関係データベースシステムにおける時系列イベント分析処理ベンチマークの検討
A Study on Performance Benchmark for Time-series Event Analytics on a Relational Database System

川道 亮治 † **早水 悠登** † **合田 和生** † **喜連川 優** †‡
Ryoji Kawamichi Yuto Hayamizu Kazuo Goda Masaru Kitsuregawa

1. はじめに

関係データベースシステムは今日の社会において基幹的な IT 基盤として広く普及しており，様々な経済活動における取引の履歴や，ヘルスデータ，自然環境の観測データ，あるいは個々人のオンライン上の情報発信に至るまで，時々刻々と生成されるデータの蓄積・管理を担っている．多くの場合，これらのデータベース応用の扱うデータは陰に陽に時刻の情報が付加されて記録され，その時系列的な変動に着目したデータ分析のアプローチは，例えば時間的に連続するデータ系列のパターンマイニング [1] などを始めとして，様々な場面で利用が広がっている．大規模な Web サービスのモニタリングログや大量のセンサデータなど，時系列データの生成そのものが著しく高速な場合には，時系列データを取り扱う専用の処理基盤を構築する例も近年では見られるが [25, 2]，システム運用の簡便性や SQL による問合せの柔軟性といった観点から，既にデータが蓄積されている関係データベースシステムを時系列データ分析基盤として利用することが合理的である場合も多く，時系列データ分析を指向した関係データベース拡張なども複数見られる [32, 34, 23].

時系列データの分析に対するアプローチとして，パターンマイニングに代表されるデータ全体から傾向を分析する手法は 2000 年代に一定程度の成熟を見せている一方で，時系列データの個々のデータの変化に着目した分析のアプローチに関しては未だ十分な取り組みが見られない．所謂ビッグデータや IoT ブームに牽引され，今後ますますデータによって高い分解能で実社会の現象を捉えることが可能になってゆくものと期待され，データ分析においても個々の事象に着目した分析が重要性を増してゆくものと著者らは考えている．例えば購買履歴データであれば特定の顧客や特定の商品といった軸，或いはヘルスケアデータであれば特定の疾病などの軸に着目して，時系列上に点在するイベント列を関連付けた形で取得することで，集約処理や全体の傾向分析のみからは捉えられない特徴的な事象の捕捉や，あるいは個々のユーザの時系列的な挙動の類似性に着目したデータ分析などが応用として考えられる．本論文では，時系列データにおける分析対象の個々のイベントにフォーカスして，時系列的に関連するイベントを抽出する問合せ処理を**時系列イベント分析処理**と称し，その関係データベースシステムにおける問合せ処理の観点から議論したい．

時系列イベント分析処理は，特定のイベント群を抽出した後に，それに基いて時系列的に関連するイベント群を抽出するという処理の連鎖によって構成されるものとし，典型的には時系列データの格納されるテーブルが自己結合を繰り返す形の問合せとして表現できる．関係データベースにおいて，結合処理の最適化は実行効率に最も大きな影響を与える要素の一つであるが，結合回数が多く，また結合対象同士の相関関係が高いほど効率的な実行は容易でなく，その処理性能はデータベースシステムの実装によって大きく異なることが知られている．しかしながら，時系列イベント分析処理やそれに類するワークロードの処理性能はこれまで十分に分析がなされておらず，定量的に処理性能を評価する指標に関して十分に議論されてきたとはいえない状況にある．

本論文では，関係データベースシステムにおける時系列イベント分析処理について問合せの基本構造を規定し，問合せ処理の観点からその特性を議論する．そして，関係データベースにおける時系列イベント分析処理の処理性能を定量的に議論するための指標として，TPC-H ベンチマーク [9] データセットに対して時系列イベント分析処理を行う問合せを規定した *h*ESPR(TPC-**H** based **E**event **S**equence **P**attern **R**etrieval) ベンチマークを提案する．*h*ESPR ベンチマークの処理性能指標としての有効性を評価するため，オープンソース関係データベースシステムである PostgreSQL と MySQL を用いた評価実験として，PostgreSQL のバージョン間での処理性能比較，ならびに PostgreSQL と MySQL の処理性能比較を行い，各実装の特徴を分析するとともに，時系列イベント分析処理の処理性能に影響を与えうる要素に関して考察を行った．

本論文の構成は以下の通りである．まず，2. 節において関係データベースシステムにおける時系列イベント分析処理の問合せの形式的定義と，問合せ処理層における当該問合せの特性を分析し，3. 節において時系列イベント分析処理の処理性能指標として提案する *h*ESPR ベンチマークの基本設計を示す．4. 節にて *h*ESPR を用いた評価実験の結果について述べ，5. 節にて関連研究と本論文の位置づけを説明した後，6. 節にて本論文をまとめる．

2. 関係データベースシステムにおける時系列イベント分析処理

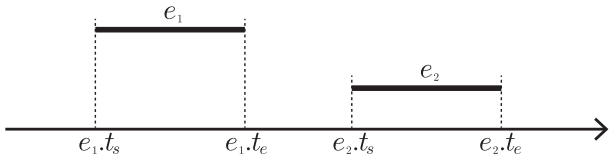
2.1. 時系列イベント分析処理の問合せの形式的定義

関係データベースシステムにおいては，データは関係モデルに規定されるタプル，およびリレーションによって表現されるため，本論文の対象とする時系列イベント分析処理の問合せについても，関係モデルおよび関係代数によって記述される問合せを対象とする．

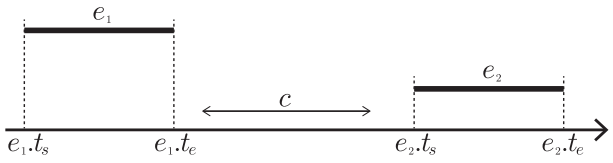
以降の議論では，時刻を表す属性値として t_s, t_e を有するタプル e を指して**時系列イベント**と称することとする．属性値 t_s, t_e は同一のドメインに属する値であることとし，当該イベントの発生しているとみなされる時刻の範囲 $[t_s, t_e]$ を表すものとする．

