尺度である．たとえば，温度は摂氏で表すこともできるし，華氏で表すこともできる．しかし，いずれで表すにしても，たとえば，倍の温度は基準となる温度と無関係に物理的な意味を持つわけではない．10 ℃の 2 倍の温度 20 ℃と 50 ℃の 2 倍の 100 ℃では，同じ 2 倍と言ってもまったく意味が異なる．10 ℃の水を 20 ℃にするのに必要なエネルギーは差の 10 ℃に比例した量であり，

50 ℃の水を 100 ℃にするのに必要なエネルギーは差の 50 ℃に比例した量である．この意味で温度は比尺度ではないが，間隔尺度である．もちろん，絶対温度を用いれば比尺度にもなる．一方，重さは比尺度であり間隔尺度でもある．比も差もそれ自身で意味を持つからである．

順序尺度は大小関係だけに意味がある尺度である．たとえば大・中・小 を $3, 2, 1$ のように番号をつけて表しただけでは，これらの数字に関しては比が 1 より大きいか，差 が 0 より大きいかだけしか意味がない．名義尺度は，大小関係にも意味がない尺度である．好き・嫌いがその例である．

しかし，以上のような区別は「意味」というどちらかと言うと主観的な側面に依存しており，見方によっては，たとえば，温度でも倍の温度がまったく意味を失うわけではなく，夏の平均気温が冬の平均気温の倍であるといったように，日常生活のある側面では，それなりの意味を持つことも多い．

また，これ以外にも**計量値** (measurement) と**計数値** (count)，**連続値** (continuous) と**離散値** (discrete)，**量的変量** (quantity) と**質的変量** (quality) の区別などもよく用いられ，直観的には理解しやすいが，後のデータ解析やモデル化にはこれだけでは不十分なことが多いだけでなく，システマティックではない．ここでは，これらをもう少しシステマティックで，しかもあまり主観に片寄らないよう,「単位」属性と「型」属性に分けて表現することにする．

単位

この**単位** (unit) 属性はしばしば省略されてしまい，データを収集した人や，その分野に精通した人には当たり前のことであっても，この属性をはっきりさせておかないと，他の人にはうまく通じないために，思わぬ誤解を生む原因になることが多い．また CGS 系のように，よく知られた単位の場合には単位を記述するだけで十分であろうが，場合によっては十分な説明や参照文献が必要になることもある．特に，時間に関しては**日付**や**時刻**と第 1 章で述べた**通日**や**通時**との区別が重要である．日付や時刻を表現するためには，後で述べるように複数のデータベクトルを組み合わせた一つの基数系として表現することになるので，単位は単に「年」,「月」,「日」,「時」,「分」などとし

ておけばよいが，通日や通時の場合には，単位だけでなく，その原点も明示する必要がある．

また**年齢** (age) も特殊である．通日と考えることもできるが，それぞれの個体で原点が異なり，0 から始まりはっきりした上限はなく，常識的な上限の目安がある点で通日とは性格が異なる．

また，ある人間の集団を追跡調査するいわゆる**コホート** (cohort) 解析を行うには生年月日のデータでなければならないが，単位を「歳」として年齢を表現した場合は，このような解析は念頭にないことを暗黙のうちに意味していることになる．このような単位に関する説明は，形式的な記述によることはなかなか困難であり，DandD では `UnitDefinition` 属性として必要に応じて文章で説明することにしている．

また，単位には，先に述べた，比尺度であるか間隔尺度であるかの区別も付随する．特に記述のない限り，比尺度であり間隔尺度であるとみなすのが自然であろう．DandD では，そのために `Scale` 属性が用意されている．

`Unit` = 単位を表す文字列の `ID` の並び
`UnitDefinition` = それぞれの単位に関する説明文の `ID` の並び
`Scale` = `"Ratio"`, `"Interval"`, `"RatioInterval"` のいずれか

`"Ratio"` なら比尺度，`"Interval"` なら間隔尺度，`"RatioInterval"` なら比尺度であり間隔尺度であることを示す．ここで，`Unit`, `UnitDefinition` に複数の ID が与えられるようになっているのは，多言語対応のためである．例えば，1 尺， 1 合，1 間など日本（語）特有の単位もある．同一言語で，複数の単位を併記したい場合などについては，詳細については 2.7 節を参照．

型

以下に解説するいくつかの**型** (data type) 属性は，実際のデータ解析やモデル化で必要となる区別に関わるものである．もちろん，生年月日や年齢のように単位属性で十分説明しきれている場合にはこの型属性は不要である．

型属性は

`DataType` = 型

の形で与える．現在のところ「型」は以下の 12 種類のうちの一つ．

まず計量値については，取りうる値の範囲が限られている，次のような場合を型で区別する必要がある．これらは，後で述べる型 Category と同じようにも見えるが，はじめから分類指標として観測された値ではない点が異なる．もちろん，単に正の実数だけを取る変量，ある範囲の値しか取りえない計量値もあるが，これらは特に「型」で区別する必要はない．変域属性を用いれば十分である．したがって，計量値の場合は型属性を省略できることも多い．

次の計量値の型のうち最初の 2 つの型は，計量値のうち観測の前に値が定まる「条件値」に属する型であり，残りは 2 つは観測して初めて値の定まる「結果値」に属する型である．計量値の型を省略した場合は型「点数」,「区間」のいずれでもない「結果値」であると考える．

条件値，結果値の記述の追加; 2002-07-01

1. **特定値** (SpecifiedValues)
   あらかじめ定められた高々可算個の値しか取らない変量であり，しかもあらかじめ設定された値であることを示す属性である．コード属性を与えれば，一見カテゴリカルデータに類似しているが，あらかじめ設定された値であるかどうかで異なる．取りうる値は変域 (Domain) 属性で記述する．
2. **等間隔** (EquallySpacedValues)
   ある一定間隔の値しか取らないような変量であることを示す型で．特定値の特別な場合である．自然数しか取らない場合など，等間隔でしか測定できない場合などがこの例である．取りうる値が有限個の場合は型 SpecifiedValues で記述してもよいが，等間隔であることを強調するとき，この属性を用いる．取りうる値は特定値の場合と同様に変域 (Domain) 属性で記述する．
3. **点数** (Score)
   100 点満点の点数，あるいは A, B, C による成績評価などの場合である．7. の順序カテゴリーと類似しているが，単に点数という評価を表す

”特定値の特殊な場合” を削除; 2002-07-04

値であって，これによってただちに類別するわけではないことを強調するためにはこの型を用いるとよい．100 点満点の点数の場合には，0 〜 100 点のように可能な点数の範囲は，特定値の場合と同様に，変域属性を利用して記述する．さらに確度属性で点数の最小単位を明示してもよい．なお，A, B, C による評価のような場合には，コード属性によってコーディングする必要があるが，順序カテゴリーと同じように，この属性に与えられた順序が評価の昇順を示すものとする．

4. **区間** (IntervalClass)
   取る値がそれぞれある限られた区間あるいは部分集合に対応するものである．この型は，得られたデータの背景に連続値を取る変量があるものの，その値が直接観測できず，あらかじめ定められたいくつかの区間のうちの一つの区間に入る値であることしかわからない場合である．たとえば，釣った魚の年齢や樹齢などは，えらの成長や年輪からある程度わかるものの，1 年単位でしか判定できないと考えたほうがよい．確度との違いは，確度が値の切捨てや丸めによって生ずる誤差であるのに対し，この場合は，そのような誤差によるものではない点にある．この型の場合，必ずしも分類指標としての意味があるとは限らず，重複する区間も考えるので，後述の 6. のカテゴリーや 7. の順序カテゴリーではない．A, B, C による評価の場合のように，変域属性を利用して，取りうる値を明示するだけでなく，対応する区間の説明をコード属性で記述する必要がある．

一方，計数値の型としては当然，次の “数え上げ” を行う「個数」属性が必要となる．

5. **個数** (Count)
   数え上げた値であることを示す．したがって非負整数のデータベクトルのはずである．

さらに，計量値でも計数値でもなく広い意味でのカテゴリカルな値の型としては，次のようなものがある．特定値，等間隔，点数，区間などの型とカテ

ゴリカルな値との違いは，カテゴリカルな値は個体の類別に役立つ一つの属性として観測されるものである点で大きく異る． 実験条件などは特定値や等間隔であってカテゴリカルな値ではない． また，点数や区間はその値で類別する意図が明確ならばカテゴリカルな値として扱ってもよいが，そうでなければ不適当である．

6. **カテゴリー** (Category)
   類別を表す変量である．この場合，コード属性によってコーディングを与える必要があるが，特定値の場合などと異り，そこに与えられたコードが変域を与えているとは限らない．たまたま，現れた種類が記述されているだけであるからである．あらかじめ取りうる値が限定されているような場合には変域属性も与える．
7. **順序カテゴリー** (OrderedCategory)
   大・中・小 のように類別変量であることに加えて，コード属性で与えられる水準の順序に意味があることを示す属性．単なるカテゴリーとの違いは，のちのモデル化の段階でのコーディングや解釈に影響を与える．
8. **論理** (Logical)
   YES か NO あるいは TRUE , FALSE などのように排反な区別を示す変量．類別変量と似ているが，YES, NO は必ずしも類別を意味しているわけではないので，カテゴリーや順序カテゴリーとは区別が必要である．好き・嫌いは，好きでないからと言って嫌いと限らないので論理値とはならない．あくまでも類別値である．また，類別変量と違って，論理変量の場合はコード属性は不要で，慣例に従って常に 1 を YES， 0 を NO に対応させる．

次の型は，どちらかというと §2.2.1 で現れるような，補助的な変量に対応する型である．

9. **識別** (Id)
   異なる対象（個体）の観測あるいは記録であることを示すだけの意味しかない整数値である．型 Category あるいは OrderedCategory とまぎ

らわしいが，カテゴリーは個体の属性の一つであり，個体を識別するものではない．従って，コード属性もついているのが普通であるが，この Id の場合にはコード属性はつかない．それに対して，この属性 Id は観測あるいは記録の対象を識別するものである． 従って，関係形式の中では，このデータベクトルだけで候補キーとなりうる．

2003-07-25 修正

10. **番号** (Number)
    同じ対象に対する繰返し観測の観測番号など，番号ではあるが Id ではなく，しかもその値の順序に特に意味がないときの型である．したがって Code 属性はつかない．
11. **順序** (Sequence)
    型 Number あるいは Id の特別な場合で，その順序に意味がある場合である．

次の型は画像一枚一枚をベクトルの一つの値と同様に扱うために必要な型で特殊であるが，動画にシンクロナイズしたデータがある場合などに役立つ．

Images の説明追加; 2002-07-01

12. **画像** (Images)
    画像のフレーム番号であることを示す型である．この型の場合，属性 URL で動画像ファイルの所在を与え，属性 FrameInterval で動画一秒あたり何フレーム存在するかを示し，属性 Format で動画像ファイルの形式を与える必要がある．このような型が必要となるのは，他のデータベクトルと画像の各フレームが対応しているような場合である．

因子

実験計画が実施されたとき，各データベクトルに対応する変量がどのような性質の変量であるかを明らかにしておけば，後でモデルを構築し，解析するとき大いに役立つ．ただし，この分類は分野によって微妙に異なる点もあるので，詳しくは本シリーズの 2.『データサンプリング』を参照されたい．

因子変量 (factor variate) あるいは単に因子とは，原因となる変量という意味を強調するときに用いる．因子変量に関しては，まず，その値，つまり水準が実験を始める前にすでに定まっているか (preset)，実験後にはじめて判明するものかを区別する必要がある．これは，その因子をランダムな変量とみなしてモデル化するかどうかと密接に関連する．前者の場合，対応するデータベクトルの型は当然，特定値か等間隔のはずである．また，実験段階では，水準を定めて実験できたとしても，実験を離れて実用段階に移行した段階ではその因子がどんな水準を取るか特定 (specify) できない，あるいは特定することが現実的ではないといった因子もある．たとえば，臨床試験のときのように，実験段階では，どの患者に投薬したかといった形で,「患者」という因子の水準は実験前に定まっているが，実験を離れた現場では患者を特定することに意味がない．製造業なら実験段階では条件設定ができても，製造現場ではその条件では製造できるとは限らない．このように，いわば**実行可能性** (feasibility) に関わる区分である．まず，この二つの側面から，さまざまな名前のついた因子の区分をしてみると，次の表 2.1 のようになる．

型に関する記述追加; 2002-07-01

表 2.1

	実験前に水準が定まる	実験前に水準が定まらない
実用段階で特定可	制御因子，標示因子	
実用段階で特定不可	変動因子，ブロック因子	環境因子, 補助因子

ここで，同一の分類に属する**制御因子** (controllable factor) と**標示因子** (nominal factor) の違いは，前者が自由に水準を設定できるのに対し，後者は実験地域や気温のように自由には設定できない因子である点にある．また，**変動因子** (variable factor) は品質管理の分野などでは**誤差因子** (error factor) とも呼ばれる因子であり，その水準の変化が品質などにどのような影響があるかを実験するために導入される因子であるのに対し，**ブロック因子** (block factor) は，条件の均一化を図るために実験をブロックに分けて実施するとき，どのブロックでの実験結果かを表す因子である．実験してみなければ値が判明しない**環境因子** (environmental factor) は，**潜在因子** (latent factor) と呼

ばれることもあるが，そのような変量でも実験の主要因ではなく，補助的な役割しか果たさない変量は**補助因子** (auxiliary factor) と呼ばれることがある．

さらに，制御因子と同じような因子でありながら**信号因子** (signal factor) として区別される因子もある．信号因子は動特性を測るときの目標とするパラメータであるが，あらかじめ設定水準が定まっており，制御因子のように，いろいろな値を試すことに意味がない因子である点が，一般的な制御因子とは性格を異にする．信号因子に関しては，2.『データサンプリング』を参照されたい．そこで，DandD ではこれらの因子を属性 `Factor` で記述する．

```
Factor = "Controllable", "Nominal", "Variable", "Block",
         "Environmental", "Auxiliary", "Signal" のいずれか
```

層・クラスタ

型 `Id` を持つデータベクトルが示す対象（個体）が，調査目的に対して均質性を持つ集団であるとき，**層** (stratum) と呼ぶ．典型的には**標本調査** (survey sampling) の**層別抽出** (stratified sampling) で現れる．たとえば，§2.4.1 で詳しく述べる，総務省の行っている家計調査での第 1 段階の抽出では，市町村を大都市層・中都市層・小都市層といったいくつかの**層** (stratum) に分けて抽出する．それぞれの層は，大都市なら大都市の家庭の生活形態は似ており，中都市になればまた違ってくるだろうといった，これまでの経験に基づいてなるべく均質な集団となるように，層が形成されている．したがって，この場合，層属性を持つのは，最終的に抽出された世帯ではなく，各世帯の属する市町村の `Id` ベクトルである．層は，ある側面に関して均質性を確保するために導入されるという意味で，実験計画でのブロックと共通する概念である．ただし，それを抽出するか配置するかの違いがある．

