第 2 章

データ

データベースシステムって何でしょうか．システムは，複数の要素が集まっていてそれらが相互作用しながら何らかの共通した目的のために動いていることを意味します．ここでは，データベースを保持管理処理するためのソフトウェアやハードウェア全体のことを指すと考えるのが良いでしょう．それでは，データベースって何でしょうか．ベースという言葉は基地という意味なので，データがたくさん整理された状態で管理されているという意味だと考えるのが良いでしょう．それでは，データって何でしょうか．この章ではデータとは何かについて考えます．

2.1 基本データ型とその等価性および順序

データのそれ以上分解できない基本的な単位を考えましょう．それは，例えば，整数であったり，文字列であったり，日付だったり，バイナリデータだったりというものです．それらを基本データ型 (primitive data type) と呼びます．基本データ型である整数型の具体的な値は例えば 1 や 2 です．これらの値は整数型のインスタンスともいいますね．皆さんはデータベースシステムよりプログラミング言語の知識をたくさん持っていると思いますから，プログラミング言語を語るときに用いられる用語を適宜持ってきて説明します．

同じ型の値をどうやって区別しましょうか．それには等価性 (equality) を使います．型 `A` のデータ `a1`，`a2` が等しいとき `a1 == a2` とかき，等しくないとき `a1 != a2` と書くことにしましょう．多くのプログラミング言語でこう書きますからね．基本データ型であれば値が同じ (値は何らかのバイト列もしくはビット列で表現されるとして，それがまったく同じという意味です) であれば等しい，違えば等しくない，という定義でまず実用上差し支えないでしょう．たとえば，整数の 1 と 2 があったとき，`1 == 1` で `1 != 2` ですね．文字列だと，`'aaa' == 'aaa'`，`'aaa' != 'aab'`，`'aaa' != 'aaaa'` などです．

等価性に加えて，同じ型の値同士でよく用いられる関係が，順序 (order) です．特に全順序がよく使われます．整数は全順序集合ですね．`1 < 2` ですし，`2 < 3` です．日付も過去より現在，現在より未来が新しいという順序を持っています．文字列は辞書順で大小関係を扱うことが多いです．プログラミング言語においては，自分で作った構造体に任意の順序をつけることもありますね．等価性は前提とすることが多いので `==` が定義されており，さらに `<` という演算子が適切に定まれば全順序を定義できます．

データベースを扱うときにデータの等価性はまず間違いなく必須ですが，順序については必ずし

も必要ありません．あるデータ型の値の部分集合を考えるとき，等価性のみ使えるデータ型はひとつずつ列挙して部分集合を表現する必要がありますが，順序を持つデータ型は範囲でも部分集合を表現することができるという特徴があります．

2.2 Record，Table，Key

データとは基本データ型の値の集合といえます．Relational database systems (関係データベースシステム，以後 RDBMS とかきます) では，基本データ型を複数まとめて，record 型というものを定義して使います．Record 型の値は record と呼ばれ，tuple 型，tuple と呼ばれることもあります．Tuple という言い方はプログラミング言語でもそのまま使いますね．Record 型の中の要素を区別するために，それらを column とか field と呼び，区別しやすいように名前をつけます．Record 型は，プログラミング言語でいうところの構造体 (struct) に相当します．

RDBMS では record 集合を管理するために，table という概念が使われます．ひとつの table には同じ record 型のデータが複数格納されます．まったく同じ record 型の異なる table を作ることは可能です．RDBMS では record 型には名前をつけ (られ) ず，table に名前をつけます．

RDBMS では record 型を入れ子にして定義する (nested) ことは想定されない場合が多いようです．これはデータ重複を防ぐ「正規形」の考え方があるためだと思われます．「正規形」「正規化」はデータベーススキーマ (Record 型，table，key やその他の制約等の定義をまとめたもの) が持っているべき性質やスキーマの正規形への変換方法を意味します．トランザクション処理とは直接関係ないので，ここではこれ以上説明しません．トランザクションでは，ひとつの record やその column をそれ以上分割されないデータアクセスの最小単位と考えます．

