WWWベースの高速データ検索システム

畑田 稔[†] 野里 真喜子[†] 遠藤 裕英[††]

本論文では，データベース全体をメインメモリに読み込み，メモリベースでデータ操作を行うWWW（World Wide Web）ベースの高速データ検索システムを提案する．データ操作言語はSQL（Structured Query Language）に準拠するが，テーブル定義方法はシンプルであり，インデックスは必要な列にシステムが自動付加するため，データベース・アプリケーション設計の負担が少ないものとなっている．文字列データのハッシング，質問式構文木の初期簡単化などにより，検索処理の高速化を図った．図書管理システムに適用し，その有効性を検証した．図書検索および雑誌名のブラウジングのクライアント応答時間は0.3秒未満となった．

High Speed Data Retrieval System Based on WWW

MINORU HATADA,[†] MAKIKO NOZATO[†] and HIROHIDE ENDOH[††]

This paper proposes a high speed data retrieval system based on WWW (World Wide Web), which loads whole database and manipulates data on main memory. The data manipulation language is based on SQL (Structured Query Language). The design of database application is easy, because the definition of table is simple, and system automatically adds indexes to some columns at need. We have also shorten the retrieval time by hashing of character string and simplifying of query syntax tree on first data row processing. Proposed architecture is evaluated for the library management system. The client response times of book catalog retrieval and journal catalog browsing have been observed less than 0.3 sec.

1. はじめに

WWW（World Wide Web）の普及により，データベースの分野でもWWWとの連携が急速に進展している[1),2)]．ユーザはWWWブラウザ以外のソフトウェアを必要とせず，慣れ親しんだブラウザでデータベースにアクセスできる，などの利点がある．

WWW–データベース連携システムでは，通常，データの更新，追加頻度よりも検索（参照）頻度の方が大きい．このため検索時間の短縮が全体としての応答時間の短縮，スループット向上の鍵となる．また，検索語を入力して，検索結果を待つという従来型のユーザ・インタフェースではなく，データベースの蓄積内容を何らかの形でビジュアル表示し，そこをナビゲートして必要な情報に到達するような，より使いやすいユーザ・インタフェースを実現する場合にも，検索時間の短縮が重要となる．

一方，パソコンの進歩は著しく，メインメモリは32～64 MBが普通になり，128～256 MBの搭載も容易となった．このため，小規模システムでは，データベース全体をメインメモリに読み込むことができるようになった．

本論文では，これら高速化のニーズとパソコンの進歩を背景として，データベース全体をメインメモリに読み込み，メモリベースでデータ操作を行うWWWベースの高速データ検索システムを提案する．また，図書管理システムへの適用について述べ，その有効性を評価する．

2. システムの概要

2.1 システム構成

提案システムの構成を図1に示す．アプリケーションとのインタフェースはSQL（Structured Query Language）[3)]に類似させた．一部，SQLにない機能を有しているが，全体的にはSQLのサブセットである．データ型は，整数，実数，日時，文字列であり，サイズを32ビットに統一した．文字列はハッシングを施し，そのハッシュ値を32ビットサイズとした．

† 株式会社日立製作所システム開発研究所情報センタ
Systems Development Laboratory, Information System Center, Hitachi, Ltd.
†† 立命館大学理工学部情報学科
Department of Computer Science, Faculty of Science and Engineering, Ritsumeikan University

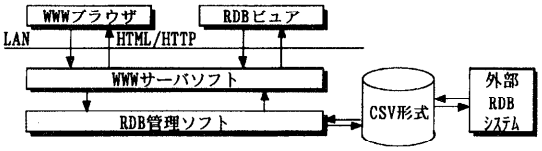


図1 WWWベースのデータ検索システム
Fig. 1 Data retrieval system based on WWW.

このデータ検索システムは，データの更新，追加，削除機能を有するため，RDB（Relational Database）管理システムとして単独でも運用できるが，別のRDB管理システム（以下では外部RDBシステムと呼ぶ）の出先システムとしても動作する．この場合，データ検索システムと外部RDBシステムのインタフェースはCSV（Comma Separated Value）形式のファイル経由である．個々のファイルはデータ検索システムか外部RDBシステムのいずれか一方でしか更新できない．たとえば，図書管理システムでは図書マスター，利用者マスターなどは外部RDBシステムが更新し，データ検索システムはリードオンリーである．逆に，貸出履歴テーブルはデータ検索システムが行の追加，更新を実行し，外部RDBシステムはリードオンリーである．

ファイルとしては，上記のデータファイルのほかにテーブル，ビューの定義ファイルが存在する．ビュー定義は後述し，テーブルの定義例を以下に示す．

create table book autoload from book.txt

この場合，bookがテーブル名，book.txtがファイル名である．オプションautoloadは，ファイルbook.txtが更新されたらファイルを読み込み，自動的にテーブルbookを更新することを表す．

データファイルは第1行が列名行で，第2行以降がデータ行である．また，各列のデータ型はデータ行から自動判定する方式をとっている．このため，各テーブルには必ずデータ行を含むことが前提となっている．

通常のRDB管理システムでは，ユーザが意識して適宜，インデックスを付与しなければならないが，提案システムではインデックスは必要に応じて自動的に付与されるため，定義は不要である．