ここで，2 つの時系列イベント e_1, e_2 に対して時系列関係 $\mathcal{T}$ を考える．$\mathcal{T}$ は時系列イベント e_1, e_2 の属性値 t_s, t_e に関する条件式から構成される命題式であるものとし，それが成り立つことを $e_1 \mathcal{T} e_2$ と表記する．$\mathcal{T}$ の例としては，時系列イベント e_1 の発生の後に時系列イベント e_2 が発生する (図 1(a))，時系列イベント e_1 の発生から一定の時間 c 以上の間隔を空けてか

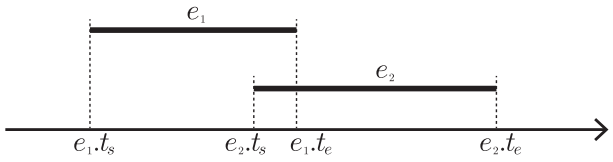
† 東京大学生産技術研究所
‡ 国立情報学研究所



(a) $\mathcal{T} = e_1.t_e < e_2.t_s$



(b) $\mathcal{T} = e_1.t_e + c < e_2.t_s$



(c) $\mathcal{T} = e_1.t_s < e_2.t_s \wedge e_2.t_s < e_1.t_e$

図 1　時系列関係 $\mathcal{T}$ の例

ら時系列イベント e_2 が生じる (図 1(b))，時系列イベント e_1 が先に発生しかつ発生している時刻の範囲が e_2 とオーバラップする (図 1(c)) などが考えられる．

時系列イベント分析処理において問合せの最も基本的なケースが，一つの時系列関係 $e_1\mathcal{T}e_2$ が成り立つ全ての $\langle e_1, e_2\rangle$ の対を取得する問合せである．時系列イベント e_1, e_2 がそれぞれリレーション E_1, E_2 に属し，時刻に関わらない選択条件 $P_1(e_1), P_2(e_2)$，および結合条件を $J_{1,2}(e_1, e_2)$ とすると，この問合せは $\langle \mathcal{T}, E_1, E_2, P_1, P_2, J_{1,2}\rangle$ の組によって規定され，また関係代数として下記のような問合せに展開される．

$$\sigma(E_1)_{P_1} \bowtie_{(J_{1,2} \wedge e_1\mathcal{T}e_2))} \sigma(E_2)_{P_2} \qquad (1)$$

複数の時系列関係が成り立つ場合の問合せに於いても，基本的には上記の二項の時系列関係 $e_1\mathcal{T}e_2$ の組み合わせへと還元し，その組み合わせによって問合せを表現可能である．以下，時系列イベント分析問合せとして代表的な例を示す．

■複数の時系列イベントが直列に時系列関係を持つケース

$$e_1\mathcal{T}_{(1,2)}e_2 \wedge e_2\mathcal{T}_{(2,3)}e_3 \wedge \cdots \wedge e_{n-1}\mathcal{T}_{(n-1,n)}e_n$$

上記のような時系列関係が成り立つ全ての時系列イベントの対 $\langle e_1, e_2, \cdots, e_n\rangle$ を取得する問合せは，式 (1) をそれぞれの時系列関係に適用することで，下記のような関係代数の問合せに展開できる．

$$\sigma(E_1)_{P_1} \bowtie_{J_{(1,2)} \wedge e_1\mathcal{T}_{(1,2)}e_2} \sigma(E_2)_{P_2} \bowtie_{J_{(2,3)} \wedge e_2\mathcal{T}_{(2,3)}e_3} \cdots \bowtie_{J_{(n-1,n)} \wedge e_{n-1}\mathcal{T}_{(n-1,n)}e_n} \sigma(E_n)_{P_n}$$

■起点となる 1 つの時系列イベントが並列に時系列関係を持つケース

$$e_1\mathcal{T}_{(1,2)}e_2 \wedge e_1\mathcal{T}_{(1,3)}e_3 \wedge \cdots \wedge e_1\mathcal{T}_{(1,n)}e_n$$

上記のような時系列関係が成り立つ全ての時系列イベントの対 $\langle e_1, e_2, \cdots, e_n\rangle$ を取得する問合せは，それぞれの $e_1\mathcal{T}_{(1,k)}e_k$ に対して式 (1) を適用することで，下記のような関係代数の問合せに展開できる．

$$\begin{aligned}\sigma(E_1)_{P_1} &\bowtie_{J_{(1,2)} \wedge e_1\mathcal{T}_{(1,2)}e_2} \sigma(E_2)_{P_2} \\ &\bowtie_{J_{(1,3)} \wedge e_1\mathcal{T}_{(1,3)}e_3} \sigma(E_3)_{P_3} \\ &\cdots \\ &\bowtie_{J_{(1,n)} \wedge e_1\mathcal{T}_{(1,n)}e_n} \sigma(E_n)_{P_n}\end{aligned}$$

ここまでの議論においては，一般性を失わないために時系列関係 $e_i\mathcal{T}e_j$ を満たす時系列イベントの対 $\langle e_i, e_j\rangle$ を取得する問合せは，異なるリレーション E_i, E_j 同士の結合としてきたが，もちろん $E_1 = E_2$ であってもよい．またリレーション E_i はデータベースのスキーマ上に存在するリレーションであってもよいし，ビュー等問合せの結果によって生成されるリレーションであってもよい．

2.2. 関係データベースにおける時系列イベント分析処理の問合せ処理

実際のデータベースシステムにおいては，スタースキーマにおけるファクト表や，オンラインサービスのアクセスログなど，単一または少数のリレーションに対して時系列イベントが蓄積されるケースが想定されるため，前節で述べた時系列イベント分析処理の問合せは，これらリレーションに対する自己結合演算の繰り返しを中心に構成されるものとなる．各リレーション間の結合条件には時系列関係 $\mathcal{T}$ が含まれるため，$\mathcal{T}$ が強い制約であるほど，リレーション間の相関性は高くなる．