(母）集団; 2002-07-01

クラスタは層と似ているが，(母）集団が調査目的に対して均質性を持つわけではなく，単に地理的に近いといった便宜上の理由で構成されている場合に用いる．したがって，関東・中部・関西といった地域的な区分で構成された集団は，地域性を調べることが目的ならば層であるが，地域性に依存しない特性を調べる場合には**クラスタ** (cluster) となる．このようなクラスタは，

いわゆる**集落抽出** (cluster sampling) で現れる．

`PopulationType= "Stratum"` あるいは `"Cluster"`

`PopulationDefinition =` 層またはクラスタ形成を説明する文の `ID` の並び

`PopulationDefinition` に複数の ID が与えられるようになっているのは多国語対応のためである．

2.2 データベクトルの構造化

データベクトルが一つしかない場合は別として，複数存在する場合には，それらの間の関係を適切に表現しておく必要がある．これがデータベクトルの構造化である．DandD ではこれを要素 `<Data>` で行う．一つのデータベクトルだけであるように見える場合でも，関係する補助変量を考慮すると単一のデータベクトルではなくなり，適切な構造化が必要になることも多い．次の配列形式はその例である．

2.2.1 配列形式

一つの変量の取る値を配列 $\{X_{i_1,i_2,\ldots,i_k}\}$ の形に配置したデータ形式が配列 (array) 形式で，本質的には一つのデータベクトルである．このデータ形式をよく考察してみると，各軸 (axis) に対応した**補助変量** (auxiliary variate) の存在に気づく．ここで**軸**とは，各添字 i_j が動く方向を示しており，この軸を j 軸と呼ぶ．配列の特別な場合である行列の場合なら，第 1 軸である行方向と，第 2 軸である列方向の二つの軸が存在する．たとえば，メッシュに区切った地図上の標高 $\{X_{i,j}\}$ を考えるとわかりやすい．この場合，メッシュが緯度，経度で区切られていれば，第 1 軸に対応する補助変量は緯度，第 2 軸に対応する変量は経度である．また，いくつかの質問に対する答の組合せそれぞれに関する回答人数を考えればこれも配列形式となる．この場合の補助変量は，それぞれの質問に対応し，その回答の並びが各軸を構成する．このような補助変量の取る値も，それぞれデータベクトルとなるが，軸の添字に対応する離散値しか取らないので，それに対応する型のデータベクトルとなる．

この配列形式を構成するには，配列の値のベクトルと軸に対応した補助変量のデータベクトルを組み合わせればよい． 配列の値は手前の添字がより優先的に変化する順序で並べる．

<Array Id=ID 名 LongName=説明文の ID の並び >
  <Axis RefId = 第 1 軸を規定するデータベクトルの ID />
    ⋮
  <Axis RefId = 第 k 軸を規定するデータベクトルの ID />
  <Value RefId = 配列の値を与えるデータベクトルの ID />
    ⋮
  <Value RefId = 配列の値を与えるデータベクトルの ID />
</Array>

Value に記述されるデータベクトルは第 1 軸の添字から動くように対応して並んでいる．添字の動く範囲は，要素 <Axis> に与えられたデータベクトルの Length 属性から明らかであるが，それぞれの添字がさらに何らかの意味を持っていることが普通である．それを説明するためにはデータベクトルの変域属性を用いる．また，要素 <Axis> の配置順は要素 <Value> の値の配置順と対応していなければならない．つまり，ここでは要素の配置順が重要な意味を持つ．また，複数の Value が与えられるようになっているので，同じ条件で複数の変量の値が観測されているような時にも，複数の配列形式をまとめて構造化できる．

なお，よく用いられる**データ行列** (data matrix) と呼ばれるデータ形式は，ここの配列形式とは異なる形式であることに注意しておこう．データ行列の第 1 軸は観測順，第 2 軸はいくつかの変量に対応しており，各列の値の均質性は確保されているが，異なる列の値は均質とは限らないので，厳密には配列形式ではない．しばしば例として用いられるフィッシャーのアイリスデータ (Iris data, [4], p.345) は， Setosa, Versicolor, Verginica の 3 種類のアイリスの分類を行うため，がくの長さと幅，花弁の長さと幅をそれぞれの種類

ごとに 50 の標本について調べたデータであるが，第 1 軸を観測順，第 2 軸を測定した部位，第 3 軸を種類にとれば，一見，3 次元配列データのように見える．しかし，配列の値はがくの長さや幅であったり，花弁の長さや幅であったりするわけで，単位は同じではあるが厳密には意味が違い，均質性が確保されているとは言いがたい．

もちろん，多変量解析などではこのようなデータを行列のように扱って線形演算などを施すが，それは「変量の違いによる非均質性を無視してモデル化したい」という積極的な姿勢を反映したもので，その段階までは配列形式としては扱わないほうがよい．アイリスデータを配列形式で扱うとしたら，がくの長さ，幅，花弁の長さ，幅 それぞれの値から成る四つの行列形式の集まりとして表すことになる．後で述べるように，関係形式を用いれば配列形式も表現できないことはないが，かなり冗長となるので， DandD ではあえて配列形式も導入している．

2.2.2 関係形式

関係形式 (relational scheme) は 1970 年に E.F.Codd が提案した**リレーショナルデータベース** (relational database) 構築の基本となるデータ形式で，いくつかのデータベクトルの集まりが一つの関係形式のデータ，つまり**リレーショナルデータ**を構成する．一見すると，各データベクトルを列とする 1 枚の表のように見えるが，各行の置かれている位置にはまったく意味がないところが，いわゆる表形式とは根本的に異なる．つまり関係形式では，行つまり記録の順序は自由に入れ替えられる．したがって，行の追加や削除の際に行位置を気にする必要はない．

数学的には，集合 $D_1, D_2, \ldots, D_n$ を**変域**（定義域，domain）とする**関係** (relation) とは，それぞれの集合の要素の組 $x_1, x_2, \ldots, x_n$ に対して真，偽を定める関数 $R(x_1, x_2, \ldots, x_n)$ のことである．この関数が真の値を取る集合を考えれば，これは**関係グラフ** (graph of relation) と呼ばれる $D_1, D_2, \ldots, D_n$ の直積集合の部分集合

$$R \subset D_1 \times D_2 \times \cdots \times D_n$$

となる．つまり，関係形式のデータは一つの関係グラフとみなすこともできる．

直観的には，集合 R は次のような一つの単純な表として考えるとよい．R の各点が表 2.2 の 2 行目以降の各行に対応する．

表 2.2 東京発フライト・スケジュール

会社名	行き先	便名	出発時刻	到着時刻
JAL	札幌	501	640	810
ANA	札幌	059	1030	1200
JAS	札幌	105	1025	1155
JAL	大阪	105	1330	1430
JAL	大阪	109	1935	2135
ANA	大阪	019	945	1045
JAS	福岡	261	1645	1825
ANA	福岡	301	655	835

この例では，$n = 5$ であり，

$$
\begin{aligned}
D_1 &= \{\mathrm{ANA}, \mathrm{JAL}, \mathrm{JAS}\} \\
D_2 &= \{\text{ 札幌，大阪，福岡 }\} \\
D_3 &= \{019, 059, 105, 109, 261, 301, 501\} \\
D_4 &= \{640, 655, 945, 1025, 1030, 1330, 1645, 1935\} \\
D_5 &= \{810, 835, 1045, 1155, 1200, 1430, 1825, 2135\}
\end{aligned}
$$

である．もちろん，実際のフライトをすべて考えれば，定義域 $D_1, D_2, \ldots, D_5$ はもう少し広く取っておく必要がある．表 2.2 の第 1 行にある会社名，行き先，便名，出発時刻，到着時刻などは各定義域の名札であり，このような名札を持つ定義域 $D_1, D_2, \ldots, D_n$ を属性として加えた R が一つのリレーショナルデータである．一方，同じデータを次の図 2.1 のような**リスト** (list) あるいは**樹**（木，tree）の形で表現することもできる．

この形式は日常的にもよく見かける形式であり，表 2.3 のような時刻表の形に書き直しても同等である．この型のデータベースは木構造，つまり階層構造が現実世界をうまく反映していて，それに即したデータの使われ方がなさ

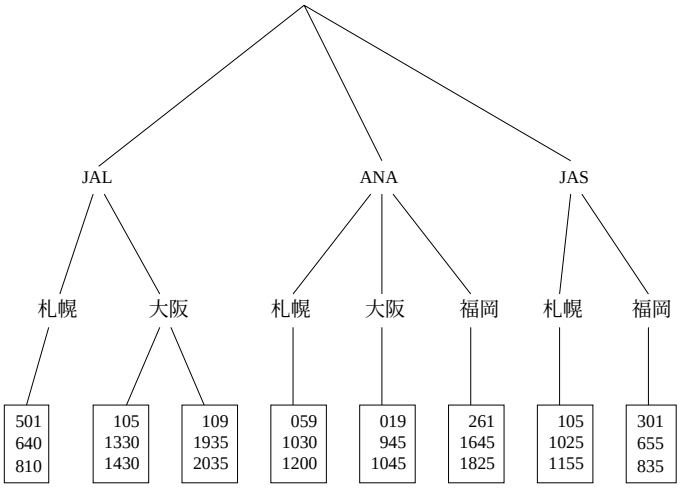

図 2.1 東京発フライトの一部のリスト表現

れればきわめて能率がよい．たとえば上の例で，JAL のスケジュールをそっくり入れ替えるとしたら部分木の置換えにすぎないので，容易な操作である．しかし，行き先や出発時刻，到着時刻で条件探索を行おうとすると，すぐ能率が悪くなるのは明らかであろう．樹の形に表したときは，その節が「JAL」や「札幌」といった具体性のあるもの，つまり**実体** (entity) であったのに対し，リレーショナルデータの場合には，会社名，行き先，…といった抽象的な**定義域** (domain) である点に注意する必要がある．一言でいえば注目の仕方が違う．このことは表 2.2 と表 2.3 を比較してみてもよい．表 2.3 では会社や行き先に関する列と残りの列では要素の数が異なり，要素の配列順に意味がある．これに対し表 2.2 のリレーショナルデータでは行の順序は自由に入れ替えられ，行の追加や削除の際に行位置を気にする必要はない．これは，各行が集合 R の各点に対応している以上当然であるが，これが関係形式の利点である．

Codd は，リレーショナルデータに対する操作として，次の八つの操作を基本的なものとした．

表 2.3 東京発フライト・スケジュールの一部

会社名	行き先	便名	出発時刻	到着時刻
JAL	札幌	501	640	810
	大阪	105	1330	1430
		109	1935	2035
ANA	札幌	059	1030	1200
	大阪	019	945	1045
	福岡	261	1645	1825
JAS	札幌	105	1025	1155
	福岡	301	655	835

1. $R \cup S$ 和 (sum)
2. $R - S$ 差 (difference)
3. $R \cap S$ 共通部分 (intersection)
4. $R \otimes S$ 直積 (product)，R, S はそれぞれリレーショナルデータ
5. $R[B]$ 射影 (projection)，B は R の定義域のいくつかの直積 (部分直積)
6. $R[A\theta B]$ 制約 (restriction)，A, B はそれぞれ R の定義域の一つで演算 θ が可能なもの
7. $R[A\theta B]S$ R と S の結合 (join)，A, B はそれぞれ R,S の定義域の一つで演算 θ が可能なもの
8. $R[A \div B]S$ R と S の商 (quotient)，A, B は R, S それぞれの定義域の部分直積で，集合として一致しているもの

ただし，R, S はそれぞれあるリレーショナルデータで，1. から 3. では同じ定義域を持つものでなければならない．また A, B はそれぞれ名札のついた定義域，θ は $=, \neq, >, \geq, <, \leq$ のいずれかの演算子である．1. から 3. は集合一般についての操作にほかならないので特に説明の必要はないだろう． 4. の直積も集合としての直積集合を作る操作で，R が n 次元空間の部分集合，S が m 次元空間の部分集合ならば，$R \otimes S$ は $n + m$ 次元空間の部分集合となる．いま，定義域の名札を**項目**と呼ぶことにすれば，直積とは， R の項

目と S の項目を合わせた項目を持つ大きなリレーショナルデータを作り出す操作であり，射影 $R[B]$ とは，B で指定された項目だけを抽出する操作である．なお，データベースの世界では「項目」は単に「属性」と呼ばれることが多い．各行の属性という意味であるが，これまで議論してきた属性とまぎらわしいので，あえて項目とよぶことにしている．制約とは，項目 A と項目 B を比較して真となる行だけを取り出す操作，結合とは，項目 A, B をキーにして二つのリレーショナルデータ R, S を一つにまとめる操作である．商とは，いわば直積の逆演算で，A に含まれない R の定義域の直積を A^c と表したとき，R を A^c に制限したとき現れる同一な記録のうち，対応する項目 A の記録の集合が，項目 B の記録の集合を完全に含む場合だけを取り出したものである．

実際には，八つの操作すべてが必要ではなく，1, 2, 4, 5, 6 の五つがあれば残りの三つは作り出せる．つまり，

$$\begin{aligned} R[A \div B]S &= R[A^c] - (R[A^c] \otimes S[B])[A^c] \\ R[A\theta B]S &= (R \otimes S)[A\theta B] \\ R \cap S &= R - (R - S) \end{aligned}$$

の関係から残りの操作が導ける．しかし実用上は，計算効率の点からもこの八つすべてを用意することが多い．

なお，いずれの操作の結果についても，重複する記録や項目は常に削除するので，上の 3 番目のように結合を選択で書き表しても問題はない．また，実際には，特定の値を含む記録だけを取り出したいことがよくあるので，リレーショナルデータ S の代わりに，{810} や {JAL} のように単一の値を与えることもできるように設計される．このような値は，A で指定された項目と同じ記録数に複製され，常に項目 B として用いられる．たとえば

$$R[A = B]\{810\}$$

で，R の項目 A に 810 を含む記録だけが取り出せる．7. の定義どおりならば，この演算の結果には，810 だけから成る項目 B が残るはずであるが，実

際には，このような冗長な項目は削除されるように設計されているのが普通である．このように，単純な抽出の場合でも

$$R[A = 810]$$

のような操作を許さず，上のような演算を用いることにしてある理由は，810という実体と定義域（項目）の混在を避けるためである．

さらに，いくつかのリレーショナルデータの間に記録の重複があったり，関係の重複もありうる．このような冗長性をなるべく減らして，効率化を図ることをデータベースの**正規化** (normalize) と言い，第 1 正規形から第 4 正規形まで具体的な正規化の手順も提案され実用されている [1–3]．

なお，固定欄形式のデータに各欄の名札がついたものは，すでにリレーショナルデータになっていると考えることもできる．各座標とみなすことにより，各行が空間の 1 点を表していると見ることができる．そうすると，一つの関係形式のデータは行数だけの点から成る直積空間上の集合であると考えられる．

