画像データベースにおけるメタデータの動的な検索手法の検討
－リレーショナル DMBS からの Exif アクセス－

末永 恭正† 井上 潮‡

東京電機大学工学部 〒101-8457 東京都千代田区神田錦町 2-2

E-mail: †02kc065@ed.cck.dendai.ac.jp, ‡inoue@c.dendai.ac.jp

あらまし 画像データベースでは、画像本体とそのメタデータを別々に管理することが一般的である。しかし、この方法ではデータベース上でデータが重複し、メタデータと画像本体の間に矛盾が生じる可能性や、データ追加時にメタデータを抽出しなければならないという問題が生じる。そこで我々はデータベースに格納された画像データからメタデータを動的に抽出する方法を検討している。具体的には画像データを BLOB 領域に格納し、その中に含まれる Exif 情報に対してデータベース組込型のライブラリを用いた SQL 文による検索を可能にする。本論文では、このライブラリの設計・実装・評価結果、そしてアプリケーションの一例として構築した画像コミュニティサイトのプロトタイプについて述べる。

キーワード メタデータ管理，画像データベース，WWW，Exif

1. はじめに

現在販売されているデジタルカメラやカメラ付き携帯電話の多くには Exif と呼ばれる画像ファイル規格が採用されている。これは TIFF 形式[1]を拡張したもので、JPEG 画像にカメラ名や撮影日時などの撮影情報が付加されたものである。Exif の最新バージョンである Exif2.2 [2]は、ExifPrint という別称があるように、家庭用プリンタでデジタルカメラや携帯電話からのダイレクトプリントを行うことを想定した規格である。具体的には、プリンタの液晶パネルに印刷対象となる画像のサムネイルや撮影した日時、画像サイズを表示したり、撮影情報をもとに画像を解析して最適な補正と印刷を自動的に行ったりするものである。

Exif は、画像処理を行う商用のソフトウェアのほとんどでサポートされており，フリーソフトでも、吉本の ExifReader [6]、杉本のカシミール 3D [7]などでサポートされている。また、SourceForge には Linux 上で動作する libexif [8]という共有ライブラリの開発プロジェクトも存在する。このライブラリは Linux の各ディストリビューションに組み込まれているパッケージとしても有名である。

以上のように、Exif は広く普及しているが、画像データベースの検索においては十分に活用されているとはいえない。その原因は、Exif からのメタデータ抽出の問題とデータベース上での管理の問題によるものと考えられる。メタデータ抽出の問題とは、画像の検索に必要十分なメタデータを抽出することが、以下の理由により困難なことである。

- Exif から取得できるメタデータの種類が限定されている。
- Exif で定義されているデータ型すべてをサポートしていない。
- Exif のバージョンに依存するため、必要なメタデータを取得できないことがある。
- データベース設計時に、検索で使用するメタデータを選択し、これを格納するための適切なテーブル構造を定義しなければならない。

一方のデータベース上での管理の問題とは、抽出されたメタデータと画像本体を一元的に管理することが、以下の理由により困難なことである。

- 画像データ内にあるメタデータとそこから抽出されたメタデータの重複が生じ、画像の挿入や変更時にユーザのミスやシステムのバグなどによって整合性が損なわれる可能性がある。
- データベースを再構築する場合に、メタデータを再入力する必要が生じる。

これらの問題を解決するために、我々はデータベース組込型のライブラリを用いた動的な Exif 情報へのアクセス方法を提案する。これは Exif 規格に基づいた画像データ全体をデータベースの BLOB(Binary Large Object)領域に格納し、メタデータとして使用する Exif 情報へのアクセスは、SQL 文から組み込み型ライブラリ経由で動的に行うものである。この手法によれば、データベース内に同じメタデータが重複することはな

くなり、また面倒なデータベース設計や画像データ登録時の前処理を必要としない。

本論文では、データベースに格納された Exif 規格の画像データから Exif 情報を動的に抽出する組み込み型ライブラリの設計・実装・評価結果、そしてアプリケーションの一例として構築した画像コミュニティサイトについて述べる。

