■7 群（コンピュータ —ソフトウェア）－ 5 編（データベース）

14 章 オブジェクト指向データベース

（執筆者：石川　博）［2015 年 6 月 受領］

■概要■

関係データベースが様々な領域に普及するにつれて，エンジニアリングやマルチメディアのような新しい領域に適用する際にその課題も現れてきた．そうした課題を解決するために提案，開発されたのがオブジェクト指向データベースである．更に，オブジェクト指向データベースの出現は純粋な意味での関係データベースにも影響を与え，オブジェクト指向の概念を取り入れたオブジェクト関係データベースが開発されることになる．

【本章の構成】

本章では，より新しい概念であるオブジェクト指向とデータベースの関連について説明する．まず，オブジェクト指向データベースの基本概念（14-1 節），オブジェクト指向データベースの問い合わせ（14-2 節），関係データベースの発展形であるオブジェクト関係データベース（14-3 節）について説明した後，オブジェクト指向の関連概念として，非正規化データベース（14-4 節）について触れる．

■7 群－5 編－14 章

14-1 オブジェクト指向データモデルの基礎概念

（執筆者：石川　博）［2015 年 6 月 受領］

14-1-1 基本概念とプログラミング言語との比較

ここでは，オブジェクト指向データベース[1]の基本概念について，プログラミング言語との対比で説明する．そもそも CAD/CAM（設計，製造）やマルチメディア，ドキュメント利用などの応用分野で，利用者がそれまでの“純粋な意味での”関係データベースで扱えない複雑なデータ構造と操作を定義し，格納・管理したいという要求が大きくなってきた．それに応えるようにオブジェクト指向データベースが生まれた．

もともとオブジェクト指向の概念はプログラミング言語に端を発する．すなわち，オブジェクトの原点はシミュレーションプログラム専用のオブジェクト指向プログラミング言語である Simula にある．その後，Simula のオブジェクト指向の考えに影響されて Smalltalk が作られた．ただし，Smalltalk（Smalltalk-80）は当時広く使われていた構造化プログラミング言語である C 言語と比べかなり独自な言語であったため C にオブジェクト指向の考えを取り入れて C++ が作られ，更にそれが現在広く使われている Java につながる．

Smalltalk のようなオブジェクト指向プログラミング言語では，オブジェクトの型は，型宣言によってではなく，実行時に決定される．これを動的束縛（Dynamic Binding）という．つまり，その時点で指定された属性やメソッド（オブジェクトに定義された操作）があればよい．ただし，この方式では実行時における属性やメソッドのサーチの負荷が大きくなる．そこで大部分のオブジェクト指向データベースでは，コンパイラで属性やメソッドの参照を解決する静的束縛（Static Binding）方式がとられる．

まず，オブジェクト指向データベースの基本となった C++ におけるオブジェクト指向の概念について，オブジェクト指向データベースとの対比で簡単にまとめておく．

- **オブジェクト，オブジェクト識別子（オブジェクトポインタ），属性（データメンバー），メソッド（メンバー関数）**：オブジェクトの属性にはデータを格納し，メッセージ（メソッドの実行を依頼）をオブジェクトに送ってオブジェクトを操作する．オブジェクトはオブジェクト識別子と呼ばれる論理的なポインタによって参照する．しかしながら，オブジェクトはプログラムの終了とともに揮発（消滅）してしまう．したがって，オブジェクト指向データベースでは，これらを永続オブジェクトに拡張する必要がある．そのために永続クラスを提供する．また，オブジェクトを作成するときに，永続性を指定する方式もある．更に，集合値属性を表現するためのクラス（List，Set，Bag，Array など）が提供される．
- **オブジェクト型，クラス，クラス階層及びそれに伴う単一/多重継承**：ユーザは新しいデータ型をオブジェクト型として定義できる．その際に既にあるクラスのサブクラスとして新しいクラスを定義する．既にあるクラスをスーパークラスという．サブクラス（特化）はスーパークラス（汎化）から属性とメソッドの定義を継承するので，これにより差分プログラミング（Differential Programming）ができる．ただし，クラスにはそれに属するインスタンス集合（データベースもしくは Extent）という概念はないので拡張しなければならない．一般に複数のクラスから継承を行うことを多重継承といい，単一のクラスからの継承

と区別する．必要に応じて属性・メソッドの定義の置き換え（Override）も許す．

- **属性とメソッドのカプセル化**：属性にはメソッドのインタフェース（メソッド名，パラメータ）のみを通してアクセスさせる．すなわち，属性やメソッドの実現などに関する情報の隠蔽機能を提供する．なお，異なるクラスが同一のインタフェースを持つ場合があり，その場合でも一般にはクラスごとに実現は異なるので，同一メッセージに対してオブジェクトごとに異なる振る舞いをする．このことをメソッドの多相性（ポリモルフィズム，Polymorphism）という．あるいは，メソッドのオーバーローディング（Method Overloading）という．
- **1 対 1，1 対 *N*，*M* 対 *N* 関連**：様々な関連を，属性を通して自由に定義できる．ただし，逆関連についてはオブジェクト指向データベースで拡張する必要がある．

