研究叢書 52

RIEBデータベースの研究

安田 豊　阿部茂行 著

神戸大学
経済経営研究所
1998

研究叢書 52

RIEB データベースの研究

安田 豊　阿部茂行 著

神戸大学
経済経営研究所
1998

RIEB データベースの研究

安田　豊　　阿部茂行　著

神戸大学経済経営研究所

1998

はしがき

著者の一人が学部学生であった頃，コンピュータに触れたくて，プログラム解析が出来る「機械計算論」という科目をとった。FORTRAN のプログラム実習はあったものの，コンピュータにそのコードを入力するのではなく，講師がそのプログラムを読んで朱を入れる人間コンピュータ機械計算であったのにがっかりした経験がある。1970年にアメリカに初めて大学院生として行ったとき，大型汎用機である IBM 360 がふんだんに使える環境にびっくりしたものだ。以来，コンピュータの世界の進歩は著しく，今ではその頃の大型汎用機を能力ではるかに凌駕するパソコンを研究者が一台といわず複数台持つ状況になってきている。コンピュータ利用の典型は，ワークステーションをコアにして，パソコンをネットワークに繋ぎ，毎日電子メールを交換し，WWW の News Group，*The Wall Street Journal Interactive Edition* などを購読し，情報を収集するというものであろう。

1994年に阿部が経済経営研究所機械計算室主任になった。そして 1995年に安田が助手として採用され機械計算室に配属された。ほどなく機種更新となり，それまでのクローズな大型汎用機オンリーから，オープンなワークステーションと大型汎用機併用の環境に移った。神戸大学自体のネットワークKHAN も増強され，まさにオープンネットワークの時代に突入した。2000年に機種が新たに更新されるが，それは疑いもなくワークステーション中心のものとなろう。このときデータベースシステムも大型汎用機からワークステーションに完全に移行することになる。現在は大型汎用機に載った従来のものと，ワークステーションに新たに構築したデータベースと 2 通り利用できるようにしている。ワークステーションに載ったデータベースは，先に述べたネットワークでの WWW 利用を前提にしたシステムで，非常に使いやすいものに出来あがったと自負している。現研究所のコンピュータ環境は，次期機種までの繋ぎで，デー

タベースのワークステーションでの構築は，我々に与えられた課題であった。3年をまたずに，つまり，安田が経済経営研究所に来て3年内に，そして阿部が京都大学に転出するまでに，大型汎用機で現在利用できている以上のものを作り上げたのである。この著はまさに我々が公約を果たした証しでもある。

第Ⅰ部では経済経営研究所のこれまでのデータベースについて振り返り，新たなデータベース構築のデザイン，データの標準化に関する細かな問題点，リレーショナル・データベースのエンジンとして採用したOracleのパフォーマンスチェック，WebDBのデザインを紹介している。第Ⅰ部の終わりの章では，データベース利用の将来を考えている。第Ⅱ部はユーザーの便宜を考えて，マニュアルとしても使えるように多国籍企業データベースなどの各データベース個別の議論を行った。そして第Ⅲ部は資料として活用できるよう，実際のプログラムと関連する技術情報を付けている。

この書物は安田・阿部の共著となっているが，データベース構築には多くの人のお世話になった。吉原英樹，石垣健一歴代所長や井川一宏現所長は，コンピュータやデータベースの重要性を理解し，財政面での協力を惜しまれなかった。実際の業務である，データの入力，チェック，維持等々については機械計算室のメンバーである吉田（井口）美香，篠原久姫子，羽路良子さん（平成10年退職）が，積極的に業務以上の協力を惜しまなかった。心からお礼申し上げる。また，いちいち名前は挙げないが，試験的にRIEBデータベースを所内で公開した際，それにコメントを寄せられた研究所の面々に感謝したい。また，本叢書には（財）国際東アジア研究センターの「東アジア経済データベース」研究プロジェクトの成果も一部含まれている。シンガポールに出張し，ISEASのデータベース状況を視察できたことは，データベースの方向性を考える上で，大いに参考になった。市村真一所長に感謝したい。

最後に，この著書の役割分担について述べておくと，阿部がはしがき，第1章，第6章を執筆，その他のほとんどは安田がドラフトを書き，阿部がそれを

編集した。全体的な枠組みについては2人で何度も話し合った。また，第1章のSECRETARYに関するところは，機械計算室でその維持・操作を担当している吉田美香さんにドラフトを書いてもらった。第Ⅱ部の例題については篠原久姫子・吉田美香さんに実際に検索してもらいその検索例を書いてもらった。また草稿にも目をとおしてもらい，そのおかげで読みやすさは大いに改善された。この叢書，そしてRIEBデータベースのプログラミングの大部分は安田の手になるもので，阿部は監修者の役割を果たしたにすぎないといえるが，見方を変えれば，機械計算室メンバーすべての汗の結晶がこの叢書に結実したということも事実である。安田，吉田，篠原，羽路と阿部の経済経営研究所での大切な共同作業の記念碑となっているのである。

秋の気配の迫る六甲台にて

阿　部　茂　行

目　次

第Ⅰ部

RIEB 経済経営データベースの開発

第1章 RIEBデータベース前史

1.1 はじめに

経済データに関しては，インターネットの普及で，ユーザーが直接データ提供機関であるアジア開発銀行などのホームページをアクセスし簡単に手に入るようになった。*KEY INDICATORS of Developing Asian and Pacific Countries*は現在Excel[(1)]ファイルでダウンロードできる。多くの機関は，個人ユーザー向けにCD-ROMを販売しはじめた。世界銀行の*World Development Indicators*(*WDI*)などが広く利用されている。10年前ぐらいまでは，まだ計算機といえば，大型汎用機が主流で，データを供給する機関(たとえば，IMFや日本経済新聞社など)はMT(Magnetic Tape，オープン・リール・テープ)でデータを提供していた。個人でMTを読む装置を持つものはほとんどなく，システムエンジニアを擁する研究機関でのみデータベースを作成し，利用することが可能であった。

世界銀行の*WDI*，IMFの*International Financial Statistics*(*IFS*)，OECDの*Main Economic Indicators*などは早くからCD-ROMで提供されてきており，汎用機がなくともパソコンでデータアクセスが可能であった。*WDI*の最近のものはWin*STARS 4.0という強力な検索ソフトを備え，非常に使いやすくなってきた。しかし異なるデータベースごとに，提供機関が独自の検索ソフトを付けて，それでしかデータにアクセスできないという状況は憂慮に値する。早い機会に統一されたデータ保存形式，検索ソフトの標準化が望まれる。そうでないかぎり，CD-ROMデータの数だけの検索ソフトをマスターしなければ，データにアクセスできないということになる。これが実現するのはまだ先の話

(1) Microsoft Excel，Microsoft社が開発した表計算ソフト。

であり，永遠に実現しない可能性もある。それだけではなく，現在ある検索ソフト，また今後出てくるであろうCD-ROM検索ソフトは，ユーザーの思い通りのソフトになっていないのではなかろうか。データがCD-ROM一枚に収まりきらない場合を想定すれば，容易にユーザーが満足できない事情が分かろう。膨大なデータを満足のいくスピードで手間なく検索するためには，まずデータを大容量で高速のハードディスクに置くべきで，検索・抽出の仕方もより高速・簡単でなければならない。

こうした新しい時代の変化に応じて，一研究機関でデータベースをどのように蓄積し，どのようなサービスを提供するかが，火急の課題となっている。例えば，シンガポールのInstitute for Southeast Asian Studiesでは香港のデータベース会社からリース契約でデータ提供を受けている。所内のワークステーションに毎月送付されたデータ全てを移行し，所員はイントラネットで利用している。これで全て用が足りているわけではなく，所員は個別に世界銀行などのCD-ROMを購入することもある。メインのデータベースとして，独自に開発するのをあきらめ，このように適当なものを完全にアウトソースするのも一つの道である。オーストラリア国立大学の豪日研究センターでは我々と同様多くのデータを購入し，加工してきている。SASをそのデータベースエンジンとして使用し，データを内外に提供している。外部の者には結構高額で提供している。一研究機関でデータベースを維持するにはコストがかかる。それを所内あるいは大学内だけで使うのではコストパフォーマンスが悪い。著作権等クリアしなければならない問題も多々あろうが，データベース維持機関にとっては，豪日研究センターのように外部に販売することで，生き残りを図るというのも今後の道の一つであろう。

幸い我々は大規模なデータベースを維持していく予算と人員と機会に恵まれた。海外の研究機関のデータベース事情を参考にしつつ，研究機関内でのデータベース利用の方法について新機軸を打ち出す（ソフトウェアの開発）ととも

に，神戸大学経済経営研究所がこれまで蓄積，維持してきたデータベースをワークステーションに新しく構築することとした。

我々は，なるべく簡単で統一的なデータの取り扱いをなすべきと考え，現状況で最も理想的なデータベースシステムを作り上げた。

本著は，こうしたソフトの開発の全貌を，利用マニュアルを含めた形で示すものである。

1.2　データベースの変遷：大型汎用機からワークステーションへ

コンピュータの世界の進歩は著しく，研究者が昔の大型汎用機をこえる能力をもつパソコンを，一台といわず複数台持つような状況になってきている。コンピュータの利用法も様変わりし，その典型的な使い方は，ワークステーションをコアにして，パソコンをネットワークに繋ぎ，毎日電子メールを交換し，WWWのNews Groupに参加したり，*The Wall Street Journal Interactive Edition*などを購読。政府関係のサイトからデータをダウンロードし，そのデータをExcelを使って，見やすい表にし，統計パッケージであるEviewsを使って回帰分析をする。NBERのサイトからWorking PaperをPDF[(2)]形式でダウンロードし，それを印刷するというものであろう。とにかく便利になった。それゆえ，コンピュータ・ユーザーの底辺は広がり，インターネットの利用者の数が爆発的に急増したのである。

このことは，これまでのデータベースのあり方との関わりで，プログラミングや，ネットワーク環境のあり方が大いに変貌せざるをえないことを意味する。

次に，我々の置かれている現在の環境をよりよく理解するために，神戸大学経済経営研究所のコンピュータ環境とデータベースシステムのデザインの変遷

(2)　Portable Document Format, Adobe社が作った電子出版のための文書フォーマット。

をまず以下で論じてみよう。

1.2.1 研究所のコンピュータシステムとデータベースシステムの変遷

まず，次の表を参考に，研究所に導入されたコンピュータとデータベースの歴史を簡単に振り返っておこう。経済経営研究所のコンピュータは1970年のミニコンに始まり，1974年の中型機，そして1978年以降大型汎用機へと更新されつづけてきた。

1970年—HITAC10（32KB/nodisk）

ミニコン，データベース的なものはない。能力的に作成不可能。

1974年—HITAC8350（256KB/120MB）

これは中型計算機で，汎用機アーキテクチャではない。

全国の大学に先駆けてデータバンクの構築に着手した。これがBEICAプロジェクトで，1978年に一応の完成をみたとある。データ規模は国民経済関係6000系列，国際金融関係*IFS*150ヶ国150,000系列。分析応用プログラムパッケージとしてSTEPS, CROTAB, SIMPLなどが作成された。HITAC10上で分散して分析処理を行うATLASも作成された。

1976年—HITAC M-150（1MB/600MB）

汎用機をはじめて導入。

SECRETARYを2年がかりで完成，BEICAバンクの全てを引き継ぎ，新SNA 10,000時系列，興銀財務1700社400項目を追加。

1983年—HITAC M240-D（8MB/3.8GB）

データベースは国際経済経営データベース。1983年に文部省科学研究費試験研究として片野彦二教授を中心に多国籍企業データベースの研究に着手。1986年からは文献センターの経常業務として開発。

1987年—HITAC M260-D（24MB/12GB）

データベースは国際経済経営データベースと多国籍企業データベース。

1992年—HITAC M640/35E（64MB/25GB）

データベースは国際経済経営データベースと多国籍企業データベース。

1996年—HITAC M640/45E（96MB/40GB）とHP9000 V715/100（256MB/21GB）等

汎用機とワークステーション群のシステム構成に。

データベースは国際経済経営データベースとRIEBデータベース。多国籍企業データベースはRIEBデータベースの一部となる。

このように，経済経営研究所のコンピュータ環境は時代とともに大きく変わってきた。それに応じて，データベースの形態も大きく変わってきた。次に，データベースのこれまでの経緯を簡単に見ておこう。

1.2.2 BEICAシステム

経済経営研究所のデータベース研究は早くから開始されている。その最初の成果が経営・経済情報制御分析システム（Business & Economics Information Control and Analysis System, BEICAシステム）である。その主導的役割を果たされた米花稔神戸大学名誉教授の名前とかけているところが，最初開発された方々のウィットに富むところである。我々は，ネーミングに関しては非常に単純にRIEBデータベースとしたのみである。

さて，このBEICAシステムであるが，その開発は1974年に導入されたHITAC8350（主記憶256KB/補助記憶120MB）上で行われた。これより前の研究所システムは1970年に導入されたHITAC10（主記憶32KB/補助記憶なし）であり，データベース的なものを開発することは能力的に不可能であった。まさに全国に先駆けた開発であったと言えよう。

BEICAシステムの中枢は経営・経済データバンクとその制御（BEIC）にあ

り，その周囲に各種の分析用アプリケーションプログラムがあってBEICより情報を受けている。分析用アプリケーションにSTEPS(Simplified Techniques for Economic Planning and Simulation)などがあって，一群の完結したプログラムとなっている。データベースとしては，国民経済データとして約4000系列(国民所得統計などから入力)，企業財務データが約500社×100系列(営業報告書，有価証券報告書などから入力)，そして国際データとして，200ヶ国×150系列(IMFの*IFS, DOTS, BOPSY*磁気テープ)がある。基本的には，このデータは

QUERY BEICA KEY(1)=1234

といった簡易コマンドで，検索・抽出することができる。ここで，BEICAがデータベース名，1234がデータコードである。基本的にユーザーはデータコード表を手元において，検索作業を行う必要があった。このシステムでは，更に，季節調整，統合，デフレート，コード変換，初等統計，階差，成長率，指数などの簡単な計算がそのシステムの中で可能であった。(参考文献［米花］)

1.2.3 SECRETARY

1978年末にはIBM互換アーキテクチャの汎用機であるHITAC M-150(主記憶1MB/補助記憶600MB)が導入され，その上でBEICAバンクの全てを引き継ぐSECRETARYの開発が始まった。(参考文献［民野1986］)

SECRETARYとはThe Software Equipment for Creation, Retrieval, Editing, Translating and Analysis through Remote Display Terminalsのことである。民野庄造氏(現姫路獨協大学助教授)が参考にしたのは杉浦一平和歌山大学名誉教授のASTRO-FOILならびにSTEPSで，これをもとに拡張したものという。SECRETARYは，主にPL/I言語[3]とJCL[4]で開発された，

(3) IBM系の汎用機のために開発された，汎用プログラミング言語。
(4) Job Control Languageジョブ制御言語。IBM系の汎用機においてバッチ処理を行なう場合に利用する言語。

汎用機上で作動するデータベースシステムである。SECRETARYの特徴として，様々なフォーマット（仕様）のデータを同一のインタフェイスで提供していること，それに加えて，ユーザーは検索ソフトウェアがどんなハードウェアやOS（オペレーティングシステム）の上で動いているかを意識しないですむということがあげられる。SECRETARYは汎用機の上で動いているシステムである。普通，ユーザーはパソコンやUnixでは意識しない汎用機特有のいくらかの約束事を知っていなければならない。例えば，ファイルのアロケーション（領域割り当て）やファイルを転送する場合のファイル編成を無視して，汎用機で作業を行なうことはできない。SECRETARYはユーザーに代わってこれらを行なってくれるので，ユーザーは汎用機のハード構成やOSを意識しないで作業を進めることができるのである。

SECRETARYは17年間使われ続けてきた。この間そのメンテナンスにはひとりのSEがフルにあたっていた。メンテナンスの内容は，磁気テープ等で受け入れたデータをVSAM[5]ファイル形式のデータベースに更新することや，その資料等を整理することである。データに関していえば，毎年更新するもののみならず，アドホックに研究所スタッフの依頼に応じて，新規にデータベースに付け加えたものも多くある。パソコンがポピュラーでなかった時代には，ユーザーがSECRETARYに期待する機能として，様々なデータベースから簡単に必要なデータを検索し，抽出すること，また，これらのデータを分析・解析することがあった。SECRETARYはその意味でAll-in-Oneのシステムであり，以下の特徴を備えている。（参考文献［民野1989］）

・対話形式を基本とし，一括処理（DO文，IF文等）も可能。

・時系列分析のほか，クロスセクション分析（行列処理）も行える。

・当研究所収録の全統計データベースをアクセスできる。

・ユーザー自身のコマンド・ライブラリー，データファイルを持てる。

(5)　インデックス付きファイル形式のひとつで，追加や更新に強い。

・コマンド及びデータの操作は，すべて画面編集方式で行える。

ただし，データ分析・解析機能については，この数年はほとんど使われていない。その理由は，ユーザーが選択できるハードウェアやソフトウェアの範囲が広がってきたことにある。

SECRETARYの操作は，フリーフォーマットの入力域にSECRETARYが用意しているコマンドを記述するコマンド形式をとっている。SECRETARYが開発された時期には，コマンド形式しか存在しなかったからである。また，SECRETARYは，コマンドを組み合わせて自由に記述でき，一度実行したコマンド文や検索結果をユーザー固有のコマンドやファイルとして登録し，繰り返し使用，また加工することができる。当時のユーザーにとっては使いやすいシステムであった。しかし，最近のグラフィカル・ユーザー・インタフェイス(GUI)を採用したメニュー形式のアプリケーションに慣れたユーザーにとってSECRETARYは直感的でなく操作がやや煩雑かもしれない。現在，ユーザーがSECRETARYから離れつつある理由の一つである。

1990年代に入ると，研究所のシステムは汎用機が依然メインであったが，これまでの専用端末に代わって，PCを端末として使うようになった。端末機能を持つソフトを導入し，PCとしても端末としても使うようにしたのである。このため，研究所でのデータ処理は，徐々に汎用機上で処理されるより，手元のパソコン上で行なわれるようになった。汎用機に蓄積されたデータをSECRETARYで抽出し，それをパソコンに転送し，その後，統計処理をパソコン側でするのが一般的な使い方であった。これが好まれた理由は，この頃になると，汎用機に搭載されていたような強力な統計ソフトが，装いを新たにパソコン用ソフトとして登場してきたことにもよる。例えば，SAS，TSP，SPSSなどはほとんど汎用機と同じようにパソコンで使えるようになった。汎用機TSPの改良版の，GUIを使ったユーザーフレンドリーなMicro TSP(現在はWindows95対応のEviewsとなっている)，Mathematicaなどが登場してき

て，汎用機で利用可能なソフトより，多機能で豊富なソフトがパソコン用に利用可能となってきたのである。簡単な記述統計を計算するなら表計算ソフトが便利であるし，ワープロはパソコン上のものを使うようになってきていたから，MathematicaやMicro TSPなどの直接計算結果を含めて，カット&ペーストでワープロに張り付けられるようになったメリットを十分利用するようになったのである。

こうした利用のためSecto123というインタフェイス・ソフトを自製し，SECRETARYの抽出データを一般的な表計算ソフトで読み込める，CSV形式[6]のデータに変換できるようにした。これをパソコンに転送するソフトと組み合わせて，ユーザーは分散処理を進めた訳である。

1998年現在でのSECRETARYの持つデータベースのスケールは，表1が示すように約404万件である。この中には，メンテナンスを中止したデータも含まれている。

もう一つ別の流れのデータベースとして，1983年から始まった多国籍企業データベースがある。多国籍企業関係資料に対する需要動向調査により，どのようなデータが研究者に必要とされているかをチェックし，調査項目を特定化した上で，1984年からデータの収集，データベースシステムの開発をスタートさせた。片野彦二氏（現名古屋学院大学教授）と定道宏氏（現京都大学教授）のリーダーシップのもと安田聖氏（現一橋大学助教授）がシステム開発を担当し，約2万系列ほどのデータベースを汎用機M260D上のシンコムシステム社のTIS上に構築した（参考文献［安田］）。安田聖氏が神戸大学を去った後，小幡範夫氏（現立命館大学助教授）がこのあとを担当した。基本的に東洋経済新報社から提供されるMTを使ってデータのアップデートを図り，データベースシステムそのものは踏襲した。このシステムは特定のマシンとソフトウェアに依存

(6) Comma Separated Value，データのファイル保存形式の一つ。データをカンマと改行で区切る単純さから，異なるソフト間のデータ交換によく使われる。

表1：SECRETARY データベースのサイズ

データベース名	データ件数	データベース名	データ件数
OECD 貿易	298,868	CITIBASE（期間が異なる）	6,160
OECD マクロ	6,750	COMPUSTAT	191,625
IMF	266,186	豪州経済統計	4,350
興銀	2,358,914	米国経済統計（米国マクロ）	1,567
日経総合経済	24,276	新 SNA	6,499
日経総合経済（期間が異なる）	24,276	世界銀行	48,999
日経卸売物価・輸出入物価	8,754	世界銀行（負債）	7,317
日経金融財務	48,336	EXSTAT（欧州財務）	387,090
日経財務・連結	208,772	アメリカ資金循環データ	619
日経地域経済	136,878		
CITIBASE（米国経済データ）	6,160	合計	4,042,396

したシステムであったため，小幡氏が去ったあと，そのままの形で継続することが困難であった。その後，多国籍企業データベースの継続を依頼された我々は，この反省から，ポータビリティに富むデータベースシステムを採用するにいたったわけである。

1.3 新時代の幕開け

1996年に研究所はコンピュータに関して方向転換をした。ワークステーションを導入したのである。データベースが汎用機に置かれている関係で，これまでの汎用機も併用することとした。ネットワーク環境をこれまでのクローズなシステムから，神戸大学ワイドな KHAN[(7)] とし，電子メイル，WWW などのインターネットサービスが提供されるようになった。各研究室に配備したパソコンも Pentium を搭載する高性能機で種々の統計処理も難なくこなせるように

(7) Kobe Hyper Academic Network，神戸大学の学内ネットワークシステム。基幹部分に ATM を利用した高速・高帯域 LAN である。

なった。

1996年当初は，過渡期で研究所のデータの多くはまだ汎用機にあり，データの抽出は依然としてSECRETARYを利用して汎用機で行い，これをパソコンに転送し，Secto123でCSV変換して統計処理をパソコンでするという形態は本質的に変わらなかった。ただ，これにワークステーションが加わり，一部の統計処理を，例えば，TSPを使ってワークステーションで行うようになった。また，情報処理センターにあるワークステーションにはSPSSが導入されており，これをサーバーとして利用するという方法も徐々に使われるようになってきた。要するにネットワークがこれまでの研究所だけのクローズな汎用機とPCだけという図式から，各研究室のパソコンが，ネットワークを通じて世界と繋がるという図式に変貌を遂げたのである。これらのネットワークサービスの多くは，クライアント・サーバー・モデルで分散処理されているが，先に示したパソコン上での統計，分析作業は，データ処理のクライアント・サーバー化を意味している。

こうなるとデータのソースは研究所汎用機だけではなくなる。インターネット上には多くのデータが存在し，それを利用する研究者も多くなる。データも何も統計に限るわけではなく，新聞記事のような文字情報，また図の情報，図書情報，NBERのWorking Papersなどの学術論文等，比較にならないほど多様なそして大量のデータが眼前に現れることとなった。

まさに，大型汎用機でしかできなかったことが，いとも簡単に安価なパソコンで実現でき，それ以上のことが実際行われている。ネットワーク経由で，大きなサイズのデータを簡単に転送できるようになったことが，パソコン，ワークステーション，汎用機の同時利用を促進した一因であることも重要である。

RIEBデータベースはかかる背景で，新しい時代にふさわしい装いを施してスタートすることとなった。先ず，それまで汎用機上にあった多国籍企業データベースを新しくワークステーション上のデータベースとして構築しなおした。

それを基本に日経総合経済ファイル，IMF の IFS 等，既存のデータの新システムへの移行を手がけるとともに，これまでに研究所に存在していなかった新しいデータベースも付け加えた。例えば，OECD 貿易ファイルの相手国別，財別，年別データの任意の検索システムがそうである。また，国連機関の統計データで欠落している台湾のデータは台湾中央銀行から IFS フォーマットのものを直接取りよせ，入力し，*International Financial Statistics* (*IFS*) と *Balance of Payments Statistics Yearbook* (*BOPSY*) の欠落部分を補った。このあたりは国際東アジア研究所 (ICSEAD) の「東アジア経済データベース研究」のプロジェクトの成果である。

1.4 本書のプラン

以下の章を簡単に紹介しておくと，第 I 部の第 2 章では新たなデータベース構築のデザインを議論し，第 3 章では実際のデータを DB データとして標準化する際の種々の問題点を指摘した。第 4 章では，リレーショナル・データベースのエンジンとして採用した Oracle のパフォーマンスを調べ，本書の中心である WebDB のプログラムデザインを第 5 章で紹介している。第 I 部を第 6 章のデータベース利用の将来で締めくくり，第 II 部はユーザーの便宜を考えて，マニュアルとしても使えるように多国籍企業データベースなどの各データベース個別の議論を行った。第 III 部は，情報公開の趣旨から実際に使ったプログラムのソース，そして種々の技術情報を載せている。

第2章　RIEBデータベースのシステムデザイン

コンピュータシステムをとりまく状況は，ダウンサイジング[8]と分散化を背景に，クライアント・サーバーモデル，エンドユーザー指向の普及など，めまぐるしく変化している。インターネットなどに見られる利用者の急増は爆発的である。また，エンドユーザーのコンピュータ環境はネットワーク接続が標準的になってきており，Pentium IIプロセッサーやPower PCプロセッサーを搭載し，2GBないし8GBのハードディスク，マルチメディア対応，CD-ROM（またはDVD）搭載というように，大きく性能が向上した。一世代前の汎用機以上の能力を個人が持てる時代となったといっても過言ではない。このことは，これまでのデータベースのあり方との関わりで，プログラミングや，ネットワーク環境のあり方が大いに変貌したことを意味する。

前章で紹介したように，神戸大学経済経営研究所でもダウンサイジングを実現し，分散化の真っ只中にある。

現在のコンピュータシステムのデザインでもっとも問題となるのは，スケーラビリティがあるかないかである。スケーラビリティとは，そのシステムが，要求に応じてその処理能力・規模を伸ばしていくことのできる程度を意味する。

今ひとつ重要な概念に，ポータビリティがある。ポータビリティとは，そのシステムの部分又は全体を，異なるプラットフォーム上に移植する場合の容易さを意味する。ある汎用機でデータベースを運用しているとしよう。汎用機からの移行が必要となったとき，それを移植できなければ，高機能で安価である

---

(8)　パソコンを含む小型で高性能なコンピュータの普及によって，より小さく安価な計算機で，大規模な処理を行なうこと。大型汎用機の情報システムをワークステーションやパソコンを使ったシステムに移行することを指す場合が多い。

としてもそうした機器は選択されない。このポータビリティと，先のスケーラビリティが今後のコンピュータ環境を考える上で最も重要である。

スケーラビリティとポータビリティが重要な現在のコンピュータ環境では，「データベースシステムを構築する」ということは，もはやどのようにプログラミングするかということではなく，「現在利用可能なソフトなどを含めたコンピュータシステムを利用して，それをどうデザインするか」ということになる。一般的なデータベースの形式を採用し，商用のデータベースエンジンを利用して，抽出等をどのようにデザインするかが重要となってきているということである。以下，具体例を交えながら議論を深めることにする。

2.1 エンドユーザー指向

ダウンサイジングと共に始まった分散処理が，今後ますます進むと，エンドユーザーの手元のデータ処理はより多くなされるようになるであろう。必ずしもコンピュータ・スペシャリストでないエンドユーザーとしての研究者にとってのメリットは以下のとおりである。

1．操作方法が同じ。分散処理の環境になければ，ホストが変わると，OS，アプリケーションが異なり，その使用法も大きく変わるのが普通で，コマンドや操作方法などに習熟するのが大変である。しかし分散処理によって，ほとんどの処理が手元のパソコンで行えるなら，例えばWindows95であれば，Ctrl+Cでコピーというように，どのソフトもWindows95での操作体系に従って作られているので，ユーザーにとっては非常に便利になっている。
2．24時間あらゆるところで。データ処理の地理的，時間的制約が外れた。ソフトを導入したパソコンとデータさえあれば，いつでもどこでもデー

タ分析が可能になったのである。

3．加工が自由。自分の望む形式・体裁での出力が得られるようになった。データ処理のソフトの数は多く，抽出したデータを例えば表計算ソフトの形式にして保存しておけば，後日，それをグラフ作成ソフトで思い通りのグラフに描いたり，それをワープロに張り付けたりが簡単にでき，きめこまかな再加工が可能になった。

従来の大型汎用機では対話型で処理するのではなくバッチ処理が基本であった。うまく設計されたソフトでもコマンドを覚えるのに結構苦労したものである。ハードウェアの低価格と高性能化はコンピュータの普及を大きく伸ばした。これにはApple社のMacOS，Microsoft社のWindows95に代表されるグラフィカル・ユーザー・インタフェイス(GUI)がコンピュータ・イリタレート(あまりコンピュータが得意でない人達)を巻き込んで，コンピュータ利用者の裾野を大きく広げたのである。社会科学系の研究者もemailやワープロだけにとどまらず，日常的にデータ処理に携わるようになった。

今やマニュアルなしで，表計算ソフトのExcelなどでデータ処理はいとも簡単に行える。大型汎用機から利用者離れが進んでいるが，それは時代の流れであり，より便利で簡単なシステムに利用者が流れることに棹さすことはできない。研究所のデータベースシステムも汎用機から離れエンドユーザーにより近く，馴染みやすいサービスに変貌を遂げる必要がある。それによって，データ処理の利用者層の拡大を実現することができるからである。

2.2　新しいデータベースの考え方:クライアント・サーバーによるデータ処理モデル

新しい時代のデータ処理の姿は基本的に次のようなものであろう。

1．データ自体はサーバーに集約して管理し，ユーザーはネットワークを通じて必要なものをダウンロードする。
2．データの解析は手元のパソコンで普段から利用しているツールを用いて行う。

すなわちデータ処理のクライアント・サーバー・モデルが普通の形態となる。データは管理者によってサーバーに集中的に追加・更新されており，これがデータベース・サーバーとなる。ユーザーはいつも利用しているパソコンで自分にとって使いやすいツールでデータを処理・分析する。これがクライアントとなる。データベースが提供してくれる機能は，自由なデータの取り扱いと二次元の表データのなかから必要な部分を抜き出す機能だけであり，複雑な分析や計算処理はすべてクライアント側で行なう。

大型汎用機上のデータベースソフトではこうした分散処理はそもそも設計段階で念頭になかった。データ抽出・解析すべてを大型機で行うという前提で，例えばSECRETARY などは設計されていたのである。勢い，ソフト自体大きなものとなりマニュアルも分厚く，達人でない限りソフトが使いこなせない，そういうものになってしまった。複雑な操作がユーザーを限定し，コンピュータは難しいもの，一部の人達だけのものという印象を与えたのである。今日では，複雑な分析処理・グラフ化処理などは，データベース・サーバー側でもクライアント・パソコン側でもどちらでも出来るようになっている。このような状況下でユーザーは手元の使いなれたパソコン・ソフトでより複雑な処理を行なう傾向にあり，サーバー側の複雑な機能は結果的にほとんど使われなくなる傾向になっている。研究所では大型汎用機とパソコンをネットワークで繋いでデータベースの利用を図っているが，SECRETARY の統計処理機能は利用されることは全くなくデータ抽出のためにだけ使われているというのが現状である。

2.3　RIEBデータベースシステムの新しいデザイン

データ処理システムを設計することは，つい最近までは，ホストコンピュータ上でソフトウェアを開発することに等しかった。しかし現在ではシステム全体のデザインがデータ処理システム設計の大きな部分を占めるようになっている。

すでに述べたが，RIEBデータベースシステムはクライアント・サーバー・モデルにもとづいており，以下のような要件を満たすべきと考えて設計されている。

・サーバー側，クライアント側ともに，高速・大容量のデータ処理ができること。
・両者の間では，抽出したデータをユーザーの扱いやすい形式で簡単に受け渡せること。
・クライアント側では，エンドユーザーが日常利用するコンピュータシステムに近い扱いやすいユーザー・インタフェイスを持つこと。

このためには，

・大量のデータを高速に扱えるデータベース・サーバーシステム
・大量のデータを扱えるパワフルなエンドユーザーのコンピュータ
・大量のデータ転送が可能な高速ネットワークシステム

がハードウェアとして必要で，それに加えて，

・上記すべての点におけるスケーラビリティ
・システム全体のポータビリティ
・直観的で一般的なユーザーインタフェイス

が必要である。このことを以下では詳述してみよう。

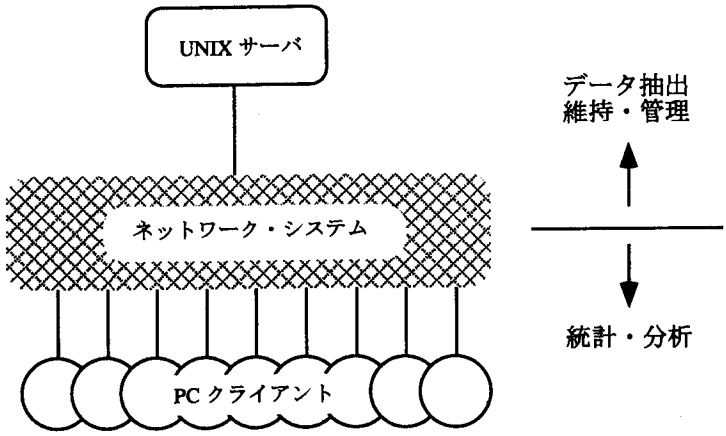

図1：RIEB データベースシステム構成図

2.3.1 システムの構成

研究所では現在2台のUnixワークステーションによるサーバーと，35台のPCクライアントによってシステムが構成されている（図1）。

これらを結ぶネットワークシステムとして，神戸大学の高速キャンパスLANであるKHANを使っている。

Unix搭載のサーバーシステムでは，高速ネットワークインタフェイスに大容量のディスクアレイを組み合わせている。UnixはSMP[9]技術などによってスケーラビリティの確保が可能なOSである。これにデータベースエンジンとして，商用RDB[10]であるOracle 7[11]を使用する。Oracleを選択したのは，これが大容量のデータを扱うことができスケーラビリティに優れているからである。

---

(9) Symmetric Multi Processor，複数のCPUによる処理能力向上のための技術。

(10) Relational Data Base，関係データベースと訳される。本書ではRDB管理システムなども含めて単にRDBと表現する。

(11) 米国Oracle社製のRDB管理システム。多種のプラットフォーム上への移植とツール群の豊富さで知られ，ここ数年間中規模以上のRDBシステム市場において最大シェアを占める製品である。

クライアントシステムは，高性能のPC/AT互換パソコンである。Intel社のPentium 133 MHz CPU，32MBのメモリ，1GBのハードディスク，230MBのMO（光磁気ディスク）の構成で，容量不足には，拡張が簡単にできる機種選択になっている。ワークステーション・パソコン共にこれからますます性能が向上していくと考えられ，それだけ製品の陳腐化も速い。これらの設備の性能向上は定期的にアップグレードで対応できるようにした。

ネットワークはKHANで，研究所機械計算室に設置したサーバーマシンはすべて相互に100Mbpsで接続している。各研究室のクライアント・パソコンへは150MbpsのATM[12]網，または100MbpsのFDDI[13]網を経由して接続している。各クライアント・パソコンへは，現時点ではほとんど10Mbpsのスピードで接続されている。各利用者が常時この10Mbpsの速度を専有できるような配置としている。今後のKHANの発展によって，サーバーとクライアント間のバンド幅は広がり，より高速になるものと考えている。

2.3.2　スケーラビリティ

現在のコンピュータシステムのデザインでもっとも問題となるのは，スケーラビリティ，すなわちシステム全体の拡張性である。

RIEBデータベースシステム全体の能力向上が継続的に必要となる要因は2つある。

1．提供されるデータ量の増加

2．一般的なコンピュータシステムの性能向上

---

(12) Asynchronous Transfer Mode，高速LAN技術としての実用化が最近急速に進んだ通信規約。

(13) Fiber Distributed Data Interface，伝送媒体に光ファイバを用いた100MbpsのLAN規格。

データは毎年増加するものであるし，データベースシステムも自己増殖するものである。コンピュータシステムはこれに対応できるものでなければならない。また，コンピュータの能力・性能は1，2年の短期間に2倍になってきている（2.3.5参照）。こうした動きにも対応できなければユーザー離れが起こる可能性が高い。スケーラビリティが求められる所以である。

データ量の増加は，単純な新年度データの追加によるものだけではない。データを提供する機関は，CD-ROMなどの新しいメディアの普及にともない，安価に大量のデータを供給するようになってきた。例えばOECD貿易ファイルは1997年から，従来のオープンリール・テープ（MT）数本が，数枚のCD-ROMに取って代わられた。CD-ROMの容量は650MB超と，MTの200MB程度に比べて圧倒的に大きい。テープでは毎年1年間分のデータのみが提供されたが，CD-ROM版では10年間の時系列データが圧縮された形で提供され，テキストに変換した場合実に1.8GBにも及ぶ。メディアの大容量化にともなって研究機関が維持するデータの量は今後飛躍的に多くなるものと考えられる。

一方，コンピュータシステムの性能は，「ムーアの法則」[14]に従う形で指数関数的に向上しつづけている。

上記のメディア容量の拡大にともなう供給データ量の増加も，このコンピュータシステム全体の能力向上に裏打ちされてのことである。

このような状況下では，一時的に高性能を謳ってもそれ自体それほど意味がなく，常に高性能を維持できることの方が重要となってくる。時代の流れ，高性能化に常に追随できるかどうかが問題なのである。データベースシステムはスケーラビリティの高いコンポーネントを用いて構築することが重要と認識する所以である。

---

(14) Intel創業者の一人であるGordon Mooreが1965年に行なった「18-24カ月ごとに新しいチップの性能は2倍になる」という予測。この傾向が持続すると，コンピュータの能力は指数関数的に大きくなるが，現在までこの予測はおおよそ正しい。資料［アスキー］に1982年以来のCPUなどの性能向上が示されている。

サーバーシステムとしてのUnixワークステーション，Oracleデータベース，クライアントシステムとしてのPC，Windows95，ネットワークシステム，これらが研究所のシステム構成であるが，どれもスケーラビリティにすぐれたものである。ある特定のメーカーに依存する特殊なもの（ことに汎用機）は採用せず，極めて一般的な製品構成となっていて，今後の高性能化にも十分追随していけるのである。

2.3.3　ポータビリティ

コンピュータシステムをとりまく技術の変化は，年々激しいものになっている。このような状況下において，システム自体のポータビリティがシステムの善し悪しを決めるもう一つの判断基準となる。

RIEBデータベースシステムの抽出・加工ソフトSECRETARYは，主にPL/I言語で記述され，データ更新などの周辺処理はJCLと呼ばれるジョブ処理制御言語によって記述されている。この二つの言語は，IBM系の汎用機システム(15)に採用されているもので，それゆえ，SECRETARYをIBM系以外の汎用機に移植することは難しかった。またワークステーションに移植するのはより一層難しいことであった。このシステムは明らかに研究機関の機種選択の幅を狭め，大型汎用機が主流の時にはまだ実害は少なくてすんだが，ダウンサイジングによるワークステーション中心の環境下ではこの2つの言語は全く用をなさない。

今後のコンピュータシステムの変化はますます激しいものになると予測され，データベースシステムはその時々に主流になるシステムへの移行を求められ続けるだろう。そのためRIEBデータベースシステムではポータビリティを重視

(15)　1960年代に発表されたIBM SYSTEM/360はその後の計算機システムの実質的な標準となった。その後継機であるSystem/370を含めて，多数の互換機が存在し，これをIBM系の汎用機と表現する。

している。

PL/I や JCL で記述された SECRETARY は明らかにポータビリティを欠如している。それではポータビリティを確保するには一体どうすればよいのだろうか? 一つの答えはどのシステムでも利用できる(商用の)データベースエンジンを使うことであろう。データだけはテキスト形式に直せる形でデータベースサーバーに保存しておき,それを抽出するデータベースエンジンに一般的なものを使う。[16] 時代のニーズに合わせてそうしたデータベースエンジンは機能を強化し,それまでの資源を無駄にしない形で多くのプラットフォームで利用できる形で変貌を遂げるであろう。データはテキスト形式であるのでどこにでも移植は簡単である。異なったハードウェアに移植する必要があっても問題は少ない。研究機関で独自にソフトを開発するにしても,どのシステムでも動く言語,例えばC言語[17]で記述するようにすれば,ますますポータビリティは高まるのである。

データベース・エンジンである Oracle はこうした要件を満たしている。大型汎用機からパソコンまでさまざまなプラットフォームに互換性を保ちながら移植され続けている。データベースの操作言語は ISO[18] によって標準化された SQL[19] であり,他の RDB システムとも互換性がある。

プログラミング言語としてはやはり ISO と ANSI[20] によって標準化された C

---

(16) Oracle にデータを読み込ませる前の段階では,どのデータもテキスト形式で用意している。Oracle の内部形式は公開されておらず,Oracle に読み込ませた後のデータの保存形式は不明。

(17) ベル研究所において Unix システムを記述するために開発された汎用プログラミング言語。

(18) International Organization for Standardization, 国際標準化機構と訳される。工業及び科学技術に関する国際規格の制定を行なっている。

(19) 現在の事実上の標準である RDB 問い合わせ言語。本来 Structrured Query Language の略であったが,現在では単に SQL とだけ記述されている。

(20) American National Standard Institute, アメリカを代表する非営利の標準化機関。

言語を採用した。C言語はもともとUnixシステムのために開発されたが標準化などによって普及し，現在では最も多くのプラットフォームで利用できる言語となっている。

データベース構築の際には細かな配慮も必要で，例えば，文字コードについては以下のことを考慮した。漢字はJIS第一，第二水準文字のみを使用し，特殊な文字・記号などは表示不可の文字（例えば'='）に置き換えてデータに保存する。また，システム固有文字や外字も利用しない[21]。これもデータのポータビリティを高めるための策である。

このように研究所の現在のデータベースシステムではポータビリティを重視しているのである。

2.3.4　ユーザー・インタフェイス

エンドユーザーにとっては，使用法は直観的に分かるものがよくマニュアルなしで使えるものがベストである。データベースのエンドユーザーは毎日使うというのは稀で，1ヶ月のうち1週間は集中して使うが他は使わないといった場合が多い。複雑なコマンド体系では，データベースを使う度に，覚えていたはずのコマンドをマニュアルを見ながら思い起こす必要がある。それゆえユーザー・インタフェイスはできるだけ直感的で，誰にでも自明のものがよい。

現在の一般的なユーザーが使用しているMicrosoft WindowsやMacintoshのソフトウェアは，GUIを利用している。例えばSECRETARY，TSP，STEPSで用いていたような，以前では多く見られたコマンド・ベースのユーザー・インタフェイスは，もはや多くのユーザーにとって馴染みの薄いものになりつつある。

---

(21)　例えば，富士通の汎用機を用いて作成された東洋経済海外進出企業データではJEF拡張漢字が，IBMの汎用機を用いて作成された興銀財務データではIBM固有文字が含まれていた。これらの文字はすべて'='や'■'として格納している。将来的には形の似た標準文字への置換などを検討した方がいいだろう。

RIEBデータベースシステムでも，簡易言語による操作を避け，極力GUIを利用する。まず，最初の試みとして，WWW[22]を利用したシステムを開発した。データの抽出操作を，WWWを利用して行なうのである。WWWに代表されるインターネットサービスは，ユーザーが毎日利用するソフトウェアの代表的なものの一つである。これをフロントエンドにすることで，ほとんどマニュアルを見ることなく作業ができる環境を作ることがここで重要なのである。

2.3.5 SECRETARYの再評価

ここでRIEBデータベースシステムとして長い間利用されてきたSECRETARYについて，スケーラビリティとポータビリティの面から再評価を試みたい。

SECRETARYは**2.3.3**で示したように，IBM系汎用機上に構築され，利用されてきた。1979年以来，研究所では大型汎用機システムをほぼ4年ごとに更新してきた。その性能は，**2.3.2**で紹介した「ムーアの法則」に従って，指数関数的に向上してきた。

表2に具体的な性能向上の経過を，導入したモデルのメモリとディスクの容量，MIPS[23]の増加によって示す[24][日経]。また，表3に，同時期のIntel社のパソコン用CPUの性能向上をiCOMP[25]の増加によって示しておいた[アスキー]。

両者をグラフで表示すると，図2のようになる。縦軸は対数でとってあり，汎用機については1977年のM150を，Intel CPUについては1978年の8086/

---

(22) World Wide Web，インターネット上の情報掲示型のサービス。現在のインターネットにおける最重要サービスの一つ。

(23) Million Instructions Per Seconds，1秒間に実行可能なCPU命令の数。必ずしも正確ではないが，おおよそのCPU処理能力の目安となる。

(24) HITAC 8350は汎用機ではないが参考までに加えた。現行モデルのM640/45Eは1999年まで利用予定であるためグラフは1999年までとした。

(25) iCOMP (Intel Comparative Microprocessor Performance)，Intelによる性能指標。Intel製CPUの性能比較の目安になる。メーカーであるIntel自身によるiCOMPの数値で性能向上を見るのは厳正さに欠けるかもしれないが，他に妥当な数値がないためこれを採用する。
http://www.intel.co.jp/jp/procs/perf/icomp/index.htm

表2：研究所が導入した汎用機の能力変化

機種	導入時期	メモリ容量	ディスク容量	MIPS
8350	1974年2月	256 KB	120 MB	0.14
M150	1976年12月	2 MB	1 GB	0.23
M240D	1983年9月	8 MB	5 GB	1.49
M260D	1987年12月	24 MB	12 GB	4.07
M640/35E	1992年2月	64 MB	25 GB	6.22
M640/45E	1996年2月	96 MB	40 GB	9.94

表3：Intel CPUの能力変化

CPU	出荷年	クロック(MHz)	iCOMP
8086	1978	10	0.6
80286	1982	8	1.9
80386	1985	16	2.5
80486	1989	33	15.0
80486	1992	66	29.7
Pentium	1994	100	81.5
Pentium Pro	1995	200	222
Pentium	1996	166	127
Pentium II	1997	266	303
Pentium II	1998	450	483

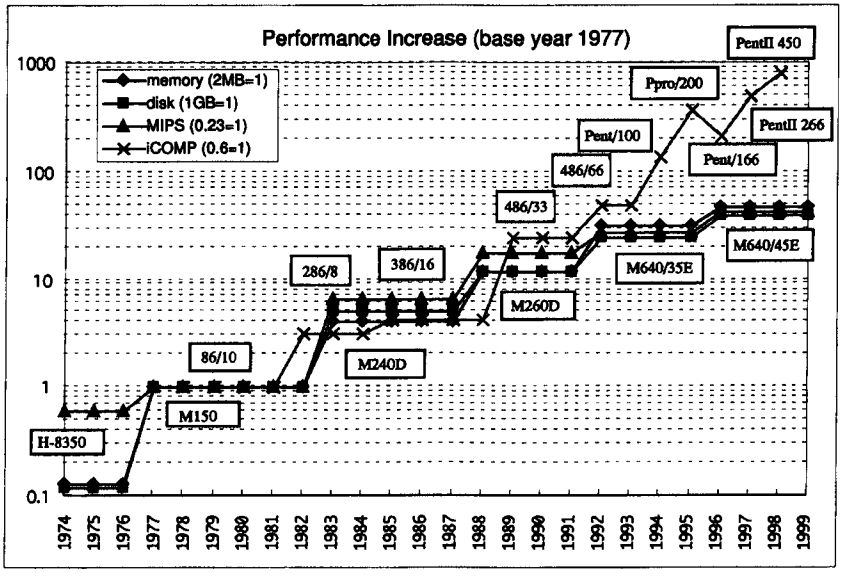

図2：研究所が導入した汎用機とIntel CPUの能力変化の比較

10MHzをそれぞれ1とした比率で示した。

汎用機については，1977年のM150以来，指数関数的に性能が向上してはいるが，その度合いは年を追うごとに緩やかになった点に注目すべきである。これは研究所の導入した大型汎用機に限ったことではなく，汎用機そのものの性能向上が鈍ってきたためである。

これに対してIntel CPUの性能向上は，止まることなく指数的に伸びている。1980年代までは，汎用機の性能と，Intel CPUの性能はほぼ同じペースで向上してきたが，1990年代に入ってから汎用機の性能向上は鈍り始め，逆にIntel CPUの性能向上は一層著しいものとなってきた。1990年代はまさにダウンサイジングの時代であり，この時期はパーソナルコンピュータを含む小型計算機の性能の向上が急で，大型汎用機はこれに追随できなかったことをはっきりと示している。

SECRETARYは1981年頃に作成されたシステムであり，大型汎用機がコンピュータの主流であった時代のプログラムであった。汎用機の性能はSECRETARYが稼動していた15年以上の長期間にわたって向上しつづけていた。しかし時代の流れとともに汎用機の性能向上は緩慢になり，その結果SECRETARYの処理能力向上も期待できなくなった。

