愛知大学、白樺高原ロッジにおけるウェブ宿泊予約システムの開発（1）
データベースアプリケーションにおけるオブジェクト指向モデリングと実装

蒋　湧、堀井　聡、多賀　康裕
愛知大学経済学部

Development of Web Reservation System for Sirakaba Lodge in Aichi University (1)
Object-Oriented Modeling and Implementation of Database Application

Yong Jiang, Satoru Horii and Yasuhiro Taga
Faculty of Economics, Aichi University

目次

1. はじめに

1.1. 背景

近年、激化するグローバル競争は着実に進んでいる。競争に打ち勝つため、企業グループは連携や統合、再編を繰り返している。各企業は自社の得意分野に経営リソースを集中させるとともに、戦力的パートナーシップやアウトソーシングを通して外部リソースを有効に活用することが迫られている。経済環境の急激な変化で、企業と顧客の関係も著しく変化してきた。従来の大量生産、大量消費の時代と違って、顧客指向の経営戦略やビジネスプロセスへの企業経営改革が求められている。

このようなビジネス環境の変化に迅速に対応するために、企業の間に（Business to Business）および企業と顧客の間に（Business to Customer）、生産・流通・経営・消費にわたる分散システム情報を迅速かつ柔軟に連携させる技術が望まれている。最近注目されるウェブアプリケーションは、それらの要求に応え、主に三つの連携技術を提供している[1]。

- 高性能な連携機能を備えたアプリケーションを低コストで実現するために、既存のインターネット／イントラネットを利用した通信技術を提供する。
- 相互運用性を高めるために、特定のベンダ OS やミドルウェアに依存しない、標準インターネット通信プロトコルや XML データ形式などを準拠したフレームワークを提供する。
- システムの開発、運用と保守において、多変な外部環境に素早く、なおかつ柔軟に対応するために、オブジェクト指向の開発法や分散システムにおけるテクノロジーを開発し、提供している。

本文は、愛知大学白樺高原ロッジ宿泊予約システムの開発過程、つまりシステムモデリング、設計・実装の過程を通して、注目されているウェブアプリケーションの仕組みを紹介する。

1.2. ウェブアプリケーションの特徴

ソフトウェアエンジニアリングの側面から、ウェブアプリケーションのアーキテクチャーとその開発テクノロジーを中心にし、ウェブアプリケーションの特徴を述べる。

- ウェブアプリケーション・アーキテクチャー

  標準的なアーキテクチャーは三つの層、つまり Web Client 層、Web Server 層と Database Server 層によって構成されている（図 1-1）。Web Client はインターネットに接続し、Internet Explorer や Netscape などウェブ・ブラウザーを用いて、Web Server へ Request を出す。Web Server は、その Request に応じて、必要な処理を行う。場合によって、Database Server へアクセスし、必要なデータを抽出する。最後に、Web Server は HTML 形式で Response を書き出す。この Response はインターネットを経由し、Web Client のブラウザー上に表示される。

このような三つの層に分離された構造は、いくつか重要な特徴を持っている。まず、Web Client と Web Server の分離で、Business Logical といわれる主な処理が Web Sever に集中したことで、セキュリティ管理などを含むシステムの Performance を大幅に向上させ、システムの保守・拡張の面にも大きなメリットが生まれる。特に、ウェブアプリケーションの利用は、Web Client 側の機種と OS に依存せず、ウェブ・ブラウザーさえあれば、利用可能である。Window Application のようなソフトウェアの配布コストは発生しないことも大きなメリットである。

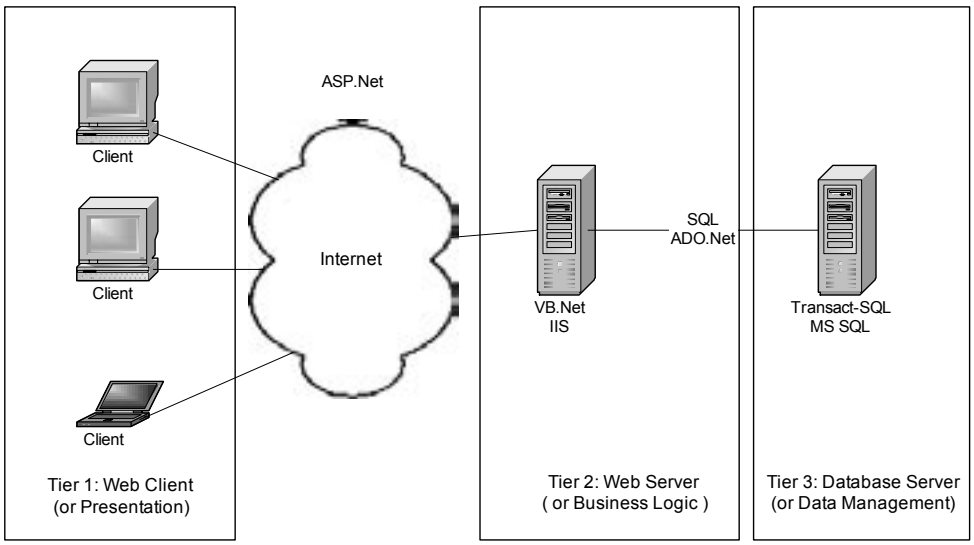


図 1-1　3 層システムアーキテクチャー

次に、Web Server と Database Server の分離で、Business Logical の処理とデータベースクエリの処理を分散することとなり、システムの安定性、保守性と拡張性に大きなメリットをもたらす。例えば、データベース処理機能を集中し、カプセル化することが可能となる。これはオブジェクト指向のシステム構造に欠かせないことである。また、高頻度なデータベースクエリの処理がデータベース・サーバーに集中することによって、ネットワーク上のデータ転送量が減らされることや、Stored Procedure を用いたキャッシュ処理など、システムの効率とセキュリティ管理に大きなメリットが得られる。

■　ウェブアプリケーション・テクノロジー

今回白樺ロッジウェブ予約システムの開発に当たって、マクロソフト社が提唱した.Net Framework テクノロジーを利用した（図 1-1）。Web Client と Web Server の間に、ユーザーの Request に応答し、Dynamic な HTML ページを返すことは、ASP.Net によって統合管理される。そのうち、

Web Page のサービスは IIS で提供し、ビジネスロジックの処理は VB.Net プログラムでコントロールする。ウェブアプリケーションとデータベースとの連携は、ASP.Net で提供されたデータコントローラ（例えば、DataGrid や DataList など）と ADO.Net によって実現される。高頻度なデータベースクエリ処理は、データベース・サーバーMS SQL2000 が提供する Stored Procedure で処理される。

1.3. 開発環境

実験に当たって、以下の開発環境を利用した。

Web Client	OS	Windows XP Pro.
	Browser	Internet Explorer 6.0
Web Server	OS	Windows Server 2003 (IIS 5.0, .Net Framework 1.0)
	開発ツール	Visual Studio.Net 2003
		Macromedia MX Studio
Database Server	OS	Windows Server 2003
	DBMS	MS SQL 2000

表 1-1 システムの開発環境

この論文は、図 1-1 に示された第 2 階層の ADO.Net と第 3 階層のデータベース設計・実装、Stored Procedure によるプログラミングを中心とする。第 2 階層の ASP.Net やオブジェクト指向のプログラミング法については、次回に論じる。本文の第 2 章からの構成は次のようになる。

２章　予約システムの概要

３章　データベースシステムの設計

４章　ADO.NET の導入

2.　予約システムの概要

この章では、白樺ロッジの利用規定、従来の予約手順とウェブ予約システムへの移行、三つの節から予約システムの概要をまとめる。

2.1. ロッジの利用規定

愛知大学白樺高原ロッジは、愛知大学の厚生施設として、表 2-1 で示された利用者が利用することができる。

本学学生	・学部生　・短大生　・大学院生 ・研究生　・科目等履修生 ・留学生別科生　・オープンカレッジ生
本学関係者	・本学教職員及びその家族 ・本学後援会（学生の父母）及びその家族 ・本学同窓会会員及びその家族
その他	本学学生又は本学関係者が、他大学学生や知人と一緒にロッジを利用する場合の「他大学学生や知人」

表 2-1　白樺高原ロッジの利用者について

ロッジの利用料金は、利用者区分によって定められている（表 2-2）。部屋のタイプは、Aタイプと Bタイプ、二種類あるが、利用料金は均一。利用料金にかかわるもう一つの要因は、利用季節である。10 月から翌年の 5 月までの利用料金は、暖房費のため 300 円増になる。食事は、朝食、夕食、特別食の 3 種類が用意されていて、特別食を申し込む場合、別料金として 800 円がかかる。

区分	1 人 1 泊 2 食付利用料金		備考
	6 月～9 月	10 月～5 月	■ 10 歳未満は半額、3 歳未満は無料。 ■ 利用料金には朝食、夕食を含む。特別食（800 円増）を希望する場合は予約時に申し込む。 ■ （）内金額は施設利用料金（素泊り）である。 ■ チェックイン 14:00　チェックアウト 10:00 でる。 ■ 10 月～5 月は暖房費 300 円を含む。
本学学生	3,000 円 (1,500 円)	3,300 円 (1,800 円)	
本学関係者	4,000 円 (2,500 円)	4,300 円 (2,800 円)	
その他	5,000 円 (3,500 円)	5,300 円 (3,800 円)	

