Oracleホワイト・ペーパー
2013年6月

Patterns everywhere – Find them fast!
Oracle Database 12cのSQLパターン・マッチング

ORACLE

概要 2
パターン・マッチングの概要 3
あらゆる場所に存在するパターン 3
パターンの検出方法 3
Oracle Database 12cを使ったSQLパターン・マッチング 3
パターン・マッチングで使用されるキーワードの概要 5
追加機能 8
まとめ 10
SQLパターン・マッチングのユースケース 11
株式市場の分析 11
ガバナンスとコンプライアンスの実現 12
オンライン顧客の行動の監視 13
コール・サービス品質の追跡 15
マネー・ロンダリングの追跡 16
侵入者の検出 17
結論 18

概要

パターンは通常、特定のイベントやアクションの繰り返しまたはシーケンスと定義され、ビジネスのあらゆる場所で発生します。データセット内の個々のパターンやパターン・グループを検索、分析、定量化する機能は、今や重要なビジネス要件になっています。この機能は、顧客の行動や関連する運用アクティビティを把握し、収益を上げる新しいチャンスを見つけ、ビジネスの大幅なコスト増の原因となる可能性のある悪意のあるアクティビティを特定するのに役立ちます。

今まで、データベース・システムのもっとも汎用的な言語であるSQLには、パターンを迅速かつ簡単に定義、追跡するための包括的なサポートがありませんでした。このため、開発者が難解で時間がかかる複雑なカスタム・コードを作成して、最適化と保守を行う必要がありました。このような開発者中心のルーチンを他のビジネス・ユーザーと共有することは、簡単ではありません。つまり、ビジネス・ユーザーは自分のデータ検出ワークフローにパターン・マッチングを組み込めないということです。このためビジネスにとっては、パターン・マッチングに対する既存のアプローチでは全体的な価値が制限されてしまいます。

Oracle Database 12cでは、SQLにネイティブのパターン・マッチング機能が追加されています。この機能によって、データセット内のパターン識別プロセスにおいて、もっとも汎用的なデータ分析言語を簡単かつ効率的に使用できます。このため、アプリケーションに組み込まれてクライアント側か中間層アプリケーション・サーバー内で実行されるカスタム・コードを使い続ける場合と比べて、パフォーマンス、管理性、スケーラビリティが大幅に向上します。また、開発者は独自のパターン・マッチング・ロジックをOracle Database 12cの新しいSQLベースのパターン・マッチング機能に切り替えることで、簡単なコードを使用でき、処理効率とパフォーマンスを上げることができます。

ビジネス・ユーザーとデータ・サイエンティストにとって、データベース内パターン・マッチングの主な利点は、豊富なパターンを既存の分析ワークフローに組み込めることです。このとき、ネットワーク・トラフィックやデータ待機時間の増加につながる複雑な外部コード・ルーチンを使用する必要はありません。

このホワイト・ペーパーでは、この新しいSQL機能の詳細と、オラクルの新しいSQLパターン・マッチングの豊富な機能を、さまざまな業界で利用する方法について説明します。

パターン・マッチングの概要

あらゆる場所に存在するパターン

世界には、たくさんのデータがあふれています。ビジネスで生成、取得され、データ・リポジトリに移動されるデータ量は、取得速度の向上に伴って急速に増えています。ビッグ・データ関連の新しいテクノロジー分野への関心が高まるにつれて、データの作成量や取得量も増えています。さまざまなアクティビティや相互作用を測定、記録するため、センサーを搭載したデバイスも増えています。企業は、このような新しいデータセット内のパターンを求める簡単かつ効率的な方法を必要としています。この広大なデータの海原には、まだ見つかっていないさまざまなパターンが眠っています。

パターンの検出方法

パターン認識の技術は、多くの企業でさまざまな方法で使用されています。典型的な例としては、時系列の財務データセットの処理による市場機会の調査、ネットワーク・ログの分析による特定のコール・パターンの調査（サービス品質の測定に使用可能）、不正検出（小さいトランザクションが続いた後に高価値のトランザクションが行われる場合は、商品やサービスの詐取を示すなど）、およびファイアウォールやセキュリティのログ分析による侵入者の検索および検出などがあります。

