PostgreSQLにおける拡張インデクス機能を有する表構造ファイルラッパ

藤田　悦郎† 　川田　明良† 　小谷　尚也† 　日高　東潮† 　山室　雅司†

† NTT サイバースペース研究所
〒 239–0847 神奈川県横須賀市光の丘 1-1
E-mail: †{fujita.etsuro,kawada.akiyoshi,kotani.naoya,hitaka.toshio,yamamuro.masashi}@lab.ntt.co.jp

あらまし　DBMS における外部データ管理に関する SQL 標準 SQL/MED において規定されている外部データラッパは、DBMS の外部にあるデータを DBMS にロードすることなく通常の表と同様に参照するための SQL 標準であり、近年のデータウエアハウスへの関心の高まりに伴い、組織内に分散するデータを効率的に統合する手法として重要性が再認識されてきている。本研究では、PostgreSQL の次期バージョン 9.1 において導入が予定されている、CSV ファイルなどの表構造ファイルに対応した外部データラッパにおいて、表構造ファイルへのインデクススキャンを可能にするための SQL/MED 規格を拡張したインデクス機能ならびにインデクススキャンの実行を最適化可能にするための問合せ最適化機能を導入することで、当該外部データラッパの参照性能を向上させる実装方式について述べ、評価実験により有効性を示す。

キーワード　SQL/MED, 外部データラッパ, 表構造ファイル, インデクス, 問合せ最適化, PostgreSQL

File Wrapper having Extended Index Function in PostgreSQL

Etsuro FUJITA†, Akiyoshi KAWADA†, Naoya KOTANI†, Toshio HITAKA†, and Masashi YAMAMURO†

† NTT Cyber Space Laboratories
1-1 Hikarinooka, Yokosuka, Kanagawa, 239–0847 Japan
E-mail: †{fujita.etsuro,kawada.akiyoshi,kotani.naoya,hitaka.toshio,yamamuro.masashi}@lab.ntt.co.jp

1. はじめに

企業や団体など組織内に分散するデータを統合して意思決定に活用するデータウエアハウスへの関心が高まってきている。このようなデータウエアハウスを構築するには通常、業務システムに蓄積されたデータを抽出 (Extract) し、利用しやすい形に加工 (Transform) し、DBMS にロード (Load) する、いわゆる ETL 処理を行う必要があるが、対象となるデータが大規模になるとロードに要する時間が増大して問題になることが知られている。このような問題を解消するため、これまでにロードの並列化などによる高速化手法が提案されているが、増え続けるデータに対して必ずしも対応できていないのが現状である。

このような中、DBMS の外部にあるデータを DBMS にロードすることなく直接参照して分析を行う手法が注目を集めてきている [3]。このような手法にはユーザ定義関数 (ストアドプロシージャ) を用いる手法や、DBMS における外部データ管理に関する SQL 標準 SQL/MED[1] において規定されている外部データラッパを用いる手法がある。いずれも DBMS の外部にあるデータを直接参照するための一連の手続きを DBMS へのアドオン形式プログラムとして作成することで実現するものであるが、後者ではプログラムの作成において問合せ処理に関する DBMS の内部知識が必要になるため、前者に比して実装が難しくなる。しかしながら、前者では問合せ最適化への対応が難しいため、結合演算などを伴う複雑な分析では実用的な性能を確保することが難しいのに対し、後者では問合せ最適化への対応が可能であることから [1]、結合演算をなどを伴う複雑な分析でも実用的な性能を確保することが可能である。両者ともDBMS へのロードを不要にすることから、大規模データにおけるロード時間の問題を解消可能であるが、データウエアハウスに向けては問合せ最適化への対応の点で後者が好適である。

本研究では、このような背景のもと、実際の業務システムに蓄積されるデータは CSV ファイルなどの表構造ファイルで与えられることが少なくないことから、DBMS として PostgreSQL[2] を対象に、PostgreSQL の次期バージョン 9.1 において導入が予定されている表構造ファイルに対応した外部データラッパ (以下、表構造ファイルラッパと呼ぶ) の参照性能の強化に焦点を

当てる。

SQL/MED では DBMS の外部にあるデータに対するインデクス機能がまだ規定されていないことから、当該表構造ファイルラッパにおける表構造ファイルへのスキャンは、表構造ファイルの先頭から末尾までを順アクセスにより読み出す順スキャンが前提となっている。しかしながら、例えば、大規模な表構造ファイルから特定の条件を満たすデータを検索するような場合、そのような順スキャンでは実用的な性能を確保することが難しい。そのため本研究では、当該表構造ファイルラッパにおいて、表構造ファイルへのインデクススキャンを可能にするための SQL/MED 規格を拡張したインデクス機能ならびにインデクススキャンの実行を最適化可能にするための問合せ最適化機能を導入することで、表構造ファイルラッパの参照性能を向上させ、これにより、大規模な表構造ファイルを対象とする分析においても実用的な性能を確保することを目指す。