注：学生の男女同室は認められない

表 2-2　白樺高原ロッジの利用料金について

利用日数は、原則として 1 回について 3 泊 4 日としているが、利用最終日において以後の予約が無いときは延長できる。

また、予約期間は、利用期間と利用目的によって変わる。

利用目的 / 利用期間	正課活動に関わる利用目的	それ以外の利用目的
夏期休暇期間中	３ヶ月前から予約ができる	２ヶ月前から予約ができる
夏期休暇期間以外	５ヶ月前から予約ができる	

注：夏期休暇期間は7月１９日～9月２０日とする。

表 2-3　予約期間について

最後に、予約金に関わる規定について、以下のようにまとめた。

- 予約日が利用開始日より１ヶ月以上前の時は、予約金を払わなければならない。
- 金額は１人１泊１０００円である。
- 振込みをした後で、７日以上前にキャンセルをした場合については、振り込み手数料を差し引いた額を返金する。

2.2. 従来の予約方法

上述の予約規定を踏まえ、従来の予約手順は以下のように行われている。

①代表者がロッジに直接電話をする

②宿泊したい日付を伝える

※１　この時に夏期休暇中の予約であるならば正課活動かどうかを伝えなければ、３ヶ月前の予約はできない

※２　学生の場合は男女別人数を伝える

③数日後に予約確認書（郵送）が送られてくる

④予約確認書に従い、予約金を振り込む

・但し、予約日が利用開始日より１週間以内の時は、確認書は送られてこない。

以上が従来の予約の方法である。

2.3. ウェブ予約システムへの移行

ロッジ周辺は交通機関が未発達なため、従来の予約手順において、いくつかの不便を感じる。例えば、予約者にとって、リアルタイムの予約状況は確認できないこと、郵送で送られてくる予約確認書は時

間がかかることなどが挙げられる。管理者側から見れば、なんと言っても予約のためにかかった手間とコスト（電話代や郵送代など）は大きい。

それらの問題点を解決するために、ウェブ予約システムの開発に臨んだ。以下に 2 つの事例を通して、ウェブ予約システムを利用した予約手順とチェックイン手順を示す。

【事例 1 ：予約者：愛知太郎君、時間：2004 年 1 月 17 日、作業内容：初めての予約】

まず、ログインページ（図 2-1）にアクセスし、「新規登録」をクリックする。

図 2-1　Login と新規登録

現れた新規登録画面に（図 2-2）、個人情報を入力し、「新規登録」ボタンを押す。そうすると、愛知太郎君の個人情報は、予約システムのデータベースに保存される。

次に Login 画面（図 2-1）が再び現れ、ログイン ＩＤとパスワードを入力して、ログインをする。

ログインが成功した場合、図 2-3 の個人環境画面が現れ、ユーザー本人の氏名と ID がページのトップに表示される。個人の環境の下では、予約、予約変更、予約削除、個人データの変更の 4 つのユーザー機能を備えている。次に、予約機能だけを紹介する。その他については、http://pegasus.aichi-u.ac.jp/Sirakaba/Login.aspx で確認できる。

予約画面の左辺に予約用のカレンダーが配置されている。画面右辺のテーブルに、カレンダーに選択された日の各部屋の予約状況が表示されている。三つの予約状態、すなわち予約済み、予約中と予約可能、が考えられる。既に予約された部屋は、赤色の「予約済」で表示する。ただいま誰かが予約している（まだ決定されていない）部屋は、緑色の「予約中」で表示する。「予約済み」と「予約中」の部屋について、再び予約することはできない。

図 2-2　新規登録画面

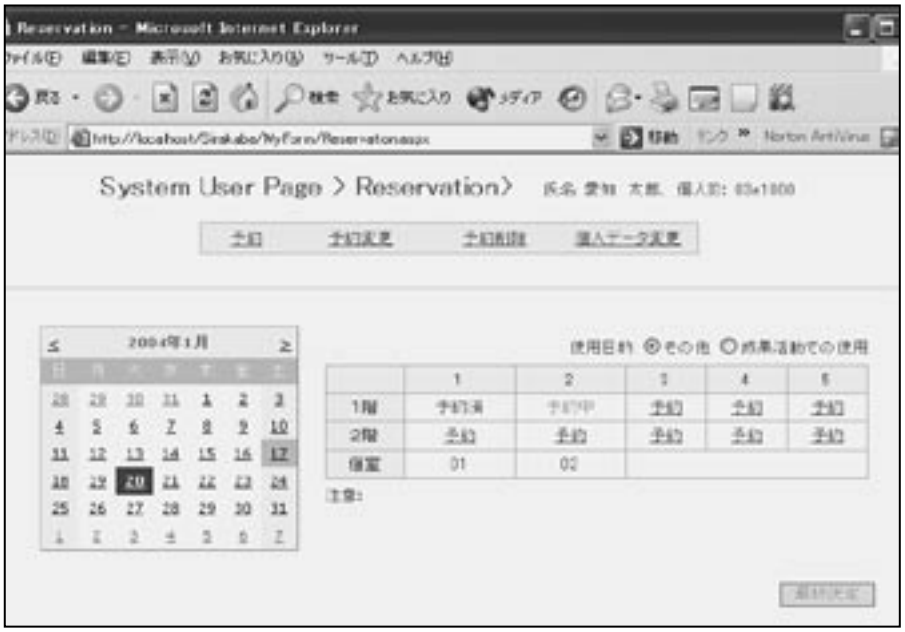


図 2-3　予約状況の確認

図 2-4　予約画面

予約可能な部屋は、黒色の「予約」ボタンが配置されている。これをクリックすると、該当する部屋の予約作業が始まる。

1 回の予約につき、複数の部屋を予約することも可能である。「予約」ボタンを押すと、画面の下部に予約テーブルが開かれ、そこに、部屋ごとの宿泊数、宿泊人数と食事の予約に関するデータを記入することができる（図2-4）。そのとき、以下の３点を特に注意する必要がある。

A)　現時点で予約している部屋は、緑色の「予約中」と変われる。この部屋の予約作業はただ今進行中であることを他のユーザーにも知らされて、同じ部屋の重複予約はできないようになった。

B)　部屋ごとの予約期間は最大3 泊4 日としている。しかし予約状況によって、部屋の空いている期間は3 泊4 日より短い可能性がある。例えば、愛知太郎君は1 月20 日の2 階2 号室（部屋番号 22）を指定しているが、この部屋が今後、何日間空いているかについて、明確に表示することは重要である。

C)　宿泊数を決めた場合、食事の予約はそれに合うように行われるかどうかのチェックは欠かせない。

予約データを記入すると、部屋ごとに「確認」ボタンを押す。すると、上に述べた B)と C)がチェックされ、問題がなければ、「予約可能」とのメッセージが表示される。そうではない場合、エラーメッセージが現れて、予約のやり直しへ進む。

ここまでの作業は、「仮予約」と呼ぶ。仮予約が成功したら、「最終決定」ボタンが使用可能の状態になる（仮予約に失敗した場合、「最終決定」ボタンは使えない状態にある）。

「最終決定」をクリックすると、予約データはシステムのデータベースに書き込まれると同時に、システムが予約者宛に予約確認メールを送信する。

図 2-5　予約後の画面

【事例２：ロッジ管理者：愛知　管理様、時間：2004 年 1 月 20 日、作業内容：チェックイン】

図 2-6　管理者のログイン

管理者である愛知管理様がログインすると、図 2-7 のような管理者ページが開かれる。

図 2-7　当日の予約者のリスト一覧

この日の日付は、2004 年 1 月 20 日であることを注意してほしい。画面上の「今日の予約リスト」に、当日部屋を予約している予約者のリストと予約している部屋の数が表示されている。そこで、前日に予約をしていた愛知太郎の名前が載っている。

図 2-8　予約詳細の確認

予約内容の詳細を確認したい場合、「予約詳細」を押すと、次の画面が現れる（図 2-8）。予約詳細に、愛知太郎君が予約した 2 つの部屋名と部屋ごとの予約期間、宿泊人数が表示されている。
部屋ごとの宿泊者の氏名を記入するために、「Check In」ボタンをクリックすると、図 2-9 のようになり、チェックイン作業が行われる。

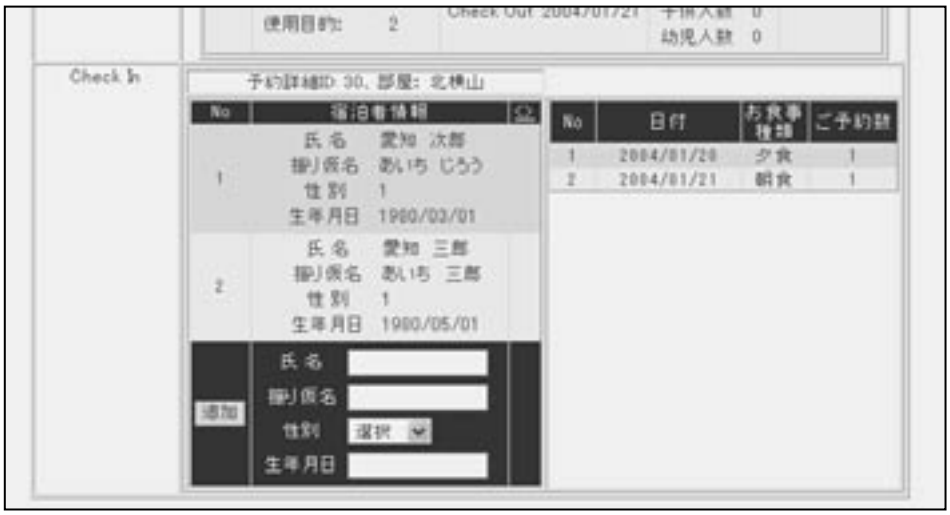


図 2-9　チェックイン作業の画面

予約システムの概要を紹介した後、次にシステムの設計と実装に移る。

3.　データベースシステムの設計

ウェブデータベースを含むアプリケーション設計の第１段階として、要件の収集、分析、データモデルの構築、UML によるクラスモデルの構築など過程が含まれる。

3.1. ユースケースによる要件の分析

ユースケース図（図 3-1）は、システムを果たすすべての機能とシステムの外部環境を把握するのに役に立つ。図に現れる人型の図形はアクターと呼ぶ。アクターとは、「システムの外側にあるオブジェクトの役割である」と定義されている。予約システムの場合、予約と関わりのあるユーザーの役割を表す。例えば、ゼミ長が代表として、ゼミ合宿を予約する場合、ゼミ長の役割はシステムの利用者（予約実行者の意味）であると同時に宿泊者でもある。合宿に参加するゼミ生は単に宿泊者になる。ロッジの管理者は、もちろんシステムの管理者である。