以下にスキーマとその record の例を示しました:

```
Schema example:

Table Human:
  (id: integer, last_name: string, first_name: string,  birthday: date)


Human record examples:
  (1, 'tanaka', 'ichiro', 2000-01-01)
  (2, 'yamada', 'hanako', 2001-08-08)
  (3, 'suzuki', 'jiro',   2000-01-01)
```

`Human` table がひとつ定義されています．`Human` の record 型は `id`，`last_name`，`first_name`，`birthday` という 4 つの column から構成されます．それぞれの column は基本データ型の `integer`，`string`，`string`，`date` という型です．3 つの record が具体例として挙げられています．

同じ record 型のふたつの record が等しいとは，素朴には全 column の値が等しいことを意味します．様々な場面で record の区別をするときに key という概念が使われます．Key とは，record を入力とする関数もしくはその値と考えることができます．より狭い定義では，record に含まれる

ひとつもしくは複数の column からなる tuple を指して key といいます．ひとつの table について column の選び方によって key の種類は複数存在します．以後，table の key と言ったときは，対応する column の tuple を指すものとし，ある record の key と言ったときは，具体的な record 内の対応する column 値の tuple を指すものとします．Key が record を区別するのに十分な情報を持っているとき，すなわち key が等価であることと record が等価であることが同値であるとき，その key は unique key といいます．

以下に key の例を示しました:

```
Key Name of Human table:
  (last_name, first_name)

Key Birthday of Human table:
  (birthday)

Key NameAndBirthday of Human table:
  (last_name, first_name, birthday)

Unique key Id of Human table:
  (id)


Name key of records:
  ('tanaka', 'ichiro')   id = 1
  ('yamada', 'hanako')   id = 2
  ('suzuki', 'jiro')     id = 3

Birthday key of records:
  (2000-01-01)  id = 1
  (2001-08-08)  id = 2
  (2000-01-01)  id = 3

NameAndBirthday key of records:
  ('tanaka', 'ichiro', 2000-01-01)  id = 1
  ('yamada', 'hanako', 2001-08-08)  id = 2
  ('suzuki', 'jiro',   2000-01-01)  id = 3
```

`Human` table に 4 つの key を定義しています．`Name`，`Birthday`，`NameAndBirthday`, そして `Id` です．RDBMS では key の定義はほぼインデクスの作成指示を意味しますが，ここでは単にこのような key を考えてみるという意味で捉えてください．`Name` key は `(last_name, first_name)` と書いてありますが，これは，`Name` key は，`Human` 型の record を入力とし，その column のうち `last_name` および `first_name` のみ取り出して tuple を生成し，それを出力とする関数と考えます．Record 集合の演算として見ると projection といいます．

ここで `Name` は unique key でしょうか？ 例の 3 つの record を見る限りでは重複しているものはなさそうなので，いまのところ unique になっているようですが，今後 record が追加された場合は unique 性が担保されなくなってしまうかも知れません．実は unique key というのは，結果としてそうなっているということではなくスキーマに与える制約のことなのです．`Human` table (型)

を定義した人，ここでは管理者とします，が想定する潜在的な record 集合がどのようなものかによって決まります．つまり，管理者がその key の unique 性を担保したい場合，システムに制約の指示を与えます．Unique key 制約が与えられた場合，システムは `(last_name, first_name)` の組が unique でなくなるような操作を許しません．たとえば，既に `('tanaka', 'ichiro')` という Name key を持つ record が存在するのに，同じ `('tanaka', 'ichiro')` を持つ別の record は，例え他の column が異なっていても追加できなくなります．(逆に，典型的な RDBMS は unique 制約がなければ何もかも同じ record を複数登録できます．) もちろん一般には `Name` は unique ではないので，Name を unique key にすると不便です．ならば `NameAndBirthday` はどうでしょうか．実際に同性同名で誕生日も同じの人がいる可能性はかなり低いでしょうがゼロというわけでもないでしょう．一般的なデータベース設計では，意図的に uniqueness を担保するために，無意味な unique id を割り当てて unqiue key として区別できるようにします．ここでは `Id` key がそれにあたります．今ならマイナンバーがあり，個人にひとつ割り当てられることを仕組みとして担保しているので，システム毎に `Id` を割り振るよりも，unique key としてマイナンバーを使った方が良いかも知れませんね．データベースの扱いに，個人情報の保護など別の社会的法律的な制約が発生しますけれど:)