以下、２章で Exif 規格を説明し、３章でライブラリの設計方針、４章ではその実装方法について述べる。５章で作成したライブラリの評価について述べる、６章では Web アプリケーションの一例を紹介し、７章で関連技術との比較、最後に 8 章で全体のまとめと今後の課題について述べる。

2. Exif の規格

2.1. JPEG ファイルの構造

Exif 規格は JPEG ファイルのフォーマットを拡張したものである。そのため、まず最初に JPEG ファイルの構造について述べる。

表１　JPEG ファイル構造

アプリケーション定義データ
⋮
サムネイル画像
画像データ

JPEG は表１のような構造になっている。アプリケーション定義データはマーカと呼ばれる２バイトの数値によって識別され、それに関連づけられたデータを保持することができる。

Exif 情報は、アプリケーション定義データの１つであり、アプリケーションマーカ（APP1）と呼ばれるマーカによって、JPEG 画像に埋め込まれる形で提供される。

2.2. Exif 情報の構造

APP1 マーカに格納されている Exif 情報の構造を表２に示す。

表２　Exif 構造

Exif Header
TIFF Header
0th IFD
Exif IFD
1st IFD
JPEG Thumbnail

Exif 情報は TIFF 形式[1]を基に定義されており、メタデータを格納するために Image File Directory(IFD) と呼ばれるディレクトリ構造を用いる。１つの IFD はタグ番号、データ型、データの個数、データ実体へのオフセットを１つのエレメントとしたディレクトリ構造である（表３）。

表３　IFD の構造

Directory Count
Directories
Tag
Data Type
Count
Offset to Data
⋮
Offset to Next IFD

IFD の先頭にはその IFD が保持しているエレメントの個数が格納され、最後尾には次の IFD へのオフセットが格納されている。IFD は１つのアプリケーションマーカ内に複数存在することが可能であり、指定された IFD とタグ番号から該当データの実体とデータ型、個数を特定し、取得することができる。また、Exif ではサムネイル画像の格納をサポートしており、JPEG と非圧縮の画像を組み込むことが可能になっている。

3. ライブラリの設計

3.1. 前提条件

本研究では, 画像データベースを一般的なリレーショナル DBMS を使用して構築するものとし、画像データは DBMS が提供する BLOB 領域に格納することを前提とする。また、画像データの BLOB 領域との入出力は、DBMS が提供する組み込み型の BLOB 関数またはユーティリティプログラムを使用することとする。一般に、DBMS は BLOB 領域に格納されたデータ内部の構造は意識しないため、標準の SQL 文では内容検索を行うことができない。そのため、BLOB 領域に格納された画像データから Exif 情報を動的に抽出するための組み込み型のライブラリを開発することが必要になる。

3.2. 設計目標

上記の前提条件に基づいたライブラリの開発に際して、以下の設計目標を設定した。

(1) SQL 文中で使用できる関数として実現する。

SQL 文から呼び出し可能な関数として実現することにより、SELECT 文による Exif 情報の出力や、WHERE 句による検索条件の設定を可能にする。

(2) Exif のバージョンに依存しない。

バージョンによって IFD およびタグ番号が追

加・変更になっても、同じライブラリでアクセスすることを可能にする。

(3) **可能な限り処理を高速化する。**
Exif 情報内部の探索処理を可能な限り高速化することにより、検索時の動的な抽出処理にかかるオーバヘッドを削減する。

(4) **動作環境に依存しない。**
CPU アーキテクチャによって規定されるバイトオーダや、OS が提供する API の相違を吸収することにより、DBMS が走行する多様な環境で利用することを可能にする。

(5) **既存のシステムにも容易に組み込める。**
インストール作業の簡易化によって、DBMS の構成変更やデータベースの再構築を不要にし、DB 管理者の負担を軽減する。