ユースケース図に使われる楕円図形は、システムを提供する機能、或いはまとまった仕事を表す。アクターと楕円の間に結ばれた直線は、ユーザーと仕事との関連を示す。

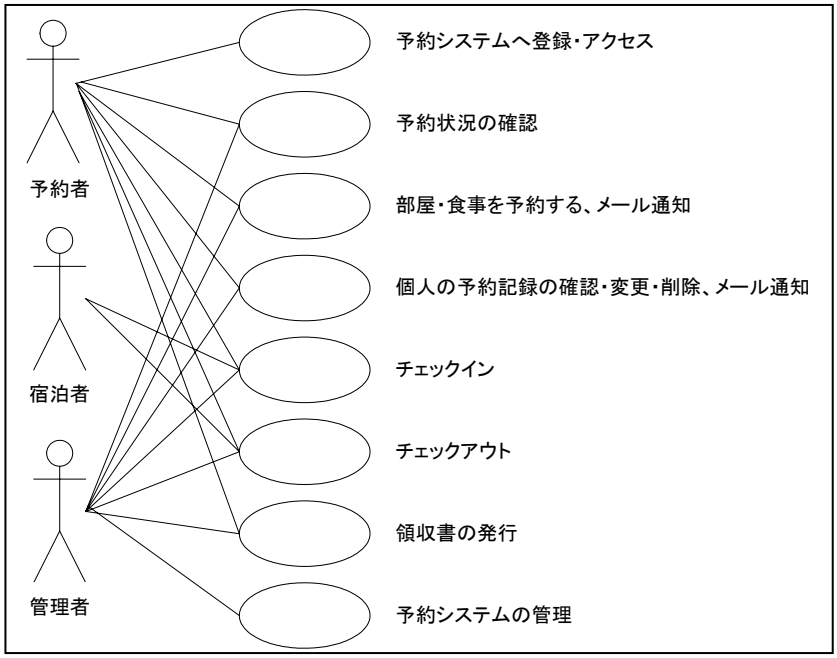


図 3-1 ユースケース図

図 3-1 のユースケース図によって、予約システムの要件は以下のように分析することができる。

- システムの利用者は、予約者、宿泊者、管理者の三つの役割（Role）を持っている。一人のユーザーは複数の役割を持つことはできるが、システムに登場する際に、そのうちの一つの役割を果たす。
- 予約者は、システムの利用者であり、新規利用する場合、ユーザー登録が必要となる。その後、ユーザー認証が成功したら、図 3-1 に示された一連の予約作業が行える。
- 宿泊者は、間接的にシステムを利用している。チェックインとチェックアウトの際に、宿泊者名簿への記入や宿泊料金の勘定などの作業がある。しかし、これらの作業は、管理員のもとで行われるので、宿泊者によるシステムのアクセスはない。
- 電話での予約も配慮し、管理員は予約者の仕事を兼任することができる。更に、管理者は、ユーザー管理や領収書発行などの作業も行える。

ユースケース図から予約システムの提供すべき機能を洗い出すことができる。システムの開発者にとって、ユースケース図に現れた機能を整理し、予約システムの初期モデル：サブシステム構成モデルを設けることは、システム開発に向けて大切なステップである。図 3-2 と表 3-1 には、それぞれ予約システムのサブシステムと対応する機能が表示されている。

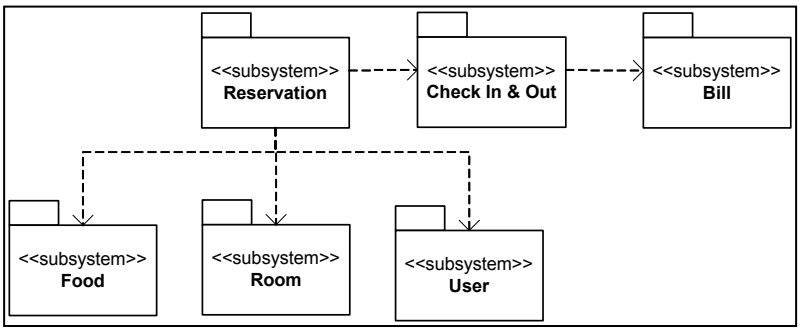


図 3-2 サブシステムの構成図

サブシステム	提供される機能
User	ユーザー登録、編集、認証
Room	部屋タイプ、料金の設定、編集
Food	食事のタイプ、価格の設定、編集
Reservation	予約状況の確認、予約、予約後の編集、キャンセルなど
Check In & Out	チェックインとチェックアウトの手続き
Bill	勘定、領収書の発行など

表 3-1 サブシステムとその機能一覧

3.2. データベース・アーキテクチャー

データベース中心型システムを、内部モデル、概念モデルと外部モデルに分割し、モデルの記述（メタモデル）をスキーマと呼び、このような3階層のモデル構造は、3 階層スキーマ・アーキテクチャーと呼ぶ（図 3-3）。

概念スキーマは､データベース全体のデータ構造を物理的な実現方法とは独立に定義したものであり、データモデルの構造、操作と制約によって構成される。例えば、オブジェクト関係データベースの場合、関係構造とクラス構造がここに入る。外部スキーマは、利用者やアプリケーションから直接見えるデータビューであり、アプリケーションの設計要望を念頭においたデータ定義である。内部スキーマはデータの格納、または物理的な実現の仕方からの定義。

図 3-3 に示されたように、アプリケーションから見たデータベースの「顔」は、最上層の外部スキーマによって定義されたデータビューである。最低層の内部スキーマこそ、データベース上のデータ構

造を表現している。この最上層の「顔」と最下層の「裏」がお互いに独立になっていることは、データの独立と言われ、データベースの管理（保守・拡張）および設計に極めて大きな意味を持つ。

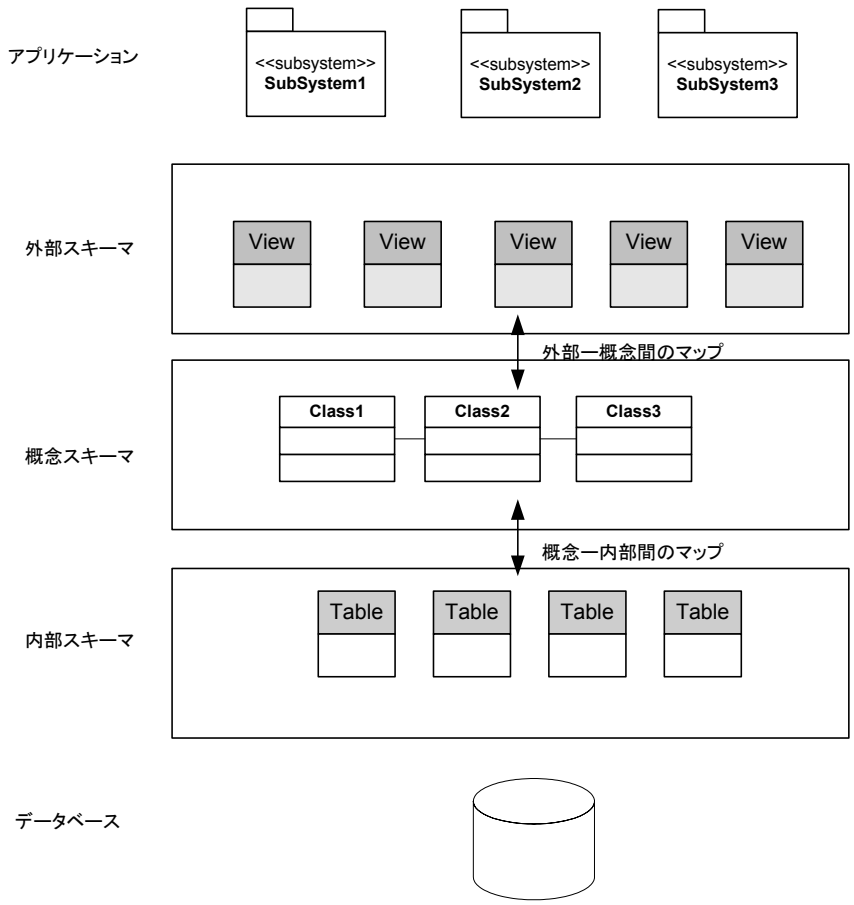


図 3-3　データベースの 3 階層スキーマ・アーキテクチャー

最上層と最下層の独立によって、仮にアプリケーションシステムの構造を変更した場合、その影響は、概念スキーマ層によって吸収され、最下層のデータベースの物理構造には及ばない。逆に、既存の最上層アプリケーションに影響を与えずに、最下層のデータベースの物理的な構造だけを変更できるようになったのである。このデータの独立をもたらすアーキテクチャーと、次節で論じるデータ構造のカプセル化は、大規模なウェブアプリケーション・プログラミングに欠かせない要素である。