SECRETARYのポータビリティが高ければ，汎用機ではない別のタイプのコンピュータ，例えば1990年代以降急速に性能を向上させてきたマイクロプロセッサを利用したワークステーションなどに移植することによって，SECRETARYの処理能力向上は維持できたであろう。しかし**2.3.3**で議論したように，SECRETARYの開発言語がPL/IとJCLというIBM大型汎用機システム用のものであったため，IBM系汎用機以外のシステムへの移植がほとんど不可能であった。

これが原因でSECRETARYは汎用機から離れることが出来ず，研究所はSECRETARY利用によるデータベースという桎梏から長期間逃れることがで

きなかったのである。しかし，性能の相対的低さと使い勝手の悪さから次第にユーザーの汎用機ばなれがおこり，ワークステーションを導入しSECRETARYとはまったく別のシステムを新たに構築する必要に迫られたのである。

2.4　RIEBデータベースの実際

現在のRIEBデータベースシステムについて，例を示しながら説明する。[26]

2.4.1　構成の概念

RIEBデータベースシステムの構成の概念図を次に示す。

左がユーザー側で，右がサーバー側，その2つを結ぶのが真中のネットワークサービスである。

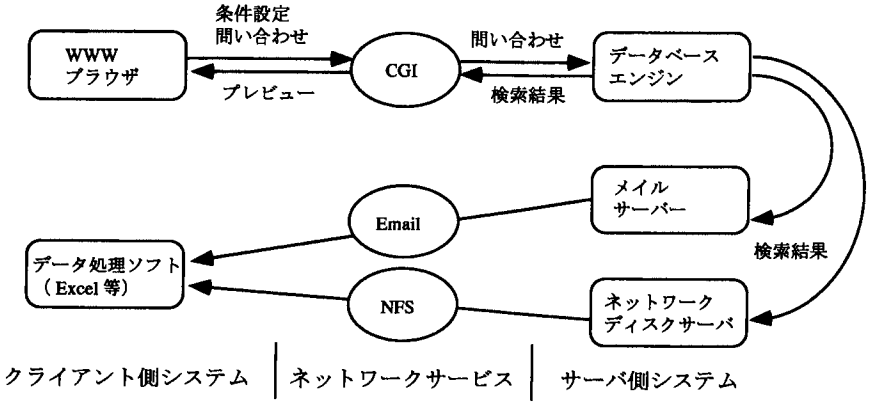

図3：RIEBデータベースシステム概念図

(26)　RIEBデータベースシステムは開発の途上にあり，ここに示された構成は変化していく可能性が高い。

図3の左側のクライアント側システムは，普通のパソコンに好みのソフトウェアを導入したものである。Netscape NavigatorなどのWWWブラウザ，Eudoraなどのメイルユーティリティ，Excelなどのデータ処理ソフトをパソコン上に導入する。こうしたソフトは同じ機能を持つものなら何であっても良く，特定のソフトウェア製品に限定されることはない。Netscape Navigatorの代わりにMicrosoft Internet Exploreであっても良いのである。

右側のサーバー側システムは，いくつかの機能に特化したワークステーション群で構成する。ワークステーション何台で実現するかは，その時の設備状況に依存するが，1998年10月現在の研究所では2台で実現している。データベース・エンジンとしては，Oracle及び自作プログラム群によるものの二種類をデータの量や性質によって使い分けている。

このクライアントとサーバーの両者を結ぶのがさまざまなネットワークサービスである。サーバー，クライアントの両サイドで機能する必要があるため，なるべく一般的なものを使用または作成して利用している。

CGI[27]はWWWシステムで標準的なブラウザによるプログラム起動のインタフェイスである。起動するプログラムに対して，各種のパラメタ設定などが可能である。Emailによるデータの受渡しは現在普及しているSMTP[28]やPOP[29]を利用している。ネットワークディスクへのアクセスにはNFS[30]を用いている。NFSはUnixワークステーションでもっとも普及しているファイル共有システムである。

---

(27) Common Gateway Interface
(28) Simple Mail Transfer Protocol，現在のインターネットにおける事実上標準のメイル転送手順。
(29) Post Office Protocol，同じくインターネットにおけるメイル参照手順の一つ。
(30) Network File System，Sun Microsystems社が開発した，ファイル共有システム。

2.4.2　操作方法

RIEBデータベースシステムは，すべてWWWによる操作インタフェイスをもち，共通の簡単な操作で作業を行うことが出来る。データベース利用者のためのマニュアルは93ページの第7章以降に改めて書くのでここでは簡単な紹介に留める。

図4に典型的な例として日経総合経済ファイルの操作インタフェイスを示しておこう。

ユーザーはWWWブラウザで次の手順でデータベース検索を利用する。

1．利用者はWWWブラウザで抽出したいデータの条件を指定する。
2．実行結果をWWWブラウザ上でチェックしデータの検索条件をさらに絞り込む。
3．抽出結果は電子メイルの添付ファイルによる送信，またはネットワークディスクに対する書き込みによって利用できる。
4．ユーザーは電子メイルまたはネットワーク・ディスクに該当ファイルを見つける。見つかった抽出データはそのままExcelなどのアプリケーションで表示でき，それを更に統計分析ソフトなどによって分析・加工することができる。

検索項目はポップアップメニューやチェックボタンなどにより，入力・選択するもので，WWWブラウザの一般的な機能を利用して検索条件が指定出来るようにしている。

具体的な条件設定・検索実行の方法については，94ページの，**7.1**にまとめたので参照されたい。

2.4.3　データベース検索結果の取得方法

図3で示したように，ユーザーは検索結果を何通りかの方法で取得できる。プレビューとしてのWWWブラウザによるもの，そしてメイルでの送付，ネッ

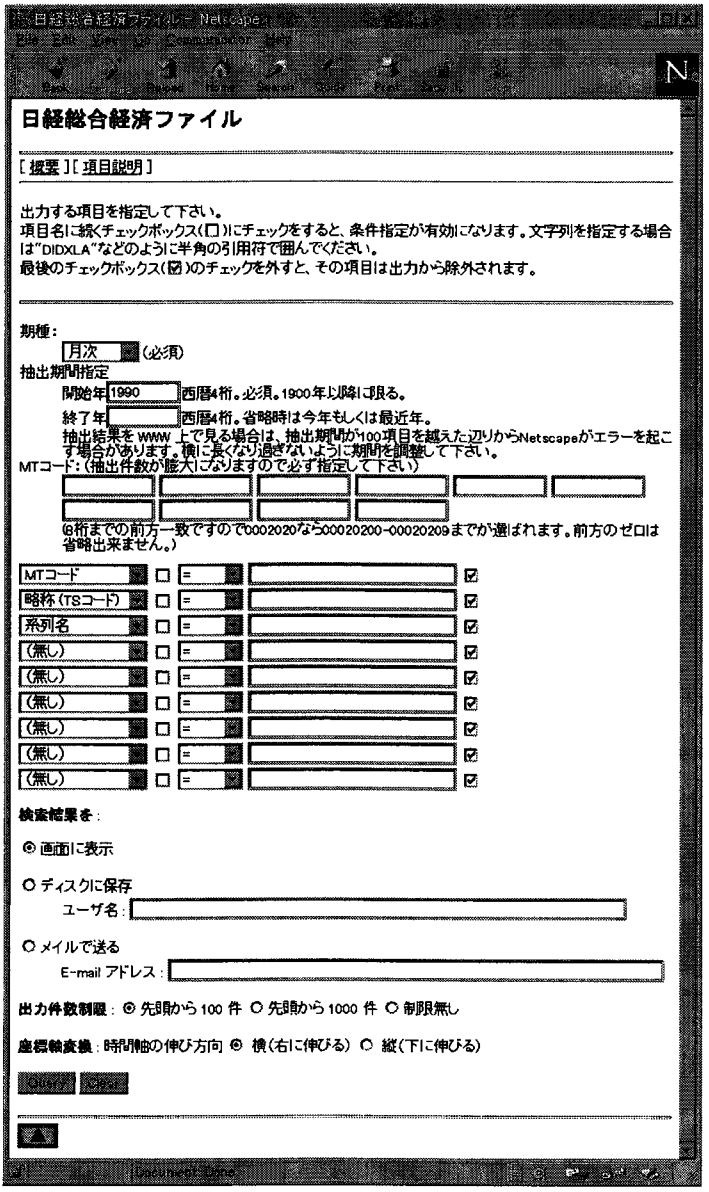

図 4 : 日経総合経済ファイル WWW インタフェイス

トワークディスクへの出力が代表的なものである。これ以外にもFTP[31]により，出力したネットワークディスクのファイルをパソコンに取り出すことも可能である。利用者からの要望によって，他の方法によるものも追加することができる。ネットワーク技術の進歩により新しい形態・プロトコルでのサービスが利用可能になったときでも柔軟にそれに対応できる。

このように一見サービス過剰の感がして無駄があるように見えなくもないが，多くの経路を用意することがシステムの柔軟性となり，ユーザーにとっての便利さにつながると考えている。将来どこで，どのような形でユーザーがこのデータベースを使うか予想できないが，このようにフレキシブルなシステムにしておくと，どんな場合でも対応が容易であるはずである。つまり，複数の経路を用意し，サーバーとクライアントが独立にいろいろな作業が出来るという緩やかな関係が，システム全体の柔軟性，ポータビリティを実現する。

(31)　File Transfer Protocol，ファイル転送のためのプロトコル，またはその機能。

第3章　データベース・データの標準化

データベースを作成するにあたって，重要な最初のステップは，いかにデータを整理し，どのような書式でデータベースのファイルとして書き込むかである。一旦保存したデータファイルからいかに任意のデータを抽出するかはデータベース検索プログラムの問題であって，これは次章以下で議論する。本章では，経済経営研究所でこれまで維持してきた大型汎用機上のデータベースを拡張して，ワークステーション上で稼動するように，新しいデータベースシステムを再構築するにあたって問題となった点を詳述する。

RIEBデータベースのデータは，基本的に，外部機関から多くはMTで購入している。例えば，多国籍企業データは東洋経済新報社から，日経総合経済ファイルは日本経済新聞社から毎年継続的に購入している。こうしたデータは一見マニュアルに従って簡単に処理できそうであるが，実際は複雑な問題が多々存在する。データそのものの問題に加えて，一般的に，次のような問題がある。

1．データの収録形式が提供機関ごとに異なっていること

2．キー項目の値，コード体系について汎用性がないこと

3．不正なデータが含まれている場合があること

4．一般的なデータ処理で扱えないサイズのデータがあること

5．月次，四半期，年次データなどがコンシスタンシーに欠けること

データの形式は提供される機関ごとにまちまちで，ことにコード体系・データの型などが多様である。データベース作成側で時間を惜しんで，こうしたデータをそのままユーザーに提供するなどの簡易なやり方をすると，ユーザー側からみれば，ユーザーフレンドリーでもなく，効率的でもない。ユーザーはまず仕様や例外を調べることから始めなければならないために，データベースに習熟するのに相当時間がかかってしまう。ユーザーのデータベース離れを誘引す

るかもしれない。データの形式やデータの型は，標準的なものをまず構築して，それにあわせて，各機関から購入したデータを整理し，データベースサーバーにデータベースファイルとして書き込み，保存することとした。このプロセスを，本書ではDBデータの標準化と呼ぶことにする。実は，データの準備というのは，このDBデータの標準化の作業を意味する。

以下では，実例をあげながら，DBデータの標準化の各ステップを示すことにする。最後に標準化の意義について述べる。

3.1 データ形式

まずデータの形式の多様性について書こう。メディアの形態，文字コード，単位など，およそデータについては形式がそれぞれについて異なっているのが普通である。

メディア：MTは普通1/2inchオープンリールテープ，であるが，時によっては8mm Exabyteテープ，4mm DATテープ，で提供される場合がある。また最近ではCD-ROMでの提供も増えてきた。このようにメディアの形態は多様であるが，それのみならず，テープ等の磁気メディアに関してはそれぞれ記録密度の相違もある。RIEBデータベースのデータ受け入れに関しては，ほとんどのものがオープンリールテープであるが，OECD貿易ファイルの最近のものはCD-ROMで受け入れている。

文字コード：1バイト文字にはASCII[32]，EBCDIC[33]，EBCDIK[34]があり，さらに日立は特殊なカナまじりのEBCDICを使用している。2バイト文字ではさらに

(32) American National Standard Code for Information Interchange, 1963年にアメリカで標準化された文字コード。現在でもパソコンやワークステーションなどで広く使われている。

(33) Extended BCD Interchange Code, 1964年にIBMによって定義された文字コード。IBM汎用機を中心に使われている。

(34) 国内で作られた，EBCDIC符合の英小文字部分をカナに置き換えた文字コード。

事情は複雑で，JIS[35]でもシフトコード付き，無しのもの，Shift JIS[36]，EUC[37]といった比較的良く使われるものから，JEF[38]，KEIS[39]，IBMホストコード[40]と汎用機で頻繁に使われるものまで，実に多種多様である。RIEBデータベースでは，クライアントのパソコンシステムに合わせて1バイト文字はASCII，2バイト文字はShift-JISでコードを統一することにした。また，2バイト文字の英数字記号は，コードの統一のため可能な限り1バイト文字に置き換えることとした。

数値表現：可視表現，バイナリ[41]，パック形式，ゾーン形式，BCD[42]などさまざまな形式がある。バイナリ形式ではエンディアン[43]の相違に注意する必要がある。

符合，小数点位置：符合についてはパック10進法や，可視表現で全く独自に符合カラムを設ける場合などがある。小数点位置はさらに複雑であり，COBOLやPL/Iでは，仮想小数点形式が採用されていて，この形式をよく見かけるが，定式はない。例えば日経総合経済ファイルでは，15桁中の下から5桁めに小数点位置を固定した上で，小数点以下の有効桁数を別カラムに一桁の数値で指定している。

欠損値：単純に空白あるいはヌルコードが入る場合もあれば，欠損値を示す項目を別にとる，特定値（NA，-，99など）を入れるなどさまざまである。例えば日経総合経済ファイルでは，特定値-9999999999.99999が欠損値を示す。

---

(35) JISが規定した漢字のための文字セット。
(36) マイクロソフト社が開発した，パソコンで広く使われる漢字コード。
(37) Extended Unix Code，Unixで漢字を利用するために標準化された文字コード。
(38) 富士通汎用機で使われる漢字コード。
(39) 日立汎用機で使われる漢字コード。
(40) IBM汎用機で使われる漢字コード。EUC，JEF，KEISなどと違い，JIS文字セットと根本的に異なるため，変換が困難である。
(41) OS，言語，CPUの内部表現そのまま。
(42) Binary Coded Decimal，二進化十進符合。
(43) CPUのバイト並び方向。例えばインテル系プロセッサとSparc系プロセッサではエンディアンが異なる。

単位：通貨単位などについても同じ変数が，年によって定義が異なっている場合がある。ある年次までは数値項目に '1000'，通貨単位に 'US$' となっていても，翌年から '100' '万US$' と変更してある場合もある。同じ名前の変数で，単位が%になったり，1985年を100とした指数であったりする場合もある。

これらの形式の相違は，マニュアルを丹念に読み，それで判読できなければデータ出版元へ直接聞き合わせることになる。データを整備するということは，このように極めて労働集約的な作業である。データの性格を熟知する必要があり，この面での専門家の育成も必要である。余談になるが，オーストラリア国立大学の経済関係のデータベースを維持している豪日研究センターでは，データに非常に詳しい職員一人が長年，こうしたことを担当していた。

3.2 キー項目のコード

3.2.1 異なったコード体系

経済・経営データにおいてキーとなる項目のコード体系は標準化されていない。

例えば国コードは，IMF は数字3桁，MT 提供の OECD データは数字4桁の，いずれも独自のコードを用意している。ただ，新しい動きもある。OECD は1997年，CD-ROM で貿易ファイルを提供し始めたが，この際，アルファベット3桁の ISO 3166-1[44] で規格化された国コードを基本に，わずかに拡張したコードを用いるようになった。希望的観測であるが，各データベース提供機関独自

(44) ISO 3166-1: 1997, *Codes for the representation of names of countries and their subdivisions - Part 1: Country codes* は，ISO が定めたアルファベット2文字，3文字，数字3桁による国コードである。日本は JP，JPN，392 となる。二文字コードは（イギリスが gb ではなく uk を用いている例外を除き）インターネットのドメイン名にも利用されている。
http://www.iso.ch/cate/d24591.html, ftp://ftp.ripe.net/iso3166-country codes などを参照。

のコードの使用は，今後減少していくであろう。

貿易業種は一般的な標準国際貿易分類（SITC）が使われているようである。これにしても，Rev.2を使うのか，Rev.3を使うのかといった問題は依然残されている。OECDの貿易ファイルについて，ある年次まではRev.2で，その翌年からRev.3といった場合，時系列全体をデータベース化するためには，Rev.2とRev.3のコード変換表を用意して，どちらかのコード体系で統一する必要があるのである。

国内のデータについては，残念ながら，東洋経済新報社の海外進出企業用の産業区分も，SITCとまったく違った独自コードを使っている。日本経済新聞社の国内の会社コードも独自のもので，国内で一般に普及している企業識別コードたる4桁，ないしは5桁の証券コードとは異なっている。ちなみに日本興業銀行は4桁の証券コードを使用している。

以上，簡単にみたようにコード体系は各データによってまちまちのものが採用されている。やむをえない側面もあるが，今後はデータ提供機関の方で，ISO等々での規格作りを促進し，それを積極的に採用すること，時系列の途中で，違ったコードを使う必要が生じたときには，新しい基準で古いデータも見直して提供することなどが望まれる。

現状では，しかしながら，これは望むべくもない。したがって，現在のRIEBデータベースシステムでは，データ提供者が用意するコード体系をそのまま採用することとした。

3.2.2　コードの一貫性

経済・経営データは日進月歩の拡張をみせている。そしてデータ項目自体が頻繁に変わる。企業コード，国コードなどがキー項目として利用される場合が多いが，企業の統廃合，新規参入は毎年起こるわけであるし，国家にしても旧ソ連の崩壊により，リトアニア等新しい国が沢山誕生した。

もう少し詳しく言うと，国内企業のコードについては，標準的な証券コードは4桁整数であり，扱える数が少ないため，上場企業の変遷にともなって，以前に倒産した企業のコードをしばらく経ってから新しい企業に割り当てるなどの処置が行なわれきた。新しく5桁コードも用意されてはいるが，余り利用されていない。

このような状況にあるため，長期にわたる時系列データを扱う場合には，コードの一貫性をいちいち確認しておく必要がある。単純な入れ替わり程度であれば，データを補正し，DB データの標準化をはかることも有効であるが，コードの変化が頻繁で多数ある場合，データからコードの変遷を追跡して対応するのには非常な手間を要する。[45] 完全な修正を目指すより，多少妥協して，常に新しいデータを入手し，過去のデータもすべて新しいコード体系のものに入れ換える方が有効であろう。コード体系の変化を視野に入れた，現実的なメンテナンス計画を立てることが重要である。

3.3 スクリーニング

コード体系の問題が解決できたとしても，データそのもののエラーは避けがたい。人の手で，一次データが入力されている限り，単純なミスが含まれると考えていた方が良い。

通常，データ提供機関が，どのようなデータのチェック体制をとっているのかは明らかにされないが，データソースから入力，出版する過程のどこかでミスが発生することは避けられない。

---

(45) 例えば時系列データとしての利用を余り重視していない東洋経済海外進出企業ファイルなどにおいては，国コードにかなり頻繁な入れ換えが行なわれていた。このデータを DB データの標準化を実施するために，国コードの変遷も合わせて追跡したが，それにはツールを使っても30時間以上を費やし，それでも修正は完全にはできなかった。現在は追跡した情報とともに利用に供している。

このために，仕様上あり得ない値がデータに含まれることがある。例えば，以下のようなケースである。

・文字項目にヌル記号が含まれている。(132ページ**7.5.4**参照)

・採録開始時期の日付が，採録終了時期より後になっていたりして矛盾がある。(同上)

・JEF漢字項目の空白コードとしてx4040とxA1A1の二種類がある。(110ページ**7.2.3**参照)

こうした明らかなミスをそのままにして，データ提供機関からデータが提供される場合がある。これはMTでのデータ提供の場合のみおこりうることではなく，出版物でも誤字・脱字があるのと同じである。しかし，こうしたミスをチェックすることなく，データベースとしてサービスに出すと，ユーザーは誤った統計データに気づかず，間違った統計分析結果を導くといった，深刻な問題を引き起こすことになる。

現実的な自衛対策としては，受け入れ側で上記にあげたような事例があるかないか，ある程度機械的に検証するしかないであろう。正しいデータが何であるかを受け入れ側で調査し判明するものであれば，入力し直すなどの対処が可能であるが，普通は望むべくもない。

つまり，データ受け入れの段階でデータの信頼性を検査し，異常の発見できなかったデータのみをデータベースに収録するということが出来る最大限の自衛手段なのである。この作業をスクリーニングと呼ぶ。

データの不正パターンはさまざまであり，過去に受け入れたデータの場合はある程度，どのような矛盾があるか見当がつき，機械的にチェックすることも可能である。しかし，初めて受け入れる時にはそのデータに最適の方法で自動的，機械的に検査することはほとんど不可能である。全ての項目について，データ仕様書とデータ内容を見比べながら，各項目ごとに個別に確認する作業を繰り返すしかない。この作業は基本的には手作業で，検査ツールもその度ごとに

新たにプログラミングすることになる。

現在では，こうした作業がデータベース管理者の大きな負荷となっている。データ利用の促進のために，データの信頼性についての情報公開が必要であろう。データ提供機関がデータの整合性のチェックの有無，その方法など，情報公開してくれれば受け入れ側の作業も楽になる。我々も利用ライセンス契約に反さない範囲で，データの検査内容を公開していきたいと考えている。

3.4 多倍長データ

経済・経営データにおいては，金額，数量などに整数値を用いる場合が多く，その桁数も一般に大きい。

現在の通常の計算機は，整数を32bitsで表現している場合が多い。符合の1bitを除いた31bitsは $2^{31} = 2,147,483,648$ であり，10進数で10桁でしかない。確実に扱える範囲となるとさらに9桁までとなり，符合なしとして32bits全部扱えるようにしても，やはり9桁どまりである。経済・経営データで9桁は日常的であり，20桁程度までは十分ありうる。$2^{10} = 1024 \cong 10^3$ であるから，10進数で20桁となると，楽に70bits程度になってしまい，最近ようやくプログラミング言語で普通に扱えるようになってきた64bits整数でも不充分である。

この種の，計算機がネイティブに扱えない桁数を多倍長という。これら多倍長の数値を扱える環境が必要である。Oracleデータベースを含めてデータベースシステムは一般に多倍長整数に対応しており，Oracleデータベースの中では，多倍長の整数はそのまま保存できる。Excelなどのソフトウェアにおいても，有効桁数がまだ15桁程度であるが，それでも多倍長演算はサポートされており，クライアント側で20桁程度の処理が問題なくこなせる環境が徐々に整いつつある。

問題は利用するツール群で，それがC言語で記述されていた場合は，32bits

整数の限界で計算誤りを起こす場合がある。COBOL, PL/I 言語などでは多倍長計算をサポートしているが，自作のツールについては，桁数が十分に確保されていることを確かめる必要がある。PL/I で記述されている SECRETARY においても，整数桁数は10桁程度に制限されており，それ以上の値を持つデータは扱えない。

新しいこの RIEB データベースシステムでは，ツール群は必要がない限り整数項目でも数値として扱わず，可変長文字列として処理することで多倍長数値に対応した。数値として演算する必要が出た場合は，C 言語で開発した多倍長演算ライブラリを用いて処理している。ちなみにこのライブラリでは，50桁以上の有効桁数の計算が可能となっている。

3.5　年換算データにおける過去の年次の補正

経済・経営のデータでは年次と年度が区別される。ことに経営のデータでは，例えば，決算データは決算の期日が変わったりすると，年度に換算して，他の企業データと比較可能なように補正する場合がある。こうしたデータの修正処理において問題が往々にして生じる。

日本興業銀行の財務データファイルの場合の年換算における問題については145ページの **7.7** で述べているので参照されたい。

興銀のケースでは，過年度の全てのデータについて遡及して修正したものが提供されていれば，問題はなかった。一部のみを修正し，かつ，その詳細が仕様書に十分説明されていなかったことが問題をより複雑にした。

もともと，整合的な時系列データを提供するために補正を行っているのである。その補正の意味を正確に掌握した上で，データの最終的なチェックを行わない限り，問題は大きくなる一方であろう。

3.6 分散処理とエンドユーザー指向からみた標準化の意義

DB データの標準化したものは，そうしていないデータに比較して信頼性が増し，かつ取り扱いが容易であることは自明であろう。

標準化によって，ユーザーはデータの使用に関して迷うことなく，目的のデータを検索・抽出して，本来の統計分析に移行することが出来る。標準化せずに，提供機関のオリジナルなデータをそのまま使う場合には，いちいち仕様や例外に注意しながらデータ処理を行なう必要があり，非効率は免れない。

まさに DB データを標準化することの意義はここにある。コンピュータはますます分散処理環境を強化し，エンドユーザー指向を強めている。そのため従来では集中処理システム側で行われていた例外データの処理などもエンドユーザー側に求められるようになった。こうした傾向のもと，DB データの標準化の意義は一層高まるものと思われる。その理由は以下の通りである。

2.1 でコンピュータのダウンサイジングとその普及によって，データの利用者の裾野が広がってきたことを議論した。その一方，**3.3** で述べたように，データのチェックを含むデータの質を向上させる作業はデータ提供機関に任されたままであって，それがどのようになされているかについては情報公開されていない。エンドユーザーが真面目にデータ分析をしようとすればするほど，データの質を確かめるだけで多くの時間がとられることになる。ユーザーが無意識のままエラーを含んだデータを使って分析し何らかの結論を出したとすると，それは大きな社会的問題となりうる。

したがって，データ処理のエンドユーザー指向，分散処理化にとって，受け入れ時点での厳密なデータの標準化は必須の要件なのである。

厳密なデータの検査と DB データの標準化のプロセスは，RIEB データベースシステムの開発・維持の中枢を占める。この作業は極めて労働集約的であることから，研究所機械計算室には大きな負荷でもあるが，その意義は大きく，

標準化したデータの価値，そしてこのトータルなシステムの信頼性は今後上がっていくことであろう。

3.7　データ提供の新傾向：CD-ROM化の問題

最近では，CD-ROMが普及し，価格も安くなり，PCで利用できることから，個人で購入するケースも増えている。操作性も非常によくなっており，ネットワーク対応のものも発売されている。CD-ROMサーバーを構築して，ネットワーク対応CD-ROMをそのままの形で，所員の利用を図るというのも一手であろう。また，著作権が許せば，各研究者のハードディスクに丸ごとコピーして利用するというオプションもある。

ただ，*IFS*，日経NEEDS，*DRI Basic Economics*，*World Development Indicators*等々と揃えていくと，コストもかかり，操作法もすべて覚えるのでは億劫になる。日経は契約をすれば，オンラインでIMFのデータも含めてカバレッジの広い経済・経営データを提供してくれる。しかし，高価なこともあるがそれ以上にそのような日経のデータは必ずしもすべての研究者のニーズをいつも満たすとは限らない。とはいえ，データを必要とするケースがそれほどなく，量もさほどでもない場合は，日経などと契約して独自のデータベースを作らないというオプションの方が，コストパフォーマンスに優れているのかもしれない。研究所では，機械計算室という組織があり，コンピュータの専門家が常駐し，予算があり，ユーザーは常時多くのデータを利用するという環境下にあったので少し話が違う。所内用にデータベースを開発，維持，研究する方が，効率的なのである。

ここ数年のあいだに，CD-ROMによってデータが提供されるようになってきたことは，個人ユーザーにとっては歓迎すべきと思われるが，データベースを開発維持していく立場からすると必ずしも楽観はできない。

CD-ROM を含めた新しい大容量メディアの登場によって，提供されるデータ量がそれを扱うコンピュータシステムにとって大きくなり過ぎてしまうことが起きはじめている。

これだけなら量的な問題で済み，ハードディスクやネットワーク機能を増強するだけで済むが，パソコンで使うことを前提に提供されるデータは CD-ROM に添付される検索ソフトウェアを通してのみ利用可能な場合が多い。これを RIEB データベースに組替えようとすると，途端に大きな障害となってしまう。

以下にその具体的な例を一つ示そう。OECD 貿易ファイルのケースである。

3.7.1 OECD 貿易ファイル

OECD は1993年までは磁気テープでデータを提供してきた。データは固定長の単純なフォーマットで記録されており，そのフォーマットも公開されていた。これを利用するには，コンピュータの専門家 (SE) が処理可能な形式に変換することをまず行った。SE ならずとも，多少のプログラミングができるユーザーならば，容易にそれが行なえた。

データ提供の形態が CD-ROM となったとき，Windows 上で簡単にデータの検索，抽出ができるアプリケーションソフトウェア Beyond 20/20[46] が添付されてきた。これは GUI ベースの実に分かりやすいユーザーインタフェイスをもったソフトで，エンドユーザーでも簡単にデータにアクセスできるようになったが，データはアプリケーション専用のフォーマットで記録されることとなった。このため，添付されたアプリケーションを用いる以外の方法でデータを利用することができなくなってしまったのである。

添付されているアプリケーションは万能ではない場合が多いため，ユーザー

---

(46) Beyond 20/20, カナダの IVATION Datasystems Inc. 製アプリケーション。Windows 対応。http://www.ivation.com/

がより自由な分析を自分たちのツールで試みたいと感じることは多い。

もう少し，具体的にBeyond 20/20の問題点をあげてみよう。

1．レポート国ごとに個別のテーブルを用意し，それを検索するようなシステムが組まれている。したがって，複数のレポート国を対象に検索するときには，対象国ごとに何回も同じ操作を繰り返さなければならない。結構，面倒な操作が必要となり，オペレーションミスを引き起こしかねない。
2．データ更新は一年に何回かあるが，更新のあったレポート国のみのCD-ROMが提供されるため，最新のデータを参照するためには，どの国に関してはどのCD-ROMを参照するべきかを把握しておかなければならない。また，ひとまとまりのデータが数枚のCD-ROMにわたるというのも気にかかる点である。

上記の点は，常に全件を含む最新のデータを，過去のデータとの整合性をとりながら，ワークステーション等に移行し，維持し，検索に利用することで解決できる。Beyond 20/20はPCでのCD-ROM検索にのみ対応しているため，構造的にワークステーションに移行したデータに対して機能しないと思われるので検索システムは自製せざるを得ない。

これを可能にするには，ハードディスクに保存するデータが一般的な形式であること，そのフォーマット仕様が公開されていることが最低限必要であるが，OECD貿易ファイル及びBeyond 20/20はこれらの要件を満たしていない。

ただ，Beyond 20/20には検索結果をdBase[47]形式やCSV形式といった，一般的なフォーマットにデータ変換して，ファイルとして保存する機能があるため，

(47)　アシュトンテイト社が開発したパソコン用RDBMS。

これを使って，全件を使いやすいフォーマットに変換し，ファイルに保存することとした。そしてこのデータファイルを検索するプログラムを独力で開発することにしたわけである。

問題は，しかしながら，レポート国ごとの全データを変換し，ファイルに出力するには高速のデスクトップパソコンでも相当の時間[48]がかかるうえに，この作業をレポート国分，30回も繰り返し行なわなければならないことである。

この方法で作り上げた二次処理可能なデータは，一国あたり70MB程度，全レポート国合計で2GBを超えるサイズになり，当然，処理に必要な作業領域もギガバイト単位のものとなった。現在のワークステーションとディスクの速度では，ギガバイト単位のデータ処理には常に1時間単位の時間を要する。

CD-ROMで提供されるデータが，MTの時のように，フォーマットが公開され標準的な手順で読めるなら，Beyond 20/20を使わずとも，CD-ROMから直接データを読むことができるので，相当楽になる。しかし，Beyond 20/20でのみ読めるようCD-ROMに保存されたデータを読み取るプログラムを独自に作成することは非常に困難である。趣味的にトライすることは出来ても，ソフトが変われば同じ努力を2度3度としなければならない。

我々は，そこでCD-ROMにMTのときのように一般的で単純な形式で保存したものを提供してもらえないか，提供する予定はないかということをOECDに問い合わせたが，そのような予定はないとの回答が返ってきたのみである。

3.7.2 CD-ROM化に際して望まれること

以上述べたOECD貿易ファイルのCD-ROMで起こった問題は決して特別の場合ではない。日本経済新聞社が提供するCD-ROMデータでも同様の状況・

(48) Pentium 200MHz/64MB/12倍速CD-ROM/Windows95のマシンで，約2時間を要する。

対応である。ただし日本経済新聞社はCD-ROMと並行にテープ[49]によるデータ提供もあるので，我々はCD-ROMの利便性より，テープの汎用性を選択し，運用している。しかしテープでの販売量は限られてしまう。企業としては，CD-ROMを大量に販売する方が，利益につながるであろうから，今後もこの傾向は続くであろう。

データが新しいメディアで安価に提供され，ユーザーからみてコンピュータのスペシャリストやサービス機関に頼らずとも使えるというのは，確かに望ましいことである。オープンリール・テープは，そのドライブ装置自体も非常に高価なもので，物理的に大きなサイズに比して，扱える容量の小ささ，また保存管理の面倒さなどから，現在では，大変扱いにくいメディアとなっている。これに比較して，CD-ROMは，今やほとんどのコンピュータで標準に読むことができ，コンパクトな大きさ，長期間保存がきくこと，複製が容易なことから，ユーザーから見ても，データ提供機関から見ても，魅力のあるメディアなのである。

このように考えると，販売対象がそれまでの研究機関から研究者個人へと変化し，それに呼応するように，データ提供が大量のデータと，それを検索するソフトをつけたCD-ROMとなり，それ自体が完結したシステムとなった。

その結果が，特定のアプリケーションのみで読めるデータの提供なのである。このことで，データは欲しいものの，添付される検索ソフトでは機能的に満足できないユーザーのフラストレーションは高まることとなった。

新しい利用者層のために変化することは重要であり歓迎したい。しかし，従来のオープンでよりシンプルな仕様でのデータ提供がなくなることは，データ処理の可能性そのものを小さくしてしまうことを意味し，まるで既製服のみが市場に出回り，テイラーメイドの服を着用できない社会の到来のような，面白みのないデータ利用環境になるのではなかろうか。

(49)　多くのデータがオープンリール型と8mmテープの両方で用意されている。

CD-ROM化してデータを提供する際には，オリジナルデータを扱いやすくオープンな仕様のもとに提供する手段が用意されることを望みたい。

この点は，第6章でまた論じることにする。(50)

(50) 極端な例だが，従来から磁気テープで提供していたフォーマットのまま，テープに記録して提供する代わりにCD-ROMに書いて提供して欲しいと幾つかのデータ販売会社に提案したが，現在のところ受け入れられていない。

第4章　RDBによるデータベース・エンジン

RIEBデータベースは，一言でいえば，最近のネットワーク環境に対応して，PCクライアントなどからWWWページの検索画面で必要な検索条件を指定することにより，抽出結果をメールやネットワークディスクなどで受け取るシステムである（18ページの**2.3.1**を参照のこと）。RIEBデータベースシステムの開発には，2つの側面がある。一つは，データを標準化し，データベースファイルとして保存することであり，これは前章で議論した。もう一つは，WWWページを経由してユーザーから検索条件を受け取り，データベースから検索条件に合ったデータを抽出する検索エンジンの設計・チューンアップである。RIEBデータベースシステムでは，このエンジンの一部に，商用のRDB管理システムであるOracleを採用した。

4.1　経済・経営データとRDB

経済・経営データを扱うためのデータベースエンジンとして，RDBシステムは適している。その最大の理由は，そのデータ構造にある。経済・経営データは一般に二次元の表で置き換えることができる場合が多い。例えば，変数名を列にその時系列データを行に配置することを考えればこのことは容易に分かる。RDBはまさに二次元の表に対して，論理演算（和，積など）を行なうものとなっている。

また，RDBは現在実用的なデータベースシステムとして最も普及している。RDBにはパソコン用の比較的小さなMicrosoft Access[51]のようなソフトから，

(51) Microsoft社製のRDBシステム。単独でもデータベースシステムとして機能するが，ネットワーク越しにOracleなどの商用RDBとも連係できる。

Oracle のように商用で，非常に規模の大きいものまである。例えば，Microsoft Access は，個人的な文献管理，住所管理などに使われている。逆に Oracle などによる大規模な実用例としては，全国チェーンの販売店舗の流通管理システムのような巨大なものがある。

このように RDB はスケーラビリティに極めて優れたデータベースシステムである。RDB はまた，二次元表によるデータ管理と SQL という一般的な言語による操作ができるという点で共通性があり，ポータビリティも非常に優れている。スケーラビリティとポータビリティの二点は，**2.3** で示したとおり，データベースエンジンとして要求度の高いものであり，RDB システムは幸いこの両者を備えているのである。

4.2 Oracle の特徴

Oracle は商用 RDB の市場では大きなシェアを持つ製品である。多様なプラットフォーム，つまり，パソコンから，ワークステーション，大型汎用機までに対応していて，実際に稼動している。RIEB データベースシステムのエンジンとしては，Oracle は，スケーラビリティとポータビリティの条件を十分満たしており，推奨できるシステムの一つである。

ことに，各種のミドルウェアと呼ばれるサポートソフトウェアの数が豊富なことも Oracle が歓迎される要因で，例えば，Oracle WebServer[52] は，かなり複雑な業務処理を WWW インタフェイスに対応して可能にする各種のツールを提供してくれる。Microsoft Access，PowerBuilder[53] などいくつかの製品は，ネットワーク経由で Oracle RDB を利用する機能を持っている。こうしたツー

(52) Oracle RDB エンジンとの連係を強めた Oracle 社製 WWW サーバー。
(53) Power Soft 社が開発したデータベースアプリケーション開発ソフト。Windows 上でデータベースを利用する業務アプリケーションの開発環境を提供する。

ル群はシステム開発の自由度を大きくし，作業を容易にする。

こうしたことから，研究所RDBエンジンとしてOracleを選択したのである。

4.3　Oracleのパフォーマンス

Oracleは，非常に強力なソフトであるが，その扱いは困難を極める。チューンアップしないと，検索速度は思ったように出ない。RIEBデータベースを，Oracleで検索させたときのパフォーマンスを現状で計測してみた。

この実験に用いた設備は日立のワークステーションで，そのスペックは表4の通りである。

表4：研究所Oracleサーバのスペック

ハードウェア	HP9000/725	PA7100LC 100MHz (100.1 SPECint92, 137 SPECfp92) / 256MB RAM / 30GB RAID5 disk array
OS	HPUX-10.20	
RDBMS	Oracle 7.3	
データ	OECD ITCS	一カ国あたり137万件，合計28カ国，3800万件

収録データは表5のような簡単な形式とした。

データはレポート国一ケ国あたり137万件，DBデータ標準化済みのテキストデータで130MBに達し，全レポート国データは3800万件，同3.7GBにもなる。これをOracleにロードすると，2.1GB程度と少なくなる。[54] repcode,

(54)　テーブル作成時の充填効率指定は90%とした場合。

parcode, ie, sitcの各キー項目には別個にインデックスを張るが，このインデックスは1GBを大きく超えるディスクスペースを使用する。

この条件下で，パフォーマンスの目安になりやすい代表的な処理に要する時間を表6に示す。これらの処理は件数カウントと抽出であるが，抽出には項目データを読んでディスクへ書く処理が加わるため，時間が余分にかかっている。また，一般にデータベースの処理速度計測にはTPS(Transactions Per Second)などが利用されるが，ここではSQLPlus[55]アプリケーションにおけるコマンドあたりの処理時間，すなわちレスポンスタイムを秒単位で示した。我々のデータベースシステムは繰り返し行なわれる小規模なトランザクション[56]の処理速度を追求しておらず，時折行なわれる大規模な検索のレスポンスタイムだけが問題となる使い方をしているためである。この学術的利用におけるアクセスの傾向と特徴については**4.4**に述べる。

表5：Oracle実験用テーブル仕様

項目名	属性	
repcode	char(3)	レポート国コード
parcode	char(3)	パートナー国コード
ie	char(1)	輸出・輸入フラグ
sitc	varchar(5)	SITCコード最大5桁による業種分類
v1988	number(13,1)	1988年度の値
	⋮	
v1995	number(13,1)	1995年まで同上(8項目)

(55) Oracleに標準で添付されている問い合わせユーティリティ。Oracleが扱える全てのSQL文をコマンド形式で実行できる。

(56) データベース処理で行なわれる一まとまりの参照，更新手続きを指す。

表6：Oracleの処理能力

	データ量	一カ国 28カ国		137万件（DB標準化テキストデータで130MB） 3800万件（28カ国，同3.7GB）
	Unixコマンド			Unixコマンドによるファイルアクセス速度
1.	cp	一カ国	90秒	733KB/sec，130MBのファイル読み書き
2.	grep -c	一カ国	56秒	1.1MB/sec，130MBのテキストファイルを読む
3.	wc -l	一カ国	204秒	620KB/sec，130MBのテキストファイルを読む
	Oracleユーティリティ			SQLによるデータベースへのアクセス速度
4.	SQLLoader	一カ国	5分	130MBのテキストファイルを読み込み，75MB程度の領域に書き込む
5.	SQLLoader	28カ国	140分	3.7GBのテキストファイルを読み込み，2.1GB程度の領域に書き込む
	Indexなし			
6.	count( )	一カ国	10秒	137万件分の件数カウント
7.	count( ) repcode='JPN'	一カ国	23秒	137万件分の件数カウント
8.	count( ) parcode='JPN'	一カ国	21秒	137万件中5000件分の件数カウント
9.	count( )	28カ国	454秒	3800万件分の件数カウント
	Indexあり（137万件対象）			（137万件のIndex作成には3分かかる）
10.	count( ) repcode='JPN'	一カ国	11秒	全件の件数カウント
11.	count( ) parcode='JPN'	一カ国	1秒以下	5000件分の件数カウント
12.	sum (v) parcode='JPN'	一カ国	2秒	5000件分の数値項目の合計
13.	select*parcode='JPN'	一カ国	2秒	5000件を抜きだし
14.	select*sitc='0240'	一カ国	3秒	60件を抜きだし
	Indexあり（3800万件対象）			（3800万件のIndex作成には2時間かかる）
15.	count( ) repcode='JPN'	28カ国	12秒	137万件をカウント
16.	count( ) parcode='JPN'	28カ国	1秒	15万件をカウント
17.	sum( ) parcode='JPN'	28カ国	19秒	15万件の数値項目の合計
18.	select*parcode='JPN'	28カ国	70秒	15万件を抜きだし
19.	select*sitc='0240'	28カ国	125秒	16000件を抜きだし
20.	select*parcode='JPN' and sitc='0240'	28カ国	1秒以下	60件を抜きだし

この実験は，Oracle の一部の機能の能力を確かめただけではあったが，Oracle データベースエンジンとしての高速性がよく発揮されている。これだけのパフォーマンスを自製のプログラムで得ることは非常に困難である。

実験結果をまとめると次のようになろう。

- データをデータベースにロードするのにかかる時間は，普通にファイルをコピーする時間の数倍にすぎない（事例 **1.4.5.**）。
- インデックスを利用しない全件の行数カウントでも Unix コマンド（grep, wc）を利用するより数倍高速である（事例 **2.3.6.**）。
- インデックスは膨大なデータの中から僅かのものを検索する時に極めて有効であるが，全件が対象となるような場合では効率的でない（事例 **6.7.10.11.**）。
- 逆にインデックス付き検索において，該当件数が同じであれば，母数となるデータが増えても検索時間はほとんど変わらない（事例 **10.15.**）。
- 全体の 5 %以下程度の抽出を行なう場合では，インデックスを付けることによって20～100倍近く高速化できることもある（事例 **8.11.16.**）。
- インデックス付きでの検索は絞り込むほど高速化が顕著になる。絞り込んだ処理は，全3,800万件を対象にしても非常に高速である（事例**11.16.20.**）。

データベースにおいてインデックス検索は特に重要な高速化手法であり，効率的検索システム構築のため，Oracle を用いた OECD 貿易ファイルによるデータベースについては，すべてのキー項目（repcode, parcode, sitc, ie）をインデックス化した。

特に注目すべきは事例20である。これはパートナー国コードと SITC コードの指定によって，充分に絞りこんだ検索が行なわれた場合である。このような検索が，ユーザーが恐らくもっとも頻繁に行う問い合わせであろう。例えば，

レポート国を3ヶ国，パートナー国を7ヶ国，SITCを5つ，指定し，全3,800万件からの抽出を実行するとしよう。この処理に要する時間は1秒から4秒程度であり，120件の出力が得られる。このパフォーマンスは，全件数の膨大さを考えれば，驚くほど速い。サーバーの能力が上がれば，パフォーマンスも上がると考えられるから，今後のデータの増加を考慮に入れても，十分，実用的である。

4.3.1　47産業分類による試行

119ページ**7.3.4**に示している47産業分類に絞った33万件を対象に，様々な抽出を試みた。Oracleの能力を試す意味もあり，全3800万件での検索と比較して，各検索・抽出のスピードがどう変化するのかを見定めることにより，今後のRIEBデータベースをどのように構築していくかの参考になる。表7にその代表的な検索例をあげ，検索条件に合致するデータの件数を調べるために要した時間と，抽出に要した時間を示しておこう。

結果は，一般に，条件設定の複雑さは検索にかかる時間とは余り関係なく，むしろ出力行数に比例して処理時間が増えていることが分かる。例えば50件だけヒットするような条件を指定すると，常に1秒以内で終る。大雑把に言えば，抽出の場合で1万件4秒程度になるだろう。

比較のためにSITC5桁のデータ3,800万件から数十件だけヒットするように条件を設定すると1秒で終る（53ページ，表6の事例20.）。処理に必要な時間は主としてデータの出力時間であり，これは出力件数に比例している。インデックス化が高速化に貢献していることが前節の表6から推測できる。

OECD貿易ファイルのように，一般に数種のキーコードだけで絞って検索するようなデータであれば，Oracleのスピードは非常に速く，億単位のデータ系列があるとしても数秒以内に検索できると考えられる。

4.4 商用RDBの学術的利用における諸問題

RDBソフトウェアには商品化されているものが多い。商用のRDBは，しかしながら，経済・経営データのデータベース管理を目的としては開発されていない。もっと複雑なシステム，利用法を守備範囲としているのである。RIEBデータベースと商用RDBについて，その目的とするところの相違，問題点などを表8に列挙しておこう。

RIEBデータベースに商用ソフトが使えるとしても，上記に示したとおり，目的，使用法が商用ソフトの想定しているものとは異なっている。それゆえ，RIEBデータベースでは運用上の問題がいくつか生じる結果となった。以下はその詳述である。

表7：47産業分類テーブルでの検索例

検索条件	対象件数	カウント時間	抽出時間
a. パートナーが韓国のみ parcode='KOR'	2400件	1秒以内	1秒以内
b. レポートが韓国のみ repcode='KOR'	10000件	1秒以内	4秒
c. レポート，パートナーのいずれかに韓国 repcode='KOR'or parcode='KOR'	12000件	3秒	5秒
d. パートナーが三国のいずれか parcode=any ('KOR','TWN','NZL')	7000件	1秒以内	4秒
e. レポートが三国のいずれか repcode=any('KOR','TWN','NZL')	30000件	1秒	9秒
f. レポートとパートナーが，それぞれ別の三国のいずれか repcode=any('KOR','TWN','CHN') and parcode=any ('USA','FRA','NZL')	900件	1秒以内	1秒
g. レポートとパートナーのどちらかが別の三国のいずれか repcode=any ('KOR','TWN','CHN') or parcode=any ('USA','FRA','NZL')	40000件	7秒	15秒
h. g.から更に業種を三つに絞る (repcode=any ('KOR','TWN','CHN') or parcode=any('USA','FRA','NZL')) and class=any ('01','02','03')	2600件	1秒以内	1秒

表8：商用利用と学術的利用の相違

	商用アプリケーション	経済・経営データ処理
トランザクション	小規模のものを同時に多数，連続的に	ほぼRead Onlyで，大規模なものを不連続に
更新	検索と同時並行に少しずつ	月に一度，年に一度だけ，ほぼ全件書き換え
障害復旧	高速な復旧と，障害時点からの中断されたトランザクションの再現を望む	それほど急がず，中断されたトランザクションは放棄しても良い
アクセスのパターン	定期的，定型的で，固定の経路から	不定期，非定型で，さまざまな経路から
処理内容	複雑なロジックを含む	ロジックを含まない場合が多い

4.4.1　チューニングの相違

商用RDBシステムは一般に，チューニングと呼ばれる処置を施すことによって，システムの動作を最適化し，データベースシステム全体の性能を向上させることができる。つまりデータの量，規模，データの性質などに応じて，システム全体を最適化させる必要があることを意味している。たいていの場合，商用RDBシステムは初期設定として小規模な商業利用向けにチューニングされた状態で，出荷されている。こうした初期設定が，研究利用に適さないのは自明であろう。