図 1･1 に逆関連，集合値属性，インスタンス集合の点で拡張された C++ ベースのオブジェクトモデルの例を示す．この記述は ODMG[2]（Object Data Management Group，オブジェクト指向データベースの業界標準を制定する団体であり，その標準規格そのものも表す）の構文に従っている．例えば，produces の逆関連は，Product の is_produced_by である．また produces は結果として Product の集合を返す．更に producers は，Producer のインスタンス集合への参照関係を定義する．なお，C++ バインディングにおける ODMG（他に Smalltalk と Java バインディングがある）ではオブジェクト作成時に永続性を指定する．

```
class Producer: public d_Object {
public:
 // properties:
    d_String pname;
    d_Rel_Set <Product, _is_produced_by> produces;
     d_Rel_Ref <Employer, _operates> is_operated_by;
 // operations:
     int monthly_sales (year, month);
          …
 // extent
    static d_Ref <d_Set <d_Ref<Producer>>> producers;
    static const char *const extent_name;
};
```

図 1・1　オブジェクトモデル

以下にオブジェクト指向データベースの基本概念をまとめておこう．第一にオブジェクト指向データベースにおける基本単位はオブジェクトである．そのオブジェクトの構造と操作を表すものがオブジェクトの型である．オブジェクトの型はクラスで定義される．ただし，クラスには一般に単なるオブジェクトの型だけでなく，クラスに属するオブジェクトの集合（Extent）も関連付けることができる．

クラスに対して，それに属する個々のオブジェクトをインスタンスという．インスタンスはオブジェクト識別子によって一意に区別される．型にはオブジェクトの属性（プロパティ）の型とメソッドの型があり，属性型は代入できるデータ型を規定する．例えば，文字列型を持つ属性に実数型の値を代入しようとするとエラーになる．これを定義域制約という．通常，属性

の操作はメソッドを通してのみ行える．このことをカプセル化または情報隠蔽という．これにより，データの独立性を保障する．

メソッドの型はパラメータや結果の型を含めたインタフェース（シグニチャ，Signature ともいう）を規定する．それに対して，メソッドの定義本体は実現を規定する．オブジェクトにメッセージを送ることでメソッドを起動しオブジェクトを操作する．メッセージを送受信するオブジェクト間には協調関係（Collaboration）があるという．メッセージが送られたオブジェクトのクラスもしくはそのスーパークラスに，メッセージで指定されたインタフェースを持つメソッドが定義されていないとエラーとなる．これらはコンパイル時にチェックできる場合（すなわちオブジェクトの型があらかじめ分かっている場合）と，実行時でしかチェックできない場合（オブジェクトの型があらかじめ分からない場合）がある．いずれにせよ，型の概念はユーザが防衛的なプログラミングを行うことを許す．メソッドの実現については，インタフェース自体を変更しない限り，それを変更することができ，これにより操作に関する独立性を保障する．オブジェクトの間の協調関係やスーパークラスとサブクラスによる階層関係（汎化特化関係）のほかに，親オブジェクトと子オブジェクトの間の部品関係（集約と分解）を定義することができる．以上説明したオブジェクトの基本概念（オブジェクト指向パラダイム）をまとめると**図 1･2** のようになる．

① オブジェクト
  a. 外界の事象を表現する単位
  b. 属性（プロパティ）と動作（メソッド）が一体化
  c. 識別子による管理
② クラス
  a. 個々のオブジェクト（インスタンス）に共通の性質と動作をまとめる
  b. プロパティはメソッド（インタフェース）でしか操作できない⇒カプセル化
③ メッセージ
  a. オブジェクトに動作を依頼する単位
  b. メッセージを受信するとメソッドを実行
④ 関係
  a. オブジェクト間協調関係：メッセージの送受信
  b. 部品関係：親子関係（集約と分解）
  c. スーパークラス・サブクラス関係：階層関係（汎化と特化）
⑤ 継承
  a. スーパークラスからプロパティとメソッドを継承⇒差分プログラミング
⑥ 多相性（ポリモルフィズム）
  a. 同じメッセージでもオブジェクト（クラス）が異なれば，違う振る舞いをする

図 1・2　オブジェクトの基本概念

14-1-2　関係データベースとの比較

次に“純粋な意味での”関係データベースとの対比でオブジェクト指向データベースについて述べる．

まず，オブジェクト指向データベースでは，システムが提供する文字列や整数といった原始的な型（リテラル，Literal 型）だけでなく，ユーザが既存の型を組み合わせて新しく型を定義