3.3. 概念スキーマの設計

概念スキーマは、上層と下層の中間にはさまれ、適切に設計された理論データ構造とマッピング機能を持つことで、システム構造とデータベース構造の差異を吸収する。概念スキーマの設計は、データ構造設計の中心であり、設計目標は、すべて異なるアプリケーションとこれらのデータ関連ニーズを理論的にサポートすることである。次に、アプリケーションのサブシステムごとに、予約システムの概念スキーマを示す。

3.3.1. ユーザー・サブシステムのデータ構造

ユースケースで分析したように、システムの利用者として、宿泊者、予約者、管理者がある。システムでの役割 (Role）によって、三者の属性と操作は異なっている。

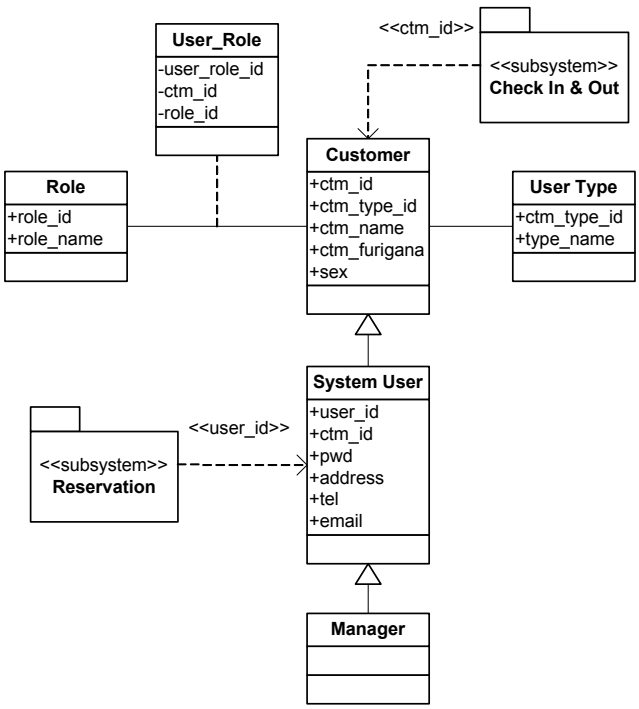


図 3-4　ユーザー・サブシステムのデータ構造

クラス Customer には、氏名、性別などすべてのユーザーが持っている共通属性が集まっている。そのうち、User Type はユーザーの身元属性を示す。ユーザーは、学生であり、教職員であり、或いは OB（OG）であり、学外関係者である。すべでのユーザーは、唯一（One and Only One）の身元属性を持

つことであるので、クラス Customer とクラス User Type の間に多対一の関連を持つ。更に、この身元属性によって、宿泊の料金単価と予約期間が変わるため、外部サブシステム Reservation と Check In & Out は、クラス Customer と System User に依存していることを明記している（図 3-4）。

User Type と違って、User Role はユーザーの役割を表す。一人のユーザーが複数の Role を果たすことができるので、クラス Customer と Role の間に多対多の関連になる。そのため、関連クラス User_Role が導入され、ユーザーが演じる役割の詳細を記憶しておく。

また、クラス Customer、System User、Manager の間には汎化関係が示されている。クラス Customer は全ユーザーの共通属性を「汎化」し、普通の宿泊者クラスに当たる。宿泊者は、直接にシステムを利用しないので、属性に PWD や Email などを入れる必要はない。宿泊者と対照的に、予約者は、直接、システムを利用して予約や編集などの作業を行うので、Login のための PWD は欠かせない。また、ロッジからのメール連絡と After Service のあて先は予約者に指定されるので、連絡先の情報も必要である。従って、クラス System User はクラス Customer の特化クラスとして、Customer の属性を継承しながら、予約者としての特性を追加しておく。現段階では、管理者と予約者を同じデータ構造を持っていることを考えている。しかし、管理者が予約者により多くの操作権限を持っていることは、間違いないだろう。クラスの操作部分の紹介は次回に譲る。

3.3.2. 予約サブシステムのデータ構造

まず、予約システムには、予約者の情報は欠かせないので、User サブシステムに依存している。また、予約情報はチェックイン時に必要となり、Check In & Out サブクラスに影響を与える（図 3-5）。

予約に当たって、利用する部屋の情報、食事の情報、利用目的（それによって、予約期間が変わる）などが必要になるので、それぞれ Room、Food Type と Purpose 関係クラスを設けた。また、ウェブ上の予約は、同時に多数の人が予約できるので、他人が予約している情報を知ることも（予約確定の前の情報）極めて重要である。これらの情報は、クラス Temp Reservation でとられる。クラス Reservation と Reservation Detail は予約情報を扱っている。具体的に、一件の予約事象に対し、一つの Reservation インスタンスが生じる。この一件の予約事象に対し、複数の部屋を予約することもありうる。これら部屋ごとの予約情報は、クラス Reservation Detail で扱う。

予約作業の基本属性として、部屋 ID、チェックイン・チェックアウトの日付、泊まる人数などが挙げられる。クラス Reservation Base はこれらの基本属性を集約している。クラス Reservation Base は抽象（abstract）クラスであり、継承専用のクラスとして、インスタンスは作れない。Temp Reservation と Reservation Detail は、Reservation Base を継承し、インスタンスを作り出す（図 3-5）。クラス Reservation Base を設けることで、プログラムコードの再利用性が向上される。

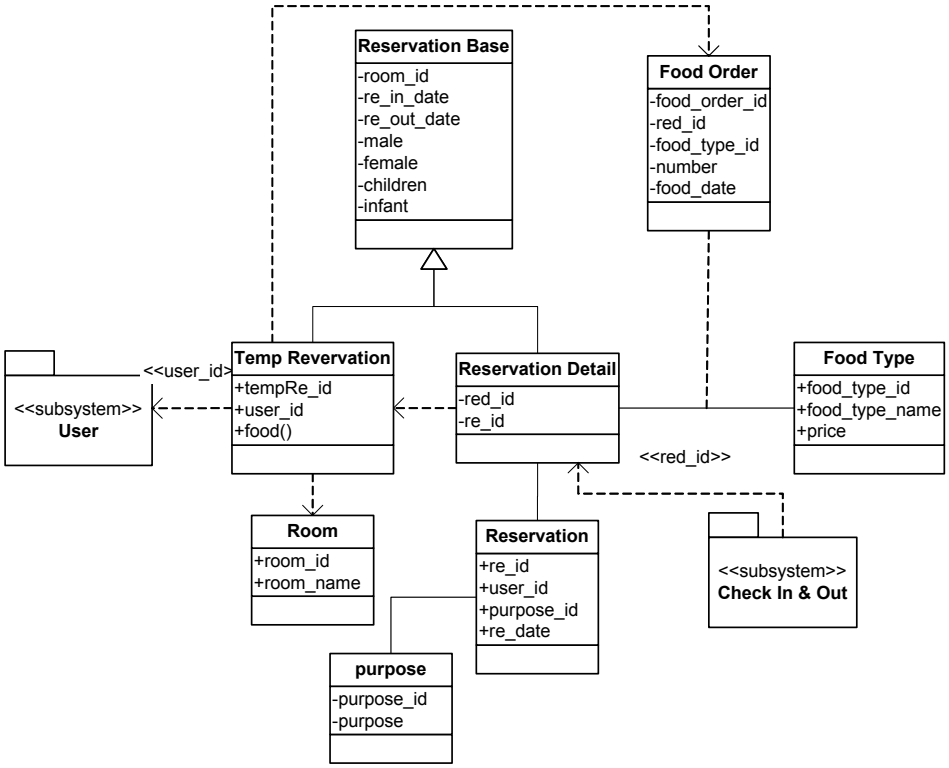


図 3-5 予約サブシステムのデータ構造

3.3.3. チェックイン・チェックアウト・サブシステム

予約者 ID と予約詳細 ID がわかれば、チェックインすることはできる。Check In & Out サブシステムは、User サブシステムと Reservation サブシステムに依存する（図 3-6）。

予約サブシステムから、部屋ごとの予約情報（チェックイン・アウト日付、宿泊人数、食事数）を取得し、チェックイン作業を行う。チェックインは、宿泊者ごとに行われ、個人情報や宿泊期間（check_in_date と check_out_date で定める）を記憶しておく。

宿泊者ごとの食事は、クラス Food Confirm で扱う。チェックインの時、食事の Order を確認すると、毎日厨房に詳細な食事リストを出せる。また、チェックアウトの時、Food Confirm から個人ごとの食事代の勘定ができる。

予約サブシステムの Food Order クラスは、Food Confirm クラスと違う性格を持っている。Food Order は宿泊期間内、部屋ごと複数の宿泊者における食事の Order 総数である。この数値は、ロッジ側の食