WWWサーバソフトは，WWWブラウザまたは当システム専用のRDBビュアからのリクエストに応じて，HTML（HyperText Markup Language）形式のファイルをWWWクライアントへ送信する機能のほか，RDB管理ソフトにデータ操作ステートメントを送り，検索結果を受け取ったり，データベースの更新（行の追加，削除，更新など）を指示する機能を有している．

2.2 WWW-データベースインタフェース

WWWサーバとデータベースの連携方式としては，CGI（Common Gateway Interface）方式およびAPI（Application Program Interface）方式が広く使われている．

CGI方式はWWWサーバソフトとデータベース管理システム（DBMS）の間に両者の仲介をするプログラムを置く方式であり，汎用性は高いが，アクセスごとに仲介プログラムの起動が必要となるため，オーバヘッドが大きい．API方式では，WWWサーバソフトがDBMSのクライアントソフトに位置付けられるため，プログラム起動のオーバヘッドはない[4]．しかし，それぞれ独立に開発されたWWWサーバとDBMSを使用するために，そのインタフェースのオーバヘッドが回避できない．そこで，提案システムでは，インタフェースのオーバヘッドを極小化するために，WWWサーバソフトとデータベース管理ソフトを一体化した．

WWWサーバソフトとRDB管理ソフトのインタフェースとなるデータ操作関数を**表1**に示す．ここで，チャネルは，テーブルの識別子である．Execute関数を実行すると，チャネルが戻される．他の関数はこのチャネルを引数の1つとする．

3. RDB管理ソフトの概要

SQLステートメントの実行フローを以下に示す．

Step 1 字句解析，構文解析を行い，中間言語にあたる構文木を生成する．

Step 2 このSQLステートメントが参照するテーブルがビュー（仮想テーブル）であれば，再帰的に，ビューに対するSQLステートメント実行関数をコールして一時テーブルを作成する．

Step 3 構文木中の列名をテーブルポインタ，列番号に変換する．

Step 4 構文木を解釈実行する．Union演算子があれば，次のSelectステートメントを実行する．出力行はこれまでの出力テーブルに付け加えられる．

3.1 字句解析および構文解析

字句は自然言語の単語にあたる．字句解析は，文字の列としてのSQLステートメントを字句の列に区切り，字句を内部表現に変換する．

構文解析では，入力として字句の列を受け取り，これに対応する解析木（内部表現を構文木と呼ぶ）を構成する．手法は下降型解析法と上昇型解析法に大別される．下降型構文解析は解析木を根（文法の出発記号）から葉（字句）の方向に作っていくもので，下向き構文解析ともいわれる[5]．本研究では，PascalやCの構

表 1 データ操作関数
Table 1 Data manipulation functions.

関数名	入力引数	出力引数または関数戻り値	機能概要
Execute	command：SQL コマンド文字列	channel：チャネル	データ操作を実行する.
Request	channel：チャネル column：列番号 row：行番号	string：文字列	セルの値を求める.
Terminate	channel：チャネル	なし	チャネルを閉じる.

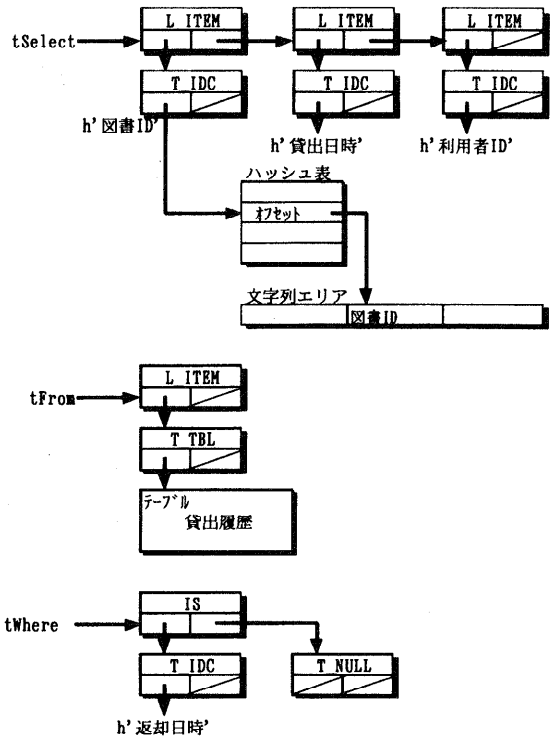


図 2 構文木の例
Fig. 2 An example of syntax tree.

文解析に用いられている再帰下降解析法を採用した.

図 2 に SQL ステートメント

select 図書 ID, 貸出日時, 利用者 ID
from 貸出履歴
where 返却日時 is null

に対する構文木を示す. 構文木は Select 句, From 句, Where 句の 3 つに分かれる. SQL ステートメント全体を 1 つの構文木にすることも考えられるが, 枝が多くなるだけで, 明白なメリットが見つからなかったため, 句ごとに分ける方式を採用した.

文字列は, ハッシングをして, そのハッシュ値をセルの値としている. 図 2 では, たとえば, 文字列「図書 ID」のハッシュ値を h'図書 ID' と表記している.