リレーショナルデータベースはリレーショナルデータの集まり，いわば何枚かの単純な表の集まりとして保存する方式で，多少の冗長性はあるものの，汎用であることとその操作の容易さから **SQL** (Structured Query Language) といったリレーショナルデータベース専用の標準言語も開発され，現在もっとも広く用いられているデータベースの形式である．本シリーズの 3.『データマイニング』にも解説があるので参照されたい．

先に述べたように，配列形式も関係形式に書き表すことができる．配列 $\{X_{i_1,i_2,\ldots,i_k}\}$ に付随した補助変量 $I_1, I_2, \ldots, I_k$ を用いて，最初の k 列をこれら k 個の補助変量の値，その後に対応する $\{X_{i_1,i_2,\ldots,i_k}\}$ の値を，たとえば

$$X_{1,1,\ldots,1}, X_{1,1,\ldots,2}, \ldots, X_{1,2,\ldots,1}, X_{1,2,\ldots,2}, \ldots, X_{m_1,m_2,\ldots,m_k}$$

のように，後の添字ほど速く動く順に並べた 1 列を置いた関係形式とすればよい．ただし， $m_1, m_2, \ldots, m_k$ は添字それぞれの動く範囲（上限）である．もちろん，こうすると，たとえば第 1 列は最初の $m_2 m_3 \cdots m_k$ 個の要素は

すべて 1 であり，その後に同じ個数の 2 が並び，$\cdots$ のように，きわめて冗長になる．これが DandD で配列形式も許すことにした大きな理由である．

一方，**不均等配列** (lagged array) を配列形式で表すには無理があり，関係形式を用いたほうがよい．不均等配列は，添字の動く範囲が一定ではない配列で，もっとも簡単な例は患者の入院から退院までの記録である．患者によって入院期間は一定ではなく，したがって，患者ごとの記録数も一定ではない．このようなデータを，たとえば，第 1 軸を患者の識別番号，第 2 軸を体温に取って，配列で表そうとすると不均等配列になってしまい，配列形式で表そうとすると，最大の記録数に合わせて第 2 軸を設定し，記録がない部分は欠損値 NA で埋めるなどの操作が必要になり，冗長性が増すだけでなく，扱いもやっかいになる．このような場合，新たに日付の変量を導入して第 1 列を患者の識別番号，第 2 列を日付，第 3 列をその患者のその日付での体温とすれば，関係形式で表現できる．もちろん，第 1 列，第 2 列には重複した値が多く現れ冗長になるが，扱いは楽になる．

```
<Relational Id = ID 名  LongName = 説明文の ID の並び >
    <Value RefId = 関係形式を構成するデータベクトルの ID の並び />
    <Value RefId = 関係形式を構成するデータベクトルの ID の並び />
            ⋮
    <Value RefId = 関係形式を構成するデータベクトルの ID の並び />
</Relational>
```

RefId に ID を与えられたデータベクトルの持つ変域属性はちょうど関係形式の定義域 $D_1, D_2, \ldots, D_n$ に相当する．この属性は配列形式との関連付けや，さまざまなデータ取得過程の記述などに利用できる．また，変域属性を共有するデータベクトルは外部キー (foreign key) であり，一つの関係形式で変域属性を持つデータベクトル全てが候補キー (candidate keys) を構成する．ここで，単独の ID だけでなく，複数の ID の並びを与えられるようになっているのは，異るデータベースから取得したデータベクトルを結合して一つの Value も構成できるようにするためである．その場合，変域属性，コード属性などさまざまな属性の併合も必要となる．すでに説明した配列形式の場

正規型との関係, 並べ替え質問の記述の多様性に対する答え要; 2002-07-01, 2003-06-16
Systems の導入; 2003-11-16

合は，このような結合は許していないことに注意． また，関係形式のうちのいくつかの列が年月日や緯度経度などの基数系を構成している場合は，

<Value RefId = 関係形式を構成するデータベクトルの ID の並び
Systems = 基数系，主キーなどを示す ID の並び/>

のように該当する列に対して Systems 属性を与える．この Systems 属性は Appendix 要素にある基数系や主キーを参照している．詳しくはこの章の **2.4** を参照されたい．

2.2.3 その他の形式

不均等配列のような場合，すでに触れた**リストの形式**も許すことにすれば冗長性は減少させられるが，ポインタを導入するか， LISP のように括弧を多用した表現形式が必要になり，扱いは格段に複雑になる．また，リスト，つまり木構造の一般形である**ネットワーク**（network，グラフ）**形式**まで導入すると，目的によってはさらに便利になることは確かである．たとえば，先のフライトデータをネットワーク形式に書き換えると，図 2.2 のようになる．この**ネットワーク・データベース**には親となる節と子となる節があり，図 2.2 の場合，会社名や行き先が親節である．どちらの親節からも容易に各便にアクセスできるところがこのネットワーク・データベースの特徴である．しかし，すぐ複雑になるところが問題である．なお，木構造はネットワーク構造の特殊なものであり，根元 (root) という特殊な節があって，これ以外の節は互いに素ないくつかの木に分けることができるネットワークが，木である．上のネットワークの例は，行き先と会社名の二つの木を重ねたものとみなすこともできる．

また，最近ではネットワーク形式と関係形式の混合とみることもできるオブジェクト指向の形式の研究も始まっているが，データの中身に中心的な興味があるデータサイエンスでは，関係形式を基本とし，必要に応じて配列形式を用いるほうが，少々能率が悪くてもその扱いの単純さから適していると思われる．とくに XML で一般的なリストやネットワークを表現するのは複雑になりすぎるので， DandD では，あえてデータ形式を関係形式と配列形

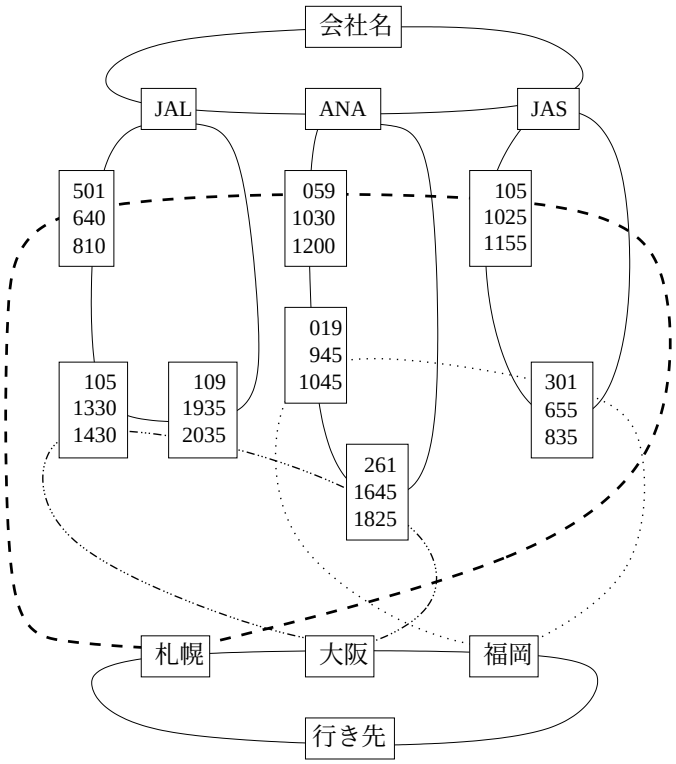

図 2.2 ネットワーク データベース

式に制限している.

2.3 特別な意味を持つ構造

データベクトルそれぞれについてさまざまな属性が付随することはすでに説明したが，いくつかのデータベクトルを配列形式あるいは関係形式に構造化して，はじめて生まれる属性もある．これらの属性はデータベクトルの属性と同様，解析やモデル化の重要な手がかりを与える．ただ，複数のデータベクトルが関連する属性であるので，データベクトルの属性ほどには簡単に

は記述できない．そこで DandD では直接，属性として与える代わりに，そのような属性を持つ配列形式や関係形式を分類し，異なる名前をつけることによって，そのような属性を表現する方法を取っている．先の要素 <Array> や <Relational> を用いた構造化はそのような属性のない場合である．また，同時にいくつもの属性が存在する場合には，同一の構造を複数の異なる名前で定義することによって表現する．

2.3.1 グラフ，関連度表

グラフ (graph) はいくつかの頂点を辺で結んだ図形であるが，関連表 (association table) と呼ばれる行列で表現できる．この行列の各軸には頂点を並べ，値が 1 ならその頂点同士は結び，0 ならば結ばないことにすればよい．このようなグラフは**無向グラフ** (undirected graph) と呼ばれるが，辺に向きのある**有向グラフ** (directed graph) も，第 1 軸の頂点から第 2 軸の頂点へ矢印で結ぶ慣例に従えば，同じように関連表で表現できる．

いくつかの変量の相互の関連の度合いを表す，**関連度表** (association table) もグラフの一般化とみなすことができる．つまり頂点に変量を割り当て，関連度を辺の上に置いたグラフをつくればよい．

さらに，トーナメントの勝敗表も有向グラフの一般化とみなすことができる．第 1 軸のチームが第 2 軸のチームに勝てば 1，負ければ 0 の行列がトーナメントの勝敗表である．引き分けの場合には対称な位置に 1 を置けばよい．また，対戦しなかった場合は NA を埋めればよい．

DandD では，このようなさまざまな表をグラフとして統一的に扱う．形式としては配列形式である．

<Graph Id = ID 名 LongName = このグラフの表す内容に関する説明への ID の並び >

<Axis RefId = 第 1 軸を規定するデータベクトルの ID />

<Axis RefId = 第 2 軸に対応するデータベクトルの ID />

<Value RefId = 第 1 軸に並んだ頂点と第 2 軸に並んだ頂点の関連度を表すデータベクトルの ID />

```
</Graph>
```

要素 <Value> に対応するデータベクトルの型が Logical で対称行列なら無向グラフ，そうでなければ有向グラフである．

構造 Permutation 廃止, Constraint で記述; 20020-07-01

2.3.2 時系列データ

時系列 (time series) は時刻あるいは日付のデータベクトルとそれに対応した（複数の）値のベクトルの関係形式として表せる．時間に依存して値が定まるわけであるが，時間の系列そのものよりも値の動きに興味があるといった意味で**時間軸** (time axis) の役割が特殊である．

```
<TimeSeries Id = ID 名  LongName=説明文の ID の並び >
    <Condition RefId = データを類別するための条件となるデータベク
                       トルの ID の並び />
              ⋮
    <Condition RefId = データを類別するための条件となるデータベク
                       トルの ID の並び />
    <TimeAxis RefId = 時系列の時間軸を規定するデータベクトル
              Systems = 基数系の ID の並び />
              ⋮
    <Value RefId = 時系列の値を与えるデータベクトルの ID の並び />
              ⋮
    <Value RefId = 時系列の値を与えるデータベクトルの ID の並び />
</TimeSeries>
```

Condition は，条件の異る複数の時系列を一本にまとめたような場合，その条件の違いを知るために必要なデータベクトルである． 複数あってよい．TimeAxis は時系列の観測時点の並びであるが，年月日のように複数のデータベクトルで構成されている場合は，別途，基数系を構成した上で参照する．また，かならずしも規則的にしかも順序に並んでいる必要はない．特に，

Condition が存在する場合は，時刻が原点にもどることもしばしばである．ただし，このように Condition を用いた場合， ブラウザーでの条件選択も可能としたほうがよいので，データは外部データとすることが望ましい． 2002-11-06, 2003-06-16 追加

2.3.3 点過程データ

点過程 (point process) データは，ある事象の生起時点の並びである．この一般化が空間点過程 (spatial point process) データで，本シリーズの 7.『空間データモデリング』でも解説されているように，何らかの事象が起こった時間と位置に関する記録である． さらに，位置に**マーク** (mark) と呼ばれる値が付随していることもある．時系列と類似しているが，時系列の場合は，時間が一つの流れを定める軸に対応するだけであるのに対し，空間データの場合は，時間や位置そのものに興味がある点が異なる．

```
<PointProcess Id = ID 名 LongName = 説明文の ID の並び >
   <Condition RefId = データを類別するための条件となるデータベク
                      トルの ID の並び />
          ⋮
   <Condition RefId = データを類別するための条件となるデータベク
                      トルの ID の並び />
   <Point RefId = 生起した事象の座標を与えるデータベクトル
          Systems = 基数系の ID の並び  />
          ⋮
   <TimePoint RefId = 生起時点を与えるデータベクトル
              Systems = 基数系の ID の並び />
          ⋮
   <Mark RefId = 生起事象に付随した値を与えるデータベクトル
                 の ID の並び />
          ⋮
```

```
    <Mark RefId = 生起事象に付随した値を与えるデータベクトル
                  の ID の並び />
  </PointProcess>
```

Condition の役割は時系列と同じである． ただし， TimePoint や Point の値には重複もありうる．また，マーク付き点過程は，時系列と点過程の 2 つの関係形式に分けて組織化することもできる．この違いは，点過程とそこで発生する値を一体化したものとして考えるかどうか，見方の違いからくる．

RandomCensoring はデータベクトルの属性に移管; 2002-07-01, 2003-06-16

2.4 複数のデータベクトルの間に存在する関係

基数系の Appendix への移行： 2003-11-16

時系列や点過程における時間軸は，複数のデータベクトルからなる基数系を構成していることがある．また関係形式ではいくつかの項目 (データベクトル) が主キーを構成していることもある．このようなデータ構造の一部が特別な意味を持つ場合，具体的な構造を別途記述し，その関係を明らかにする必要がある．DandD ルールでは Systems 属性を，データベクトルをあらわす Value 要素などに与え，Appendix 要素に記述した構造を参照させることで複数のデータベクトル間に存在する関係を表現する．

2.4.1 主キー

主キーは関係形式のレコード，つまり各行を一意に識別することができる項目をさし，第一正規形以上の関係形式には必ず存在する．主キーは必ずしも一つの項目とは限らず，複数の項目を組み合わせて主キーとして用いる場合こともある．

```
<MainKey Id = ID 名>
    <Key RefId = データベクトルの ID の並び />
    <Key RefId = データベクトルの ID の並び />
                ⋮
</MainKey>
```

複数の関係形式が同じ MainKey 要素を共有していれば，それらのデータベクトルは **2.2.2** で述べた外部キーを構成していることになる．つまり，キーのドメインを手がかりにすることで一つの関係形式としてまとめることできる．

2.4.2 制約

データベクトルの変域属性で記述できるのは，一つのデータベクトルについてだけの制約であった．条件付き制約を含め，複数のデータベクトルにまたがる値の制約を記述するためには，次のような特殊な関係形式を必要とする．