料の事前準備に役に立つ。Food Confirm クラスは、宿泊者全員における毎日二食のリストであり、料理の準備と個人ごとの食費の算出に役に立つ。

チェックアウトの日に、check_out_date を生じ、利用した食事の最終確認ができる。

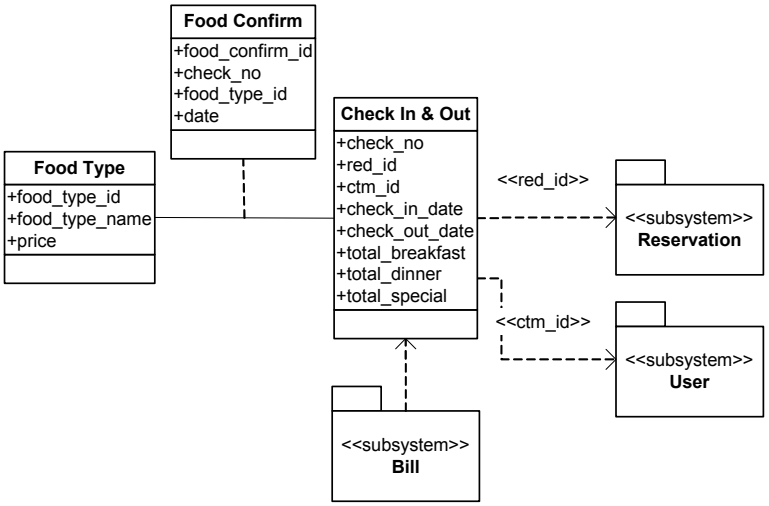


図 3-6　チェックイン・アウト・サブシステムのデータ構造

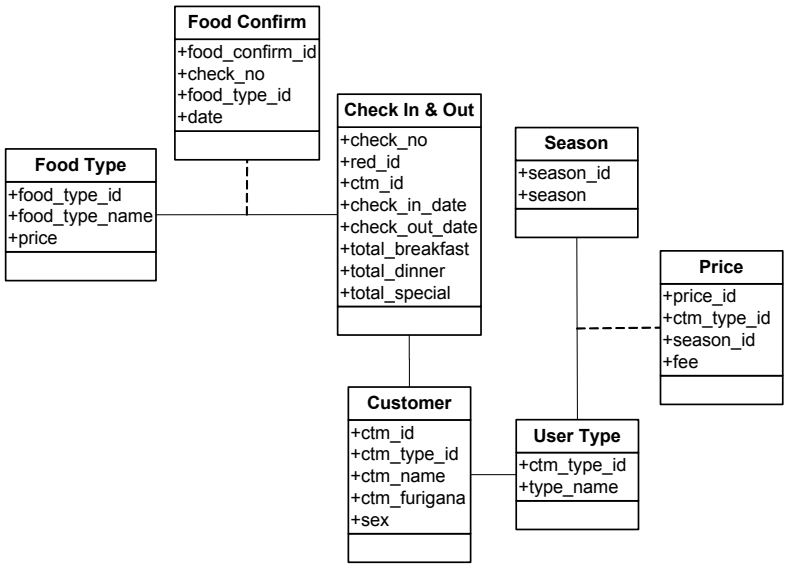


図 3-7　　勘定サブシステムのデータ構造

3.3.4. 勘定サブシステム

勘定サブシステムは、チェックイン・アウト・サブシステムの拡張として定義する（図 3-7）。ロッジの宿泊単価は、ユーザータイプと宿泊期間によって決めるので、Price クラスは Season クラスと User Type クラスの関連クラスとして定義される。個人ごとの勘定は、宿泊単価、宿泊期間と食費によって算出される。