し利用できる．これをタプル型という．また，集合値を表現する手段として List，Set，Bag，Array などがある．例えば，図 1･1 で生産者の名前は文字列型を持つが，経営者は経営者クラスの型を持ち，経営者クラスのオブジェクト（オブジェクト識別子）を代入できる．このような属性を参照属性という．つまり，オブジェクト指向データベースには，複合オブジェクト（Complex Object）を直接的に表現する手段が備わっている．

一方，“純粋な意味での”関係データベースにおけるテーブルの型として許されるのはシステムで提供される文字列などの原始的な型だけであり（簡単な構造を持ったデータ型としては日付が許されが），ユーザが新たな型を定義し，それを属性の型として利用することはできない．したがって，関係データベースでは複合オブジェクトを直接的に表現する手段はない．

次に，オブジェクト指向データベースでは，データ構造としてデータの操作（メソッド）も同時に定義できる．既にあるデータ型（スーパークラス）のサブクラスとして，新しくデータ型を定義し，スーパークラスの属性とメソッドを継承できる．プログラミング言語との親和性が重視される．

一方，“純粋な意味での”関係データベースにではデータに固有の操作もアプリケーションプログラムに埋め込まれてしまい，DBMS を通した共有が難しい．更に，属性とメソッドの継承という概念はない．

更に，オブジェクト指向データベースではオブジェクト間の関連を直接的にモデル化することができる．関連は属性値としてオブジェクト（オブジェクト識別子）を格納する参照属性を利用することにより基本的に表現できる．関連には一般に 1 対 1，1 対 *N*，及び *M* 対 *N* 関連があり，オブジェクト指向データベースの属性は単一値だけでなく値の集合を持つことができる．前者を単一値属性，後者を集合値属性（多値属性ともいう）という．関連する 2 つのオブジェクトに単一値の参照属性を持たせれば 1 対 1 関連が実現でき，片方のオブジェクトに集合値の参照属性を持たせ，もう一方のオブジェクトに単一値の参照属性を持たせれば 1 対 *N* 関連に，両方のオブジェクトに集合値の参照属性を持たせれば *M* 対 *N* 関連が実現できる．また，関連には方向性が存在し，したがって逆関連が考えられる．関連とその逆関連は一貫性を維持するように同期して変更しなければならない．また，関連では参照しているオブジェクトが必ず存在しなければならないという参照完全性を維持することが重要になる．参照完全性に対してはユーザが自分で管理するもの，システムが自動的に維持するもの，あるいは存在しないオブジェクトへの参照が起きた時点でシステムがチェックするものなどがある．**図 1･3** にオブジェクト間の関連を図式化して示す．

これに対して“純粋な意味での”関係データベースではオブジェクト識別子の概念がないので，まずテーブルの属性（外部キー）の値として他のテーブルの主キーを格納する．そのうえで，アプリケーションが外部キーと主キーによる 2 つのテーブルの結合演算を含む問い合わせを実行することにより，関連がはじめて実現できる．1 対 1 関連の場合は実体に対応するテーブル（エンティティテーブル）に外部キーを格納し，エンティティテーブル同士を結合する．1 対 *N* 関連の場合は *N* 側の実体に対応するテーブルに，他方のテーブルのキーを外部キーとして格納する．“純粋な意味での”関係データベースの場合には集合値属性が提供されないので，*M* 対 *N* 関連の場合は関連に相当するテーブル（リレーションシップテーブル）を用意し，そこに複数テーブルの外部キーを格納し，エンティティテーブルとリレーションシップテーブルを結合する．単純なリテラル型に対応する集合値属性の場合も新たにテーブルを作り，元のテー