3.3. 設計方針

開発するライブラリが対応する DBMS は PostgreSQL[3][4]とした。その理由は、オープンソースであること、組み込み関数の作成が容易なことによる。

本ライブラリの基本機能は、Exif のバージョンに依存しないようにするため、2.2 節で述べた IFD とタグ番号の組み合わせで値を取得することにした。Exif がサポートするデータ型すべてを取得可能とし、分数型など SQL 文がサポートしていないデータ型に関しては、倍精度浮動小数型に変換するかバイナリ型で出力して、その後の処理を呼び出し元のアプリケーションに委ねることにした。

さらに、DB 管理者による本ライブラリの導入および検証を容易にするため、本ライブラリの関数を用いて Exif 情報の出力を行うテストアプリケーションを提供することにした。

3.4. 処理方式

SQL 文から本ライブラリの関数が呼び出されると PostgreSQL から、BLOB 領域に格納された画像データと、IFD を識別する整数値、取得対象のタグ番号が引数として渡される。これらの引数を受け取った関数は、画像データが JPEG 規格であるか、そして Exif 規格であるかをチェックする。このチェックを通過すると画像データのバイトオーダを取得する。もし自身の動作する CPU のバイトオーダと異なる場合にはデータを読み込む際にバイトオーダの変換が必要なことを示すフラグを true に、それ以外では false に設定する。

また Exif 規格では、IFD に存在するディレクトリのタグ番号は昇順に並んでいることが規定されている。そのため、取得対象のディレクトリを決定する部分では C 言語の標準関数である bsearch()を利用してバイナリサーチを行う。もし該当データが存在すれば取得結果を、存在しない場合は NULL を返す。

本ライブラリが提供する関数のすべてには、STRICT と IMMUTABLE の２つのオプションが付加される。STRICT オプションとは、関数の引数に NULL が渡された場合、システムが自動的に戻り値も NULL であるとみなす PostgreSQL 固有のキーワードである。これは、2.2 節で述べたように Exif では IFD とタグ番号で取得すべきデータが決定され、本ライブラリの提供する関数の引数に NULL を渡すことが許されないからである。IMMUTABLE 属性とは、同一引数に対する関数の呼び出しには、常に同一の結果を返すことを保証する、PostgreSQL 固有のキーワードである。IMMUTABLE 属性を付加することで関数インデックスを利用することができ、Exif 情報を検索条件として使用する場合にインデックススキャンを利用できるというメリットもある。

4. 実装

4.1. ライブラリの構築

PostgreSQL の組み込み関数として作成する関係上、メモリ管理などは PostgreSQL から提供されている関数を使用する。関数インタフェースは最新の Version-1 規格とし、データの受け渡しでプログラマが意識すべきことを低減した。

PostgreSQL 用共有ライブラリとテストアプリケーションを構築する場合、エントリポイントやメモリ管理などビルドターゲットごとに異なる点が多数存在する。１つのソースファイルで異なるプラットフォームに対応できるようにするため、プリプロセッサによる条件コンパイルや、Makefile によるビルドターゲットごとのファイル差し替えによってこの問題を解決した。

また、テストアプリケーションとして PostgreSQL 向け共有ライブラリと同一の Exif 解析ロジックを持ったコンソールアプリケーションを提供する。Windows 環境ではコンソールアプリケーションに加え、GUI アプリケーション（図１）も提供する。