図 3-8 は予約システム全データ構造を示す。

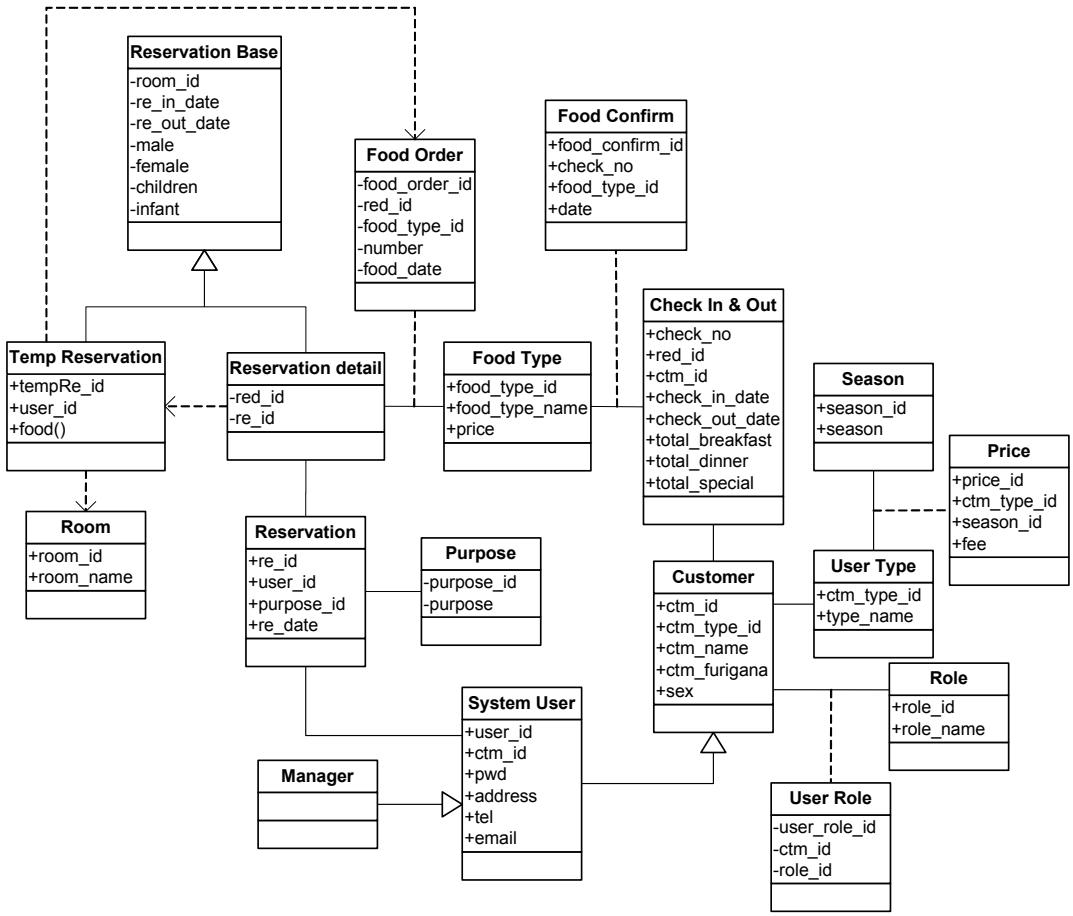


図 3-8 予約システムのデータ構造

内部スキーマとして、データベースの構造は以下のようになる。

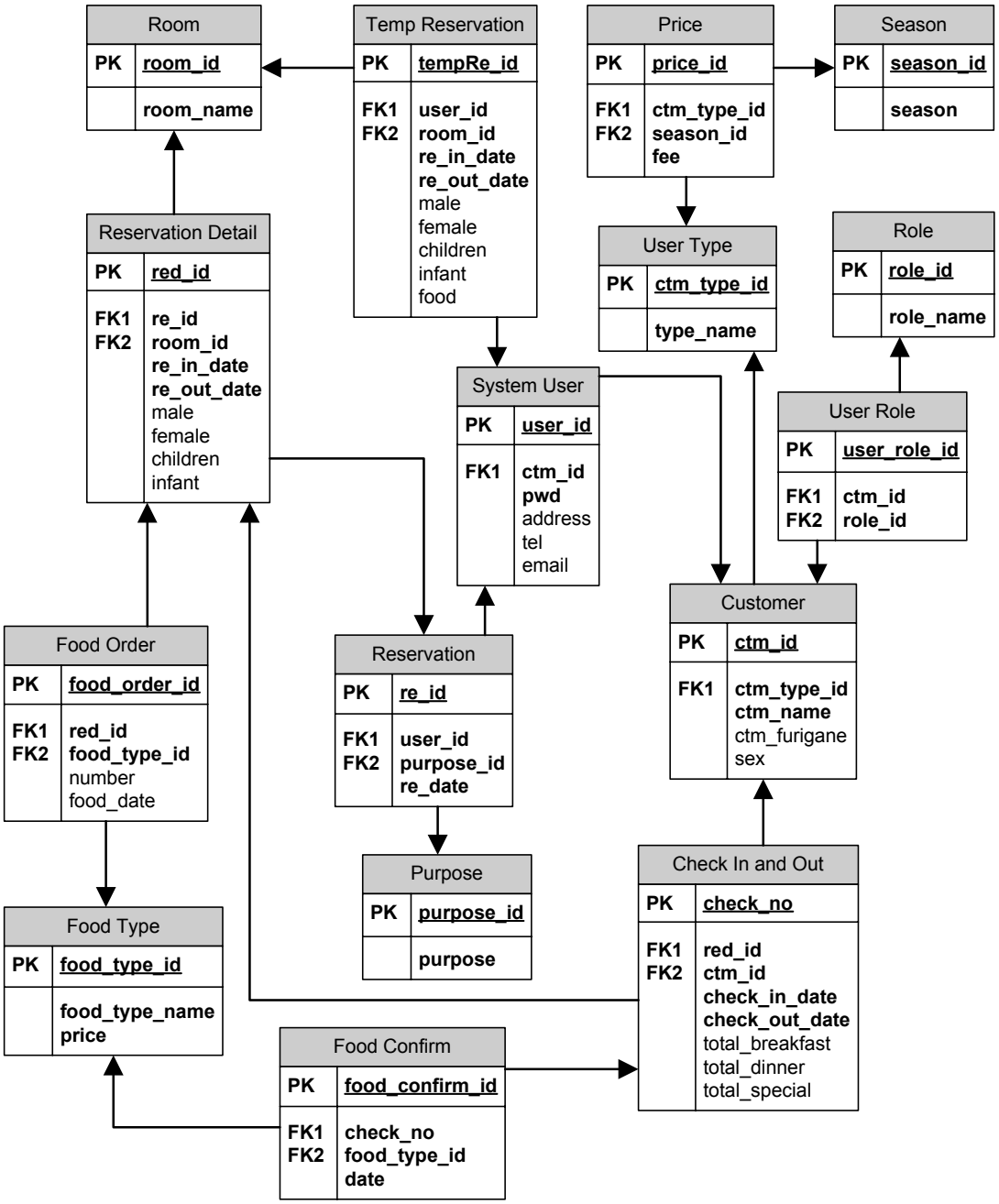


図 3-9　データベースの構造

3.4. データベースの実装とデータビューの作成

データベースの実装を始めるためには、アクセス対象となるデータベース製品、つまり、データベース管理システム（DBMS）が必要となる。本予約システムは、Microsoft SQL Server 2000 を使用し、データベース・プログラミングの実験を進めてきた。データベースの実装作業には、テーブルの作成、テーブル間のリレーション関係の定義、データビューの作成などの作業を含んでいる。Microsoft SQL Server 2000 と Microsoft Visual Studio.Net 開発環境を利用した場合、データベースの実装作業は以下の二通りがある。

- SQL Server に付属する GUI ツール「SQL Server Enterprise Manager」による実装
- Visual Studio.NET のサーバ・エクスプローラによる実装

SQL Server Enterprise Manager によるデータベースの実装方法については、文献「2」と「3」を参照することができる。次は、Visual Studio.NET（以下略 VS.Net）のサーバ・エクスプローラを利用したデータベースの実装と管理について解説する。

図 3-10　サーバ・エクスプローラ

サーバ・エクスプローラ・ウィンドウを開いて、そこに「Data Connections」と「SQL Servers」があり、MS SQL2000 Server がインストールされている場合、どちらからもデータベースに接続することができる（図 3-10）。MS SQL Server ではなく、MS から提供された無料の MSDE（正式名称は Microsoft SQL Server 2000 Desktop Engine）を利用された場合、「Data Connections」からデータベースと接続できる。本文は SQL Servers がインストールされているケースについて解説する。

「SQL Servers」フォルダを開くと、SQL Server Enterprise Manager と同じように、データベースの一覧が確認できる。その中「SirakabaLodgeDB」は、予約システムのデータベースである。この下層に

はデータベースに含まれる「Database Diagrams」、「Tables」、「Views」、「Stored Procedures」、「Functions」が格納されている（図3-11）。

図 3-11　サーバ・エクスプローラから見たデータベースの構造

新規テーブルを作成したい場合、「Tables」フォルダを右クリックし、「New Table」コマンドを選択すれば、開発環境のメインフォームに、「SQL Server Enterprise Manager」と同じテーブル新規画面が開ける。作成されたテーブルの一覧が、「Tables」フォルダの下層に格納されている。例えば、Customerテーブルを選択すると、その下層に、Customerテーブルに含まれた全てのColumnが表示されると同時に、開発環境のメインフォームにおいて格納されているレコードを列挙することができる（図3-12）。このように、VS.NetにはSQL Server Enterprise Managerの作業環境を取り入れたため、開発効率は大幅に向上することになった。

次に、このサーバ・エクスプローラの環境で、データビューを作成する様子を紹介する。データビューは、アプリケーションから見たデータ構造であり、SQL文を使って、データベースに格納されているデータを自ら望んだ形に再形成することができる。

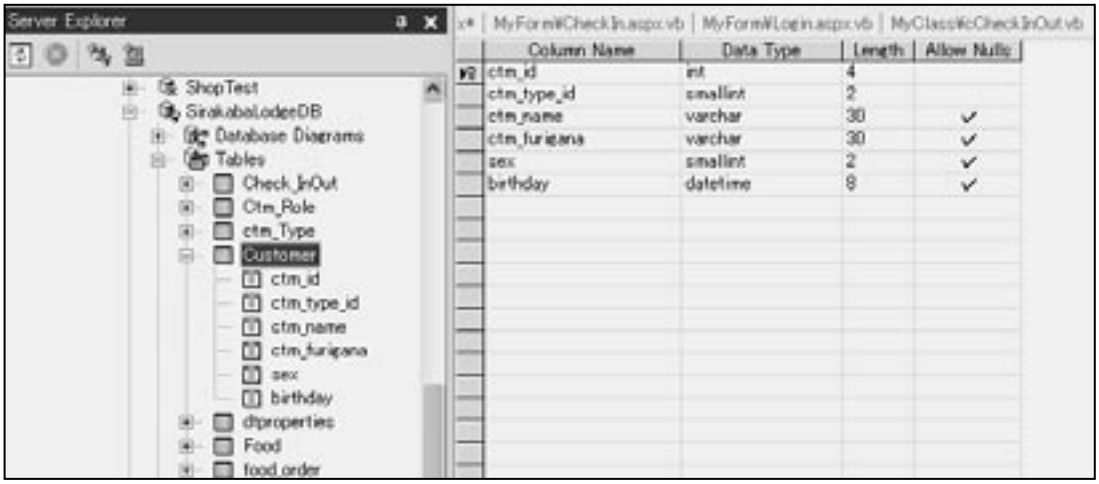


図 3-12 サーバ・エクスプローラから見たテーブルの構造

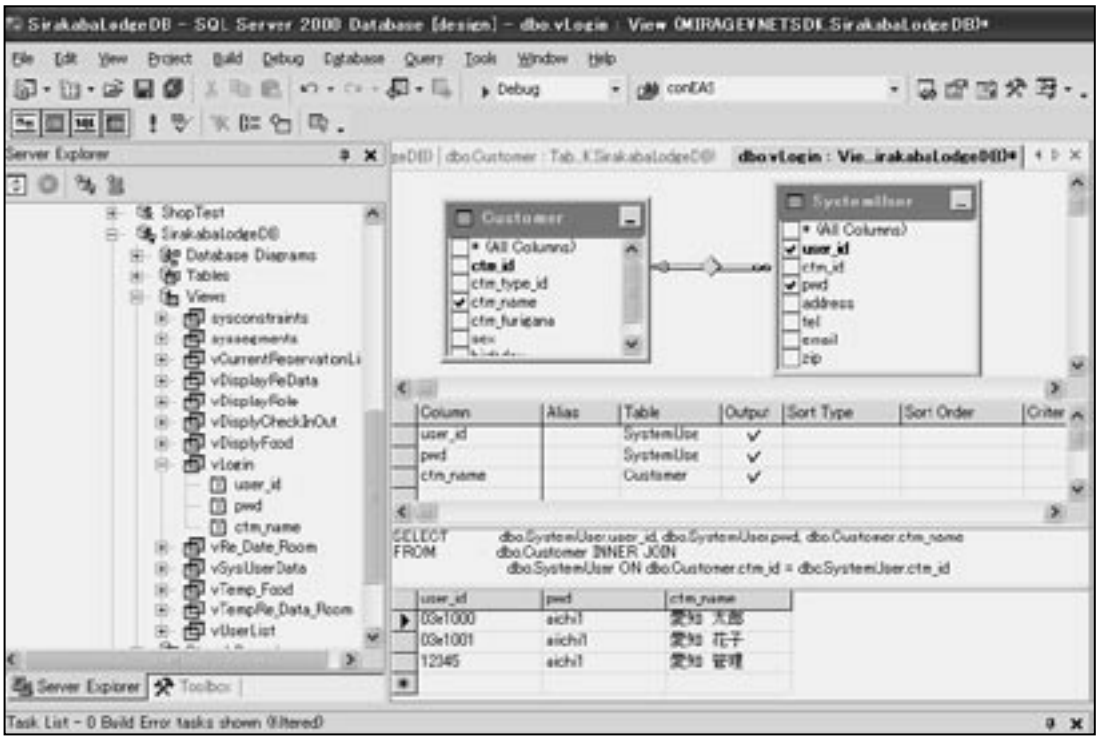


図 3-13　サーバ・エクスプローラの環境でデータビューを作成する

「Views」フォルダを右クリックし、「New View」を選択すると、データビューの新規画面に変わる。

図 3-13 には、ログインのため使われる「vLogin」の作成状況を表示している。ユーザーログインをするときに、ユーザーID、パスワードとユーザー名が必要である。それらの情報は Customer と System User 二つのテーブルから user_id、pwd、ctm_name の 3 つの Culomn を抽出し、データビューを作成する。作成作業は、ビジュアルな形で行われる。まず、関わったテーブルを挿入し、表示されたテーブルのなかで、抽出したい Column の左側にチェックを入れる。チェックを入れ終わったら、クエリの実行をすると、最下段に実行結果が表示される。これで、ビューに名前をつけて保存すればビューは完成する。