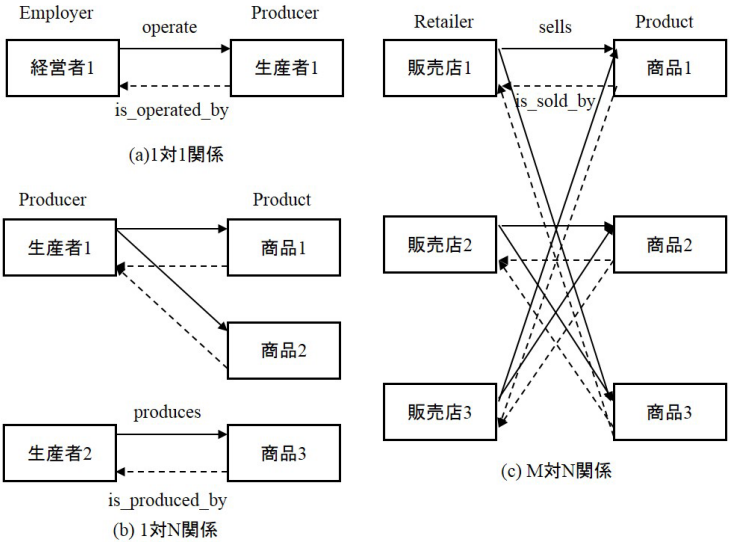

図 1・3　オブジェクト間の関連

ブルのキーと値の組を格納する．もちろん，定義域制約や参照完全性を維持することは関係データベースによってもできる．

最後にオブジェクトの永続性に関する注意を述べておこう．オブジェクト指向データベースのインスタンスは，常に永続的とは限らない．この点も関係データベースとは異なる．永続性の指定には，そのクラスをあらかじめ永続的オブジェクトクラスのサブクラスとして定義しておく方式（ObjectStore[3]）がある．また，実際にインスタンスを作成するときに永続的であることを宣言する方式 ODMG の C++ バインディングや，更に永続的オブジェクトから参照されていれば，芋づる式に永続的になるという方式（ODMG の Smalltalk/Java バインディング）もある．

■7 群－5 編－14 章

14-2 オブジェクト指向データベースの問い合わせと問い合わせ言語

（執筆者：石川　博）［2015 年 6 月 受領］

オブジェクト指向データベースにおける問い合わせ機能は，データベースに格納されているオブジェクトの集合から必要なオブジェクト（の集合）を選択し，操作する機能である．問い合わせでは，オブジェクトの検索だけでなく，検索したオブジェクトに対してメッセージを送ってオブジェクトを操作することもできる．ユーザが発した非定形の問い合わせ（アドホックな問い合わせ）を対話的に処理する場合と，アプリケーションプログラムの中に問い合わせが埋め込まれて実行される場合がある．問い合わせの処理の仕方としては問い合わせを毎回解析し，実行するインタプリタ方式と，問い合わせを解析した後，実行コードを生成し，2 回目以降は実行コードを直接実行するコンパイラ方式がある．インタプリタ方式はあらかじめ問い合わせの形が分からないアドホックな問い合わせの処理に適し，コンパイラ方式はプログラムに埋め込まれた問い合わせの処理に適する．

問い合わせ言語は，前述したオブジェクトのスキーマ（クラス）を定義するオブジェクト定義言語（Object Definition Language：ODL）とオブジェクトを検索操作するオブジェクト問い合わせ言語（Object Query Language：OQL）から成る．ユーザはこれらを組み合わせて利用する．アプリケーションプログラマは，これらをプログラムに埋め込んだり，専用の API として提供される関数群を組み合わせたりして利用する．アプリケーションではスキーマ変更は許されない場合が多い．C++ バインディングの ODL の例は既に図 1･1 に示した通りであるが，個別のプログラミング言語に寄らない ODL は**図 2･1** のようになる．

```
class Producer
(extent producers)
{ attribute string pname;
  relationship set<Product> produces inverse Product:: _is_produced_by;
  relationship Employer is_operated_by inverse Employer::operates;
  long monthly_sales (in short year; in short month);
        …
};
```

図 2・1　オブジェクトモデル

基本的な OQL の問い合わせは，SQL 同様に以下のような SELECT-FROM-WHERE 構文で指定する．

SELECT 検索対象リスト
FROM 繰り返し定義リスト
[WHERE 検索条件]
[GROUP BY グループ化属性リスト]
[HAVING グループ選択条件]
[ORDER BY 整列指定リスト]

問い合わせを構成する単位は一般にパス式（Path Expression）と呼ばれ，永続的オブジェクト名かまたは繰り返し変数に，関連する属性名の列を 0 個以上連ねたものである．パス式を組み合わせて一般に式ができる．検索対象は式であり，検索したいオブジェクトを指定する．繰り返し定義も繰り返し変数と式から成る．検索条件も式であり，オブジェクト式同士やオブジェクト式とリテラルの比較を比較演算子（等号，大小比較，集合の包含関係など）で行い，更に必要に応じてそれらを論理演算子で結合する．グループ選択条件や整列指定も式である．なお，パス式やエクステント名単独を含めて，一般に式そのものが問い合わせとなる点で OQL は SQL と異なる．

オブジェクトの属性の一部を指定した場合は，関係データベースの射影に相当する．それらの値の組（タプル）の多重集合が結果として返される．その際，必要に応じて値の重複を除く．オブジェクトの検索結果ではオブジェクト識別子のみを持ち，元のクラスへのポインタの役割を持たせる．更に，複数のオブジェクトを組み合わせて新しく値の組を作り出すこともできる．この場合，単なる値の集合とするか，新しいクラスのインスタンスとするかシステムで別れる．

関係データベースの問い合わせ結果は必ずテーブルに格納されるのに対して，オブジェクト指向データベースの問い合わせ結果の格納に関しては以下のような選択枝が考えられる．