Select 句はリスト構造で, その要素は式扱いである. From 句もリスト構造であるが, その構成要素はテーブル（ビューを含む）に限定されている（以下, テーブルリストの先頭を主テーブル, 2 番目以降を副テーブルと呼ぶ）. Where 句は全体が 1 つの式扱いである.

3.2 テーブルの構造

テーブルの構造を図 3 に示す. テーブルの各セルはデータ 4 バイト, タイプ 1 バイトからなる. 列のタイプがテキスト型の場合, テーブルのセルにはハッシュ値（ハッシュ表の行番号）が入る. ハッシュ表には文字列を指すオフセットが登録されている. 文字列エリアは, 必要に応じて拡大するダイナミック割り当てを行っているため, アドレス（ポインタ）ではなく, オフセットを登録している. また, 列名は文字列のため, 列名行のセルには列名に対応するハッシュ値が入る.

3.3 ビュー実行

From 句のテーブルリストにビュー名が含まれている場合, まず, このビューが実行される. ビューの定義は Select ステートメントである. この Select ステートメントの実行によって, 1 つの SQL ステートメントの開始から終了までの間だけ存在する一時テーブルが生成される. 一時的であるということを除けば, 通常のテーブルとまったく同じである.

3.4 SQL 実行

データ操作ステートメントは, Select, Update, Insert, Delete の 4 つに大別されるが, データベースの参照が中心となるシステムで最も重要なのは Select ステートメントである. Select ステートメントの処理の流れを以下に示す.

Step 1 先頭行から順に選択関数 Eval(Where, nR) を実行して, この関数の戻り値が真（true）となる行番号を得て表にする. 引数の Where, nR は, それぞれ Where 構文木および行番号である.

Step 2 Group By 句があると, 行の並べ替えを行う（テーブル実体ではなく, 前ステップで得た行番号表を並べ替える）. これにより, 同一グループに属するものが連続して並ぶ. この後, Count, Avg などの集計関数を実行する. Having 句は選択範囲の絞り込みに使われる.

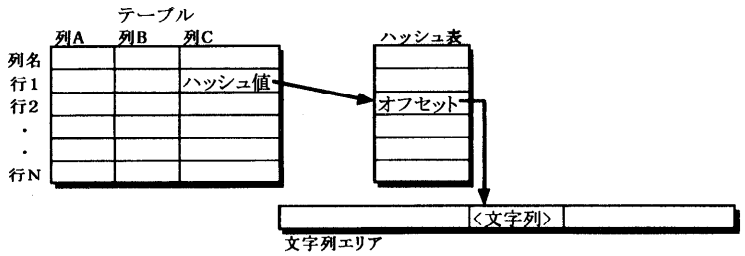
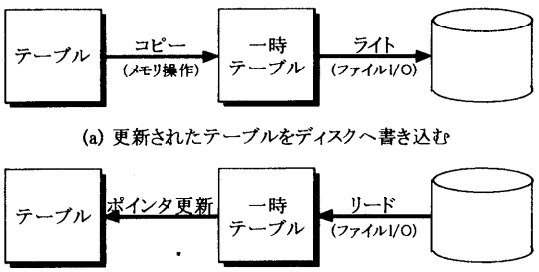


図 3 テーブルのデータ構造
Fig. 3 Data structure of table.



(a) 更新されたテーブルをディスクへ書き込む

(b) ファイル更新を検知し、テーブルを更新する

図 4 データベースの更新
Fig. 4 Update of database.

Step 3 出力テーブルの各列を定義する選択リストに従って出力テーブルを作成する.

Step 4 Order By 句があると，出力テーブルの並べ替えを行う.

Step 5 Distinct 指定があると，まったく内容が等しい行が連続した場合，1 つの行のみを残して重複行を削除する.

SQL ステートメントの実行では I/O 処理は発生せず処理時間が短い．このため，SQL ステートメントの並列処理は行っていない.

3.5 データベースの更新

Update, Insert, Delete ステートメントを実行すると，テーブルの内容が変化する．一定時間後に，そのときのテーブルの内容をファイルに書き込む（図 4 参照）．テーブル変化の検出およびファイルへの書き込み操作は，SQL ステートメントの実行スレッドとは異なる専用のスレッドで行っている.

まず，テーブルの複製を作る．メモリ間コピーのため，テーブルのロック時間は短い．次に，複製テーブルの内容をディスクに書き込む．これには数秒要するが，元のテーブルにロックをかけているわけではないので，並行して SQL コマンドを実行できる.

リード専用（Select ステートメントのみ実行）テーブルの場合，外部 RDB システムがこのテーブルに対応したファイルを更新したとき，ファイルを自動的に読み込み，テーブルを更新する機能を持つ．この場合も，まず，ファイルを一時テーブルに読み込んだ後，ロックをかけて，以前のテーブルの内容を破棄して，新たなデータに切り替えている．メモリコピーすら起こらないため，ロックタイムはきわめて短い.

4. データ検索の高速化

4.1 文字列のハッシング

文字列データの管理にハッシュ法を採用した．2 つの文字列が等しいかどうかは，セルの値（ハッシュ値）が等しいかどうかで判断できるため，文字列検索が整数検索と同じ速さで行われる．また，5.1 節で述べるように，文字列データ格納エリアの削減効果も得られた.