Oracle Databaseでは、LIKE条件や正規表現などの機能を使用した、シンプルなパターンの検出がすでにサポートされています。開発者やビジネス・ユーザーは、これらの機能を使用して、一部のパターン・マッチング処理をOracle Database内に戻すことで、パフォーマンス、スケーラビリティ、セキュリティの向上を実現してきました。

より高度なパターン・マッチング要件の場合、アプリケーションとデータベースの開発者は、別のパターン・マッチング・テクノロジー（専用ソフトウェアなど）を実装する必要がありました。または、（アプリケーション内で直接、または中間層プラットフォーム内で）さまざまな言語を使用してアプリケーションにカスタム・コードを追加していました。このような場合は一般的に、多くのデータが移動されます。たとえば、データベースからのデータの取得、パターン・マッチング・コードを実行してから同じデータベースや他のターゲット・アプリケーション/データベースにデータを戻すなどの処理に伴うデータ移動です。

多くの場合、データベースに対してこのようにデータを出し入れすると、パフォーマンスの最適化の複雑化、データ待機時間の増加、データ・セキュリティの低下などに関連するさまざまな問題が発生します。データ・マイニングなど高度な分析処理で定められている場合と同様に、（分析のためにデータを移動するのではなく）分析をデータに移動することが理想的です。パターン・マッチングで開発者とビジネス・ユーザーに必要なのは、既存のアプリケーション処理やBIレポート内に組み込んで（Oracle Database内などの）インプレース・データを処理できる、豊富なSQL表示機能です。

Oracle Database 12cを使ったSQLパターン・マッチング

Oracle Database 12cでは、連続するパターン（行ストリーム）を定義するコアのルール・セットの実装により、Perlの正規表現機能を採用した、まったく新しいネイティブのSQL構文が導入されています。この新しい行間のパターン検索機能によって、シングル・レコードの文字列内でパターンを照合する、正規表現の既

存機能が補完されます。12cのMATCH_RECOGNIZE機能を使用すると、文字や文字セットについてパターンを定義し、行の境界を越えてこれらのパターンを検索できます。

この新しいMATCH_RECOGNIZE()という句によって、SQL内でのネイティブな宣言的パターン・マッチング機能が実現されます。以下は、株価が下落してから上昇する（下落&上昇または'V字型'パターン）、一般的な株価パターンの識別に使用される構文の簡単な例です。

```
SELECT *
FROM stock_ticker
MATCH_RECOGNIZE (
 PARTITION    BY    symbol
 ORDER BY timestamp
 MEASURES STRT.timestamp AS start_timestamp,
       FINAL  LAST(DOWN.timestamp)  AS  bottom_timestamp,  FINAL
       LAST(UP.timestamp) AS end_timestamp
 ALL ROWS PER MATCH
 AFTER  MATCH  SKIP  TO  LAST  UP
 PATTERN(STRT DOWN+ UP+) DEFINE
       DOWN AS DOWN.price < PREV(DOWN.price),
       UP AS UP.price > PREV(UP.price)
)MR
ORDER BY MR.symbol, MR.timestamp;
```

この例では、市場取引データが含まれるstock_ticker表が、パターン・マッチング・プロセス用のデータ・ソースになります。

MATCH_RECOGNIZEという句は、分析対象のデータ・ストリーム（この例ではstock_ticker表）を論理的に分割して並べ替えるためのものです。上記のコード抽出でわかるとおり、この新しい句の構文は、非常に機能豊富で包括的、かつわかりやすいものです。

MATCH_RECOGNIZE構造の**PATTERN**句によって、レコード・ストリーム内で識別するパターンが定義され、正規表現構文が使用されます。次に、**DEFINE**句でSQLに似た構文を使用して各パターン変数が記述され、動作（イベント）における各行または行間の変更が識別されます。パターンを定義するために使用する構造は、正規表現の宣言的言語（Perlなど）に精通している開発者には、よく知られています。