① 検索結果は検索対象クラスのインスタンス集合のサブセットになるので，元のクラスのサブクラスのインスタンスになる．

② システム定義のクラス（例えば set や bag クラス）またはそのサブクラスのインスタンスになる．その場合，クラスの構造は問い合わせの検索対象指定で決定される．ODMG はこれに対応する．

③ 値の集合（または多重集合）として格納し，クラスと関連付けては管理しない．

問い合わせのスコープに関して関係データベースにはない問題が発生する．すなわち，スーパークラスのインスタンス集合はサブクラスのインスタンス集合を包含するので，スーパークラスを含む問い合わせでは指定されたクラスだけを対象とする場合とそのサブクラスも合わせて対象とする場合が考えられる．更に，スーパークラスを検索の対象とし，サブクラスの属性を一緒に指定した場合，その属性を持っていないサブクラスも対象となる場合がある．それを許すシステムも許さないものもある．

検索対象にはメッセージを含んだパス式を指定することが可能なシステムがある．特にメソッドでしか属性値を参照できないシステムでは，この機能が必須である．その場合，ユーザ定義のメソッドを実行するだけでなく，システム定義のメソッドを指定することもでき，更新操作をそれで行うことができる．しかし，メソッドに副作用がある場合は注意が必要になる．更新その他の操作はプログラムの中で通常のプログラミング言語のように行うシステムが多い．そうすることによって検索結果に対する操作を行うことができる．

前述したように，問い合わせの単位であるパス式は，クラスとその属性の列を指定したものであるから，オブジェクトのナビゲーション（Navigation，参照属性に沿ったオブジェクトのアクセス，リンクトラバース Link Traverse ともいう）が集合的に指定できる．これにより，複数段の属性指定をした場合は，関係データベースの結合演算に相当することを簡略に指定できることになる．もちろん，SQL における結合条件やネストされた問い合わせ（相関問い合わせも含む）も許される．

一般に，問い合わせの結果はオブジェクトの（多重）集合になるので，UNION，

INTERSECTION，DIFFERENCE などの集合演算を適用できる．オブジェクト式の値は一般には（多重）集合になるので，集合と要素の比較や集合同士の比較，更に量限定子（exists，all）を指定できる．パラメータ付きビューや ORDER BY，GROUP BY（+HAVING）や集約関数も利用できる．

複合オブジェクトの検索はパス式でできるし，更にメソッドを起動してその操作もできる．オブジェクトのバージョンを作成できるシステムでは問い合わせ中で特定のバージョンを指定することができる．

図 2･2 に ODMG の OQL 問い合わせの例をいくつか示す．図 2･2(a)では Producer の extent の集合が返される．図 2･2(b)では producer1 という永続的名前の付いた producer オブジェクトの経営者（is_operated_by）オブジェクトが一つ返される．図 2･2(c)では生産者の集合が結果として返される．もし distinct が指定されなければ，多重集合が返される．図 2･2(d)では QualityWinery ワイナリーの生産する製品がすべて ValueCellar 販売店で売られているかチェックする．結果は Boolean 型である．図 2･2(e)ではいったん，struct<string, short>型のリストが昇順に整列されて返されるので，その最初から 3 番目までの要素の集合を結果とする．図 2･2(f)では，メソッドの実行結果の最大値を返す．

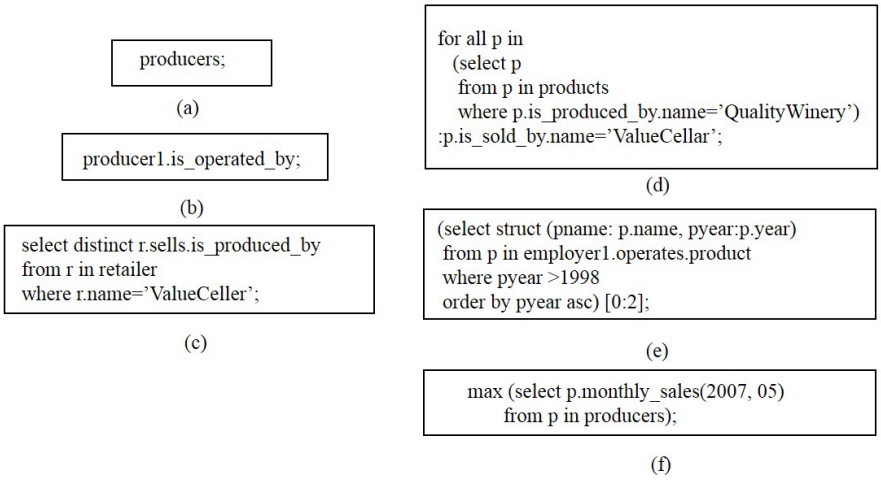

図 2・2　OQL 問い合わせ

問い合わせを効率的に実行する場合は，インデクスが利用できる．インデクスとしては関係データベースと同様に B+木，ハッシュが考えられる．値の大小比較は B+木が適する．値の等号による比較やオブジェクト識別子の比較はハッシュが適する．更に，クラス階層の全インスタンスを対象にインデクスを作成し，利用することや複合オブジェクトの参照関係に対してインデクスを作成して利用することも考えられる．しかし，当然ながらデータの更新が行われるとそれに伴ってインデクスを更新しなければならない．特にクラス階層や複合オブジェクトのインデクスではこの負荷が大きくなる可能性がある．