ハッシュ法を使わない場合，図 3 において，テーブルのセルには文字列エリアのオフセットを格納することになる．同じ文字列が出現した場合，重複して文字列エリアに格納され，そのオフセットがセルに登録される．したがって，2 つの文字列が等しいかどうかはセルの値だけでは判断できず，実文字列の照合を行わなければならない．このため，従来の RDB 管理システムでは，データベースアプリケーション設計者が必要に応じてインデックスを付加し，文字列の高速検索を実現している．提案システムでは，ハッシュ法の効果とテーブル全体がメインメモリに常駐し，数万行のテーブル検索時間はわずかなことから，テーブルの結合を除いて，インデックスは使用していない.

SQL ステートメントで Like 演算子を使った場合および検索結果をクライアントに送信する場合，テーブルの全行についてハッシュ値から対応する文字列を取り出すため，オーバヘッドの削減が重要である．一方，ハッシュ表の探索・登録はシステム起動時が中心となる．字句解析でも探索・登録が起こるが，SQL ステートメント中の各字句について一度だけである．また，文字列連結などにより SQL ステートメントの実行でもハッシュ表への登録が生じるが，通常は多発しない.

これらのことから，探索・登録アルゴリズムが簡単で，ハッシュ値対応の文字列取り出しが高速に行われる線形走査法[6]を採用した．

線形走査法では，登録済み文字列の削除は楽ではない．ハッシュ表には削除マークを付けておき，空きとは区別しなければならない．また，エントリごとに参照カウンタを設けなければならない．文字列を登録するとき，参照カウンタを1にセットし，以後同じ文字列を参照するたびに+1する．参照が消滅したとき−1して，値が0になったとき削除の対象にする．提案システムではこのような煩雑さとオーバヘッド増加を避けるため，テーブル更新の結果，ある文字列が使われなくなっても削除しない方式とした．

しかし，Select ステートメントの実行によって発生するハッシュ表への一時的な登録は次のようにして削除する．

Select ステートメントの実行開始から終了までに発生したハッシュ表への登録はハッシュ表とは別のリストで管理する．レスポンスをクライアントに返した後，このリストを元にハッシュ表の登録を削除し，空きに戻す．文字列エリアは書き込みポインタの値を Select ステートメント実行開始時に戻す．

Insert，Update，Delete ステートメントではテーブル更新にともなうものと一時的な登録が混在するため，いっさい削除していない．

4.2 構文木の簡略化

Where 句の評価はテーブルの全行に対して行われるため，オーバヘッド削減が重要である．Like 演算子は左辺と右辺の文字列中の全角（2バイトコード）の英数字およびカタカナをそれぞれ半角（1バイトコード）に，また，英大文字は英小文字に変換して文字列の照合を行うが，たとえば，「書名 like '%Java%'」（%は任意の長さの文字列を表すワイルドカード）の場合，「Java」から「java」へ毎行の処理で同じ変換を行うのは無駄である．このため，第1行に対する処理で構文木の簡略化を行い，第2行目からの Where 句の評価時間の短縮を図っている．「like」に限らず，「=」，「is」などについても同様である．

また，比較式の右辺は行に依存しない定数となることが多い．この場合，第1行の処理で構文木の簡単化を行い，第2行からの処理を単純化する．たとえば，Where 句が「雑誌 ID = Delspc(Left('2 CACM',3))」の場合，第2行からは「雑誌 ID = '2'」と同等の処理となる．

4.3 インデックスの自動付加

RDB を利用した図書管理システムは，貸出履歴テーブルと図書/利用者などのマスターテーブルで構成され，履歴テーブル上のレコードは各マスターテーブルのキー項目で関連付けられる．

```
Create view Q_貸出中 as
    select 図書 ID, 貸出日時, 所属, 氏名
    from 貸出履歴, 利用者マスター
    join 貸出履歴.利用者 ID = 利用者マスター.利用者 ID
    where 返却日時 is null
```

(a) ビュー「Q_貸出中」の定義

```
select 図書 ID, 書名, 著者, 発行所, 貸出日時, 所属, 氏名
    from 図書マスター, Q_貸出中
    join 図書マスター.図書 ID * = Q_貸出中.図書 ID
    where 書名 like '%Book%' and 著者 like '%Auth%'
    and 発行所 like '%Pub%'
```

(b) 主 Select ステートメント

図 5 図書検索 Select ステートメント
Fig. 5 Select statement of book catalog retrieval.

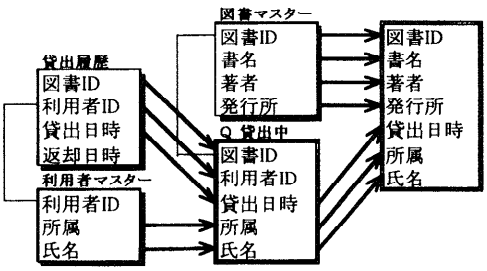


図 6 図書検索でのテーブル関連図
Fig. 6 The relation among the tables in book catalog retrieval.