MATCH_RECONGIZE句の構築には、4つの基本的な手順があります。

1. 分析対象の'イベント・ストリーム'の識別に必要なパーティション/バケット、および順序を定義する
2. イベント・パターンと、パターン内の個々のイベントを識別するパターン変数を定義する
3. メジャーを定義する：パターンに関連するソース・データ・ポイント、パターン・データ・ポイント、および集計
4. 出力の生成方法を決定する

次の項では、上記の簡単な4つの手順での、MATCH_RECOGNIZE句におけるもっとも重要なキーワードの概要を説明します。この機能について詳しくは、Oracle Database 12c SQLのドキュメントを参照してください。このドキュメントは、Oracle Technical Network（OTN）のWebサイト[1]で入手できます。

パターン・マッチングで使用されるキーワードの概要

この項では、MATCH_RECOGNIZE句を構成するもっとも重要なキーワードの概要を説明します。キーワードは上記の4つの手順に従ってまとめられています。

手順1：データのバケットと並べ替え

データのバケットと並べ替えは、“**PARTITION BY**”と“**ORDER BY**”というキーワードによって制御されます。

PARTITION BY

この句によってデータが論理グループに分けられるため、各グループ内で必要なパターンを検索できます。“PARTITION BY”はオプションの句であり、通常はその後に“ORDER BY“句が続きます。

```
PARTITION BY symbol
ORDER BY timestamp
```

この例では、stock_ticker表のデータを、株価表示記号ごとの個別のレコード・ストリームに分けるようにします。これで、各株価表示記号内の下落/上昇のV字型パターンを検索できます。

ORDER BY

データの順序は非常に重要です。データの順序によって、データセット内で検索するパターンの“可視性”が制御されるためです。MATCH_RECOGNIZE機能では、ORDER BY句を使用してデータが整理されるため、“イベント・ストリーム”全体の行シーケンスの存在を、行の境界を越えてテストできます。

```
PARTITION BY symbol
ORDER BY timestamp
```

[1] Oracle Database のドキュメント・ライブラリへのリンク：http://www.oracle.com/technetwork/jp/indexes/documentation/index.html

この例では、各株価表示記号のグループ内で、タイムスタンプ情報に基づいてデータを並べ替えています。これはおそらく、データセットのもっとも一般的な並べ替え方法です。ほとんどのパターンは、何らかの形で時間に依存しているためです。

手順2：パターンの定義

PATTERN句では、正規表現を使用して、ある特定のパターンに関する条件を定義します。その後の構文**DEFINE**句を使用して、関連するパターン変数を定義します。

PATTERN

この項では、次の3つの基本要素を定義します。

1. 一致する必要があるパターン変数
2. パターン変数が一致する必要があるシーケンス
3. 一致する必要があるパターンの頻度

例：

```
PATTERN(STRT DOWN+ UP+)
```

この例では、イベント・ストリームでのパターンを定義します。ここでは、1つまたは複数の'DOWN'イベント'DOWN'（開始イベント'STRT'から始まる1つまたは複数の'UP'イベントが後に続く）が表示されるようにします。

DEFINE

PATTERN句は、パターン変数に依存します。つまり、これらの変数を定義する句が必要です。定義には、DEFINE句が使用されます。これは、行を特定のパターン変数にマッピングする際に必要な条件を指定するための必須の句です。

例：

```
DEFINE
       DOWN AS DOWN.price < PREV(DOWN.price),
       UP AS UP.price > PREV(UP.price)
```

この例では、'DOWN'イベントを現在の価格が前の行の価格より低い行として、また'UP'イベントを現在の価格が前の行の価格より高い行として定義しています。この例からわかるとおり、パターン句で使用されるイベント'STRT'の定義はありません。パターン変数に定義は必要ありません。未定義のパターン変数にいずれかの行をマッピングすれば、always-trueイベントが作成されます。この例では、これですべてのレコードが特定のパターンに対してテストされる可能性があるイベントと見なされます。"STRT"イベントをMEASURE句に追加すると便利な場合があります。これについては、次の項で説明します。