我々は，RIEBデータベースの処理効率を上げるため，Oracleに初期設定とは異なるチューニングを施した。

主として以下のような点についての対処が必要であった。

1．扱うデータ量が非常に大きなものになるため，性能が多少犠牲になる可能性もあるが，ディスクアレイ[57]を積極的に利用することとした。Oracle

(57) Disk Array，複数のディスクを組み合わせて大容量の一本のディスクに見せかける技術。高速化技術でもあるが，利用形態によっては単体ディスクより性能が落ちる可能性もある。

には表領域を意識的に複数のディスクに分散配置したデータファイルで構成して高速化する方法もある。しかし非定型な利用が多く，データ総量の変化が激しい学術的利用でこれを利用するには管理コストが掛かり過ぎるので採用は見送った。

2．各種のバッファは大きめに確保する。

3．プロセス数など，多重度に関わるパラメタは大きめにする必要はない。

4．扱うテーブルがRead Only になる傾向が強いので，テーブルの充填効率は高く設定する。

5．扱うテーブルはまた大きくなる傾向もあるので，各種の表領域はExtents[58]も含めて大きく設定する。

6．巨大なテーブルのレコードを更新したり削除したりするときには非常に大きな一時領域が必要となる。しかし更新・削除は商用データベースのように頻繁に行うわけではないので，一時領域とユーザー領域は同一の表領域にとる。

7．同様に巨大なテーブルのレコードを更新・削除する際には，大きなロールバックセグメント[59]を要求する。これも常時使うことはなく，同時に複数の処理を行なうこともないため，この領域は数を少なく，それぞれを大きくとる。

8．クラッシュ・リカバリーは重視しないため，ログは小さくとって自動的に上書きするように設定する。事実上のログなし運用である。また，可能な限り各SQL コマンド処理にはUNRECOVERABLE を設定し，ログに出力する無駄を省く。

9．ログ領域，ロールバックセグメントなど，更新，削除の時に発生する大

(58) データ量の増加に応じてファイルサイズを拡張する時の単位量。10MB の表領域の Extents を 1MB としておけば，10MB で不足した時には 1MB ずつ拡張される。

(59) 更新系のトランザクションが中断された時に，更新前の状態に復元するために必要な情報を記録する領域。Oracle では普通ディスク上に確保される。

量のログ出力が全体の処理時間を長引かせる原因となりがちなので，これらのファイルはデータを保存するディスクアレイとは別のディスクに書かせるようにして，同じディスクに対するアクセス競合を防ぐ。

これらのチューニングはまったく試行錯誤によってなされたもので，我々の目的にぴったりのマニュアルや指針があったりするわけではない。もっともチューニングについては断片的な情報を与えてくれるマニュアルはあるが，痒いところに手が届くような虎の巻のようなものがない。試行錯誤ながらこれまでのところ比較的うまくチューンアップができている。

4.4.2　インターネットからのアクセス

商用RDBは一般にクローズドシステムで運用される。

つまりインターネットを介して，不特定多数の利用者にサービスを出すことを想定していない。商用データベースによっては，その利用ライセンス料が利用者数に比例して高くなる場合があり，この意味からも，データベースそのものの著作権の問題は別としても，不特定多数の利用者にサービスを提供するような運用は難しい。

それに付け加えて，ネットワークを介してデータベースにアクセスする場合は，そのアクセス制限，電子メイルやWWWとの連係などにおいて，他のネットワークサービスと利用者情報を共有することが重要になってくる。例えば，ネットワークサービスを利用するためのユーザー名，パスワードと，データベースを利用するためのユーザー名，パスワードは，同一のものであることが望ましい。しかしOracleを含めて，現在のデータベースシステムの，ユーザー管理システムは，そのOSのユーザー管理システムと全く同期していない。これではデータベースシステムを，ネットワークサービスに密に連係したシステムとすることができない。

商用データベースシステムは，閉じた企業内データ処理のために利用されることが多かったために，このような仕様で問題なく動作したのだと思われる。逆に既存のコンピュータシステムのOSと利用者情報を共有したのでは，細かいアクセス管理が実現できなかったために，OSが提供する利用者管理機能を用いずに，独立した機構を構築したのであろう。実際，Oracleにおいてユーザーとデータに対して設定できるアクセス条件は，Unixが本来持っているアクセス権管理システムより，はるかに細かな設定が可能である。

オープン環境（複数のコンピュータがもつ，公開された規格に基づく機能を組み合わせて構築するようなコンピュータ環境）における，使い勝手の良いデータベースシステムを構築するためには，このデータベースシステムの閉鎖性（または独自性）が問題となる。オープン環境に馴染まないのである。

今後のデータベースシステムは閉じたものではなく，オープンな情報提供にも広く用いられるだろう。Microsoft Accessなど，エンドユーザー向け製品のユーザーインタフェイスの発達により，全てのコンピュータユーザーがデータベースユーザーとなる可能性もある。そのためにも他の既存のコンピュータシステムやOSとの融合が進むことを望む。

4.4.3 クラッシュ・リカバリーとバックアップ

データの一貫性を維持することはデータベースシステムの主要な機能の一つである。商用RDBシステムは，このデータの保全と，そのために必要となるトランザクションの保全を非常に重視している。すなわち，停電や機器の故障による，不意のシステムダウンがあったとしても，データは必ず正しい状態に復旧できなければならず，更新途中のデータがあったとしても，途中で停止するのではなく，必ず更新前の状態か，更新後の状態に完全に復旧できる保証がなければならない。そうすることによって，大切なデータの整合性を保ち続けているのである。

そのため，一般に商用データベースでは，厳重なログ管理や，木目細かくスケジュールが組めるバックアップ機能を用意している。こうした処理のために，システム資源を消費し，検索や更新の性能が多少犠牲になるのは明らかであるが，商用データベースシステムでは，安全で頑強なデータベースであることが優先される。それに比べ学術的利用の場合は，商用利用に見られるようなトランザクションの保全がそこまで重要でない場合が多い。それは**4.4**で示したように，検索・更新におけるアクセスの性質が大きく異なるためである。例えば顧客データのように，商用では一レコード単位で毎日多くのクライアントから入力・更新するのが普通であるが，学術的利用では，更新するときは決まったクライアントから毎月一度だけ多量に行うということをする。そのため，学術的利用向けには思い切ってクラッシュリカバリーの機能や，厳重なバックアップの機能は利用しない方が，検索・更新処理の効率が向上する場合がある。現状のRDBシステムはそのような運用を想定していない。そのため，RIEBデータベースに対応できるようOracleをチューンアップできないところもあった。例えば，Oracleではログを記録する設定を解除することができない。その結果，不要であるにもかかわらず，異なるディスクにそのログを出力するという設定をせざるを得なかった。これは明らかに資源の浪費である。

データベースシステムが今後ますます普及するにつれて，学術的利用のみならず，現状では想定していないようなアクセスをする分野での利用が起こってこよう。この多様なアクセス要求に応えられるよう，運用上の設定について，Oracleなどの商用データベースソフトにはより柔軟性を高めることが望まれる。

4.4.4　商用RDBシステムが学術的利用で普及しない理由

商用RDBシステムの学術的利用が広く普及していれば，前述のチューニングに関する問題は，メーカーやユーザーの努力によって，その多くが解決して

いるはずである。チューニングは多く経験によって最適化されるものであるが，そのための充分な事例，実績が出ているだろうからである。

現実には商用RDBシステムの学術的利用はまだまだである。その理由として次を指摘することができる。

1．高価なこと

一般の商用RDBシステムは大規模なデータを扱う企業で使われている。業務を安全に遂行するために2重3重のテストが施されたシステムである。システムの規模は大きく，そのデータベース利用が，企業の業績の浮沈にかかわるので，価格の高さより性能の高さ，安全さを求める。それゆえ，製品価格は一般に高い。

逆に，学術機関ではデータベース，特にソフトウェアだけに予算を多く確保することが難しい。多くの商用RDBソフトウェアはインストールするマシンの性能によって価格が異なっており，より高性能の機種に変更する場合には，コストが相当かかると思って間違いがない。

2．保守を要すること

ソフトウェアには一般にアップデートがつきものであり，商用RDBとて同じである。バグフィックスやトラブルサポートなどは結構頻繁にあり，保守なしで運用するのには危険が伴う(60)。バージョンアップについても，これをしないで利用することもできなくはないが，現状ではコンピュータのOSなどのバージョンアップが比較的頻繁にあるので，RDBソフトを新しいOSで動作させるために，RDBソフトをアップデートする必要がどうしても生じる。この保守の費用も一般的にかなり高額で，学

(60) 逆に商用RDBがもっと普及して多くの事例が出てくれば，ユーザー間で情報を共有するなどして保守無しでの運用を試みることも出来るが，現在の状況ではそのようなことは望めない。

術機関での利用は限られたものになりがちである。

3．ドキュメントの未整備

商用RDBは普通，ソフトウェアだけを購入するのではなくて，システムとして購入する。顧客にあったシステムを構築した上で，納入するのであるから，顧客側としては完成したシステムを利用するだけである。学術機関では，ソフトウェアだけを購入してあとはすべて自前で運用する場合が多い。提供されるマニュアル等は，一般に，詳細ではあるが不親切で，専門的なデベロッパー向けのものとなっている。すなわち運用するためには，学術機関のデータベース管理者にソフトウェア・デベロッパーと並ぶ能力を要求することになり，現実的ではない。(61)

こうした点で，学術研究機関，特にコンピュータのサポート部門を持たないところでは，商用RDBの利用が困難になっているのではないかと思われる。経済経営研究所がOracleを採用することができたのは，保守費用を含めてレンタルで調達することができ，かつ所内に運用能力が備わっていたという幸運に恵まれていたためである。

4.5　大阪大学のデータベース外部委託開発の事例

データベースの構築は資金・人両面で余裕のある部署ではじめてできる作業である。資金はあるが，人がいない場合，ベストのオプションは外部委託をすることであろう。計算機設備の調達の際に，データベースシステムの開発，保守，運用を含めて外部機関に依頼し，商用RDBを用いたシステムを構築することも一つの選択肢なのである。これはアウトソーシングの一例である。

---

(61)　Oracleサーバーのマニュアルの厚さは合計で1mを楽に超える。管理者は最低でもこのマニュアル群の多くを繰り返し読むことになるだろう。

大阪大学大型計算機センターでは，そのような形で，Oracle RDB を用いた文献データベースを構築している。計算機設備の更改に伴い，それまで汎用機システム上で運用されていたデータベースを，Oracle を用いたシステムに移行することを含めて，計算機設備を調達し，運用している。安田は外注先の開発担当者および外注元である大阪大学大型計算機センターの担当者に聞きとり調査を行う機会をもった。その調査内容を今後の参考のために，以下にまとめておく。

大阪大学大型計算機センターでは，同一構造のシステム上に，BIOSIS,[62] CHEM-J,[63] TAXA,[64] の三つの文献データベースを構築している。これらは全て英文の文献データベースであり，タイトル，著者名の他にキーワード，要約などが含まれている。一レコードあたり約 1KB 程度のデータベースである。複数の単語による全文検索に似た利用ができる。

これらの文献データベースのなかで規模が最大の BIOSIS について，その概要を表 9 に紹介しておくことにしよう。大阪大学大型計算機センターでは，1980年以来 BIOSIS データを，米国 BIOSIS 社から毎年購入している。この1980年以降のデータをすべてデータベース化している。

表 9：BIOSIS データベースの概要

名称	BIOSIS
データ件数	880万件
項目数	20項目/件
データ容量	25GB
システム構成	CPU PA-8000 180MHz×2 / RAM 1GB / HD 100GB HP-UX 10.20 / Oracle 7.3.3

(62) BIOSciences Information Service, BIOSIS 社が作成，提供している，生物科学全般の文献情報データベース。蓄積データ数880万件。

(63) 日本で発行されている化学関係の定期刊行物を主な対象とした文献情報データベース。蓄積データ23.5万件。

(64) 種子植物の細胞遺伝学，細胞分類学などを対象とした，文献情報データベース。蓄積データ16000件。

開発はハードウェアを納入したメーカーの経験充分なプログラマ集団が行なった。開発に要した作業量（労働インプット）については明らかにされなかったが，最低でも能力を持ったスタッフ三人以上が開発に携わっており，半年程度の比較的短期間で開発を終了したという。

4.5.1　チューニングの問題

この事例でもやはりチューニングの問題に直面していた。

BIOSISデータそのものは年間70万件程度の量であるが，全文検索に近いキーワード検索をさせるために，新たに構築する，キーワードテーブルの大きさが，年間で3000万件にも達するという。

これが原因で，当初は検索速度が遅く，更新にも非常な時間を要していた。

すべてのデータをひとまとめに保存した場合，インデックスの大きさも非常に大きなものになることは避けられない。そこで，インデックスの大きさを抑え，検索速度を向上させるために，データを年ごとに分割して保存するようにした。つまり1980年以来の，17年間のデータを，17のテーブルとして保存しているのである。実際にユーザーが検索を行なう場合には，ユーザーには見えないが同じ検索条件が自動的に各テーブルに一度ずつ，合計すると17回指定され，検索が行なわれている。その結果をあたかも一つの巨大なテーブルから検索したかのごとく，まとめて表示するという方法をとっている。このため，ユーザーからみれば，テーブルが17に分割されていることを意識することはない。

当初は，それでも一件の検索・抽出に，140秒程度を要していたという。原因は大きくなったキーワードテーブルと，それらのインデックスがディスク上でフラグメンテーション(65)を起こしていたためである。これを解消するために，更新時には既存のインデックスを破棄し，更新後に全てのインデックスを作り

(65)　Fragmentation，ディスクなどにデータが書き込まれる時に，そのデータが連続した領域に収まらなくなる現象。

直すスタイルにした。これによって一回の検索で1000レコード以下のマッチであれば，1秒から3秒程度とレスポンスタイムが劇的に向上した。

すべてがうまく行ったわけではなく，そのかわり全インデックスの更新作業には非常な時間を要することになった。更新は月に一度行われる。変化のあったレコードだけの提供を受け，これを既存のデータベースにロードし，インデックスを再構築する。データの少ない年のテーブルの更新に2時間，長い年の場合には14時間以上を要するという。前回の全年インデックス再構築には40時間かかった。これは学術系データの更新スタイルに，Oracle がうまく適合しなかった典型的な事例といえよう。

第5章　WebDBの開発

RIEBデータベースは，27ページの図3などで例示したように，WWWインタフェイスや各種のネットワークサービスと密接に連係したシステムである。このシステム全体をWebDBと呼ぼう。WebDBは，比較的小さなC言語で書かれたプログラムとUnixコマンド群を，シェルスクリプト(66)で組み合わせて構築されている。我々はこの開発にあたり，どのようなプラットフォーム上でも動作するように，一般的なプログラムをできるだけ利用し，利用できない場合はC言語などで新たに必要なプログラムを作成した。その結果，Unixシステムであればどこでも動作する可能性が高く，ポータビリティに優れたシステムが構築できた。

ユーザーはこれらのプログラムを一切意識することはない。なぜなら，エンドユーザーが接するのは，WWWを用いた検索インタフェイスだけで，そこでパラメタを入力することによって，間接的に各コマンドやプログラムを呼びだし，その結果だけがユーザーの手元に届くという仕組みになっているからである。

WebDBは，そのデータベース・エンジン部分（図3参照）に，独自に開発した二種類の検索コマンドを用意している。一つはOracle RDBを利用するoraselコマンドであり，あとの一つはGNU grepなどを利用するdbselコマンドである。ここでは，普段ユーザーが意識しない内部のプログラムの呼びだしがどのように行なわれているかを焦点に，説明を試みる。なお実際のプログラム等は第8章を参照されたい。

---

(66) Unixにおけるコマンドインタプリタであるshellによる簡易なプログラム記述。

5.1 orasel の概要

orasel は，WWW インタフェイスから与えられる典型的なデータベース検索の問い合わせを，それと等価なはたらきをする SQL による問い合わせに変換して，Oracle データベースに実行させることを目的に開発した。Oracle RDB にアクセスするにはさまざまな方法があるが，orasel は最もシンプルな Oracle のSQL 問い合わせユーティリティである，SQLPlus を利用する。図 5 に orasel を用いた場合の検索処理の流れについて示す。27ページの図 3 と比較すれば，その構成が理解しやすくなるであろう。

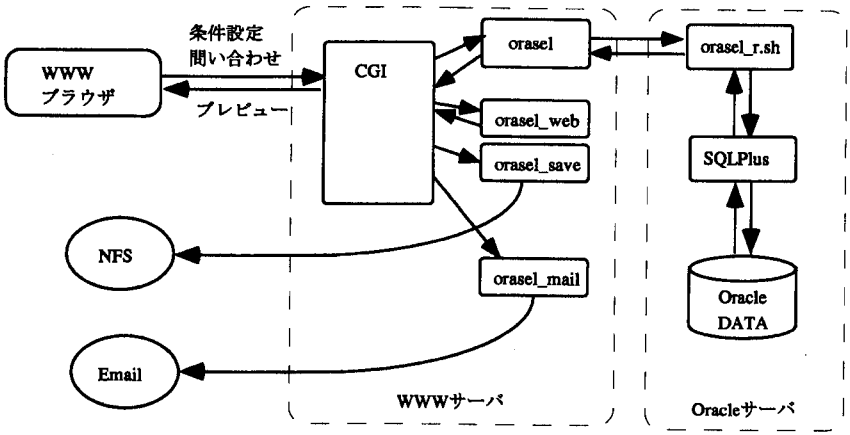

図 5 : orasel を用いた場合の検索処理の流れ

図 5 に示したように，1998年時点での研究所は，二台の Unix ワークステーションを用いて，WWW サーバーと Oracle データベースサーバーを独立させたシステム構成としている。そのため，orasel は WWW サーバー上で起動され，そこからネットワーク経由で Oracle データベースサーバー上の SQLPlus コマンドを起動することになる。SQLPlus による検索結果は，TAB セパレー

ト[67]されたテキスト形式のファイルとして，WWWサーバーに返される。これにWWWサーバー上で後処理を加え，表示・転送・ディスク出力などの方法で，エンドユーザーに提供する。

具体的なWWWからの検索処理の流れは以下のようなものになる。

1．HTML形式のWWWページを閲覧し，条件指定などを表示に従って行なう。検索実行ボタンを押すとCGIプログラムが起動する。
2．CGIプログラムが，与えられた入力値から，oraselプログラムに有効なオプション列を作成して，oraselプログラムを起動する。
3．oraselプログラムは与えられたオプションに対応するSQL記述を合成し，orasel-r.shプログラムを起動して，ネットワーク経由でSQLPlusに所定のSQL記述を引き渡す。SQLPlusは指定されたテーブルから設定された条件に該当する情報を切り出し，結果はTABセパレートのファイルとしてoraselプログラムに返す。
4．oraselプログラムは，呼び出し元であるCGIプログラムに結果を返し，CGIプログラムは結果に必要な後処理を加え，指定の形式で出力する。具体的には，ユーザーの指定によりWWW上に表示，メイルで結果送信，ディスクに出力のいずれかの方法を選び，それに適合する後処理プログラム(orasel_web, orasel_mail, orasel_save)をオプション付きで起動する。

5.2　oraselのインプリメンテーション

5.2.1　データ仕様

(67)　TAB記号と改行文字によって，項目とレコードを区切る形式。簡易なデータ交換用フォーマットとして用いられる。

orasel は以下の二種のデータを使用する。

・項目定義ファイル

Oracle RDB として保存されているデータの項目名，値の型，項目見出しを定義する。

一行目に項目名（英数字），二行目に値の型（9：数字，X：英数字，N：英数字及び漢字，それぞれ括弧内に桁数），三行目に項目見出し（英数字及び漢字）を記述する。

・データ

定義ファイルに定められた形式のデータ。Oracle RDB 上に，一つのテーブルとして保存されている。

データは Oracle 上のテーブル（表）であり，複数の項目を持つレコードが複数行並ぶ形態である。項目の値の長さは，あらかじめ仕様から得た最大長を定義してあり，桁あふれなどが起きないようにしてある。

具体例として，2 行 3 列の非常に小さなデータの場合を以下に示す。論理的なデータ構造が表形式であることをわかりやすくするために，これ以降のデータの例示は，全て以下のような表の形式で行なう。

定義ファイル（sample. def）

id	name	price
X(3)	N(10)	9(8)
ID	品名	金額

データテーブル（sample）

A34	電化製品 B	59000
A40	鉄・アルミ	120000

図 6：Oracle 上のデータの例

項目定義ファイルの物理的なファイルフォーマットは，項目(列)がTAB記号，行が改行文字によって区切られたテキストファイルである。データファイルはOracleテーブルであり，その物理的なデータ保存のフォーマットは非公開となっている。[68]

5.2.2　oraselコマンド仕様

oraselに与えることができるオプションを以下に示す。[ ] は必要なければ省略することが可能であることを示す。a | b はa bいずれか一方を指定することを示す。

orasel -h *header* -d *table* [-l *limit*] [-a | -o] [*cond...*] =*column...*

-h　項目定義ファイルには *header* を利用する

-d　データベース上のテーブル名は *table* である

-l　出力は最大 *limit* 行

-a　条件記述 *cond* が複数あった場合，すべての条件が満たされていれば(AND)真(デフォルト)

-o　条件記述 *cond* が複数あった場合，少なくとも一つの条件だけでも満たされていれば(OR)真

項目名 *column* には，項目定義ファイル *header* の一行目に書かれた項目名を指定する。これはテーブル *table* 上での項目名に一致する。

条件記述 *cond* のフォーマットは以下の通りである。

cond = *column mark value* (空白なしで詰めて入力)

column = 項目定義ファイル *header* の一行目に定義した項目名

(68)　プログラマはOracleデータの物理的な構成に頓着せずプログラムを書いてもいいようになっていて問題はない。

mark = '=', 'not =', '>', '<', '= any' のうちいずれか
それぞれ Equal, Not Equal, Larger than, Less than,
列挙した値のどれか一つが Equal[69]を意味する。
value = *column* の型に適合する値

cond と *column* の間にある「=」は必須であり，それより前の引数を *cond*, 後ろを *column* と解釈する。*cond* に含まれる等号としての「=」との区別は，前後に連続したパラメタが全くなく，単独の「=」一文字だけであることによって行なう。

先の例，sample.def と sample テーブルを用いた，orasel による検索例を以下に示す。出力結果はやはり TAB セパレートのテキストファイル形式であるが，以下の例示では見やすさのために内容を表形式で示している。

問い合わせ : orasel -h sample.def -d sample = name price

結果 :

品名	金額
電化製品 B	59000
鉄・アルミ	120000

図 7 : orasel による問い合わせと結果の例

問い合わせの例は sample テーブルから，sample.def 定義ファイルで定義されている項目 name, price のみを抜き出すというものである。その結果，元の sample テーブルの項目のうち二列だけが抜き出されている。また，結果の一行目には定義ファイルの三行目に定義された項目見出しが各項目ごとに付けられる。

条件設定の記述が，SQL に似た文法となっている。orasel の主な役割は

(69) 78ページに '=any' を用いた例があるので参照されたい。

CGIプログラムから与えられたオプション記述を，SQLによる記述に変換することである。そこで，oraselのオプション構造をSQLと同じようにしておくことによって，プログラムの作成がより簡単になる。[70] 逆にこうすることでoraselはSQLで表現できるほとんどの機能を持つことができる。以下にoraselによる記述とSQLによる記述の比較をする。上がorasel，下がSQLの例である。

・品名，金額項目だけを抜き出す。

```
% orasel -h sample.def -d sample=name price
SQL> select name price from sample
```

・品名，金額項目だけを抜き出す。条件として金額が100000より大きい。

```
% orasel -h sample.def -d sample "price>100000"=name price
SQL> select name price from sample where price>100000
```

このようにoraselとSQLは極めて近い表現となっていることがわかる。参考のために幾つか具体的な検索の記述例を示しておこう。

・orasel -h sample.def -d sample -l 100=id name price
全項目について，最初の100件だけ抽出する。

・orasel -h sample.def -d sample id='A34'=name price
id項目の値がA34である行の，項目name，priceだけ抽出する。

・orasel -h sample.def -d sample "price>100000"=id name price
全項目について，price項目の値が100000を超える行だけ抽出する。
'>'記号はUnixコマンドにおいてはリダイレクション指定と間違える恐れがあるため，引用符で囲っている。

・orasel -h sample.def -d sample id='A34' "price>100000"=id name

(70)　もちろんSQLと全く同じにすれば変更の必要がなくなるが，SQL自身それほど整然としたコマンド記述ではないため，そこまで合致させる価値はない。

price

全項目について，id 項目の値が A34 であり，かつ price 項目の値が100000を超える行だけ抽出する（上例のデータでは一件も該当しない）。

・orasel -h sample.def -d sample -o id='A34' "price>100000"=id name price

全項目について，id 項目の値が A34 であるか，もしくは price 項目の値が100000を超える行だけ抽出する。（上例のデータでは二件とも該当する）

・orasel -h sample.def -d sample "id=any('A34', 'A50')"=id name price

全項目について，id 項目の値が A34 もしくは A50 である行だけ抽出する（上例のデータでは一件が該当する）。

'( )' 記号は Unix コマンドにおいてはサブシェル指定と間違える恐れがあるため引用符で全体を囲っている。

表形式のデータについて，項目の選択や行の選択などが簡単な条件設定によって行なえることがわかるだろう。そしてそれらの記述は，プログラムで変換される SQL に非常に近い。

5.3 dbsel の概要

dbsel は，Oracle RDB による検索エンジンが実用になるまでのプロトタイプとして開発された。4.4.1 で示したように，Oracle RDB を学術的利用向けにチューンして実用レベルにまで煮詰めるのには時間がかかり，またその過程において，頻繁にデータベース全体の運用停止，再構築を迫られる。Oracle RDB エンジン以外の部分はそれとは関係なく作成可能であるため，Oracle のチューニングが完了するまで Oracle を用いずに開発が行なえるようにしてお

く方が開発効率が良い。これがプロトタイプを必要とした理由である。

ただ，Oracleは大規模なRDBを扱うのに適したシステムであって，小量のデータを扱う場合には必ずしも効率が良いとはいえない。こうした場合，暫定的プロトタイプであるdbselの方が小回りがきき，効率が良い場合が多くある。そこで，データの特性に応じてoraselと併用しているのが現状である。

dbselの開発目標は以下のようなものである。

・高速であること
・移植性が高いこと
・簡単な操作でデータを扱えること
・他の検索エンジンへの切り替えが容易であること

何よりもまず高速でなければならない。また，**2.3.3**で示したポータビリティを備えていなければならない。そして検索のための条件記述等はできるかぎり簡単であるべきである。

dbselは小量のデータを扱う場合には効率が良いが，データ量が増大した場合にはdbselをやめて，oraselなどの大量データを扱うのに適した検索コマンドに移行する必要がある。この移行をスムーズに果たすために，コマンドオプションの記述方法やデータ形式などには一般的なものを用いなければならない。

高速性と移植性については以下に説明する。

5.3.1　高速性・移植性

当初は高速性を実現するために，全てを標準的なC言語で記述して作成する予定であった。

しかし始めてみると，C言語で記述したプログラムのうち，ファイル読み出

しと検索機能に関する部分の速度が思うように上がらなかった。FSF[71]のGNU[72]ソフトウェア群の行検索プログラムgrepの方がより高速であり，どのようにプログラムを工夫してもGNU grepを超える性能を出すことはできなかった。

FSFには世界的にも飛び抜けて優秀なプログラマが集まっており，彼らが書くプログラムもまた極めて優秀である。研究所の一エンジニアが書くプログラムが，それを性能面で超えることが出来ないのはある意味必然である。作成中の検索プログラムには，更に検索のための多様な機能が追加されて，もっと低速になることがわかっていたため，はやい段階でC言語によるプログラム開発をあきらめて，代わりにGNUユーティリティの組み合わせによる検索エンジンを開発することとした。

つまり，dbselはGNU grep, gawkをはじめとする，複数の既成のUnixユーティリテイと，それらを組み合わせるためのシェルスクリプトで構成されている。

これらのコンポーネント選択は主として高速性のためであるが，結果としてdbselの移植性を高めることにもなった。すなわち標準的なUnixシステムであれば，多くのGNUユーティリティがすでに移植されており，それ以外のOSコマンドなどははじめからシステムが持っているものがほとんどだからである。

5.3.2 全体の動作

oraselの場合と異なり，1998年時点での研究所のシステム環境においては，dbselはすべてWWWサーバー上で処理される。WWWを経由した検索処理は以下の流れで行われる。

(71) Free Software Foundation, Richard Stallmanが主催する団体。多くのソフトウェアを無償で自由に使えるように開発し，配布している。

(72) GNU's Not UNIX, FSFが作るシステム，またはそのためのソフトウェアに冠される名前。

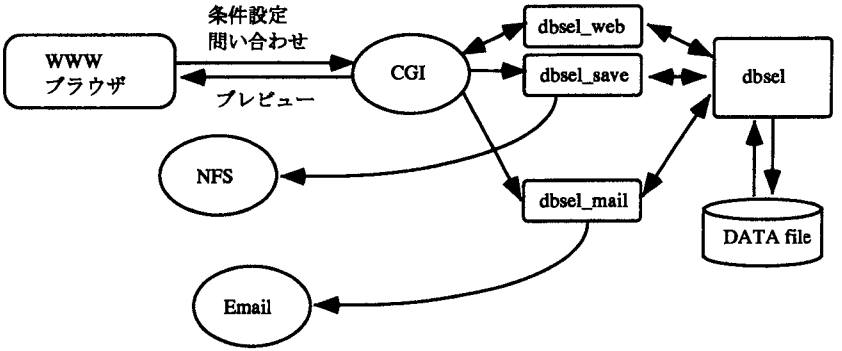

図8：dbselを用いた場合の検索処理

1．HTML形式のWWWページを閲覧し，条件指定などを表示に従って行なう。検索実行ボタンを押すとCGIプログラムが起動する。

2．CGIプログラムが，与えられた入力値から，有効なオプション列を作成。ユーザーの指定によってWWW上に表示，メイルで結果送信，ディスクに出力のいずれか一つを選び，それに適合するプログラム（dbsel_web, dbsel_mail, dbsel_save）にオプションを与えて起動する。

3．各検索プログラムは汎用の検索プログラム（dbsel）に所定のオプションを引き渡して起動する。指定されたデータファイルから設定された条件に該当する情報を抽出する。

4．日経総合経済ファイルのように時系列データを含む場合は，出力結果に含まれる時系列データ部分を，幾つかのプログラムを用いて加工する。

5．結果を受け取ったCGIプログラムは，必要な後処理を加えて2．で指定された形式で出力する。

5.4 dbselのインプリメンテーション

5.4.1 データファイル仕様

dbselは以下の二種のファイルを使用する。

・項目定義ファイル

データファイルに保存されているデータの項目名，値の型，項目見出しを定義する。

一行目に項目名(英数字)，二行目に値の型(9: 数字，X: 英数字，N: 英数字及び漢字，それぞれ括弧内に桁数)，三行目に項目見出し(英数字及び漢字)を記述する。

・データファイル

定義ファイルに定められた形式のデータを保存する。

データは表形式で，複数の項目を持つレコードが複数行並んでいる。各項目はTAB文字コードで区切られており，レコードは改行コードで区切られている。各項目の値の長さの制限は32,768バイトである。[73] 値の内容としてはTABコードと改行コードを除くASCII文字，EUC漢字[74]ならばどのようなものでもデータとして含めることができる。すなわち空白，記号などを含めても良いが，Vertical TABなどの制御コードは含めない方が，画面表示などの後処理においては無難であろう。数値項目は可視形式，すなわち10進数文字列として保存

(73) 内部処理のプログラムが用意している変数の長さが現在のバージョンではこれだけという意味で，必要であればプログラムを修正して伸ばすことが出来る。ただしプログラムの実行により多くのメモリを消費するので，システムへの負荷が大きくなる。

(74) 漢字コードにEUCを選択したのは，dbselが利用しているGNUユーティリティや標準Unixコマンドが EUC文字コードを処理するのに向いているためである。

する。

dbselにおいては，項目定義ファイルとデータファイルの両方とも，各項目がTAB記号，行が改行文字によって区切られたテキストファイルである。参考までに，oraselで示したのと同じ内容の2行3列の非常に小さなデータの場合の，物理的なファイル内容を以下に示しておこう。TAB記号を [tab]，改行文字を [ret] で表している。

定義ファイル(sample.def)

```
id[tab]name[tab]price[ret]
X(3)[tab]N(10)[tab]9(8)[ret]
ID[tab]品名[tab]金額[ret]
```

データファイル(sample.dat)

```
A34[tab]電化製品B[tab]59000[ret]
A40[tab]鉄・アルミ[tab]120000[ret]
```

図9：dbselのデータの例

5.4.2　dbselコマンド仕様

先述のデータファイルを検索するために，dbselに与えることができるオプションを以下に示す。[ ] は必要なければ省略することが可能であることを示す。a | bはa bいずれか一方を指定することを示す。

dbsel -h *header* -d *database* [-l *limit*] [-a | -o] [*cond...*] *column...*

-h　項目定義ファイルには *header* を利用する

-d　データベースファイルは *database* を利用する

-l　出力は多くとも *limit* 行に制限する

-a　条件記述 *cond* が複数あった場合，すべての条件が満たされていれば

(AND)真(デフォルト)

-o 条件記述 *cond* が複数あった場合，少なくとも一つの条件だけでも満たされていれば(OR)真

項目名 *column* には，項目定義ファイル *header* の一行目に書かれた項目名を指定する。

条件記述 *cond* のフォーマットは以下の通りである。

cond = *column*, *mark*, *value*

column = 項目定義ファイル header の一行目に定義した項目名

mark = '=', '!=', '>', '<', '~' のうちいずれか

それぞれ Equal, Not Equal, Larger than, Less than,

文字列の部分マッチを示す。

value = *column* の型に適合する値

先の例，sample.def と sample.dat ファイルを用いた，dbsel による検索例を以下に示す。出力結果はやはりTABセパレートのテキストファイル形式であるが，以下では見やすさのために内容を表の形式で示した。

問い合わせ：dbsel -h sample.def -d sample.dat name price

結果：

品名	金額
電化製品 B	59000
鉄・アルミ	120000

図 10: dbsel による問い合わせと結果の例

問い合わせの例は sample.dat データファイルから，sample.def 定義ファイルで定義されている項目 name, price のみを抜き出す，という内容である。その結果，元の sample.dat データの項目のうち，二列だけが抜き出されている。

また，結果の一行目には，定義ファイルの三行目に定義された項目見出しが各項目ごとに付けられる。

条件設定の記述はSQL，oraselに似た文法とした。例えば71ページの**5.2.2**で示したのと同様の，非常に簡単な検索例を以下に示す。

・品名，金額項目だけを抜き出す。

```
% dbsel -h sample.def -d sample.dat = name price
```

・品名，金額項目だけを抜き出す。条件として金額が100000より大きい。

```
% dbsel -h sample.def -d sample.dat "price, >, 100000" name price
```

この記述が，**5.2.2**で示したoraselによる検索記述と，SQLによる記述の両方に，かなり近い表現となっていることがわかる。

こうすることによって，プログラマが検索エンジンをdbselから他のもの，例えばoraselや，今後新しく現れるデータベースエンジンによるものに変更する際に，修正作業を軽くできるのである。

以下に具体的な検索の記述例を示す。

・dbsel -h sample.def -d sample.dat -l 100 id name price

全項目について，最初の100件だけ抽出する。

・dbsel -h sample.def -d sample.dat id, ==, A34 name price

id項目の値がA34である行の，項目name, priceだけ抽出する。

・dbsel -h sample.def -d sample.dat "price, >, 100000" id name price

全項目について，price項目の値が100000を超える行だけ抽出する。

'>'記号はUnixコマンドにおいてはリダイレクション指定と間違える恐れがあるため，引用符で囲っている。

・dbsel -h sample.def -d sample.dat id, ==, A34 "price, >, 100000" id name price

全項目について，id項目の値がA34であり，かつprice項目の値が100000を超える行だけ抽出する（上例のデータでは一件も該当しない）。

・dbsel -h sample.def -d sample.dat -o id, ==, A34 "price, >, 100000"
id name price
全項目について，id 項目の値が A34 であるか，もしくは price 項目の値が100000を超える行だけ抽出する（上例のデータでは二件とも該当する）。

表形式のデータについて，項目の選択，行の選択などが簡単な条件設定によって実現できることがわかるだろう。

5.5 時系列データの扱い

多くのデータには，項目に時系列データが含まれる。OECD 貿易ファイルの場合は，年次データで 9 年分ほどしかないので，普通に表形式のデータとして保存して扱うことに問題はない。具体的には，以下のような形式で OECD 貿易ファイルのデータは保存されている。

report	partner	I/E	SITC	1988	1989	1990	1991...
CAN	JPN	E	0011	1533.7	1786.0	1249.2	458.6...
CAN	JPN	E	00111	1448.5	1743.5	1249.2	458.6...
CAN	JPN	E	00141	1199.7	648.0	676.1	104.3...

図11: OECD 貿易ファイルのデータの例

この場合は時系列項目も，その他の項目と同じように扱えばよい。しかし日経総合経済ファイルなどでは，データの収録期間が長く，時系列項目をそのまま通常の項目として扱っていると，例えば40年間の月次データを保存するために480項目を用意しなければならなくなる。日経総合経済ファイルを処理しているのは dbsel であり，dbsel がこれを処理できるように設計することは不可能ではないが，多くの他のデータベースエンジンはこの構造に耐えられない。例えば Oracle は version 7 において254項目までしか扱えない。月次データに

してせいぜい20年程度までしか扱えないことになり，これでは実用的でない。[75]

dbselは多種のRDBシステムへの移行性を高く保つために，前処理，後処理も含めて極力他のデータベース・システムと共通の構造にしておく必要がある。つまり項目数を増やすことなく，長期間の時系列データを扱えるようにする工夫が必要なのである。

そこで時系列データは，一つの項目に長い文字列として保存することにした。例えば1991年から1995年までの時系列データを年次で持たせる場合，5年分の時系列データを並べて一つの項目とする。具体的に前述のデータに時系列データを含めた例を考えてみる。

id	name	dfrom	dnum	data
A34	電化製品B	1990	4	59000:59020:58900:58807
A40	鉄・アルミ	1992	5	120000:120004:120020:120032:12030

図12: 時系列データを含むデータの例

dfrom項目の値が1990というのは，時系列データが1990年から始まることを示す。dnum項目の値が4というのは，時系列データが4つ収納されていることを示す。続くdata項目のところに，4年分のデータをコロン(:)区切り文字によってまとめた一つの長い文字として保存している。

dfromの値は西暦1年から数えて何番目の時系列データであるかを示している。$dfrom = (year-1)*n+1$として求める。*year*は時系列データ開始西暦年，nは年次データの場合1，半期データは2，四半期データは4，月次データは12となる。年次データの場合，dfromの値は西暦年と一致して上の例のようになる。

こうして出来た長大な時系列データをdbselコマンドで選択する時は，単に

(75) Oracle verion 8でようやく1000項目まで扱えることになったが，まだまだ256項目程度までしか扱えないRDBは多い。

出力する項目名として data と一つ指定するだけで良い。これで保存された時系列データの全てが出力にそのまま現れる。もちろんユーザーはデータベースに保存された形の出力を望んでいるわけではなく，時系列データが個別の項目に分けられた以下のような形式を好むであろう。

ID	品名	1990	1991	1992	1993
A34	電化製品 B	59000	59020	58900	58807

図13: 時系列データが個別の項目に分けられた例

また，該当期間の部分だけを取り出す処理も必要である。そのため，これらの後処理を行なうプログラム trim を開発した。記述言語は C 言語である。

5.5.1 時系列データ抜き出しプログラム trim

時系列データを文字列としてパックした状態から，判読しやすい期種ごとの時系列にデータを区切って取り出すプログラムが trim である。これは C 言語で開発したプログラムで，その中身は第 8 章に示した。利用できるオプションを以下に示す。

trim [-s *skip*] [-e *col*] *start* [*limit*]

-s: データの先頭から *skip* 行を，見出しとみなして読み飛ばす。

-e: 時系列パック化データより左に，*col* 個の通常項目がある。

start: 取り出したい時系列データの開始位置。

limit: *start* から *limit* 個の項目までを取り出す。
指定なければ最後まで。

簡単な時系列データを例題としてとりあげ，それを trim コマンドによって処理し，文字列としてパックされた時系列データが，期種ごとに区分される事例を以下に示す。

データ：

A34	電化製品B	1990	4	59000:59020:58900:58807
A40	鉄・アルミ	1992	5	120000:120004:120020:120032:12030

コマンド：trim -e 2 1991 4

結果：

A34	電化製品B	59020	58900	58807	
A40	鉄・アルミ		120000	120004	120020

図14：trimによる抜き出し処理の例

trimコマンドのオプションのうち，最初の-e 2は，左から2項目まで通常の項目があり，時系列項目はそれに続いている，ということを示している。次の1991 4によって，1991年の項目以降，4項目を抜き出す指定がされている。つまり1991-1994年のデータを取り出していることになる。それ以前，それ以後の項目は切り落とし，該当範囲のデータがない場合には空項目を出力している。

実際の利用では，見出し行をこれに加えることが多い。時系列項目の見出し行を合成するプログラムなどを含めて，前処理，後処理のためにこまごまとしたプログラムを幾つも作成したが，trimはそのうちもっとも重要なものである。

5.6　サンプルデータ

ほとんど完全なデータのサンプルを，図15に一つ示しておく。このデータは日経総合経済ファイルの一つのレコードである。データファイル中の区切り文字はTAB記号であるが，以下の例では＊で示した。60バイト位置で行を改めているが，データ中にはこのような改行はなく，一件のデータの最後に改行があるだけである。

```
FSHLTC*FINANCIAL OUTSTANDING-PERSONAL-TRADE CREDIT RECEIVED (LIA
BILITIES)*100MILLION YEN*1*L*0*FY*195301*199501*200000*ECONOM
IC STATISTICS MONTHLY*19961008*10.0*00030539*4*1953*43*4385:57
66:6568:7253:9538:11962:14493:17416:20427:28070:36642:47364:
51458:58814:71931:85528:106081:111846:121678:120904:124540:1
49479:196255:229492:238551:273994:283592:310239:341836:37434
2:403214:431823:476969:465727:477756:488643:536267:583736:63
7629:594983:558164:625750:698906
```

図15：データのサンプル（日経総合ファイル）

この一件のデータが，項目ごとに分解されると，表10のように解釈される。

表10：項目ごとに分解されたデータのサンプル（日経総合ファイル）

略称	FSHLTC
系列名	FINANCIAL OUTSTANDING-PERSONAL-...
単位	100 MILLION YEN
属性	1
AG	L
更新中止記号	0
期種	FY
収録開始期	195301
収録終了期	199501
速報開始期	200000
出典名	ECONOMIC STATISTICS MONTHLY
更新日	19961008
有効桁数	10.0
MTコード	00030539
期種シンボル	4
データ開始番号	1953
データ個数	43
データ	4385:5766:6568:7253:9538:11962:14493:17416:...