以下、本論文の構成は次の通りである。第 2 章において SQL/MED において規定されている外部データラッパならびに PostgreSQL9.1 において導入が予定されている表構造ファイルラッパの概要について述べた上で本研究における課題を述べる。第 3 章において提案する表構造ファイルラッパの実装方式について述べ、第 4 章において著者らが開発したプロトタイプによる評価実験を示し、有効性を議論する。最後に第 5 章において本研究をまとめる。

2. 背景と課題

2.1 SQL/MED における外部データラッパの概要

外部データラッパは、外部データソースの種類や形式毎に、SQL/MED に規定されたインタフェースにしたがう DBMS へのアドオン形式プログラムとして作成される。そして、当該プログラムを DBMS に登録することで、外部データソースに格納されているデータを通常の表と同様に SQL で参照することが可能である。SQL/MED では以下の SQL 構文が規定されている。

- 構文 1: CREATE FOREIGN DATA WRAPPER

参照対象となる外部データソースに対応した外部データラッパを定義したアドオン形式プログラムを DBMS に登録する。

- 構文 2: CREATE SERVER

参照対象となる外部データソースが存在するサーバのホスト情報を DBMS に登録する。

- 構文 3: CREATE USER MAPPING

参照対象となる外部データソースが存在するサーバにおける問合せ実行ユーザのユーザ情報を DBMS に登録する。

- 構文 4: CREATE FOREIGN TABLE

参照対象となる外部データソースが存在するサーバにおける問合せ対象データを外部表として定義する。

また、外部表に対する問合せは以下の手順により処理されることが規定されている (図 1 参照)。

- 手順 1: DBMS と外部データラッパが連動することにより、外部表に対する問合せを、外部データソースに実行させる部分 (以下、外部データソース部分と呼ぶ) と DBMS が実行する部分 (以下、DBMS 部分と呼ぶ) に分解する。

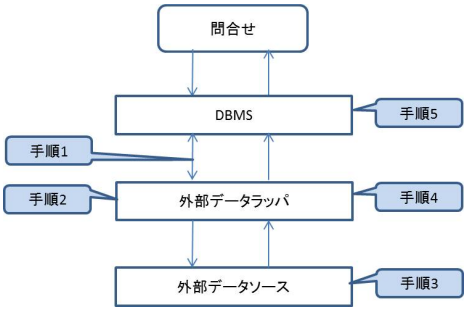


図 1　外部表に対する問合せ処理の流れ

- 手順 2: 外部データラッパが、外部データソース部分を、当該部分を実行する外部データソースの解釈可能な問合せ形式に変換の上、外部データソースに実行を要求する。
- 手順 3: 外部データソースが、外部データソース部分を実行し、結果を外部データラッパに返却する。
- 手順 4: 外部データラッパが、外部データソースから返却された結果を、DBMS の解釈可能なレコード形式に変換の上、DBMS に返却する。
- 手順 5: DBMS が、外部データラッパから返却された結果に基づいて DBMS 部分を実行し、結果をクライアントに返却の上、処理を終了する。

なお、手順 1 における外部データソース部分と DBMS 部分への問合せの分解では、それぞれの部分の問合せ実行計画を最適化する。この問合せ最適化は外部表に対する問合せを効率的に処理する上で極めて重要であることが報告されている [4][7]。

なお、本研究で提案する拡張インデクス機能を有する表構造ファイルラッパでは、表構造ファイルに対する順スキャンもしくはインデクススキャンを表構造ファイルラッパが実行し、結合演算などのスキャン以外のデータベース演算を PostgreSQL が実行することとしている。

2.2 PostgreSQL9.1 における表構造ファイルラッパ

PostgreSQL では、リモートサーバ上の PostgreSQL インスタンスを外部データソースとする外部データラッパ (以下、PostgreSQL ラッパと呼ぶ) の提供を目的として、PostgreSQL8.4 において 2.1 節に示した構文 1 から構文 3 までが実装され、これにより、リモートサーバ上の PostgreSQL インスタンスへ接続するときに必要となるリモートサーバのホスト情報およびユーザ情報をローカルサーバ上の PostgreSQL インスタンスのシステムカタログにおいて管理できるようになった。

また、このような取り組みを契機として、表構造ファイルラッパの提供に向けた検討も開始され、PostgreSQL の次期バージョン 9.1 では構文 1 から構文 4 までの完全な提供とともに、表構造ファイルへの順スキャンを前提とした問合せ処理機能の導入が予定されている。しかしながら、当該表構造ファイルラッパでは表構造ファイルへのインデクススキャンはもとより、問合せ最適化への対応が考慮されておらず、今後の課題となっている。