4. ADO.Net の導入

.NET アプリケーションからデータベースにアクセスするためには、「ADO.NET」と呼ばれる.NET Framework のアクセス機能を利用する。この ADO.NET の正体は、簡単にいってしまえば、.NET Framework のクラス・ライブラリに標準で備わっている一連のクラス群である。プログラムの形態が Windows アプリケーションであっても Web アプリケーションであっても、これらのクラスを利用して、データベースを扱うことができる。

4.1. .NET データ・プロバイダ

ADO.NET の実体は.NET Framework クラス・ライブラリに含まれる一連のクラスの集まり、それらのクラスはさらに次の 2 つに分類することができる。

- .NET データ・プロバイダを構成するクラス群
- DataSet クラスを中心とする非接続型のクラス群

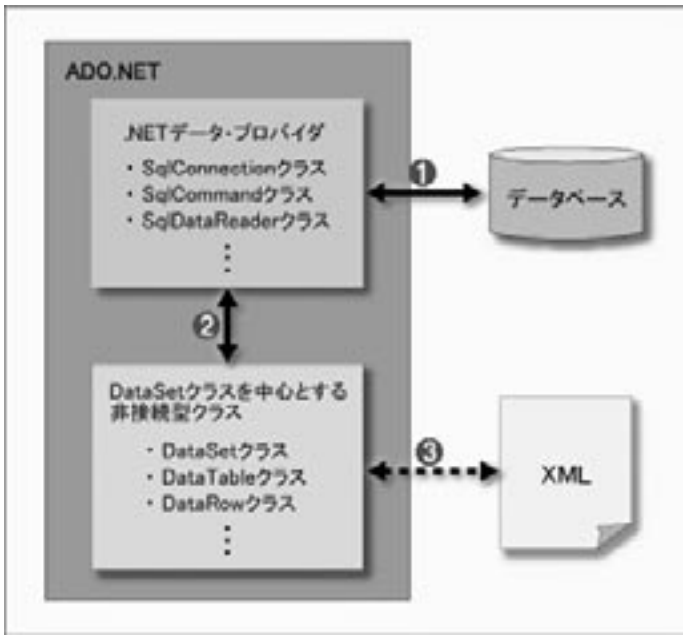
前者は、データベースに接続したり、データベースに対してコマンド（SQL 文）を実行したり、検索結果をデータベースから取得したりするために、データベース・アクセスに直接かかわる一連のクラスで構成される。これら一連のクラスは「.NET データ・プロバイダ」と呼ばれる。

クラス名	機能
SqlConnection	SQL Server との接続を表すクラス
SqlCommand	SQL Server に対して実行するコマンドを表すクラス
SqlDataReader	SQL Server からデータを読み取るクラス

表 4-1　データ・プロバイダ

表 4-1 で示した 3 つのクラスはすべて System.Data.SqlClient 名前空間に属するクラスで、.NET データ・プロバイダを構成しているクラスの一部である。

一方、.NET データ・プロバイダ関連のクラスとは別に、System.Data 名前空間に属している DataSet クラスを中心とする一連の「非接続型クラス」が ADO.NET には用意されている。これらのクラスはデータベースから読み取ったデータをメモリ上に表形式で保持するための新しい仕組みを提供している。非接続型クラスに対して、明示的なデータベースとの接続が前提となる.NET データ・プロバイダのクラスは、「接続型クラス」と呼ばれることもある。ADO.NET を構成する 2 つのクラスの集まりを図にすると次のようになる。



出典：ADO.NET 基礎講座 http://www.atmarkit.co.jp

図 4-1　ADO.Net の構成

この図のポイントをまとめると次のようになる。

- ADO.NET を構成するクラスは、.NET データ・プロバイダを構成するクラスと、DataSet クラスを中心とする非接続型クラスの 2 つに大別できる。
- .NET データ・プロバイダを構成するクラスが、データベースと接続し、データのやりとりを行う（①）。
- 非接続型クラスでは、.NET データ・プロバイダを経由してデータベースにアクセスできる（②）。
- 非接続型クラスは、データベースと同様に、XML 文書のデータにもアクセスできる（③）。

次は.NET データ・プロバイダのクラスの例として、予約システムで使用する System.Data.SqlClient 名前空間の次の 3 つのクラスを挙げて説明する。

- SqlConnection クラス
- SqlCommand クラス
- SqlDataReader クラス

ここで「Sql」で始まる名前のクラスは、すべて SQL Server にアクセスするためのクラスである。.NET データ・プロバイダを構成するクラスの中でも、これらの SQL Server 専用のクラスには、「.NET Framework Data Provider for SQL Server」という名前が付けられている。

前例の愛知太郎君のログイン（図 2-1）に対し、ADO.NET を使ってコードで記述してみよう。
プログラムからデータベースにアクセスする場合、まずデータベースに接続する（データベースをオープンする）という作業が必要になる。ADO.NET でデータベースに接続するには、SqlConnection クラスを利用する。接続ためのパラメータとして、

- 接続先となるデータベース・サーバー： server=MIRAGE¥NETSDK;
- 使用するデータベース名：database=SirakabaLodgeDB;
- SQL サーバーのユーザーID： user id=jiang;
- SQL データベースへのアクセス パスワード：pwd=aichi1

を設定する必要がある。SqlConnection クラスによりデータベースに接続するためには、これらと同じ内容の情報を「接続文字列」として、セミコロン区切りの 1 つの文字列で指定しなければならない。このための文字列は表 4-2 の（5）行のようなものになる。

この接続文字列を SqlConnection クラスのコンストラクタにパラメータとして指定し、（6）行のように SqlConnection オブジェクトを作成する。そして Open メソッドを呼び出して、データベースとの接続を行う。データベースにアクセスし終えたら Close メソッドを呼び出して、接続を閉じなければならない。データベースへの接続数は限られているので、可能な限り早く閉じるべきである。
次に、データベースへ発行する select 文を、SqlCommand クラスにより用意する。次の例では、ログインで使われる基本コードであり、（7）行で select 文を記述した文字列 sql と、（6）行上で作成した SqlConnection オブジェクトの 2 つのパラメータをとる SqlCommand クラスのコンストラクタを使用して、（8）行のように SqlCommand オブジェクトを作成している。SqlCommand クラスのコンストラクタの第 2 パラメータとして指定した SqlConnection オブジェクト（変数 conSirakaba）による接続を通じて、select 文がデータベースへ送信されることになる。

```
(1) Dim strCon As String
(2) Dim  conSirakaba As  SqlConnection
(3) Dim sql As String
(4) Dim cmdSql As SqlCommand

(5) strCon = "server=MIRAGE¥NETSDK; database=SirakabaLodgeDB;user id=jiang; pwd=aichi1"
(6) conSirakaba = New SqlConnection(strCon)
(7) sql = " select ctm_name from [dbo].vLogin where user_id='03e1000' and pwd='aichi1' "
(8) cmdSql = New SqlCommand(sql, conSirakaba)
(9) conSirakaba.Open()
(10) Dim dr As SqlDataReader = cmdSql.ExecuteReader()
(11) conSirakaba.Close()
```

表 4-2　ログインのための基本コード：ADO.Net によるデータベースのアクセス

ただしこの時点では、まだ送信はされない。select 文と SqlConnection オブジェクトを装備した SqlCommand オブジェクトを作成しただけである。そのため、このコードを記述する位置は SqlConnection クラスの Open メソッドを呼び出す前でも後でも問題はない。
続いて、SqlCommand オブジェクトに対して ExecuteScalar メソッドを実行する。これにより select 文が送信され、データベース・サーバから返される処理結果が SqlDataReader クラスのオブジェクトを通して返される（10 行）。　次の図は、ここまでに登場した 3 つのオブジェクトの関連をまとめたものだ。

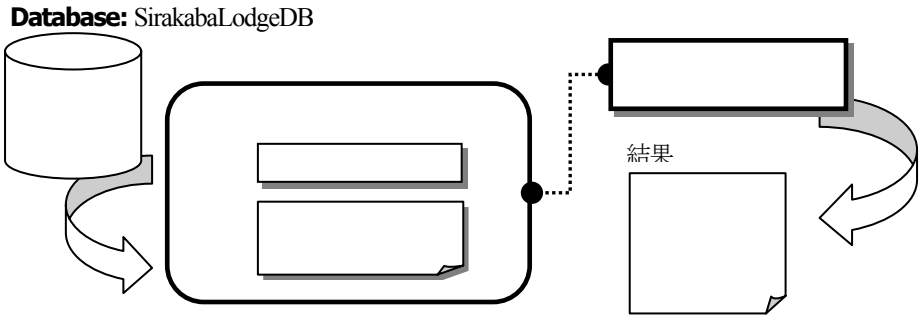


図 4-2　3 つのオブジェクトの関連

4.2. オブジェクト指向の実装へ

ウェブアプリケーションの 3 層システムアーキテクチャーを持つことによって（図 1-1）、アプリケーションの Presentation 層、Business Logical 層と Data Manage 層が互いに独立することができるようになる。この独立こそ、オブジェクト指向の開発法によってもたらすシステムの安定性、保守性、拡張性の効果を発揮することができる。

ADO.Net の導入に連れ、実装の際に Business Logical 層と Data Manage 層の分離は容易になったことが明らかになった。次に、同じ Login の事例を通して、ADO.Net クラスが一箇所に集中した、完全なオブジェクト指向の実装方法を示す。