第6章 RIEB データベース利用の将来

6.1 WWW上の経済統計データ事情

インターネット時代になって，統計データの利用は裾野が広がり，一気に多様化した。データの提供機関は，工夫を凝らして，利用を促進している。統計を集計している機関が発表する統計データは当然どこより最新のものでその利用価値は高い。日本の官公庁は，例えば，日本銀行は予測値をも含めてリアルタイムでWWWに統計情報を提供している。

データがどこのWWWにあるかを調べるにはいくつかのサービスがあって，それを元に探していくことが普通のやり方である。

例えば，*Resources for Economists on the Internet*では，経済学者が必要とするような情報のリンクが網羅されており，統計データについても各国のデータについてリンクが張られている。このサイトはUniversity of Southern MississippiのBill Goffeが主催するものである。ここのホームページから手繰っていくとWorld and Non US Dataというリンクが張ってあって（http://econwpa.wustl.edu/EconFAQ/World/index.html）そこには日本銀行，アジア開発銀行，世界銀行，IMF，OECD，UN等などのリンクがある。こうしたサイトには，データを無料でWWWに掲載しているところもあれば（アジア開発銀行や日本銀行），サンプルだけを見せて有料の登録料を払えば，WWWで見られるところ（国連），またCD-ROMの販売の広告だけを載せているところ（OECD）もある。

日本の官庁の統計資料は，充実しており，日本の大蔵省の統計資料のサイト（http://www.mof.go.jp/siryou.htm）には，租税及び印紙収入，貿易統計，生活関連商品の輸入通関価格の動向，財政資金対民間収支，国債及び借入金現

在高，資金運用部月報，金融先物取引等実績，国際収支状況，対外の貸借に関する報告書，国際収支に関する報告書，外貨準備高，オフショア勘定残高，対外及び対内直接投資状況（年度ベース），対内及び対外証券投資等の状況（月ベース），開発途上国に対する資金の流れについて，法人企業統計調査，景気予測調査に関する統計がある。例えば，貿易統計のところをみると，最新の統計が新地域区分による貿易推移という注釈などを含めて示してあって，見逃せない。

総務庁統計局の統計センターのホームページでは，（http://www.stat.go.jp/）統計データとして，人口に関する基本的な統計として国勢調査人口推計，住民基本台帳，人口移動報告，住宅・土地の状況を明らかにする統計として住宅・土地統計調査，国民の就業・不就業の状況を明らかにする統計として労働力調査，労働力調査，特別調査，就業構造基本調査，求職状況実態調査，国民の生活時間・余暇活動などを明らかにする統計として社会生活基本調査，事業所・企業の実態を明らかにする統計として事業所・企業統計調査，サービス業基本調査，個人企業経済調査，個人企業営業状況調査，科学技術の研究に関する統計として科学技術研究調査，家計の実態を明らかにする統計として家計調査，単身世帯収支調査，貯蓄動向調査，全国消費実態調査，物価に関する統計として消費者物価指数，小売物価統計調査，全国物価統計調査，地域に関する総合統計として地域メッシュ統計社会・人口統計体系，産業連関表などを常時閲覧することができる。

日本銀行のダウンロードのページ（http://www.boj.or.jp/down/down_f.htm）には，各種金利，マネーサプライ，物価指数，短観，外国為替相場，実質実効為替レート，国際収支統計，銀行等対外資産負債残高，平成７年基準卸売物価指数遡及表（平成７年１月～９年11月），平成７年基準卸売物価指数遡及表（平成７年１月～９年11月）などとExcelで読めるファイルが提供されていて非常に役に立つ情報が提供されている。

国際機関の中ではADBが結構サービスがよくて，各国のテーブルとトピッ

クごとのテーブルを用意してくれている。Regional Tablesとしては*Population, Poverty and Inequality Indicators, Environment Indicators*などをはじめとして40種の統計が提供され，そして各国別の統計としては，ADB加盟国を中心に*KEY INDICATORS of Developing Asian and Pacific Countries*そのものがExcelファイルでダウンロードできる。このように，統計を収集する機関が研究者や一般に無料でデータを公開するのは大歓迎である。ことに政府や国際機関ではできるだけ無料として欲しい。そしてその加工も自由にできるようにしてもらえれば，ユーザーにとって使いやすいデータベースが世界のいたるところで提供されるようになるであろう。IMFの様にホームページを開けばCD-ROMの宣伝だけというのでは，あまりに寂しい。

6.2　RIEBデータベースとWWW上の統計データの連係

このように最近は，最新のデータはネットサーフィンをすればたちどころに情報を得ることが出来るようになった。しかし，前節の説明でもわかるように，データが利用できるのは，基本的に出版されたものと同じような，表形式である。インデックスをたどって，例えば，ADBのYield of Paddy and Maizeという項目を探し，それをダウンロードしても国際比較の表形式のデータが得られるに過ぎない。RDB的な検索をして，必要なだけのデータが検索，抽出できるシステムとはなっていないのである。いってみれば，本の目次を探して，必要なデータが載っている表を探し出す。そしてその表をとりあえず，ダウンロードして，他の表と合わせて，利用する。基本的には図書館の参考資料を沢山目前の端末に出すというにすぎない。折角，インターネットで電子情報が手軽に取れるようになったのだから，もう少しRDB的検索機能が充実してきてもよさそうなものである。

基本的な時系列データは自前のデータベースで検索，抽出するようにして，

最新のデータはWWWで探すという利用の仕方が一番ポピュラーな利用法となろう。

しかし，WWWでの探し方が，今後は変わってこようし，その方向性について，リクエストを出しておくのも，無駄ではなかろう。

- まず，公共の機関は，無料でデータを公開してほしい。
- 機器のワールドスタンダードがあるように，データ提供の方法のワールドスタンダードがあってしかるべきであろう。
- 例えば，アジア開発銀行，OECD，世界銀行，IMFなどが先ず，同じフォーマットでデータをCD-ROMのみならず，WWWでも提供してくれれば，それにあわせて各国の統計機関もデータ提供をするようになるであろう。そうすると，ユーザーは戸惑うことなく，データの利用ができる。
- もう一歩進んで，公共機関がデータの転載・加工を基本的に認めれば，こうしたデータを収集するサイトを世界中に相当数作って，そこでは，例えば，我々が提示するような標準化したデータを保持し，それを検索・抽出するプログラムをWWWに搭載するようにすればどれほど研究活動が効率的になるかしれない。

我々のRIEBデータベースがベストというつもりはないが，試行錯誤を経てWWWデータベースの世界標準的なものができあがれば，RIEBデータベースもそれに合わせて変貌させ，ユーザーが，WWWでもイントラネットでも非常に親和性の有る一体化したデータベースシステムを利用できるようにするのが理想的である。

6.3 データの広がり

ここまではデータといえば統計データのみを考えてきたが，普通は文字情報のデータベースの方がポピュラーであろう。ニューヨークタイムズや朝日新聞

などをはじめとして，新聞もインターネットでの公開についてはいろいろと試行錯誤を続けている。NBERなど，ワーキングペーパーで有名なサイトも印刷物を購入しているところには，WWWでのダウンロードを可能としている。*Wall Street Journal*も同様である。このように電子メディアで情報が公開され始めると，その情報を蓄積したものがデータベースとして重要な価値をもつことになる。

記事検索のサービスもいろんなところで提供されている。日経は日経テレコンに加入していれば有料でWWW上で記事検索ができる。*Far Eastern Economic Review*は記事データをCD-ROMで最近販売した。

静止画像や音のデータ，動画など，文字通りマルチメディアにデータベースはこれからは対応していくであろう。サーバー側もクライアント側もこうしたデータに十分対応できるようになった現在，今後は扱うデータの幅がますます広がり，情報の質と量はますます増え，かつ廉価に提供されてこよう。

RIEBデータベースは現在のところ統計データに限定しているが，マルチメディア情報は少し先としても，こうした新聞記事のような文字情報は近い将来その守備範囲とすることになろう。そうしたとき，ますます，パソコンレベルでのCD-ROMの利用とWWWの情報の利用との共存をどう効率よくとるかが重要となってくるのである。

単にリンクを張るだけでなく，そうしたユーザーにとって利用価値の高いシステムを構築することを真剣に考えていかなければならない。その一歩がWWW上でのデータベースの国際標準の構築で，我々の今回のRIEBデータベースはその一つのたたき台として今後いろんなところで使われ，システムが進化していくことを願っている。(76)

---

(76)　大阪市立大学からシステムの提供について相談を受けている。

第Ⅱ部

RIEB データベース・マニュアル

第7章　RIEBデータベース・マニュアル

神戸大学経済経営研究所では，早くからデータベースを構築して研究者の利用に供してきた。クローズドな大型汎用機環境からインターネットと接続したオープンなワークステーションへとコンピュータの環境が変貌を遂げるに合わせて，データベースシステムを新たに再構築し，ユーザーがマニュアル無しで簡単に使えるものとした。Windowsユーザーがほとんどなので，そのOSのGUIと違和感のないシステムを構築したのである。

とはいっても，どのように使うかについて多少の手ほどきは必要であろう。

この章では，データベース利用のマニュアルとしても役立つように，具体的に例題などに解答を与えながら，現在のデータベースの利用方法を示しておくことにした。

データベースとしては多国籍企業データ，OECD貿易ファイル，日経総合経済ファイル，DRI米国マクロ経済データ，IMF *Economic Information System* データ，をこれまでにワークステーション用に再構築した。

これらのデータベースはいずれも，同じような操作方法で利用できるようにしているため，代表的な日経総合経済ファイルの操作方法を示しておけば，他はそれほど苦労なく操作できる。したがって，日経総合経済ファイルに関しては，丁寧な説明を試みるが，他のデータベースの操作方法については簡単に相違点ぐらいを述べるに留めている。

また，各データベースのテクニカルな問題点もこの章に含めた。データベースの利用にあたっては，こうした点を十分考慮の上，利用されることを望む。

これまでのように汎用機にLOGONしてSECRETARYを通してデータを取得する場合は，汎用機の利用自体が研究所に関係する研究者に限られていたのであまり著作権に関しては神経質にならなくて良かった。RIEBデータベース

は，しかし，著作権の関係上所内限定のイントラネットで公開している。このデータベースはWWWシステムを利用しており，研究所所内にだけそのアクセスを許可し，外部の不特定多数がインターネットを介してアクセスすることはできない仕組みになっている。これは，データ提供機関と契約を結び，アクセス権が研究所に関係する研究者に限るものとしているからである。

7.1 一般的な操作方法

代表的な条件設定・検索実行の方法を，日経総合経済ファイル（図16）を例にとってまず示してみよう。

利用者はパソコンなどのWWWブラウザを用い，目的のデータベースの検索画面を呼び出す。その画面をみれば操作方法は大体自明であるが，以下に利用の手順を示す。

1．WWWブラウザを用いて，データベースサーバーに検索の条件，出力したい項目を指定する。
2．実行結果をWWWブラウザ上でプレビューし，さらに抽出条件を調整する。
3．結果を確かめた上で，電子メイルの添付ファイルによる送信，またはネットワークディスクに対する書き込みを行わせる。
4．ユーザーは，電子メイル，またはネットワークディスクをチェックし，TABセパレート形式で保存されているファイルをExcelなどで読み込んで利用する。

7.1.1 検索条件・出力項目設定

検索インタフェイスの画面を図16で示しているが，そこには検索条件を指定するいくつかの設定項目がある。期種，抽出期間指定のところでは開始年と終

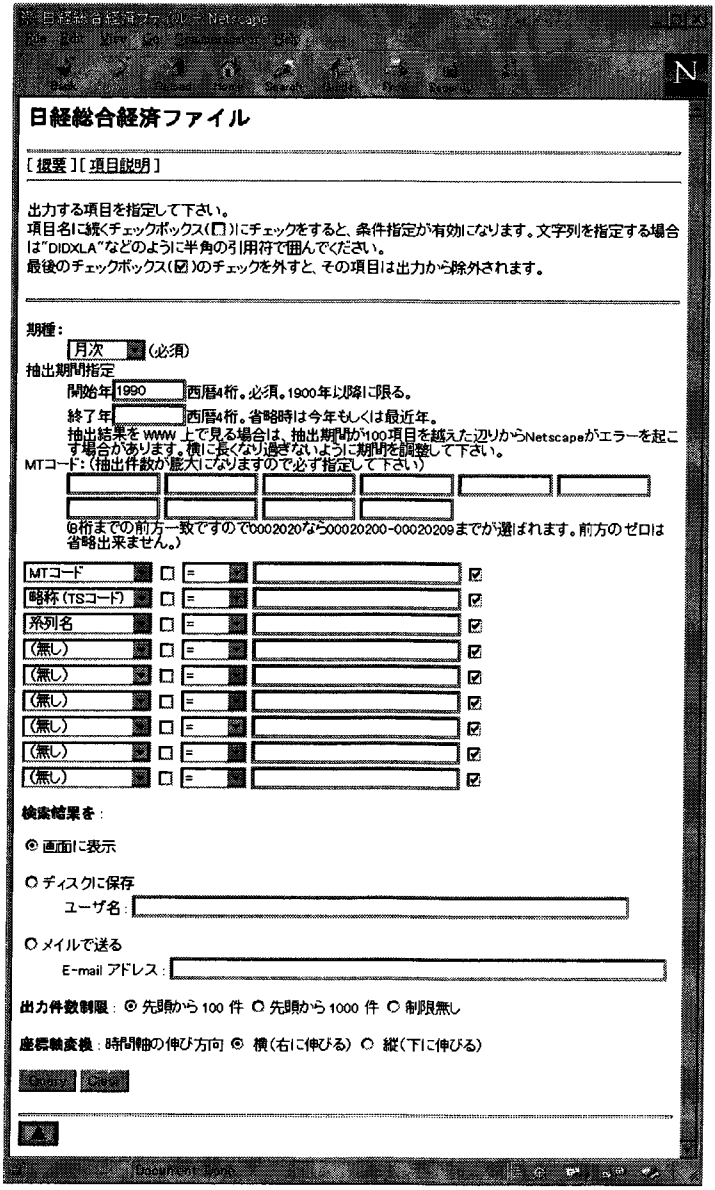

図 16: 日経総合経済ファイル WWW インタフェイス

了年，MT コード，それからいくつかの出力項目指定，そして検索結果に対するオプションの各設定箇所が見える。

ここで設定する期種と抽出期間の検索条件は，基本的に AND の関係で処理される。期種が月次で，抽出期間が1990年から1995年であるとすると，その条件をすべて満たすものが，つまり1990年 1 月から1995年12月までの月次データだけが抽出されることになる。

一方，MT コード設定フィールドのように，複数の値を列挙するような条件設定は，OR の関係として処理される。つまり MT コード欄に00020200, 00030200, 00040200 と列挙した場合は，MT コードがこれらの値のいずれかであれば，条件に合致したとみなすように設計している。

最後のブロックの出力項目指定のところでも，検索条件を指定できる。しかし，これはそれまでの部分との AND で処理される，すなわち全ての条件を満たすものだけを抽出するので，例えば，MT コードで項目を何か指定しているとするとその中から，ここで指定した条件のものだけが選択される。MT コードのところで指定をしないで，ここのブロックだけで検索指定することも可能である。

以下に，検索ページの上から順を追って，各設定項目の設定方法について具体的に説明しておこう。

この操作方法は，日経総合経済ファイルに限らず，すべての WWW 検索ページに共通と理解してよい。

・期種コード

日経総合経済ファイルは，登録されている系列によって，年次，半期，四半期，月次などの期種に分かれている。期種を選択しない限り，どの系列が選ばれているかが，コンピュータは判断できないので，ユーザー側で明確に選択する必要がある。この選択は，プルダウンメニューを指定することで実行される。次の図は，期種指定のフィールドをクリックし，期種を

指定するプルダウンメニューを表示させている状態を示している。

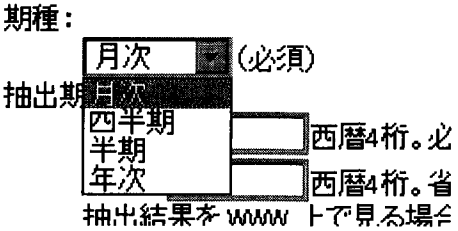

図17：期種指定フィールド

・抽出期間指定

時系列データを扱う場合は，抽出する初めの年と，終りの年を指定しなければならない。2000年問題を避ける意味でも，西暦4桁で入力するように設定している。データベースにない年をここで指定した場合は指定エラーとなる。ここを入力しないで次の条件設定に移ると，系列で利用可能なすべての期間が指定された（これがデフォルト）と解釈する。

・MTコード指定

日経総合経済ファイルは，MTコードをキーコードとして検索する仕様になっている。例えばMTコード00020200は，ORDER RECEIVED FOR CONSTRUCTION-GRAND TOTAL（BIG 50 CONTRACTORS）に対応している。ユーザーは手元にコード表を持っている必要がある。また，MTコードは列挙して指定できるように設計している。MTコードは常に8桁で，同じ分野の系列が一塊でコードに割り振られているため，常に前方一致でコード指定を行えるようにしている。つまり，00020200から00020209までのデータを抽出したい場合は，単に，0002020と指定すれば良いようになっている。

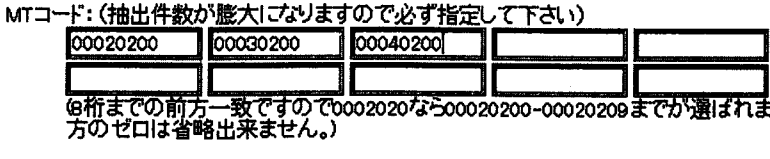

図18: MT コードの指定

普通は，ここまでの設定で，検索の準備は整ったことになる。抽出する系列の属性が分かっている場合はこのまま検索・抽出を行えばよいが，系列に付属する詳しい情報，例えば，単位なども同時に出力しておきたい場合は，出力項目指定を利用する。

・出力項目指定と検索条件の指定

日経総合経済ファイルの場合には，データ・キーとなる MT コード等の他に，データを特定する略称，系列名称，などの情報を同時に出力することができる。各系列ごとにその他収録開始期などの幾つかの属性項目がある。ここでそれらを指定することによって，属性項目を出力カラムに加えることや，指定した属性を持つデータを検索条件にすることができる。

この出力項目指定では，必要ならば $=$，$>$，$<$，$not =$ などの検索条件を指定することもできる。先の MT コード指定では，MT コード(の前方一致によるパターン)の列挙による指定しかできなかったが，この機能を利用すれば，例えば MT コードの範囲指定など，より自由度の高い検索条件の設定が可能である。この点でもフレキシビリティが向上している。

出力項目指定フィールドにどのように適切なパラメタを入力していくかを次に示そう。

図19: 出力項目指定フィールド

各フィールドの意味を，左側から説明する。

1．出力項目指定

検索結果を出力する際，そのデータに付属する情報を同時に出力する方が便利であろう。データは一般に多くの属性を持っている。例えば，図20のプルダウン表示でもわかるとおり，系列名，出典名，単位などである。出力カラムに追加したい属性項目を，プルダウンメニューのリストを手繰って指定する。検索する系列の属性を2つ以上出力結果に含めたいときは，例えば，一番上の出力項目指定をマウスでクリックし，プルダウンメニューから“系列名”を選択し，次の出力指定項目で，“出典名”を選択するというように，各指定ボックスに一つづつ入るように，指定していく。デフォルトではMTコード，略称，系列名が選択されている。属性項目の一覧表，各項目の仕様・定義については検索ページ

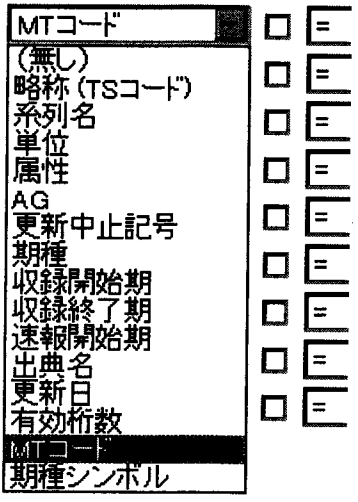

図20：属性項目のプルダウン表示

上の「項目一覧」のリンクをクリックすれば情報を得ることができる。

2．条件設定チェックボックス

項目を単に出力カラムに追加して出力するだけならここのチェックをする必要はない。データ検索の条件として MT コードを用いてより高度な設定がしたいとか，系列名で検索したい場合にこのチェックボックスを利用する。ここをチェックすることによって，それより右に書かれた条件設定が有効になる。デフォルトではチェックはオフになっている。

3．条件設定フィールド

すぐ右の値フィールドに書かれた内容と属性項目との関係，つまり検索条件を指定する。＝，＞，＜，*not* ＝ などが設定可能である。

4．値フィールド

条件の値を入力する。数値はそのまま，文字列は "Sample" のように二重引用符で囲んで入力する必要がある。

5．出力指定チェックボックス

条件指定に利用するだけで，属性項目自体の出力を望まない場合には，ここをオフにすれば，検索条件の設定は有効なまま，属性項目のを出力カラムに加えないようにできる。

例えば，「略称 ＝ "DIDXLA"」という条件で検索するためには，出力項目指定には「略称」を選び，条件設定チェックボックスにチェックし，条件設定フィールドは「＝」を選べばよい。

図21：略称による条件指定の例

その場合，得られた結果のうち，「略称」項目カラムの出力は，全て「DIDXLA」となる。この結果は自明のことなので，略称カラムの出力が不要の場合は，最後の出力指定チェックボックスのチェックを外せばよい。ここの条件設定を有効に使えば，高度な検索ができる。この出力項目指定における各行の条件設定はANDの関係で処理されるので，同じ項目に対する条件設定を2行にわたって指定すると，範囲指定による絞り込みが可能となる。例えばMTコードは数字である。この場合，このコードからこのコードまでという範囲を指定することが容易にできる。列挙によるMTコード指定をせずに，出力項目指定による範囲指定を次のようにすればよいのである。

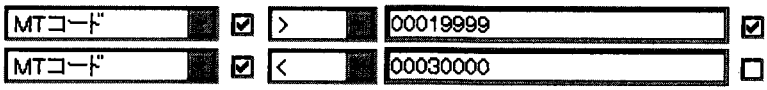

図22: MTコードによる範囲指定の例

これによってMTコードが00020000から00029999までのものを抽出することができる。この場合，何もしないとMTコードが2列にわたって出力されることになる。1列で十分なので2行目の出力指定チェックボックスを外している。

7.1.2　検索結果の表示・受け取り指定

出力の条件が設定できたら検索の実行を指示する。それにはQueryボタンをクリックすればよいが，その前に出力先を指定する必要がある。

検索結果の出力先として3つを用意した。これらは27ページの図3に示した，クライアント側システムへ向かう矢印にそれぞれ対応している。

1．ブラウザ画面上への出力。

2．ネットワークディスク上のユーザーのディレクトリへの出力。

3．電子メイルの添付ファイルとしての送信。

この選択は，次の図の，該当するラジオボタンをチェックすればできる。ネットワークディスクに出力する場合は，出力先のディレクトリ名となるユーザー名を指定する必要がある。同様に電子メイルによる転送を選択した場合は，宛先のメイルアドレスを入力する必要がある。

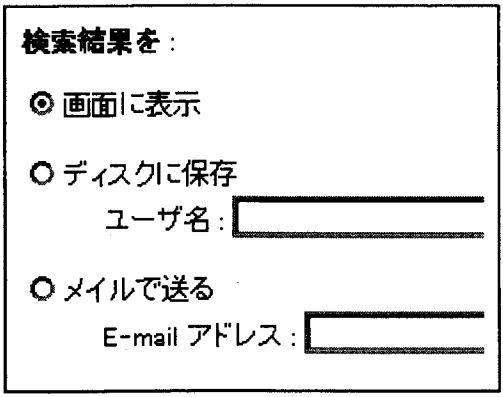

図23：出力先の指定

出力件数の制限を設定する場合は，次のラジオボタンで適当なものをクリックする。

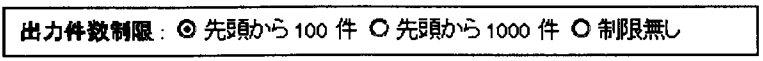

図24：出力件数の制限

出力結果の縦軸と横軸を入れ換えるには，次のラジオボタンをクリックする。

図25：出力結果の座標軸変換

出力先に関する設定はこれですべてで，この設定が完了すれば，次のQueryボタンをクリックして検索処理を実行する。右のClearボタンは各種設定を全て消去するためのものである。

出力先の設定のデフォルトは画面表示，先頭から100件，時系列データの並びは横である。検索条件を誤って指定していないかどうか，このデフォルトの出力で，検索条件を確かめてから，出力先を最終的に指定することを推奨する。

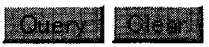

図26: Queryボタン

7.1.3　プレビュー

検索結果のプレビューとするのに適当なサイズ，出力件数100件をデフォルトとした。(図27)

期種：年次：期間 1995 年から 1998 年まで

MTコード	略称	系列名	収録開始期	収録終了期	有効桁数	199501	199601	199701
00100220	CREG	CAPITAL TRANSFERS FROM RESIDENTS-GENERAL GOVERNMENT	195501	199601	10.1	-867.94	-2082.12	
00100204	DEPOF	OTHER DEPOSITS-LIABILITIES-FINANCIAL INSTITUTIONS	195501	199601	10.1	19784.8	12736	
00100192	SSECF	BILLS & BONDS, SHORT TERM-ASSETS-FINANCIAL INSTITUTIONS	195501	199601	10.1	616.6	1048.2	

図27: 検索結果のWWW上でのプレビュー

こうしたプレビュー結果が簡単に見られれば，条件設定の指定を試行錯誤で繰り返すことも気にならず，効率良く最終的に目指すデータの抽出にたどりつくであろう。ブラウザのBackボタンをクリックすれば，元の条件設定画面に戻ることができ，そこでは前の設定条件がそのまま残っているため，変更箇所だけに限定して新たな検索条件の調整を行えばよいのである。

条件設定が最終のものとなった段階で，検索結果をメイル添付もしくはディスク保存のどちらかに設定し，出力件数を制限なしとして Query ボタンをクリックする。このように検索・抽出を実行してデータを受け取るのがベストである。

たとえ検索結果が画面表示で間に合うぐらいであっても，ブラウザによる画面表示からデータ結果をとるべきではない。なぜなら，もともと，WWW ブラウザは膨大なデータを正確に表示するようには作られていない。正確なデータを表示するより，表示速度・見ばえの良さに開発の目標が向けられているからである。実際 Netscape Navigator のあるバージョンでは，横に非常に多くの項目を持つデータを TABLE 形式で表示させようとしたらパソコン自体がハングアップする場合もあった。画面出力はプレビューにとどめ，正しい抽出結果は画面出力以外の方法を利用するのを推奨する所以である。

7.1.4 検索結果の受け取り処理

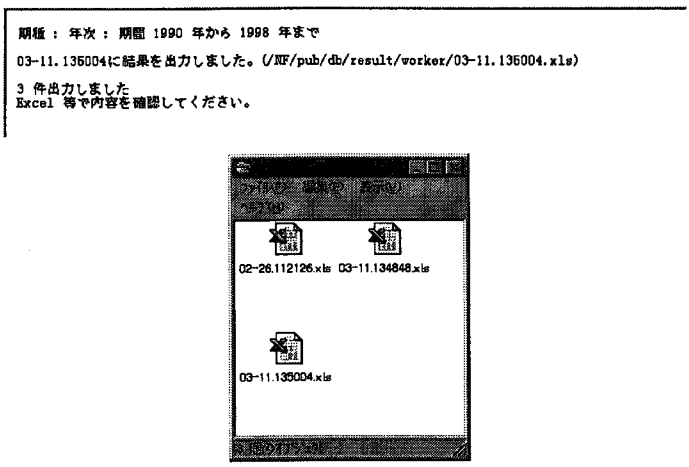

図28: 検索結果のネットワークディスクへの出力例

検索結果をネットワークディスクに出力した場合，WWW画面上には出力したファイル名が表示される（図28上）。そのディレクトリは研究所LANに接続されたクライアントパソコンからは，NFSの機能を利用して直接フォルダとして開くことができる（図28下）。

利用者はフォルダを開き，該当ファイルをExcelなどで直接読み込むだけで，抽出結果を扱える状態になる。Excelがインストールされた Windows95などでは，ファイルは最初からExcelの文書アイコンで示されるので，そのアイコンをダブルクリックするだけでよい。

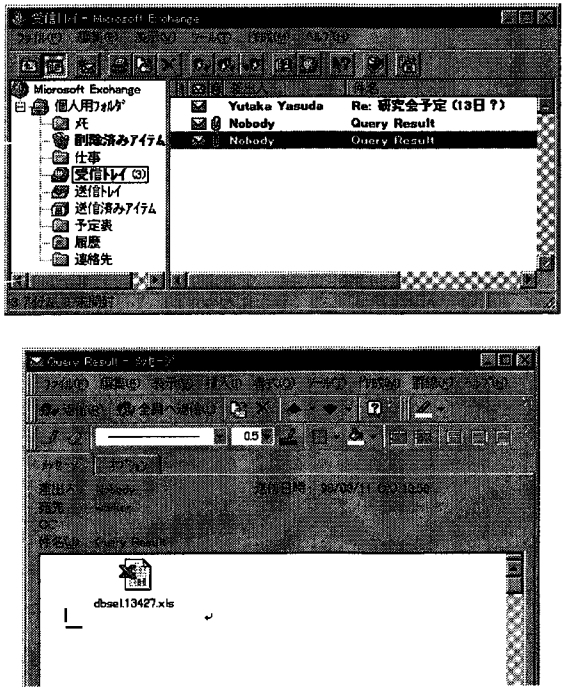

図29：検索結果のメイルへの添付

検索結果をメイルに添付して出力した場合には，指定したメイルの宛先に添

付ファイル付きのメイルとして届く。Windows95 対応の多くの email ソフトを使えばその添付ファイルをダブルクリック（開封）するだけで Excel が起動し，そのまま編集，解析作業に移れるはずである。図29は Microsoft Exchange[76]での受信例である。メイルを開くと，添付ファイルは Excel のファイルとして見え，抽出結果を Excel で解析できるようになる。Excel がインストールされた Windows95 における Microsoft Exchange などでは，このアイコンをダブルクリックするだけでよい。

最後に検索結果を Excel で開いた状態を，図30に示しておこう。

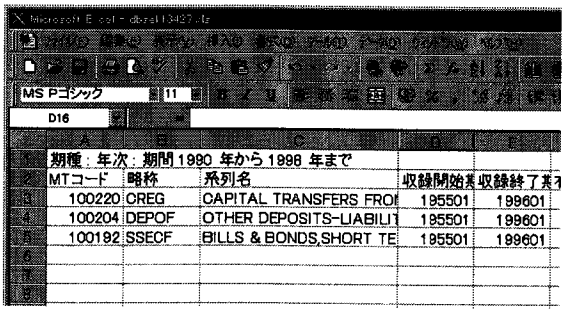

図30：検索結果の Excel への取り込み例

次の節からは，各データベースの解説を行う。

7.2 多国籍企業データ

7.2.1 概要

多国籍企業データベースは，基本的に東洋経済新報社から海外進出企業データの MT を購入し，それをアレンジしてデータベースとして提供しているものである。東洋経済新報社は独自に数千社におよぶ日本の企業や海外の日系企

(76) Microsoft 社が Windows95 に標準的に用意したメイルユーティリティ。

業にアンケート調査し，そこで得た情報を『海外企業進出総覧』などとして出版しているが，MTにはこの出版物のもととなったデータを収録している。

データはこれまでその年度のデータのみ毎年MTで提供されてきている。MTで受け入れた際の仕様を表11に示しておく。また図31にWWW検索インタフェイスを示す。

表11：海外進出企業データ仕様

レコード長	2600バイト
件数	148,104件（1997年度時点の合計）年間約18,000件程度
容量	約385 MB（DBデータの標準化後52 MB）
1バイト文字コード	EBCDIC
2バイト文字コード	JEF（富士通汎用機コード）ただしJEF拡張漢字，外字あり 漢字項目中の英数字は2バイト文字コードを利用
主キー項目	年度，会社通番，国コード（独自）
他のキー項目	業種コード（独自），日本側出資企業の会社コード（独自）

収録項目は表12の通りである。

表12：海外進出企業データの項目仕様

現地法人	国コード，一連番号，州特別区コード，国名（英語表記，漢字表記），社名（英語表記，漢字表記），代表者名，所在地（英語表記，漢字表記），電話番号
業種	業種コード，業種名，事業内容
資本金	金額，単位，通貨単位
従業員	従業員合計，日本からの派遣社員数，うち役員
進出状況	各種フラグ，年月，その他
日本側出資企業 （16社まで） 現地側出資合計(%)	上場区分，企業コード，社名，出資比率，日本側企業数，日本側出資合計（%）
合弁の相手先 （5社まで）	社名，出資比率
業績関連	項目区分，年月，金額，単位，通貨単位，$換算額
投資目的	

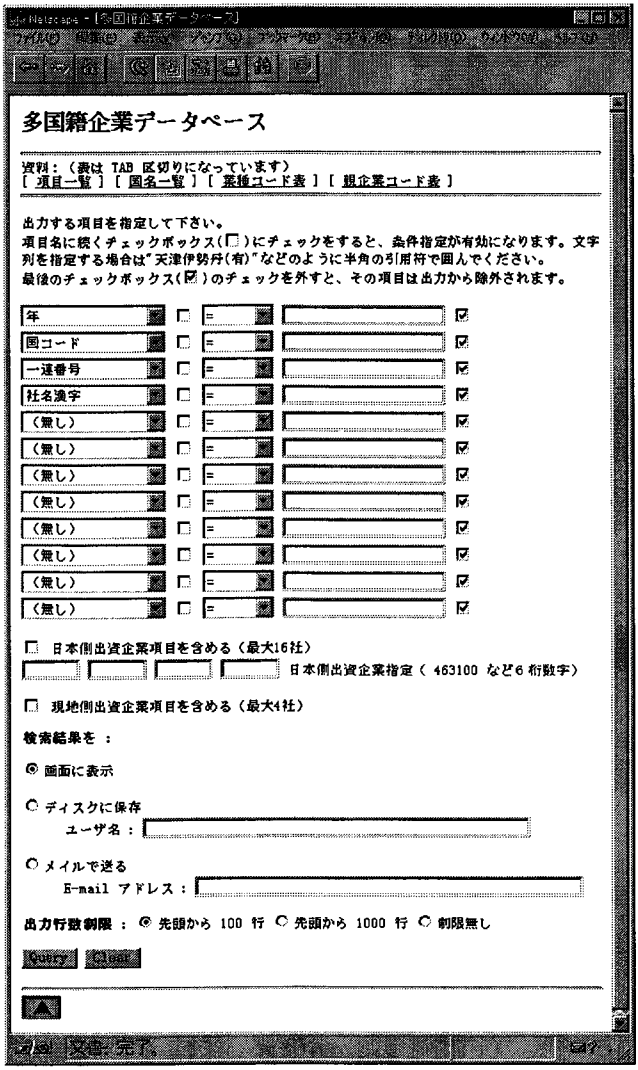

図31: 多国籍企業データ WWW インタフェイス

7.2.2　操作方法

海外進出企業データでの，主要な検索キー項目は次のものである。

表13：海外進出企業データのキー項目仕様

年	西暦4桁数字。この年は東洋経済新報社から出版されている，『海外進出企業総覧』の年と同じである。データはアンケートを集計したもので，アンケート調査の日時などについては，それぞれの年の出版物で確認されたい。普通は，出版年の前年末にアンケート調査を実施しているようである。
国コード	数字3桁の独自コード。
一連番号	国ごとに付けられた現地法人コード。数字5桁の独自コード。現地法人は，国コードと一連番号のペアで特定できる。
業種コード	数字4桁の独自の業種コード。
親企業コード	日本側出資企業を示す。数字6桁の独自コードである。

国名（国コード），業種コード，親企業コードの一覧については，検索ページ上のリンクをクリックすることにより，その情報を得ることができる。また，海外進出企業データの各項目の一覧も，同じくリンクをクリックすればWWW画面に示される。

・「日本側出資企業項目を含める」チェックボックス

これを指定することによって，親企業の現地法人に対する出資状況が出力項目に追加される。具体的には上位16社までの上場区分，企業コード，社名，出資比率，日本側企業数，日本側出資合計（%）の各項目である。

・日本側出資企業指定フィールド

ここに親企業コードを列挙することによってその親企業が出資している現地法人のデータが出力できる。

・「現地側出資企業項目を含める」チェックボックス

これを指定することによって現地側出資企業に関する情報が出力項目に追加される。具体的には現地側出資合計（%）と上位4社までの合弁の相手

先に関する社名と出資比率の各項目である。

7.2.3 DB データの標準化に際しての問題点

東洋経済新報社の海外進出企業データは結構データの精度にばらつきがあり，技術的にも仕様を満たさない異常なデータがかなり含まれている。ことに過去の年次のデータと最新のデータとの整合性はとれていない場合がある。仕方ないことではあるが，中国，台湾，香港，韓国などの漢字文化圏を含んでいるので，一般的でない文字の利用も多い。

全般的なデータの質 仕様との不整合なデータが結構あって，これがDB データの標準化作業で大きな障害となった。例えば，国コードと国名が仕様書（書類）と一致していない，漢字コードとして JEF の定義にない不正なコードが含まれている，空白コードが JEF では x4040であるが，それ以外に xA1A1 が使われているなどである。MT の出荷の際に各項目間の論理的整合性をチェックしていれば簡単にこれらのミスには気がつくと思えるが，どのように扱っているのか実際のところは分からない。

我々は独自で DB データを標準化（33ページ第 3 章参照）する際に，これらの異常データは可能な限り排除した。

過去の年次のデータとの互換性 経済経営研究所では，1984年から，東洋経済新報社と共同で海外進出企業データを開発してきた。それゆえ，データは1984年のものから存在するが，1984，1985年のものは，現在のデータとは収録項目，フォーマットなどにかなりの差があり，今回の作業で，データベースに含めることは断念した。また，1986年のデータについても，DB データの標準化の作業中に，多数のエラーを発見し，修復が不可能と分かったので，これもあきらめざるを得なくなった。最終的には，1987年以降のデータだけを採用することにした。1990年にフォーマットの変更が一度実施され，その時，失った項目も幾つかある。逆にフォーマット変更により新設された項目もある。この場合，1990年以

前のデータの該当する項目については空白値を埋めることとした。この点，利用の際には注意が必要である。

漢字　2バイト文字コードはJEFコードであり，拡張漢字や外字が非常に多く含まれている。特に中国漢字などがJEF拡張漢字に若干含まれており，データ全体の中ではJIS第一，第二水準に変換できない文字が多数あった。例えば国コード106（台湾），通し番号00070の社名「台湾＝旦（股）」などである。全データ148,000件中，約6,000件のデータにこのような特殊文字が使われている。

データ処理においては，一般的でない文字を多用することは，非効率である。データ処理のために，例えば社名や地名の項目には出版用とは別に，JIS第一水準，第二水準の文字で代替したフィールドを用意するなどの対処，もしくは，将来的に外字の標準化などが考えられる。

キー項目のコード　ほとんどのキー項目のコード体系は独自のものであり，標準的なコードの他のデータと共用することは，現状のままでは困難である。

もっとも問題となるコードは国コードである。海外進出企業データは，元来，毎年の出版物のための基礎データという性格上，それほど，過去のデータとの整合性に注意を払う必要がなかったのであろう。我々としては，これでは困るが，現実は，例えば，マカオは92年まで129，93年から109という国コードで登録されていたりする。幸い，両コードは他の国には使われていない。リヒテンシュタインは900，235，260と三度もコード変更され，このうち235は現在エストニアに割り当てられている。このように，DBデータの標準化はまったく労働集約的で，時間のかかる作業である。一旦これをすますと，しかし，オリジナルよりコンシスタントなデータを提供できることも事実である。多国籍企業データベースでは，時系列データとして過去のデータを収録しているが，国コード情報を扱う場合には，したがって，十分注意する必要がある。(77)

(77)　国名もデータに含まれているが，細かな綴りの変更などがあり，国名のほうが国コード以上に信頼できない。

こうした不整合は，ある程度まではDBデータの標準化作業で，手作業で修正したが，不整合すべてを取り除くには至らなかった。どの場合でも，単年では矛盾がないのがせめてもの救いである。

データ提供会社の編集方針によっては，単純に単年データを毎年蓄積していくだけでは，時系列データを蓄積することにはならない良い事例である。

7.2.4 検索例題

例題 1

1990年時点でタイに進出している家電業界の企業名を調べよ。

検索例

《多国籍企業データベース》[78]にリンクのある《国名一覧》で，タイの国コードをブラウザの検索機能を使って調べる。《業種コード表》で，家電というキーワードを使い，該当しそうなコードを調べる。

以上を下調べとして，次に検索ページの指定項目に条件を指定していく。出力項目指定で「年」を選ぶ。条件設定チェックボックスをチェックする。条件指定フィールドで「＝」を選び，値フィールドに1990を入力する。次に出力項目指定で「国コード」を選び，条件設定チェックボックスをチェックする。条件指定フィールドで「＝」を選び，値フィールドにタイの国コード(111)を入力する。

その次の出力項目指定では「業種コード」を選び，条件設定チェックボックスをチェックする。条件指定フィールドで「＝」を選び，値フィールドに家電の業種コード(2630)を入力する。

デフォルトでは，全ての出力指定チェックボックスにはチェックしてあるので，そのままにする。

---

(78) 検索例における《.....》は，WWWページ名もしくはリンクの名前を意味する。

例題 2

現時点での松下電器もしくは三洋電機を親会社とする，中国とマレーシアでの日系企業（海外進出企業）の法人名，資本金，設立年月日，従業員数，業種を調べよ。

検索例

《Web経由でのデータベースアクセス》で，多国籍企業データベースの最新が何年版であるかを確認する。《多国籍企業データベース》にリンクのある《親企業コード表》で，松下電器と三洋電機のコードをブラウザの検索機能を使って調べる。同じ方法で，中国とマレーシアの国コードを《国名一覧》で調べる。

以上が検索に入る前の下調べで，次に検索ページの指定項目に条件を指定していく。出力項目指定で「年」を選び，条件設定チェックボックスをチェックする。条件指定フィールドで「＝」を選び，値フィールドに確認した最新年を入力する。次に出力項目指定で「国コード」を選び，条件設定チェックボックスをチェックする。条件指定フィールドで「＝」を選び，値フィールドに中国の国コード（105）を入力する。これらの出力指定項目に入力した検索条件はANDで処理される。従って，中国の検索結果を手に入れた後，マレーシア（113）に変更して検索する必要がある。

以降の出力指定項目では，それぞれ「社名」，「進出年月」，「従業員合計」と「業種名」を選択する。

資本金については，出力指定項目を3つ指定しなければならない。「資本金額」，「資本金単位」と「資本金通貨」である。

デフォルトでは，全ての出力指定チェックボックスにはチェックしてあるので，そのままにする。

「日本側出資企業項目を含める（最大16社）」のチェックボックスをチェックし，「日本側出資企業指定」項目に，松下電器と三洋電機の親企業コード，

(675200 676400) を入力する。

7.3 OECD貿易ファイル

7.3.1 概要

OECD ITCS[79]貿易ファイルは，SITC[80]コードによって産業別，相手国別に分類された輸出入データである。レポート国はOECD加盟国を中心とした30カ国以上，パートナー国として全世界をカバーする200カ国以上がその収録範囲である。SITC Revision 2による分類コードを例にとると，2500分類×30×200×2で，単純計算で3000万件，実際には全件で3800万件に及ぶ膨大なデータである。

データは1993年まではMTで，単年データのみが提供されてきたが，1997年からは数枚のCD-ROMで提供されはじめ，容量の制約が幾分外れたことから，過去の年次のデータも含まれるようになった。

1．SITC Revision 2による，最近の過去9年間のデータを含めたもの。

2．SITC Revision 3による，最近の過去9年間のデータを含めたもの。

3．SITC Revision 2による，1961年から1990年までのデータを含めたHistoric版と呼ばれるもの。

まず最初に1の版をデータベース化した。

次にOECDから送付されてきたままのデータの仕様を示す。

---

(79) International Trade by Commodities Statistics

(80) Standard International Trade Classification，国連が作成した標準国際商品分類。Rev. 1, 2, 3と改訂が重ねられている。商品分類を5桁の数字で行い，最新のRevision 3では一番詳細な分類で3,000以上の品目がある。

表14：OECD貿易ファイル仕様

件数	3800万件（1997年度時点の合計）
容量	約4GB
1バイト文字コード	ASCII
2バイト文字コード	なし
主キー項目	レポート国，パートナー国，SITCコード，輸出入

収録項目は以下の通りである。

表15：OECD貿易ファイルの項目仕様

キー	レポート国，パートナー国，SITCコード，輸出入
貿易額	（年度ごと，整数10桁以上）

7.3.2　二種類の検索システム

OECD貿易ファイルは他のデータベースと比べるとそのデータ量が非常に大きい。検索の際の主要なキーとなるSITC分類コードの種類も2500種を越える。SITCコードは5桁分類まであり，必要な産業を5桁分類のみで指定するのは数が多くなりすぎて，効率が悪くなるであろう。貿易の全体の流れをもっと大きな産業分類で見たい場合のことを考えて，47部門に集約したデータを別途用意した（**7.3.4**に後述）。

我々はOECD貿易ファイルに関しては，二種類の検索システムを構築した。

1．全件版。SITC5桁コードによる最も詳細なデータ。

2．47産業版。SITCコードによる分類を，47産業に集約したもの。

図32に全件版，図33に47産業分類版のWWW検索インタフェイスを示しておいたが，操作方法は大体自明であろう。

7.3.3　操作方法

基本的な操作方法は，前節94ページの**7.1**に詳しい説明がある。それと重複しない程度の説明をここでは試みる。

OECD貿易ファイルにおける，主要な検索キー項目は次のものである。

表16：OECD貿易ファイルのキー項目仕様

レポート国	OECD加盟国および台湾などの国コード。JPN，USAなど三桁の英大文字を用いている。コード体系としてはISO 3166-1を用いる。
パートナー国	レポート国との貿易相手国コード。フォーマットはレポート国と同じ。
輸出入	Import/Exportを，'I'と'E'の英字一文字で表す。

国コードが分からなければその一覧は検索ページの最上部にあるレポート国コード表のリンクをクリックすることにより閲覧することができる。これに業種分類が加わるが，全国版と，47産業分類版ではそれぞれキーとなる分類コードが異なる。

表17：全国版と47産業分類版のキー項目の相違

	キー	形式
全国版	SITC コード	1桁から5桁までの数字列。
47産業分類版	Class コード	00から47および99の数字二桁。

同様にSITCコードについてもSITCコード表のリンクをクリック，47産業分類のClassコードについても，Classコード表のリンクをクリックすればそれらの情報を閲覧できる。120ページの，表18も参照。このSITCおよびClassコード表は列挙による指定が可能である。

・SITCコードの桁の意味

SITCコードの付け方は分かりやすい。以下に例示する。

21191 Parings & other waste of leather

21199 Hides and skins, n.e.s. raw (fresh, salted, dried etc)

SITCコードが2119で始まるデータはこの2つだけだが，この二つの合計が，

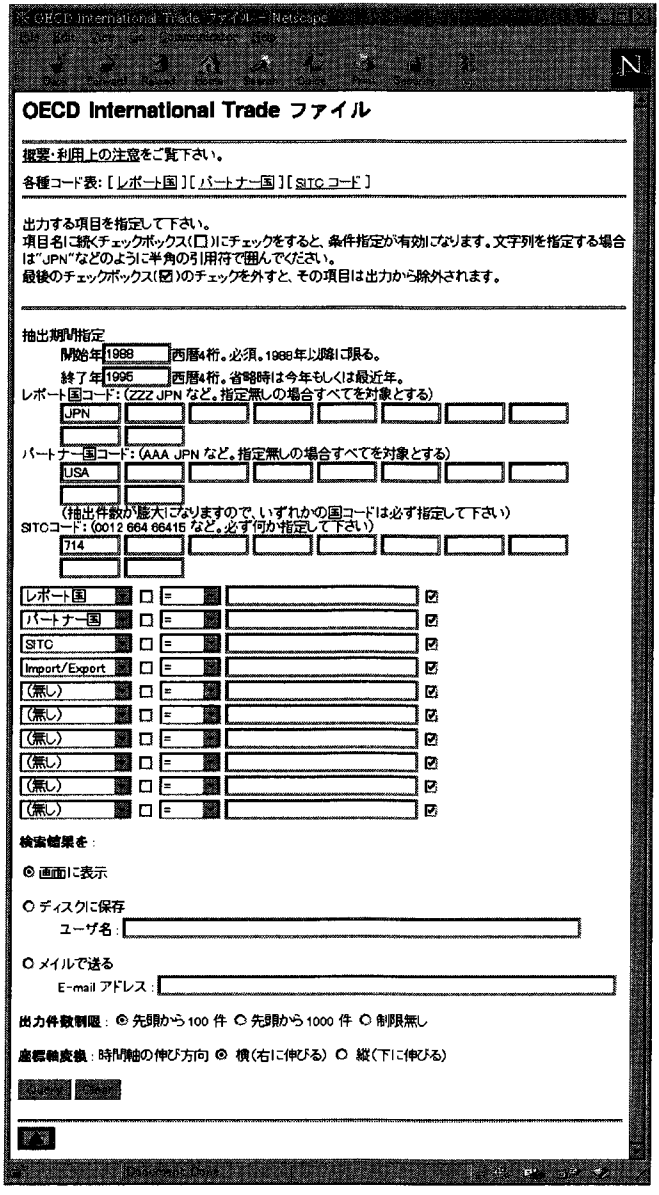

図32: OECD 貿易ファイル WWW インタフェイス（全件版）

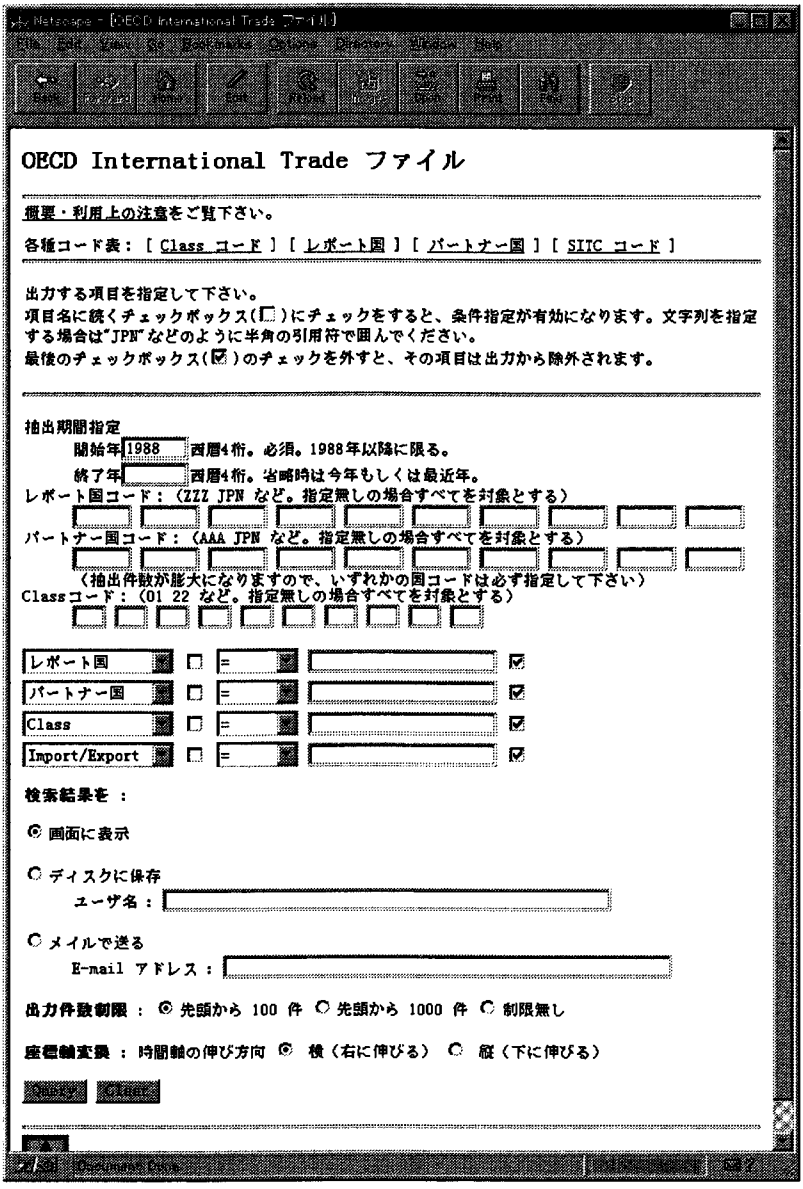

図33：OECD 貿易ファイル WWW インタフェイス(47産業分類版)

2119 Hides and skins, n.e.s waste and used leather

として登録されている。同様に211で始まるデータ，すなわち2110から2119まで（実際のデータは2111, 2112, 2114, 2116, 2117, 2119だけであるが）の合計は，SITCコード211として登録されている。以下，210から219は21に，20から29までは2に，それぞれ集計されて登録されている。全件版では，これらすべての桁数のSITCコードで検索が可能である。したがって，2と指定すれば，SITCの一桁分類の2の産業の貿易額が相手国別に抽出できる。

・出力項目の初期設定

もともと含まれている属性項目が，レポート国，パートナー国，SITCコード，輸出，輸入のキー項目だけなので，出力項目指定にはデフォルトとしてすべてを指定している。出力が不要な場合は項目指定から外しておけばよい。

OECDは，貿易ファイルをCD-ROMで提供し始めてから，保存形式に関する仕様書などが添付されてこない。その代わり，CD-ROMデータを検索するBeyond20/20という検索ソフトウェアが添付されている。これを使えばRIEBデータベース検索システムとは独立にCD-ROMの内容を直接検索・閲覧することができる。

7.3.4　47産業分類

貿易の全体的な流れを見るには，全件データより，ある程度統合したデータのほうが使いやすい場合がある。SITC 1桁またはSITC 2桁のデータを別に用意することも可能である。これは全件検索でもできることであるが，必ずしも，経済分析にそうした分類が適しているとはいえない［阿部］。

そこで［Petri］や［Krause］が定義した表18に示した47産業分類を採用することにした。産業分類コードは01から47までで，これに2つのコードを追加

した。分類コード00はSITCコード全部の合計，具体的には全貿易額となる。分類コード99は，分類01から47までの合計である。00と99は若干の相違がある。

このように産業を統合することにより，全体のデータ件数は1/100にまで小さくなり，その結果検索効率が大幅に高まった。

表18：47産業分類表

Class	SITC	内容	Class	SITC	内容
01	61	皮革	25	761	テレビ
02	63	木材製品	26	762	ラジオ
03	661-3	鉱物製品	27	763	音響製品
04	667	宝石	28	775	家電
05	671	銑鉄	29	78	自動車
06	68	非鉄金属	30	791	鉄道車両
07	65	繊維	31	885	時計
08	664-6	ガラス・陶器類	32	892	印刷物
09	793	造船	33	896-7	アンチーク・宝石加工
10	81	配管用品	34	898	楽器
11	82	家具	35	51	化学製品
12	83	旅行用品	36	52,57,59	その他の化学製品
13	84	衣類	37	54	医薬品
14	85	履物	38	56	化学肥料
15	893	プラスチック製品	39	58	プラスチック
16	894	玩具	40	71-5	非電気機械
17	895	事務用機器	41	764	通信機器
18	899	その他製造業製品	42	77(-775)	家電以外の電気機械(77から775を引いたもの)
19	53	塗料	43	792	航空機
20	55	香水	44	87	科学器具
21	62	ゴム	45	881-4	写真用品
22	64	紙	46	9	その他
23	672-9	鉄鋼	47	0-4	非製造業製品
24	69	金属製品			
			00	Total	全SITC分類の合計
			99	-	クラス分類01-47の合計