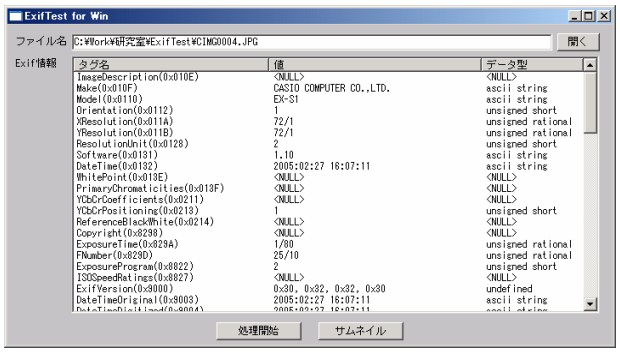

図１　GUI アプリケーションの実行例

本ライブラリが提供する関数とその引数、機能を表４に示す。

表４　提供関数一覧

関数名	引数	戻り値	機能
GetExifString	bytea, integer, integer	text	取得対象の Exif を文字列型で取得する。
GetExifThumbnail	bytea	bytea	サムネイル画像を取得する。
GetExifBinary	bytea, integer, integer	bytea	取得対象の Exif をオリジナルのデータ型で取得する。
GetExifInt	bytea, integer, integer	integer	取得対象の Exif を整数型で取得する。
GetExifDouble	bytea, integer, integer	double precision	取得対象の Exif を倍精度浮動小数点型で取得する。
GetExifDataType	bytea, integer, integer	integer	取得対象の Exif のデータ型を取得する。
GetExifDate	bytea, integer, integer	timestamp	取得対象の Exif をタイムスタンプ型で取得する。

Exif から取得できるデータ型は大きく分けて小数型、整数型、文字列型に分類できる。それぞれのデータ型を PostgreSQL 標準のデータ型に変換して呼び出し元に返却する。さらにサムネイル画像とタイムスタンプを取得する関数を用意し、今後の Exif のバージョンアップやメーカ独自定義のタグにも対応できるよう、バイナリデータをそのまま呼び出し元へ返却する関数も用意した。また、呼び出し元が取得しようとする Exif のデータ型を判断できるよう、その値を返す関数も実装した。

4.2. ビルドターゲットとインストール方法

今回作成したライブラリとテストアプリケーションは、Windows と Linux の２種類の OS 上で動作する PostgreSQL に組み込んだ状態で確認した。他の OS でも PostgreSQL の動作する環境であればこのライブラリをコンパイルし、組み込むことが可能である。

インストールに関しては、コンパイルから行う場合は PostgreSQL のヘッダを参照する関係上、PostgreSQL のソースツリーがあることが前提条件であり、バイナリファイルをコピーする場合は PostgreSQL が参照可能なディレクトリに DLL 形式のライブラリをコピーする。そして組み込み型関数を登録する CREATE FUNCTION 文を実行し、インストールが完了となる。

5. 性能評価

前述のように、本ライブラリは PostgreSQL に組み込み、BLOB 領域に登録された画像から動的に Exif を取得するものである。本ライブラリを使用するにあたって、DBMS から関数に渡される引数をスタックに積む際のオーバヘッド、そして Exif 解析時に発生するオーバヘッドによる DBMS からのレスポンスタイムの増加が考えられる。

本章ではこれらの起こりうるデメリットが、運用上許容できる範囲であるかを検証する。

5.1. 評価環境

作成した共有ライブラリの動作を検証するために、表５に示す２種類のプラットフォームと、表６に示す２種類の画像ファイルを用意した。

表５　評価プラットフォーム

項目	サーバ１	サーバ２
OS	Vine Linux 3.1	FedoraCore4 Linux
カーネル	2.4.17	2.6.11
CPU	PowerPC 266MHz	CeleronD 2.5GHz
メモリ	128MB	512MB
スワップ	256MB	1024MB
HDD	80GB 7200rpm	80GB 7200rpm
DBMS	PostgreSQL 8.1.0	

表６　登録画像

画像名		画像１	画像２
画像内容			
カメラ		CASIO EX-S1	DoCoMo N900iS
バイトオーダ		Big Endian	Little Endian
サイズ	Width	640px	240px
	Height	480px	269px
ファイルサイズ		134,350byte	11,618byte

サーバ１には家庭でファイルサーバとして使用するもの、サーバ２には一般的な PC を用意し、パーソナルユースでも実用に耐えうるかを検証する。これは、ホームサーバとなりうるマシンに組み込んでもストレ

スない動作が可能であるか、そしてこれらのマシンで軽快な動作が可能であればスペックがはるかに高いエンタープライズ向けデータベースサーバに本ライブラリを組み込んだ場合ではさらにそのオーバヘッドが問題にならないレベルになると考えられるからである。