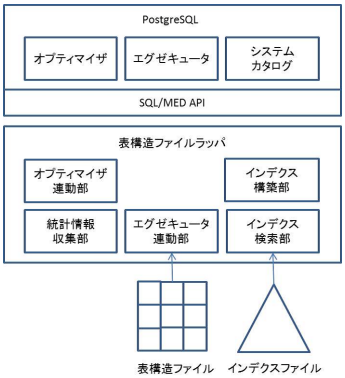


図 2　システム構成

2.3　本研究における課題

本研究では、PostgreSQL の次期バージョン 9.1 において導入が予定されている表構造ファイルラッパにおいて、以下の対応を図ることにより、当該表構造ファイルラッパの参照性能を向上させ、これにより、大規模な表構造ファイルを対象とする分析においても実用的な性能を確保することを目指す。

- 本研究の第 1 の課題: PostgreSQL の次期バージョン 9.1 において導入が予定されている表構造ファイルラッパでは、表構造ファイルへの順スキャンが前提となっていることから、当該表構造ファイルラッパにおいて、表構造ファイルへのインデクススキャンを可能にするためのインデクス機能 (以下、拡張インデクス機能と呼ぶ) を導入する。
- 本研究の第 2 の課題: 拡張インデクス機能を導入した当該表構造ファイルラッパにおいて、インデクススキャンの実行を最適化可能にするとともに、PostgreSQL によって実行される結合演算などの実行方法を最適化可能にするための問合せ最適化機能を導入する。

3.　提案する表構造ファイルラッパ

著者らは、PostgreSQL の次期バージョン 9.1 において導入が予定されている表構造ファイルラッパをベースに、2 章で述べた第 1 の課題 (拡張インデクス機能の導入) および第 2 の課題 (問合せ最適化機能の導入) に対応した表構造ファイルラッパを実現した。図 2 にそのシステム構成を示す。詳細については後述するが、本研究で提案する表構造ファイルラッパのモジュールの機能概要は以下の通りである。

- オプティマイザ連動部: PostgreSQL のオプティマイザと連動し、2 章に示した問合せ処理の手順 1 を実行する。
- エグゼキュータ連動部: PostgreSQL のエグゼキュータと連動し、2 章に示した問合せ処理の手順 2 から手順 5 までを実行する。表構造ファイルへの順スキャンおよびインデクススキャンを実行する。
- 統計情報収集部: 2 章に示した問合せ処理の手順 1 における問合せ最適化で用いられる表構造ファイル・データに関する統計情報を収集して PostgreSQL のシステムカタログに格納する。
- インデクス構築部: 表構造ファイルへのインデクススキャンを実行するためのインデクスを構築する。
- インデクス検索部: 表構造ファイルへのインデクススキャンを実行するとき、インデクスを検索して検索条件を満たす表構造ファイル上のレコード相当データを特定する。

課題 1 に対応した拡張インデクス機能はインデクス構築部・インデクス検索部・エグゼキュータ連動部により構成され、課題 2 に対応した問合せ最適化機能はオプティマイザ連動部および統計情報収集部により構成される。なお、PostgreSQL の次期バージョン 9.1 において導入が予定されている表構造ファイルラッパでは、実質的に上記エグゼキュータ連動部のみが実装される予定である。

3.1　拡張インデクス機能

ここでは、表構造ファイルへのインデクススキャンを可能にするための拡張インデクス機能について詳しく述べる。

3.1.1　インデクスの構築と検索

本研究では、インデクスとして B 木をサポートすることとした。PostgreSQL では通常の表に対する B 木をサポートしているが、この B 木ではレコードへのポインタが、レコードの格納されるブロックのブロック番号とブロック内のオフセット番号により表現されるため、表構造ファイルにそのままでは適用できない。そのため本研究では、表構造ファイル上のレコード相当データの開始オフセットとバイト長をレコードへのポインタとしてもつ、表構造ファイルに向けの B 木を定義し、この B 木を表構造ファイルラッパにおけるインデクス構築部で構築し、表構造ファイルラッパにおけるインデクス検索部で検索することとした。