表19に47産業に集約したデータ仕様について示す。

表19：47産業分類版データ仕様

件数	40万件（1997年度時点の合計）
容量	約20MB（標準化済み）
1バイト文字コード	ASCII
2バイト文字コード	なし
主キー項目	レポート国，パートナー国，輸出入，Classコード

7.3.5　DBデータの標準化の問題点

OECD貿易ファイルは，比較的扱いやすいデータであった。

1．原データはすべて仕様に準じて記録されており，異常なデータは含まれていない。

2．SITCコードという標準的なコードを使っている。また国コードも標準的なISO 3166-1に準拠した3文字アルファベットコードである。このように極めて標準的なコードを使っているので，抽出データの再利用がしやすい。

OECDは最近までMTで1年のデータのみを提供してきたが，CD-ROMで提供を始めるに際して過去のデータも含めるようになった。

過去のデータも遡及して更新しているようで，今後は国名など主要なキーコードが新しく付け加わったり変更があっても比較的簡単に処理できると思われる。

OECDデータについてもこのように評価できる点ばかりではない。過去のデータ全件を提供するのではないからである。一部のみのアップデートの場合は，既存のデータとの整合性を損ねないようデータを維持していく際注意が必要である。例えば今回データベース化したCD-ROM版では，1988年から1996年の9年間分のデータが当初収録されていた。数ヶ月後にアップデート版とし

て届いたCD-ROMには，レポート国の内更新された国だけについて1988年から1997年のデータが収録されている。

ただRevision 2については1961年から1990年までのHistoric版がある。これと最新のデータを接続して1961年から最近年までの時系列データを作成するのは容易である。OECD側でこのようなHistoric版を随時供給してもらえると便利である。[81] CD-ROMのように大容量メディアでデータが提供される場合には，できる限り過去のデータすべてを含む形で提供される方が，データを維持する側からすればあり難いのである。

7.3.6 検索例題

例題1

日本のASEANへの1980年から現在までのテレビの輸出入額を調べよ。

検索例

テレビというのは47産業分類より細かいので，全件版を使って検索する。

《OECD International Tradeファイル》にリンクされている《レポート国》から日本の国コードを，《パートナー国》からASEAN各国の国コードを調べる。ASEANには現在9ヶ国が加盟しているが，国によって加盟年が違うので，ここではASEANの主要5ヶ国に限定して検索することにする。

《SITCコード》でWebブラウザの検索機能を使い，“television”で検索する。テレビのSITCコードは，カラーテレビ・モノクロテレビ・テレビ(カラーとモノクロを合計したもの)の3つのコードが存在することが分かった。

以上の下調べをした後，検索ページの指定項目に条件を指定していく。「開始年」は現在のところ1988年までということなので，1988を指定する。また，「終了年」は入力をしなければ，最新の年がデフォルトであるので省略する。

---

(81) OECDはCD-ROMによるこの形態でのデータ提供を開始したばかりで，今後どのような供給形態がとられるのかは不明である。

「レポート国コード」に日本の国コードのJPNを入力する。「パートナー国コード」の項目に，ASEAN5の国コード(IDN, MYS, PHL, SGP, THA)をそれぞれ入力する。

「SITCコード」にテレビのSITCコード(761, 7611, 7612)を入力する。「Import/Export」に条件を指定しないことで，輸出入両方のデータを検索できる。

例題2

日本と中国との47産業分類での産業内貿易指数を計算したい。産業分類版がどのように使えるか例示せよ。

検索例

ここでは47産業分類版を使って検索する。

産業内貿易指数を計算するには，部門別の2国間の輸出入額が基礎データとして必要である。

検索に必要なコードは，日本と中国の国コードと，47産業分類されたClassコードである。Classコードについては，全てのコードを検索する場合，コード指定を省略することで，全件を検索することができる。従って，個別にClassコードを調べる必要はない。

「レポート国コード」に日本と中国の国コード(JPN, CHN)を入力し，「パートナー国コード」にも，日本と中国の国コード(JPN, CHN)をそれぞれ入力する。「Classコード」の入力は省略する。

産業内貿易指数の計算については省略する。

7.4　日経総合経済ファイル

7.4.1　概要

日経総合経済ファイル(生産・出荷・在庫統計付き)は，NIKKEI NEEDSとして良く知られた，日本経済新聞社のデータベースの一つのデータで国民経済計算，生産・企業経営，金融・財務など，日本のほとんどのマクロデータを収録したものである。オリジナルなデータは，経済企画庁，大蔵省，通産省等政府発表のもの，業界団体発表のものなどである。

データは毎年MTで過去の年次・年度データも含めて提供される。

図34にWWW検索インタフェイスを示した。

日経総合経済ファイルの仕様は次の通り。

表20: 日経総合経済ファイルデータ仕様

レコード長	960バイト
件数	105731件(1997年)
容量	約100MB(DBデータの標準化後26.7MB)
1バイト文字コード	EBCDIC
2バイト文字コード	JIS(ただしシフトコードなし) 漢字項目中の英数字は2バイト文字コードを利用
主キー項目	MTコード，系列コード

収録項目は典型的な経済統計データであり次のようになっている。

表21: 日経総合経済ファイルの項目仕様

系列	系列コード，系列名，出典名，MTコード
属性	単位，属性，集計コード，更新中止の記号，期種，期種コード
期間	収録開始期，収録終了期，速報開始期，更新日
データ	(整数10桁+小数5桁)

7.4.2 操作方法

基本的な操作方法については，すでに，94ページの**7.1**に詳しく説明した。

日経総合経済ファイルにおける，主要な検索キー項目は以下のようなものである。

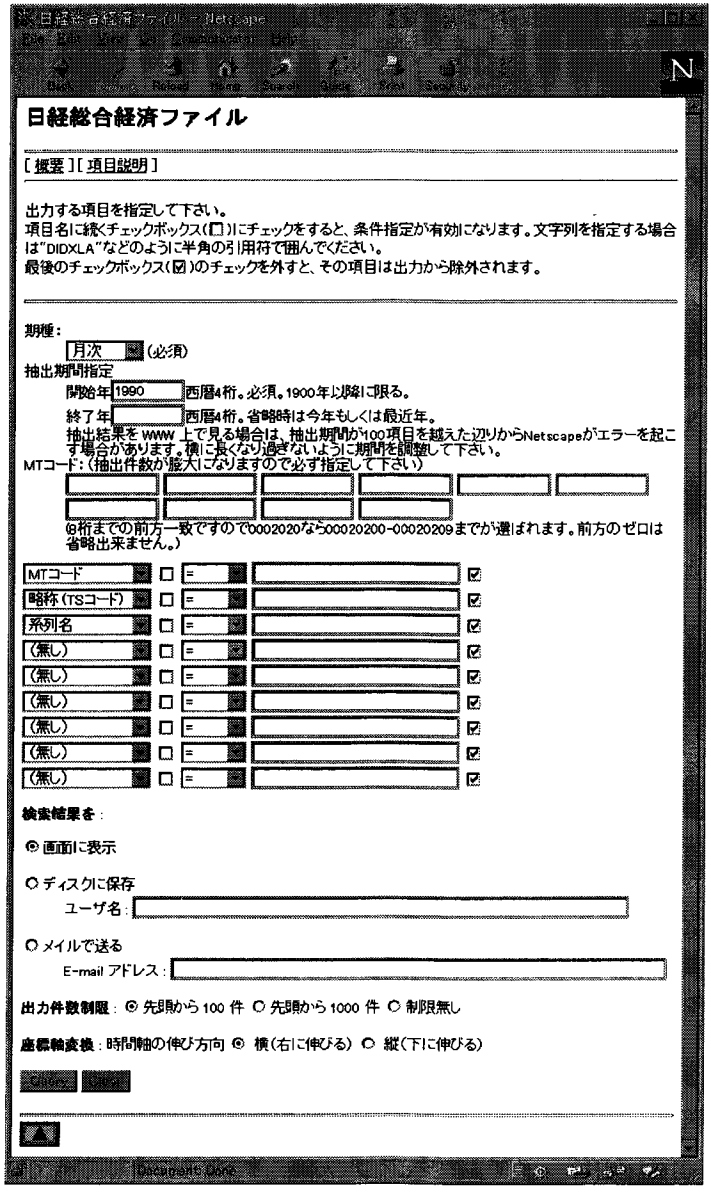

図34：日経総合経済ファイル WWW インタフェイス

表22: 日経総合経済ファイルのキー項目仕様

期種	データの各系列は，月次，四半期，半期，年次を，プルダウンメニューから選択し，指定する。
年	西暦4桁数字。
MT コード	各系列を表す分類コードで，このデータ独自の数字8桁のコードである。指定した数字は常に前方一致で照合される。

MT コードの数は数万件にもなり，その一覧表は印刷物として MT と同時に日本経済新聞社から提供される。これは研究所機械計算室に常備しているので，検索の際にはこれをまず参照されたい。マニュアルなしで検索できるようにするにはこれを WWW に載せる必要があるが，著作権そしてその大きさからして現在のところ見送っている。

7.4.3 DB データの標準化に関しての問題点

日経総合経済ファイルも，OECD 貿易ファイルと同じく，結構扱いやすいデータである。

1．元データはすべて仕様に準じて記録されており，異常なデータは含まれていない。

2．過去のデータも含めて，毎年全件の提供がある。

これによって国名など主要なキーが変わったとしても，これにも自動的に対応できる。決算期が変わることによる，過去のデータの修正についても，正確に行われているようで，それを我々のデータベースに反映するのは簡単である。

3．会社コードなどについては独自ではあるが，他の日本経済新聞社のデータと共通のコードを使っている。

日本経済新聞社は日本の統計データの提供者としては最大のもので，今後，ここで考慮した以外のデータを購入する場合を考えると，独自であっても共通のコードを使っていることは，我々の作業量を軽減することに役立とう。

以上の理由で，これまで，DBデータの標準化に際して，特に大きな問題は発生していない。

フォーマット変更　日本経済新聞社は従来のフォーマットに加えて，新しく日本語の系列名などが追加されたフォーマットによるデータの提供を開始している。現在はまだこれに対応していないが，1998年度のデータ受け入れから対応する予定である。

7.4.4　検索例題

例題

系列名とMT コード名を含めて日本の経常収支の動向を月次で収集可能な限り集めよ。

検索例

日本の経常収支については，日本銀行の国際収支統計月報に載っている。そこで，機械計算室常備のデータ説明書の目次から国際収支統計月報の中の「国際収支」という項目を見つけ，当該ページを見る。すると，国際収支の総括表については，経常収支の合計と貿易・サービス収支，所得収支，経常移転収支という系列があり，貿易・サービス収支は更に貿易収支，サービス収支という系列に，その貿易収支は輸出と輸入に分かれている。MTコードは530800から530808までで，全部で9つの系列になる。また収録されているのは1985年1月以降の月次データである事がわかった。

以上が検索に入る前の下調べで，次に検索ページに指定項目を入力していく。期種は月次にする。期間は，1985年を開始年とする。終了年は指定しない。次にMTコードの指定は前方一致でできるので，0053080とする。こうすれば，もとの6桁の数字のうち最後の1桁は任意であるので，9つの系列すべてが取れる。この際，MTコードと系列名の出力用のチェックボックスをチェックしておく。

7.5 DRI BASIC Economics (米国マクロ経済データ)

7.5.1 概要

DRI BASIC Economics データは，米国DRI/McGraw-Hill社が出版しているデータを日本経済新聞社が再編集して販売しているものである。

データは毎年MTで，過去のデータもすべて含めて提供されている。

図35にWWW検索インタフェイスを示す。

次に仕様を示す。

表23: DRI BASIC Economics データ仕様

レコード長	416バイト
件数	70032件 (1997年度時点の合計)
容量	約30MB (標準化後9.5MB)
1バイト文字コード	EBCDIC (2バイト文字はなし)
主キー項目	系列コード

収録項目は以下の通りである。

表24: DRI BASIC Economics の項目仕様

系列	系列コード，系列名，出典名
属性	単位，属性，集計コード，更新中止の記号，期種，期種コード
期間	収録開始期，収録終了期，更新日
データ	(整数10桁＋小数5桁)

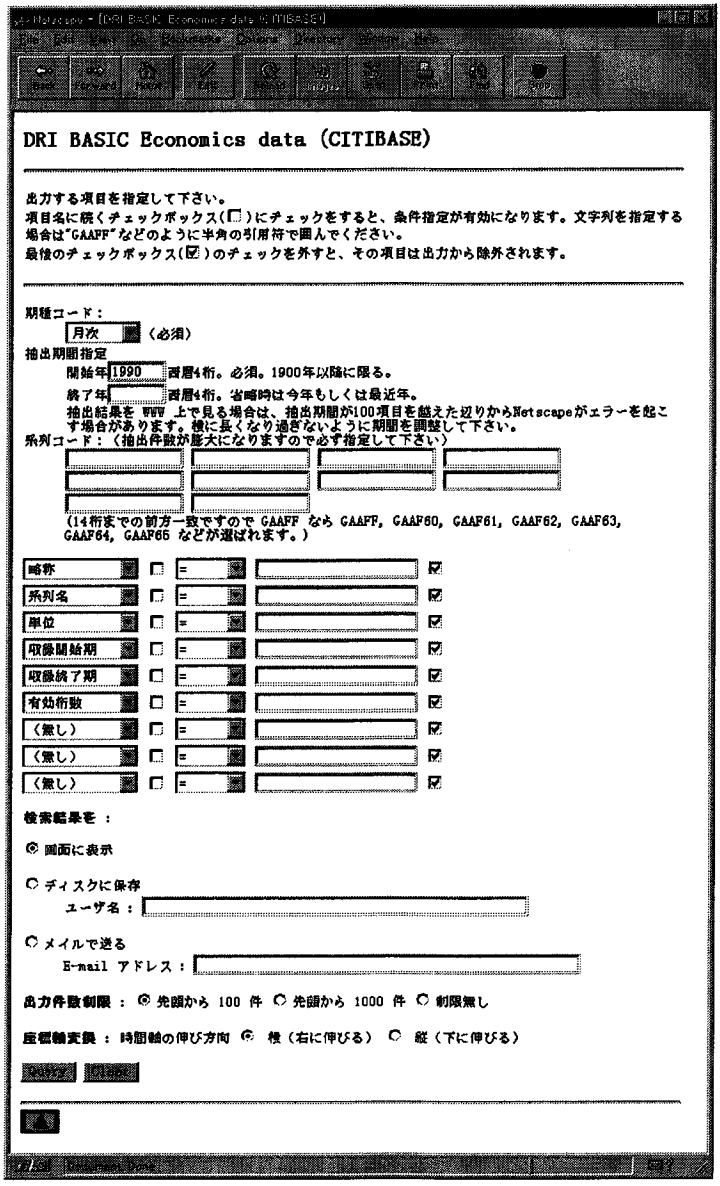

図35: DRI BASIC Economics データ WWW インタフェイス

7.5.2 操作方法

基本的な操作方法については94ページの7.1に詳しく説明したとおりであるので，そちらを参照されたい。

*DRI BASIC Economics*データにおける，主要な検索キー項目は以下のとおりである。

表25: DRI BASIC Economicsのキー項目仕様

期種	データの各系列は，月次，四半期，年次のいずれかの期種で，プルダウンメニューから選択，指定する。
年	西暦4桁数字。
系列コード	各系列を表すコードで，独自の英大文字と数字の組合せによる最大14桁のコードである。指定した英数字は常に前方一致としてマッチングされる。

系列コードの数は数万件にもなり，その一覧表は日本経済新聞社からMTテープと同梱して送られてきている。研究所機械計算室に常備しているので，検索の際にはこれをまず参照されたい。

7.5.3 DBデータの標準化についての問題点

米国マクロ経済データは，他の日本経済新聞社のデータと同じく仕様書どおりに提供されてはきたが，詳しくチェックするといくつかの異常なデータが発見できた。

系列名項目の異常 系列名項目の値（文字列）中の特定位置に，ヌル記号が数バイト含まれているものが44件あった。これらは空白文字に置き換えた。

また，系列名項目の内容が途中で切れているものが多い。項目の長さは仕様上72バイトあるが，最長でも69バイトのものしかない。そして多くの項目は長くても66，67バイト長に収まっており，その中では後ろが切れているように思える項目が多い。以下はその実例である。

```
PCE:NONDUR GODS:PRICE INDEX:BENCHMARK-YRS WGHT(INDEX,87=100)(T7.1)
GROSS DOM PROD:PCE:DURABLE GOODS:FIXED 1987 WGHT(INDEX,87=100)(T7.
IND BUS TAX & NONTAX LIAB+TRAS PAY-SUB+SURP GOVT ENTERP(BIL.1987$)(
LOANS & SEC @ ALL COML BANKS:C&I LOANS TO NON-U.S.ADDRESSEES(BIL$,SA)
```

最初の1行は66バイト目でうまく終っている例である。次の行は同じ66バイト長であるが，括弧が明らかに閉じていない。次の行も同じである。仕様を100バイトないし200バイトにしてもフルな情報を与えてもらう方が，ユーザーの立場からすると有り難い。

単位項目　このデータには単位を表現する項目があり，そこには**MIL.$,NSA**や**1987＝100,SA**といった記述が含まれているべきものである。

問題は空白のデータが多くまた項目名自体に単位が含まれ，単位項目に示された単位と異なる場合があることである。例えば，

略称	- 系列名	- 単位
GANFRQ	- PCE:NET FOREIGN REMITTANCES (BIL.87$)	- BIL. 87$, SAAR
GANMO	- PURCHASES OF PRIVATE STRUCTURES - OTHER MINING	- BIL. 92$, SAAR

また，略称WTNOQ（月次データ）で以下のようなものがある。

略　称: WTNOQ
系列名: SALES, BUSINESS, MERCHANT WHOLESALES, OTHER NONDURABLES; MIL OF C
単　位: BIL.C92 $, SA MR

系列名項目の後半に，不完全な単位名でMIL OF Cとmillion of constant 92 pricesであろうと思われる記述があるが，単位項目の値はBIL.C92 $, SA MR (constant 92 price billion dollars, seasonally adjusted (SA) by moving average method (MR)) である。[82]

(82)　執筆時点でこれらの点についは日経に問い合わせ，回答待ちの状態である。

すべてをそのまま標準化したデータファイルに保存することにするが，DRI データに関しては単位情報は系列名項目と単位項目に分散して載っており，多くはないが一見矛盾があるように思えるケースも含まれていることに注意しておく必要があろう。

採録開始時期，終了時期 採録開始時期と，採録終了時期の前後関係がおかしなデータがある。PRM，PRM1 - PRM17 までの18件であり，開始が8301終了が8001である。(つまり1983年に開始して，1980年に終了したことになる。) このPRM で始まる系列名のものは他にPRM18があるが，これについては開始が8301で終了が9901すなわち未来のデータとなっている。すべて更新日は970729である。

そしてこれらすべてのデータには時系列の値が入っている。対処の方法がないのでこのまま保存する。

7.5.4 検索例題

例題

四半期データで，1990年代のアメリカの失業率の動向を示せ。

検索例

機械計算室に常備してある DRI BASIC Economics Data Dictionary のコード表を使って，失業率 (Unemployment rate) のコードを探す。系列コードは LHURR であるが，この系列は月次コードしかない。

以上が検索に入る前の下調べで，次に検索ページに指定項目を入力していく。期種は月次にする。期間は，1990年を開始年として，終了年は指定しない。系列コードはLHURR を入力する。

これを四半期に変換する等の方法については省略。

7.6 IMF Economic Information System データ

7.6.1 概要

IMF Economic Information System データは，IMF (International Monetary Fund) のEIS (Economic Information System) が提供する経済統計データであり，*International Financial Statistics* (*IFS*), *Direction of Trade Statistics* (*DOTS*), *Balance of Payments Statistics Yearbook* (*BOPSY*), *Government Finance Statistics Yearbook* (*GFSY*) の四つからなる。

データは毎年，過去のデータもすべて含めて，MTで提供されている。

図36，37，38，39にWWW検索インタフェイスを，それぞれ示す。

仕様，収録項目などは基本的に共通で，次に仕様を示す。

表26: IMF データ仕様

レコード長	88バイト
件数	334378件 (1998年度時点の合計)
容量	約223MB (DB データの標準化後232 MB)
1バイト文字コード	EBCDIC (2バイト文字はなし)
主キー項目	年度，国コード，Subject コード

収録項目は以下の通りである。

表27: IMF データの項目仕様

キー	年度，国コード，国名，Subject コード，パートナー国コード
属性	単位，データソースコード，バージョンコード，項目内容，期種
データ	整数6桁

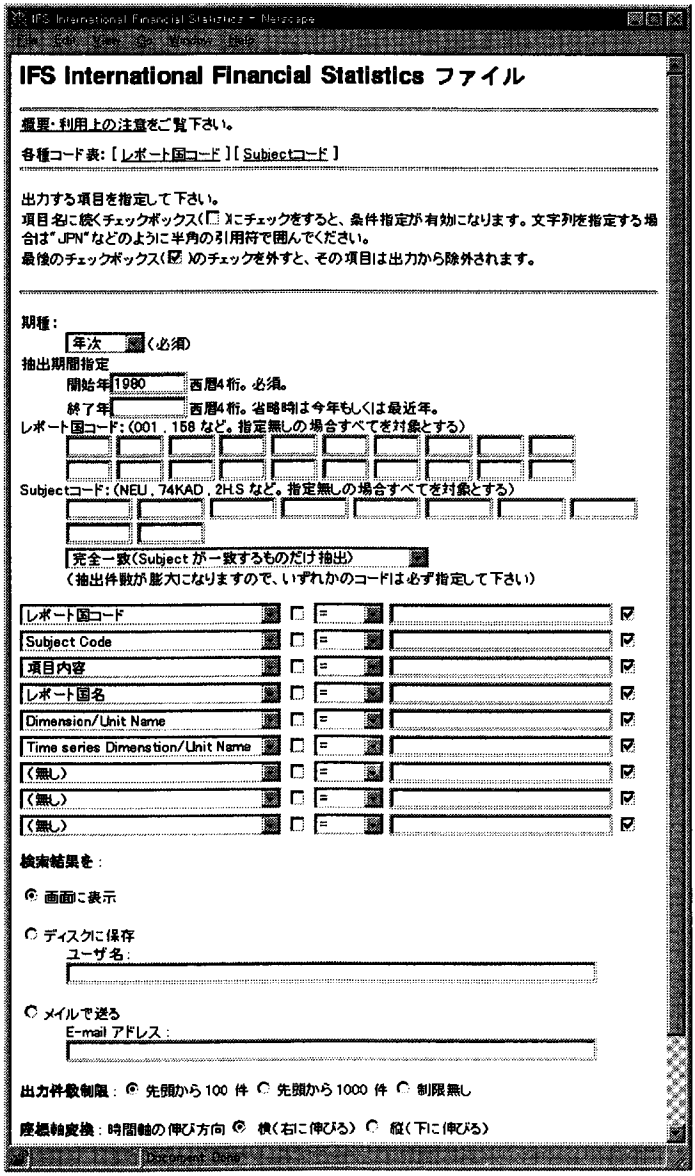

図36: IFS データ WWW インタフェイス

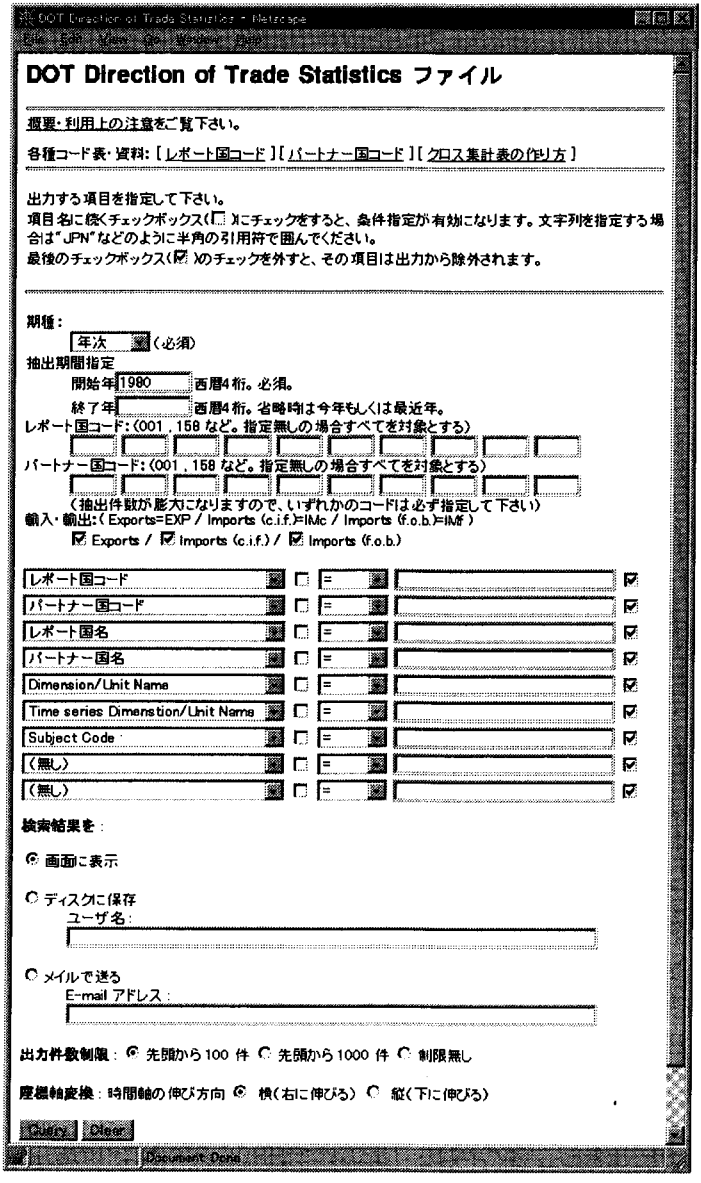

図37: DOTSデータ WWW インタフェイス

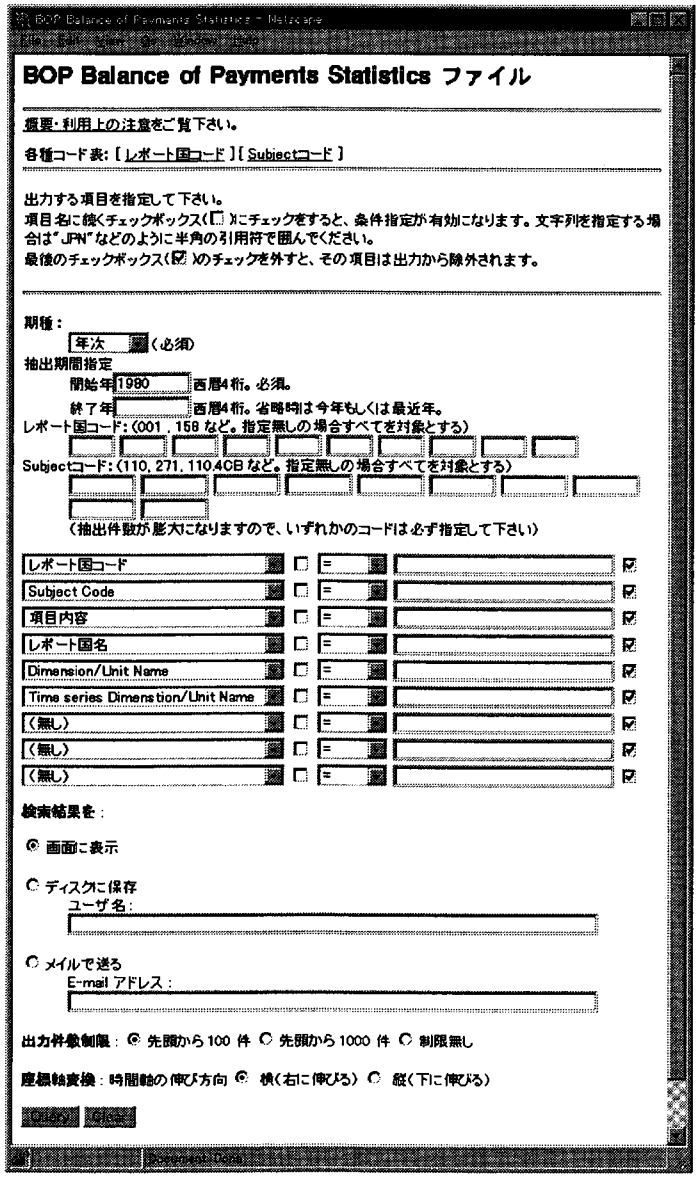

図38: BOPSY データ WWW インタフェイス

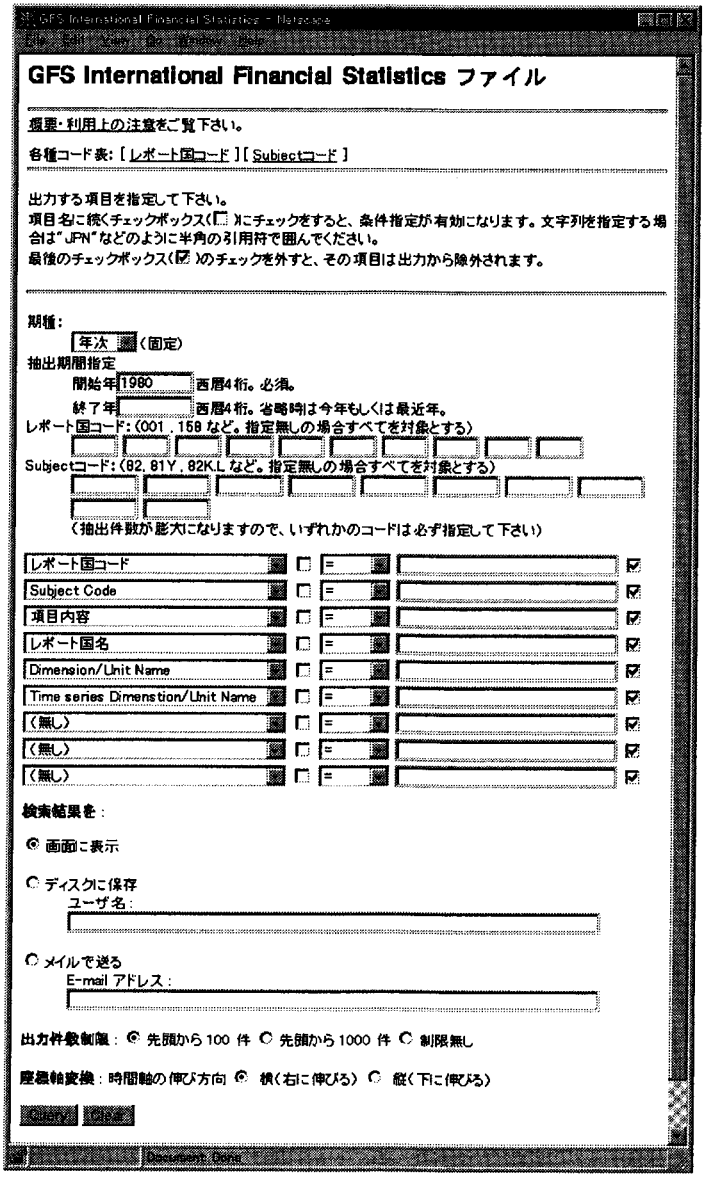

図39: GFSY データ WWW インタフェイス

7.6.2 操作方法

基本的な操作方法には94ページの7.1と同様であるので，参照されたい。

IMF データにおける，主要な検索キー項目は以下のようなものである。

表28: IMF データのキー項目仕様

期種	データの各系列は，月次，四半期，年次，の期種についてあり，どの期種かを，プルダウンメニューで選択して指定する。データによっては四半期，月次がないものもある。
年	西暦4桁数字。
レポート国コード	数字3桁の独自コードである。
Subject コード	*IFS, DOTS, BOPSY, GFSY* それぞれで異なる系列コードである。英数字6桁以内の独自のコードである。

国コード，Subject コードは，それぞれの検索ページ上のリンクから一覧を得ることができる。

- Subject コードは，それぞれの検索ページ上のリンクから一覧を得ることができるが，各データごとに出版物として *YEARBOOK* があるので，参照するにはこちらの方が詳しくて便利であろう。
- *IFS* に関しては，Subject コードの指定が前方一致か，完全一致かのどちからをプルダウンメニューを選択することで指定できる。
- *DOTS* データの場合は，Subject コードの代わりにパートナー国コードがキーコードとなる。

  また，*DOTS* データを元にトレード・マトリックス（クロス集計表）を作成することが多いと考えられるので，Excel を使ったクロス集計表の作成方法を，検索ページ上のリンク先に示してある。

  DOTS においては，輸出，輸入の区別も重要なので，輸出・輸入チェックボックスでチェックし出力に加えることができる。
- *GFSY* データについて

  GFSY データに関しては，Subject コード自体を検索する手順が複雑であ

る。IMFデータの仕様書の*GFSY*に関する表と，*YEARBOOK*の両方を参照し作業する必要がある。それについては検索ページ上に利用上の注意としてまとめてあるので参照されたい。

7.6.3　DBデータの標準化の問題点

IMFデータは，データの異常とはいえないまでも，整合性があるといえない場合が結構あった。

データの欠落と集計　例えば四半期データにおいて，データの欠落がある。ある*IFS*データ（Subject-Code: 12a/Country: BELGIUM(124)/項目内容: CLAIMS ON GOVERNMENT）の事例を見てみよう。

このデータレコードの不連続部分は次のようになっている。

year	Q1	Q2	Q3	Q4
1979	000790	000790	000790	000790
1980	000800	000800	000800	000800
1981				002410
1982				002750

つまり81年のQ1, Q2, Q3データは欠落し，Q4だけが提供されている。しかしその値に注目すると，79，80年のそれぞれ四半期の値に比べて，明らかに大きい。ちょうど4倍ほどにもなる。91年から再び四半期の値がすべて埋まった形で提供されているが，値をみると確かに四半期のものと分かる。四半期のデータがなく，年データを四半期データシリーズにQ4として埋めたようだが，その旨，説明が欲しいところである。年データで確かめると，1981年が241.6，1982年が275.5となっていて，数値はあっている。

今回の標準化に際しては，このままの形で収録してあるので，使用にあたっては注意が必要である。

Break/footnote Indicator　MT上では1行あたり12項目並んでいる各データの時系列数値の部分には，各値ごとにBreak/Footnote Indicatorと呼ばれる

属性が付与されている。例えば,

year	A	M1	A	M2	A	M3	A	M3	A	M4	A	M5
1992	U	617440	U	603530	U	590250	U	585660		581470		565910

のように，各数値にUなどの属性が付与されている。仕様書によればUは“Indicates that original data value is not to be overlaid with subsequent calculated data”である。他に例えばXは“Identifies data observations estimted from lower-frequency data by extrapolation routines”などがあり，該当数値の出所，正確さなどについての但し書きが含まれている。

これらも含めて，*YEARBOOK*にはコメントが詳細に述べられており，これらを参照せずに数値だけをそのまま処理すると，正確さに欠ける可能性がある。

XやUといったBreak/Footnote Indicatorは一つのデータ全体に付いている情報ではなく，各データポイントに個別に振られた情報であるので，期間分の情報をどうやって反映させるか，うまい方法が見つからず，データベースには収録しないことにした。ユーザー側でXやUであろうと思われるデータは*YEARBOOK*等で注意して使われたい。

キー項目　キーとして利用されるのはSubjectコードと呼ばれるもので，データ中には5バイトで保存されている（以下Data Subject）。しかしデータ中のSubjectコードは，各データの*YEARBOOK*においてインデックス的役割を果たしている項目（以下Book Subject）と，そのバイト並び順が異なっている。

例えば*IFS*の場合であれば，Book Subjectの並び順は，Data Subjectの1-3-4-5-2バイト順となる。Data Subjectで空白だったバイトは「.」（ドット）で置換される。但し前方，後方のドットは省略され，途中のドットだけがそのまま残される。以下に*IFS*におけるBook SubjectとData Subjectの例を示す。（空白は△で表記した。）

表29: Subjectコードの例

Book Subject	Data Subject	English Description
2EB	△△2EB	OUTSTANDING LOANS: SAF, ESAF, TF
2EU	△△2EU	OUTSTANDING SFF PURCHASES
1A.S	△S1A△	GOLD
1AMS	△S1AM	GOLD AT MARKET PRICES

一般に利用者は，*YEARBOOK*に記されたBook Subjectでデータを扱っているため，上記のルールによってData SubjectからBook Subjectを変換・合成し，提供することとした。但し変換ルールであるバイト並び順は，それぞれのデータによって異なるので注意が必要である。以下に要約を示す。

表30: Subjectコード変換ルール

IFS, GFSY	1-3-4-5-2の順にして前後の空白は除去，途中の空白は．に変更。但し*GFSY*のコード参照ルールは複雑であり，このコードは*YEARBOOK*には直接記載されていない。
BOPSY	1-3-4の順に並べ後ろに5-2と，別の項目の1バイトをつける。
DOTS	輸出入のフラッグとして利用しているため，Book Codeとして変換はせず，"7D0△△" = EXP，"7D1△△" = IMC (c.i.f.)，"7D1△V" = IMF (f.o.b.)とした。

項目説明のタイプミスとばらつき　データには各項目の英文による項目説明が含まれている。しかしその記述はタイプミスを含めてばらつきが多く，統一的に扱えない。例えば"OFFICIAL RATE"となるべきデータの幾つかのO(オー)は0(ゼロ)に間違えられており，"0FFICIAL RATE"となっている。

他にも綴りの混乱が多く含まれている。例えば*IFS*の，ある系列には，32種類もの異なる綴りが存在する。入力する際の単純な誤りと思われるが，人間の目には同じに映ってもコンピュータは異なると認識してしまうのでこのあたりもIMF側で整合性チェックを行って欲しいところである。以下に例(一部)を示す。

EFF OUTSTANDING
EFF:OUTSTANDING
EXTENDED FACILITY OUTSTANDING
EXTENDED FUND FACILITY

7.6.4 CD-ROM化について

IMFデータも従来オープンリールテープによって提供されてきたが，1999年の4月以降からは，CD-ROMに変更する旨の予告があった(1998年9月)。1998年11月現在，移行措置としてテープとCD-ROMの両方によるデータの提供が行われている。このCD-ROMには三つの異なる形式のデータが含まれている。

1．従来のテープ版のフォーマットそのままのデータファイル。
2．従来のテープ版とは異なる新フォーマットのデータファイル。浮動小数点形式を採用するなど，従来版フォーマットの欠点が幾つか改善されている。
3．Mircosoft Accessのデータベースファイル形式によるもの。新フォーマットのデータをそのままAccess化している。

予告によると，今後データはMicrosoft Accessのデータベースファイル形式のみで提供されるとある。CD-ROMには上記のとおり新旧二種類のオープンな仕様の順ファイルが含まれているが，これらは移行措置によるものであって，1999年4月以降は提供されない。Accessフォーマットへの移行に際して，2桁しか取られていなかった西暦情報が2000年問題[83]に対応するために4桁になったり，数値の表現が通貨単位を固定した整数形式だったのを浮動小数点形式とすることで，急激なインフレなどによって過去のデータが全てゼロとなる

(83) 西暦が下2桁で扱われていたために，西暦2000年を1900年と計算機が誤解して起きるトラブル。

問題を解決した点は評価できる。しかし，これによってOECD貿易ファイルで経験したのと同じく，データ全件を容易に取り出すことができなくなる可能性は残されている。Accessも，OECD貿易ファイルの検索ソフトウェアBeyond 20/20と同じく，データベースのデータをTAB形式なり，CSV形式なりで出力する機能を持っている。それほどの困難もなく，全件データを取り出せるとは考えられるが，MTの場合以上に手間と時間がかかることは間違いない。IMFがAccessを用いた理由も，単にそれが現在最も普及しているパソコン用RDBソフトであるためと推測される。つまり，将来に新しいソフトウェアが普及するたびにそのソフトに切り替わる可能性が高く，そこで常に全件データを取り出す機能が提供されるとは限らない。我々としてはむしろ新フォーマットのデータをそのままCD-ROMで提供し続けるもらう方が有難いが，1998年11月現在そのような計画は発表されていない。

7.6.5　検索例題

例題1

1970年，1980年，1990年，最近年のASEAN5ヶ国（フィリピン，タイ，シンガポール，インドネシア，マレーシア）の域内貿易依存度を計算したい。これに必要なデータを抽出せよ。

検索例

域内貿易依存度の計算には*DOTS*のデータでASEAN各国のものとそれ以外の国のものが必要となる。対世界のデータがあればそこからASEAN各国の貿易額を引くことにより，残余の世界との貿易を計算できる。したがって，抽出すべきデータはASEAN各国と対ASEAN，対世界の貿易額ということになる。ASEANには現在9ヶ国が加盟しているが，国によって加盟年が違うので，今回はASEAN5に限定して検索することにする。レポート国は5ヶ国分，パートナー国についてはこれらに加えてWorldのコードも入力する。コードはそ

れぞれ《レポート国コード》《パートナー国コード》ページからブラウザの検索機能を使って調べておく（フィリピン566，タイ578，シンガポール576，インドネシア536，マレーシア548，World 001）。

以上が検索に入る前の下調べで，次に検索ページに指定項目を入力していく。まず，期種を年次にする。開始年と終了年は，1970年から1970年（つまり1年分）と，1980年から1980年，1990年から1990年，最近年から最近年までの4回検索する。レポート国とパートナー国のコードを入力する。輸出と輸入のチェックボックスはすべてオンにしておく。検索結果をExcelを用いて必要なところだけを1つの表に作り直して計算に使う。域内貿易依存度の計算については省略。

例題2

アジア通貨危機の発生前後の関係諸国（タイ，インドネシア，韓国）の為替レートの変動をその変数名の正確な定義とともに抽出せよ。

検索例

アジア通貨危機は1997年7月に発生した。レポート国コードは《レポート国コード》ページからブラウザの検索機能を使って調べておく（タイ578，インドネシア536，韓国542）。*IFS YEARBOOK* によれば，為替レートに関するSubjectコードはaeがper US Dollar,End of Periodなので，このコードを利用する。

以上が検索に入る前の下調べで，次に検索ページに指定項目を入力していく。期種は月次にする。開始年は1997年，終了年は1998年とする。レポート国コードを3ヶ国分入力する。Subjectコードはaeを入力する。条件はデフォルトである，完全一致のままにしておく。変数名は，出力項目指定で「項目内容」を選択すればよいが，これもデフォルトで指定されている。

7.7　興銀財務データファイル

7.7.1　概要

興銀財務データファイルは，日本興業銀行および興銀データサービスが収集した上場企業の財務データであり，データソースは有価証券報告書である。

データは毎年過去10年分のデータがMTで提供される。決算期の相違を勘案して年調整したものと，原系列の2つがある。

このデータについては，DBデータの標準化の作業中に各種のデータの不整合，仕様書との不一致を発見したため，日本興業銀行の担当者と相談しながら標準化を進めていた。その作業中に，日本興業銀行から1999年以降は，東洋経済新報社に委託して新しいフォーマットでデータ提供を再開する予定とのことでこの作業はとりあえず中止することとした。

財務データに関しては，これを機会に日本経済新聞社のものに切替えていく予定である。

標準化は完成するに至らなかったが，この過程で幾つかの問題点が明らかになった。一つの標準化のケース・スタディとして，決算期変更に対する年換算について説明しておこう。

7.7.2　年換算ロジック

興銀財務データは，決算期に関する年度補正を施したデータと，そうでないデータの二種類のフォーマットで提供される。この年換算作業に関しては，仕様書には単に以下のように記されている。

> 「期間中に変則決算が行なわれた場合，最近期のデータについて損益計算書項目は，営業月数によって12か月に換算して年データとする。貸借対照表項目は，そのまま使用します。」

ところが，変則決算の，つまり決算期の変更が行なわれる場合の実際の換算

ルールは以下の通りである。

1．損益計算書項目については，

(a) 変更期の最近データについてだけ月数によって12か月分を倍率化して算出。

(b) 変更期以降のデータについては何もしない。

(c) 変更期以前のデータについては実質的にその前年度データとして挙げられていた数字を採用する。

2．貸借対照表項目については，

(a) 変更期以降のデータについては何もしない。

(b) 変更期と，それ以前のデータについては実質的にその前年度データとして挙げられていた数字を採用する（変更期の換算なしのデータは消失する）。

3．1．2．以外の項目については，2．の場合に同じ

以下に，ある会社の項目コード110000（貸借対照表項目）をとりあげてサンプルを示そう。この会社では1992年に決算期が9月から3月に変更されており，具体的には1991年9月の次のデータは1992年3月のものである。換算データの1991年データ（49134）は，換算前の1990年データ（49134）であることがわかる。この会社は更にさかのぼると1978年に9月，1963年に12月に決算期が変更されている。

年換算なし

採録年月	198709	198809	198909	199009	199109	199203	199303	199403	199503	199603
データ	33192	40690	42650	49134	54057	66287	77839	73160	61517	62792

年換算あり

採録年月	198703	198803	198903	199003	199103	199203	199303	199403	199503	199603
データ	43625	33192	40690	42650	49134	66287	77839	73160	61517	62792

7.7.3　年換算データの時系列的不連続性

例えばまた別の会社の項目コード110000データについて，1996年に提供されたテープと，1997年に提供されたテープの二つを比較する。この会社では1996年から決算期が9月から3月に変更になった。そのために97年版のテープでは，年換算ありデータにおいて，前年の決算期の情報がずれ込んできている。結果，換算なしデータに見られる199509決算の売上高(87068)が消えている。

96年テープ(決算期変更がなかったので換算ありなしとも同値)
採録年月 198609:198709:198809:198909:199009:199109:199209:199309:199409:199509
換算なし 36609 :44029 :52477 :48374 :51393 :64472 :65667 :73083 :95952 :87068
採録年月 198609:198709:198809:198909:199009:199109:199209:199309:199409:199509
換算あり 36609 :44029 :52477 :48374 :51393 :64472 :65667 :73083 :95952 :87068

97年テープ(決算期変更で，換算ありデータは1996以外，前年の値がずれ込み)
採録年月 198709:198809:198909:199009:199109:199209:199309:199409:199509:199603
換算なし 44029 :52477 :48374 :51393 :64472 :65667 :73083 :95952 :87068 :99361
採録年月 198703:198803:198903:199003:199103:199203:199303:199403:199503:199603
換算あり 36609 :44029 :52477 :48374 :51393 :64472 :65667 :73083 :95952 :99361

テープには毎年最近10年間分のデータが含まれて提供されるが，この換算があるために換算付のデータを使って10年分以上のデータを簡単に作成することができなくなっている。

例えば上記の1997年テープに含まれる1987-1996年データと，1987年テープに含まれる1977-1986年データを単純につなぎ合わせた場合，1987年テープにおいては決算期の変更による換算を受けておらず，1997年テープは換算済みであるため，つぎめにあたる，未補正の1986年データと換算済み1987年データが同値(36609)となってしまう。

これを修正するには，過去データにおいても決算期変更換算を遡及するほかないが，このままの換算ロジックでは以前に決算期の変更が行なわれていた場合，二年度分ずれ込むことになる。つまるところ換算なしデータを元にして時系列データを用意し，独自のロジックによって換算作業を行なうほかないので

ある。

年換算などの補正処置が行なわれた時系列データを安易に蓄積すると，かえって問題となる場合があることを示している。

第Ⅲ部

テクニカル・ノート

第8章　プログラム，HTMLなどのソース

8.1　実際のプログラムの構成

実際に検索に使っているプログラムを具体的に以下に示そう。そこにあげた例は，orasel用にはOECD貿易ファイルのプログラム，dbsel用には東洋経済多国籍企業データと時系列データ処理用の一例として日経総合経済ファイルのプログラムである。データベースに応じて，若干の修正が必要であるが，これら代表的なプログラムを見れば，容易にどのようなプログラムで動いているかが分かるであろう。

oraselプログラム群，dbselプログラム群，汎用補助Cプログラム群の三つに大きく分けて以下に示す。

・oraselプログラム群
  - HTMLファイル
    *oecd-j.html
      OECD貿易ファイルWWWインタフェイス（HTML）
  - CGIプログラム（シェルスクリプト）
    *oecd-query.cgi
      WWWインタフェイスで設定されたパラメタを検索プログラムに使えるオプション列に変換。orasel, orasel_web, orasel_mail, orasel_saveを呼び出す。
  - 検索プログラム群（シェルスクリプト）
    *orasel
      Oracle RDBによるデータベースから，TABセパレートな形式でデー