結合演算は関係データベースにおいて最も負荷の高い処理の一つであり，関係データベースシステムは多くの場合ネステッドループ結合，ハッシュ結合，ソートマージ結合等，複数の結合アルゴリズムを備え，結合対象のデータ量やデータ分布，結合のカーディナリティ[§]等の統計情報を基に，問合せ最適化器が好適する結合アルゴリズムを選択することで，問合せ処理性能の向上を図る．そのため，結合演算に関する統計情報の見積り，特に結合のカーディナリティの見積りの精度が問合せ処理性能に大きく影響を与えることが知られている [22, 20]．

結合のカーディナリティ見積りにおいて，大きな誤差を生じうる要因として，(1) 多数の結合演算による誤差の増幅，(2) 結合条件における属性値間の相関性，の 2 点が挙げられる．(1) に関しては，結合演算の性質上，結合ごとにカーディナリティは乗じられて増幅されてゆくため，例えば 5 つのリレーションの結合で，各結合におけるカーディナリティ見積り誤差が 20% の場合には，最終的に誤差は 207% まで増幅される．また (2) に関しては，複数の属性値間のデータ分布に基いて，相関性の強い結合におけるカーディナリティ推定の高精度化に関する取り組みも多数見られるものの [6]，いまだその有効性は限定的であり，一般的にはリレーション同士の相関性が強いほど結合カーディナリティ推定の精度は悪くなる傾向がみられる．

以上の議論より，多数の自己結合演算から構成され，各結合演算におけるリレーション間の相関性が高い傾向にある時系列イベント分析の問合せは，関係データベースシステムにおいて効率的に実行することが容易ではないといえる．つまり，時系列イベント分析処理の性能を定量的に評価可能なベンチマークを設計することで，関係データベースシステムが効率的に実行

[§] 結合後に出力されるタプル数

することが本質的に容易でない問合せにフォーカスして，各実装の問合せ処理性能を評価することができる．

また，これら問合せ実行前に正確なカーディナリティ推定が困難である問合せ実行の効率化に関しては，従前の問合せ実行前に最適な問い合わせ実行計画を選択するという枠組み [7] を超えて，動的な問合せ最適化 [36, 21] や，実行時のフィードバックを用いたカーディナリティ推定の精度向上 [28, 13, 17, 22] といった様々なアプローチが提案されていることから，時系列イベント分析処理ベンチマークにより，このような先進的な手法の有効性を評価することも可能となることが期待される．ただし，本稿の目的はベンチマークそのものの妥当性を検討することであり，これらの手法の有効性の評価については別稿に譲ることとしたい．

3. *h*ESPR ベンチマーク

本節では，関係データベースシステムにおける時系列イベント分析処理の処理性能を定量的に評価するための指標として著者らが提案する *h*ESPR ベンチマークの基本設計について説明する．

*h*ESPR ベンチマークは，関係データベースシステムに蓄積される時系列データを分析対象とするため，データセットとしては関係データベースシステムの典型的なワークロードとして広く用いられている TPC-H ベンチマーク [9] のデータセットを採用する．TPC-H データセットは卸売業者の帳票データを模した構造となっており，注文履歴や注文明細など，時系列データとしての特性をもつデータを含むため，*h*ESPR ベンチマークの目的に好適である．TPC-H データセットはデータベース分野において幅広く評価用データセットとして用いられており，またその特性も良く解析されており [4]，データの入手も容易であるため，TPC-H データセットを用いることは *h*ESPR ベンチマークの追試可能性に大きく寄与するものと考える．

*h*ESPR ベンチマークにおいて，関係データベースシステムの性能評価に用いる問合せとして，複数のビジネス上の問合せを想定し，表 1 に示す 7 種類の問合せを設計した．それぞれの問合せの SQL 文は末尾の 付録 A に記載した．

時系列イベント分析処理のワークロードに対する評価可能な対象範囲を広げるため，2.1 節で述べた時系列関係 $\mathcal{T}$，および時系列イベントのリレーションの種類 (スキーマに存在するリレーション，結合により生成されるリレーション，集約演算により生成されるリレーション) が異なる組み合わせとなるよう考慮した上で，各問合せの設計を行った．

表 1　*h*ESPR ベンチマークの問合せ一覧

Q.1	ある 1 か月間の注文それぞれに対し，その注文日以降 1 ヶ月間の，同じ顧客による注文履歴一覧を取得する
Q.2	特定の顧客群について，ある 3 ヶ月間のそれぞれの注文明細のうち，その注文日から 3 ヶ月以内にその注文商品が 100 個以上売れたものの一覧を取得する．
Q.3	ある 6 ヶ月間内にブランド”Brand#11” の商品を購入したそれぞれの注文のうち，その注文日から 6 か月以内に同じ顧客がブランド”Brand#21” の商品を購入し，その後，最初の注文日から 12 か月以内に同じ顧客がブランド”Brand#31” の商品を購入したもの一覧を取得する
Q.4	ある 1 ヶ月間内のそれぞれの注文明細に対し，その後同じ顧客が同じ商品を購入した最初の 5 つの注文を取得する
Q.5	ある 6 ヶ月間に購入されたすべての商品に対し，その 6 ヶ月間の前の 6 ヶ月間に同じ商品を購入した注文の一覧を取得する
Q.6	特定の顧客群について，ある 1 ヶ月間の注文の数と同数の注文を，その期間の直後から順に取得し，注文の一覧を取得する
Q.7	ある年度で，年初からの売り上げの累計が 100 万ドルを超えた日を取得する