検討課題: 制約を Which ではなく具体的な名前で, 2002-07-15

```
<Constraint Id = ID名 LongName = どのような制約であるかの説明へ
                                 の ID の並び >
    <Which RefId = 制約の対象となるデータベクトルの ID の並び />
              ⋮
    <Which RefId = 制約の対象となるデータベクトルの ID の並び />
    <Rank  RefId = 順位ベクトルの ID の並び Of=順位の対象である
                   データベクトルの ID の並び />
              ⋮
    <Rank  RefId = 順位ベクトルの ID の並び Of=順位の対象である
                   データベクトルの ID の並び />
    <Given RefId = 条件付き制約の条件を与えるデータベクトルの ID の
                   並び />
              ⋮
    <Given RefId = 条件付き制約の条件を与えるデータベクトルの ID の
                   並び />
</Constraint>
```

Permutation の移行; 2002-07-01, 2002-07-15

この属性を利用すれば，嗜好調査で好みの順に並べて回答させたときのような，結果が並べ替えになっていることの記述も可能である．関係形式で，変

域 (Domain) を共有しているだけでは，並べ替えになっていることまで記述できないが，このような回答を並べ替えの対象を表す列とその順位の列を含む関係形式で表したうえ，Rank に順位ベクトルの ID をあたえ，その属性 Of に順位付けの対象となったデータベクトルを与えればよい．Which や Given に複数の ID が与えられるようになっているのは Relational の時と同じように複数のデータベースから取得したデータベクトルを結合できるようにという配慮からである．

IDREFS に変更; 2002-07-01

2.4.3 区間

観測によっては，値は直接得られないが，ある値からある値の範囲であるということだけはわかる場合がある．このようなデータは単一のデータベクトルでは表現できず，最小値と最大値のデータベクトルの組で表すことになる．データベクトルの型 Interval との違いは，型 Interval ならば，あらかじめ定められた区間のいずれかに値が落ちるのに対し，ここでは観測ごとに最小と最大の値が変化するようなデータを考えている．もちろん最小と最大のどちらかだけが記述できる場合も多い．

```
<Interval Id = ID 名 LongName = 説明文の ID の並び >
    <Min RefId = 最小値の並びを与えるデータベクトルの ID />
    <Max RefId = 最大値の並びを与えるデータベクトルの ID />
</Interval>
```

以上はもっぱら主目的のデータに関するものであったが，以下はどちらかと言うと，いくつかのデータベクトルが組で持つ属性を付随的に記述するものである．

多次元の区間や，複雑な集合の記述: 検討課題 2002-07-01

2.4.4 基数系

基数系 (radix) は年月日のような日付，時刻の時分秒，角度の度分秒のように一つの値を表すのにいくつかの基数を用いて表現するときに必要となる一つの系である． ただし，年，月，日 だからといって常に基数系として構成する必要はない．これらが他の関係形式の列として現れていたり，時系列

の TimeAxis として基数系が引用されていたときに必要となるが，配列の軸として引用されているようなときには，構成する必要はない．

日時

第 1 章でも述べたように，通日を用いない限り，年月日は暦に依存している．暦にも西暦，和暦，中国暦をはじめとしてさまざまな暦があり，その基数系，つまり年・月・日の桁の繰上がりも同一ではない．時差も考慮するならば，どの地域での日付かを明記する必要がある．**曜日** (week) も**暦** (calendar) の構成要素ではあるが，日付からいつでも再現できるので，データ取得の段階で記録する必要はない．曜日と日付の対応が間違っていたりすると，かえって混乱のもとになる．

```
<Time Id = ID 名 LongName = 暦の特定と地域の特定の ID の並び >
   <Era RefId = 平成，昭和などの年号を表すデータベクトルの ID />
   <Year RefId = 年を表すデータベクトルの ID の並び />
   <Month RefId = 月を表すデータベクトルの ID の並び />
   <Day RefId = 日を表すデータベクトルの ID の並び />
</Time>
```

なお通常，要素 <Era> に与えられた ID を持つデータベクトルの型は順序因子であり，コーディング属性をもつ．すべて同じ年号の場合には，省略してもよい．年月日と同様，時刻 (clock) も時分秒から成る一つの基数系を構成する．

```
<Time Id = ID 名 LongName = 説明文の ID の並び >
    <Hour RefId = 24 時間制での時刻を与えるデータベクトルの ID の
                                        並び />
    <Minute RefId = 分を与えるデータベクトルの ID の並び />
    <Second RefId = 秒を与えるデータベクトルの ID の並び />
</Time>
```

度

角度，緯度・経度などは伝統的に 60 進 (sexagesimal) を基本とする度分秒の基数系で表現される．これを明示するのが以下の構造である．

```
<Sexagesimal Id = ID 名 LongName = 説明文の ID の並び >
   <Degree RefId = 度を与えるデータベクトルの ID の並び />
   <Minute RefId = 分を与えるデータベクトルの ID の並び />
   <Second RefId = 秒を与えるデータベクトルの ID の並び />
</Sexagesimal>
```

2.4.5 座標系

空間の位置を定める座標も一つの系を成し，大きく直交座標系と極座標系に分かれる．

直交座標系

直交座標系 (orthogonal coordinate) に関しては，原点をどこに取ったかの記述はもちろんのこと，x, y, z 軸の方向についての記述も欠かせない．いわゆる左手系，右手系などだけでなく，具体的な現象に即した方向の記述が必要である．もちろん原点に関する記述もなければ困る．

```
<OrthogonalCoordinate Id = ID 名 LongName = 説明文の ID の並び
Definition = 原点についての記述の ID の並び >
   <X Definition = x 座標軸の方向についての記述への ID の並び
      RefId = x 座標を与えるデータベクトルの ID の並び />
   <Y Definition = y 座標軸の方向についての記述への ID の並び
      RefId = y 座標を与えるデータベクトルの ID の並び />
   <Z Definition = z 座標軸の方向についての記述への ID の並び
      RefId = z 座標を与えるデータベクトルの ID の並び />
</OrthogonalCoordinate>
```

2 次元座標の場合は X,Y だけを用いればよい．

極座標系

極座標系 (polar coordinate) は平面上の位置を始線からの偏角と原点からの距離の組で表す座標系であり，たとえば，森の中で周囲を見回したときの木の位置などはこの座標系で観測される．偏角は反時計廻りに測るものとする．

<PolarCoordinate Id = ID 名 LongName = 説明的な名前の ID の並び
Definition = 原点についての記述の ID の並び >
<R RefId = 原点からの距離を与えるデータベクトルの ID の並び />
<Theta Definition = 偏角を測る基準となる始線の説明の ID の並び
RefId = 偏角を与えるデータベクトルの ID の並び />
</PolarCoordinate>

ここで，複数の ID が与えられようになっているのは，多言語対応のためである．渡り鳥の飛び立つ方向の記録など，いわゆる**方位データ** (directional data) の場合には距離は観測されず，偏角に相当する方位だけが観測される場合も多い．以下は，極座標を一部用いた 3 次元座標系の典型例である．

円柱座標系

円柱座標系 (cylindrical coordinate) は，2 次元平面上の極座標に高さを加えた 3 次元の座標系であり， 2 次元の極座標系を特殊なものとして含む．これに関しても原点をどこに取ったか，極座標を考える平面つまり基準面をどうとるかの記述だけではなく，偏角をどの方向に測るか，高さ z をどの方向に測るかの記述も必要である．

<CylindricalCoordinate Id = ID 名 LongName = 説明文の ID の並び
Definition = 原点と基準面についての記述の ID の並び >
<R RefId = 基準面に射影したときの原点からの距離を与える
データベクトルの ID />
<Theta Definition = 偏角を測る基準となる始線についての記述
の ID の並び

RefId = 基準面に射影したときの偏角を与える
データベクトルの ID />
<Z Definition = z 軸の方向についての記述の ID の並び
RefId = z 座標を与えるデータベクトルの ID />
</CylindricalCoordinate>

ここで，複数の ID が与えられようになっているのは，多言語対応のためである．

球面座標系

球面座標系 (spherical coordinate) は星の位置を示すのによく用いられる座標系で，二つの偏角，方位と仰角，その星までの距離で表す座標系である．天文学では，伝統的に仰角は天頂を基点として測ることが多いが，必ずしもこれにこだわる必要はない．

<SphericalCoordinate Id = ID 名 LongName = 説明文の ID の並び
Definition = 原点と基準面についての記述の ID の並び >
<R RefId = 原点からの距離を与えるデータベクトルの ID の並び />
<Theta Definition = 方位を測る基準となる始線についての説明の
ID の並び
RefId = 方位を与えるデータベクトルの ID の並び />
<Phi Definition = 仰角を測る基準となる始線についての記述の ID
の並び
RefId = 仰角を与えるデータベクトルの ID の並び />
</SphericalCoordinate>

ここで，複数の ID が与えられるようになっているのは，多言語対応のためである．

緯度経度

地球上の位置は**緯度** (latitude) と**経度** (longitude) の組で表されることが多いが，これは球面座標系の特殊なものである．地球の中心を原点として，方位を赤道面で測ったのが経度であり，仰角を赤道面を基準に測ったのが緯度である．習慣的にグリニッジを 0 として東回りに測った方位角を東経，西回りに測った方位角を西経と呼ぶが，東回りだけを 360 度まで用いて測った経度を用いることもある．緯度も，赤道を 0 として，北緯，南緯と南北に分けて測った仰角を用いることが多いが，北緯を正，南緯を負とした緯度を用いると計算が楽である．

```
<LongitudeLatitude Id = ID 名 LongName = 説明文の ID の並び >
   <Longitude Definition = 経度の測り方の説明の ID の並び
              RefId = 経度を与えるデータベクトルあるいは
                      度分秒の基数系の ID の並び />
   <Latitude Definition = 緯度の測り方への ID の並び
             RefId = 緯度を与えるデータベクトルか,
                     度分秒の基数系の ID の並び />
</LongitudeLatitude>
```

ここで，複数の ID が与えられるようになっているのは，多言語対応のためである．

2.5 データ取得計画

計画的な実験計画や標本調査に基づいてデータが取得されたときは，その計画の記述をしておかないと，計画の意図がモデル化の段階まで伝わらない．DandD では要素 <DataSampling> の中でこれらを統一的に記述する．この要素には属性 LongName , Date があり，前者にはデータ収集者名あるいは機関名，さらには連絡先の住所，メールアドレスなどを与え，後者には，データ取得の日付，期間などを与える．まず実験計画と標本調査に共通なこととして，観測の偏りを最小限に抑えるために行われる**ランダム化** (randomize)

に関する記述がある．きちんとした計画に基づいたデータ取得でない場合でも記述しておくとよい．

2.5.1 ランダム化

もっとも簡単なランダム化は個体（観測対象，抽出単位）をランダムに抽出（**無作為抽出**， random sampling）あるいは配置することであり，後者は抽出数が個体数と等しい非復元抽出である．しかし，標本調査での無作為抽出や実験計画のブロック内でのランダム化などはより複雑で，同一条件を満たす個体を選んだうえで行われるランダム化である．また何段階にも渡ってランダム化されることがあるため，個体それぞれを単位とするランダム化ではなくその一部の属性，たとえば世帯の属する市町村，個人の属する会社，実験条件などに関するランダム化であることも多い．

このように多様なランダム化を表現するには，ここまで用いてきた単純な表現の組合せでは限界がある．そこで，ここでは SQL の選択文を借用した次のような記述方式を用いて記述する．

SELECT X FROM R WHERE α

要素 <Data> に現れた ID が R の関係形式の要素 X の要素のうち条件式 α を満たす添字の要素だけを抜き出すことを表す．この選択文の代わりに直接データベクトルの ID を与えてもよい．

```
<Randomize Id = ID 名 LongName = ランダム化の目的の説明文の
                                    ID の並び
          For = "(i in 条件変数の変域を与える ID )" >
    <Randomized Distinct = "YES" または"NO" >
          選択文
    </Randomized>
    <From Size = 選択数 >
          選択文
    </From>
    <Rate>
```

```
            選択文
    </Rate>
  </Randomize>
```

ここで，属性 For に与えられた文は，各 SQL 選択文の条件 α に現れる変数 i の定義とその動く範囲を規定しており，次の例のように複数の変数を定義することもできる．上の記述の意味は，要素 <From> の選択文で指定された要素のうち，Size 個をランダムに抽出した結果が要素 <Randomized> の選択文で指定された要素と，集合として一致することを表す．属性 Distinct が "NO" ならば，重複も許して比較し，一致することを示す．省略時は "YES" である．必要ならば要素 <Rate> の選択文で抽出確率（のベクトル）を与える．

例 1

```
<Randomize Id = "rRifle"
           LongName = 実験に用いるライフルのランダム化に関する
              説明文の ID の並び
           For = "(i in DayDomain)(j in ShooterDomain)" >
  <Randomized>
  SELECT Rifle FROM RifleTest
  WHERE (SELECT Shooter FROM RifleTest WHERE Day=i)=j
  </Randomized>
  <From Size = "1" >
  RifleDomain
  </From>
</Randomize>
```

これは，DayDomain と ShooterDomain の値を組み合わせて実験日と実験者を指定したとき，どの Rifle を用いて実験するかを，利用できるライフル RifleDomain のうちから，ランダムに選択して行う実験計画を記述している．ここで， RifleTest はこの実験結果をまとめた関係形式である．

例 2

総務省が行っている家計調査の第 1 段階の無作為抽出は次のように記述できる．

```
<Randomize Id = "rCity" LongName = 第 1 段階の無作為抽出の説明
                                   文の ID の並び
          For = "(i in CityStrataDomain)" >
   <Randomized Distinct = "YES" >
   SELECT City FROM HouseHoldSurvey WHERE Stratum = i
   </Randomized>
   <From>
   SELECT city FROM SamplingTable WHERE stratum = i
   </From>
   <Rate>
   SELECT Prob FROM SamplingTable WHERE stratum = i
   </Rate>
</Randomize>
```

この第 1 段階の無作為抽出では，同一の市町村層 Stratum = i に属する市町村のうちから，いくつかの市町村 City が，各市町村の人口に比例した抽出確率 Prob で抽出される．さらに第 2 段階の無作為抽出では，それぞれ抽出された市町村から何軒かの調査世帯が選ばれているので，調査結果を表す関係形式である HouseHoldSurvey には同じ市町村がその軒数だけ重複して現れる．第 1 段階の抽出を記述する際には，そのような重複は除いて考える必要があるので Distinct = "YES" としてある．上の記述は，このように <Randomized> で特定された市町村が，<From> と <Rate> の指定で抽出された市町村になっていることを示している．また，SamplingTable も関係形式であり，<Data> で次のように定義されている．

```
<Relational Id = "SamplingTable"
            LongName = "市町村，その所属市町村層と抽出確率" のよ
```

うな説明文の ID の並び >

```
    <Value Id = "city" />
    <Value Id = "stratum" />
    <Value Id = "Prob" />
</Relational>
```

さらに，データベクトル Stratum の属性は次のように与えられている．

```
<DataVector Id = "Stratum" DataType = "Id"
            Domain = "CityStrataDomain" >
```