典型的なセッションでは比較的少ない数の複合オブジェクトを使って比較的長時間にわた

り作業を行う．そこでいったんコアメモリに読み込んだオブジェクトを効率良くアクセスすることに主眼が置かれる．オブジェクト識別子による参照はコアメモリ内では，通常のポインタとして変換することが考えられる．

問い合わせ言語の標準化動向としては ODMG（現在のバージョンは ODMG 3.0）という規格が業界の標準化団体（ODMG）で提案されている．ODMG にはオブジェクト指向データベースベンダーが参加しおり，提案するモデルと言語は C++ や Smalltalk，Java などの複数の言語へのマッピングが可能である．

オブジェクト指向データベースの研究プロトタイプとして，GemStone，Orion，Jasmine などがあった．商用システムとしては現在 ObjectStore [3] や Objectivity [4] などがあり，オープンソースとしては Perst [5] などがある．

■7 群－5 編－14 章

14-3 オブジェクト関係データベース

（執筆者：石川　博）［2015 年 6 月 受領］

オブジェクト指向データベースの出現は関係データベースそのものにインパクトを与えることになった．それは関係データベースにオブジェクト指向機能を追加するという形で実体化した．こうした関係データベースをそれまでのもの（すなわち“純粋な意味での”関係データベース）と区別する意味でオブジェクト関係データベースという．標準に先行する形で商用システムである Informix や Oracle などでオブジェクト指向の機能が追加された．それらを具体的に見てみることにしよう（参考書 [1]）．

まず，オブジェクトの定義について説明する．Oracle ではオブジェクト型（object type）を宣言し，それをカラムの定義に用いて別の表を定義することができる．また，オブジェクト型を使ってタプルそのものを定義するオブジェクト表（object table）も許す．その場合のタプルはオブジェクト識別子を持ち，他の表でオブジェクト型に宣言されたカラムから参照される．オブジェクトのアクセスのために Oracle の SQL でドット式が利用できる．ただし，型階層における継承の機能はない．また，カプセル化の機能もない．**図 3･1**(a)では可変長配列を使ったカラム型を定義し，更にそれを使ってオブジェクト表を定義する．図 3･1(b)は単純な問い合わせ例である．

```
CREATE TYPE retailer_name_t AS OBJECT (retailer_name VARCHAR(32));
CREATE TYPE retailer_list_t AS VARRAY(10) OF retailer_name_t;
CREATE TYPE product_t AS OBJECT (product_name VARCHAR(32), is_sold_by retailer_list_t);
CREATE TABLE product OF product_t;
```

(a)

```
SELECT p.produt_name, p.is_sold_by FROM product p;
```

(b)

```
CREATE TYPE retailer_name1_t AS OBJECT (retailer_name VARCHAR(32), description VARCHAR(32));
CREATE TYPE retailer_list1_t AS TABLE OF retailer_name1_t;
CREATE TYPE product1_t AS OBJECT (product_name VARCHAR(32), is_sold_by retailer_list1_t);
CREATE TABLE product1 OF product1_t;
```

(c)

product1

product_name	is_sold_by	
	retailer_name	description

(d)

図 3・1　Oracle のオブジェクト

多値属性を宣言するには，Oracle の場合は表型（table type）と可変長配列（VARRAY）が利用できる．既に図 3･1(a)では可変長配列を用いた．特に表型を利用するとネストされた表（nested

table）が定義できる．言うまでもないが，これは第一正規形を破ることになる．図 3･1(c)は，図 3･1(a)における配列の代わりに表を使ってオブジェクト型を定義する．これをカラムの定義に用いればネストされた表を定義することができ，図 3･1(d)にそのようにしてできた表を例示する．一方 Informix の場合は，SET，MULTISET，LIST が多値属性を宣言するために提供される．

Informix では行型（row type）が Oracle のオブジェクト型に相当する．Oracle 同様にそれを別の表のカラムの定義に利用できる．また，行型を用いて型付表（typed table）を定義することができ，これは Oracle のオブジェクト表に相当する．Oracle と異なり，Informix では型階層を許す．すなわち，既存の行型のサブタイプ（下位型）として新しく行型を宣言できる．その場合サブタイプは，フィールドとメソッドをスーパータイプ（上位型）から継承する．また，スーパータイプの検索においては，その型のインスタンスそのものを対象とする場合と，それ以下のサブタイプのインスタンスも合わせて対象とする場合を選択できる．**図 3･2**(a)に型階層（liquer と wine）の定義を示す．図 3･2(b)では wine 表も対象になるが，図 3･2(c)では liquer 表だけが対象となる．ただし，属性は，上位型で定義され，下位型でも継承されているものに限定される．なお，Informix にはカプセル化の機能はない．

```
CREATE ROW TYPE liquer(
   pname VARCHAR(32),
   year NUMERIC(4));
CREATE ROW TYPE wine (
   origin VARCHAR(32),
   color CHAR(8))
UNDER liquer;
```