4. 評価実験

*h*ESPR ベンチマークの性能指標としての有効性を評価するため，オープンソース関係データベースである PostgreSQL ならびに MySQL を用いて評価実験を行った．評価においては，*h*ESPR ベンチマークの各クエリの実行時間を処理性能の指標として，PostgreSQL のバージョン間での処理性能の比較，および PostgreSQL と MySQL の処理性能の比較を行った．以下にその詳細を説明する．

4.1. 実験環境と設定

実験環境の諸元を表 2 に示す．

当該環境において，下記に示すオープンソース関係データベースシステムの実装をそれぞれ用いて，*h*ESPR ベンチマークの問合せ実行時間を計測し，それぞれの処理性能を比較した．

表 2　実験環境諸元

Dell PowerEdge R720xd server	
Processor	Intel Xeon E5-2680 (2 sockets, 16 cores)
Memory	64GB (x8 8GB 1,600MHz DDR3 DIMM)
Storage	ext4 on RAID6(22D+2P) x24 NL-SAS 10Krpm HDDs
OS	CentOS 7.3.1611 (Linux 3.10.0)

- PostgreSQL 7.4
- PostgreSQL 8.4
- PostgreSQL 9.2
- PostgreSQL 9.6
- PostgreSQL 10.0dev (実験実施時点における最新の開発版)
- MySQL 5.7

実験に先立ち，それぞれの関係データベースシステムの実装毎に，TPC-H ベンチマークのデータセットを scale factor = 10 で生成，ロードしたデータベースを作成した．

問合せ実行時間の測定に際しては，ext4 ファイルシステムの物理的なファイル割当位置による入出力パフォーマンスへの影響を最小化するため，表 2 に示す RAID6 ボリュームの先頭に 100GB の測定用データベース格納領域を作成し，測定開始直前に当該領域にファイルシステムを新規作成し，測定対象とするデータベースのファイルを移動した上で，実験を行った．

また，PostgreSQL の各バージョンについて，問合せ最適化

におけるコスト見積り誤差を可能な限り最小化するため，[14] に示されるテスト用問合せの実行時間計測に基づくコスト変数の較正を事前に行った．

4.2.PostgreSQL の各バージョンの処理性能比較

*h*ESPR ベンチマークの各問合せについて，PostgreSQL の各バージョンで問合せ実行時間を測定した結果を図 2 に示す．

一般に，関係データベースシステムの開発が進みバージョンが上がると，問合せ処理性能も向上することが期待される．図 2 の結果から，測定結果全体の傾向としては，バージョンが上がるににつれて実行時間が短くなる傾向が見られ，特に Q.1，Q.3，Q.4 に関しては PostgreSQL 7.4 から PostgreSQL 8.4 にかけての性能の向上が顕著である．

一方で，バージョンが上がることで性能が低下するケースも散見される．最も顕著な例は，Q.6 における PostgreSQL 8.4 から PostgreSQL 9.2 にかけての 96 倍の実行時間増加である．これは，Q.6 に含まれる相関副問合せに要するコストの見積り方式が変更されたことに従い，副問合せ内で本来であれば走査しないほうが実行時間を抑えられる索引を追加で走査していることで，大幅に入出力処理およびタプルに対する演算処理の負荷が増加したことに起因するものである．PostgreSQL 9.2 以降は，バージョンが上がるごとに実行時間は短くなる傾向にあり，一定の改善は見られるものの，依然として PostgreSQL 7.4 や PostgreSQL 8.4 と比べて実行時間が極めて長い．同様の理由により，Q.7 においても PostgreSQL 8.4 から PostgreSQL 9.2 にかけて実行時間が約 7 倍増加している．これらの結果から，時系列イベント分析のように問合せ最適化が本質的に容易でない問合せにおいては，ある種の変更が著しい性能低下を招きうることがわかる．

また，Q.2 においては最新の開発版 PostgreSQL 10.0dev において 1.6 倍の処理性能の低下が見られる．PostgreSQL 9.6 においては問合せ実行計画の一部にハッシュ結合が用いられていたが，これが PostgreSQL 10.0dev ではネステッドループ結合に変化しており，Q.2 においては結果的に実行時間の増加につながっている．

大幅な変化ではないものの，Q.3，Q.4 および Q.5 においては, PostgreSQL 8.4 から PostgreSQL 9.2 にかけて 5〜13% 程度実行時間が増加しており，また Q.4 では PostgreSQL 9.6 から PostgreSQL 10.0dev にかけて 18% 実行時間が増加している．Q.4 においては，PostgreSQL 8.4 と PostgreSQL 9.2 では同一の問合せ実行プランを使用しているため，問合せ実行器における性能低下が生じているものと思われる．それ以外の実行時間増加は，いずれの場合も，結合順序の入れ替わりなど，問合せ実行計画の一部が変化したことによるものである．

以上の結果より，*h*ESPR ベンチマークを用いた問合せ処理性能の PostgreSQL バージョン間比較により，バージョンアップにより処理性能は全体的に改善傾向ではあるものの，問合せ最適化器の変更による最大で 96 倍の顕著なものを含め，複数の性能低下が生じていることが明らかとなった．つまり，*h*ESPR ベンチマークによって，従前のベンチマークでは捉えることの出来なかった性能上の問題点を新たに検出することが可能となり，これは時系列イベント分析処理を用いたベンチマークの有効性を示す結果といえる．

4.3.PostgreSQL と MySQL の処理性能比較

異なる関係データベースシステム同士の性能比較として, それぞれ最新の安定版リリースである PostgreSQL 9.6 と MySQL 5.7 において *h*ESPR ベンチマークを実行し，処理性能の比較を行った．それぞれの問合せ実行時間を図 3 に示す．図 3 においては，実行時間が対数尺度でプロットされていることに注意されたい．