このテーブル構成で貸出分析，図書検索などを行うには，貸出履歴テーブルを各マスターテーブルと結合しなければならない．テーブル結合は，検索処理の中で最も時間がかかる処理である．本研究では，結合処理方法として，連結先のテーブルのみソートしてから結合を行う方法を採用した．このインデックスはアプリケーションで意識しなくとも，システムが自動的に付加する方式とした．具体的な処理手順を事例にそって述べる．

図書検索の SQL ステートメントおよびテーブル関連図をそれぞれ**図 5** および**図 6** に示す．ここで，Book，Auth，Pub は具体的な検索語に置き換わる．

ビュー「Q_貸出中」では，「貸出履歴」が主テーブルである．まず，Where 句の評価実行により，「貸出履歴」テーブル（N 行）の対象となる行を抽出する．副テーブル「利用者マスター」の列「利用者 ID」（結合で使用されるキー項目）にインデックスを付ける．列のデータ型によらず，整数型と見なしてソートしておく．次に，バイナリサーチで副テーブルを検索する．

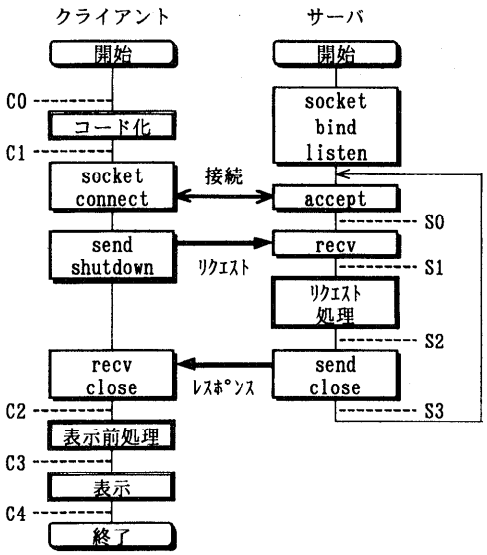


図 7 クライアント–サーバ通信モデル
Fig. 7 Client-server communication model.

図 5 (b) の Select ステートメントについても同様であるが，副テーブルはビューによって作られた一時的なテーブルであるため，キー項目「図書 ID」へのインデックスは Select ステートメントを実行するたびに付加される．

ソートには，コンパイラ標準ライブラリのクイックソート関数を使うことができる．しかし，この関数は，比較関数のアドレスを引数として受け取り，項目値の比較のたびに関数コールが発生する．比較回数の目安は $N \times \log(N)$ となるため，行数 N が大きいとき，このオーバヘッドは大きい．そこで，本研究では比較処理を含めたクイックソート関数を新たに作成し，関数コールを回避した．

なお，インデックスは結合におけるキー項目についてのみ使用しており，通常の列に対してはインデックスはいっさい使用していない．

5. 応答時間の実測結果および考察

5.1 測定方法

図書管理システムを事例として応答時間を実測した．クライアントとサーバ間の通信を中心とした処理の流れを**図 7** に示す．図書検索および雑誌ブラウジングを実行し，要所（C0, C1, C2, C3, C4 および S0, S1, S2, S3）で計時した．計時には，マシンサイクルレベルの精度が得られるパフォーマンスカウンタの値を用いた．ネットワークが比較的すいている状況で 5〜10 回測定して平均値を求めた．

表 2 図書管理システムの主要テーブル
Table 2 Main tables in the library management system.

テーブル	行数	列数
利用者マスター	750	3
図書マスター	20,000	10
雑誌受入記録	7,500	6
雑誌マスター	600	9
貸出履歴	25,000	8
合計	53,850	–

サーバと同一ハブにつながったクライアントと高速デジタル回線で LAN 間接続した遠隔地にあるクライアントを用いた．本稿では，以下，前者をローカルアクセス，後者をリモートアクセスと呼ぶ．測定に用いたサーバおよびクライアントマシンの仕様は次のとおりである．

- サーバ：Pentium Pro☆ 180 MHz プロセッサ，64 MB メモリ，Windows NT☆☆ Workstation 3.51
- クライアント：Pentium 120 MHz プロセッサ，32 MB メモリ，Windows 95

図書管理システムにおける主要テーブルを**表 2** に示す．この表以外にハッシュ表 434 KB，文字列 1 MB が存在し，全体では約 3.7 MB のメモリが使用される．一方，ファイルサイズは全体で約 4.8 MB であり，23%の圧縮効果が得られている．

クライアント側のソフトウェアとしては，精度の高い時間計測を行うために，先に開発した RDB ビュア[4] を用いた．なお，RDB ビュアは後述するブラウジングを実現するために機能拡張を行った．

5.2 図書検索

実際のシステムの図書検索では図 5 (b) の Select ステートメントで，貸出日時から時刻を取り除いた貸出日を出力させ，また，検索結果は受入日時（図書マスターの項目）が新しい順に並べ替えている．

5.2.1 サーバ処理時間

図書検索で書名でのフルテキストサーチを行ったときのサーバ処理時間（高速化対策前，対策後の値）を**表 3** に示す．ここで，デコードは HTTP（HyperText Transfer Protocol）の規約に従ってコード化して送られてきたデータを元に戻す処理である．検索 1，2 は検索語が英字であり，検索 3 は漢字である．英字の場合，全角/半角，大文字/小文字を区別しないため，あらかじめ書名を半角・小文字に変換しているが，検索

☆ Pentium Pro は米国 Intel Corp. の商標である．
☆☆ Windows NT, Windows 95 は米国 Microsoft Corp. の登録商標である．

表 3 図書検索でのサーバ処理時間（単位：ms）
Table 3 Server processing time of book catalog retrieval (ms).