なお本研究では、表構造ファイルに対しも通常の表と同様に CREATE INDEX 構文を用いてインデクスが定義できるようにした。以下に、CSV 形式の表構造ファイル「/data/csv/UserVisits.dat」に対し、外部表「UserVisits」を定義し、そのカラム「visitDate」に対し、表空間「indexspc」内にインデクス「uservisits_idx」を定義する例を示す。CREATE INDEX 構文における USING 句はインデクス種別を示すためのものであり、ここでは表構造ファイル向けの B 木であることを示すインデクス種別「csv_btree」を指定している。

```
CREATE FOREIGN TABLE UserVisits (
        sourceIP VARCHAR(16),
        destURL VARCHAR(100),
        visitDate DATE,
        adRevenue FLOAT,
        userAgent VARCHAR(64),
        countryCode VARCHAR(3),
        languageCode VARCHAR(6),
        searchWord VARCHAR(32),
        duration INTEGER)
SERVER csv_server
OPTIONS (filename '/data/csv/UserVisits.dat',
        format   'csv',
        delimiter '|',
```

```
        quote '%');
CREATE INDEX uservisits_idx
ON      UserVisits
USING   csv_btree (visitdate)
TABLESPACE indexspc ;
```

3.1.2 表構造ファイルへのインデクススキャン

表構造ファイルへのインデクススキャンの処理の流れは通常の表へのそれに準じるものである。すなわち、PostgreSQL のエグゼキュータがエグゼキュータ連動部に検索条件を満たすレコード 1 件の返却を要求するとエグゼキュータ連動部がインデクス検索部を介して当該検索条件を満たす表構造ファイル上のレコード相当データ 1 件の位置情報を取得し、当該位置情報に基づいて表構造ファイルから対応するレコード相当データをフェッチし、PostgreSQL のレコード形式に変換して PostgreSQL のエグゼキュータに返却する。この一連の処理を逐次的に繰り返すことで、表構造ファイルへのインデクススキャンが実現される。なお通常の表では、更新系トランザクションへの対応から、ロックを用いたインデクスの走査を行うが、表構造ファイルでは更新を考慮しないこととしたことから、そのようなロックを用いずにインデクスを走査して高速化するようにした。

3.2 問合せ最適化機能

ここでは、表構造ファイルへのインデクススキャンの実行を最適化可能にするとともに、PostgreSQL によって実行される結合演算などの実行方法を最適化可能にするための問合せ最適化機能について詳しく述べる。

PostgreSQL ではコスト情報に基づく問合せ最適化 [8] をサポートしている。この問合せ最適化では、問合せを処理可能な問合せ実行計画のそれぞれに対して実行時間に相当する実行コストを推定することで、実行時間がより短い問合せ実行計画を選択することが可能である。問合せ実行計画の実行コストは、問合せ対象となる表に対して施されるすべてのデータベース演算の実行コストを積算することで、問合せ実行計画全体の実行コストが推定される仕組みになっており、この推定のベースになるのが対象表へのスキャンについての以下の実行コストである。

- スタートアップコスト: 対象表へのスキャンを開始するまでに必要な CPU コストと IO コストの合計値
- トータルコスト: 対象表へのスキャンを終了するために必要な CPU コストと IO コストの合計値

例えば、結合演算を伴う問合せ実行計画では、このスタートアップコストおよびトータルコストに基づいて結合演算の実行コストが算出され、問合せ実行計画全体の実行コストに計上される。また、このスタートアップコストおよびトータルコストは対象表へのスキャン方法別に算出され、順スキャンを実行する問合せ実行計画では順スキャンについてのスタートアップコストおよびトータルコストに基づく問合せ実行計画全体の実行コストが、インデクススキャンを実行する問合せ実行計画ではインデクススキャンについてのスタートアップコストおよびトータルコストに基づく問合せ実行計画全体の実行コストがそれぞれ推定される。

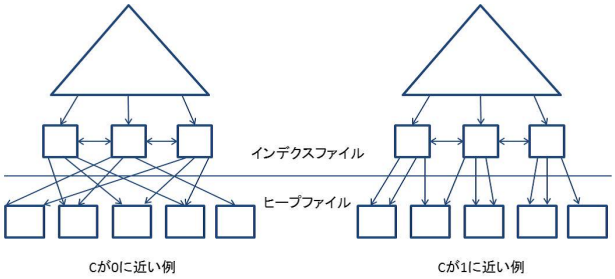


図 3 係数 c の概要

そこで本研究では、表構造ファイルに対しても順スキャン、インデクススキャンそれぞれについてのスタートアップコストおよびトータルコストを算出できるようにすることで、表構造ファイルに対する問合せにおいても前述したコスト情報に基づく問合せ最適化を可能にする仕組みを実現し、これにより、表構造ファイルへのインデクススキャンの実行を最適化可能にするとともに、PostgreSQL によって実行される結合演算などの実行方法を最適化可能にした。

表構造ファイルへの順スキャン、インデクススキャンそれぞれについてのスタートアップコストおよびトータルコストの算出で通常の表と大きく異なる点は、トータルコストにおける IO コストの算出方法である。そこで以下では、通常の表における順スキャン、インデクススキャンそれぞれについての当該コストの算出方法について説明する。