図 3 からわかるように，*h*ESPR ベンチマーク 7 種類の問合せのうち 5 種類において PostgreSQL 9.6 が MySQL 5.7 よりも実行時間が短く，特に Q.1，Q.3，Q.4 においては 47 倍以上の実行時間の差が見られる．これは，MySQL 5.7 の実装する結合アルゴリズムがネステッドループ結合のみであることに起因する．PostgreSQL 9.6 においては，Q.1，Q.3，Q.4 それぞれの問合せ実行計画の一部において多量のレコードを結合する処理に関してハッシュ結合を用いている一方で，当該処理に関して MySQL 5.7 ではネステッドループ結合と索引走査により都度テーブル空間に対してランダムな入出力を発行せざるを得ないため，入出力に要する時間が PostgreSQL 9.6 と比して極めて長く，実行時間の大幅な違いに繋がった．

一方で，Q.6，Q.7 については PostgreSQL 9.6 に比して MySQL 5.7 はそれぞれ 63 倍，7 倍実行時間が短いという結果が得られいる．これは，前節の実験結果に示されるように PostgreSQL 9.6 が比較的非効率な索引走査を問合せ実行計画として選択した一方で，MySQL 5.7 では PostgreSQL 7.4 や 8.4 と同様の索引を用いた索引走査を行っているため，結果として PostgreSQL 9.6 と比して大幅に短い実行時間を達成している．

一般論として，MySQL 5.7 は結合アルゴリズムとしてネステッドループ結合のみを実装しており，複数の結合アルゴリズムを実装する関係データベースシステムと比べて，データ分析を行うワークロードにおいては処理性能が低い傾向にあると言われており，上記の実験結果の傾向とも合致する．一方で，時系列イベント分析処理のように問合せ最適化が本質的に容易でない場合，Q.6 の PostgreSQL 9.6 の結果に示されるように結果として極めて効率の悪い問合せ実行計画が選択されてしまうケースも存在するため，問合せ実行器において優れた結合アルゴリズムを実装するだけではなく，その適切な活用のための枠組みが重要であることが改めて確認された．

5. 関連研究

時系列データに対する分析のアプローチとして，一連のデータ系列をある種の波形と見なし，定期的に出現するデータ系列を検出するパターンマイニングに関して数多くの研究が行われている．パターンマイニングそのものの技術は 1980 年代より人工知能分野において [10] などの取り組みが見られ，過去のデータ系列から未来のデータ系列生成が決定される場合に，その生成規則を推定することを目的としたものであったが，そのデータベースシステムへの応用として Agrawal らは [1] においてトランザクション履歴から特徴的な繰り返しとして出現するパターンのみを抽出するアルゴリズムを複数提案した．また Han らはその拡張として，[15] において繰り返しが部分的に欠損する場合でもパターンを検出可能なアルゴリズムを提案した．その他にも，アルゴリズムの改善 [16] やビットマップを利用することで長いパターンマイニングの効率化を図るアプローチ [3]，応用に特化したパターンマイニングアルゴリズムの深化 [33] など，多様な取り組みが見られる．本論文の対象とする時系列イベント分析は，特定の分析軸によって絞り込まれたデータに対し，時間軸上での前後関係をトレースする分析処理であり，パターンマイニング技術とは想定する用途が異なるが，分析軸の候補とする情報を得るための相補的な技術としてパター