タを切り出す汎用的検索プログラム。orasel_r.sh を呼び出す。

＊orasel_r.sh

Oracle が動作しているデータベースサーバーマシン上で，Oracle RDB に標準的な SQLPlus ユーティリティを起動する。

＊orasel_web

orasel を使って取り出した結果を，HTML 形式に変換して出力する。

＊orasel_mail

orasel を使って取り出した結果を，TAB セパレートの Excel ファイルに変換し，メイルへの添付ファイルとして送信する。

＊orasel_save

orasel を使って取り出した結果を，TAB セパレートの Excel ファイルに変換し，ディスクに書く。

- 補助 C プログラム群（C 言語）

＊pickhead.c

定義ファイルから各種の情報を引き出す。例えば項目名（Ex. ccode）から見出し（Ex. “会社コード”）を得る。

・dbsel プログラム群

- HTML ファイル

＊touyou-j.html

多国籍企業データ WWW インタフェイス（HTML）

- CGI プログラム（シェルスクリプト）

＊touyou-query.cgi

WWW インタフェイスで設定されたパラメタを検索プログラムに使えるオプション列に変換。dbsel, dbsel_web, dbsel_mail, dbsel_save を呼び出す。

- 検索プログラム群（シェルスクリプト）

＊dbsel

SQL的な表現でTABセパレートなデータから指定のデータを切り出す汎用的検索プログラム。

＊dbsel_web

dbselを使って取り出した結果を，HTML形式に変換して出力する。

＊dbsel_mail

dbselを使って取り出した結果を，TABセパレートのExcelファイルに変換し，メイルへの添付ファイルとして送信する。

＊dbsel_save

dbselを使って取り出した結果を，TABセパレートのExcelファイルに変換し，ディスクに書く。

- 補助Cプログラム群（C言語）

＊pickitem.c

定義ファイルから各種の情報を引き出す。例えば項目名（Ex. ccode）から見出し（Ex. “会社コード”）や項目番号(2)を得る。

＊trim.c

出力されたデータから指定範囲の期間データのみを抜き出す。

＊calcdnum.c

指定された期間情報から，該当の項目の先頭からの位置（番号）を得る。

・汎用補助Cプログラム群（C言語）

- common.h

自製のC用標準ライブラリ。（各種ツール）

- printhead.c

項目位置（番号）から，当該期間を示す文字列（199701など）を出力する。

- rmatrix.c

出力表の形式を90度回転させる。

8.2 orasel プログラム群

8.2.1 HTML ファイル

oecd-j.html

OECD 貿易ファイル WWW インタフェイス (HTML)

```
<head>
<title> OECD International Trade ファイル </title>
</head>
<body> <!—bgcolor="ffffff">
<h2> OECD International Tradeファイル </h2>
<hr>
<a href="abstract-j.html"> 概要・利用上の注意 </a> をご覧下さい。<p>
各種コード表 :
[<a href="repcode.txt"> レポート国 </a>]
[<a href="country.txt"> パートナー国 </a>]
[<a href="sitc.txt">SITC コード </a>]
<hr>
<form action="/cgi-bin/uncgi/oecd-query.cgi">
出力する項目を指定して下さい。<br>
項目名に続くチェックボックス (<input type="checkbox" name="dummy 1">) にチェックをすると，条件指定が有効になります。文字列を指定する場合は " JPN" などのように半角の引用符で囲んでください。<br>
最後のチェックボックス (<input type="checkbox" name="dummy2" checked>) のチェックを外すと，その項目は出力から除外されます。<p>
<hr>
<dl>
  <dt> 抽出期間指定
  <dd> 開始年 <input name="from" size=6 value="1988"> 西暦 4 桁。必須。1988年以降に限る。
  <dd> 終了年 <input name="until" size=6 value=""> 西暦 4 桁。省略時は今年もしくは最近年。<br>
  <dt> レポート国コード: (ZZZ JPN など。指定無しの場合すべてを対象とする)
```

```
  <dd>
      <input name="rep1" size=5> <input name="rep2" size=5>
      <input name="rep3" size=5> <input name="rep4" size=5>
      <input name="rep5" size=5> <input name="rep6" size=5>
      <input name="rep7" size=5> <input name="rep8" size=5>
      <input name="rep9" size=5> <input name="rep10" size=5> <br>
  <dt> パートナー国コード:(AAA JPN など。指定無しの場合すべてを対象とする)
  <dd>
      <input name="par1" size=5> <input name="par2" size=5>
      <input name="par3" size=5> <input name="par4" size=5>
      <input name="par5" size=5> <input name="par6" size=5>
      <input name="par7" size=5> <input name="par8" size=5>
      <input name="par9" size=5> <input name="par10" size=5> <br>
  (抽出件数が膨大になりますので，いずれかの国コードは必ず指定して下さい)
<br>
  <dt> SITC コード: (0012 664 66415など。必ず何か指定して下さい)
  <dd>
      <input name="sitc1" size=5> <input name="sitc2" size=5>
      <input name="sitc3" size=5> <input name="sitc4" size=5>
      <input name="sitc5" size=5> <input name="sitc6" size=5>
      <input name="sitc7" size=5> <input name="sitc8" size=5>
      <input name="sitc9" size=5> <input name="sitc10" size=5> <br>
</dl>
<select name="arg1">
      <option value=""> (無し)
      <option value="repcode" selected> レポート国
      <option value="parcode"> パートナー国
      <option value="sitc"> SITC
      <option value="ie"> Import/Export
</select>
<input type="checkbox" name="arg1c">
<select name="arg1f">
      <option value="=" selected> =
      <option value="!="> not=
      <option value="&gt;"> &gt;
      <option value="&lt;"> &lt;
</select>
<input name="arg1v" size=20>
<input type="checkbox" name="arg1o" checked>
<br>
```

```
<select name="arg2">
      <option value=""> (無し)
―――――(省略)――――――――――
<input name="arg10v" size=20>
<input type="checkbox" name="arg10o" checked>
<br>
<br>
<p>
<b> 検索結果を </b>: <p>
<dl>
<dt> <input type="radio" name="type" value="html" checked> 画面に表示
<p>
<dt> <input type="radio" name="type" value="save"> ディスクに保存
<dd> ユーザー名: <input name="user" size=50>
<p>
<dt> <input type="radio" name="type" value="mail"> メイルで送る
<dd> E-mail アドレス: <input name="email" size=50>
</dl>
<b> 出力件数制限 </b>:
<input type="radio" name="limit" value="100" checked> 先頭から100件
<input type="radio" name="limit" value="1000"> 先頭から 1000件
<input type="radio" name="limit" value="all"> 制限無し
<p>
<b>座標軸変換</b>:
時間軸の伸び方向
<input type="radio" name="reverse" value="n" checked> 横 (右に伸びる)
<input type="radio" name="reverse" value="r"> 縦 (下に伸びる)
<p>
<input type="submit" value="Query"> <input type="reset" value= "Clear">
</form>
<hr>
<a href="../index-j.html"> <img alt="back" src="/icon/back.gif">
</a>
</body>
```

8.2.2 CGI プログラム (シェルスクリプト)

oecd-query.cgi WWW インタフェイスで設定されたパラメタを検索プログラムに使えるオプション列に変換。orasel, orasel_web, orasel_mail, orasel_

save を呼び出す。

```
#! /bin/sh
echo Content-type: text/html
echo
TR=/usr/ucb/tr
NKF=/NF/local/Solaris2J/bin/nkf
Q_SEL=/NF/local/comp/bin/orasel
Q_SHOW=/NF/local/comp/bin/orasel_web
Q_SAVE=/NF/local/comp/bin/orasel_save
Q_MAIL=/NF/local/comp/bin/orasel_mail
CALCDNUM=/NF/local/comp/bin/calcdnum
TRIM=/NF/local/comp/bin/trim
PRINTHEAD=/NF/local/comp/bin/printhead
RMATRIX=/NF/local/comp/bin/rmatrix
DBY=rieb.oecd_c
dbhead=/NF/local/comp/lib/oecd/query/dbhead
TMPF0=/home/tmp/oecdsel0_$$
TMPF=/home/tmp/oecdsel_$$
LASTYEAR=1995# データを更新した時にはあわせて修正せよ
cond=
item=
WWW_termc="4" # 年次
fac=01
db=$DBY
if ["$WWW_from" -lt 1985]; then
  echo' 開始年指定が無い，もしくは古過ぎます。'
  exit
fi
dfrom=$WWW_from
if ["$WWW_until" -gt 1995]; then
  echo' 終了年指定が新し過ぎます。'
  exit
fi
if [-z "$WWW_until"]; then
  u=$LASTYEAR
fi
if [-n "$WWW_until"]; then
  u="$WWWuntil"
fi
```

```
duntil=$u
dnum=`expr $duntil-$dfrom+1`
if [$dnum -lt 1]; then
  echo 'ERROR: 開始, 終了年指定に誤りがあります'
  exit
fi
repnum=0
repitem=
for i in 1 2 3 4 5 6 7 8 9 10
do
  eval rep="\$WWW_rep""$i";
  if [-n "$rep"]; then
    if ["$repnum" -eq 0]; then
      repitem="repcode=any ('""$rep"
    else
      repitem="$repitem""', '""$rep"
    fi
      repnum=`expr $repnum+1`
  fi
done
if [$repnum -ne 0]; then
  repitem="$repitem""')"
fi
parnum=0
paritem=
for i in 1 2 3 4 5 6 7 8 9 10
do
  eval par="\$WWW_par""$i";
  if [-n "$par"]; then
    if ["$parnum" -eq0]; then
      paritem="parcode=any ('""$par"
    else
      paritem="$paritem""', '""$par"
    fi
      parnum=`expr $parnum+1`
  fi
done
if [$parnum -ne 0]; then
  paritem="$paritem""')"
fi
```

```
if [$repnum -eq 0 -a $parnum -eq 0]; then
  echo' レポート国かパートナー国のどちらかを最低指定してください';
  exit
fi;
sitcnum=0
sitcitem=
for i in 1 2 3 4 5 6 7 8 9 10
do
  eval sitc="\$WWW_sitc""$i";
  if [-n "$sitc"]; then
    if ["$sitcnum" -eq 0]; then
      sitcitem="sitc=any ('""$sitc"
    else
      sitcitem="$sitcitem""', '""$sitc"
    fi
      sitcnum=`expr $sitcnum+1`
  fi
done
if [$sitcnum -ne 0]; then
  sitcitem="$sitcitem""')"
fi
itemnum=0
for i in 1 2 3 4 5 6 7 8 9 10
do
  eval tmp="\$WWW_arg"$i; arg=`echo $tmp | $NKF -e`
  eval tmp="\$WWW_arg"$i"c"; argc=`echo $tmp | $NKF -e`
  eval tmp="\$WWW_arg"$i"f"; argf=`echo $tmp | $NKF -e`
  eval tmp="\$WWW_arg"$i"v"; argv=`echo $tmp | $NKF -e`
  eval tmp="\$WWW_arg"$i"o"; argo=`echo $tmp | $NKF -e`
  if ["$argc"="on"]; then
      c="$arg""$argf""'""$argv""'"
      cond="$cond"" ""$c"
  fi
  if ["$argo"="on"]; then
    if [-n "$arg"]; then
      item="$item"" ""$arg"
      itemnum=`expr $itemnum+1`
    fi
  fi
done
```

```
if [-z "$dnum"]; then # 終了年指定無し
  msg=' 期間 '$WWW_from' 年から'
# cnt=$itemnum
else
  msg=' 期間 '$WWW_from' 年から '$u' 年まで'
# cnt=`expr $itemnum+$dnum`
fi
hp=$WWW_termc", "$dfrom", "$dnum #時系列データ部分のヘッダ出力パラメタ
if ["$WWW_reverse"="r"]; then
  $Q_SEL -d $db -h $dbhead -l $WWW_limit -p $hp "$repitem" \
         "$paritem" "$sitcitem" $cond "=" $item>$TMPF0
  $RMATRIX -r -f $TMPF0>$TMPF
  revp="-l 0"
else
  $Q_SEL -d $db -h $dbhead -l $WWW_limit -p $hp "$repitem" \
         "$paritem" "$sitcitem" $cond "=" $item>$TMPF
  revp="-l 1"
fi
case $WWW_type in
  html)
    cnt=`$RMATRIX -x -f $TMPF`
    $Q_SHOW -m "$msg" $revp $cnt<$TMPF; ;
  save)
    if [-z "$WWW_user"]; then
      echo' ユーザー名を入力してください';
    fi;
    if [-n "$WWW_user"]; then
      $Q_SAVE -u "$WWW_user" -m "$msg" <$TMPF | $NKF -j
    fi; ;
  mail)
    if [-z "$WWW_email"]; then
      echo 'Email アドレスを入力してください';
    fi;
    if [-n "$WWW_email"]; then
      $Q_MAIL -u "$WWW_email" -m "$msg" <$TMPF | $NKF -j
    fi; ;
esac
rm -f $TMPF0 $TMPF
exit
```

8.2.3　検索プログラム群（シェルスクリプト）

orasel

Oracle RDBによるデータベースから，TABセパレートな形式でデータを切り出す汎用的検索プログラム。orasel_r.shを呼び出す。

```
#!/bin/csh
#
#
set prog=$0
set basedir=/NF/local/comp/bin
set pickhead="$basedir""/pickhead"
set printhead="$basedir""/printhead"
set awk=/NF/local/Solaris2J/bin/gawk
set head=/NF/local/Solaris2J/bin/head
set rsh=/bin/rsh
set RORASEL="$basedir""/orasel_r.sh"
set SQLCOM="/NF/tmp/orasel_$$.sql"
set SQLTMP="/NF/tmp/orasel_$$.tab"
set wktbl="orasel$$"
set db=
set dbhead=
set item=
set cond=
set con='and'
set outlim=all
set hp=
set hparm=
set condflg="true"
set condnum=0
set i=1
set l=${#argv}
while ($l>=$i)
    if ("$argv[$i]" !="") then
        switch ("$argv[$i]")
        case -n:
                set verb="false"; breaksw
        case -v:
```

```
            set verb="true"; breaksw
        case -h:
            set i=`expr $i+1`; set dbhead="$argv [$i]"
            breaksw
        case -d:
            set i=`expr $i+1`; set db="$argv [$i]"
            breaksw
        case -l:
            set i=`expr $i+1`; set outlim="$argv [$i]"
            breaksw
        case -a:
            set con="and"; breaksw
        case -o:
            set con="or"; breaksw
        case -p:
            set i=`expr $i+1`
            set hparm=`echo $argv [$i] | sed -e 's/,//g'`
            breaksw
        case-*:
            echo ' Invalid option appeared. '; exit 1
        case =:
            set condflg="false"; breaksw
        default:
            if ("$condflg"=="true") then
                if ("$condnum"==0) then
                  set cond="$argv [$i]"
                else
                  set cond="$cond"' '"$con"' '"$argv [$i]"
                endif
                set condnum=`expr $condnum+1`
            else
                set item="$item"' '"$argv [$i]"
            endif
            breaksw
        endsw
    endif
    set i=`expr $i+1`
end
if ("$db"=="") then
        echo 'Something happen. (Database file name missing.)'
```

```
        echo 'Bye.'
        exit 1
endif
#項目チェック，ヘッダ出力
if ("$item" =="") then
        echo 'Error, At least one item needs.'
        exit 1
endif
#ユニーク化(selectする項目名はuniqでないとcreate tableが
#できない。
set item="`$pickhead -i -u -t '      ' -1 $item < $dbhead`"
set rc=$status
if ("$rc" != "0") then
        echo 'Something happen.(on item variable, pickhead)'
        echo 'Bye.'
        exit 1
endif
set headline="`$pickhead -i -u -t'    ' -2 $item < $dbhead`"
set rc=$status
if ("$rc" != "0") then
        echo 'Something happen.(on headline variable, pickhead)'
        echo 'Bye.'
        exit 1
endif
if ("$hparm" =="") then
        # ヘッダ出力(時系列なし)
        echo "$headline"
        # SQL時系列項目はなし
        set item2=""
else
        # ヘッダ出力(時系列あり)
        echo -n "$headline"'    '
        $printhead $hparm
        # SQL時系列項目列を用意(v1985などが項目名となる)
        set item2=`$printhead -p 'v' $hparm`
endif
# SQL作成
# echo "<pre>ORASEL program"
# echo $db
# echo "$condnum: $cond: $outlim"
```

```
# echo "$item"
# GO AHEAD
# $SQLCOM ファイルに SQL 文を作成する。
# 前処理記述を設定
echo '--automatic generation' >$ SQLCOM
cat >> $SQLCOM << EOM
define WKTBL=&1
define SQLTMP=&2
drop table &WKTBL;
set colsep '     '
set newpage 0
set pagesize 0
set heading off
set termout off
set echo off
set feedback off
set linesize 1000
EOM
## テーブルから抜き出す SQL 記述を合成
echo "create table $wktbl as select" >> $SQLCOM
# select ..... from に入る項目名並びにはカンマを入れる
echo $item | sed -e 's/ /,/g' >> $SQLCOM
if("$hparm" !="") then
        echo "," >> $SQLCOM
        echo $item2 | sed -e 's/ /,/g' >> $SQLCOM
endif
echo "from $db" >> $SQLCOM
if("$condnum" !=0) then
        echo "where $cond" >> $SQLCOM
endif
echo "/">> $SQLCOM
## 抽出結果を TAB 形式ファイルに出力する SQL 記述を合成
echo "spool &SQLTMP" >> $SQLCOM
# extract するための項目並びには TAB を入れる
echo "select" >> $SQLCOM
echo $item | sed -e "s/ / || '     ' || /g" >> $SQLCOM
if ("$hparm"!="") then
        echo" || '      ' || " >> $SQLCOM
        echo $item2 | sed -e "s/ / || '     ' || /g" >> $SQLCOM
endif
```

```
echo " from $wktbl" >> $SQLCOM
echo "/" >> $SQLCOM
## 後処理記述を設定
cat >> $SQLCOM << EOM
spool off
drop table &WKTBL;
quit
EOM
# GO AHEAD
rm -f $SQLTMP
# remote shell でRDBホストのプログラムをキックする。
# SQLplusのメッセージは不要なので廃棄
$RORASEL $SQLCOM $wktbl $SQLTMP > /dev/null
set rc=$status
if ("$rc" !="0") then
  echo "ERROR: Something happen on RDB server."
  echo "(return code=$rc)"
  exit $rc
endif
# 抽出結果を標準出力に(件数制限はここで加える)
if ("$outlim" == "all") then
  cat $SQLTMP | sed -e 's/ *$//'
else
  cat $SQLTMP | sed -e 's/ *$//' | $head -"$outlim"
endif
rm -f $SQLCOM $SQLTMP
exit 0
```

orasel_r.sh

Oracleが動作しているデータベースサーバーマシン上で，Oracle RDBに標準的なSQLPlusユーティリティを起動する。

```
#
setenv ORACLE_SID eb
setenv ORAENV_ASK NO
source /usr/local/bin/coraenv
set path=(~oracle/bin $path)
setenv NLS_LANG japanese_japan.ja16SJIS
```

```
set TMP=/tmp/orasel_remote$$.sql
if ("$1" == "") then
        echo "Usage: $0 sql-file.sql [arg1] [arg2]...."
        exit 1
endif
touch $TMP
chmod 700 $TMP
echo 'username/password' > $TMP
cat $1 >> $TMP
set prog="@""$TMP" ; shift
sqlplus $prog $argv
set rc=$status
rm -f $TMP
exit $rc
```

orasel_web

orasel を使って取り出した結果を，HTML 形式に変換して出力する。

```
#!/bin/csh
set prog=$0
set basedir=$prog:h
set awk=/NF/local/Solaris2J/bin/gawk
set sed=/NF/local/Solaris2J/bin/sed
set sedfile="$basedir""/needssel_web_sedf"
set cnt=1
set item=
set msg=
set opt=
# デフォルトは1行目に横線(HR)を引く。嫌なら0を与えよ。
set line=1
while ("$1" !="")
        switch ($1)
        case -n:
                set verb="false" ; breaksw
        case -v:
                set verb="true" ; breaksw
        case -m:
```

```
                set msg="$2" ; shift ; breaksw
        case -l:
                if("$2" == "0") then
                        set line="-1"
                else
                        set line="$2"
                endif
                shift ; breaksw
        case-*:
                echo '   Invalid option appeared.('"$1"')'
                 exit 1
        default:
                set cnt=$1
                breaksw
        endsw
        shift
end
cat << EOM
<head>
<title>DB Select results table</title>
</head>
<body>
EOM
echo $msg
echo '<br><table>'
$awk -F'          '  ' \
          BEGIN{cnt=0} { \
          if (cnt == 1) { \
                printf "<tr><th colspan=%s><hr></th></tr>\n",\
                        ,coln; \
           }; \
           printf "<tr><td>%s</td></tr>\n", $0; \
           cnt=cnt+1; \
         } ' coln=$cnt l=$line | $sed -f $sedfile
cat << EOM
</table>
</body>
EOM
```

orasel_mail

orasel を使って取り出した結果を，TAB セパレートの Excel ファイルに変換し，メイルへの添付ファイルとして送信する。

```
#!/bin/csh
set prog=$0
set awk=/NF/local/Solaris2J/bin/gawk
set mpack=/NF/local/Solaris2J/bin/mpack
set nkf=/NF/local/Solaris2J/bin/nkf
set crlf=/NF/local/comp/bin/crlf
set TMP=/home/tmp/dbsel.$$.xls
set item=
set msg=
while ("$1" !="")
        switch ($1)
        case -n:
                set verb="false" ; breaksw
        case -v:
                set verb="true" ; breaksw
        case -u:
                set dest="$2" ; shift ; breaksw
        case -m:
                set msg="$2" ; shift ; breaksw
        case -*:
                echo '      Invalid option appeared.' ; exit 1
        default:
                breaksw
        endsw
        shift
end
echo "$msg" | $nkf -s | $crlf -c > $TMP
$nkf -s | $crlf -c >> $TMP
$mpack -s 'Query Result' $TMP $dest
cat << EOM
<head>
<title>DB Select table style</title>
</head>
<body>
```

```
EOM
echo "$msg"
echo '<p>'
echo $dest ' 宛てにメイルで結果を送信しました。<p>'
expr `/usr/ucb/wc -l < $TMP` -1
echo ' 件出力しました。<br>'
echo ' 受信して Excel 等で内容を確認してください。<p>'
cat << EOM
</body>
EOM
rm $TMP
```

orasel_save

orasel を使って取り出した結果を，TAB セパレートの Excel ファイルに変換し，ディスクに書く。

```
#!/bin/csh
set prog=$0
set basedir=$prog:h
set pubdir=/NF/pub/db/result
set id=/usr/bin/id
set date=/usr/bin/date
set head=/NF/local/Solaris2J/bin/head
set awk=/NF/local/Solaris2J/bin/gawk
set mpack=/NF/local/Solaris2J/bin/mpack
set nkf=/NF/local/Solaris2J/bin/nkf
set crlf=/NF/local/comp/bin/crlf
set label=`"$date" +"%m-%d.%H%M%S"`
set msg=
while ("$1" !="")
        switch ($1)
        case -n:
                set verb="false" ; breaksw
        case -v:
                set verb="true" ; breaksw
        case -u:
```

```
                set dest="$2" ; shift ; breaksw
        case -f:
                set label="$2" ; shift ; breaksw
        case -m:
                set msg="$2" ; shift ; breaksw
        case -*:
                echo '     Invalid option appeared. ' ; exit 1
        default:
                breaksw
        endsw
        shift
end
cat << EOM
<head>
<title>DB Select table style</title>
</head>
<body>
EOM
echo "$msg"
echo '<p>'
$id $dest >& /dev/null
if ("$status" !="0") then
        echo 'そのようなユーザーは存在しません :' $dest '<p>'
        goto skipout
endif
set destdir="$pubdir"'/'"$dest"
if (! -d $destdir) then
        mkdir $destdir
        if ($status !=0) then
                echo '出力先ディレクトリを作成できませんでした'\
                        $destdir '<p>'
                goto skipout
        endif
        chmod a+x $destdir
endif
set destfile="$destdir"'/'"$label"'.xls'
echo "$msg" | $nkf -s | $crlf -c > $destfile
$nkf -s | $crlf -c >> $destfile
echo "$label"' に結果を出力しました。('"$destfile"') <p>'
expr `/usr/ucb/wc -l < $destfile` -2
```

```
echo '件出力しました<br>'
echo 'Excel等で内容を確認してください。<p>'
skipout:
cat << EOM
</body>
EOM
```

8.2.4　補助Cプログラム群（C言語）

pickhead.c

定義ファイルから各種の情報を引き出す。例えば項目名（Ex. ccode）から見出し（Ex. “会社コード”）を得る。

```
/*
 pickhead.c
 Usage : ./pickhead [-t str] [-i] [-n] [-#] item1 item2.. <header
 t :  delimiter string is str
 i :  show item name instead of number
 n :  pick up by numbers
 # :  item line number
 データベースのヘッダファイル（TAB区切り）と与えられた項目名を照らし合わせて，
 項目名を項目の番号に，またその逆を行う。
 ヘッダファイルの何行目を出力に利用するかを指定しながら使うこと。
 version 0.1
 version 0.2:
 version 0.3: -x, -u を作る
*/
#include <stdio.h>
#include <fcntl.h>
#include "common.h"
#define LF          (char) 10
#define TAB         (char) 9
#define NULLC       (char) 0
char *prog;
#define BUFLEN        1024
#define IM            1024
int itemu [IM];
char item [IM] [BUFLEN], itemx [IM] [BUFLEN];
```

```
int itemm=0, itemxm=0;
#define KM          1024
char key [KM] [BUFLEN];
int keym=0;
int keyn [KM];
#define NUMBER 0
#define ITEM 1
#define ASIS 0
#define UNIQ 1
#define EXPAND 2
#define EXPCHR '_' /* Expand delimiter character */
main(argc, argv)
int argc;
char **argv;
{
  char str [BUFLEN], delim [1024];
  char *s;
  int rc, i, l, k, cc, line=1, mode1, mode2, uniq;
  FILE *ifp;
  prog=argv [0];
  rc=0;
  delim [0]=TAB; delim [1]=NULLC;
  mode1=ITEM; /* 指定はカラム名で行う */
  mode2=NUMBER; /* 表示はカラム番号で行う */
  uniq=ASIS; /* 出力項目名がだぶった時にどうするか */
  for(i=1; i<argc; i++) {
    switch(argv [i] [0]) {
    case '-':
      if(argv [i] [1]=='i') {
        /* Show the item name itself mode. */
        mode2=ITEM;
      } else if(argv [i] [1]=='n') {
        /* Pick up by numbers mode. */
        mode1=NUMBER;
      } else if(argv [i] [1]=='u') {
        /* Only Uniq item should be printed. */
        uniq=UNIQ;
      } else if(argv [i] [1]=='x') {
        /* make item name uniq by Expanding. */
        uniq=EXPAND;
```

```
    }else if(argv[i][1]=='t') {
      /* Delimiter specified. */
      if(i+1<argc) {
        strcpy(delim, argv[i+1]); i++;
      } else {
        fprintf(stderr,
                "%s : -t option require delimiter.\n",prog);
        rc=1;
      };
    } else {
      if((line=atoi(&argv[i][1]))==0) {
        fprintf(stderr,
                "%s : unknown option. (%s)\n", prog, argv[i]);
        rc=1;
      };
    };
    break;
  default:
    strcpy(key[keym], argv[i]);
    if(mode1==NUMBER) {
      if((l=atoi(key[keym])) !=0) {
        keyn[keym]=l-1;
      } else {
        fprintf(stderr,"%s : number selection required. (%s)\n",
                prog, argv[i]);
        rc=1;
      };
    };
    keym++;
  };
};
if(keym==0) {
  fprintf(stderr,"%s : no item specified.\n",prog);
  rc=1;
};
if(line>2) {
  fprintf(stderr,"%s : line number should be 1 or 2.\n",prog);
  rc=1;
};
if(rc!=0) {
```

```
  usage(); exit(1);
};
/* for(i=0; i<keym; i++) printf("%d %s\n", i, key[i]); /**/
ifp=stdin;
/*
  ifp=fopen(infile, "r");
  if(ifp==0) {
    printf("Cannot open list file %s.\n", infile);
    return(-1);
  };
  */
itemm=0; /* 一行目を用意 */
i=0;
cc='\0';
while(cc!=LF) {
  cc=getc(ifp);
  switch(cc) {
  case EOF:
    fprintf(stderr,"%s: end of file reached.\n",prog);
    exit(1);
  case LF:
  case TAB:
    item[itemm++][i]=NULLC;
    i=0;
    /* printf("--%s--"n",item[itemm-1]); /* */
    break;
  default:
    item[itemm][i++]=cc;
  };
};
itemxm=0; /* 二行目を用意 */
i=0;
cc='\0';
while(cc!=LF) {
  cc=getc(ifp);
  switch(cc) {
  case EOF:
    fprintf(stderr,"%s: end of file reached.\n",prog);
    exit(1);
  case LF:
```

```
  case TAB:
    itemx [itemxm++] [i]=NULLC;
    i=0;
    /* printf ("--%s--\n",itemx [itemxm-1]); /* */
    break;
  default:
    itemx [itemxm] [i++]=cc;
  };
};
if (itemm!=itemxm) {
  fprintf (stderr,
          "%s: item number is not same. (line 1: %d / line 2:%d)\n",
          prog, itemm, itemxm);
  rc=1;
};
/* キー項目がdbhead項目に合致するかチェック */
if (model==ITEM) { /* 項目名で指定したが，そんな項目名はない */
  for (i=0;i<keym;i++) {
    for (l=0;l<itemm;l++) {
      if (strcmp (key [i], item [l])==0) {
        keyn [i]=l;
        break;
      };
    };
    if (l==itemm) {
      fprintf (stderr,"%s: item not found. (%s)\n", prog, key [i]);
      rc=1;
    };
  };
} else { /* 数字で指定したが，それが最大項目番号を越えている */
  for (i=0;i<keym;i++) {
    if (keyn [i]>=itemm) {
      fprintf (stderr,
              "%s : item number is out of range. (%s)\n", prog, key [i]);
      rc=1;
    };
  };
};
if (rc!=0) {
  usage(); exit(1);
```

```
};
/* Uniq checkフラグを消しておく */
for(i=0;i<itemm;i++) itemu[i]=0;
/* キー項目をdbhead項目と置換して出力 */
l=0;
for(i=0;i<keym;i++) {
  if(mode2==NUMBER) {
    if(l!=0) printf("%s",delim);
    printf("%d",keyn[i]+1);
  } else {
    if(line==1) {
      sprintf(str,"%s", item[keyn[i]]); /* 1行め */
    } else {
      sprintf(str,"%s", itemx[keyn[i]]); /* 2行め */
    };
    switch(uniq) {
    case ASIS: /* ユニークチェックなしにそのまま出力 */
      if(l!=0) printf("%s", delim);
      printf("%s", str);
      break;
    case UNIQ: /* ダブっている項目は出力しない */
      if(itemu[keyn[i]]==0) { /* この項目はじめてなので出力 */
        if(l!=0) printf("%s", delim);
        printf("%s", str);
      };
      break;
    case EXPAND: /* ダブっている項目は，名前に番号を追加して出力 */
      if(itemu[keyn[i]]==0) { /* この項目はじめてなのでそのまま出力 */
      } else {
        /* ダブっていたので番号を付ける */
        sprintf(str,"%s%c%d", str, EXPCHR, itemu[keyn[i]]+1);
      };
      if(l!=0) printf("%s",delim);
      printf("%s",str);
      break;
    default: /* Never reach */
      fprintf(stderr,"%s : never reach this statement.\n",prog);
      exit(1);
    };
    itemu[keyn[i]]++; /* 出力した数を保存 */
```

```
    };
    l++;
  };
  if(l!=0) printf("\n");
  fclose(ifp);
  exit(rc);
}
int usage()
{
  fprintf(stderr,
        "Usage : %s [-t str][-i][-n][-x | -u][-#] item1 item2.. <header\n"
        ,prog);
  fprintf(stderr,"      t  : printout delimiter string is str\n");
  fprintf(stderr,"      i  : show item name instead of number\n");
  fprintf(stderr,"      n  : pick up by numbers\n");
  fprintf(stderr,"      u  : printout only uniq item name\n");
  fprintf(stderr,"      x  : make uniq item name by expanding\n");
  fprintf(stderr,"      #  : line number(1 or 2)\n");
}
```

8.3　dbselプログラム群

8.3.1　HTMLファイル

touyou-j.html

多国籍企業データWWWインタフェイス(HTML)

```
<head>
<title>多国籍企業データベース</title>
<!-- Touyou Keizai Data Access-->
</head>
<body> <!--bgcolor="#ffffff">
<!-- 東洋経済新報社 海外進出企業データ -->
<h2>多国籍企業データベース</h2>
<hr>
資料：(表はTAB区切りになっています)<br>
```

```
[<a href="abstract-j.html">概要</a>]
[<a href="format.txt">項目一覧</a>]
[<a href="kuni50.txt">国名一覧</a>]
[<a href="gyou_all.tab">業種コード表</a>]
[<a href="oya_all.tab">親企業コード表</a>]
<hr>
<form action="/cgi-bin/uncgi/touyou-query.cgi">
出力する項目を指定して下さい。<br>
項目名に続くチェックボックス(<input type="checkbox" name="dummy1">)
にチェックをすると，条件指定が有効になります。文字列を指定する場合は　天津伊勢丹(有)　などのように半角の引用符で囲んでください。<br>
最後のチェックボックス(<input type= "checkbox" name= "dummy2" checked>)のチェックを外すと，その項目は出力から除外されます。<p>
<p>
<select name="arg1">
        <option value="">(無し)
        <option value="year" selected>年度
        <option value="country">国コード
        <option value="seq">一連番号
        <option value="seqsub">州特別区コード
        <option value="countrye">国名英語
        <option value="countryj">国名漢字
        <option value="companye">社名英語
        <option value="companyj">社名漢字
        <option value="addresse">住所英語
        <option value="addressj">住所漢字
        <option value="phone">電話番号
        <option value="industry">業種コード
        <option value="indname">業種名
        <option value="indnamex">事業内容
        <option value="president">代表者名
        <option value="capital">資本金額
        <option value="capunit">資本金単位
        <option value="capunitx">資本金通貨
        <option value="employee">従業員合計
        <option value="dispnum1">派遣社員数
        <option value="dispnum2">派遣役員
        <option value="sinflag">進出フラグ
        <option value="sindate">進出年月
        <option value="sinmisc">進出他
```

```
        <option value="japcnum">日本側企業数
        <option value="japccap">日本側出資合計%
        <option value="genccap">現地側出資合計%
        <option value="sctype">業績区分
        <option value="scdate">業績年月
        <optionvalue="scamount">業績金額
        <optionvalue="scunit">業績単位
        <optionvalue="scunitx">業績通貨
        <optionvalue="scdoller">業績$換算額
        <optionvalue="invest">投資目的欄
</select>
<input type="checkbox" name="arg1c">
<select name="arg1f">
        <option value="==" selected>=
        <option value="!=">not=
        <option value="&gt;">&gt;
        <option value="&lt;">&lt;
        <option value="~" selected>like
</select>
<input name="arg1v" size=20>
<input type="checkbox" name="arg1o" checked>
<br>
<select name="arg2">
――― (省略) ―――――――――
        <option value="&lt;">&lt;
        <option value="~" selected>like
</select>
<input name="arg12v" size=20>
<input type="checkbox" name="arg12o" checked>
<br>
<p>
<input type="checkbox" name="nihon"> 日本側出資企業項目を含める (最大16社)
<br>
<input name="jc1" size=6>
<input name="jc2" size=6>
<input name="jc3" size=6>
<input name="jc4" size=6>
日本側出資企業指定 (463100 など6桁数字)
<p>
<input type="checkbox" name="genti"> 現地側出資企業項目を含める (最大4社)
```

```
<p>
<b>検索結果を</b> : <p>
<dl>
<dt><input type="radio" name="type" value="html" checked> 画面に表示
<p>
<dt><input type="radio" name="type" value="save"> ディスクに保存
<dd>ユーザー名 : <input name="user" size=50>
<p>
<dt><input type="radio" name="type" value="mail"> メイルで送る
<dd>E-mail アドレス : <input name="email" size=50>
</dl>
<b>出力行数制限</b>:
<input type="radio" name="limit" value="100" checked> 先頭から100行
<input type="radio" name="limit" value="1000"> 先頭から1000行
<input type="radio" name="limit" value="all"> 制限無し
<p>
<input type="submit" value="Query"> <input type="reset" value="Clear">
</form>
<hr>
<a href="index-j.html"><img alt="back" src="/icon/back.gif"></a>
</body>
```

8.3.2 CGI プログラム(シェルスクリプト)

touyou-query.cgi

WWW インタフェイスで設定されたパラメタを検索プログラムに使えるオプション列に変換。dbsel, dbsel_web, dbsel_mail, dbsel_save を呼び出す。

```
#!/bin/sh
echo Content-type: text/html
echo
TR=/usr/ucb/tr
NKF=/NF/local/Solaris2J/bin/nkf
RETURN=/home/yasuda/WWW/cinema/Post/return-j.html
Q_SHOW=/NF/local/comp/bin/touyousel_web
Q_MAIL=/NF/local/comp/bin/touyousel_mail
Q_SAVE=/NF/local/comp/bin/touyousel_save
```

```
DB=/NF/pub/db/touyou/touyou.data
DBHEAD=/NF/local/comp/lib/touyou/query/dbhead
cond=
item=
for i in 1 2 3 4 5 6 7 8 9 10 11 12
do
  eval tmp="\$WWW_arg"$i  ; arg=`echo $tmp | $NKF -e`
  eval tmp="\$WWW_arg"$i"c" ; argc=`echo $tmp | $NKF -e`
  eval tmp="\$WWW_arg"$i"f" ; argf=`echo $tmp | $NKF -e`
  eval tmp="\$WWW_arg"$i"v" ; argv=`echo $tmp | $NKF -e`
  eval tmp="\$WWW_arg"$i"o" ; argo=`echo $tmp | $NKF -e`
  if ["$argc"="on"]; then
       fakeargv=`echo $argv | sed -e 's/ /_/g'`
# パラメタに空白が混じっていたらdbsel手続きまで一文字列として渡せない
# 為に，やむなく空白を　に変更する。dbselはawkに掛ける直前に戻す
       c=$arg,$argf,$fakeargv
       cond="$cond"' '"$c"
  fi
  if ["$argo"="on"]; then
    if [-n "$arg"]; then
       item="$item"" ""$arg"
    fi
  fi
done
# gawkで
# (jlcode ~ "^00123$|^00134$" || j2code ~ "^00123$|^00134$"....)
# なるようにするために，ここでは "^00123$|^00134$" だけを作る。
# touyouselプログラム中でこれは16項目分に展開される。
# (jlcode,~,"^00123$|^00134$",||,j2code,~,"^00123$|^00134$"....)
# を作るようにする。"," はdbsel中でsedによって " " に換えられる。
  jcnum=0
  jcitem=
  for i in 1 2 3 4
  do
    eval jc="\$WWW_jc""$i";
    if [-n "$jc"]; then
      if ["$jcnum" -eq 0]; then
        jcitem="\"^""$jc""$"
      else
        jcitem="$jcitem""|^""$jc""$"
```

```
      fi
      jcnum=`expr $jcnum+1`
    fi
  done
  if [$jcnum -ne 0]; then
    jcitem="$jcitem""\""
  fi
if [$jcnum -eq 0]; then
  jcond=
else
  jcond="-JC""$jcitem"
fi
# 日本側出資企業項目を出力する場合
jopt=
if ["$WWW_nihon"="on"]; then
  jopt="-JL"
fi
# 現地側出資企業項目を出力する場合
gopt=
if ["$WWW_genti"="on"]; then
  gopt="-GL"
fi
case $WWW_type in
  html)
        $Q_SHOW -h $DBHEAD -d $DB -l $WWW_limit \
                $jopt $gopt $jcond $cond $item ;;
  save)
        if [-z "$WWW_user"]; then
          echo 'ユーザー名を入力してください' ;
        fi;
        if [-n "$WWW_user"]; then
          $Q_SAVE -h $DBHEAD -d $DB -l $WWW_limit -u "$WWW_user" \
                $jopt $gopt $jcond $cond $item
        fi ;;
  mail)
        if [-z "$WWW_email"]; then
          echo 'Email アドレスを入力してください';
        fi ;
        if [-n "$WWW_email"]; then
          $Q_MAIL -h $DBHEAD -d $DB -l $WWW_limit -u "$WWW_email"\
```

```
                $jopt $gopt $jcond $cond $item
        fi ;;
esac
#cat $RETURN
exit
```

8.3.3　検索プログラム群（シェルスクリプト）

dbsel

SQL的な表現でTABセパレートなデータから指定のデータを切り出す汎用的検索プログラム。

```
#!/bin/csh
#
#
set noglob
set prog=$0
set basedir=/NF/local/comp/bin
set pickitem="$basedir""/pickitem"
set awk=/NF/local/Solaris2J/bin/gawk
set head=/NF/local/Solaris2J/bin/head
set item=
set cond=
set con='\&\&'
set outlim=all
set db=
set dbhead=
while ("$1" !="")
        switch ($1)
        case -n:
                set verb="false" ; breaksw
        case -v:
                set verb="true" ; breaksw
        case -h:
                set dbhead="$2" ; shift ; breaksw
        case -d:
                set db="$2" ; shift ; breaksw
```

```
        case -l:
                set outlim="$2" ; shift ; breaksw
        case -a:
                set con="\&\&" ; breaksw
        case -o:
                set con="\|\|" ; breaksw
        case -*:
                echo '    Invalid option appeared.' ; exit 1
        case *,*:
                set cond="$cond ""$1" ; breaksw
        default:
                set item="$item ""$1" ; breaksw
        endsw
        shift
end
if ("$db"=="") goto usage
if ("$dbhead"=="") goto usage
if ("$cond"=="") then
        set cl=
else
        set condline=`$pickitem -t ' '-1 $cond < $dbhead`
        set rc=$status
        if ("$rc" !="0") then
                echo 'Something happen on ' $prog
                echo 'Bye.'
                exit 1
        endif
        set cl=`echo '$'"$condline" | sed -e 's/ / '"$con"' $/g' \
                -e 's/,/ /g' -e 's/_/ /g' `
#       touyou-query.cgi などで，比較文字列に空白があった場合，アンダー
#       スコアに入れ換える処理をしている。その補正を sed で行う。
endif
set itemline=`$pickitem -t ' ' -1 $item < $dbhead`
set rc=$status
if ("$rc" !="0") then
        echo 'Something happen on ' $prog
        echo 'Bye.'
        exit 1
endif
set il=`echo '$'"$itemline" | sed -e 's/ /,$/g'`
```

```
set i2=`echo "$itemline" | sed -e 's/ [0-9][0-9]*/%s/g' -e 's/ / \\t/g'`
set headline="`$pickitem -i -n -t'       ' -2 $itemline < $dbhead`"
echo "$headline"
if ("$outlim"=="all") then
  $awk -F'       ' "$c1"' |printf "'"$i2"'\n",'"$i1"'|' $db
else
  $awk -F'       ' "$c1"' |printf "'"$i2"'\n",'"$i1"'|' $db \
          | $head -"$outlim"
endif
exit 0
usage:
echo "Usage: $prog -h dbhead -d dbfile [item[,|==|>..|,value]]..."
echo " -h : dbhead is item header file"
echo " -d : dbfile is database file"
exit 1
```

dbsel_web

dbselを使って取り出した結果を，HTML形式に変換して出力する。

```
#!/bin/csh
set prog=$0
set basedir=/NF/local/comp/bin
set dbsel="$basedir""/dbsel"
set awk=/NF/local/Solaris2J/bin/gawk
set cnt=0
set item=
set outlim="all"
set opt=
set db=
set dbhead=
while ("$1" != "")
        switch ($1)
        case -n:
                set verb="false" ; breaksw
        case -v:
                set verb="true" ; breaksw
        case -h:
```

```
                set dbhead="$2" ; shift ; breaksw
        case -d:
                set db="$2" ; shift ; breaksw
        case -l:
                set outlim="$2" ; shift ; breaksw
        case -a:
                set opt="$opt"" -a" ; breaksw
        case -o:
                set opt="$opt"" -o" ; breaksw
        case -*:
                echo '     Invalid option appeared.' ; exit 1
        default:
                set item="$item""$1";
                set cnt=`expr $cnt+1`
                breaksw
        endsw
        shift
end
cat << EOM
<head>
<title>DB Select Report</title>
</head>
<body>
<table>
EOM
$dbsel -h $dbhead -d $db $opt -l $outlim $item | $awk -F'       ' ' \
        BEGIN{cnt=0} \
        {if (cnt==1) { \
             printf "<tr><th colspan=%s><hr></th></tr>\n", coln;\
          }; \
          printf "<tr><td>%s</td></tr>\n",$0; \
          cnt=cnt+1; \
        } ' coln=$cnt | sed -e 's:           :</td> <td>:g'
cat << EOM
</table>
</body>
EOM
```

dbsel_mail

dbselを使って取り出した結果を，TABセパレートのExcelファイルに変換し，メイルへの添付ファイルとして送信する。

```
#!/bin/csh
set prog=$0
set basedir=/NF/local/comp/bin
set dbsel="$basedir""/dbsel"
set awk=/NF/local/Solaris2J/bin/gawk
set mpack=/NF/local/Solaris2J/bin/mpack
set nkf=/NF/local/Solaris2J/bin/nkf
set crlf=/NF/local/comp/bin/crlf
set TMP=/tmp/dbsel.$$.xls
set cnt=0
set item=
set outlim='all'
set db=
set dbhead=
while ("$1" !="")
        switch ($1)
        case -n:
                set verb="false" ; breaksw
        case -v:
                set verb="true" ; breaksw
        case -h:
                set dbhead="$2" ; shift ; breaksw
        case -d:
                set db="$2" ; shift ; breaksw
        case -u:
                set dest=$2 ; shift ; breaksw
        case -l:
                set outlim=$2 ; shift ; breaksw
        case -*:
                echo '     Invalid option appeared.' ; exit 1
        default:
                set item="$item ""$1";
                set cnt=`expr $cnt+1`
                breaksw
        endsw
```

```
        shift
end
$dbsel -h $dbhead -d $db -l $outlim $item \
          | $nkf -s | $crlf -c > $TMP
$mpack -s 'Query Result' $TMP $dest
cat << EOM
<head>
<title>DB Select report</title>
</head>
<body>
EOM
echo $dest ' 宛てにメイルで結果を送信しました。<p>'
expr `/usr/ucb/wc -l < $TMP` -1
echo ' 件出力しました。<br>'
echo ' 受信して Excel 等で内容を確認してください。<p>'
cat << EOM
</body>
EOM
rm $TMP
```

dbsel_save

dbsel を使って取り出した結果を，TAB セパレートの Excel ファイルに変換し，ディスクに書く。

```
#!/bin/csh
set prog=$0
set basedir=/NF/local/comp/bin
set pubdir=/NF/pub/db/result
set dbsel="$basedir""/dbsel"
set id=/usr/bin/id
set date=/usr/bin/date
set head=/NF/local/Solaris2J/bin/head
set awk=/NF/local/Solaris2J/bin/gawk
set mpack=/NF/local/Solaris2J/bin/mpack
set nkf=/NF/local/Solaris2J/bin/nkf
set crlf=/NF/local/comp/bin/crlf
```

```
#set TMP=/tmp/dbsel.$$.xls
set label=‘"$date"+"%m-%d.%H%M%S"‘
set cnt=0
set item=
set outlim='all'
set db=
set dbhead=
while ("$1" !="")
        switch ($1)
        case -n:
                set verb="false" ; breaksw
        case -v:
                set verb="true" ; breaksw
        case -h:
                set dbhead="$2" ; shift ; breaksw
        case -d:
                set db="$2" ; shift ; breaksw
        case -u:
                set dest=$2 ; shift ; breaksw
        case -l:
                set outlim=$2 ; shift ; breaksw
        case -f:
                set label=$2 ; shift ; breaksw
        case -*:
                echo '     Invalid option appeared.' ; exit 1
        default:
                set item="$item ""$1";
                set cnt=‘expr $cnt+1‘
                breaksw
        endsw
        shift
end
cat << EOM
<head>
<title>DB Select Result</title>
</head>
<body>
EOM
$id $dest >& /dev/null
if ("$status" !="0") then
```

```
        echo 'そのようなユーザーは存在しません :' $dest '<p>'
        goto skipout
endif
set destdir="$pubdir"'/'"$dest"
if(! -d $destdir) then
        mkdir $destdir
        if($status !=0) then
                echo '出力先ディレクトリを作成できませんでした'\
                        $destdir '<p>'
                goto skipout
        endif
        chmod a+x $destdir
endif
set destfile="$destdir"'/'"$label"'.xls'
$dbsel -h $dbhead -d $db -l $outlim $item \
        | $nkf -s | $crlf -c > $destfile
echo $label 'に結果を出力しました。('"$destfile"')<p>'
expr `/usr/ucb/wc -l < $destfile` -1
echo '件出力しました。<br>'
echo 'Excel等で内容を確認してください。<p>'
skipout:
cat << EOM
</body>
EOM
```