項目		検索 1		検索 2		検索 3	
		対策前	対策後	対策前	対策後	対策前	対策後
デコード		1.6	1.6	1.7	1.6	1.7	1.7
構文解析		2.5	2.5	2.5	2.7	2.5	2.5
ビュー実行	構文解析	0.5	0.5	0.5	0.5	0.5	0.5
	選択処理 V	22.8	15.3	23.1	15.4	23.0	15.4
	テーブル作成	10.8	10.0	10.8	10.0	10.8	10.0
選択処理 M		162.8	99.1	163.6	99.8	147.0	77.9
インデックス作成		5.7	3.2	5.7	3.2	5.7	3.2
出力テーブル作成		6.9	6.7	2.4	2.3	2.0	1.8
出力テーブルソート		0.2	0.2	0.1	0.1	0.1	0.1
出力データ作成		3.4	3.3	0.3	0.3	0.3	0.2
合計		217.2	142.4	210.7	135.9	193.6	113.3

表 4 図書検索でのクライアント応答時間（単位：ms）
Table 4 Client response time of book catalog retrieval (ms).

項目	検索 1	検索 2	検索 3
サーバ処理	142.4	135.9	113.3
通信（ローカル：リモート）	14.7 : 80.6	7.0 : 37.7	7.2 : 36.3
クライアント処理	17.2	2.2	2.3
表示	23.5	10.4	10.6
合計（ローカル：リモート）	197.8 : 263.7	155.5 : 186.2	133.4 : 162.5

3では変換がいらないため，選択処理が22%速くなっている．

主Selectステートメントはビューより複雑なため，構文解析（字句解析を含む）時間は，ビューの構文解析の5倍の時間がかかっている．ビューの選択処理は25,000行が対象であるが，選択条件（Where句）がシンプルなため，高速に行われる．

高速化対策により，全体で75～80 ms，率では35～41%の処理時間短縮が図れた．選択処理Mおよび選択処理Vの高速化はそれぞれ主としてLike演算の改善および構文木の簡単化による．インデックス作成の高速化は，比較処理で関数コールのオーバヘッドを削除したことによる．

なお，ここでのインデックス作成時間は約1,500行（貸出中の本の数）が対象のため，3.2 msという小さい値であるが，利用者ごとの貸出中図書の検索では，インデックス作成行数は40,000行（図書20,000行．雑誌は表2の7,500行とは別の12,500行も対象となり，合わせて20,000行）と大きいため，インデックス作成時間が高速化対策前で225 ms，対策後で114 msと大きい．インデックス作成時間が貸出中図書検索時間全体の43%を占めているため，このケースではインデックス作成の高速化の効果は大きい．

5.2.2 クライアント応答時間

クライアント応答時間（C4–C0）は，通信時間，サーバ処理時間，クライアント処理時間，および表示時間からなる．クライアントからみた通信の経過時間は，C2–C1である．これにはサーバ処理時間S2–S1が含まれる．したがって，クライアントから見た正味の通信時間は(C2–C1) − (S2–S1)である．

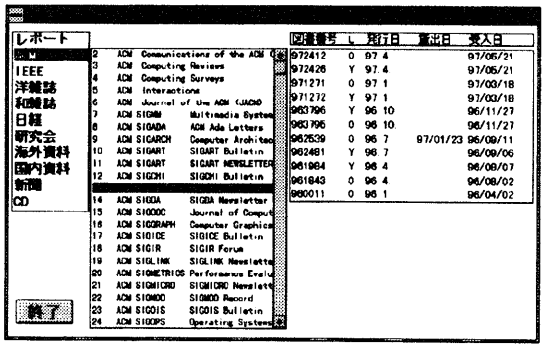


図 8 雑誌ブラウジング画面
Fig. 8 Screen of journal catalog browsing.

図書検索でのクライアント応答時間を**表4**に示す．ローカルアクセスでは133～198 ms，リモートアクセスでは163～264 msであり，ユーザに待ちを感じさせない性能が得られた．

文字列照合に時間がかかることから，サーバ処理時間がクライアント応答時間全体の53～84%を占める．

```
create view Q_雑誌マスター as
    select 分類, 分類名, 雑誌 ID, 誌名, 発行所
    from 雑誌マスター, 雑誌分類
    join 雑誌マスター.分類 = 雑誌分類.分類
create view Q_雑誌 as
    select 図書 ID, 雑誌 ID, 誌名, 発行日, 発行所, 所在, 受入日時
    from 雑誌受入記録, 雑誌マスター
    join 雑誌受入記録.雑誌 ID *= 雑誌マスター.雑誌 ID
```

(a) ビューの定義

```
SQL1 : select 分類名 from 雑誌分類
SQL2 : select 雑誌 ID, 誌名 from Q_雑誌マスター
       where 分類名=Delspc('SQL1')
SQL3 : select 図書 ID, 所在, 発行日, 貸出日=Getdate(貸出日時),
       受入日=Getdate(受入日時)
       from Q_雑誌, Q_貸出中 join Q_雑誌.図書 ID *= Q_貸出中.図書 ID
       where 雑誌 ID = Delspc(Left('SQL2',3)) order by 受入日 desc
```

(b) 主 Select ステートメント

図 9 雑誌ブラウジング Select ステートメント
Fig. 9 Select statement of journal catalog browsing.