手順3：メジャーの定義

この項では、式の評価から派生したり、特定の一致に関連する式の評価によって計算される出力メジャーを定義します。

MEASURES

メジャーは、パターン内の個別のデータ・ポイントである場合があります。たとえば、開始/終了値、averageやcountなどのイベント行セットの集計、パターン変数識別子などのSQLパターン・マッチング固有の値などです。次に、パターン内のデータ・ポイントの定義方法の簡単な例を示します。

```
MEASURES STRT.timestamp AS start_timestamp,
         FINAL   LAST(DOWN.timestamp)   AS   bottom_timestamp,
         FINAL LAST(UP.timestamp) AS end_timestamp,
```

この例では、株価の長期間のV字型パターンが識別されるようにします。V字型を識別するため、このパターンの特性として次のメジャーを追跡します。

- V字型の開始日付（start_timestamp）
- V字型の底部の日付（bottoom_timestamp）
- V字型の最終日の日付（end_timestamp）

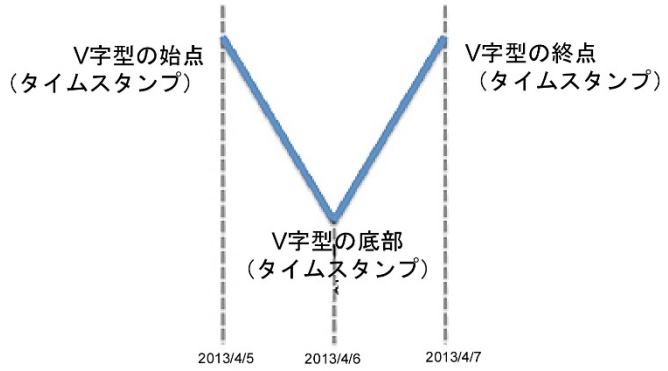

図1 – 株価表示データセットの典型的なV字型パターン

特定のパターンと一致する行の取得方法のコンテキストで、メジャーを確認することは重要です。1つのパターン内で1つのイベントが複数回発生する可能性があるためです。たとえば上記のパターンでは、V字の底部に達する前に、DOWNイベントが複数回発生していた可能性があります。このため、開発者は出力に含めるイベントを決定する必要があります（最初の値、最後の値、最低値、最高値など）。予想どおりの正しい要素が返されるようにするには、メジャーを厳密に定義することが非常に重要です。

手順4：出力の制御

[ALL ROWS | ONE ROW] PER MATCH

行ストリームでパターンが識別されたら、（パターンごとの）サマリー情報や、詳細情報をレポートする機能が必要になるかもしれません。このプロセスの一部としてサマリーを生成できることは非常に重要です。"`PER MATCH`"句では、一致に関するサマリー・データか詳細データを作成できます。これら2種類の出力は、次のキー・フレーズで制御されます。

- ONE ROW PER MATCH - 一致ごとに1つのサマリー行が生成されます。これがデフォルトです。
- ALL ROWS PER MATCH - 複数行にわたる一致で、一致の行ごとに1つの出力行が生成されます。

以下は、制御のための追加の文が含まれる構文の例です。ここでは、一致が確立されるとプロセスが再起動します。

```
ALL ROWS PER MATCH
AFTER MATCH SKIP TO LAST UP
```

上記の例で、株価表示のV字型に当てはまるレコードがすべてレポートされるようにします。前に定義したメジャーは、一致行ごとに計算され、株価表示の表の列値に追加してレポートされます。出力の種類を決定したら、次にすべてのパターンが一致した場合の制御フローを決定します。

AFTER MATCH SKIP

`AFTER MATCH SKIP`句によって、一致が確立された場合に行パターン・マッチング・プロセスを再開できるポイントが決定されます。この句を制御するオプションは、次のとおりです。

- AFTER MATCH SKIP TO NEXT ROW|PAST LAST ROW- 現在の一致の最初/最後の行の後の行で、パターン・マッチングが再開されます。
- AFTER MATCH SKIP TO FIRST| LAST <pattern_variable> - パターン変数にマッピングされた最初/最後の行で、パターン・マッチングが再開されます。
- AFTER MATCH SKIP TO <pattern_variable> - パターン変数にマッピングされた最後の行で、パターン・マッチングが再開されます。