(a)

```
SELECT * FROM liquer WHERE year> 2000;
```

(b)

```
SELECT * FROM ONLY (liquer) WHERE year> 2000;
```

(c)

図 3・2　Informix のオブジェクト

オブジェクトのメソッドの定義は，Oracle では手続き言語（PL/SQL）を提供し，それによってメソッドを定義する．なお，PL/SQL でユーザ定義関数を記述することもできる．一方，Informix ではメソッドの概念はなく，より一般的なユーザ定義関数がその役割を果たす．その定義には手続き言語の SPL を用いる．

実現上ではメソッドが誤りを含んでいる場合でもデータベースに致命的な障害を与えないようにするため，インタプリタで 1 ステップずつ安全性を確認しながら実行することや，メモリ領域を DBMS とは別にすることなどが考えられる．また，効率的実行のためにメソッドの入力や出力をテーブルとしてキャッシュしておくことも考えられる．更に，いったんメモリ上に持ってこられたオブジェクトは繰り返し使われる可能性が高いので，メモリ内でのアクセスが

より効率的となるような方策も必要になる．

イメージ，オーディオ，ビデオなどのいわゆるマルチメディアデータは，BLOB（Binary Large OBject）型として格納される．その最大長は Oracle で 128 TB であり，Informix で 4 TB である．BLOB にアクセスするためには，外部関数であるメソッドやユーザ定義関数を利用する．これを通して BLOB 型のデータにアクセスする限り，カプセル化の機能が提供できる．特にメディアデータの格納には，オブジェクトの BLOB の機能を用いる．BLOB はデータベース管理システムにより，通常の属性とは別に管理される．同様に，ドキュメントなどの巨大な文字列は CLOB（Character Large OBject）型として格納される．

新しいデータ型を定義したら，それに関連してアクセスメソッドをユーザが定義できる必要がある．そこで，アクセスメソッドに関するインタフェースだけをシステムが規定して，実現をユーザに任せる方法や，ユーザによる拡張可能なインデクス構造（例えば GiST 6)）を提供する方法が考えられる．併せて，システムの最適化に対しては，追加したアクセスメソッドがどの条件で使えて，どれだけコストがかかるか，選択率はどれくらいかを知らせる必要がある．

ユーザ定義のデータ型とその操作に関しては，抽象的データ型（Abstract Data Type：ADT）という概念がある．システムでは ADT の型とメソッドのインタフェースしか関知しない．インタフェースを実現するコードはユーザが登録する．通常，ADT は DBMS の特定の応用分野で必要になる．そこで，各商用システムでは，ADT のメソッドのコード並びに型を定義する DDL，アクセスメソッドなどをまとめて DBMS 拡張機能（Data Cartridges や Extender などと呼ばれる）として提供する．

オブジェクト指向データベースとオブジェクト関係データベースとはもちろん共通点を持つが，まとめると以下のような点で違いがある．オブジェクト指向データベースは，プログラミング言語との親和性を重視するものが多い．また，一度に扱うオブジェクトの数はそれほど多くなく，むしろ一つ一つが大きくて複雑である．更に，オブジェクトはメモリ上で繰り返しアクセスされることが多く，それを扱うセッション，もしくはトランザクションは長時間にわたる傾向がある．

最後に個々のシステムではなく，標準としてのオブジェクト関係データベースについて述べる．すなわち，SQL99（正式名 SQL:1999）[7] のオブジェクト指向機能がそれに相当する．SQL99 では行型の宣言を行うことができ，行型を通して表を定義することができる．属性としてオブジェクト（型）への参照型を定義できる．その際に参照整合性を満足しなければならないことは言うまでもない．参照されるオブジェクトへのアクセス（Dereference という）では矢印（->）を用いる．抽象データ型として，内部状態と振る舞い（操作）をひとまとまりとしたユーザ定義型の機能を提供する．類似する複数のユーザ定義型は型階層を作る．更に，CREATE TABLE の際に UNDER 句で指定された上位表（supertable）の副表（subtable）として表を定義できる．型階層を通しては属性と操作の定義の継承と置き換えを許す．ただし，多重継承は許されない．また，多値属性を宣言するために ARRAY 型（SET，MULTISET，LIST は提案では見送られた）が提供される．巨大オブジェクトを格納するために BLOB と CLOB を提供し，位置付け子（ロケータ，locator）により参照される．なおマルチメディア（フルテキスト，空間データ，静止画）に関しては SQL/MM として別に提案されている．