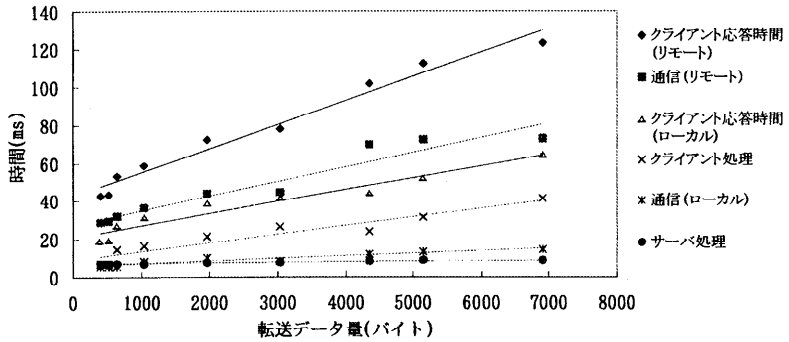


図 10 クライアント応答時間対転送データ量
Fig. 10 Client response time vs. transfer data size.

5.3 雑誌ブラウジング

図 8 に雑誌ブラウジング画面を示す．これはビジュアル・インタフェースの実現に向けた第一歩である．左端が雑誌の分類リストである．マウスボタンをクリックする必要はなく，マウスカーソルを雑誌分類語の上に移動させただけで，トリガーとなり，サーバに問合せコマンドが送られる．サーバからのレスポンスデータ（雑誌タイトルリスト）が中間に表示される．マウスを雑誌タイトルの上に移動すると，その雑誌の各号の発行日，受入日，貸出状況が右端に表示される．

このようなブラウジングでは特に迅速な応答が求められる．応答に時間がかかるようならば，マウスボタンをクリックして次の画面が表示されるのを待つインタフェースよりも使い勝手が悪いものとなる．

雑誌ブラウジングに関するビューおよび画面左端，中央，右端に対応する SQL ステートメント SQL1, SQL2, SQL3 を**図 9** に示す．ここで，引数'SQL1' にはマウスポインタが指す「分類名」が入る．Delspc 関数は引数の文字列からスペースを除去した文字列を返す．引数 'SQL2' にはマウスポインタが指す「雑誌 ID と誌名」が入る．SQL3 では，Left 関数により左から 3 バイトの雑誌 ID が取り出される．Delspc 関数により雑誌 ID の文字数が 3 バイト未満のとき余分なスペースを除去する．Getdate 関数は日付時刻型の引数を受けて時刻部を取り去り日付だけを返す．

雑誌タイトルリストの検索コマンド SQL2 に対するクライアント応答時間を**図 10** に示す．ここで，クライアント処理時間は表示時間を含んでいる．

サーバ処理時間が短いため，リモートアクセスの場合，通信時間がクライアント応答時間の大半を占めるが，クライアント応答時間は 0.1 秒未満と小さいため，実用上十分な性能である．

```
insert into 貸出履歴 (図書ID, 利用者ID, 所属, 貸出日時, 貸出IPA)
values('TID', 'UID', Lookup(利用者マスター.所属, 利用者マスター.利用者ID, 'UID'),
       Getdate(), 'IPA')
```

(a) 図書貸出処理における Insert ステートメント

```
update 貸出履歴 set 返却日時=Getdate(), 返却IPA = 'IPA'
where 返却日時 is null and 図書番号 = 'TID'
```

(b) 図書返却処理における Update ステートメント

図 11 Insert/Update ステートメント
Fig. 11 Insert/Update statement.

表 5 サーバ処理時間（単位：s）
Table 5 Server processing time (s).

	CGI 方式	API 方式	提案方式
図書検索	2.74～2.77	2.42～2.47	0.11～0.14
雑誌ブラウジング：SQL2	0.30～0.45	0.05～0.22	0.007～0.009
雑誌ブラウジング：SQL3	1.34～1.56	1.12～1.30	0.10

各号の所蔵情報検索コマンド SQL3 に対するサーバ処理時間はビュー「Q_雑誌」,「Q_貸出中」の実行時間が大半を占める．実測結果はそれぞれ 58 ms，16 ms であり，雑誌タイトルには関係しない．サーバ処理時間全体は 100 ms である．クライアント応答時間としては，この値に通信時間，クライアント処理・表示時間が加わり，ローカルアクセスでは 124 ms，リモートアクセスでは 149 ms である．

なお，ビュー「Q_雑誌」の実行結果は，雑誌の貸出，返却によって変化しない．そこで，外部データベースシステムでこのビューを実行し，その結果を雑誌マスターとする方式をとれば，ビュー「Q_雑誌」は不要となり，クライアント応答時間は 58 ms 短縮される．