通常の表では、順スキャンについてのトータルコストにおける IO コストは、対象表を先頭から末尾まで順アクセスにより読み出すコストとして次式により算出される。

$$\text{seqscan\_total\_IO\_cost} = T \times \text{seq\_page\_cost} \tag{1}$$

ここで、seq_page_cost はディスクへの順アクセスで 1 ブロックを読み出すコスト、T は対象表のブロック数である。

一方、インデクススキャンについてのトータルコストにおける IO コストは、検索条件を満たすレコードの存在するブロックを対象表よりフェッチするコストとして次式により算出される。

$$\begin{aligned}&\text{idxscan\_total\_IO\_cost} = \\ &\text{max\_IO\_cost} + c^2 \times (\text{min\_IO\_cost} - \text{max\_IO\_cost})\end{aligned} \tag{2}$$

ここで、c は対象表に格納されているレコードがどの程度インデクス順に配置されているかを示す 0 から 1 までの値をとる係数であり、図 3 に示すように、c が 0 に近ければレコードがインデクス順とは無関係に配置されていることを、c が 1 に近ければレコードがインデクス順に近い形で配置されていることをそれぞれ示す。また、min_IO_cost、max_IO_cost はそれぞれ次式で定義される。

$$\begin{aligned}&\text{min\_IO\_cost} = \\ &\text{random\_page\_cost} + (t-1) \times \text{seq\_page\_cost}\end{aligned} \tag{3}$$

$$\text{max\_IO\_cost} = t \times \text{random\_page\_cost} \qquad (4)$$

ここで、random_page_cost はディスクへのランダムアクセスで 1 ブロックを読み出すコスト、t は対象表よりフェッチするブロック数である。min_IO_cost は t 個のフェッチ対象ブロックがディスク上で連続配置されている場合を想定したフェッチコストであり、先頭ブロックについてはランダムアクセスで、後続ブロックについては順アクセスでそれぞれフェッチするコストである。対して max_IO_cost は t 個のフェッチ対象ブロックがディスク上でランダムに配置されている場合を想定したフェッチコストであり、t 個のブロックすべてをランダムアクセスでフェッチするコストである。なお、t は PostgreSQL における共有バッファおよび OS によるキャッシングを考慮した次式 [5] により算出される。

$$t = \begin{cases} \min(2TNs/(2T+Ns), T) \\ \quad \text{when } T \leqq b \\ 2TNs/(2T+Ns) \\ \quad \text{when } T > b \text{ and } Ns \leqq 2Tb/(2T-b) \\ b + (Ns - 2Tb/(2T-b)) \cdot (T-b)/T \\ \quad \text{when } T > b \text{ and } Ns > 2Tb/(2T-b) \end{cases} \qquad (5)$$

ここで、N は対象表におけるレコード数、s は検索条件を満たすレコードの選択率、b はキャッシングのために利用できるバッファブロック数である。

以上において、seq_page_cost、random_page_cost および b は設定ファイルに指定した値が用い、T、N、s および c は対象表におけるブロック数やレコード数あるいはカラム毎のデータヒストグラムなど統計情報を収集する専用コマンド ANALYZE により収集されたデータに基づく値を用いる。

このように、通常の表ではトータルコストにおける IO コストの算出において表のブロック構造を利用しているため、表構造ファイルにそのままでは適用できない。しかしながら、表構造ファイルを格納するファイルシステムがブロック単位の IO を実現していることから、本研究では、表構造ファイルに対しても通常の表を模倣したブロック構造を導入することで、当該 IO コストを算出できるようにした。具体的には、通常の表がデフォルト 8KB を単位とするブロック構造を採用していることから、表構造ファイルを 8KB ブロックで強制的に分割し、この分割数を表構造ファイルのブロック数 T として (1) 式により表構造ファイルへの順スキャンについての当該 IO コストを算出するようにした。また、この T を用いて (5) 式によりフェッチブロック数 t を算出し、この t を用いて (3)(4) 式により min_IO_cost および max_IO_cost を算出することで、(2) 式により表構造ファイルへのインデクススキャンについての当該 IO コストを算出するようにした。ただし、seq_page_cost、random_page_cost および利用可能バッファブロック数 b については表構造ファイル向けに調整可能な値を用いるようにし、表構造ファイルにおけるレコード数 N や検索条件を満たすレコードの選択率 s あるいは表構造ファイルに格納されているレコードの配置特性 c については表構造ファイル向けに拡張した ANALYZE コマンドにより収集した統計情報に基づく値を用いるようにした。

なお、表構造ファイルへの順スキャン、インデクススキャンそれぞれについてのスタートアップコストおよびトータルコストの算出処理については表構造ファイルラッパにおけるオプティマイザ連動部に、表構造ファイル向けに拡張した ANALYZE コマンド相当機能については統計情報収集部にそれぞれ実装した。