また，CityStrataDomain はすべての市町村層を与えるデータベクトルであり，次のような属性を持つ．

```
<DataVector Id = "CityStrataDomain" Length = "168"
            DataType = "Id" PopulationType = "Stratum"
            PopulationDefinition = "都道府県庁所在地および人口
            100 万人以上の市については各市を 1 層とし，その他の市
            については地方，都市階級で分類後，世帯数比率，人口増加
            率，人口集中地区の有無，人口比率，産業的特色等を考慮し
            て 71 層に分類．町村に関しては，地方ごとに地理的位置，
            世帯数比率，人口増加率を考慮して 48 層に分類" といっ
            た説明文への ID の並び >
```

このように，一言で無作為抽出といってもきちんと記述しようとすると，さまざまなことを記述する必要が生ずるが，例 4 で第 2 段階，第 3 段階の無作為抽出なども含めた記述を与えるので参照されたい．PopulationDefinition に複数の ID が与えられるようになっているのは，多言語対応のためである．

例 3

Cox and Snell [5] のデータ例 Q は，2 種類の毛糸 (worsted yarn) の強さの実験結果であるが，それぞれの毛糸について六つの糸巻き (bobbin) をランダムに選び，それぞれの糸巻きからランダムに 4 本の試験片 (samples) が切

り取られている．つまり，計 48 本の試験片の強度の測定が実験計画されているが，このランダム化のメカニズムは次のように記述される．

```
<Randomize Id = "FirstStep" LongName = "第 1 段階のランダム化"
                                        といった説明文の ID の並び
          For = "(i in YarnDomain )" >
   <Randomized>
   SELECT Bobbin FROM Strength WHERE Yarn=i
   </Randomized>
   <From Size = "6" >
   BobbinDomain
   </From>
</Randomize>
<Randomize Id = "SecondStep" LongName = "第 2 段階のランダム化"
                                        といった説明文の ID の並び
          For = "(i in YarnDomain)(j in BobbinDomain)" >
   <Randomized>
   SELECT Samples FROM Strength
   WHERE (SELECT Bobbin FROM Strength WHERE Yarn=i)=j
   </Randomized>
   <From Size = "4" >
   SampleDomain
   </From>
</Randomize>
```

ここに現れたさまざまな変量についてさらに詳しく述べる余裕はないが，この実験計画は，入れ子状の実験計画 (nested design)，あるいは一般的に枝分かれ実験計画と呼ばれるものである．見かけは多段階層別抽出と見かけはまったく変わらないことに注意されたい． このように形式的に記述してみると，これまで独立に扱われてきた実験計画と標本調査をある程度統一的に扱うことができることがわかるであろう．

2.5.2 システマティックな抽出，意図的な抽出

標本調査では，ランダムに抽出することが煩雑になりすぎるため，その代わりに規則的に抽出するいわゆる**系統抽出** (systematic sampling) と呼ばれる抽出法がある．記述法は <Randomize> とほとんど同じであるが，属性 Rate が不要になる代わりにどのような規則でシステマティックに抽出したかの説明を与える属性 Pattern が新たに登場する．

```
<Systematic Id = ID 名
            LongName = 系統抽出の目的の説明の ID の並び
            Pattern = 抽出規則の説明の ID の並び
            For = "(i in 条件変数の変域を与える ID)" >
   <Sampled Distinct = "YES"または"NO" >
       選択文
   </Sampled>
   <From Size = 選択数>
       選択文
   </From>
</Systematic>
```

ここで，複数の ID が与えられようになっているのは，多言語対応のためである．また，ランダムでもなく系統的でもないが，過去の経験に基づく意図的な抽出 (purposive sampling) によって偏りをなくした標本抽出が行われることもある． この場合は，タグ名として Systematic の代わりに Purposive を用いる．ただし，その意図はさまざまであるので，属性 Pattern ではなく，属性 LongName で記述する．以上の 3 種類の組合せによって標本抽出や実験計画の基本的な記述が行える．

例 4

例 2 で取り上げた家計調査は総合的に次のように記述される．

```
<DataSampling Id = "HouseHoldSampling"
              LongName = "総務庁の家計調査の 3 段階無作為層別
                          抽出" といった説明文の ID の並び >
  <Randomize Id = "rCity" LongName = "第 1 段階の無作為抽出"
             For = "(i in CityStrataDomain)" >
    <Randomized Distinct = "YES" >
    SELECT City FROM HouseHoldSurvey WHERE Stratum=i
    </Randomized>
    <From>
    SELECT city FROM SamplingTable WHERE stratum=i
    </From>
    <Rate>
    SELECT Prob FROM SamplingTable WHERE stratum=i
    </Rate>
  </Randomize>
  <Purposive Id = "block"
             LongName = "過去の抽出と重複が起こらないように
                         調査ブロックを選ぶ" といった説明文の
                         ID の並び
             For = "(i in CityStrataDomain)(j in CityDomain)" >
    <Sampled Distinct = "YES" >
    SELECT Block FROM HouseHoldSurvey
    WHERE (SELECT City FROM HouseHoldSurvey
    WHERE Stratum=i)=j
    </Sampled>
    <From Size = "Unspecified" >
    BlockDomain
    </From>
  </Purposive>
  <Randomize  Id = "rUnit"
```

```
                LongName = "第 2 段階の調査単位の無作為抽出" と
                                いった説明の ID の並び
                For = "(i in CityStrataDomain)(j in CityDomain)
                        (k in BlockDomain)" >
        <Randomized Distinct = "YES" >
        SELECT Unit FROM HouseHoldSurvey
        WHERE (SELECT Block FROM HouseHoldSurvey
        WHERE (SELECT City FROM HouseHoldSurvey
        WHERE Stratum=i)=j)=k
        </Randomized>
        <From Size = "12" >
        UnitDomain
        </From>
    </Randomize>
    <Randomize Id = "third" LongName = "第 3 段階の無作為抽出"
                                        といった説明の ID の並び
            For = "(i in CityStrataDomain)(j in CityDomain)
                    (k in BlockDomain)(l in UnitDomain)" >
        <Randomized Distinct = "YES" >
        SELECT HouseHold FROM HouseHoldSurvey
        WHERE (SELECT Unit FROM HouseHoldSurvey
        WHERE (SELECT Block FROM HouseHoldSurvey
        WHERE (SELECT City FROM HouseHoldSurvey
        WHERE Stratum=i)=j)=k)=l
        </Randomized>
        <From Size = "6" >
        HouseHoldDomain
        </From>
    </Randomize>
</DataSampling>
```

これを見ると，単純に 3 段階無作為抽出といっただけでは説明しきれていない系統抽出も中間に存在することがわかる．無作為抽出は例 2 で述べた市町村を抽出する段階，ブロックを抽出する段階，調査世帯を抽出する段階で行われており，単位区を抽出する段階，開始月コードを定める段階は系統抽出である．また，ここには現れない調査員の割当ても系統的に行われている．これらすべては <Randomize> あるいは <Systematic> を用いて別途記述されている．

2.5.3 実験計画

実験計画で常に強調されるのは次のフィッシャーの “実験の 3 原則” である．

1. 反復 (repetition)
2. 無作為化 (randomization)
3. 局所管理 (local control)

この 3 原則は次のように説明されることが多い．同一条件での反復実験がなければ，偶然変動なのか条件の違いによる変動なのかが区別できないので，反復実験が必要であるというのが第 1 の原則であり，実験の目的以外の影響は，それらの原因を無作為化によって確率的な誤差に転化することによりモデルを単純化できるというのが第 2 の原則である．第 3 の原則は，実験全体を対象に無作為化して目的以外の影響を除くことが技術的あるいは経済的に困難な場合，実験をいくつかのブロックに分け，各ブロックごとに無作為化を行うとよい，ということである．もちろん，何種類かのブロックを導入すれば，複数の，目的以外の影響も無作為化できる．

ただし，ここでの**無作為化**は英語の randomize から類推されるような，先の <Randomize> で記述したような「ランダム化」だけを意味しているわけではないので，注意が必要である．たとえば,「完全無作為化実験計画」と呼ばれる計画は，すべての因子のすべての水準の組合せ（**処理**， treatment）での実験を同一回実施する実験計画である．確かに，この計画には，それをどのような順序で実験するかをサイコロを振ってきめるといった意味で，文字

どおり「ランダム化」の部分も含まれるが，処理をバランスさせて実験を実施することも含めて「無作為化」という一つの言葉で表現することが多いので誤解を生みやすい．このバランスさせる部分は処理の**釣合い** (balance) というべき部分である．釣合いがとれていれば，後の線形モデルによる解析の段階で係数の推定量が互いに無相関となり，各因子の効果を分離できるのでこの性質が重視される．

そこで， DandD では，フィッシャーの 3 原則の最初の**反復**は，反復を表す変量のデータベクトルに型属性として `Number` を与えることにより記述し，どのような「ランダム化」で実験順序などを確率的な誤差に転化しているのかは，要素 `<Randomize>` で記述する．**局所管理**は，まず要素 `<Randomize>` とデータベクトルの因子属性の組合せで記述する．この実験計画の項では，この計画がどのような釣合いの取れた実験なのか，どのような**交互作用** (interaction) のある実験なのかを記述することによってフィッシャーの原則，**無作為化**を表現する．

実験が計画通り実施されている限り，釣合いや交互作用は，各データベクトルに各因子の水準がどのように出現しているかを調べれば分かることであるが，計画の意図を明示するためにも，また，必ずしも計画どおり実施できないこともあることを考慮すれば，形式的に記述しておくことが望ましい．このことが，後の解析に用いるモデルを規定し，その精度の評価も与えることにつながることは，以下に掲げる歴史的に積み重ねられてきたさまざまな名前のついた実験計画と，それに基づく解析を参照すればわかるであろう．

1. 実験全体を無作為化の対象とする**完全無作為化実験計画** (completely randomized design)
   (a) その中でも，すべての処理，つまり対象となる因子の水準の可能な組合せを同一回数実験する**要因実験計画** (factorial design)
   (b) 一部の処理についてしか実験を実施しない**一部実施要因実験計画** (fractional factorial design)
   (c) 一部実施要因実験計画の中でも，実施する処理については同一回数実験する，つまり釣合いを保ちながら実験する**釣合い型一部実施要因実**

験計画 (balanced fractional factorial design)

(d) 釣合い型一部実施要因実験計画の中でも，直交配列を用いた**直交型一部実施要因実験計画** (orthogonal fractional factorial design)

2. 1 種類のブロックを導入したブロック実験計画

(a) 各ブロックの中で，すべての処理が 1 回だけ実施される**完備ブロック実験計画** (complete block design)，**乱塊法** (randomized block design) とも呼ばれる

(b) 各ブロックの中で一部の処理しか実験しない**不完備ブロック実験計画** (incomplete block design)

(c) 不完備ブロック実験計画の中でも，実験全体ではすべての処理が同一回実験されており，異なる処理が同じブロックで実験される回数つまり**会合数**が一定である**釣合い型不完備ブロック実験計画** (balanced incomplete block design, BIBD)

(d) 釣合い型ブロック実験計画の会合数の条件を緩めた**部分釣合い型不完備ブロック実験計画** (partially balanced incomplete block design, PBIBD)

3. 2 種類のブロックを導入したブロック実験計画

(a) 完備なブロックを組み合わせる**ラテン方格法による実験計画** (Latin square design)

(b) 完備なブロックと釣合い型ブロックを組み合わせる**ユーデン方格法による実験計画** (Youden square design)

(c) 完備なブロックと部分釣合い型ブロックを組み合わせる**シュリカハンデ方格法による実験計画** (Shrikahande square design)

この実験計画を記述する項目は次のような構成をとる.

`<ExperimentalDesign Id =` ID 名 `LongName =` 実施された実験計画に関する総合的な説明の `ID` の並び `>`

`<Balance What =` 釣合いのとれた因子の組合せ，あるいは会合数の等しい処理番号の組の並び

`Block =` 実験されるブロックの `ID` またはその並び

```
            Treatment = 処理の番号を与えるデータベクトルの ID
            Lambda = 会合数 />
    <Interaction What = どのような交互作用があるかの記述 />
</ExperimentalDesign>
```

まず，ブロックが存在しない場合には，<Balance> の属性 What に釣合いが取れるように計画された因子の並び，あるいはその組を列挙することによって記述する．ある因子のすべての水準が同一回出現するように実験されていれば，その因子に関しては 1 次の釣合いが取れていると言い，二つの因子のすべての水準の組合せが同一回数出現するように実験されていればそれらの因子は 2 次の釣合いが取れていると言う．以下同様に 3 次，4 次，··· と，より高次の釣合いも定義されるが，当然，高次の釣合いから，より低次の釣合いは導かれる．1 次の釣合いはその因子の対応するデータベクトルの ID を並べることによって表し， 2 次以上の釣合いは (A B) のようにカッコで囲んで表す．たとえば，

```
<Balance What = "(A B)(A C)D" />
```

は因子 A と B，A と C に関しては 2 次のバランスをし，D に関しては 1 次のバランスをしていることを表している．

ブロックが存在する場合には，因子の組合せよりもさらに細かく釣合っているかどうか，つまり会合数が一定であるかどうかが問題となる．部分釣合い型の場合のように，特定の処理の組の実験だけに注目して同一ブロックで実施される回数が等しいかどうかを問題にする実験計画もあるので，一般的に記述するためには，要素 <Data> に処理の定義を与える関係形式を作り，その各行の識別番号ベクトルの ID を Treatment に与え，その番号の組を What に並べることによって，会合数の等しい処理の組を表現する．また，Block には注目するブロック因子（データベクトル）の ID あるいはその並びを与え，Lambda には会合数を与える．たとえば，

```
<Balance What = "(1 4) (2 5) (3 6)"
         Block = "A"
         Treatment = "treatment"
```

```
Lambda = "4" />
```

は，treatment で規定された処理番号 1 と 4，2 と 5，3 と 6 は同じ会合数 4 であることを示している．異なる会合数が存在する場合は， Balance を複数用いて記述すればよい．

各因子の因子属性についての記述? 2002-07-04

交互作用の存在は， A:B:C のような一般的な記法で表現する．たとえば

```
<Interaction What = "A:B A:B:C" />
```

は A と B の間の交互作用と， A, B, C の間の 3 次の交互作用が存在することを記述している．もちろん，これだけで記述しきれないことがあるが，それは属性 LongName に文章による説明を与えることによって補足する．