8.3.4 補助Cプログラム群(C言語)

pickitem.c

定義ファイルから各種の情報を引き出す。例えば項目名(Ex. ccode)から見出し(Ex.　会社コード　)や項目番号(2)を得る。

```
/*
 pickitem.c
 Usage : ./pickitem [-t str] [-i] [-n] [-#] item1 item2.. < header
 t :  delimiter string is str
 i :  show item name instead of number
 n :  pick up by numbers
```

```
 # :  line number
 データベースのヘッダファイル(TAB区切り)と与えられた項目名を照らし合わせて，
 項目名を項目の番号に，またその逆を行う。
 ヘッダファイルの何行目を参照に利用するかを指定しながら使うこと。
*/
#include <stdio.h>
#include <fcntl.h>
#include "common.h"
#define LF        (char)10
#define TAB       (char)9
#define NULLC     (char)0
char *prog;
#define BUFLEN      1024
#define KM          1024
int keyf [KM];
int keyn [KM];
char key [KM] [BUFLEN], keyx [KM] [BUFLEN];
int keym=0;
#define IM          1024
char item [IM] [BUFLEN];
int itemm=0;
#define NUMBER 0
#define ITEM 1
main(argc, argv)
int argc;
char **argv;
{
  char str [BUFLEN], delim [1024];
  char *s;
  int rc, i, l, k, cc, line=0, model, mode2;
  FILE *ifp;
  prog=argv [0];
  rc=0;
  delim [0]=TAB; delim[1]=NULLC;
  model=ITEM; /*指定はカラム名で行う */
  mode2=NUMBER; /*表示はカラム番号で行う */
  for (i=1;i<argc;i++) {
    switch (argv [i] [0]) {
    case '-':
      if (argv [i] [1]=='i') { /* Show the item name itself mode. */
```

```
      mode2=ITEM;
    } else if(argv[i][1]=='n') { /* Pick up by numbers mode. */
      mode1=NUMBER;
    } else if(argv[i][1]=='t') { /* Delimiter specified. */
      if(i+1<argc) {
        strcpy(delim,argv[i+1]); i++;
      } else {
        fprintf(stderr,
                "%s : -t option require delimiter string.\n",prog);
        rc=1;
      };
    } else {
      if((line=atoi(&argv[i][1]))==0) {
        fprintf (stderr,"%s : unknown option. (%s)\n", prog, argv [i]);
        rc=1;
      };
    };
    break;
  default:
    keyf [keym]=0;
    strcpy (str,argv [i]);
    if((s=(char*)strchr(str,','))==NULL) {
      keyx [keym] [0]=NULLC; /* no extra string */
    } else {
      strcpy(keyx [keym],s); /* keep the extra string */
      *s=NULLC; /* chop the string at the ',' */
    };
    if(mode1==ITEM) {
      strcpy(key [keym++],str);
    } else {
      if(atoi(str)!=0) {
        keyn [keym++]=atoi(str)-1;
      } else {
        fprintf(stderr,
                "%s : number selection required. (%s)\n",
                prog,argv [i]);
        rc=1;
      };
    };
  };
```

```
};
if(keym==0) {
  fprintf(stderr,"%s : no item specified.\n",prog);
  rc=1;
};
if(rc!=0) {
  usage(); exit(1);
};
/* for(i=0;i<keym;i++) printf("%d %s\n", i, key[i]); */
ifp=stdin;
/*
 ifp=fopen(infile,"r");
if(ifp==0) {
  printf("Cannot open list file %s.\n", infile);
  return(-1);
};
 */
/* skip the extra line */
for(i=0;i<line-1;i++) {
  while(1) {
    fgets(str, 256, ifp);
    if(str[strlen(str)-1]==LF) break;
  };
};
itemm=0;
l=0;
while(1) {
  cc=getc(ifp);
  switch(cc) {
  case EOF:
    fprintf(stderr,"%s : end of file reached.\n",prog);
    exit(1);
  case LF:
  case TAB:
    str[l]=NULLC;
    /* printf("--%s--\n",str); */
    strcpy(item[itemm++],str);
    if(cc==LF) goto skipout;
    l=0;
    break;
```

```
    default:
      str[l++]=cc;
    };
  };
skipout:
  for(i=0;i<keym;i++) {
    for(l=0;l<itemm;l++) {
      if(model==ITEM) {
        if(strcmp(key[i],item[l])==0) {
          keyf[i]=1; keyn[i]=l;
          break;
        };
      } else {
        if(keyn[i]==l) {
          keyf[i]=1; strcpy(key[i],item[l]);
          break;
        };
      };
    };
  };
  l=0;
  for(i=0;i<keym;i++) {
    if(keyf[i]==0) {
      fprintf(stderr,"%s : item not found. (%s)\n",prog,key[i]);
      rc=1;
    } else {
      if(l!=0) printf("%s",delim);
      if(mode2==NUMBER) {
        printf("%d%s",keyn[i]+1,keyx[i]);
      } else {
        printf("%s%s",key[i],keyx[i]);
      };
      l++;
    };
  };
  if(l!=0) printf("\n");
  fclose(ifp);
  exit(rc);
}
int usage()
```

```
{
  fprintf(stderr,
        "Usage : %s [-t str][-i][-n][-#] item1 item2.. <header\n"
        ,prog);
  fprintf(stderr,"    t : delimiter string is str\n");
  fprintf(stderr,"    i : show item name instead of number\n");
  fprintf(stderr,"    n : pick up by numbers\n");
  fprintf(stderr,"    # : line number\n");
}
```

trim.c

出力されたデータから指定範囲の期間データのみを抜き出す。

```
/*
  Triming data.
  1997.2 Yasuda.
*/
#include <stdio.h>
#include <sys/types.h>
#include "common.h"
#define TAB       (char)(9)
#define NULLC     (char)(0)
#define LF        (char)(10)
#define CR        (char)(13)
#define DH        TAB         /* ヘッダ項目区切り文字 */
#define DD        ':'         /* データ項目区切り文字 */
#define MAXBUF 32768
char ifname[256], ofname[256], inbuf[MAXBUF];
FILE *ifp, *ofp;
int pad=0, ext=0, skip=0, start, limit=0;
main(argc, argv)
int argc;
char *argv[];
{
  extern char *optarg;
  extern int optind;
  int ch, i;
```

```
ifp=stdin;
ofp=stdout;
/* option recognise */
while ((ch=getopt(argc, argv, "pe:s:")) !=EOF)
  switch((char)ch) {
  case 'p': /* Padding */
    pad=1;
    break;
  case 's': /* Skip line */
    if(!isdigits(optarg)) usage();
    skip=atoi(optarg);
    break;
  case 'e': /* Extra field */
    if(!isdigits(optarg)) usage();
    ext=atoi(optarg);
    break;
  case '?': /* Help message */
  default:
    usage();
  }
/* Skip the position of argc, argv [] */
argc -=optind;
argv +=optind;
if((argc!=1)&&(argc!=2)) {
  usage();
  exit(4);
};
if(!isdigits(argv[0])) usage();
start=atoi(argv[0]);
if(argc==2) {
  if(!isdigits(argv[1])) usage();
  limit=atoi(argv[1]);
};
fgets(inbuf, sizeof(inbuf), ifp);
for(i=0;i<skip;i++) {
  cutlf(inbuf); fprintf(ofp,"%s\n", inbuf);
  fgets(inbuf, sizeof(inbuf), ifp);
};
while(!feof(ifp)) {
  cutlf(inbuf);
```

```
    trim(inbuf);
    fgets(inbuf, sizeof(inbuf), ifp);
  };
  close(ifp);
  close(ofp);
  exit(0);
}
/*-------------------------------------------------------------*/
/*
  データフィールドの必要な領域だけを取り出す。該当がない行は破棄。
  trim(inbuf)
  pad    : パディング
  ext    : 前方に付く余分なフィールド
  start  : 取りだし開始フィールド番号（190702など）
  limit  : 取りだしフィールド数（0で無制限取りだし）
*/
int trim(buf)
    char *buf;
{
  int rc=0, i, j, k, l, dstart, dlimit, pstart, plimit;
  char hbuf[MAXBUF], bbuf[MAXBUF], str[256];
  pstart=start; plimit=limit; /* 指定期間情報を作業変数に確保 */
  if(ext==0) {/* スキップ項目あるなし分岐 */
    l= -1; /* fake的なコード */
    hbuf[0]=NULLC;
  } else {
    l=skipkey(DH, buf, ext);
    strncpy(hbuf, buf, l); hbuf[l]=NULLC;
  };
  i=getkey(DH, &buf[l+1], str); str[i]=NULLC; l+=i+1;
  dstart=atoi(str);
  i=getkey(DH, &buf[l+1], str); str[i]=NULLC; l+=i+1;
  dlimit=atoi(str);
  /* printf("Head!%s!\n", hbuf); printf("Body!%s!\n", &buf[l+1]); */
  if((dstart+dlimit)<pstart) {
    /* 指定期間の開始時に届くデータが存在しない */
  } else if((plimit!=0)&&((pstart+plimit-1)<dstart)) {
      /* 指定期間の打ち切り指定がありながら，最も古いデータに届かない */
  } else {
    /* データが存在する */
```

```
    k=0;
    if(dstart<pstart) {/* データを指定期間までスキップさせる */
      i=skipkey(DD, &buf[1], pstart-dstart);
      l+=i;
      dlimit-=pstart-dstart;
      dstart=pstart;
    } else if(dstart>pstart) {/* 指定期間に至るまでパディングする */
      for(i=pstart;i<dstart;i++) {
        bbuf[k++]=DH;
      };
      if(plimit!=0) plimit-=dstart-pstart;
      pstart=dstart;
    }; /* この時点ではスタートは揃っている */
    i=pstart;
    while(1) {
      if(i>(dstart+dlimit-1)) break; /* データがなくなった */
      if((plimit!=0)&&(i>(pstart+plimit-1))) break; /* 打ち切り指定に到達 */
      j=getkey(DD, &buf[l+1], str); str[j]=NULLC; l+=j+1;
      if(i != dstart) { /* 最初の項目の前にはデリミタをつけない */
        bbuf[k++]=DH;
      };
      strncpy(&bbuf[k], str, j); k+=j;
      i++;
    };
    bbuf[k]=NULLC;
    if(k!=0) {
      if(hbuf[0]==NULLC) { /* ヘッダなしはボディだけ出力 */
        fprintf(ofp,"%s\n",bbuf);
      } else { /* ヘッダ付きはデリミタもつける */
        fprintf(ofp,"%s%c%s\n",hbuf,DH,bbuf);
      };
    };
  };
  return(rc);
}
/*-------------------------------------------------------------------*/
int usage()
{
  fprintf(stderr," Usage: trim[-p][-s #][-e #]start[limit]\n");
  fprintf(stderr,"    -p: padding enable. (Unsupported yet)\n");
```

```
  fprintf(stderr,"    -s: # skip line.\n");
  fprintf(stderr,"    -e: # extra field exists.\n");
  fprintf(stderr," start: start number of triming.\n");
  fprintf(stderr," limit: limit number of triming.\n");
  exit(-1);
}
```

calcdnum.c

指定された期間情報から，該当の項目の先頭からの位置（番号）を得る。

```
/*
  期間情報からデータの項目番号を得る
*/
#include "common.h"
main(argc, argv)
in    targc;
char    *argv[];
{
  int  n,rc=0;
  unsigned char typ,per[256];
  if(argc !=3) usage();
  if(strlen(argv[1])!=1) usage();
  typ=argv[1][0];
  strcpy(per,argv[2]);
  n=calcdnum(typ,per);
  printf("%d\n",n);
  return(rc);
}
int usage()
{
  fprintf(stderr,"Usage: calcdnum type per\n");
  fprintf(stderr,"type: data period type parameter.");
  fprintf(stderr,"(one of 1(M),2(Q),3(H),4(Y))\n");
  fprintf(stderr," per: period data. (like 199701)\n");
  exit(1);
};
```

8.4 汎用補助Cプログラム群(C言語)

common.h

自製のC用標準ライブラリ。(各種ツール)

```
/*
  Cut the extra LF or CR from the tail of string.
*/
int cutlf(str)
char *str;
{
  register int ll;
  ll=strlen(str);
  if(str[ll-1]==13 || str[ll-1]==10) {
    str[ll-1]=0;
  };
  return(0);
}
/*
  Cut the extra spaces from the head and tail of string.
*/
int cutsp(str)
char *str;
{
  int i,l,start,end,ll;
  start=-1;
  ll=strlen(str);
  for(i=0;i!=ll;i++) {
    if((start==-1)&&(str[i]!=' ')) { /* 最初の空白以外文字を記録 */
      start=i;
    };
    if(str[i]!=' ') { /* 最後の空白以外文字を記録 */
      end=i;
    };
  };
  if(start==-1) { /* 全文字空白だった */
    str[0]=(char)(0);
    return(0);
```

```
  };
  /* str[from-end]に非空白文字列あり */
  l=0;
  for(i=start;i<=end;i++) {
    str[l++]=str[i];
  };
  str[l]=(char)(0);
  return(0);
}
/*
  文字列が数字列かどうかを判定する。
*/
int isdigits(str)
unsigned char *str;
{
  int i,ll;
  ll=strlen(str);
  for(i=0;i<ll;i++) {
    if(!isdigit(str[i])) return(0);
  };
  return(1);
}
/*
  common routine for timeseries database.
*/
/*
  期種コードと期間から，西暦ゼロ年第一期からの項目番号を算出
  datacnt=calcdnum(termc,per);
  termc='1', per="199702" など
*/
int calcdnum(termc,per)
    char termc,*per;
{
  int n,fac;
  char s1[16],s2[16];
  switch(termc) {
  case '1': fac=12; break; /* 月次データ */
  case '2': fac= 4; break; /* 四半期データ */
  case '3': fac= 2; break; /* 半期データ */
  case '4': fac= 1; break; /* 年次データ */
```

```
  default:
    fprintf(stderr,"calcdnum(): Illegal term code(%c).\n",termc);
    return(-1);
  };
  if(!isdigits(per)) {
    fprintf(stderr,"calcdnum(): Illegal period code(%s).\n",per);
    return(-1);
  };
  strncpy(s1,per,4); s1[4]='\0';
  strncpy(s2,&per[4],2); s2[2]='\0';
  n=atoi(s1) * fac+atoi(s2) -1;
  /* -1 は 199701,CY の場合に1998となって直観に反しないようにするため */
  return(n);
}
/*
  西暦ゼロ年第一期からの項目番号と期種コードから，期間文字列を算出
  calcdnumr(per,termc,dnum);
  termc='1', dnum=23965 -> per="199702" など
*/
int calcdnumr(per,termc,dnum)
     int dnum;
     char termc,*per;
{
  int rc=0,fac;
  switch(termc) {
  case '1': fac=12; break; /* 月次データ */
  case '2': fac= 4; break; /* 四半期データ */
  case '3': fac= 2; break; /* 半期データ */
  case '4': fac= 1; break; /* 年次データ */
  default:
    fprintf(stderr,"calcdnumr(): Illegal term code(%c).\n",termc);
    return(-1);
  };
  if((dnum<0)||(dnum>(2100*12))) {
    fprintf(stderr,"calcdnumr(): Illegal dnum code(%d).\n",dnum);
    return(-1);
  };
  sprintf(per,"%d%02d",dnum/fac,dnum%fac+1);
  /* +1 は calcdnum() の処理との対応 */
  return(rc);
```

```
}
/*
  Count delimited key from string. Return the number of fields.
  Return-1 when error occured.
  */
int countkey(delim,str)
     char delim;
     char *str;
{
  register int i,n=1,ll;
  ll=strlen(str);
  for(i=0;i<ll;i++) {
    if(str[i]==delim) {
      n++;
    };
  };
  return(n);
}
/*
  Skip delimited key from string. Return the length of skip bytes.
  もしもスキップ項目数をゼロとしたら-1が返る。
  */
int skipkey(delim, str, n)
     char delim;
     char *str;
     int n;
{
  register int i,l=0;
  if(n==0) return(-1);
  i=0;
  while(1) {
    switch(str[i]) {
    case 0:     /* end of string, NULL */
    /* case 10:    /* LF */
    /* case 13:    /* CR */
      return(i);
    default:
      if(str[i]==delim) {
        l++;
        if(l==n) return(i); /* 最後のデリミタを含めたければi+1を返せ */
```

```
      };
      i++;
    };
  };
}
/*
  Get delimited key from string. Return the length of key.
  */
int getkey(delim, str, key)
     char delim;
     char *str,*key;
{
  register int i,l,ll;
  l=0;
  ll=strlen(str);
  for(i=0;i<ll;i++) {
    if(str[i]==delim) {
      key[l]=(char)0;
      return(l);
    } else {
      key[l++]=str[i];
    };
  };
  key[l]=(char)0;
  return(l);
}
```

printhead.c

項目位置(番号)から，当該期間を示す文字列(199701 など)を出力する。

```
/*
   Print timeseries data item string.
   時系列データの為のヘッダ 199701 199702... など，を表示する。
   1997.2 Yasuda.
*/
#include <stdio.h>
#include <sys/types.h>
#include "common.h"
```

```
#define NN        40    /* 項目数制限がなければデフォルトで40項目 */
#define TAB       (char)(9)
#define NULLC     (char)(0)
#define LF        (char)(10)
#define CR        (char)(13)
#define DH        TAB      /* ヘッダ項目区切り文字 */
int ext=0, start, limit;
main(argc, argv)
       int argc;
       char *argv[];
{
  extern char *optarg;
  extern int optind;
  char termc, term[256];
  int ch, i, j;
  /* option recognise */
  while((ch=getopt(argc, argv, "pe:s:")) !=EOF)
    switch((char)ch) {
    case 'e': /* Extra field */
      if(!isdigits(optarg)) usage();
      ext=atoi(optarg);
      break;
    case '?': /* Help message */
    default:
      usage();
    }
  /* Skip the position of argc, argv[] */
  argc -=optind;
  argv +=optind;
  if((argc!=2)&&(argc!=3)) {
    usage();
    exit(4);
  };
  if(strlen(argv[0])!=1) usage();
  termc=argv[0][0];
  if(!isdigits(argv[1])) usage();
  start=atoi(argv[1]);
  if(argc==3) { /* 出力項目数の指定 */
    if(!isdigits(argv[2])) usage();
    limit=atoi(argv[2]);
```

```
  } else { /* 指定無し */
    limit=0;
  };
  /* 項目番号を付ける */
  for(i=0;i<ext;i++) {
    putchar(DH);
  };
  if(limit==0) { /* 出力項目数指定無し。デフォルト個数を出力 */
    j=NN;
  } else { /* 出力項目数指定あり */
    j=limit;
  };
  for(i=0;i<j;i++) { /* 項目数分だけ時系列タイトルを吐く */
    if(i!=0) putchar(DH);
    calcdnumr(term, termc, start+i);
    printf("%s",term);
  };
  if(limit==0) { /* 出力項目数指定無しの場合は最後に目印に ---- と出力 */
    putchar(DH);
    printf("%s","----");
  };
  putchar(LF);
  exit(0);
}
int usage()
{
  fprintf(stderr,"Usage  : printhead [-e #] type start [limit]\n");
  fprintf(stderr,"-e     : # extra field exists.\n");
  fprintf(stderr,"type   : data period type parameter.\n");
  fprintf(stderr,"start  : start period.\n");
  fprintf(stderr,"limit  : limit number.\n");
  exit(-1);
}
```

rmatrix.c

出力表の形式を90度回転させる。

```
/*
    Reverse Matrix
    1997.2 Yasuda.
    ver 0.1 standard version. (Too slow)
    ver 0.2 Memroy crunching version.
    1997.4 Yasuda.
    ver 0.3 Memroy crunching version with dynamic memory allocation.
    ver 0.4 Hand optimization from v0.3.
    ver 0.5 More Memory crunching from v0.4.
    ver 0.6 Reduce Memory requirements from v0.5. (x100 perf. from 0.3)
    ver 0.7 Separate reading for limited memory allocation. (x50-80 perf.)
    ver 0.71 Disabled standard input feature for fseek().
    BUG: SPARC は int=32bit, 1G 程度まで扱えるので int で取っているが,
         それ以上のデータを扱うためには LONG! とコメントのある宣言を int 型から
         long 型へ変更せよ。
         一行当たり MAXBUF (65536) bytes 以下であること。
    BUG: v0.7 からは小さなファイルでない限りは seek() の為に標準入力からファイ
         ルを入れないこと。
   */
#include <stdio.h>
#include <sys/types.h>
#include "common.h"
#define TAB       (char)(9)
#define NULLC     (char)(0)
#define LF        (char)(10)
#define CR        (char)(13)
#define DELIM     TAB         /* 項目区切り文字 */
#define MAXBUF 65536          /* バッファサイズ。入力ファイルレコード長でもある
 */
#define CHECK       1  /* 行列サイズを調べるだけ */
#define CHECKC      2  /* 行列のカラム数を返す */
#define CHECKL      3  /* 行列の行数を返す */
#define REVERSE     4  /* 回転する */
#define MCOL        2000      /* 当初カラム数上限 */
#define MLINE       2000      /* 当初行数上限 */
#define MAXMEM      1048576  /* 最終割り当てメモリ上限 (1MBytes) */
int pad=0,mode=REVERSE,skip=0; /* 各オプションとデフォルト値設定 */
char ifname[256], ofname[256];
FILE *ifp, *ofp;
long ifsp;                /* 入力ファイルのテーブル開始 seek point */
```

```
char buf [MAXBUF];        /* 蓄積バッファメモリ */
int *bufp;                /* buf の各行ごとのスタートバイト位置 */
int tcol, tline;          /* マトリックスの最大カラム数, 行数 */
int *ncol;                /* 各行のカラム数 */
int *bline;               /* 各カラムごとのバイト数 */
int *scol;                /* 各行の最初のアイテムの対 matbuf バイト位置
                             Ex. strcpy(&matbuf[scol[i]+j], str); */
char *matbuf;             /* セルの蓄積バッファ。(ファイルイメージに近い)
*/
int matbufp;              /* カレントの matbuf 差し込み位置 LONG!
                             Ex. strcpy(src, &matbuf[matbufp]) */
int msize;                /* ファイルサイズ */
int mcol=MCOL;            /* 当初確保配列要素数(カラム数に同じ) */
int mline=MLINE;          /* 当初確保配列要素数(行数に同じ) */
main(argc,argv)
      int argc;
      char *argv[];
{
  extern char *optarg;
  extern int optind;
  int ch,rc=0,i,j,l,m;
  char str[256];
  ifname[0]=NULLC; ofname[0]=NULLC;
  bline=(int*)malloc(sizeof(int)*mcol);
  for(i=0;i<mcol;i++) bline[i]=0; /* 初期化が必要 */
  ncol=(int*)malloc(sizeof(int)*mline);
  /* option recognise */
  while((ch=getopt(argc, argv, "pcxyrs:f:?"))!=EOF)
    switch((char)ch) {
    case 'p': /* Padding enable */
      pad=1;
      break;
    case 'c': /* Size check only */
      mode=CHECK;
      break;
    case 'x': /* Column size check only */
      mode=CHECKC;
      break;
    case 'y': /* Line size check only */
      mode=CHECKL;
```

```
    break;
  case 'r': /* Reverse */
    mode=REVERSE;
    break;
  case 's': /* Skip line */
    if(!isdigits(optarg)) usage();
    skip=atoi(optarg);
    break;
  case 'f': /* 入力ファイル */
    strcpy(ifname, optarg);
    break;
  case '?': /* Help message */
  default:
    usage();
  }
/* Skip the position of argc, argv[] */
argc -=optind;
argv +=optind;
if((argc!=0)||(strlen(ifname))){
  usage();
  exit(4);
};
ifp=fopen(ifname,"r");
if(ifp==0){
  printf("Cannot open input matrix file. (%s)\n",ifname);
  usage();
};
ofp=stdout;
switch(mode){
case CHECK: /* size check only */
case CHECKC: /* size check only */
case CHECKL: /* size check only */
  if(fskip(0, skip)<0){
    rc=1;
    goto breakout;
  };
  checksize(mode);
  switch(mode){
  case CHECK: /* size check only */
    printf("column=%d, line=%d \n", tcol, tline); break;
```

```
    case CHECKC: /* size check column only */
      printf("%d"n", tcol); break;
    case CHECKL: /* size check line only */
      printf("%d\n", tline); break;
    };
    break;
  case REVERSE: /* reverse */
    if(fskip(1, skip)<0) {
      rc=1;
      goto breakout;
    };
    ifsp=ftell(ifp); /* 行列開始位置を記憶 */
    checksize(mode);
    reverse();
    break;
  default:
    fprintf(stderr, "rmatrix: main() Illegal mode switch.\n");
    rc=9;
  };
  breakout:
    close(ifp);
    close(ofp);
    exit(rc);
  }
  /*-------------------------------------------------------------*/
  /*
      指定行数をスキップする。該当行数までに eof が来たらエラーを返す。
      fskip(mode, line)
      mode=0: スキップ
      mode=1: スキップしつつそのまま出力
      line    : 行数
      */
  int fskip(mode, line)
       int mode, line;
  {
    int rc=0, i;
    for(i=0;i<line;i++) {
      fgets(buf, sizeof(buf), ifp);
      if(feof(ifp)) {
        fprintf(stderr, "rmatrix: fskip(), endof file reached.\n");
```

```
      rc=-1;
      break;
    };
    if(mode==1) {
      cutlf(buf); fprintf(ofp,"%s\n",buf);
    };
  };
  return(rc);
}
/*
   行列の縦横項目数を調べる。modeによっては数えつつ格納。
   checksize(mode)
   mode: 0 check only, 1 check and count bytes.
*/
int checksize(mode)
     int mode;
{
  int rc=0,i,j,m,p;
  char c;
  tcol=0; tline=0;
  fgets(buf, sizeof(buf), ifp);
  while(!feof(ifp)) {
    cutlf(buf);
    if(mode!=REVERSE) { /* checkの場合は単に数えるだけ */
      /* その行の項目数を数える */
      i=countkey(DELIM, buf);
      /* 項目数が過去最大かどうかをチェック */
      if(tcol<i) tcol=i;
    } else { /* check 以外ならサイズ等設定(tcol,tline,ncol,bline) */
      /* メモリ確保 */
      if((tline+1) > mline) {
        /* fprintf(stderr,"#mline (%d line) %d\n",tline, mline); */
        mline=mline+(mline / 2); /* 1.5倍ずつに増やす */
        ncol=(int*)realloc(ncol, sizeof(int)*mline);
      };
      /* fprintf(stderr,"#here %d -%d\n",mcol, mline); */
      p=0; /* buf[] での相対位置 */
      i=1; /* 行ごとのアイテム数 */
      while(1) {
        c=buf[p++];
```

```
        bline[i-1]++; /* バイト数カウント */
        switch(c) {
        case NULLC:
          goto skipout;
        case TAB:
          i++; /* カラム数カウント */
          if((i) > mcol) { /* カラム数が増えたのでメモリ確保 */
            /* fprintf(stderr,"#col (%d line) %d\n",i, mcol); */
            m=mcol; /* ちょっとキープ */
            mcol=MAX(i, mcol+(mcol / 2)); /* 1.5倍ずつに増やす */
            bline=(int*)realloc(bline, sizeof(int)*mcol);
            for(j=m;j<mcol;j++) bline[j]=0; /* 初期化が必要 */
          };
        default:
        };
      };
    skipout:
      ncol[tline]=i;
      if(tcol < i) tcol=i;
    };
    tline++;
    fgets(buf, sizeof(buf), ifp);
  };
  msize=0;
  for(i=0;i<mcol;i++) {
    if(bline[i] > MAXMEM) { /* 一カラムで割り当てメモリオーバー */
      fprintf(stderr,
              "Too large data file. (%d bytes per one column)\n");
      fprintf(stderr,
              "Enlarge memory allocation. (currently %d bytes)\n",
              bline[i], MAXMEM);
      usage();
    };
    msize+=bline[i];
  };
  return(rc);
}
/*
  回転行列を作る
  reverse()
```

```
 */
int reverse()
{
  int rc=0;
  register int i, j, m, l, n, nn, p;
  char c, *key;
  /* メモリ確保 */
  bufp=(int *)malloc(sizeof(int)*tline);
  for(i=0;i<tline;i++) bufp[i]=0; /* 初期化が必要 */
  scol=(int*)malloc(sizeof(int)*tline);
  matbuf=(char *)malloc(MIN(msize, MAXMEM));
  m=0;
  j=1; /* 最初のカラム位置 */
  for(i=0;i<tcol;i++) {
    if((m+bline[i]) > MAXMEM) { /* 一つ前のカラムまでなら OK! */
      fseek(ifp, ifsp, SEEK_SET);
      rc=fload(j,i);
      rc=reverse_sub(j,i);
      m=bline[i];
      j=i+1;
    } else {
      m+=bline[i];
    };
  };
  fseek(ifp, ifsp, SEEK_SET);
  rc=fload(j,i);
  rc=reverse_sub(j,i);
  return(rc);
}
/*
   ファイルを(必要とあれば部分的に)読んでメモリに格納
   fload(col1, col2)  [col?=1..tcol]
   (col1,1)-(col2,tline)で表現される部分を読んでmatbufにストア
   メモリ中では
   matbuf[0] は(col1, 1)から始まる -> &matbuf[scol[line]+offset]
   ncol[]は最初から最後まで参照
   scol[]は最初から最後まで設定し直し
   bufp[]は最初から最後まで、前回の呼びだしの値に対して積み増し再設定
   * 読み込み高速化のためにbufp[n]を用いて最前の読みだしでどのバイト位置ま
   でを読んだかを記録しているため，fload()の呼びだしは前回のcol2+1が今回の
```

```
   coll になるように，連続していなければならない *
   */
int fload(coll, col2)
    int coll, col2;
{
  int rc=0, i, l, m, p;
  char c;
  matbufp=0;
  l=0;
  fgets(buf, sizeof(buf), ifp);
  while(!feof(ifp)) {
    cutlf(buf);
    /* 一行(の部分)を変数領域に格納 */
    if(ncol[l]<coll) { /* そんな項目は無い */
      /* printf("skip! ncol(%d)=%d\n", l, ncol[l]); */
    } else {                /* データはまだ存在する */
      i=0;                    /* 行ごとの処理したアイテム数 */
      p=bufp[l];              /* buf[] でのバイト位置 */
      scol[l]=matbufp;      /* 蓄積したテーブルの行ごとの開始バイト位置
    */
      while(1) {
        c=buf[p++];
        switch(c) {
        case NULLC:
          matbuf[matbufp++]=NULLC;
          goto skipout;
        case TAB:
          matbuf[matbufp++]=NULLC;
          if(i==(col2-coll)) goto skipout;
          i++; /* カラム数カウント */
          break;
        default:
          matbuf[matbufp++]=c;
        };
      };
    skipout:
      bufp[l]=p;
    };
    l++;
    fgets(buf, sizeof(buf), ifp);
```

```
  };
  return(rc);
}
/*
   回転行列を作る
   reverse_sub(col1, col2)  [col?=1..tcol]
   (col1, 1)−(col2, tline)で表現されるmatbufの内容を転置して出力する
   メモリ中では
   matbuf[0]は(col1, 1)から始まる  −> &matbuf[scol[line]+offset]
   ncol[], scolは最初から最後まで参照
   */
int reverse_sub(col1, col2)
     int col1, col2;
{
  int rc=0;
  register int i, j, m, l, n, nn, p;
  char c, *key;
  for(i=col1-1;i<col2;i++) {
    p=0;
    buf[p]=NULLC; /* 出力バッファの用意 */
    for(j=0;j<tline;j++) {
      /* printf("%d−%d !%s!\n", i, ncol[j], &matbuf[scol[i]]); */
      if(i>(ncol[j]-1)) { /*そんな項目は無い */
        key="";
      } else { /* 該当項目あり */
        key=&matbuf[scol[j]];
      };
      if(j!=0) { /* 最初の項目以外は項目区切り文字を項目の前に挿入 */
        buf[p++]=DELIM;
      };
      while(1) {
        c=*key;
        scol[j]++;
        if((buf[p]=c)==NULLC) break;
        p++;
        key++;
      };
    };
    fprintf(ofp,"%s\n",buf);
  };
```

```
  return(rc);
}
/*-------------------------------------------------------------------*/
int usage()
{
  fprintf(stderr," Usage : rmatrix [--{c|x|y|r}] [-p] [-s #] -f fname\n");
  fprintf(stderr,"    -p : padding enable. (Unsupported yet)\n");
  fprintf(stderr,"    -c : check only mode.\n");
  fprintf(stderr,"    -x : check column only mode.\n");
  fprintf(stderr,"    -y : check line mode.\n");
  fprintf(stderr,"    -r : reverse mode.\n");
  fprintf(stderr,"    -s : # skip line.\n");
  fprintf(stderr,"    -f : use fname file for source matrix.\n");
  exit(-1);
}
```

第9章　パフォーマンステスト

RIEBデータベースの現在の使いやすさはソフトウェアの開発のみで達成されたわけではない。与えられたコンピュータ資源の能力をどのように引き出すかが重要で，そのために我々はいろいろなパフォーマンステストを行った。この章ではその詳細を報告しておくことにする。

9.1　システム構築における能力検証の重要性

RIEBデータベースシステムは我々が設計・開発し研究所のコンピュータに導入したシステムである。RIEBデータベースを効率的に運用するためにはデータベースのソフトウェアの能力とともに，使用するコンピュータシステムのパフォーマンスをチューンアップして高めておく必要がある。そのためにもまず行うべきはパフォーマンステストである。

コンピュータシステムの性能は，CPUに代表される主要な部品の個別の性能指標や，各種のベンチマークテストの結果を参考に推定することができる。例えばCPU処理能力を測るSPECint，SPECfp，[84] システムのデータベース処理速度を測るTPCベンチマーク[85]などがそれである。

しかしこれらの指標は，いつどこでもその能力が発揮できると解釈できるものではなく，自動車の燃費の公表数値のように理想的な条件下でのシステムの一部分の性能数値にすぎない。実際の道路を走ったときの燃費に似て，RIEBデータベースのシステム全体でどれだけの性能を出しているのかを測る指標と

(84) The Standard Performance Evaluation Corporation, http://www.spec.org/
(85) Transaction Processing Performance Council, http://www.tpc.org/

はなっていない。そのため実際に使用する場面を想定したテストを行ない，その結果を実測する必要がある。

そこで我々は研究所機械計算室の現環境のもと，導入したほとんどの製品，また，試用に供された幾つかの製品について，その性能や機能を実測し検証した。こうした幾多の能力テストを経て，RIEB データベースシステムを設計・開発，インプリメントしていったのである。

これまでに我々が行ったパフォーマンステストの要約をまず示し，次節からその詳細を述べることにしよう。

- **Fast Ethernet に関するパフォーマンステスト（二件）**

  1996年以来，機械計算室ではサーバーマシンを 100Mbps の通信速度を持つ Fast Ethernet によって相互接続している。これはサーバー間連係処理の根幹をなす部分であるが，Ethernet の構造的性質から，理論的上限 100Mbps のうち，どれだけの能力を実際に出せるのかは試してみないと判らない。例えば 10Mbps Ethernet では，発表当初は 3Mbps を超えるトラフィックは処理できないといわれていた。しかし各部品，コンピュータの高速化によって，現在では容易に 8Mbps 程度のトラフィックを処理できることが知られている。現在の Fast Ethernet 技術が，サーバーマシンにとってどの程度有効なのかを実測した訳である。

- **Ethernet スイッチにおける輻輳制御機能の検証**

  100Mbps 程度の高速ネットワークでは，そのトラフィックが増え，混雑するにつれて輻輳の制御が大きな問題になる。Fast Ethernet 導入当初では輻輳制御の機能がスイッチになかったため，混雑した状況ではデータ転送が停滞するケースがあったが，最近の製品には多くこの輻輳制御機能が用意されている。その動作と，効果を実測した。

- **NFS サーバー及びディスクアレイに関するパフォーマンステスト（二件）**

データ量が増加傾向にある現在，大容量ファイルサーバーは非常に重要である。NFS(Network File System)とディスクアレイは，ファイルサーバーを支える基本技術であり，これらの能力を総合的に検証した。

・**DLTテープドライブに関するパフォーマンステスト**

バックアップはサーバー運用に欠くことができない。特にファイルサーバーの容量が急増している現在，高速で大容量のバックアップが要求されている。この目的のために導入したDLT(Digital Linear Tape)の能力を検証しその効果を確かめた。

9.2 Fast Ethernetに関するパフォーマンステスト　その1

9.2.1 概要

1996年4月100Mbps末端ネットワークインタフェイスの最も評判の高い100BaseT製品を機械計算室に導入した際にその実力を試してみた。現在のワークステーションではファイル転送など日常の使用で100Mbpsのほとんどを簡単に使い切ってしまうことが判明した。このことから短期間でさらに高速な次世代製品に交替する可能性があると考えられるが，製品自体の価格が安く，導入後即効果が得られるという点で評価できる。100Mbpsを簡単に使い切ることということで，ハブ部分にはダム製品ではなくスイッチ製品を採用することを推奨したい。

9.2.2 実測の環境

二台のワークステーション間で一つのバイナリファイルを転送して転送レートを測定するという簡単なテストを行った。

ハードウェア

転送の実測に使用したシステムは以下の通り。

・Sun SPARC Station 20

Super SPARC 150MHz, 64MB RAM, 2GB HD (Narrow SCSI), Solaris 2.5

SBus 上の純正ボードによる 100BaseT インタフェイスと Narrow SCSI バス上に搭載した Barracuda ディスクを使用。

・Sun Ultra 1 170E

Ultra SPARC 167MHz, 128MB RAM, 2GB HD 2 台 (FastWide SCSI), Solaris 2.5

オンボードの 100BaseT インタフェイスと FastWide SCSI バス上に搭載した上記と同じ Barracuda ディスクを使用。

・CISCO Catalyst 2100

100Base 2 ポート (Repeater), 10Base 25 ポート (Switch)

CISCO の製品は元 GrandJunction 社の製品として有名である。100BaseT インタフェイスをリピータとして 2 つ，他に10BaseT/5 インタフェイスをスイッチとして25ポート持っているが，今回は後者を使用しなかった。

ソフトウェア

FTP で20MBytes のファイルをバイナリモードで転送し，その速度を実測することにした。tcptest[86] などを利用する方法もあるが，現実にユーザーの利用環境で，どれぐらい時間がかかるのかを実測しようとした。FTP は TCP であるので，UDP で実装されている NFS などより負荷がかかると考えられているが，敢えて使ってみることにした。転送の実測はいずれの場合も結果が安定するまで，同一の作業を繰り返し行った。

転送テストに使用したデータはバイナリファイル20MB である。量としては小さいが，ユーザーが実際に転送する平均的な量と考えられる。また，ファイ

(86) TCP プロトコルによるデータ転送能力を調べるためのソフトウェア。

ルのサイズを大きくしてもほとんど転送レートに変化は無いことが分かったので，このファイルを実測に使用した。

9.2.3 転送テスト

まずハードディスクからハードディスクへのローカルコピー（cpコマンドによる）の転送テストを行なった。ディスク入力と出力のバランスを見るために，ハードディスクから/dev/nullへの転送テストも行ったが，ゼロタイムで終了するため全くテストにならなかった。

表31：SS20，Ultra1のローカルファイル転送能力

	SS20 Sec	MB/sec	Mbps	Ultra1 Sec	MB/sec	Mbps
HD →HD	15sec	1.3	10.4	4sec	5.0	40.0
HD→(NULL)	N/A			N/A		

次に同じファイルをFTPによって転送するのに要した時間を示す。

表32：SS20，Ultra1のネットワークファイル転送能力

	SS20 device		Ultra device		results Sec	MB/sec	Mbps
Server	HD	→	HD	Client	3.8sec	5.1	40.8
Server	HD	←	HD	Client	6.1sec	3.2	25.6
Server	HD	→	(NULL)	Client	2.5sec	7.7	61.6
Server	(NULL)	←	HD	Client	2.8sec	7.1	56.8
Client	HD	→	HD	Server	3.8sec	5.1	40.8
Client	HD	←	HD	Server	5.5sec	3.6	28.8
Client	HD	→	(NULL)	Server	2.5sec	7.9	63.2
Client	(NULL)	←	HD	Server	2.0sec	9.6	76.8

実測結果から分かること

1．ディスクへの書き込みを行なう側が高速マシンである方が速い。

2．最も速いのは，サーバーが高速でそのディスクから低速マシンの /dev /null への転送を行なう場合であった。
3．逆に最も遅かったのは，サーバーが低速で，高速マシンのディスクから，サーバーのディスクへの転送を行なう場合であった。

ディスクの読みと書きを要求するケースにおいては 3MBytes/sec から 5MBytes/sec の間の値を出している。低速，高速マシン双方のローカルコピーテストと較べてもかなり近い値を出しており，これ以上は望めない程度の値となっている。ワークステーションのディスクとしては Fast SCSI, 7200rpm の Seagate 社製 Barracuda を搭載しており，このディスクのデータ転送能力，SCSI の能力などから推しても 5MBytes/sec という値は妥当である。

最も速いケースでさらに実験を続ける。

1．出力先として/dev/null を指定して，ディスク書き込みを行なわない設定にする。
2．ディスクの読み出しを高速マシン側で行う。
3．負荷の重いサーバープログラムを高速マシン側で実行する。

この場合の転送能力は 9.6MBytes/sec すなわち 77Mbits/sec に相当し，Fast Ethernet の100Mbps という論理的最大速度のほぼ80%という，極めて高い転送能力を示している。[87] 10Mbps Ethernet では10Mbits/sec の 8 割程度までが限界であることが経験的に判っており，そこから推測して Fast Ethernet でもこの程度が限界ではないかと考えられる。

(87) Ethernet パケットにはデータ以外にヘッダ等のビットが含まれていることに注意。即ち実際にはこれより数%多いデータを送っている。また，FTP ではデータパケット以外に Ack パケットなどがあるため，更にオーバーヘッドが含まれている筈。

1996年12月に，Ultra1Eをもう一台使って再テストを試みた。クライアントとしてSS20を使用した場合に得られた実験結果より大きな値を得ることはできなかった。FTP転送の多重度を上げた場合，転送レートの合計はむしろ減少したので，Ultra1Eをクライアントとして使用しても，FTPで出せる速度は10MBytes/sec以下程度であることが判明した。

9.2.4　まとめ

今回は100BaseTインタフェイスとドライバを含めたソフトウェアがどの程度の性能を出しうるかを計測するために，FTPによる単純なファイル転送実験を行なった。

その結果，最も速い機器を組み合わせた環境下では，ネットワークの論理スピード限界値の8割までスピードがでていることが判明した。今回は実際の使用環境下で8割の能力が出ていることを実測できたので，新たにtcptestなどを含めたこれ以外の測定方法による実験は不必要と考え，これ以上テストを行わないことにした。

今回の実験ではネットワーク上にこの二台のマシンしか存在しない状況を作ってテストを行った。パケットの衝突が生じるような状況でどうなるかは判らないが，我々は，能力の限界まで容易に出せる環境でFast Ethernetを使用していることが分かったのである。これを逆から見れば，こうしたワークステーションを例えば数台接続するだけで，パケットの衝突を激しく起こしてしまう可能性があることを示している。

今後の末端ネットワークインタフェイスは最低100Mbpsを持つべきだと考えている。しかし100Mbpsは大きな値ではなく，現状の製品で容易にこれを使い切ってしまう。Fast Ethernetなどの製品は，端点のインタフェイスとしては，すでに次世代技術とは呼べないと考えるべきだ。恐らくその寿命は短く，次世代製品としてもう一桁高速な製品が必要だ。そして現在の製品で能力のほ

とんどを使い切っているため，相互接続にはダム型ハブではなく，その効果が期待できる限り，スイッチ製品を選ぶべきであろう。

ただし多数のクライアントが一台のサーバーに集中的にアクセスするような環境下では，スイッチ製品は無駄であるどころかパケットの紛失を招いてスループットを下げることになりかねない。この場合にはむしろダムハブの方が，安定した高い性能を得られるはずである。

9.3 Fast Ethernet に関するパフォーマンステスト その2

9.3.1 概要

1997年12月に導入した PC/AT 機に Fast Ethernet 製品を取り付け，この性能を計測した。Sun Ultra1E で実測した数値を上回る結果を出し，はからずも PC/AT 機の性能の高さを実証することにもなった。Fast Ethernet に対する評価は前回と同じで，導入後すぐに役にたつ製品と考えてよい。ただし次世代の 1Gbps 製品への移行に関しては，ネットワークインタフェイスの能力よりも，サービス能力の方が問われることになる。

9.3.2 実測の環境

二台のホストコンピュータ間で一つのバイナリファイルを転送しその転送レートを計測する簡単な方法で行った。

ハードウェア

転送テストに使用したシステムは以下の通り。

・Sun Ultra 1 170E

Ultra SPARC 167MHz, 128MB RAM, 2GB HD 2台 (FastWide SCSI), Solaris2.5

オンボードの100BaseTインタフェイスとFastWide SCSIバス上に搭載した7200rpm Barracudaディスクを使用。

・PC/AT Linux

Pentium 266MHz, 128MB RAM, 4GB HD, Linux 2.0.31

DEC DE500 (10/100Mbps Ethernet for PCI Bus)

PC/ATはASUS社のLX440 chipsetボードを用いたもの。PCIバス上の100BaseTインタフェイスと，Ultra ATAのIDEディスクを用いている。

・Zylan OMNI Switch, 100BaseT 4ポート，10BaseT 8ポート(Switch), ATM

Zylan社の製品。100BaseTに関してはスイッチではなくダム型ハブである。この製品は他にATMインタフェイス，10Baseインタフェイスをスイッチとして8ポート持っているが，今回はこれを使わない。

ソフトウェア

FTPで65MBytes, 260MBytesのファイルをバイナリモードで転送する。転送テストはいずれの場合も結果が安定するまで同じ作業を繰り返し行う。

転送テストに使用したデータはバイナリファイル65MBのものと260MBのものである。65MBというサイズでは，ファイルの内容は完全にメモリに乗ってしまうので，実際には二度目以降のアクセスではディスクアクセスを行なわない。260MBではメモリを完全にオーバフローさせるので，そのようなキャッシュの効果は無い。実測には同じデータファイルを使用した。

9.3.3 転送テスト

FTPの転送レートがどのようなマシンの組み合せで最速になるかは以前の実験で明らかになった。そこで，今回は最も速度が出ると分かっているケースについてのみ実験した。

大きなファイルを FTP によって転送した場合の数値を以下に示す。

表33: PC/AT と Ultra1 の FTP 転送能力

		PC/AT device		Ultra device		results Sec	MB/sec	Mbps
65MB	Client	(NULL)	←	HD	Server	8.8sec	7.4	59.4
	Server	HD	→	(NULL)	Client	6.0sec	10.9	87.3
260MB	Client	(NULL)	←	HD	Server	49.3sec	5.3	42.4
	Server	HD	→	(NULL)	Client	33.0sec	7.9	63.3

この実験で，サーバーを PC/AT とし Ultra をクライアントとして FTP 転送する場合が最速と分かったが，この環境下で 65MBytes のファイル転送をその多重度をあげて計測した結果を以下に示す。

表34: PC/AT と Ultra1 の FTP 転送能力（多重度を上げた場合）

		PC/AT device		Ultra device		results Sec	MB/sec	Mbps
2 多重	Server	HD	→	(NULL)	Client	11.0sec	11.9	94.9
4 多重	Server	HD	→	(NULL)	Client	22.3sec	11.7	93.7

9.3.4 まとめ

1．FTP 転送に関しては前回の結果と比較して PC/AT の方が能力が高いことが分かった。

2．前回の最大値 9.6MBytes/sec，76.8Mbits/sec を上回る実測値 10.9MBytes/sec，87.3Mbits/sec を得た。

3．多重度を上げた場合の実測値 11.9MBytes/sec，94.9Mbits/sec が示すように，FTP オーバーヘッド，Ethernet MAC ヘッダ，プリアンブルなどの転送を含めて，この速度は理論的限界値に近いものである。

FTPは単純なサービスであり，連続して長大なパケットを出せる。また，今回は全てオンメモリで処理が済んでいて，ディスクアクセスによって処理が停滞することがない。Sun Ultra1Eワークステーションを用いたテストから一年半が経った。今回の実験で安価なPCクライアントですら，この程度の処理が実現できることが分かった。これに対応するべきサーバーのネットワーク・インタフェイスとして，OC12 (625Mbps) ATMやGigabit Ethernet (1Gbps)も出始めている。ボトルネックはインタフェイスの速度ではなく，今後サービス能力の方に発生する可能性が高い。

また，ワークステーションと比較することで，LinuxなどのUnix LikeなOSを搭載したPC/AT機の能力が，非常に高いことも分かった。

9.4　Ethernetスイッチにおける輻輳制御機能の検証

9.4.1　概要

1998年10月，10/100Mbps Ethernetスイッチに備わっているバックプレッシャーによるトラフィック抑制機能の効果を検証する機会を得たので試してみた。その結果，抑制機能がついていないスイッチでは，100Mbpsホストから10Mbpsホストへの転送については明らかに輻輳によるパケットの紛失があり，実際のアプリケーションでは転送効率は非常に悪くなると考えられるが，バックプレッシャー機能が，これをよく抑えることを確認できた。それによって安定した高速ネットワーク環境が構築できるだろう。価格も安く，推奨したい。

9.4.2　実測の環境

ハードウェア

転送テストに使用したシステムは以下の通り。

・Sun SPARC Station 5

Micro SPARC 110MHz, 64MB RAM, Solaris2.5

オンボードの10BaseTインタフェイスを使う。

・Sun SPARC Station 20

Super SPARC 150MHz, 64MB RAM, Solaris2.5

SBus上の純正ボードによる100BaseTインタフェイスを使う。

・Sun Ultra 1E

Ultra SPARC 160MHz, 128MB RAM, Solaris2.5

オンボードの100BaseTインタフェイスを使う。

・CISCO Catalyst 2100, 100Base 2ポート(Repeater), 10Base 25ポート(Switch)