また、サーバ１と２では CPU のバイトオーダが異なるため、同じ画像を登録した場合でも、バイトオーダ変換作業でのオーバヘッドがどの程度になるのかを計測することができる。

画像ファイルはデジタルカメラで撮影したものと携帯電話で撮影したものの２種類を用意する。それぞれ画像サイズ・バイトオーダが異なるため、サーバ１と２でベンチマークを行うことでバイトオーダや画像サイズが本ライブラリでの Exif 解析においてどのような影響を与えるかを計測する。

PostgreSQL には次のようなテーブルを作成する。今回のテストでは検索速度とサーバにかかる負荷を計測するため、データベースに登録された列と Exif を動的に取得する場合の、全データのシーケンシャルスキャンを実行する時間を計測する。

表７　テストテーブルの構成（表名は test）

列名	データ型	備考
str	text	予め取得したカメラ名が格納される
img	bytea	画像ファイルが格納される

5.2. 評価方法

登録件数の増加に伴うシーケンシャルスキャンの速度と負荷を計測するために、表７に示すテーブルに 1 件、10 件、100 件、1000 件と順次登録する。サーバ上で動作する他のアプリケーションの割り込みによる誤差などを考慮するために計５回計測し、その平均の実行時間で動作スピードを検証する。計測には次の３つの SQL 文を実行する。

①SELECT str FROM test;
②SELECT img FROM test;
③SELECT GetExifString(img, 272, 1) FROM test;

これら３つの SQL 文の実行時間を計測するために、本ライブラリの評価用にクエリ実行時間計測用アプリケーションを作成した。このアプリケーションでは、表５で示した各サーバに libpq と呼ばれる PostgreSQL インタフェースライブラリを通してアクセスし、それぞれの SQL 文を実行する直前と直後の時間を SVr4, 4.3BSD に準拠した関数である gettimeofday()を用いてマイクロ秒まで取得する。クエリ実行直前と直後の時間差をクエリ実行時間とする。