表 1　Rankings 表のスキーマ

カラム名	データ型
pageURL	VARCHAR(100)
pageRank	INTEGER
avgDuration	INTEGER

表 2　UserVisits 表のスキーマ

カラム名	データ型
sourceIP	VARCHAR(16)
destURL	VARCHAR(100)
visitDate	DATE
adRevenue	FLOAT
userAgent	VARCHAR(64)
countryCode	VARCHAR(3)
languageCode	VARCHAR(6)
searchWord	VARCHAR(32)
duration	INTEGER

表 3　実験に用いたデータ

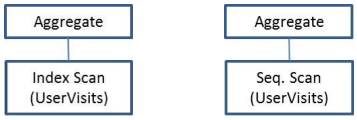
	レコード数	ファイルサイズ (バイト)
Rankings 表	592,759	35,564,548
UserVisits 表	155,000,000	20,128,765,289

4. 評価実験

ここでは、PostgreSQL9.0(開発バージョン) をベースに著者らが開発したプロトタイプを用いて、提案する拡張インデクス機能ならびに問合せ最適化機能について有効性を評価する実験を行った。また、ETL を用いる従来方式との比較実験も行った。実験に用いたサーバの仕様は、 CPU が Intel Xeon 3.00GHz x 2、メモリが 8GB、OS が Linux 2.6.18、ファイルシステムが ext3 である。また、実験に用いたデータは HTTP サーバトラヒックデータ [6] である。このデータは次元表「Rankings」と事実表「UserVisits」に対応した CSV 形式のデータで構成され、レコード数の設定が可能である。表 1 に Rankings 表のスキーマを、表 2 に UserVisits 表のスキーマをそれぞれ示す。また、表 3 に Rankings 表、UserVisits 表それぞれに設定したレコード数などを示す。なお本実験では、インデクスは UserVisits 表のカラム「visitDate」にのみ定義した。

4.1 拡張インデクス機能ならびに問合せ最適化機能に関する評価実験

ここでは、文献 [6] で使用されている問合せを参考に作成した以下の問合せを用いて拡張インデクス機能ならびに問合せ最適化機能について有効性を評価した。



(a) (b)

図 4　問合せ 1 を処理する問合せ実行計画

- 問合せ 1: 検索条件を満たすレコードの選択率に応じて表構造ファイルへの最適なスキャン方法が選択されるかを評価する。なお、DD は 16 から 22 まで変更する。

```
SELECT count(*)
FROM   UserVisits
WHERE  visitdate
BETWEEN Date('2000-01-15') AND Date('2000-01-DD');
```

- 問合せ 2: 問合せ 1 と同様に表構造ファイルへの最適なスキャン方法が選択されるかを評価するとともに、最適な結合方法が選択されるかを評価する。

```
SELECT uv.sourceip,
       avg(r.pagerank) as avgPageRank,
       sum(uv.adrevenue) as totalRevenue
FROM   Rankings r, UserVisits uv
WHERE  r.pageurl = uv.desturl AND
       uv.visitdate BETWEEN
       Date('2000-01-15') AND Date('2000-01-16')
GROUP  BY uv.sourceip;
```

実験では、これらの問合せを実行したとき、問合せを処理可能な問合せ実行計画のそれぞれに対して実際の実行時間に応じた実行コストを推定するかどうか、また、それにより、実行時間が最も短い問合せ実行計画を選択するかどうかを評価することとした。またこの中で、インデクススキャンの動作をあわせて確認することとした。

4.1.1　問合せ 1 に関する結果

問合せ 1 を処理する問合せ実行計画は、図 4 に示すように、UserVisits 表に対して順スキャンを実行する問合せ実行計画およびインデクススキャンを実行する問合せ実行計画の 2 つである。図 5 に検索範囲 DD を 16 から 22 まで変更しながら問合せ 1 を実行したとき、それぞれの問合せ実行計画に対して推定された実行コストを示す。DD が 16 から 19 まではインデクススキャンを実行する問合せ実行計画が、DD が 20 から 22 までは順スキャンを実行する問合せ実行計画が選択される結果となっている。図 6 に選択された問合せ実行計画の実行時間を示す。図 6 では参考のため、選択されなかった問合せ実行計画についても強制的に実行し、その実行時間を示している。

これらの結果から、順スキャン、インデクススキャンいずれを実行する問合せ実行計画に対しても実際の実行時間に応じた実行コストが推定され、その結果、実行時間が最も短い問合せ実行計画が選択されていることが分かる。なお、インデクススキャンについてはインデクスを実行する問合せ実行計画において問題なく動作することが確認された。