Table ひとつにつき，その unique key の中で主要なものをひとつ選び，primary key と呼びます．明示的な primary key がない場合は，隠し column が用意され，table 内で unique な整数が割り当てられ，primary key として扱われることが多いです．(MySQL InnoDB はそのような実装となっています．)

ある table について，任意の key 値を指定すれば，複数の record がマッチし得ます．もちろん存在しない key 値を指定すればマッチするのは 0 個です．例えば，Human table から Birthday key の値として `2000-01-01` を指定すると，`id 1` と `3` の 2 つの record がマッチします．Unique key の場合は高々 1 つの record がマッチします．データベースにおいてデータを指定する最も基本的な操作が，ある table において key 値を与えて table を構成する record 集合の部分集合を指定する操作です．等価性を用いる場合は，指定したい等価な key 値の集合を与えます．順序を用いる場合は，指定したい key 値の範囲を与えます．演算子や関数を使ってより複雑な条件を指定することも出来ますが，最終的には，複数 table の複数 record に何らかの順番でアクセスすることになります．

複数の columns から構成される Key の順序を考える場合は辞書順を考えます．2 つの整数型からなる key があって，`(a, b)` と表すとき，その key の順序は例えば `(1, 1) < (1, 2) < (2, 1)` となります．もちろん，任意の key について，任意の順序を定義し得るわけですが，自動的に決まる順序として辞書順が採用されるシステムが多いです．逆に，ある順序で扱いたいからそうなるように key を定義するとも言えるでしょう．RDBMS の実装によっては，基本データ型が持つ自然な順序を ascending (昇順)，その逆順を descending (降順) としてそれぞれ `ASC`，`DESC` の演算子で扱えるものがあります．

■コラム: 順序と全順序

集合が順序や全順序を持つためにはある性質を満たす必要があります．具体的には集合の元についての二項関係を表す演算子 `<=` が反射律，反対称律，推移律を満たせば (半) 順序であり，加えて，集合の任意の 2 つの元が `<=` で比較可能である場合に全順序といいます．

単純なルールとして，その型の任意の値を整数や実数に割り当てる関数 (単射写像) を用意すれば，全順序の性質を満たします．整数や実数の tuple に割り当てても良いです．全順序の型で作る tuple 型は辞書順を考えれば全順序となります．Key は基本データ型の tuple でしたね．Key を構成する全ての column の型が全順序の性質を持っていれば，Key にも自然な全順序が定義されます．

2.3 データの関係とポインタ

データは構造 (関係) を持っています．プログラミング言語では，基本データ型に加えて，構造体とポインタ (参照) 型があれば，任意のデータ構造を表現することが出来るでしょう[*1]．

RDBMS はポインタ型を直接的には扱わない特徴があります．では RDBMS でデータ同士の関係を表すにはどうすれば良いのでしょうか．それは，共通の部分データを持つことで表現します．R1, R2 という record 型があり，それぞれが C1 という column を持つものとします．R1.C1 (R1 型における C1 column という意味) と R2.C1 が等価であるレコード同士，すなわち，`R1.C1 == R2.C1` である R1 型の record と R2 型の record は関係があるという意味になります．同一 record 型 (もしくは 同一 table 内) の record 同士に関連を持たせたい場合は，R1.C1 と R1.C2 という二つの column を定義しておいて，`R1.C1 == R1.C2` という形で関係を持たせることができます．これらの関係は，一般に，1:1，1:N，M:N という 3 種類のパターンに分類して考えます．

無理矢理ポインタでどのような表現になるかを考えてみましょう．1:1 の関係は，record と record をお互いがお互いを指している状態を表します．1:N は，N 側の各 record が 1 側の record を指している状態，1 側は配列などを持っていて，そこに N 側を指すポインタが複数格納されている状態を表します．M:N は M 側の各 record が配列などで N 側の record を指すポインタを複数保持している状態と，N 側の各 record が配列などで M 側の record を指すポインタを複数保持している状態と考えることができます．こんな複雑な関係をポインタで管理したくないですね:)