この製品は元GrandJunction社の製品として有名であろう。100BaseTインタフェイスを2つ，リピータとして持っている。10BaseTインタフェイスをスイッチとして25ポート持つ。

・ACCTON ES3008-TX, 10/100BaseT 8ポート(Switch)

このスイッチは10/100Mbps自動切替えで，全ポートがスイッチとして機能する。これにはバックプレッシャーによるトラフィック抑制機能がついている。

ソフトウェア

神戸大学総合情報処理センター岡村耕二助手(当時)が作成した連続メディアシミュレータプログラムを使用した。このプログラムは送受信ホストの両側で動作させ，送出側は一定間隔で長いUDPパケットを，受信ホストの状況を無視して送り続け，受信側は，そのうちどれだけ受けとれたかを調べる。

9.4.3　スイッチにおけるパケットの紛失について

例えば以下の図のように，同じ速度のポートを三つ持つスイッチに，A，B，Cの三つのホストが接続されていたとする。

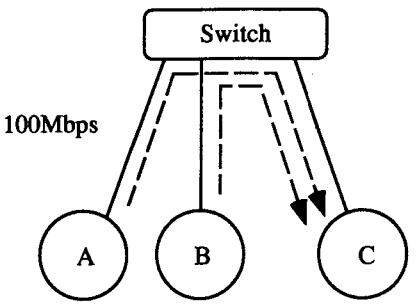

図40：集中によってパケットが紛失する場合

ここでAからC向けのパケットが転送されている途中で，BからC向けのパケットが送信された場合，スイッチはA→Cの転送が終了するまでのあいだ，B→Cパケットの転送を待つために，バッファ内にB→Cパケットを保持する。そしてA→Cの転送が終了した後，保持したパケットをCに向けて送信する。

これが成立するのはあくまでCポートに向けられたトラフィックに余裕のある場合だけで，A→C転送とB→C向けに大量のトラフィックが流れ続けた場合，バッファはいつかあふれてしまう。このあふれを知ることができないA，Bホストは，あふれたパケットを再送する必要があると分からないため，最終的にこのパケットは失われてしまう（パケットの紛失と呼ぶ）。

このバッファあふれは，速度差のある2ホスト間の転送でも発生する。以下の図のように，100MbpsのホストAと，10MbpsのホストBが同じスイッチに接続されている場合で考える。

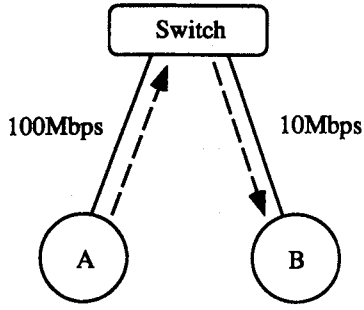

図41：ポート速度差によってパケットが紛失する場合

このとき，A→Bに連続して10Mbpsを上回る出力がなされた場合，スイッチは全てのパケットを受けとってバッファに入れ，徐々にBポートに出力するよう努めるが，バッファを使い切った時点でそれ以上のパケットを蓄積することができず，結果的には紛失となる。

このようなパケットの紛失は，TCPなどの上位プロトコルやアプリケーションによって対処されることになるが，そのタイムアウト値は一般に大きく，長時間の待ちを発生させ，全体の転送を停滞させてしまう。すなわち，回線がすいている間は高速に機能するが，混み始めると非常に遅くなるようなネットワークとなるのである。これでは実用性を著しく欠いたネットワークといわざるをえない。

これらの問題を解決するために，スイッチには輻輳制御の機能が重要なのである。

衝突検知による輻輳制御

振り返って単なるハブ（リピータ）による接続の場合，A→C，B→Cへの転送量が多くなった場合，ハブ上でパケットの衝突が起きる。この衝突信号は送出ホストに伝えられ，しばらく回線が空くのを待った上で，パケットの再送

が試みられる。この衝突検知と再送，送出抑制の機構はCSMA/CD[88]と呼ばれ，Ethernetを支える根本的な技術であるが，多くのスイッチはこの衝突信号を伝えないために，Ethernetが本来持っている輻輳制御が機能しなかったのだと考えることができる。

現在，新しくIEEE 802.3xで規定されているようなフロー（あふれ）制御の技術もあるが，これをサポートしないネットワーク機器も多い。すなわち，現時点で輻輳制御のために確実に使用できるのは，唯一Ethernetの衝突検知信号しかないのである。

出力先のポートの転送能力を上回るトラフィックが来た時に，スイッチはパケットをバファリングする。しかしバッファがあふれそうな状況になった時には，スイッチはパケットを受けとらず，衝突信号を入力ポートに出すのである。こうすることによってパケットを送出しようとしているホストは，パケットをゆっくりと再送することができる。

こうした衝突検知信号（ジャム信号と呼ぶ場合もある）を用いて輻輳制御を行なう手法は，特に規格化されているわけでもなく，決まった呼び方もない。ここでは比較的良く使われている用語「バックプレッシャー」を使用する。

9.4.4　テストの結果

テスト手法

速度差のあるインタフェイス間で発生するあふれによるパケットの紛失が，バックプレッシャー機能によって抑制されていることを確認する。連続メディアシミュレータによって，100KBytsのUDPパケットを100Mbpsインタフェイスを用いて，10ミリ秒ごとに100パケット送る。この10Mbpsインタフェイス側のホストでの受信率を，バックプレッシャー機能がないCatalyst 2100と，

(88) Carrier Sense Multiple Access with Collision Detection, Ethernetの基本的動作原理。

それがある ES3008-TX とで比較する。

プレッシャーあり (ES3008-TX)

以下に受信サイト側 (10Mbps インタフェイス) の一秒ごとの受信状況を示す。

表35: バックプレッシャーのある場合の転送状況

Period (sec)	BandWidth (Mbps)	Loss (%)	lost/recv
1.010	9.703	0.000	(0/97)
1.002	9.603	0.000	(0/96)
1.002	9.603	0.000	(0/96)
1.001	9.603	0.000	(0/96)
1.001	9.603	0.000	(0/96)
1.001	9.603	0.000	(0/96)
1.002	9.603	0.000	(0/96)
1.001	9.603	0.000	(0/96)
1.001	9.603	0.000	(0/96)
1.001	9.603	0.000	(0/96)

一秒ごとの受信パケット数, バンド幅などを表示している。パケットに関しては紛失なく, 10Mbps のバンド幅いっぱいを使って受信していることが分かる。この時送信側は, 10sec かけて送り出している。

表36: バックプレッシャーのある場合の転送時間

Period (sec)
10.005

プレッシャーなし (Catalyst 2100)

再び受信サイト側の受信状況を示す。

表37: バックプレッシャーのない場合の転送状況

Period (sec)	BandWidth (Mbps)	Loss (%)	lost/recv
1.010	9.703	0.000	(0/97)
—	—	—	(有効データなし)

同じ送出を試みたところ，受信側では最初の一秒間に送られた97パケットは正しく受けとったが，次の一秒から満足に受けとらなかった。紛失したものと思われる。

送出側は1.062秒で全パケットを送出し終っており，その間に送られたほとんど全てのパケットを紛失してしまったことがわかる。

表38: バックプレッシャーのない場合の転送時間

Period (sec)
1.062

考察

送出ソフトウェアは，決められた時間間隔で，相手側の受信状況に関係なくパケットを送出するように作られている。

バックプレッシャー機能のないスイッチを間に入れた場合，最初のバッファに取り込むことができたパケットは通ったが，残りのパケットは紛失している。しかし送信ホストはそうした現象がおきていることを知らずに，そのまま送信しつづけ，1sec少々で全パケットの送信を終えてしまった。

バックプレッシャー機能のあるスイッチを間にはさんだ場合，バッファがあふれそうになった時点で，スイッチは衝突信号を送る。これによって送信側はパケット送出を一時停止して，紛失したパケットの再送を行ない，最終的に10 secほど掛けてゆっくりパケットを送信することができている。これによってパケットが紛失することは完全に防げた。

9.4.5 まとめ

テストに使用したES3008-TXは，実売価格で18万円程度の製品である。バックプレッシャー機能のない最も安い10/100Mbps 8portスイッチは既に10万円

を切っているが，この程度の価格差で，パケットの紛失によるヘビートラフィック時のトラブルを未然に回避できるのであれば，投資効果は充分にあるといえる。現時点でIEEE 802.x Flow Controlが一般的でなく，バックプレッシャー以外に通用する輻輳制御の手段がEthernetにない限り，Ethernetスイッチにはバックプレッシャー機能がついたものを選ぶべきだ。

前節で書いたように，Catalyst 2100を用いて100BaseTの実験を行ないその高速性を確認したが，多数のクライアントが一台のサーバーに集中的にアクセスするような環境では，スイッチ製品は無駄であるどころかパケットの紛失を招いてスループットを下げることになりかねない。この場合にはむしろダムハブの方が安定した高い性能を得られるはずである。今回のこの実験で，この問題はバックプレッシャー機能によって解決することができたと考える。

9.5 NFSサーバー及びディスクアレイに関するパフォーマンステスト

9.5.1 概要

1996年12月，DEC社製Unixマシンを試用する機会を得たので，NFSサービス能力及びディスクアレイ能力のテストを行った。

その結果，今回計測できた範囲でのAlpha Server 1000のNFS性能は最大約10MBytes/sec程度になり，その能力を約10MBytes/secで読み出し可能なディスクアレイが支えていることが判明した。比較の対象としてSun Microsystems社製のSPARCCenter2000を使用した計測も行った。SPARCCenter 2000においても最大7.4 MBytes/secを記録し，低速のCPUでもSMP[89]構成であれば多重度が上がるにつれて効率が増し，最新の高速CPUマシンに迫る値を出せることが示された。

---

(89) Symmetric Multi Processing，複数のCPUを対称的に配置して処理能力を高めたハードウェア構成。

また，今後NFSなどのバンド幅を要求するネットワークサービスにとって，最大の障害はネットワークインタフェイスであるとの結果を得た。またSMP構成も非常に有効に機能するといえる。

9.5.2 実測の環境

ハードウェア

・DEC AlphaServer 1000A 5/300

CPU: Alpha 21164/300MHz x 1

RAM: 192MB

DISK: PCI SCSI RAID Controller (3channel) KZPSC-BA x 1 + 32MB chace MS100-AB

4.3GB Wide SCSI Disk RZ29B-VW x 9 (HDのうち3台は本体に格納，残り6台は外付け。コントローラが持つ3つのSCSIチャネルに3台ずつ接続)

NIC: FDDI Controller DEFPA-AB (SAS) x 1

OS: Digital Unix version 3.2G rev.62

ディスクは8台でRAID 5を一つ構成する。細かなパーティションに分けてはいない。残りの1台はホットスタンバイの予備ディスクとなる。ファイルシステムはDEC社のadvfsを適用。nfsdは32個を指定して走らせた。

・SPARCCenter2000

CPU: SuperSPARC 50MHz x 6

RAM: 768MB

DISK: Fast Wide Differential SCSI Controller x 3

PrestServe (SPARCCenter2000専用の2MB chace) x 6

2.1GB Fast Wide Differential SCSI Disk x 18 (HDは全て本体に格納，

コントローラ1つに対しディスクを6台ずつ接続)

NIC: FDDI Controller (SAS) x 1

OS: Solaris2.4J

ディスクはRAIDを構成していない単純なufsファイルシステムである。

nfsd は32個を指定して走らせた。

上記の AlphaServer1000 (以降 Alpha1000) 2セット (同一スペック) とSPARC Center 2000 (以降 SC2000) 1台を FDDI で接続してテストする。

この接続には FDDI スイッチである DEC 社の GIGA Switch を使用。(90)

計測方法

現実的な NFS サービスの状況を再現するために, /usr/local などに保存されている大小種々のファイル群を, 他の Unix マシンからそっくりコピーして再構成した。この20万ファイル, 合計6GB程度のデータの中から454ファイルを任意に選択し, それぞれ先頭から512KBずつ (合計238MB) 読むことで行った。

テストプログラムはC言語で記述し, テストするOS上での標準コンパイラによって最適コンパイルされたものを使用した。比較のために複数ファイルの読み順を違うようにして以下の二種類を用意した。

shotf　指定のファイルを指定量連続で読む。

mshotf 指定のファイルを指定量並行に読む。

このプログラムがどのような順序でデータを読むかを以下に示す。ここでは a1, a2, a3 のファイルがあり, 各ファイルのブロックを先頭から a1-1, a1-2, a1-3 (eof) と表現する。各ファイルについて先頭から2ブロックずつ読むとす

(90) 本来この構成であれば DEC 社製品では Full Duplex FDDI が使用できるはずであるが, Digital Unix 側のドライバソフトウェアの問題で今回は適用できなかった。

れば，アクセスされるブロックは，それぞれ以下の順となる。

shotf　a1-1, a1-2, a2-1, a2-2, a3-1, a3-2

mshotf　a1-1, a2-1, a3-1, a1-2, a2-2, a3-2

・shoftf は各ファイルを順にオープンし，1024バイトずつ読み込み，512Kバイトに達したら次のファイルの処理を行う。これを shot-read と呼ぶことにする。

・mshotf は各ファイルをまず全てオープンし，一つのファイルの先頭1024バイトを読んだ後，次のファイルの1024バイトを読むという処理を行う。[91] これを繰り返して，各ファイルあたり 512K バイトを読めば終る。これを mshot-read と呼ぶことにする。

この実験をローカルディスクと NFS ディスクに対して行い，そのスループットを計測した。また，プロセス多重度を上げた場合のスループットの状況も計測した。

多くのシステムにおいて，1つのプロセスで 454 ファイルを処理する場合と，2つのプロセスで，それぞれ 227 ファイルを処理する場合とでは，単位時間あたりの処理量が異なる。これは処理要求の多重度が上がった場合の処理効率を表している。実際のサーバー運用においては，多くのユーザーが個別の処理を同時に要求する場合が多い。NFS のアクセス傾向はまさにそうした種類のものであるため，NFS サーバーはプロセス多重度の高い状態で，より高い処理能力を発揮するのが望ましい。そこで，今回のテストでは多重度による処理能力の変化がどうなるかが重大な関心事なのである。

NFS における Read/Write 比

今回のテストは Read を中心としており，Write 能力については計測の対象としていない。それは今回検証しようとしている，一般的な利用状況の NFS

(91)　実際には OS の決めるバッファサイズに従って読まれるはずなので，1024バイト毎という表現は正確ではない。

サーバーにおいては，そのアクセスが著しくReadに偏っているためである。

ネットワーク利用やテキスト処理など一般的な利用に供されているNFSサーバーにおけるReadとWriteの比率は一般にReadの方が大きい。今回これを実際にある大学で運用されているNFSサーバーを用いて計測した。

図42に24時間nfsstatコマンドで計測した，あるファイルサーバーでのNFSリクエスト数を示す。横軸は時間帯，縦軸は15分あたりのリクエスト数である。NFSリクエストは20種類近くあるが，そのうち主要な種類上位10種類をreadとwriteに分類して合計し，その他はmiscとした。このNFSサーバーが提供しているNFSディレクトリでは通常のUnixシステムの/usr/local及びユーザーのホームディレクトリが中心である。

さらに詳しく見るために，一日のうち混雑する時間帯の11:30-18:30での主要なcallの合計を表39に示す。

Read 93%に対してWrite 7%という極端にRead処理に偏った結果となった。ここから，NFSマシンではRead能力の高いものが必要ということが分かるのである。

表39: 繁忙期におけるNFSリクエストの詳細

(read-command)	count	(write-command)	count
getattr	355276	setattr	12998
lookup	712183		
readget	288446	write	92907
readdir	70486	create+remove+rename	2670
readlink	8799		
(total)	1435190	(total)	108575
	(93%)		(7%)

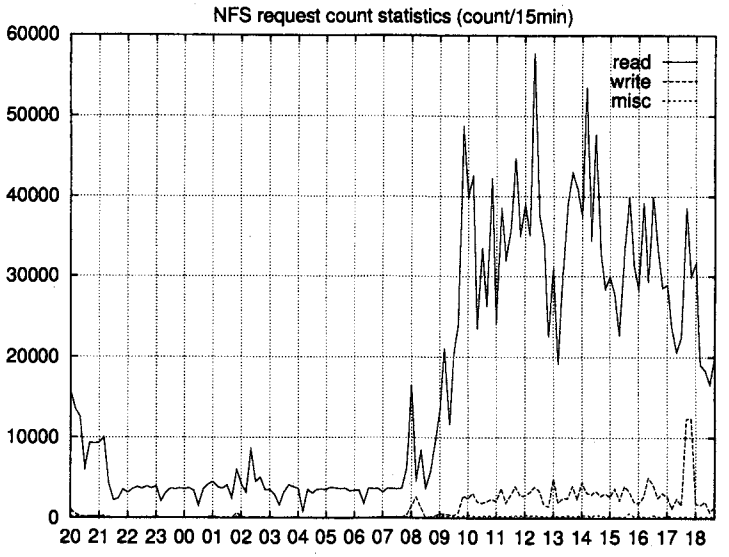

図42: NFSリクエストのRead/Write比

9.5.3 計測結果

Alpha1000

Alpha1000のマシンからAlpha1000のディスクを読ませた。

表40: Alpha1000のNFSアクセス能力（単位は全てMBytes/sec）

（MxNはMプロセスで各Nファイルを示す）

	1x454	2x277	4x113	8x56	16x28
local shot	4.9	6.26	8.6	7.9	8.2
local mshot	4.4	5.2	6.7	7.9	9.3
NFS shot	3.7	4.9	5.9	7	8
NFSmshot	4.25	5.9	6.1	7.9	9.9

プロセス多重度が上がるに連れてスループットは上がる傾向にある。また，ローカルディスクに対してよりも，むしろNFSディスクに対しての方が多重

度が上がったときには高速になる。shotf, mshotf プログラムがローカルディスクを読むよりも，同内容をネットワークを介して nfsd に読ませる方が高速なのである。

これ以上に多重度とパフォーマンスを上げられるのかもしれないが，16プロセスを同時に実行している状況で NFS サーバー側は相当に重くなっており，また 10MBytes/secともなると，恐らくは FDDI ネットワークインタフェイスの転送能力限界である 100Mbps に近づいているので，実験はこの多重度で留めた。

SC2000

SPARCCenter2000 の計測も比較のために行った。Alpha1000 二台のマシンをクライアントに，SC2000 の NFS ディスクを読ませた。

表41: SC2000 のローカルディスクアクセス能力 (単位は全て MBytes/sec)

(MxN は M プロセスで各 N ファイルを示す)

	1x454	2x277	4x113	8x56	16x28
local shot	—	—	—	—	—
local mshot	—	—	—	—	4.5

表42: Alpha1000-SC2000 の NFS アクセス能力 (単位は全て MBytes/sec)

(LxMxNはLホストMプロセスで各Nファイルを示す)

	1x1x454	1x2x277	2x1x227	2x2x113	2x4x56	2x8x28
NFS shot	1.9	5.2	5.3	5.4	6.26	7.2
NFS mshot	—	—	—	—	—	7.4

テストに使用したファイル群は，二つの SCSI インタフェイス，三台のドライブに分散している。この SC2000 は実際に多くのユーザーにサービスを提供しているマシンなので，さらに負荷をかけるなどの詳細なテストは実施できな

かった。NFSアクセスのテストではデータがキャッシュに載っている可能性も高く，厳密な計測は，サービス運用を止めて行う必要がある。

9.5.4　考察

NFSサーバーとしてのAlpha1000の評価

まず，Alpha1000のNFSサービス能力は高いことがわかった。最大でほぼ10MBytes/secという値は，その速度でランダムにディスクを読み続けること自体が難しくなるほどの値である。逆にいえばディスクシステムの読みだし能力の高さがこの値を支えているともいえる。CPUやバスの能力は10MBytes/sec程度ではまだ余裕がある。例えば以下に数十メガバイトのローカルファイルの全データが，メインメモリ上のキャッシュに読み込まれた状態での，Alpha1000における順読みの速度を示す。

表43：Alpha1000のキャッシュ読みだし速度（単位はMBytes/sec）

	file.A
local seq-read	52.5

これはCPUやメモリ転送能力が50MBytes/sec以上あることを示している。すなわち10MBytes/secという数字は，システム能力の上限ではなく，ディスクアクセスや，ネットワークインタフェイスがボトルネックになっていることがわかる。

NFSアクセスについては，多重度が上がるにつれて4.25MBytes/secから9.9MBytes/secへとスループットが上がっており，これは良い傾向といえる。

SMPの効果

SC2000は，CPU単体の速度ではAlpha1000に比べて半分以下の性能しかない。そのかわりCPUを6つ搭載したSMP構成とすることによって，処理の多重度が上がったときに負荷を分散して対応する設計となっている。150ユー

ザーの telnet login，300 プロセス実行，NFS サービスを100台に対して行うというような状況でも，このマシンは安定して実用的なサービスを提供することができる。CPU とメモリを使うベンチマークプログラムを実行すると，CPU m 個で多重度が n 倍 $(n > m)$ になった場合に，ほぼ n/m 倍時間で処理が終了することがそれを裏付けている。

この SC2000 では NFS を介しての shot-read で多重度が上がるにつれ 2M Bytes/sec から 7MBytes/sec まで計算どおり多重度が上がるという結果が得られた。mshot-read による最大スループットは，最大多重度の場合で 7.4MBytes /sec という高い値を出している。つまり一世代前の CPU で，最新の CPU を搭載したマシンに匹敵する性能を得ることができたのである。SMP は処理性能を向上させる有効な手段だといえる。

ボトルネック

現在の Unix ワークステーションでは容易に 10MBytes/sec の NFS 出力が可能なことが判明した。これは 80Mbits/sec であり，物理層までのパケットヘッダ部分を含めると更に数%大きな値となる。この他に NFS リクエストのパケットなどもトラフィックに含めて考えると，今回テストに使ったマシン全てのネットワークインタフェイスである，FDDI インタフェイスは，その能力である 100 Mbps のほとんどを使い切っているといえる。FDDI は 90Mbps 以上出せることが経験的に知られてはいるが，我々は残りの 20Mbps をどこまで利用できるかどうかについてはこれ以上注意を払わない。

今後の高速サーバーに求められるのは高速なネットワークインタフェイスであることがこれで確実となった訳である。

高速ネットワークインタフェイス

現在のところサーバーマシンに使えるネットワークインタフェイスとして標準的なものは FDDI，Fast Ethernet が知られている。クライアント側が 80 Mbps 程度の出力を要求できる性能を持っている現在では，このどちらも現在

のサーバーには容量不足といえる。特にFast EthernetはCSMA/CD方式であるため，100Mbpsの帯域のほとんどを使うようなシーンでは，FDDIより不利となろう。

ATMはOC3（155Mps）では実効転送能力が100Mbpsを割る可能性があり，OC12（622Mbps）でなければ意味がない。Gigabit Ethernet等，1Gbpsの製品もそろそろ出はじめているが，これらについてはテストしていないので判断がつかない。

単一のネットワークインタフェイスの速度を上げることとは別に，複数のインタフェイスを並列に使用してバンド幅を拡大するという方法もある。このような構成が可能なものとしてFibre Channelがある。一つで266Mbpsのバンド幅を持つネットワークインタフェイスを，一台のマシンに4つ搭載し，合計1Gbpsの一つのネットワークインタフェイスとして扱える。Gigabit Ethernetはこの技術が基礎になっており，現在注目されている。

サーバーマシンは今後，複数台に機能分散する傾向がある。その結果，CPU，I/O，ネットワークなど，アクセスが集中する部分のうち，最も大きなネックになるのはネットワークの基幹部分そのものとなる可能性が高い。近い将来ネットワーク技術に求められるものは複数台のサーバー間を太いパイプで結ぶ能力である。

9.5.5　計測時の問題点

今回の能力試験は，中規模のNFSサーバーが，実際にどの程度のサービス能力があるかを検証するためであった。そのためにベンチマークプログラムを作成し，計測を行なったのであるが，ベンチマークテストは往々にして実際の使用時のパフォーマンスを反映した数値を返さないことがある。能力検証のための実測は，そのようなことがないよう，注意深く行なわなければならない。

例えば今回ではRAIDドライブそのものの能力テストも多数試みたが，ど

れもはっきりと性能を示す結果を得られなかった。主たる原因はディスクのフラグメンテーションと，各所に埋め込まれたキャッシュである。以下に事例として示す。

フラグメンテーション

順ファイルをただ連続して高速に読むだけのプログラムを用いて，70MB 程度のデータを読ませた場合，ディスクアレイを初期化した最初の頃に書いたデータと，その後入出力を繰り返した後で作った，そのデータのコピーとでは，読み出し速度は大きく違う。つまりディスクのフラグメンテーションの程度によって結果が大きく変わってくる。以下に意図的にフラグメンテーションの程度だけが異なり，全く同内容の三つのファイルを作って，それらの平均的な読みだし速度実測してみた。

表44: フラグメンテーションによる読みだし速度の差（単位は全て MBytes/sec）

	file.A	file.B	file.C
local seq-read	8.2	5.3	4.1

file.A は明らかに newfs 直後に書いたデータであるが，後のファイルはそれぞれその複製を cp コマンドで作ったものである。最終的に15個程度複製を作ったが，どのファイルでの値を正確なデータと見るべきかは，判断のしようがない。

キャッシュ

Alpha1000 ではキャッシュと呼べるものがあちこちに付いており，これが正確な計測には障害となる。ディスクドライブにあるキャッシュは容量も小さく，連続した大量のデータをアクセスするにはほとんど機能しないので除外できる。しかし SCSI コントローラ上にあるキャッシュは，32MB と大きいので，少なくとも32MB 以下のファイルを繰り返してアクセスするようなテストではディスクの Read 能力評価にはならない。

最近のUnixシステムでは，メインメモリの量はかなり大きく，それが空いている限りディスクキャッシュとして使われる。NFS経由でのアクセスではサーバー，クライアントの両側でキャッシュを利用する場合もある。このメインメモリ上のキャッシュに全データが載っていた場合のスループットを以下に再び示す。

表45：Alpha1000のキャッシュ読みだし速度（単位はMBytes/sec）

	file.A
local seq-read	52.5

この数値は実際のディスクの転送能力を大幅に上回るものである。この効果が計測結果を狂わせないようにするために，今回の計測前には必ず一度ディスクボリュームをマウントし直すことにした。これによってメインメモリ上のディスクキャッシュの影響は排除される。

現実に極めて近い環境を再現し，それを反映したベンチマークを実施することは相当に難しいのである。

9.6　HP9000/725のNFSサーバー能力に関するパフォーマンステスト

9.6.1　概要

1997年4月，機械計算室に導入されているHP社のHP9000/725マシンのNFSサービス能力に関するテストを行った。

実験の結果，HP9000/725はディスクアレイの効果もあり，読み出しで3.2 MBytes/sec，書き込みで0.7MBytes/secと，マシンの規模に対して良好な能力を示した。特に書き込みではディスクアレイは有効に働いている。Fast Ethernetインタフェイスは読み出しの最大能力が10Mbpsを大きく超えているため，非常に有用である。研究所サービス用NFSサーバーとして適当な能

力を有していることが判明した。

ただ，Fast Ethernet インタフェイスの能力上限が 4MBytes/sec 程度である可能性があり，その点では不安が残る。

9.6.2 実測の環境

ハードウェア

・HP HP9000/725

CPU: HPPA PA7100LC 100MHz x 1

RAM: 256MB

DISK: 4GB Wide SCSI Disk ST15230N x 5

NIC: 100BaseT (HP 製の EISA バス上の拡張カードによる)

OS: HP-UX 10.01A

ディスクは 5 台で RAID 5 を一つ構成する。HP-UX の制約からパーティションを 2GB ずつに分けている。ファイルシステムは HP 社の hfs を適用。nfsd は標準の 4 つを走らせている。

・HP HP9000/715

CPU: HPPA PA7100LC 66MHz x 1

RAM: 64MB

DISK: 4.3GB Wide SCSI Disk ST15230N x 2

NIC: 10BaseT

OS: HP-UX 10.01A

ディスクは RAID を構成していない。

・Sun Ultra 1 170E

CPU: UltraSparc 167MHz x 1

RAM: 128MB

DISK: 2.1GB Fast Wide SCSI Disk x 2

NIC: 100BaseT (On board)

OS: Solaris2.5.1J

ディスクはRAIDを構成していない単純なufsファイルシステムである。nfsdは16個を指定して走らせた。

・Sun SparcStation 20

CPU: SuparSparc 150MHz x 1

RAM: 64MB

DISK: 2.1GB Fast Wide SCSI Disk x 7

NIC: 100BaseT (Sun社製のSBUS上の拡張カードによる)

OS: Solaris2.5.1J

ディスクはDiskSuiteによってRAIDを構成している。nfsdは16個を指定して走らせた。

各マシンは100 Baset Tと10 Base Tの両方のインタフェイスを持つスイッチに接続されている。

また，今回のテストでは全てNFS version 2，即ちUDPパケットによる8KBブロックサイズでの処理が行われた。

計測方法

テストプログラムはC言語で記述し，最適コンパイルされたものを使う。

fgetf　　指定のファイルを指定量連続で読む。

mreadf　指定のファイルを指定量並行に読む。

mwritef　指定のファイルを指定量並行に書く。

このプログラムがどのような順序でデータを読むかを以下に示す。ここではa1, a2, a3のファイルがあり，各ファイルのブロックを先頭からa1-1, a1-2, a1-3 (eof) と表現する。各ファイルについて先頭から2ブロックずつ読み書き

する場合，アクセスされるブロックは，それぞれ以下の順となる。

fgetf	a1-1, a1-2, a2-1, a2-2, a3-1, a3-2
mreadf, mwritef	a1-1, a2-1, a3-1, a1-2, a2-2, a3-2

・fgetf は各ファイルを順にオープンし，先頭から順に最後まで読み，次のファイルの処理を行う。

・mreadf, mwritef は各ファイルをまず全てオープンし，一つのファイルの先頭1024バイトを読んだら，次のファイルの1024バイトを読むという順に処理を行う。[92] これを繰り返して，各ファイルあたり 1M バイトに達したら終る。

これをローカルディスクと NFS ディスクに対して行い，そのスループットを比較した。

9.6.3 計測結果

各種読み書き

10Mbps Ethernet に接続された H9000/715 から Fast Ethernet に接続された H9000/725 と Ultra1E に対して 1MBytes のファイル40個の読み書きテストを行う。

---

(92) 実際には OS の決めるバッファサイズに従って読まれるはずなので，1024バイトずつという表現は正確ではない。

表46: 10Mbps クライアントからの NFS アクセス (単位は全て MBytes/sec)

クライアント	HP715	HP715	HP715	HP715	HP715
アクセス先	local	HP725	HP725	Ultra	SS20
ディスク種別	bare	array	bare	bare	array
cp		0.71		0.48	0.15
fgetf	0.74	0.65*		0.67*	
mreadf	1.5	0.89*		0.91*	0.87
mwritef	2.85	0.67	0.2	0.31	0.17

(*ではコリジョンが多発)

Fast Ethernet クライアントからの読み書きテストの数値は以下の通りである。

表47: 100Mbps クライアントからの NFS アクセス (単位は全て MBytes/sec)

クライアント	Ultra	Ultra	Ultra	HP725	HP715	HP725
アクセス先	local	HP725	HP725	local	local	Ultra
ディスク種別	bare	array	bare	array	bare	brea
fgetf	1.09	2.35	2.35	1.18	1.18	0.8
mreadf	1.99	3.25	3.21	2.96	1.6	1.9
mwritef	3.33*	0.74	0.2	4.21*	4*	0.31

(*では数回の実験結果が不安定)

9.6.4 考察

クライアントマシンが 10Mbps Ethernet マシンの場合

mreadf では，どのケースでも 10Mbps Ethernet の上限値に近い，約0.9M Bytes/sec 程度の数値が得られている。[93] この状況下では NFS 性能の上限は試せないが，書き込みに関しては明らかに NFS サーバや使用ディスクの種類に

(93) この時，各ホストが接続されている Zylan の OMNI Switch において，データが Fast Ethernet から 10Mbps Ethernet へと向かう途中で，コリジョンを多発させている。これはバックプレッシャーによる輻輳制御が OMNI Switch にないためで，それがあった場合は，もう少し高い数値が出た可能性がある。

よる差が見られる。恐らくはディスクアレイがシステムに対して早めの処理完了通知を出しているためと考えられるが，HP9000/725 のディスクアレイに対する書き込みが非常に早く終了する。対して Sun の DiskSuite などはほとんど時間を短縮していないように見える。しかし HP9000/725 でもアレイでないディスクに対しては，同じくアレイでない Ultra1E のディスクに対する書き込みに関して，CPU 性能差（SPECint では HP 100 とすると Ultra 252）程度に差が現れているといえる。つまり，特に書き込み処理においては HP9000/725 のディスクアレイが有効に機能しているといえる。

クライアントマシンが Fast Ethernet マシンの場合

読みだしは 10Mbps Ethernet の上限を遥かに超えて HP9000/725 のディスクアレイで 3.25MBytes/sec を出している。ディスクアレイを使わない場合ではこれが 3.21MBytes/sec に下がるが大きな差ではない。対して Ultra1E に対する読みだしは 1.9MBytes/sec であるが，これは一概に上記の数値とは比較できない。手元に Fast Ethernet インタフェイスを持つ実験マシンが 3 台用意できなかったため，10Mbps Ethernet からテストしたときのように同一のクライアントから読み書きをしていないためである。

HP9000/725 への書き込みでは 0.74MBytes/sec となり，10Mbps Ethernet マシンから行った場合とあまり差がない。0.74MBytes/sec は約 6Mbps に相当し，10Mbps Ethernet の伝送速度がネックになっていなかったことを示している。この数字はディスクアレイに対してであり，そうでないディスクに対してはさらに 0.2MBytes/sec 程度にまで落ちる。これは同じくディスクアレイでない Ultra1E への書き込み速度 0.31MBytes/sec よりも低く，Ethenret マシンから行った場合と全く同じ数字を示している。つまりボトルネックはネットワーク以外の部分にあるのである。

FTP 転送速度との比較

FTP によるデータ転送における，Ultra，HP725 マシンの転送能力を参考ま

でに以下に示す。

表48: Ultra, HP9000/725 の FTP 転送能力（単位は全て MBytes/sec）

HP725	→	←	Ultra
Client	3.15	3.6	Server
Server	3.6	3.2	Client

ftpd は HP については OS 付属のもの，Ultra については wu-ftpd を使用している。転送ファイルのサイズは 40MB であり，出力先は常に /dev/null としてディスク出力のオーバーヘッドをなくし，バイナリモードで転送している。この状況下でも，せいぜい 4MBytes/sec 程度，すなわち 32Mbps 程度までしか転送速度が上がらない。

実験機の Ultra1 は過去に SS20 と Fast Ethernet によって接続した状態で，7MBytes/sec 以上を容易に出している。それに対して HP9000/725 との間では 4MBytes/sec を下回る数字しか出せておらず，これは NFS による読み出しテストが出した数値とほとんど同じ値である。つまり，HP725 の Fast Ethernet インタフェイスは，これ以上の速度を出せない可能性がある。

9.6.5 まとめ

1. HP9000/725 の NFS サービス能力は，読みだしに関して，多重度 1 のリクエストで 3MBytes/sec を超える良好な値を示す。
2. HP9000/725 の NFS サービス能力は，書き込みに関して，ディスクアレイの効果が大きいが，それでも 1MBytes/sec を下回る。
3. Fast Ethernet インタフェイスの能力は読み出しにおいては有用である。ただし FTP 転送実験でも 3.9MBytes/sec 程度の性能しか出ておらず，このインタフェイスの性能上限に達している可能性がある。

4．Sun の DiskSuite は高速化には余り貢献していない。

結果的に研究所内 NFS サーバーとしては，現行の Sun SS20 の DiskSuite によるサービスを能力で大きく上回り，置き換えには問題がないと思われる。

9.7 DLT ドライブに関するパフォーマンステスト

9.7.1 概要

今回 BoxHill 社製の DLT (Digital Linear Tape) を導入したので，Sun 社純正の DAT ドライブとの性能比較によるパフォーマンステストを行った。その結果，DLT の性能は DAT のおよそ 3 倍[94]であることが判明した。デバイスの価格も 3 倍だが，一本あたりのテープ容量は 3 倍を大幅に上回り，費用対効果は良好だといえる。

9.7.2 実測の環境

二つのワークステーションと二つのテープドライブを使って，それぞれバックアップ速度を測る，単純な方法で行った。

ハードウェア

バックアップテストに使用したシステムは以下の通り。

・Sun SPARC Station 5

Micro SPARC 115MHz, 64MB RAM, 2GB HD 2 台 (Narrow SCSI), Solaris 2.5

・Sun Ultra 1E

Ultra SPARC 160Mhz, 128MB RAM, 2GB HD 2 台 (Wide SCSI), Solaris 2.5

(94) カタログ値では 4 倍。

・Sun DAT

5GB/tape, 400KB/sec transfer rate, WIDE SCSI Interface

・BoxHill DLT4000

20GB/tape, 1.5MBytes/sec trasfer rate, Narrow SCSI Interface

（ドライブはQuantum社の製造である。）

バックアップテストではワークステーションの外付けディスクから，DATもしくはDLTにバックアップ出力している。ワークステーションは負荷の無い状態でテストし，対象となる全ての機器は同一のSCSIバス上に存在する。

ソフトウェア

バックアップユーティリティには，Solaris 2標準のufsdumpを用いた。このコマンドは一般的なSolaris 2ユーザーの多くが，日常のバックアップに使うだろう。ディスクキャッシュのヒット率などの要素が影響していないことは二度実行して確かめている。

データ

バックアップテストに使用したデータは合計500MBであるが，これらの中には240MB 程度の大きな圧縮ファイルが含まれている。残りの260MB程度はWWWディレクトリなどであり，テキスト，バイナリデータなどが細々と含まれている。純粋にテープドライブの能力を測るには不適当であるが，現実のバックアップ状況を比較的忠実に再現していると思われる。非現実的な部分といえば，このディスクが比較的新しいためにフラグメンテーションをほとんど起こしていないことであろう。

9.7.3　バックアップテストの結果

数値

ufsdumpが出力するメッセージによると，DATもDLTも共に63KBytes/

record で記録している。[95] その結果転送データはほぼ 1Mblock，497MB となった。

表49: ufsdump によるバックアップ時間

種別	Sparc Staion 5 実時間	転送レート	Ultra 1E 実時間	転送レート
DAT	約19分20秒	442KB/sec	約12分40秒	674KB/sec
DLT	約6分30秒	1320KB/sec	約4分05秒	2141KB/sec
(HD)	約7分40秒	1125KB/sec	約6分00秒	1443KB/sec
(null)	約4分20秒	2021KB/sec	約3分00秒	2904KB/sec

DAT，DLT 共にデータ転送のための準備時間などはほとんどなく，このままのレートでテープ限界まで書き込み続けることができると思われる。DAT，DLT 共に圧縮機能を働かせてテストしている。「(HD)」とは，ufsdump の入力と出力を同じディスクに対して行った参考テスト，「(null)」とは，ufsdump の出力を /dev/null に対して行った参考テストである。

DAT を 1 とした場合の，各デバイスの転送レートの比 DAT: DLT: (HD): (null) は以下の通り。

表50:各デバイスの転送レートの比

	DAT	DLT	(HD)	(null)
SS5	1	3.0	2.5	4.6
Ultra1	1	3.2	2.1	4.3

各転送レートの，ホストマシンによる差 (比率) は以下の通り。

表51: 各転送レートのホスト性能による差

	DAT	DLT	(HD)	(null)
Ultra/SS5	1.5	1.6	1.3	1.4

(95) これが最適値かどうかは判らない。

考察

DLTはDATに対して転送レートでほぼ3倍の値が出た。ホストマシンさえ速ければDLTは2.1MBytes/secという性能を出している。ドライブの転送能力のカタログ値は1.5MBytes/secであるため，これは圧縮の効果と思われる。このホストではディスク間でのファイルコピーで5MBytes/secという性能を出しているが，/dev/nullに対するufsdumpバックアップでは2.9MBytes/secまで落ちている。つまりufsdumpの処理は，ホスト本来のデータ転送能力に対して，かなりの負荷になっていることがわかる。つまり，DLTによるバックアップは，せいぜいあと30%から50%程度の能力向上が見込めるにすぎない。(DATでは2.9MBytes/secまで，まだ400%近い開きが有る。)即ちこの状況ではいかなるテープドライブを導入してもDLTの倍の性能は出ないのである。

参考までに他のドライブとの比較を行う。数値はカタログ値である。

表52:テープドライブのコストパフォーマンス比較

種別	転送レート	容量	GB/Hour	実勢価格	テープ価格(Yen/1GB)
DAT 4mm	400KB/sec	4GB	1.4GB/h	17万円	3500円(875円)
EXB 8mm	500KB/sec	7GB	1.8GB/h	20万円	3500円(500円)
DLT4000	1.5MB/sec	40GB	5.4GB/h	70万円	2.7万円(675円)

ドライブ単価は，性能にほぼ比例して上がっている。逆にメディアのビット単価はそれほど変わらない。圧縮が効かない場合でもせいぜい倍になる程度である。即ち，それだけの性能が必要なのであれば，高価なドライブを導入する価値があるといえる。

9.7.4　まとめ

DLTはちょうど現在のワークステーションが出せるバックアップ出力の，最大値に近い能力を持っている妥当な価格の製品である。その能力が必要であれば，DLTを導入する価値がある。

DLT では圧縮無しで20GB，圧縮付きでその倍記録できるとカタログにはあるが，実際のところは30GB 程度であろう。バックアップ時間は 2.1MBytes/sec から，1GBytes/8min = 30GBytes/4Hour 程度だろうか。ディスクアレイのサイズとして 30GB は現在比較的普通であり，これをバックアップするのに要する時間はせいぜい 4 ～ 5 時間と見積もれる。この程度の時間であれば，充分に実用的な運用が可能である。

参 考 文 献

(URL 確認日は1998年10月)

『ASCII』Vol. 21, 7 pp. 346, アスキー (1997)

阿部茂行『アジア経済研究』経済経営研究所叢書48, 神戸大学経済経営研究所 (1998)

ジャニス・ウィンザー著, 日本サン・マイクロシステムズ監訳『Solaris 上級システム管理』アスキー (1995)

小幡一郎編著『Oracle 実践 Q&A』ソフト・リサーチ・センター (1994)

各種 Oracle 7.3 マニュアル, 日本オラクル (1997-1998)

マイケル・J・コアリー他著, 小幡一郎翻訳『ORACLE データベースチューニング』翔泳社 (1997)

民野庄造『行列処理と経済経営情報分析システム -SECRETARY-』経済経営研究所叢書経営機械化シリーズ19, 神戸大学経済経営研究所 (1986)

民野庄造『財務分析と推論言語』「会計・経営情報システムをめぐる諸問題」経済経営研究叢書経営機械化シリーズ20, 神戸大学経済経営研究所 (1989)

『IBM 戦略情報'94 日経ウォッチャー IBM 版』「日経ウォッチャー IBM 版」pp. 42-44, 日経 BP (1993)

米花稔『経営・経済情報制御分析システム -BEICA-』「経営・経済情報分析システムの新展開」経済経営研究叢書経営機械化シリーズ16, 神戸大学経済経営研究所 (1975)

M. ポニャトスキー著, 林秀幸　小畑喜一監訳『HP-UX 10.x システム管理』トッパン, 1996

安田聖『多国籍企業データベース・システム』経済経営研究年報36号, 神戸大学経済経営研究所, 1986

ケン・ランディ著, ハルペン・ジャック　鈴木武生訳『日本語情報処理』ソフトバンク, 1996

デビッド・ロックマン著, 日本オラクル株式会社監訳『Oracle7 SQL 入門』アスキー, 1996

Krause, L. "The Structure of Trade in Manufactured Goods in the East and Southeast Asian Region" in C. I. Bradford and W. H. Branson, ed., *Trade and Structual Change in Pacific Asia,* Chicago: University of Chicago Press, 1987

Petri, Peter "Market Structure, Comparative Advantage, and Japanese Trade under the Strong Yen" Chapter 2 in Paul Krug-man, ed., *Trade with Japan,* The University of Chicago Press, 1991

索　引

研　究　叢　書（既　刊）

第１号　生産と分配に対する貿易効果の分析　片野　彦二著　1961年

第２号　国際貿易と経済発展　川田冨久雄著　1961年

第３号　国際私法の法典化に関する史的研究　川上　太郎著　1961年

第４号　アメリカ経営史　井上　忠勝著　1961年

第５号　神戸港における港湾荷役経済の研究
柴田銀次郎・佐々木誠治・秋山　一郎・山本　泰督共著　1962年

第６号　企業評価論の研究　小野　二郎著　1963年

第７号　経営費用理論研究　小林　哲夫著　1964年

第８号　船内労働の実態　佐々木誠治著　1964年

第９号　船員の雇用制度　山本　泰督著　1965年

第10号　国際私法条約集　川上　太郎著　1966年

第11号　地域経済開発と交通に関する理論　野村寅三郎著　1966年

第12号　国際私法の国際法典化　川上　太郎著　1966年

第13号　南北貿易と日本の政策　川田冨久雄著　1966年

第14号　インド経済における所得分配構造　片野　彦二著　1968年

第15号　ラテンアメリカ経済統合の理論と現実　西向　嘉昭著　1969年

第16号　会計情報とEDP監査　中野　勲・大矢知浩司共著　1972年

第17号　国際収支と資産選択　井川　一宏著　1974年

第18号　経営計測システムの研究
Business & Economic Information Control and Analysis System
定道　宏著　1978年

第19号　日本・オセアニア間の海上輸送とオセアニア主要港の現況
佐々木誠治著　1978年

第20号　軽量経済システムSREPS-BEEICA　定道　宏・布上　康夫共著　1979年

第21号　海上運賃の経済分析　下條　哲司著　1979年

第22号　国際法上の船籍論　嘉納　孔著　1981年

第23号　ブラジル経済の高度成長期の研究　西島　章次著　1981年

神戸大学経済経営研究所

研究叢書（既刊）

第24号　資本蓄積過程の分析
－理論的枠組とオーストラリア経済への適用－　下村　和雄著　1983年

第25号　会計情報公開論　山地　秀俊著　1983年

第26号　企業の国際化をめぐる特殊研究　井上　忠勝・山本　泰督・
下條　哲司・井川　一宏・山地　秀俊共著　1983年

第27号　海運における国家政策と企業行動　海運経済専門委員会著　1984年

第28号　オーストラリアの金融システムと金融政策　石垣　健一著　1985年

第29号　会計情報公開制度の実証的研究
－日米比較を目指して－　山地　秀俊著　1986年

第30号　配船の理論的基礎　下條　哲司編著　1986年

第31号　仮想電子計算機と計算機言語システム
－世界軽量経済モデル分析システム－　川上　太郎著　1986年

第32号　期待効用理論　－批判的検討－　伊藤　駒之著　1986年

第33号　アメリカ企業経営史研究　井上　忠勝著　1987年

第34号　反トラスト政策　－経済的および法的分析－
カールケイゼン・ドナルド　F.ターナー共著
根岸　哲・橋本　介三共訳　1988年

第35号　会計情報システムと人間行動　中野　勲編著　1989年

第36号　国際金融経済論の新展開
－変動為替相場制度を中心として－　井澤　秀記著　1989年

第37号　労働市場研究の現代的課題　小西　康生・三木　信一共著　1989年

第38号　香港企業会計制度の研究　中野　勲編著　1989年

第39号　国際比較統計研究モノグラフ1　能勢　信子編著　1990年

第40号　経済発展と環太平洋経済
西向　嘉昭・石垣　健一・西島　章次・片山　誠一共編著　1991年

第41号　労使問題と会計情報公開　山地　秀俊著　1991年

第42号　経営財務と会計の諸問題　森　昭夫編著　1992年

第43号　国際比較統計研究モノグラフ2　小西　康生編著　1993年

第44号　アメリカ現代会計成立史論　中野　常男・高須　教夫・山地　秀俊共著　1993年

研究叢書（既刊）

第45号	ネットワーク環境における情報システムの研究	宮崎　耕著	1994年
第46号	財務情報分析と新情報システム環境	民野　庄造著	1995年
第47号	税効果会計	梶原　晃著	1995年
第48号	アジア経済研究	阿部　茂行著	1997年
第49号	会計とイメージ	山地　秀俊・中野　常男・高須　教夫著	1997年
第50号	地域保健医療情報システム －加古川地域における地域情報化戦略－	小西　康生・中村　利男著	1997年
第51号	原価主義と時価主義	山地　秀俊編著	1998年

RIEBデータベースの研究　　研究叢書 52

（非売品）

平成11年3月1日　印刷
平成11年3月30日　発行

著　者　神戸大学経済経営研究所助手
安田　豊（やすだ　ゆたか）
神戸大学経済経営研究所教授
阿部茂行（あべ　しげゆき）

発行所　神戸市灘区六甲台町2－1
神戸大学経済経営研究所

印　刷　神戸市灘区友田町3丁目2－3
中村印刷株式会社