この句のデフォルトは、AFTER MATCH SKIP PAST LAST ROWです。

追加機能

`MATCH_RECOGNIZE`句を使用すると、非常に高度なパターンを検索できます。出力に含めることができる2つのキーワードによって、これらのパターンの作成とテストがずっと簡単になります。

- MATCH_NUMBER
- CLASSIFIER

これら2つのメジャーを問合せとして含めることで、出力をフィルタリングできます。BIユーザーがこれらのメジャーを使用すると、ダッシュボードやレポートでのデータの表示方法を制御しやすくなります。開発者がこの種類の情報のあらゆる用途を検索することは多くあります。MATCH_RECOGNIZE句の結果の監査が必要な場合には、出力の一部としてこれら2つのメジャーを含めることをお勧めします。これらのメジャーが提供する情報は非常に重要であるためです。

MATCH_NUMBER

ほとんどのパターンでは、特定のパーティション内に大量の一致が生成されがちです。MATCH_NUMBERでは、パターン一致ごとに、見つかった順序で1から始まる連番が返されます。このため、パターン条件の適用方法の分析がずっと簡単になります。

CLASSIFIER

CLASSIFIERによって、特定の行に適用されているパターン全体の変数名が返されます。これは、行パターン・マッチングによる行のマッピング先のパターン変数です。MATCH_NUMBER機能と同様に、開発者はテスト中にこの機能を使用して、パターンが正しく定義されており、予想した結果が返されているかどうかを特定できます。

以下のコードは、これら2つのメジャーの論理和を示しています。

```
MEASURES
       STRT.tstamp AS start_tstamp,
       FINAL LAST(DOWN.tstamp) AS bottom_tstamp,
       FINAL LAST(UP.tstamp) AS end_tstamp,
       MATCH_NUMBER() AS match_num,
       CLASSIFIER() AS var_match
```

上記の文からの出力は、次のようになります。

```
SYMBOL     START_TST BOTTOM_TS END_TSTAM MATCH_NUM VAR_MATCH
---------  --------- --------- --------- ---------- --------------
ACME       05-APR-11 06-APR-11 10-APR-11          1 STRT
ACME       05-APR-11 06-APR-11 10-APR-11          1 DOWN
ACME       05-APR-11 06-APR-11 10-APR-11          1 UP
ACME       05-APR-11 06-APR-11 10-APR-11          1 UP
ACME       05-APR-11 06-APR-11 10-APR-11          1 UP
ACME       05-APR-11 06-APR-11 10-APR-11          1 UP
ACME       05-APR-11 06-APR-11 10-APR-11          1 UP
ACME       10-APR-11 12-APR-11 13-APR-11          2 STRT
ACME       10-APR-11 12-APR-11 13-APR-11          2 DOWN
ACME       10-APR-11 12-APR-11 13-APR-11          2 DOWN
ACME       10-APR-11 12-APR-11 13-APR-11          2 UP
```

パターン内の集計の追加

パターン・マッチング・ワークフロー内で重要な要素の集計を作成することは、一般的な要件です。SQLを使用してパターン・マッチング・ロジックを定義すると、標準SQLの演算子を使用したパターン・マッチング・プロセス中に、データの操作方法を簡単に拡張できます。SQLの集計（**COUNT**、**SUM**、**AVG**、**MAX**、**MIN**など）を、**MEASURES**と**DEFINE**の両方の構文に組み込んで、より豊富な結果セットを作成できるようにな

りました。これでアプリケーション・コードを大幅に簡素化できるため、特に開発者にとって便利です。ビジネス・ユーザーは、この機能を利用してパターンやパターン・グループを迅速かつ効率的にまとめられるため、より簡単に分析を行うことができます。