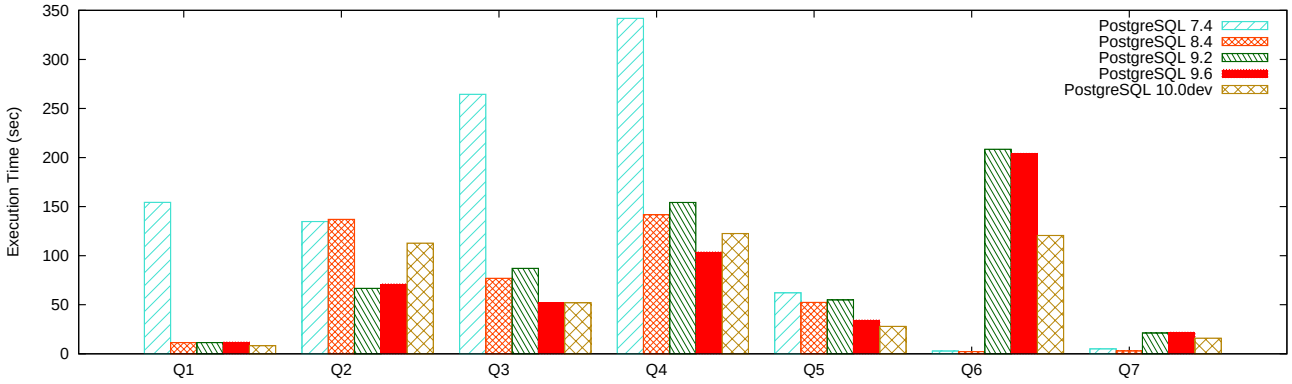


図 2　PostgreSQL の各バージョンの問合せ実行時間

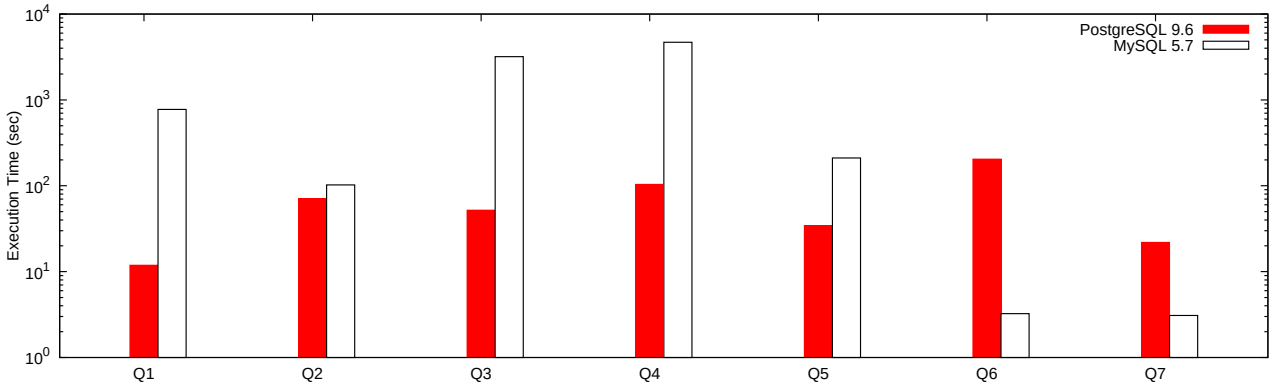


図 3　PostgreSQL 9.6 と MySQL 5.7 の問合せ実行時間

ンマイニングは活用可能である．

時系列データからのデータ検索という観点では，与えられたデータ系列に類似するデータ系列を検索する類似パターン検索に関しても多数の研究が行われている．類似パターン検索における先駆的な取り組みとして，Faloutsos らは [11] において部分的なデータ系列を高次元の特徴空間上の点にマッピングし，R*-tree 索引を用いるアプローチを示した．Jagadish らは [35] において，特徴空間における距離指標をユークリッド距離ではなく Dynamic Time Warping (DTW) を用いることでよりロバストな索引付けが可能であることを示し，Koegh らは [18] において DTW を用いた効率的な完全一致検索の手法を提案した．またフーリエ変換による特徴空間の圧縮 [5] や，その拡張と一般化 [26] などの研究も見られる．基本的に類似パターン検索の対象は連続的なデータ系列のパターンに基づく検索である一方で，本論文で議論する時系列イベント分析処理はデータの問合せ対象が連続的なデータ系列には限られないため，目的を異にする技術である．

データベースシステムにおける時系列データの取り扱いをより効率化する，あるいはより簡便にするためにデータモデルや問合せ記述言語を拡張する取り組みもみられる．Seshadri らは [31] においてシーケンスデータを対象として関係モデルを拡張した SEQ なるデータモデルを提案し，また SEQ における問合せ最適化技法を [30] において示している．SQL のシーケンスデータや時系列データ応用のために拡張する提案としては，Ramakrsihnan らによる SRQL[27]，Sadri らによる SQL-TS[29]，Snodgrass による TSQL2[32]，Law らによるストリームデータを指向した拡張 [19] などが提案されている．本論文では，標準的な関係データモデルおよび SQL をにおける時系列データ処理の性能について議論しており，これらの研究とは前提が異なる．

データベースシステムのベンチマークに関しては，Transaction Processing Council によって定められた各応用向けのベンチマークが幅広く利用されており，特にデータ分析に関するベンチマークとして [9] がデータベース製品の性能指標として用いられている他，典型的な分析処理のワークロードとして数多くの研究において用いられており，その特性も良く分析されている [4]．そのほかにも，データウェアハウスにおいて一般的に用いられるスタースキーマを対象とする Star Schema Benchmark [24] や，MapReduce 等のクラウド向けワークロードのベンチマークである YCSB[8]，TPC-DS を半構造データ処理向けに拡張した BigBench[12] などが提案されている．一方で，時系列データの分析ワークロードを想定したベンチマークは我々の知る範囲において提案されておらず，本論文において提案する *h*ESPR ベンチマークは新規な取り組みである．

6. 結論

本論文では，関係データベースシステムにおけるデータ分析として，これまで十分に特性が明らかにされていなかった時系列データ分析処理について，対象とする問合せを規定するとともに，その問合せ処理における特性の分析を行い，多数の自己結合とリレーション間の高い相関性に起因して，問合せ最適化によって効率的な問合せ実行計画の選択が本質的に容易でないことを示した．また当該分析に基づいて，時系列イベント分析処理の処理性能指標である *h*ESPR ベンチマークの基本設計を示した．オープンソース関係データベースシステムである PostgreSQL と MySQL を用いた *h*ESPR ベンチマークの評価実験により，MySQL に比して PostgreSQL が全体的に優れた処理性能を示すものの，一部の問合せにおいては最大で 63 倍以上性能が低い事象や，PostgreSQL のバージョンアップに伴う最大 96 倍の顕著な性能低下を捉えるなど，関係データベースシステムの性能評価としての *h*ESPR ベンチマークの有効性を確認した．

今後はより問合せの充実を図ることでワークロード特性の網羅性を向上させるとともに，より大規模なデータベースや，多数の関係データベース実装を用いた評価実験を行うことで，時系列イベント分析処理のベンチマークの性能指標としての有効性の議論を深化させてゆきたい．

謝辞

本研究の一部は，国立研究開発法人新エネルギー・産業技術総合開発機構 (NEDO)「 IoT 推進のための横断技術開発プロジェクト ／先進 IoT サービスを実現する革新的超省エネルギー型ビッグデータ基盤の研究開発」 に拠る．

参考文献

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 3–14, Washington, DC, USA, 1995. IEEE Computer Society.

[2] M. P. Andersen and D. E. Culler. Btrdb: Optimizing storage system design for timeseries processing. In *14th USENIX Conference on File and Storage Technologies (FAST 16)*, pages 39–52, Santa Clara, CA, 2016. USENIX Association.

[3] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. Sequential pattern mining using a bitmap representation. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 429–435, New York, NY, USA, 2002. ACM.

[4] P. Boncz, T. Neumann, and O. Erling. Tpc-h analyzed: Hidden messages and lessons learned from an influential benchmark. In R. Nambiar and M. Poess, editors, *Performance Characterization and Benchmarking*, volume 8391 of *Lecture Notes in Computer Science*, pages 61–76. Springer International Publishing, 2014.

[5] K.-P. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*, pages 126–133, Mar 1999.

[6] S. Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 34–43, New York, NY, USA, 1998. ACM.

[7] S. Chaudhuri. Query optimizers: time to rethink the contract? In U. Çetintemel, S. B. Zdonik, D. Kossmann, and N. Tatbul, editors, *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2009, Providence, Rhode Island, USA, June 29 - July 2, 2009*, pages 961–968. ACM, 2009.