この検証を行うことにより、登録画像数の増加による処理時間の増加や、画像の種類・サーバマシンの違いによる負荷の違いを計測することができる。

5.3. 評価結果

5.2 節で示した検証方法によって得られた計測結果のグラフを図２～図５に示す。各図は、横軸がテーブルサイズ、縦軸が実行時間の両対数グラフになっている。

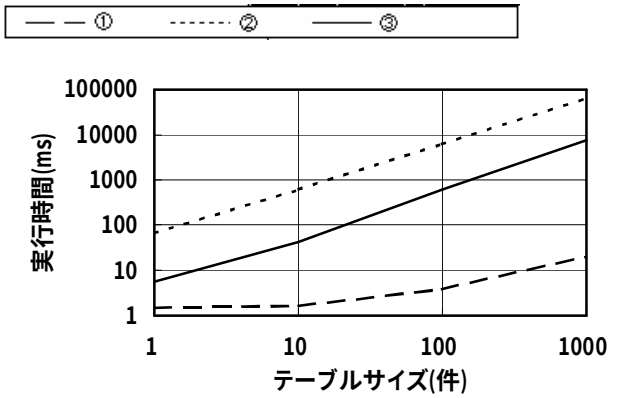

図２　サーバ１と画像１の組み合わせ時の性能

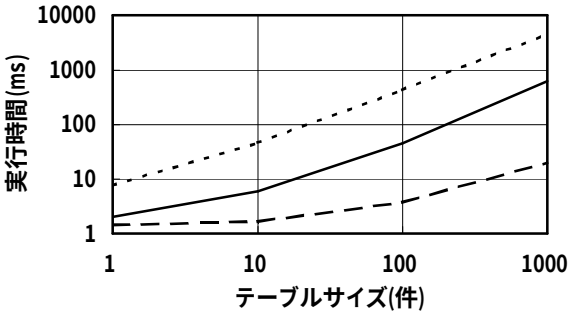

図３　サーバ１と画像２の組み合わせ時の性能

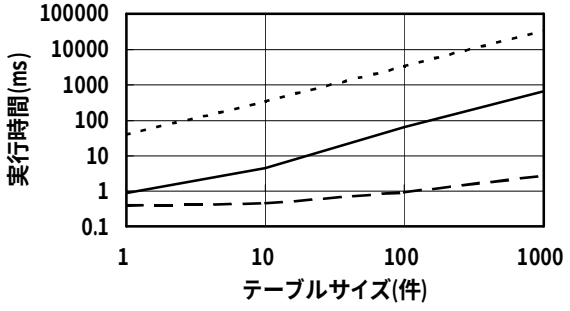

図４　サーバ２と画像１の組み合わせ時の性能

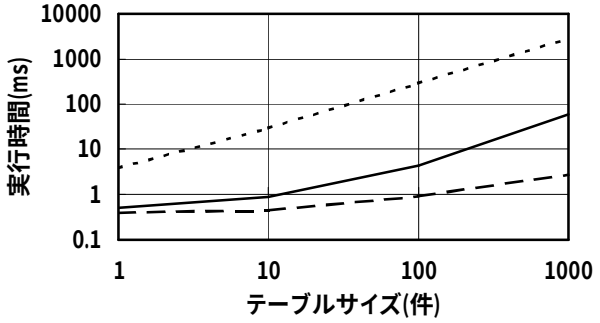

図５　サーバ２と画像２の組み合わせ時の性能

すべてのグラフにおいて、①、③、②の順で実行速度が速い。図２と図４の組み合わせと図３と図５の組み合わせで比較すると、データベースサーバとなるマシンの性能に比例して①のクエリと③のクエリの実行時間の差が小さくなる。また、図２と図３の組み合わせと図４と図５の組み合わせで比較すると、BLOB 領域に格納する画像サイズに関連して①のクエリと③のクエリの実行時間が小さくなる。さらに、この組み合わせにおいては、②のクエリと③のクエリの傾きがほぼ同一であることから、BLOB 領域に格納されたデータを取得するコストと、そこからメタデータを取り出すコストは比例関係にあると考えられる。

これらのことより、設計目標の１つに挙げた、高速な処理は十分達成されたとは言えないが、検索条件を絞り込むことにより取得する Exif データを限定できる場合や高速なマシンで利用する場合、画像データ本体ではなく、そこに格納されたメタデータのみを必要とする場合には十分に実用に耐えうると考える。

6. 画像コミュニティサイトへの応用

本ライブラリが画像データベースの構築に利用可能であること、そして Exif に格納されているメタデータが画像データの検索に有用であることを確認するために、Exif を用いた画像コミュニティサイトを作成した。このサイトの主な機能は次の通りである。

(1) Exif 情報の表示
(2) Exif 情報の検索機能
(3) サムネイル画像の抽出
(4) XML への出力機能

(1)～(3)に関しては、作成したライブラリによって動的に画像データから Exif 情報を取得し、活用している。(4)に関しては、ユーザが指定した画像に関する情報を XML ファイルに出力し、画像とともに ZIP アーカイブすることでクライアントへダウンロードさせる機能である。この XML ファイルにも Exif 情報が含まれる。

このサイトのサーバソフトウェアはすべてオープンソースソフトウェアで構築されている。詳細を表８に示す。

表８　サーバ環境

OS	FedoraCore3 Linux
カーネル	2.6.11-1.27_FC3smp
CPU	Intel Pentium4　3.0GHz
メモリ	1GB
スワップ	1052248 kB
DBMS	PostgreSQL 8.0.3
Web サーバ	Apache 2.0.54
AP サーバ	Jakarta Tomcat 5.5.9
JDK	Java2 Standard Edition 5.0 Update 3
JDBC	PostgreSQL JDBC3 8.0 Build 304