5.4 データ更新

図書管理システムでは，図書貸出処理および図書返却処理で**図 11** に示す Insert ステートメントおよび Update ステートメントにより，貸出履歴テーブルの更新が発生する．ここで，TID，UID はユーザが入力した図書 ID，利用者 ID を表す．また，IPA は図書貸出，返却処理を行ったクライアントパソコンの IP（Internet Protocol）アドレスを表す．

Insert ステートメントのサーバ処理時間は 2 ms である．クライアント応答時間はローカルアクセスが 8ms，リモートアクセスが 32 ms である．

Update ステートメントの Where 句は簡単であり，サーバ処理時間は 26 ms である．クライアント応答時間はローカルアクセスが 31 ms，リモートアクセスが 55 ms である．

5.5 従来方式との比較

CGI 方式，API 方式，提案方式のサーバ処理時間を比較した結果を**表 5** に示す．CGI 方式，API 方式では，市販のパソコン RDB 管理システムを用いた．また，両方式では，検索および並べ替え処理の高速化のために，図書 ID，利用者 ID，雑誌 ID などにインデックスを付与した．

実測結果では，CGI 方式と API 方式の差は 20～30 ms である．提案方式は，CGI 方式に比べて 7 倍以上，API 方式に比べて 11 倍以上という高い性能が得られている．

6. おわりに

本論文では，小規模システムを対象として，データベース全体をメインメモリ上においてメモリベースでデータ操作を行う WWW ベースのデータ検索システムを提案した．

提案システムのデータ操作コマンドは SQL に準拠しているが，通常の RDB 管理システムと比較すると，テーブル設計の負担は少なく，また，インデックス設計が不要なため，アプリケーションの開発が容易であるという特徴を有している．

システム構成としては，WWW サーバ部とデータ操作部の一体化を図り，インタフェースのオーバヘッドを極小化した．また，データ操作部では構文木の簡単化，関数コールの回避などの工夫により高速化を図った．

事例として，図書管理システムに適用して，その有効性を検証した．

今後の課題としては，通信時間の短縮があげられる．HTTP/1.0 プロトコルでは，リクエストごとに接続・切断を行うために通信時間の最小値が RTT（Round Trip Time）× 2 となる[7]が，ある期間接続したままで交信する方式をとれば，通信時間の最小値を RTT

にまで下げることができる．また，今回開発したシステムでは，クライアントの負担削減のために，レスポンスデータはテキスト形式またはHTML形式としている．Java☆などを使用し，何らかの形で圧縮したレスポンスデータを返す方式をとれば，通信時間の短縮が可能となる☆☆．

参考文献

1) WWW–データベース連携システム構築法，日経BP社 (1996).
2) 大磯，小野沢，木下，仲山，早瀬：イントラネットのためのオブジェクト指向データベース技術，ソフト・リサーチ・センター (1996).
3) 西尾章治郎（監修）：実践SQL教科書，アスキー出版局 (1996).
4) 畑田　稔，遠藤裕英：WWW-RDB連携システムの開発，情報処理学会論文誌，Vol.38, No.2, pp.349–358 (1997).
5) 正田輝雄，石畑　清：コンパイラの理論と実現，共立出版 (1988).
6) 石畑　清：アルゴリズムとデータ構造，岩波書店 (1989).
7) Stevens, W.R.: *TCP/IP Illustrated*, Volume 3, pp.9–16, Addison-Wesley (1994).

(平成 9 年 8 月 26 日受付)
(平成 10 年 7 月 3 日採録)

☆ Java は，米国およびその他の国における米国 Sun Microsystems, Inc. の商標である．

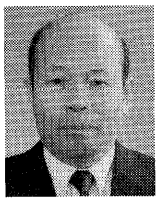
☆☆ たとえば，日付時刻型データを文字列ではなく，内部形式のバイナリデータで送れば，17 バイトのところが，4 バイトに短縮される．

畑田　稔（正会員）

1972 年京都大学大学院工学研究科博士課程修了．同年（株）日立製作所入社，現在，システム開発研究所に勤務．制御系の安定問題，大規模システムの解析と評価，複合マイクロコンピュータシステム等の研究を経て，インターネット・イントラネット技術を活用した情報サービスシステムの研究開発に従事．1971 年電気学会論文賞受賞．京都大学工学博士．システム制御情報学会会員．

野里真喜子（正会員）

1986 年津田塾大学学芸学部数学科卒業．同年（株）日立製作所入社．システム開発研究所に勤務．現在，研究者向け情報サービス業務に従事．



遠藤　裕英（正会員）

1941 年生．1965 年京都大学工学部電子工学科卒業．同年（株）日立製作所入社．中央研究所主任研究員，マイクロエレクトロニクス機器開発研究所部長，システム開発研究所情報センタ長を経て，1998 年 4 月から立命館大学理工学部教授．この間，制御用計算機，パターン認識装置，ワープロ・パソコン，研究所情報サービスシステム等の研究開発に従事．京都大学博士（工学）．電子情報通信学会，ACM 各会員．