2.6 背景情報

前節まで説明してきたデータの背景情報は，次のように形式的な記述ができる背景情報だけであった．

1. 各データベクトルの属性
2. 時系列，グラフなど関係形式あるいは配列形式の分類
3. 実験計画，標本調査，打切りなどデータ取得の環境

これらの記述に関しても，汎用性に重点を置いて形式化を行っているので，どうしても記述しきれない部分が残る．そこで， 1. と 2. に関しては，要素 <DataVector> と要素 <Data> の各要素に対して，追加的な属性 Explanation を用意して，そのような部分の記述を自由にできる余地を残してある．この属性には，要素 <Appendix> の子要素 <Text> の ID を与えるように DTD で定めてあるので，単純な説明文ではなく，ある程度構造をもった背景情報の記述もできる．また，このように追加的な属性は ID で参照する形をとっているので同一の属性を重複し定義する必要はなく，複数のデータベクトルや関係形式，配列形式などで共用できる．

さらに，次のような全体的な記述は要素 <BackGround> にまとめて記述することにしている．

1. 改訂の記録
2. 参考文献

これ以外の，記述しきれない背景情報は部分要素 <Introduction> の Titile 属性に要素<Appendix>の子要素の ID を与える形で自由文形式で記述することにしている．今後，形式的な記述が可能であることが判明したものついては順次，追加する予定である．

2.6.1 改訂の記録

データを取得後も，校正 (calibration) や浄化 (cleaning) を行ったり，冗長な情報を削除したり，不足するデータを追加したり，再組織化を行うことがあるが，その場合，もとの DandD インスタンスはそのままに保存して新しい DandD インスタンスをつくる方がよい．その際，もとの DandD インスタンスの存在場所やどのような改訂を施したのかという情報，つまり，いかにデータが**変容** したかを記述しておく必要がある．

DandD ルールではデータの変容を Relatives として記述する．いわば，DandD インスタンスの親子，兄弟の関係である．この要素で記述すべき背景情報は，どのような変更がデータに加えられたのか，どのデータベクトルが継承されたのかである．

なお，Relatives は，単なる DandD インスタンス同士の関係を示すだけなく，従来の解析ソフトウェアでは困難であったデータ解析の履歴を示すという側面もある．

```
<Relatives>
    <Parent Id = ID 名 URL= 親となるインスタンスの URL>
      <Inherit RefId= このインスタンスにおけるデータベクトルの ID 名
               ParentId = 親インスタンスでのデータベクトルの ID 名
               ParentLongName = 親インスタンスでのデータベクトル
               の LongName />
      <Inherit RefId= このインスタンスにおけるデータベクトルの ID 名
               ParentId = 親インスタンスでのデータベクトルの ID 名
```

```
            ParentLongName = 親インスタンスでのデータベクトル
            の LongName />
      ...
    </Parent>

    <Child Id = ID 名 URL = 子となるインスタンスの URL >
      <Inherit Id= 継承させるデータベクトルの ID 名>
        継承させるデータベクトルの作成手順
      </Inherit>
      <Inherit Id= 継承させるデータベクトルの ID 名>
        継承させるデータベクトルの作成手順
      </Inherit>
   ...
</Child>

<Brother Id = ID 名 URL = 兄弟となるインスタンスの URL>
</Relatives>
```

現行の DandD ルールでは「継承させるデータベクトルの作成手順」として自由文による記述を許している．しかし，汎用性の面からも形式的な記述ができたほうが都合がよい．現在，DandD プロジェクトでは SQL を用いた形式的な記述を検討している段階である．

2.6.2 参考文献

データの背後にある現象に関するこれまでの知見，類似のデータに関する解析やモデル化の報告など，さまざまな参考書 (Book)，論文 (Article)，さらには電子媒体など (Other) が存在する．それらについて記述する．以下は，標準的な記述方式に従っている．

```
<Reference RefId = 参考文献参照リストの ID の並び />
```

```
<Appendix>
<RefList Id = 参考文献リストの ID
         RefId = 参考文献の ID の並び />
   <Book Id = ID 名 >
     <author> 著者名 </author>
      <year> 発行年 </year>
      <title> 表題 </title>
      <publisher> 出版社 </publisher>
      <address> 出版地 </address>
   </Book>
   <Article Id = ID 名 >
      <author> 著者名 </author>
      <year> 発行年 </year>
      <title> 論文題目 </title>
      <journal> 雑誌名 </journal>
      <volume> 巻 </volume>
      <number> 号 </number>
      <pages> ページ </pages>
   </Article>
   <Other Id = ID 名 >
      <author> 著者名 </author>
      <year> 発行年 </year>
      <title> 表題  </title>
      <access> アクセス方法 </access>
    </Other>
</Appendix>
```

2.7 外部データ

この部分の記述新設； 2002-07-01
この部分の記述改訂； 2003-11-15

DataVector の値が空ならば，そのデータベクトルの実体は外部に存在することになるが，この場合，その外部データへのアクセス方法をはじめ，DandD ルールと整合性を保つために必要な処理などが，そのデータベクトルの属性として与えられている必要がある．これらの属性が与えられていなければ，すべて欠損値 NA のベクトルであると見なす．これは，組織的にある期間のデータが欠けていて，データベースにも存在しない場合を想定するためである．ただしこの場合，最低限 Length 属性と MissingType 属性は与える必要がある．

この部分追加: 2002-09-05
この部分の改訂: 2003-11-15

現在のところ，外部データとしては，関係形式データベースに存在するデータか，それと同等な扱いができる，ファイル中のデータに限っている．今後のインターネットやソフトウェアの進歩を踏まえると，DandD ルールが対応すべき外部データの種類は増えていくにちがいない．DandD ルールでは外部データを扱う手続きを次の 3 つに切り分け，データベクトルの属性としてあたえている．このように切り分けることで，対応する外部データの種類が増えてもいずれかの手続きを追加するだけですむ．

- 外部データへのアクセス情報 (Access)
- 取得のためのプロトコル情報 (Protocol)
- 取得後，データベクトルに加工するための処理情報 (PostProcessing)

最初の 2 つの属性には <Appendix> におかれた同一名の要素の ID を与え，PostProcessing には <Appendix> におかれた <Code>, <ScanFormat>, <Arithmetic> の ID をその処理順に与える．

この部分の記述追加； 2003-11-15

2.7.1 Access

<Access> は，ネットワーク上での外部データの所在とアクセス条件を示す要素で, <Appendix> の下に置かれる．

<Access Id= ID 名 IP = 外部データの IP アドレス UserId = ユーザ名 Password = パスワード />

IP は文字通り，実体の所在を示す IP アドレスである．UserId はデータベースシステムや FTP サイトなどへログインするためのユーザ名で，Password はそのログインパスワードになる．もちろん UserId や Password は省略可能な属性であり，その場合 UserId のみデフォルトの値 anonymous が設定される．

この部分の記述追加； 2003-11-15

2.7.2 Protocol

<Protocol> 要素は文字コードを示す Encoding とトランスポート層のプロトコルを指定する Physical 属性を持つ．この情報と <Access> の情報を組み合わせて，TCP/IP といったネットワークでの物理的な通信方法を定めている．データベースやファイルでの日本語コードが統一されているとは期待できないので，Encoding で，そこで用いられている言語コードを記述しておく．指定できる言語コードは，EUC_JP, SJIS などDTD に定義されているが，省略時は UTF-16 である． したがって，ASCII だけの場合にも明示的にこの属性を与える必要がある． RDB の場合にはデータベース構築時にあらかじめ日本語コードが指定されていることが多いのでそれを指定する．

http://java.sun.com/j2se/1.3/ja/docs/ja/guide/intl/encoding.doc.html

を参照されたい．

現在，DandD ルールでは Protocol の子要素としてアプリケーションレベルの通信プロトコルを示す <HTTP> や <FTP>, さらにデータベース通信のための<JDBC> が用意されている．

FTP で取得する外部データ

```
<Protocol Id= ID 名 Encoding = エンコーディング名 Physical = TCP または UDP >
  <FTP Suffix= ファイルの絶対パス Offset = オフセット数
    Lines = ファイルにおける実体の行数 />
</Protocol>
```

Suffix は URL でいうところのファイルパス部に相当し，Access 要素の IP 属性と組み合わせることでファイルを一意に定めることができる．Offset, Lines はファイル内のヘッダーのようなデータ以外の情報を取り除くための属性である．

HTTP で取得する外部データ

<Protocol Id= ID 名 Encoding = エンコーディング名 Physical = TCP または UDP >
  <HTTP Suffix= ファイルの絶対パス Offset = オフセット数
    Lines = ファイルにおける実体の行数 />
</Protocol>

のように記述する．<HTTP> の属性は <FTP> のそれと同様である．

データベースに存在する外部データ

データベースにある外部データの記述は，Protocol 以下に

<JDBC DatabaseName= データベース名
      DatabaseServerType = データベースシステムの種類>
      SQL を利用した問い合わせ
<JDBC/>

のように記述する．DatabaseServerType として指定できるデータベースシステムについては，最新の DTD を参照されたい．

なお，JDBC は Java DataBase Connectivity の略で，Java 言語を用いてデータベース通信を行うためのフレームワークである．後述の DandD サーバは現在 Java で実装しており，この仕掛けを用いてデータベースサーバからデータを取得している．

この部分の記述追加；2003-11-15

2.7.3 PostProcessing

PostProcessing は外部データを取得後，データベクトルとして埋め込むことができるような処理を施すための属性である．

PostProceessing = 取得後の処理の ID の並び

現在，用意されている処理は <Code>, <ScanFormat>, <Arithmetic> の 3 つであり，後述の DandD サーバはこれらの要素の ID の並び順にデータベクトルに処理を施す．なお，以下で説明する verb+¡Code¿+ はデータベクトルのコーディングに用いられる<Code> と同様の記述方法だが，そのセマンティックスは異なり，コーディングされた外部データを数値であるデータベクトルに変換するために用いられる．

Code

DandD ルールでは，データベクトルの値としては数字のみを許し，文字列の場合は コード属性によって対応関係を示すことになっているが，外部データの場合には必ずしもこのルールに従っているとは限らない，むしろデータベースでは文字列のまま貯蔵されているのが普通である．また，逆に数値であっても型属性まで調べればわかるが，SELECT 文で取り出したときは，あくまでも文字列でしかない．また，型属性は必ずしも統一されていないので，これに頼るのも危険であるし，欠損値を NA で表したときには数値型ではありえない．DandD インスタンスでは，PostProcessing 属性が<Code> を参照しているか否かで，文字列か数値かを判断する． したがって，外部データで数値でなければ，この要素は必須である．

ただし，外部データの場合は SELECT 文で一部だけ取り出す可能性も考えるとコード属性を一意的に与えることが困難であるので，

```
<Code Id= ID 名 />
```

のように参照先が空であることも許す．これは属性を与えないこととは意味が異なり，文字列データではあるものの，データが追加される可能性があるのでコードの種類が確定しないということを表している．Domain を共有し

2002-08-05
Id 名など, すべてがコードになる場合の扱い? 2002-07-22
省略時の扱い; 2002-07-19, 2003-07-25 修正

ているような場合も，一貫性を確保するためには，この Domain を定義するデータベクトル自身の Code 属性を空としておく必要がある．

なお，3 章で述べるの DandD サーバは，このデータへのアクセスがあった段階で実際に現れた相異なる文字列からこの属性を作り出すとともに，それに応じたコーディングをおこなって，その結果である整数値の並びを返す．クライエントは，これを受け取ったあと，もう一度<Code> をサーバに問い合わせることによって，DandD サーバの作り出したコード属性を知ることができる．この場合， 当然データベクトルには Code 属性を与えてはいけない．

一方，この属性が与えられていて空でないときは，Code 属性の存在によって扱いが異なる．明示的に Code 属性も与えられていれば，PostProcessing 属性が参照する <Code> 要素とその並びの順に対応をつけ，水準のマッピングを行い，その結果である整数値の並びを返す．したがって，複数のコードでの表現も可能であり，外部データでの表現と違ったコーディングに変換することが可能となる．異なるデータベース間の表現の違いを統一するのにも，多国語対応にも役立つ．

ScanFormat

CSV のようなデータファイルに存在する外部データでは，各行の一部分だけがデータベクトルの値になることがほとんどである．また，データベースにある日付型のデータでも，月だけが必要になるということも考えられる．このような取得後のデータの一部を取り出す仕掛けが ScanFormat である．ScanFormat に与える書式は C 言語の scan 関数に対する書式と同じで，たとえば"%*d-%d-%*d" で, 2002-07-01 のような形式の日付表現のうちの月だけを数値として取り出すことができる．もちろん，01 のように頭の 0 は読み飛ばされるのは C 言語の場合とおなじである． また，"%*d %d %*d" のように空白を置いた場合には対象の文字列中の任意個の空白文字 (空白，タブ, 復帰, 改行) に対応するのも C 言語と同じである．ただし， 欠損値を含んでいる場合 2002-NA-NA のような表現も許すので， %d は十進数だけでなく NA も含む表記と解釈する．

同様に 2002-Feb-01 のような形式の場合，書式"%*d-%s-%*d" で，月だけを取り出すことができるが，この場合も， 2002-NA-01 のような表現が現れる可能性を考慮する必要がある．当然のことながら， %s で読んだ場合は，文字列であるので，PostProcessing 属性は <Code> 要素を参照している必要がある．関係形式データベースには， Date 型などが定義されているにもかかわらず，あえてこのような機能を実装したかというと，このような型は SQL の標準ではなくデータベースサーバに依存する点，また，データベースサーバ以外ではこのような機能はサポートされていないのが普通だからである．

NA に関する注意追加; 2002-07-11

Date 型などに関する記述追加 2002-10-25
Arithmetic 2003-11-16

Arithmetic

西暦などのデータでは，繰り返しを避けるために下 1, 2 桁 だけを記述しておくことがある．また，データが複数のファイルやデータベースに分散している場合では，データごとに単位が異なるといった問題も生じる．これらの数値にまつわる問題は演算を導入することで解決できる．たとえば，前者の西暦の問題は 2000 などの数値を加えればよいし，後者も乗算，除算だけで解決できることがほとんどである．DandD ルールでは演算処理のための要素 <Arithmetic> が用意されている．

```
<Arithmetic Id = ID 名 > x + 2000 </Arithmetic>
```

ここで x はデータベクトルの各値が代入される．DandD サーバが対応している演算, 関数としては

- 四則演算 (\+,-,*,/)．例 x/200
- 三角関数 (sin,cos,tan)．例 sin(x)
- 自然対数 (log), 指数 (exp)．例 exp(2*x)
- べき乗 (pow)．例　pow(x,2) (これは x^2)
- 根号 (sqrt)．例 sqrt(x)

がある．また，演算の優先順位を定めるための括弧 (,)， も利用することができる．

2.8 外部エンティティティ

XML の機能の一つである外部エンティティーは， Appendix 中の Text だけで許される．

追加，未完成; 2002-07-04

2.9 多国語対応

多国語対応のため， DandD 関係はすべて UTF-16 を共通な文字コードとする．また，言語の切り替えを可能なように，どの要素であっても，以下に掲げる属性には半角スペースで区切って複数の ID を与えられる．順に， Title の Language 属性に与えられた言語識別文字列に応じた言語で記述されていることを表すものとする．単一の ID しか与えられていない場合は，言語によらず共通な属性であるとみなす． この場合でも，この Language 属性は省略できない．一方， Language に複数の ID が与えられているにも関わらず，単一の ID しか与えられていない属性については，その ID を繰り返し用いる．言語識別記号としては，English, Japanese, French Spanish などの文字列を自由にもちいてよい．同じ言語でも複数の表現がありうる．特に Code では「ひらがな」,「カタカナ」などの可能性もある．