５章で述べたように、開発したライブラリは動作速度が遅いため、単純に SELECT 文を実行すると画像枚数の増加に伴って DBMS からのレスポンスタイムが増加する。そのため、画像を格納しているテーブルのプライマリキーで取得対象画像を絞り込み、Exif 情報を取得する際に発生するオーバヘッドを極力減らすようにしている。

Exif 情報による検索条件の設定では、カメラ名と撮影日時による絞り込みが可能になっている。この２つに関しては、3.4 節で述べた IMMUTABLE 属性によりサポートされる関数インデックスを設定することで検索条件に指定された場合の高速化を図っている。

サムネイル画像は画像データから動的に抽出することによりデータベース容量の削減を実現している。

XML 出力機能では、ユーザが指定した画像と、それに関するデータをサーバ側で ZIP アーカイブし、ダウンロードすることができる。XML 文書の定義する DTD は Web サーバ上に配備され、XML 文書展開時に自動的にそこを参照し、妥当性を検証するようにしている。

このサイトを利用するユーザは、画像の所有者名、タイトル、撮影日時、カメラ名を組み合わせて検索することができる（図６）。表示されるサムネイルは画像データから動的に抽出したものである。画像データ詳細表示画面（図７）では画像ビューワとして Java Applet を使用し、画像の拡大・縮小・回転・移動を自由に行うことができる。表示項目には Exif 情報から動的に抽出した、撮影したカメラ名と撮影日時が含まれる。

このサイトにアップロードした画像はその閲覧権限を一般公開、登録ユーザのみ、指定したユーザグループ参加者またはユーザのみ、所有者自身のみと４段階に設定することができる。ユーザグループとは、ユーザ同士のコミュニティであり、各ユーザがグループを自由に立ち上げることができる。既存のグループに参

加したい場合はグループ管理者に参加要求を送信し、その結果参加が許可されればそのグループに属することができる。

図 6　Exif を使用した検索画面

図 7　画像データ詳細表示画面

7. 関連技術

本論文で提案したリレーショナル DBMS 上でメタデータを扱う機能と類似した機能は、現在いくつかのベンダによって提供されている。本章では、それらに採用されている技術の特徴と、提案手法との相違点について述べる。

7.1. Oracle interMedia

Oracle interMedia は、Oracle 社の Oracle Database の Standard Edition および Enterprise Edition に付属する、異機種間メディアデータを他の情報と統合したフォーマットで管理する機能である。

マルチメディアデータの格納は、本ライブラリと同様に、BLOB 領域に格納する。マルチメディアデータに含まれるメタデータは interMedia の制御により、システムテーブルに格納され、その値を利用する。また、interMedia は拡張可能であり、標準でサポートしていないメディアデータに対しても、パッケージまたは PL/SQL プラグインの形でサポートが可能になる。一方、本論文で提案したライブラリはメタデータを BLOB 領域に格納されたデータから直接読み込むことができるため、この点が異なる。

7.2. DB2 Extender

DB2 Extender とは、IBM 社の DB2 Universal Database 向けの機能拡張パッケージである。画像データを対象とする Image Extender では、本ライブラリと同様に BLOB 領域に格納された画像データに対してユーザ定義関数・データ型を用いてアクセスする。画像サイズやサムネイルなど、画像に関連するデータは、前述の interMedia と同様に DBMS 内部で格納され、その値を利用する点が本ライブラリと異なる。

8. まとめ

本論文では、画像データ中に格納されている Exif 情報を活用した画像データベースの検索手法を提案した。画像データからのメタデータ抽出処理の問題と、データベースでの管理上の問題を解決するため、画像データを BLOB 領域にそのまま格納し、検索時に DBMS 組み込み型のライブラリによって Exif 情報を動的に抽出し、検索条件の判定やメタデータの表示に使用する方法について述べた。さらに、性能面でも実用に耐えること、具体的なアプリケーションの構築に利用できることを示した。