[8] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, pages 143–154, New York, NY, USA, 2010. ACM.

[9] T. P. P. Council. Tpc-h benchmark specification, 2008.

[10] T. G. Dietterich and R. S. Michalski. Discovering patterns in sequences of events. *Artificial Intelligence*, 25(2):187 – 232, 1985.

[11] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Rec.*, 23(2):419–429, May 1994.

[12] A. Ghazal, T. Rabl, M. Hu, F. Raab, M. Poess, A. Crolotte, and H.-A. Jacobsen. Bigbench: Towards an industry standard benchmark for big data analytics. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 1197–1208, New York, NY, USA, 2013. ACM.

[13] A. Ghosh, J. Parikh, V. S. Sengar, and J. R. Haritsa. Plan selection based on query clustering. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 179–190. VLDB Endowment, 2002.

[14] H. Hacigumus, Y. Chi, W. Wu, S. Zhu, J. Tatemura, and J. F. Naughton. Predicting query execution time: Are optimizer cost models really unusable? In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)*, ICDE '13, pages 1081–1092, Washington, DC, USA, 2013. IEEE Computer Society.

[15] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*, pages 106–115, Mar 1999.

[16] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. Freespan: Frequent pattern-projected sequential pattern mining. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '00, pages 355–359, New York, NY, USA, 2000. ACM.

[17] N. Kabra and D. J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans.

SIGMOD Rec., 27(2):106–117, June 1998.

[18] E. Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th International Conference on Very Large Data Bases*, VLDB '02, pages 406–417. VLDB Endowment, 2002.

[19] Y.-N. Law, H. Wang, and C. Zaniolo. Query languages and data models for database sequences and data streams. In *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pages 492–503. VLDB Endowment, 2004.

[20] V. Leis, A. Gubichev, A. Mirchev, P. Boncz, A. Kemper, and T. Neumann. How good are query optimizers, really? *Proc. VLDB Endow.*, 9(3):204–215, Nov. 2015.

[21] V. Markl, V. Raman, D. Simmen, G. Lohman, H. Pirahesh, and M. Cilimdzic. Robust query processing through progressive optimization. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, pages 659–670, New York, NY, USA, 2004. ACM.

[22] G. Moerkotte, T. Neumann, and G. Steidl. Preventing bad plans by bounding the impact of cardinality estimation errors. *Proc. VLDB Endow.*, 2(1):982–993, Aug. 2009.

[23] Oracle. Oracle8i time series user's guide, 1999.

[24] P. E. O ' Neil, E. J. O ' Neil, and X. Chen. The star schema benchmark (ssb). *Pat*, 200(0):50, 2007.

[25] T. Pelkonen, S. Franklin, J. Teller, P. Cavallaro, Q. Huang, J. Meza, and K. Veeraraghavan. Gorilla: A fast, scalable, in-memory time series database. *Proc. VLDB Endow.*, 8(12):1816–1827, Aug. 2015.

[26] D. Rafiei and A. O. Mendelzon. Querying time series data based on similarity. *IEEE Transactions on Knowledge and Data Engineering*, 12(5):675–693, Sep 2000.

[27] R. Ramakrsihnan, D. Donjerkovic, A. Ranganathan, K. S. Beyer, and M. Krishnaprasad. Srql: Sorted relational query language. In *Proceedings. Tenth International Conference on Scientific and Statistical Database Management (Cat. No.98TB100243)*, pages 84–95, Jul 1998.

[28] N. Reddy and J. R. Haritsa. Analyzing plan diagrams of database query optimizers. In *Proceedings of the 31st International Conference on Very Large Data Bases*, VLDB '05, pages 1228–1239. VLDB Endowment, 2005.

[29] R. Sadri, C. Zaniolo, A. Zarkesh, and J. Adibi. Optimization of sequence queries in database systems. In *Proceedings of the Twentieth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '01, pages 71–81, New York, NY, USA, 2001. ACM.

[30] P. Seshadri, M. Livny, and R. Ramakrishnan. Sequence query processing. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, SIGMOD '94, pages 430–441, New York, NY, USA, 1994. ACM.

[31] P. Seshadri, M. Livny, and R. Ramakrishnan. Seq: A model for sequence databases. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 232–239, Mar 1995.

[32] R. Snodgrass. *The TSQL2 Temporal Query Language*. The Springer International Series in Engineering and Computer Science. Springer US, 2012.

[33] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan. Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explor. Newsl.*, 1(2):12–23, Jan. 2000.

[34] Timescale. Timescaledb: Sql made scalable for time-series data, 2016.

[35] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings 14th International Conference on Data Engineering*, pages 201–208, Feb 1998.

[36] S. Yin, A. Hameurlain, and F. Morvan. Robust query optimization methods with respect to estimation errors: A survey. *SIGMOD Rec.*, 44(3):25–36, Dec. 2015.

付録 A. 問合せ SQL 一覧

測定に使用した問合せ SQL の一覧を掲載する.