2002-11-28 追加

2002-10-11， 2002-11-25 追加訂正

LongName : Appendix の Name タグ要素の ID の並び
Code : Appendix の Code タグ要素の ID の並び
Title : Appendix の Text タグ要素の ID の並び
Definition : Appendix の Text タグ要素の ID の並び
Explanation: Appendix の Text タグ要素の ID の並び
PopulationDefinition: Appendix の Text タグ要素の ID の並び
Unit: Appendix の Text タグ要素の ID の並び
UnitDefinition: Appendix の Text タグ要素の ID の並び
Reference: Appendix の RefList タグ要素の ID の並び

Appendix の Text タグを Name タグに， Explanation タグを Text タグに変更, 2002-10-25

2.10 DandD インスタンス

DandD インスタンスは文字コードとして UTF-16 を用いた XML ファイルであるので， 先頭は

```
<?xml version="1.0" encoding="UTF-16"?>
 <!DOCTYPE DandD SYSTEM "DandD_2.1.0.dtd">
```

のようにはじめる．1 行目は XML の Version と用いる文字コードを宣言し，2 行目はこのインスタンスが従う DTD を宣言しており，サーバはこの宣言に応じた処理をおこなう．このあとは<DandD> で開始し， </DandD> で閉じる．XML の規約では， DTD はインスタンスと同じ場所に存在しなければならないので，存在しなければエラーとなる．ただし，クライエントサーバシステムで，サーバにインスタンス取得まで依頼しせずクライエントからインスタンスを送った場合, つまりローカルなインスタンスファイルをブラウザで眺めるようなときは例外的で，現在のところインスタンスとともに存在する DTD は参照されない．そのサーバの作業領域に存在する DTD が用いられるので　DTD が存在しなくてもエラーとはならない．ネットワークに接続されていないときも考慮すると上記のように記述しておいたほうがよいが，ネットワーク接続を前提とすれば， URL で記述しておけば，DTD の存在は気にする必要がなくなる．

Appendix の Text タグ中で，図などを参照する場合には DOCTYPE 宣言に次のようにして Unparsed objects の宣言を追加する． ファイルの存在場所は URL で記述する必要がある．

```
<!DOCTYPE DandD  SYSTEM
"DandD_2.1.0.dtd"[
<!ENTITY Mycar SYSTEM
"http://www.stat.math.keio.ac.jp/DandDII/Examples/MyCar.JPG" NDATA JPG>
<!ENTITY 愛車 SYSTEM
"http://www.stat.math.keio.ac.jp/DandDII/Examples/MyCar.JPG" NDATA JPG>
<!NOTATION JPG SYSTEM "DandDBrowser">
]>
```

現在のところ扱える画像ファイルは，GIF, JPG である．このようにして宣言した Unparsed Object は Text 中<Unparsed file="Mycar"/> あるいは <Unparsed file="愛車" /> のように引用できる．