■7 群－5 編－14 章

14-4 データベースの正規化と非正規形データベース

（執筆者：石川　博）［2015 年 6 月 受領］

データベースの論理設計に用いられる関係データベースの正規形について説明し，非正規形データベースについて言及する．一言で言えば，正規化，すなわち関係データベースを正規形にするとは，冗長性を抑えつつ，更新（挿入，修正，削除）に伴う問題点（不整合）を減少させながら，関数従属性と主キーに基づいてリレーショナルスキーマを洗練化するプロセスそのものである．

基本的な正規形には，第一正規形，第二正規形，第三正規形がある．データベースが第一正規形であるとは，属性値には単純な値しか入れることができないということである．言い換えればリレーションの属性の値としてリレーションを許さないことである．ただし，入れ子にしたほうが現実に即している場合がある．そこで，SQL ではカラムに繰り返し項目（Repeating Group）を許している．例えば，既に導入したワイン内容は繰り返し項目を使用すると，ワイン ID の値が複数ある．ちなみに，リレーションを属性の値として含むリレーションを，一般に入れ子になったリレーション（Nested Relation）あるいは非正規形データベースという．

第二正規形であるとは，データベースが第一正規形であり，かつ候補キーによって，残りのすべての属性（候補キーに含まれない属性，非主属性 Nonprime Attribute という）が決められる（関数従属する）ことである．したがって，候補キーの一部 X で，残りの属性 A が決められること（候補キーに関する部分的関数従属性という）があってはならない．

第三正規形は，データベースが第二正規形であり，かつ候補キーが他の属性を決定し，その属性が更に別の属性を決定するというような間接的な決定（それを推移的関数従属性という）はしないことである．言い換えれば第三正規形は推移的関数従属性がないという．

データベースの論理的設計では，通常は元のデータベースを，その冗長性を取り除いて分解（分割）して，この第三正規形（場合によっては Boyce-Codd 正規形）を達成することを目指す．その際に分解されたデータベースが持たすべき条件は以下の 2 点である．

① 分解によってできたデータベースを自然結合すると，不必要な（Spurious）タプルを生成せずに元のデータベースを完全に復元できる．

② 元のデータベースが満たすべき制約（ここでは関数従属性）のチェックは，すべて分解後のデータベースに対する制約をチェックすることで行える．

そもそもこうした正規形が導入された主な理由は，それがないと冗長なデータを格納すること（Redundant Storage）になるばかりでなく，更新操作（挿入，修正，削除）に際して以下のような不整合（Anomaly）が起きてしまうからである．

① **挿入不整合**（Insertion Anomaly）：関数従属性に関連するデータがすべて決まらないとタプルを挿入できなくなる．

② **修正不整合**（Update Anomaly）：1 つの関数従属性の情報を正しく維持するために，複数のタプルを修正しなければならなくなる．

③ **削除不整合**（Deletion Anomaly）：タプルの削除に伴って必要な関数従属性の情報が失われてしまう．

上記の 3 つを総称して更新不整合（Update Anomaly）ということにする．更新整合性は第一

正規形に一般に起こりうる．特に，第一正規形で許される部分関数従属性があると，タプルの挿入，修正，削除で．同様な不整合が起こる．例えば，少なくとも AB→C，B→C という 2 個の関数従属性があるとする．一方で ABC を含む表があるとする．この場合は ABC を含む表で B が現れる回数だけ C が現れて冗長になり，また修正が複数個所必要になる．ABC という値の対が決まらなければ，B→C に対応する値の対も挿入できない．ABC に対応する値の対が削除されると，B→C の一部が失われてしまう場合がある．

更に，第二正規形で許される推移的関数従属性があるとタプルの挿入，修正，削除で同様な不整合が起こる．例えば，A→B→C という推移的関数従属性があると，ABC を含む表で B→C に対応する値の対が決まらないと，A→B に対応する値の対も挿入できない．タプル全体が削除されると，含まれる関数上属性も失われてしまう場合がある．

■参考文献

1) 石川　博：“データベース,”情報工学レクチャーシリーズ，森北出版, 2008.
2) http://www.odbms.org/ accessed 2015
3) http://www.objectstore.com/ accessed 2015
4) http://www.objectivity.com/ accessed 2015
5) http://www.mcobject.com/perst accessed 2015
6) J.M. Hellerstein, J.F. Naughton, and A. Pfeffer：“Generalized search trees for database systems,”in Proceedings of the 21st International Conference on Very Large Data Bases, Zurich, Switzerland, 1995.
7) ジム・メルトン，アラン・サイモン：“SQL: 1999 リレーショナル言語詳解,”2003.