集計を行パターン・マッチングで使用すると、特定のパターン変数にマッピングされた行セットに対して機能するため、結果セット内に実行中や最終の集計を作成できます。以下の例では、元のstock_tickerの例を拡張して、**COUNT()**と**AVG()**の集計機能を使用しています。

```
SELECT * FROM stock_ticker
MATCH_RECOGNIZE (
 PARTITION BY symbol
 ORDER BY timestamp
 MEASURES
       COUNT(*) as total_no_of days,
       COUNT(DOWN.*)AS down_days, COUNT(UP.*)AS
       up_days, AVG(DOWN.price) AS
       avg_down_price, AVG(UP.price) AS
       avg_up_price
 ONE ROW PER MATCHAFTER MATCH SKIP TO LAST UP PATTERN(STRT
 DOWN+ UP+)
 DEFINE
       DOWN AS DOWN.price < PREV(DOWN.price),
       UP AS UP.price > PREV(UP.price)
)MR
ORDER BY MR.symbol, MR.timestamp;
```

COUNT(*)の構文が拡張されて、パターン・マッチングにより機能豊富な構文になっています。これで、個々のパターンをカウントできるようになり、COUNT(DOWN.*)でパターン変数DOWNにマッピングされた行数がカウントされます。

RUNNINGとFINALという重要な2つのキーワードを使用すると、集計処理をより制御しやすくなります。これら2つのキーワードを使用すると、実行中または最終の集計を、結果セット内で返すかどうかを指定できます。これらのキーワードは、両方の集計、および標準行パターンのナビゲーション操作（FIRSTやLASTなど）と一緒に使用できます。

まとめ

SQLパターン・マッチングの一般的な構文は非常に機能豊富であり、複雑なパターンを簡単に定義できます。上記の例ではこの機能のもっとも基本的な要素の概要のみを説明しています。

詳しくは、Oracle Database 12cのデータ・ウェアハウス・ガイドやSQL言語リファレンス・ガイドを参照してください。これらのドキュメントは、OTNのWebサイト[2]で入手できます。

SQLパターン・マッチングのユースケース

この項では、さまざまなユースケースを参考に、概念的に簡素化されたSQLによって、新しいSQLパターン・マッチング機能を使用してこれらのビジネス要件を解決します。これらの例は、金融サービス、小売業、ライフサイエンス、電気通信の各業界の顧客からの実際の要件に基づいていますが、使用例は他の多くの業界にも適用できます。

株式市場の分析

株式市場は、パターンの検出と理解に対するニーズによって動いています。トレーディング部門のマネージャーは必ず、事前定義した使用例やパターンをいくつか持っており、これによってダッシュボード内でアラートがトリガーされます。機能豊富なパターン定義構文を使用すれば、図2のように、U字型、V字型、逆V字型、W字型、エリオット波動などの典型的な取引価格パターンを簡単に抽出できます。