2003-07-25 追加

2.11 DTD

```
<!-- DandD.dtd Version 2.1.0  -->

<!ELEMENT DandD ( Title, BackGround, DataSampling?, Data,
                        Analysis*,  Model*, DataBody, Appendix) >


<!-- Title -->

<!ELEMENT Title EMPTY >
<!ATTLIST Title Title    IDREFS #REQUIRED >
<!ATTLIST Title Language CDATA  #REQUIRED  >
        <!-- Language: a list of N language names for IDREFS, e.g.
                 "English Japanese French" -->
        <!-- Most of IDREFS in the following except IDREFS to
              <DataVector> should have N Id's -->
         <!-- Title: Should Refer N <Text>(s) in <Appendix> -->

<!--Back Ground-->

<!ELEMENT BackGround (Introduction?, Relatives?, Reference?) >

        <!ELEMENT Introduction  EMPTY >
        <!ATTLIST Introduction  Id    ID     #IMPLIED
                                Introduction IDREFS #REQUIRED >
        <!-- Introduction: Should Refer N <Text>(s) in <Appendix> -->

        <!ELEMENT Relatives (Parent | Child | Brother|Inherit) >

             <!ELEMENT Parent (Inherit)* >
             <!ATTLIST Parent Id ID #REQUIRED
                              URL CDATA #REQUIRED>
             <!ELEMENT Child (Inherit)* >
             <!ATTLIST Child Id ID #REQUIRED
                             URL CDATA #REQUIRED>
             <!ELEMENT Brother EMPTY >
             <!ATTLIST Brother Id ID #REQUIRED
                              URL CDATA #REQUIRED>

             <!ELEMENT Inherit EMPTY>
```

```
            <!ATTLIST Inherit RefId IDREF #REQUIRED
                              ParentId IDREF #IMPLIED
                              ParentLongName IDREFS #IMPLIED>

        <!ELEMENT Reference EMPTY >
        <!ATTLIST Reference RefId IDREFS #REQUIRED >
        <!-- Reference: Should Refer N <RefList>(s) in <Appendix> -->

<!--Data Sampling-->

<!ELEMENT DataSampling (Randomize | Systematic | Purposive |
                        ExperimentalDesign )* >

        <!ELEMENT Randomize (Randomized, From, Rate?) >
        <!ATTLIST Randomize     Id         ID      #REQUIRED
                                LongName   IDREFS  #IMPLIED
                                For        CDATA   #REQUIRED >
        <!-- LongName: Should Refer N <Text>(s) in <Appendix> -->


                <!ELEMENT Randomized (#PCDATA) >
                <!ATTLIST Randomized Distinct  (YES | NO)  "YES" >

                <!ELEMENT From  (#PCDATA) >
                <!ATTLIST From   Size   CDATA  #IMPLIED >

                <!ELEMENT Rate (#PCDATA) >

        <!ELEMENT Systematic (Sampled, From) >
        <!ATTLIST Systematic    Id         ID      #REQUIRED
                                LongName   IDREFS  #IMPLIED
                                Pattern    CDATA   #REQUIRED
                                For        CDATA   #REQUIRED >
        <!-- LongName: Should Refer N <Text>(s) in <Appendix> -->


                <!ELEMENT Sampled (#PCDATA) >
                <!ATTLIST Sampled Distinct   (YES | NO)   "YES" >

        <!ELEMENT Purposive (Sampled, From)>
        <!ATTLIST Purposive     Id         ID      #REQUIRED
                                LongName   IDREFS  #IMPLIED
                                For        CDATA   #REQUIRED >
        <!-- LongName: Should Refer N <Text>(s) in <Appendix> -->


        <!ELEMENT ExperimentalDesign (Balance*, Interaction) >
```

```
        <!ATTLIST ExperimentalDesign    Id       ID        #REQUIRED
                                        LongName IDREFS        #IMPLIED >
        <!-- LongName: Should Refer N <Text>(s) in <Appendix> -->


                <!ELEMENT Balance     EMPTY>
                <!ATTLIST Balance       What      CDATA   #REQUIRED
                                        Block     IDREFS  #IMPLIED
                                        Treatment IDREF   #IMPLIED
                                        Lambda    CDATA   #IMPLIED >
                    <!-- Block, Treatment: Should Refer <DataVector>(s) -->

                <!ELEMENT Interaction EMPTY>
                <!ATTLIST Interaction   What       CDATA  #REQUIRED >


<!--Data-->

<!ELEMENT Data  ( Relational | Array | Graph | TimeSeries | PointProcess)* >

        <!ELEMENT Relational        (Value+)  >
        <!ATTLIST Relational    Id          ID     #REQUIRED
                                LongName    IDREFS #REQUIRED
                                Explanation IDREFS #IMPLIED >
        <!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
        <!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->

                <!ELEMENT Value EMPTY >
                <!ATTLIST Value RefId  IDREFS  #REQUIRED >
                <!-- Should refer Datavector IDs, which are concatinated -->
                <!ATTLIST Value Systems  IDREFS  #IMPLIED >

        <!ELEMENT Array (Axis+, Value+) >
        <!ATTLIST Array        Id           ID     #REQUIRED
                               LongName     IDREFS #REQUIRED
                               Explanation  IDREFS  #IMPLIED >
        <!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
        <!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->

                <!ELEMENT Axis EMPTY >
                <!ATTLIST Axis RefId   IDREF    #REQUIRED >

        <!ELEMENT Graph (Axis+, Value+) >
        <!ATTLIST Graph        Id           ID     #REQUIRED
                               LongName     IDREFS #REQUIRED
                               Explanation  IDREFS  #IMPLIED >
        <!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
```

```
        <!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->


        <!ELEMENT TimeSeries (Condition*, TimeAxis,  Value+) >
        <!ATTLIST TimeSeries    Id           ID     #REQUIRED
                                LongName     IDREFS #REQUIRED
                                Explanation  IDREFS #IMPLIED >
        <!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
        <!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->

                <!ELEMENT Condition EMPTY >
                <!ATTLIST Condition RefId    IDREFS #REQUIRED >
                <!ELEMENT TimeAxis EMPTY>
                <!ATTLIST TimeAxis RefId IDREFS #REQUIRED >


        <!ELEMENT PointProcess (Condition*, Point?, TimePoint?, Mark*) >
        <!ATTLIST PointProcess  Id           ID     #REQUIRED
                                LongName     IDREFS #REQUIRED
                                Explanation  IDREFS #IMPLIED >
        <!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
        <!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->



                <!ELEMENT Point EMPTY >
                <!ATTLIST Point RefId     IDREFS #REQUIRED >

                <!ELEMENT TimePoint EMPTY >
                <!ATTLIST TimePoint RefId   IDREFS #REQUIRED >

                <!ELEMENT Mark EMPTY >
                <!ATTLIST Mark RefId  IDREFS #REQUIRED >


<!--Analysis, Model-->

        <!ELEMENT Analysis (Unparsed) >
        <!ATTLIST Analysis          Id     ID     #REQUIRED
                                    System CDATA  #IMPLIED >

        <!ELEMENT Model (Unparsed) >
        <!ATTLIST Model         Id        ID     #REQUIRED
                                System    CDATA  #IMPLIED >

<!--Data Body-->
```

```
<!ELEMENT DataBody  (DataVector)* >

<!ELEMENT DataVector (#PCDATA)* >
<!ATTLIST DataVector    Id                ID           #REQUIRED
                        LongName          IDREFS       #IMPLIED
                        Explanation       IDREFS       #IMPLIED

                        Precision         CDATA        #IMPLIED
                        Accuracy          CDATA        #IMPLIED
                        Domain            IDREF        #IMPLIED

                        Missing           IDREF        #IMPLIED
                        MissingType       IDREF        #IMPLIED

                        Invalid           IDREF        #IMPLIED
                        InvalidType       IDREF        #IMPLIED

                        Truncation        IDREF        #IMPLIED
                        TruncationType    IDREF        #IMPLIED
                        TruncationLeft    CDATA        #IMPLIED
                        TruncationRight   CDATA        #IMPLIED

                        CensoringTime     CDATA        #IMPLIED
                        CensoringNumber   CDATA        #IMPLIED
                        RandomCensoring   IDREF        #IMPLIED
                        RandomCensoringType IDREF      #IMPLIED

                        Transform         IDREFS       #IMPLIED
                        Code              IDREFS       #IMPLIED
                        Length            CDATA        #IMPLIED

                        Unit              IDREFS       #IMPLIED
                        UnitDefinition    IDREFS       #IMPLIED
                        Scale ( Ratio | Interval | RatioInterval) #IMPLIED

                        DataType ( SpecifiedValues | EquallySpacedValues |
                                   Score | IntervalClass | Count |
                                   Category | OrderedCategory |
                                   Logical | Id | Number | Sequence |
                                   Images )  #IMPLIED

                        Factor (Controllable | Nominal | Variable | Block |
                                Environmental | Auxiliary ) #IMPLIED

                        PopulationType (Stratum | Cluster) #IMPLIED
                        PopulationDefinition  IDREFS  #IMPLIED
```

```
                        Access    IDREF #IMPLIED
                        Protocol  IDREF #IMPLIED
                        PostProcessing  IDREFS #IMPLIED
                        >

        <!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
        <!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->


        <!--   Transform:  Should Refer <Transform>(s) in <Appendix>, and
                the order of Transform must be in order of transforming. -->
        <!--   Code: Should Refer N <Code>(s) in <Appendix>. This defines a
                     mapping.-->

        <!--  Acess: Refer <Acess> in <Appendix> -->
        <!--  Protocol: Refer <Protocol> in <Appendix> -->
        <!--  PostProcessing:  Refer <ScanFormat>, <PrintFormat>, <Arithmetic>
                               or <Media> in <Appendix>,  Processed in the order of the
                               referred in case of outside Data. It is regraded as the
                               Server and the body of <Code> be replaced by
                               the generated one-->


<!-- Appendix -->

<!ELEMENT Appendix (  Name | Text | Code | Ref | Unparsed |
                      RefList | Book | Article | Other |
                      Transform | Acess | Protocol |ScanFormat |
                      PrintFormat | Arithmetic | Media |
                      MainKey | Constraint | Interval | Time |
                      Sexagesimal | OrthogonalCoordinate |
                      PolarCoordinate | CylindricalCoordinate |
                      SphericalCoordinate | LongitudeLatitude )*>

<!--    Name: Plain text used for name
        Text: Text with links(refer) or unparsed objects
        Code: Quoted Strings -->

        <!ELEMENT Name  (#PCDATA) >
        <!ATTLIST Name Id ID #REQUIRED >

        <!ELEMENT Text (#PCDATA|Ref|Unparsed)* >
        <!ATTLIST Text  Id ID #REQUIRED >

        <!ELEMENT Code (#PCDATA)* >
        <!ATTLIST Code Id       ID      #REQUIRED
                       Length   CDATA   #REQUIRED >
```

```
        <!-- Code is a sequence of quoted strings, seprated by a space -->

        <!ELEMENT Ref EMPTY>
        <!ATTLIST Ref   RefId  IDREFS #IMPLIED
                        URL    CDATA #IMPLIED >
        <!-- This is used in a Text for referring an object inside or outside -->

        <!ELEMENT Unparsed EMPTY>
        <!ATTLIST Unparsed file ENTITY #REQUIRED>


    <!-- Reference List -->

        <!ELEMENT RefList EMPTY >
        <!ATTLIST RefList       Id    ID      #REQUIRED
                                RefId IDREFS  #REQUIRED >

        <!-- RefList: organized for each languge and given IDREFS to <Book>
                                        <Article> or <Other>  -->

        <!ELEMENT Book (author, year, title, publisher, volume?,
                        number?, series?, address?, edition?, month?) >
        <!ATTLIST Book    Id   ID    #REQUIRED >

        <!ELEMENT Article (author, year, title, journal, volume?,
                                number?, pages, month?) >
        <!ATTLIST Article Id   ID    #REQUIRED >

        <!ELEMENT Other (author, year, title, access) >
        <!ATTLIST Other   Id    ID    #REQUIRED >

                              <!ELEMENT author    (#PCDATA) >
                              <!ELEMENT year      (#PCDATA) >
                              <!ELEMENT title     (#PCDATA) >
                              <!ELEMENT publisher (#PCDATA) >
                              <!ELEMENT volume    (#PCDATA) >
                              <!ELEMENT number    (#PCDATA) >
                              <!ELEMENT series    (#PCDATA) >
                              <!ELEMENT address   (#PCDATA) >
                              <!ELEMENT edition   (#PCDATA) >
                              <!ELEMENT journal   (#PCDATA) >
                              <!ELEMENT pages     (#PCDATA) >
                              <!ELEMENT month     (#PCDATA) >
                              <!ELEMENT access    (#PCDATA) >

    <!-- End of Reference List Definitions -->
```

```
<!-- Transform -->

      <!ELEMENT Transform (Linear | Log | BoxCox | Cut |
                           Power | Ratio | Index)* >

            <!ELEMENT Linear EMPTY >
            <!ATTLIST Linear        Id           ID    #REQUIRED
                                    Location     CDATA #IMPLIED
                                    Scale        CDATA #IMPLIED
                                    OriginalUnit CDATA #IMPLIED>

            <!ELEMENT Log EMPTY >
            <!ATTLIST Log           Id           ID    #REQUIRED
                                    Base         CDATA #REQUIRED
                                    OriginalUnit CDATA #IMPLIED >

            <!ELEMENT BoxCox EMPTY >
            <!ATTLIST BoxCox        Id           ID    #REQUIRED
                                    Lambda       CDATA #REQUIRED
                                    OriginalUnit CDATA #IMPLIED >

             <!ELEMENT Cut EMPTY >
             <!ATTLIST Cut           Id           ID    #REQUIRED
                                     Breaks       CDATA #REQUIRED
                                     OriginalUnit CDATA #IMPLIED >

             <!ELEMENT Power EMPTY >
             <!ATTLIST Power         Id           ID    #REQUIRED
                                     Alpha        CDATA #REQUIRED
                                     OriginalUnit CDATA #IMPLIED >

             <!ELEMENT Ratio EMPTY >
             <!ATTLIST Ratio         Id           ID    #REQUIRED
                                     Denominator  CDATA #REQUIRED
                                     OriginalUnit CDATA #IMPLIED >

             <!ELEMENT Index EMPTY >
             <!ATTLIST Index         Id           ID    #REQUIRED
                                     Denominator  CDATA #REQUIRED
                                     OriginalUnit CDATA #IMPLIED >

<!-- END of Transform definition -->


<!-- Access and Protocol -->
```

```
<!ELEMENT Access EMPTY>
<!ATTLIST Access Id ID #REQUIRED
                 IP CDATA #REQUIRED
                 UserId CDATA "anonymous"
                 Password CDATA #IMPLIED >

<!ELEMENT Protocol ( JDBC | FTP | HTTP)* >
<!ATTLIST Protocol Id ID #REQUIRED
                   Encoding             ( ASCII
                                         |Cp1252
                                         |ISO8859_1
                                         |UnicodeBig
                                         |UnicodeBigUnmarked
                                         |UnicodeLittle
                                         |UnicodeLittleUnmarked
                                         |UTF8
                                         |UTF-16
                                         |Big5
                                         |Cp037
                                         |Cp273
                                         |Cp277
                                         |Cp278
                                         |Cp280
                                         |Cp284
                                         |Cp285
                                         |Cp297
                                         |Cp420
                                         |Cp424
                                         |Cp437
                                         |Cp500
                                         |Cp737
                                         |Cp775
                                         |Cp838
                                         |Cp850
                                         |Cp852
                                         |Cp855
                                         |Cp856
                                         |Cp857
                                         |Cp858
                                         |Cp860
                                         |Cp861
                                         |Cp862
                                         |Cp863
                                         |Cp864
                                         |Cp865
                                         |Cp866
                                         |Cp868
```

```
|Cp869
|Cp870
|Cp871
|Cp874
|Cp875
|Cp918
|Cp921
|Cp922
|Cp930
|Cp933
|Cp935
|Cp937
|Cp939
|Cp942
|Cp942C
|Cp943
|Cp943C
|Cp948
|Cp949
|Cp949C
|Cp950
|Cp964
|Cp970
|Cp1006
|Cp1025
|Cp1026
|Cp1046
|Cp1097
|Cp1098
|Cp1112
|Cp1122
|Cp1123
|Cp1124
|Cp1140
|Cp1141
|Cp1142
|Cp1143
|Cp1144
|Cp1145
|Cp1146
|Cp1147
|Cp1148
|Cp1149
|Cp1250
|Cp1251
|Cp1253
|Cp1254
```

```
|Cp1255
|Cp1256
|Cp1257
|Cp1258
|Cp1381
|Cp1383
|Cp33722
|EUC_CN
|EUC_JP
|EUC_KR
|EUC_TW
|GBK
|ISO2022CN
|ISO2022CN_CNS
|ISO2022CN_GB
|ISO2022JP
|ISO2022KR
|ISO8859_2
|ISO8859_3
|ISO8859_4
|ISO8859_5
|ISO8859_6
|ISO8859_7
|ISO8859_8
|ISO8859_9
|ISO8859_13
|ISO8859_15_FDIS
|JIS0201
|JIS0208
|JIS0212
|JISAutoDetect
|Johab
|KOI8_R
|MS874
|MS932
|MS936
|MS949
|MS950
|MacArabic
|MacCentralEurope
|MacCroatian
|MacCyrillic
|MacDingbat
|MacGreek
|MacHebrew
|MacIceland
|MacRoman
```

```
                                        |MacRomania
                                        |MacSymbol
                                        |MacThai
                                        |MacTurkish
                                        |MacUkraine
                                        |Shift-JIS
                                        |SJIS
                                        |TIS620
                                                ) "UTF-16"
                            Physical (TCP|UDP) "TCP">

                        <!ELEMENT JDBC (#PCDATA)* >
                        <!ATTLIST JDBC DatabaseName CDATA #REQUIRED
                                       DatabaseServerType (postgresql|postgresql7.3|access) "postgresql7.3" >

                        <!ELEMENT FTP EMPTY >
                        <!ATTLIST FTP Id CDATA #REQUIRED
                                      Suffix CDATA #REQUIRED
                                      Separator (Comma|Tab|Space) "Comma"
                                      Offset CDATA #IMPLIED
                                      Lines CDATA #IMPLIED >

                        <!ELEMENT HTTP (#PCDATA | CGI | JSP | HTTPS )*>
                        <!ATTLIST HTTP Id CDATA #REQUIRED
                                       Suffix CDATA #REQUIRED >

                                <!ELEMENT CGI (#PCDATA) >
                                <!ELEMENT JSP (#PCDATA) >
                                <!ELEMENT HTTPS (#PCDATA) >

<!-- End of Acess and Protocol -->

<!--PostProcessing -->

    <!ELEMENT ScanFormat (#PCDATA) >
    <!ATTLIST ScanFormat Id ID #REQUIRED >

    <!ELEMENT PrintFormat (#PCDATA) >
    <!ATTLIST PrintFormat Id ID #REQUIRED >

    <!ELEMENT Arithmetic  (#PCDATA) >
    <!ATTLIST Arithmetic Id ID #REQUIRED>

    <!ELEMENT Media (Movie?) >
    <!ATTLIST Media Id ID #REQUIRED
                    Format CDATA #REQUIRED>
```

```
                <!ELEMENT Movie EMPTY>
                <!ATTLIST Movie FrameInterval CDATA #REQUIRED
                                CODEC         CDATA #IMPLIED >

<!-- End of PostProcessing -->

<!-- Systems -->

        <!ELEMENT MainKey (Key)* >
        <!ATTLIST MainKey Id ID #REQUIRED>
                  <!ELEMENT Key EMPTY>
                  <!ATTLIST Key RefId IDREFS #REQUIRED>

        <!ELEMENT Constraint (Which?, Rank?, Given?) >
        <!ATTLIST Constraint    Id           ID     #REQUIRED
                                LongName     IDREFS #REQUIRED
                                Explanation  IDREFS #IMPLIED >
        <!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
        <!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->

                <!ELEMENT Which EMPTY >
                <!ATTLIST Which RefId  IDREFS  #REQUIRED >

                <!ELEMENT Rank EMPTY >
                <!ATTLIST Rank RefId  IDREFS  #REQUIRED
                               Of     IDREFS  #REQUIRED>

                <!ELEMENT Given EMPTY >
                <!ATTLIST Given RefId  IDREFS  #REQUIRED >


        <!ELEMENT Interval (Min?, Max?) >
        <!ATTLIST Interval      Id           ID     #REQUIRED
                                LongName     IDREFS #REQUIRED
                                Explanation  IDREFS #IMPLIED >
        <!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
        <!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->

                    <!ELEMENT Min EMPTY >
                    <!ATTLIST Min RefId IDREFS #REQUIRED >

                    <!ELEMENT Max EMPTY >
                    <!ATTLIST Max RefId IDREFS #REQUIRED >

        <!ELEMENT Time   (Era?, Year?, Month?, Day?, Hour?, Minute?, Second?) >
        <!ATTLIST Time  Id           ID     #REQUIRED
                        LongName     IDREFS #REQUIRED
```

```
                Explanation  IDREFS #IMPLIED >
        <!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
        <!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->


        <!ELEMENT Era    EMPTY >
        <!ATTLIST Era    RefId IDREFS #REQUIRED >

        <!ELEMENT Year   EMPTY >
        <!ATTLIST Year   RefId IDREFS #REQUIRED >

        <!ELEMENT Month  EMPTY >
        <!ATTLIST Month  RefId IDREFS #REQUIRED >

        <!ELEMENT Day    EMPTY>
        <!ATTLIST Day    RefId IDREFS #REQUIRED >

        <!ELEMENT Hour   EMPTY>
        <!ATTLIST Hour   RefId IDREFS #REQUIRED >

        <!ELEMENT Minute EMPTY>
        <!ATTLIST Minute RefId IDREFS #REQUIRED >

        <!ELEMENT Second EMPTY>
        <!ATTLIST Second RefId IDREFS #REQUIRED >

<!ELEMENT Sexagesimal (Degree?, Minute?, Second?) >
<!ATTLIST Sexagesimal   Id           ID     #REQUIRED
                        LongName     IDREFS #REQUIRED
                        Explanation  IDREFS #IMPLIED >
<!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
<!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->

        <!ELEMENT Degree EMPTY>
        <!ATTLIST Degree RefId IDREFS #REQUIRED >
        <!-- <Minute> and <Second> tags are not defined
                here but in <Time> -->


<!ELEMENT OrthogonalCoordinate (X, Y, Z) >
<!ATTLIST OrthogonalCoordinate  Id          ID     #REQUIRED
                                LongName    IDREFS #REQUIRED
                                Definition IDREFS #REQUIRED
                                Explanation  IDREFS #IMPLIED >
<!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
<!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->
```

```
        <!ELEMENT X EMPTY >
        <!ATTLIST X     Definition     IDREFS #REQUIRED
                        RefId          IDREFS #REQUIRED >

        <!ELEMENT Y EMPTY >
        <!ATTLIST Y     Definition     IDREFS #REQUIRED
                        RefId          IDREFS #REQUIRED >

        <!ELEMENT Z EMPTY>
        <!ATTLIST Z     Definition     IDREFS #REQUIRED
                        RefId          IDREFS #REQUIRED >

<!ELEMENT PolarCoordinate (R, Theta) >
<!ATTLIST PolarCoordinate       Id          ID     #REQUIRED
                                LongName    IDREFS #REQUIRED
                                Definition  IDREFS #REQUIRED
                                Explanation IDREFS #IMPLIED >
<!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
<!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->

        <!ELEMENT R EMPTY>
        <!ATTLIST R Definition IDREFS #IMPLIED
                    RefId      IDREFS #REQUIRED >

        <!ELEMENT Theta EMPTY>
        <!ATTLIST Theta Definition IDREFS #REQUIRED
                        RefId      IDREFS #REQUIRED >


<!ELEMENT CylindricalCoordinate (R, Theta, Z) >
<!ATTLIST CylindricalCoordinate Id          ID     #REQUIRED
                                LongName    IDREFS #REQUIRED
                                Definition  IDREFS #REQUIRED
                                Explanation IDREFS #IMPLIED >
<!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
<!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->

        <!---   R   is already defined in PolarCoordinate     -->
        <!--- Theta is already defined in PolarCoordinate     -->
        <!---   Z   is already defined in OrthogonalCoordinate -->

<!ELEMENT SphericalCoordinate (Rho, Theta, Phi) >
<!ATTLIST SphericalCoordinate   Id          ID     #REQUIRED
                                LongName    IDREFS #REQUIRED
                                Definition  IDREFS #REQUIRED >
<!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
```

```
                <!ELEMENT Phi EMPTY >
                <!ATTLIST Phi   Definition  IDREFS  #REQUIRED
                                RefId       IDREFS  #REQUIRED >

        <!ELEMENT LongitudeLatitude (Longitude, Latitude) >
        <!ATTLIST LongitudeLatitude     Id          ID     #REQUIRED
                                        LongName    IDREFS #REQUIRED
                                        Explanation IDREFS #IMPLIED >
        <!-- LongName: Should Refer N <Name>(s) in <Appendix> -->
        <!-- Explanation: Should Refer N <Text>(s) in <Appendix> -->

                <!ELEMENT Longitude EMPTY >
                <!ATTLIST Longitude     Definition IDREFS #REQUIRED
                                        RefId      IDREFS #REQUIRED >

                <!ELEMENT Latitude EMPTY >
                <!ATTLIST Latitude      Definition IDREFS #REQUIRED
                                        RefId      IDREFS #REQUIRED >
<!-- End of Systems -->

<!-- ENTITY -->
<!ENTITY nbs " " >

<!-- End of DTD -->
```

参考文献

[1] 三浦孝夫,『データモデルとデータベース』, サイエンス社 (1997).

[2] 増永良文,『リレーショナルデータベースの基礎 — データモデル編 —』, オーム社 (1990).

[3] C.J. Date and Hugh Darwen, QUIPULLC（訳）,『標準 SQL ガイド 改訂第 4 版』, アスキー (1999).

[4] Becker,R.A., Chambers,J.M. and Wilks,A.R., 渋谷政昭・柴田里程（訳）,『 S 言語 II』, 共立出版 (1991).

[5] Cox. D.R. and Snell, E.J., *Applied Statistics: Principles and Applications*, Chapman and Hall, MA (1981).