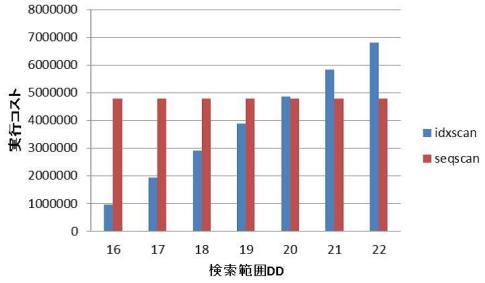


図 5　問合せ 1 を処理する問合せ実行計画に対する実行コスト

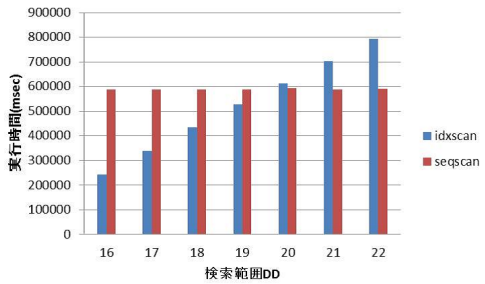


図 6　問合せ 1 を処理する問合せ実行計画に対する実行時間

4.1.2　問合せ 2 に関する結果

PostgreSQL では結合演算の結合方法としてネステッドループ結合、ソートマージ結合およびハッシュ結合をサポートしていることから、問合せ 2 を処理する問合せ実行計画は、図 7 に示すように、Rankings 表と UserVisits 表の結合演算に対する 3 通りの結合方法と UserVisits 表への 2 通りのスキャン方法の 6 つの組合せに分類される。(注1)。表 4 にそれぞれの問合せ実行計画に対して推定された実行コストおよび実際の実行時間を示す。選択された問合せ実行計画は図 7(c) に示すものである。

これら結果から、いずれの問合せ実行計画に対しても実際の実行時間にほぼ応じた実行コストが推定され、その結果、実行時間が最も短い問合せ実行計画が選択されていることが分かる。また、UserVisits 表に対して順スキャンを実行する問合せ実行計画よりもインデクススキャンを実行する問合せ実行計画のほうが実行時間が短い結果となっているが、同じインデクススキャンを用いる問合せ実行計画の中でも最も実行時間が短い結合方法が選択されており、UserVisits 表へのスキャン方法とともに結合方法が同時に最適化されていることが分かる。なお、インデクススキャンについてはここでも問題なく動作することが確認された。

4.2　従来方式との比較実験

ここでは、従来のように Rankings 表および UserVisits 表に対応する CSV 形式のデータを PostgreSQL にロードし、UserVisits 表にインデクスを構築するまでの所要時間と、本研究で提案する表構造ファイルラッパによりこれらの表を外部表

(注1)：3 通りの結合方法いずれも内表・外表のとり方に自由度があることなどから、結合方法については実際にはより多くの結合方法が考慮されるが、ここでは紙面の都合、6 つの組合せそれぞれにおいて実行コストが最も小さく推定されたもののみを取り上げて説明する。

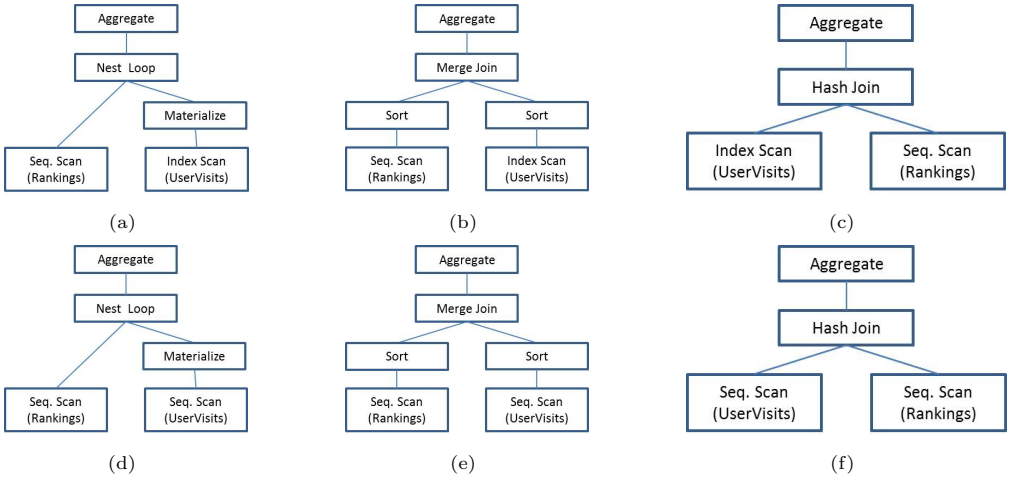


図 7　問合せ 2 を処理する問合せ実行計画

表 4　問合せ 2 を処理する問合せ実行計画に対する実行コストと実行時間

	スキャン方法	結合方法	実行コスト	実行時間 (msec)
(a)	インデクススキャン	ネステッドループ結合	109257030.36	4376993.360
(b)	インデクススキャン	ソートマージ結合	1047414.21	243093.776
(c)	インデクススキャン	ハッシュ結合	994303.47	238885.034
(d)	順スキャン	ネステッドループ結合	113062914.16	4710356.876
(e)	順スキャン	ソートマージ結合	4853298.00	591900.567
(f)	順スキャン	ハッシュ結合	4800187.26	589124.194