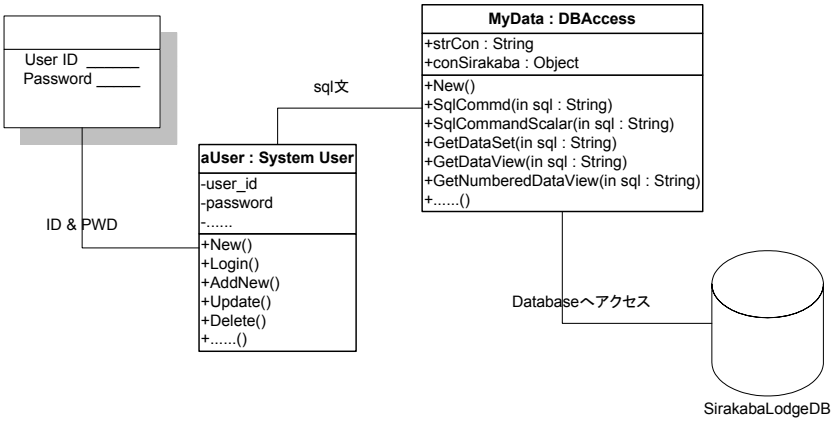


図 4-3　オブジェクト実装の事例

ログインの場合、まず、ユーザーはウェブフォーム上で、ID とパスワードを入力する。その後、フォームクラスは、System User クラスのオブジェクト aUser を宣言し、入力された ID と PWD を aUser のカプセルに入れ、このカプセルを次の Business Logical 層に渡す（表 4-3）。

実は、Business Logical は既に aUser カプセルの内部、つまり、aUser.Login()関数に含まれている。ログイン処理する場合、aUser オブジェクトの user_id と password（これらはウェブフォーム上で入力された情報と一致する）によって、SQL クエリを発行し、DBAccess クラスへ渡す（表 4-4）。

DBAccess クラスには、ADO.Net のクラスによって、作られた関数が集まっている。DBAccess クラスからは Mydata オブジェクトを宣言し、aUser オブジェクトから受け取った SQL 文を、内部関数 SqlCommandScalar(sql)に渡し、そこから、データベース SirakabaLodgeDB へアクセスする（表 4-4）。

図 4-3 には、このように Business Logical と Data Manage の分離に基づいた実装法のイメージを表している。

```
Dim aUser As New cSysUser
Dim Result As String

  aUser.user_id = txtID.Text.ToString
  aUser.pwd = txtPWD.Text.ToString
  Result = aUser.login
```

表 4-3　ログインのコード

```
Public Function login() As String
   Dim Mydata As New cDBAccess
   Dim sql As String
   Dim value As New ArrayList

   sql = " select ctm_name from [dbo].vLogin where user_id='{0}' and pwd='{1}' "
   value.Add(user_id_)
   value.Add(pwd_)

   sql = String.Format(sql.ToString, value.ToArray)
   Return Mydata.SqlCommandScalar(sql)
End Function
```

表 4-4　System User クラスの Login 関数

```
    Public Function SqlCommandScalar(ByVal sql As String) As String
        Dim cmdSql As SqlCommand

        cmdSql = New SqlCommand(sql, conSirakaba)
        conSirakaba.Open()
        Return cmdSql.ExecuteScalar()
        conSirakaba.Close()
    End Function
```

表 4-5　DBAccess クラスの SqlCoomandScalar()関数

このように、アプリケーションの Presentation 層、Business Logical と Data Manage、3 階層にわたる独立な構造を実現した。オブジェクト指向プログラミングの詳細については、次回に論じる。

4.3. SQL の Stored Procedures の使い方

Stored Procedures は Transact-SQL ステートメントの集まりをカプセル化した、データベースのオブジェクトである。前に述べたデータベースクエリの処理は、SQL サーバーの外で行われるのに対し、Stored Procedures は SQL サーバーのオブジェクトとして、サーバーに保存し、必要に応じて繰り返して使用できる。Stored Procedures を利用すると、システムの Performance を大幅に上昇させることができる。その理由として、次の 3 点が挙げられる。

a) 一つの Stored Procedures において、複数のクエリを集めることができる。以前、１クエリ処理ごとに１回のデータベース・アクセスが必要だったのに対し、いまは、複数のクエリ処理を１回のデータベース・アクセスで済ませるので、その分のデータベースの使用資源が解放される。

b) Stored Procedures は必要に応じて、繰り返し利用できる。更に、同様の処理を２回以上実行する場合、自動的にキャッシュの上で行われるので、効率が大幅に上がる。

c) Stored Procedures は SQL サーバーの内部に置かれるので、頻繁にネットワークを経由したクエリのやり取りが省ける。

次には、前の Login 事例をもう一回利用し、Stored Procedures と ADO.Net の連携について解説する。まず、VS.Net の Server Explorer を開き、予約システムのデータベース SirakabaLodgeDB フォルダを展開すると、Tables や Views の下に、Stored Procedures フォルダが表示される。開いて見ると、データベースシステム用の Stored Procedures はリストされていて、名前の前に「dt_」が付けられている。

図 4-4　Server Explorer から見た Stored Procedures の一覧

その中に、up_Login という stored Procedure が追加した。名前の前に付けている「up_」は User Procedure

との意味である。Procedure コードの内容は、下表で示す。Stored Procedure Programming の詳細について、文献の「3」を参考。

```
ALTER PROCEDURE dbo.up_login
        (
                @user_id varchar(10),
                @pwd varchar(30),
                @ctm_name varchar(30) OUTPUT
        )
AS
        select @ctm_name=ctm_name from [vLogin] where user_id = @user_id and pwd=@pwd
        RETURN
```

表 4-6　Login の Stored Procedure コード

次に、このProcedureを呼び出すためのADO.Netのプログラミングコードについて、見てみる(表4-6)。SQL サーバーに保存された Stored Procedure と連携する場合、SqlConnection、SqlCommand、SqlParameter の三つの ADO.Net のクラスが使われた。まず、Stored Procedure の名前を SPName、データベースの接続は conSirakaba で設定し、SqlCommand クラスのオブジェクト cmdSql を作り出し、コマンドタイプはStoredProcedure とする。

```
cmdSql = New SqlCommand(SPName, conSirakaba)
cmdSql.CommandType = CommandType.StoredProcedure
```

次には、引数user_idとpwd、返す値ctm_nameのために、それぞれにSqlParameterのオブジェクトReplyを宣言し、引数の場合ParameterDirectionはInputに設定し、返す値の場合、ParameterDirectionはOutputに設定する。その後、それぞれのReply.Value に対応する変数を与える。
最後に、以下のようにデータベースをOpenし、処理作業を行うと結果を返し、データベースをCloseする。

```
onSirakaba.Open()
cmdSql.ExecuteNonQuery()
Return cmdSql.Parameters("@ctm_name").Value
conSirakaba.Close()
```

表4-７には、まとまったコードを示している。

```
Public Function UP_Login(ByVal UserID As String, ByVal PWD As String,
                                                    ByVal SPName As String) As String
        Dim cmdSql As SqlCommand
        Dim Reply As SqlParameter

        cmdSql = New SqlCommand(SPName, conSirakaba)



        Reply = cmdSql.Parameters.Add(New SqlParameter("ReturnValue", SqlDbType.VarChar))
        Reply.Direction = ParameterDirection.ReturnValue

        Reply = cmdSql.Parameters.Add(New SqlParameter("@user_id", SqlDbType.VarChar, 10))
        Reply.Direction = ParameterDirection.Input
        Reply.Value = UserID

        Reply = cmdSql.Parameters.Add(New SqlParameter("@pwd", SqlDbType.VarChar, 30))
        Reply.Direction = ParameterDirection.Input
        Reply.Value = PWD

        Reply = cmdSql.Parameters.Add(New SqlParameter("@ctm_name", SqlDbType.VarChar,
30))
        Reply.Direction = ParameterDirection.Output


        conSirakaba.Open()
        cmdSql.ExecuteNonQuery()
        Return cmdSql.Parameters("@ctm_name").Value
        conSirakaba.Close()
    End Function
```

表 4-7　Stored Procedure を呼び出すためのコード

本文は、３階層の中の、データ構造の設計、ADO.NET を用いたデータベース・プログラミング、Stored Procedure の使い方を中心にして、論じた。

次回は、オブジェクト指向によるシステム開発について論じる。

５．【参考文献】

[1] Brenda Kienan “managing your e-commerce business second edition”, Microsoft Press,2001

[2] Dejan Sunderic, Tom Woodhead, “SQL Server 2000, Stored Procedure Programming”, Osborne/McGraw-Hill, 2001.

[3] Rebecca M.Riordan 著、（株）日本ユニテック訳、「SQL Server2000 プログラミング実践講座（下）」、日経ＢＰソフトプレス社、2001.

[4] Craig Bowes, Daniel Cazzulino, Mike Clark, Chris Hart, Neil Raybould, Tobin Titus, “Beginning Web Programming using VB.Net & Visual Studio.Net”, Worx Press Ltd., 2002.

[5] Dino Esposito, “ASP.NET and ADO.NET”, Microsoft Press, 2003.

[6] 小松順子 著,Visual Basic .NET で学ぶUML,日経ＢＰソフトプレス,2003.

[7] テクノロジックアート著,基礎UML,インプレス,2003.

[8] 横井与次郎著,VB.NET オブジェクト指向プログラミング入門,メディア・テック出版,2002.

[9] Rebecca M. Riordan 著, (株) クイープ訳,ADO.NET 実践講座,日経ＢＰソフトプレス,2002.

[10] ADO.NET 基礎講座 http://www.atmarkit.co.jp/fdotnet/basics/adonet05/adonet05_01.html