ポインタの参照外し (dereference) に相当する操作は，内部結合 (inner join) です．アプリケーションが自分でやっても良いですが，RDBMS に任せた方が原理的には高速です．(ただし，SQL は宣言的言語なので，query optimizer が必ずしも良い実行計画を選んでくれるわけではないという辛さがあります．そこでごにょごにょと join の順番や使う index を指定するなどのチューニングをすることで対応します．)

何故ポインタを使わないか，という問いには歴史的経緯があるようなので，興味のある人は調べてみてください[*2]．ポインタを扱わないことで多少窮屈ですが dangling pointer がない世界に住むことができます．

[*1] 配列型？ 私は好きですよ．主にヒープメモリに確保する動的配列が大好きです．

[*2] 私も詳しくないですが，http://leoclock.blogspot.com/2009/01/blog-post_07.html によると，Readings in Database Systems の中の解説記事に，書いてあるようです．

2.4 最も単純なデータベースのスキーマ

最も単純なデータベースについて考えてみましょう．まず table がひとつしかないです．その table が採用する record 型は key を表す型と value を表す型の 2 つの column で定義されます．Key は primary key すなわち unique key です．これは key-value store と呼ばれるものですね．Key は文字列型で，value はバイト列であることが多いです．Key は文字列型として自然に定義される等価性と順序 (辞書順) をサポートしています．

アプリケーションが数値型を必要とするなら，数値型を文字列型に変換して使います．10 進数を用いて数字を文字として足りない分を 0 埋めして桁数を固定した文字列に変換すれば，辞書順と数値の昇順は一致させることができます．もちろん文字集合における順序が '0' < '1' < ... < '9' を満たすことが前提となります．例えば 10 桁固定にするとして 1 は '0000000001' に変換され，10 は '0000000010' に変換されます．'0000000001' < '0000000010' ですね．負の数を扱いたいならコンピュータがそうしているように，符号を表す桁を一番最初に追加して，補数を用いることで順序を保存したまま変換できます．

スキーマを自在に定義できるデータベースシステムはもちろん実用では必須ですが，学習用として最初に作るべきは，このような単純な key-value を管理するシステムでしょう．

■コラム: NULL について

RDBMS の column はデフォルトで NULL 値が許容されているものがほとんどです．Haskell でいうところの Maybe 型，Rust でいうところの Option 型です．Unique 制約と同様に，column に NOT NULL 制約を指定することはできます．外部結合 (outer join) をするためには NULL 値が必要なのですが，現代のプログラミングの常識から考えると，デフォルトは NOT NULL にして欲しいものですね．NULL を考慮し忘れると演算が想定外の結果になってしまうことがあります．二項演算子やユーティリティ関数の引数にひとつでも NULL を渡すと結果の多くは NULL になり，これが罠となります．NULL を含めた演算ルールは三値論理という立派な名前がついているのですが，多くの場合に我々が期待する演算は，NULL (Nothing, None) 値を何らかのデフォルト値に変換して演算することなのですね．和のときは 0，積のときは 1，文字列連結のときは空文字列など..

■コラム: コンピュータサイエンスで使う数学

あなたがデータ構造やアルゴリズムについて考察したり，生み出したりしたとき，定式化をする必要が出てくることがあります．コンピュータサイエンスで用いる数学について学ぶのに良い資料として，

- Mathematics for Computer Science.
  - Eric Lehman, F Thomson Leighton, and Albert R Meyer.

– https://courses.csail.mit.edu/6.042/spring17/mcs.pdf

を挙げておきます．特に，集合の基礎と述語論理のところをきちんと学んで使えるように練習することが，論文を読むときの定理や証明を理解するための第一歩であり，自分で定式化するときの道具にもなると思います．まあ私はごく最近この資料の存在を知って勉強したので偉そうなことは言えません．この資料のライセンスは CC BY-SA 3.0 だそうです．太っ腹ですね．