Q.1

```
SELECT e.o_orderkey, e.o_custkey, f.o_orderkey,
  f.o_custkey, date_trunc('month', e.o_orderdate),
  date_trunc('month', f.o_orderdate)
FROM orders e, orders f
WHERE e.o_custkey = f.o_custkey AND f.o_orderdate
  BETWEEN date_trunc('month', e.o_orderdate)
   + interval '1 month' AND
  date_trunc('month', e.o_orderdate)
   + interval '1 month' * 2 - interval '1 day'
  AND e.o_orderdate BETWEEN cast('1994-1-1' as date)
  AND cast('1994-1-1' as date)
   + interval '1 month' - interval '1 day';
```

Q.2

```
SELECT e_orderkey,  e_partkey,
  e_orderdate + f_info[1] * interval '1 day'
    as f_min_orderdate,
  e_orderdate + f_info[2] * interval '1 day'
   as f_max_orderdate,
  f_info[3] as f_quantity, f_info[4] as f_order_count
FROM
 (SELECT e.o_orderkey  as e_orderkey,
   e.l_partkey as e_partkey,
   e.o_orderdate as e_orderdate,
   (SELECT ARRAY[min(o_orderdate) - e.o_orderdate,
       max(o_orderdate) - e.o_orderdate,
       sum(l_quantity), count(*)]
    FROM orders fo
    INNER JOIN lineitem fl ON o_orderkey = l_orderkey
    WHERE fo.o_orderdate
      BETWEEN e.o_orderdate + interval '1 day' AND
      e.o_orderdate + interval '3 month'
      AND fl.l_partkey = e.l_partkey
    GROUP BY fl.l_partkey) as f_info
```

```
 FROM
  (SELECT * FROM orders
   INNER JOIN lineitem ON o_orderkey = l_orderkey
   WHERE o_orderdate
     BETWEEN cast('1994-1-1' as date) AND
     cast('1994-1-1' as date) + interval '3 month'
     - interval '1 day'
     AND o_custkey between 1 and 150) e) as result
WHERE f_info[3] > 100
```

Q.3

```
SELECT e.o_orderkey,  e.o_orderdate,  e.o_custkey,
  e.l_partkey, f.o_orderkey, f.o_orderdate,
  f.l_partkey, g.o_orderkey, g.o_orderdate,
  g.l_partkey
FROM
 (SELECT * FROM orders
  INNER JOIN lineitem ON o_orderkey = l_orderkey
  INNER JOIN part ON l_partkey = p_partkey
                  AND p_brand = 'Brand#11') e,
 (SELECT * FROM orders
  INNER JOIN lineitem ON o_orderkey = l_orderkey
  INNER JOIN part ON l_partkey = p_partkey
                  AND p_brand = 'Brand#21') f,
 (SELECT * from orders
  INNER JOIN lineitem ON o_orderkey = l_orderkey
  INNER JOIN part ON l_partkey = p_partkey
                  AND p_brand = 'Brand#31') g
WHERE f.o_custkey = e.o_custkey AND g.o_custkey
  = e.o_custkey AND f.o_orderdate
  BETWEEN (e.o_orderdate + 1) AND (e.o_orderdate
  + interval '6 month') AND g.o_orderdate
  BETWEEN (f.o_orderdate + 1) AND (e.o_orderdate
  + interval '6 month' * 2) AND e.o_orderdate
  BETWEEN cast('1994-1-1' as date) AND
  cast('1994-1-1' as date) + interval '6 month'
  - interval '1 day'
```

Q.4

```
SELECT e.o_orderkey,  e.o_orderdate,  e.o_custkey,
 e.l_partkey, f.o_orderkey, f.o_orderdate,
 f.o_custkey, f.l_partkey
FROM
 (SELECT * FROM orders INNER JOIN lineitem
   ON o_orderkey = l_orderkey) e,
 (SELECT * FROM orders INNER JOIN lineitem
   ON o_orderkey = l_orderkey) f
WHERE f.o_custkey = e.o_custkey AND f.l_partkey
  = e.l_partkey AND
  (SELECT count(*) FROM orders
   WHERE o_custkey = e.o_custkey AND o_orderdate
   BETWEEN (e.o_orderdate + interval '1 day') AND
   f.o_orderdate) BETWEEN 1 AND 5
  AND e.o_orderdate BETWEEN cast('1994-1-1' as date)
  AND cast('1994-1-1' as date) + interval '1 month'
  - interval '1 day'
```

Q.5

```
SELECT f.l_partkey, f.o_orderkey, f.o_orderdate
FROM
 (SELECT distinct l_partkey
  FROM orders INNER JOIN lineitem
                  ON o_orderkey = l_orderkey
  WHERE o_orderdate BETWEEN cast('1994-1-1' as date)
   AND cast('1994-1-1' as date) + interval '6 month'
   - interval '1 day') e
INNER JOIN
 (SELECT l_partkey, o_orderkey, o_orderdate
  FROM orders INNER JOIN lineitem
                  ON o_orderkey = l_orderkey
  WHERE o_orderdate
   BETWEEN cast('1994-1-1' as date)
   - interval '6 month' AND cast('1994-1-1' as date)
   - interval '1 day') f
ON e.l_partkey = f.l_partkey
```

Q.6

```
SELECT *
FROM orders fo
WHERE fo.o_orderdate >= cast('1994-1-1' as date)
  + interval '1 month' + interval '1 day'
  AND fo.o_custkey
  BETWEEN 1 AND 50 AND
  (SELECT count(*) FROM orders co
   WHERE co.o_orderdate BETWEEN
    cast('1994-1-1' as date) + interval '1 month'
      + interval '1 day'
    AND fo.o_orderdate
    AND o_custkey BETWEEN 1 AND 50)
  < (SELECT count(eo.*)
     FROM orders eo
     WHERE o_orderdate BETWEEN
       cast('1994-1-1' as date)
       AND cast('1994-1-1' as date)
       + interval '1 month' - interval '1 day'
       AND o_custkey BETWEEN 1 AND 50)
```

Q.7

```
SELECT min(o_orderdate) from (
 SELECT e.o_orderdate
 FROM
  (SELECT distinct o_orderdate FROM orders
   WHERE o_orderdate
     BETWEEN cast('1994-1-1' as date) AND
     cast('1994-1-1' as date) + interval '12 month'
   AND o_custkey BETWEEN 1 and 100) e
 WHERE
  (SELECT sum(o_totalprice) FROM orders
   WHERE o_orderdate
     BETWEEN cast('1994-1-1' as date) AND
     cast('1994-1-1' as date) + interval '12 month'
   AND o_orderdate <= e.o_orderdate
   AND o_custkey between 1 and 100) >=1000000
) a
```