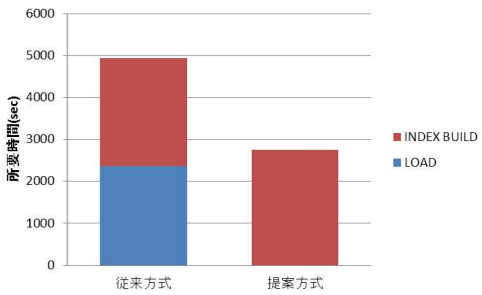


図 8　インデクス構築までの所要時間に関する比較

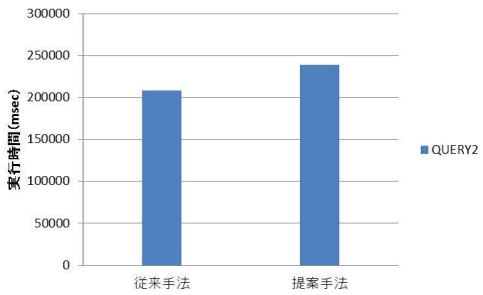


図 9　問合せ 2 の実行時間に関する比較

として定義して UserVisits 表にインデクスを構築するまでの所要時間を比較することで、提案方式の有効性を評価した。図 8 に結果を示す。提案方式では PostgreSQL へのロードが不要であることに加え、インデクス構築も従来方式とほぼ同様の時間で終了しており、所要時間を大きく短縮していることが分かる。

また、前節の問合せ 2 を用いて従来方式と提案方式の問合せ処理性能を比較した。図 9 に結果を示す。提案方式では従来方式に比したオーバヘッドが約 14%となっており、実用的な性能を達成していると考えられる。本実験は一例ではあるものの、これらの結果から、提案手法の有効性は高いものと期待される。

5. おわりに

本研究では、PostgreSQL の次期バージョン 9.1 において導入が予定されている表構造ファイルラッパにおいて、大規模な表構造ファイルを対象とした分析においても実用的な性能を確保する立場から、表構造ファイルへのインデクススキャンを可能にするための拡張インデクス機能ならびにインデクススキャンの実行を最適化可能にするための問合せ最適化機能を導入することで、表構造ファイルラッパの参照性能を向上させる実装方式について述べ、著者らが開発したプロトタイプによる評価実験によりその有効性を示した。

今後の課題としては、PostgreSQL では通常の表への順スキャン、インデクススキャンに加え、複雑な検索条件をもつ問合せを効率的に処理することが可能なビットマップスキャンを実現していることから、表構造ファイルに対してもビットマップスキャンを可能にすることが挙げられる。また、提案した拡張インデクス機能ならびに問合せ最適化機能の PostgreSQL ラッパへの拡張についても今後検討してゆきたい。

文　　献

[1] ISO/IEC 9075-9:2008, Information technology – Database languages – SQL – Part 9: Management of External Data (SQL/MED), International Organization for Standardization, 2008.

[2] PostgreSQL, http://www.postgresql.org/.

[3] J. Cohen, B. Dolan, M. Dunlap, J.M. Hellerstein, and C. Welton, “MAD skills: new analysis practices for big data,” In Proc. of VLDB, pp. 1481–1492, 2009.

[4] V. Josifovski, P. Schwarz, L. Haas, and E. Lin, “Garlic: A New Flavor of Federated Query Processing for DB2,” In Proc. of SIGMOD, pp. 524–532, 2002.

[5] Lothar F. Mackert , and Guy M. Lohman, “Index scans using a finite LRU buffer: a validated I/O model,” ACM Trans. on Database Systems, v.14 n.3, p. 401–424, 1989

[6] A. Pavlo, E. Paulson, A. Rasin, D. J. Abadi, D. J. DeWitt, S. Madden, and M. Stonebraker, “A comparison of approaches to large-scale data analysis,” In Proc. of SIGMOD, pp. 165–178, 2009.

[7] M. T. Roth, F. Ozcan, L. M. Haas, “Cost Models DO Matter: Providing Cost Information for Diverse Data Sources in a Federated System,” In Proc. of VLDB, pp. 599–610, 1999.

[8] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price, “Access path selection in a relational database management system,” In Proc. of SIGMOD, pp. 23–34, 1979.