現在普及しているマルチメディアフォーマットには、AVI や MPEG、MP3 など Exif と同じようにメタデータが組み込まれているものが多い。このようなマルチメディアデータをオン・デマンドで提供する Web サイトではメタデータを効率よく扱うことが課題となっており、データベースに登録されている内容の信頼性を高める意味でもデータの重複を回避することが望ましい。

本論文で提案した手法は、Exif 以外のメタデータが組み込まれたマルチメディアデータに対しても、原理的にそのまま適用可能であり、マルチメディアデータベースの有力な実現方法の一つになると考えられる。例えば MP3 の場合、その曲のタイトルやアーティスト名などの楽曲情報が ID3 と呼ばれるタグ形式で格納されている。このデータを活用した場合、音楽ファイルを DBMS に格納しておくだけでその音楽ファイルに関する様々な情報に対して、他にデータ格納領域を作ることなく SQL 文経由でアクセスすることができる。また、ID3 の場合はフリーソフトウェアなどによってユ

ーザが自分の好きなようにメタデータを編集しやすい環境にある。このため、本ライブラリと同様に ID3 に対応した DBMS 組み込み型共有ライブラリを作成し、利用することで、音楽ファイルとそれに含まれるメタデータの整合性を保つことができる。

今後の課題としては、データベースに格納された画像データから Exif 情報を抽出する速度の向上と、Exif 情報の書き込み機能の実現が挙げられる。また、Exif 以外のマルチメディアデータをサポートするために、ライブラリの拡張も行っていく必要がある。

抽出速度向上に関しては、Exif 情報抽出ロジックの見直しやアセンブラ命令を用いた、低レベルな部分からのプログラム最適化が考えられる。ただし、アセンブラ命令はビルドターゲットの CPU アーキテクチャに依存する場合があるため、アプリケーション構築の際は注意が必要である。

Exif 情報の書き込みに関しては、安易に実装するとメーカノートと呼ばれるメーカ独自定義データ内部に存在するデータのオフセットがずれてしまい、別のアプリケーションで変更後の Exif 情報を読み込もうとしたときにメーカノートの内容が正しく読み込めなくなってしまう可能性がある。そのため、メーカノートに影響を与えないデータ書き込み手法を考える必要がある。

Exif 以外のマルチメディアデータをサポートするためには、メタデータをリレーショナル DBMS 上で扱うためのフレームワークを作成し、各メタデータに対して１つのプラグインといった形でメタデータアクセスモジュールを提供する手法が考えられる。このフレームワークの仕様を一般に公開し、それぞれのマルチメディアデータフォーマットに対する知識のある人間が自由にプラグインを開発できる環境を整える。

このフレームワークの考え方は、上で指摘したメーカノートにも適応できる。各メーカノートフォーマットへのアクセスを１つのプラグインとして実装すると、それによってメーカノートにアクセス可能な画像であれば、メーカノートの読み込みだけでなく、メタデータ全体の更新も可能になる。

文　献

[1] Adobe Developers Association, TIFF™ Revision 6.0 Final, June-3,1992.
(http://www.adobe.com/Support/TechNotes.html)

[2] JEIDA, Exchangeable image file format for digital still cameras:Exif Version 2.2, Apr.2002.
(http://www.exif.org/Exif2-2.PDF)

[3] 日本 PostgreSQL ユーザ会
(http://postgresql.jp)

[4] Korry Douglas, Susan Douglas, PostgreSQL Second Edition, Sams Publishing, 2005.

[5] 影山瑛，最新サーブレット&JSP ハンドブック，秀和システム，東京，2002.

[6] 吉本龍司，Exif Reader
(http://www.rysys.co.jp/exifreader/jp/)

[7] DAN 杉本，カシミール 3D
(http://www.kashmir3d.com/)

[8] curtisg, hun, lutz, libexif
(http://libexif.sourceforge.net/)

[9] Oracle Corporation, Oracle interMedia ユーザーズ・ガイド, 10g リリース 1(10.1), February, 2004

[10] IBM データベース
(http://www-06.ibm.com/jp/software/data/db2/)