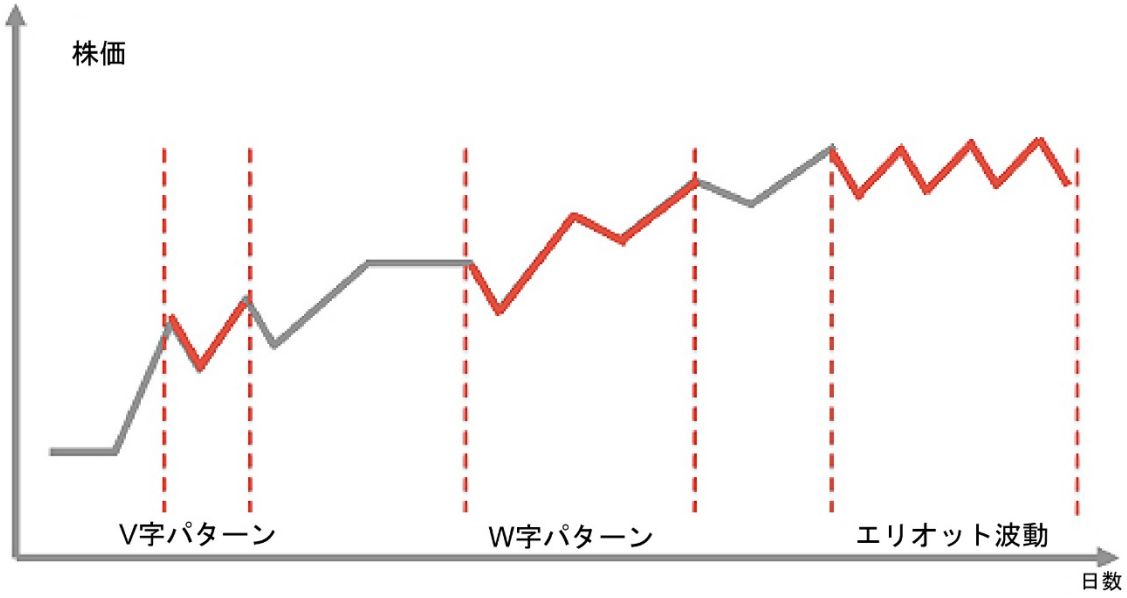

図2 – 金融市場の典型的なパターン

[2] Oracle Databaseのドキュメント・ライブラリへのリンク：
http://www.oracle.com/technetwork/jp/indexes/documentation/index.html

開発者は、MATCH_RECOGNIZE構文の豊富な機能を使用して、非常に複雑なパターンを作成できます。たとえば、8%を超えて価格が下落した後、同日または数日内の株価が元の価格より低いままである株式の検索などです。上記のパターンの構文は次のようになります。

```
...MATCH_RECOGNIZE (
       PARTITION  BY  symbol
       ORDER BY timestamp
    MEASURES
       A.tstamp as start_timestamp,
       A.price as start_price,
       B.price as drop_price,
       COUNT(C.*)+1 as cnt_days,
       D.tstamp as end_timestamp,
       D.price as end_price
   ONE ROW PER MATCH
   AFTER  MATCH  SKIP  PAST  LAST  ROW
   PATTERN (A B C* D)
   DEFINE
       B  as  (B.price  -  A.price)/A.price  <  -0.08,  C
       as C.price < A.price,
D as D.price >= A.price)
```

メジャーCNT_DAYSによって、価格が8%を超えて下落する前の初期価格より、株価が低いままである期間が計算されます。またEND_PRICEによって、株価がこの下落傾向から脱したときの価格が返されます。

パターン構文は、より複雑なパターン（複数の逆V字型が連続するパターンが含まれるエリオット波動など）もサポートします。パターン式では、1日以上の価格上昇と、その後の1日以上の価格下落を検索する必要があります。このシーケンスは指定した期間、中断なしで連続して表示される必要があります。

どのような株価開発パターンを調べる場合でも、後で他の分析機能（データセットを一面/多面的に細かく分けて分析するデータ・マイニングや多次元分析など）を使用して、結果をより詳しく分析できます。パターン・マッチング・プロセスをデータベース内に移動すると、このような組合せ分析がずっと簡単になります。データを移動する必要がなく、待機時間が短縮されるため、リアルタイムに分析できるためです。

ガバナンスとコンプライアンスの実現

監査担当者やコンプライアンス・チームは、SQLパターン・マッチングを使用して、（短期間に大量の取引が行われた）大商い株や、異常なアクティビティを示す株の有無を大量の取引データを通じて探すことができます。たとえば、大商いは、1時間に3つのトランザクションが発生し、各トランザクションが30,000株を超える場合、と定義することができます。

```
SELECT *
 FROM stock_trades
 MATCH_RECOGNIZE (
    PARTITION BY symbol
    ORDER BY tstamp
    MEASURES
           FIRST (A.tstamp) AS in_hour_of_trade, SUM
           (A.volume) AS sum_of_large_volumes
    ONE ROW PER MATCH
    AFTER MATCH SKIP PAST LAST ROW PATTERN
```

```
(A B* A B* A)
DEFINE
   A AS ((A.volume > 30000) AND
  ((A.tstamp - FIRST (A.tstamp)) < '001:00:00.00' )), B
  AS ((B.volume <= 30000)
  AND ((B.tstamp - FIRST (A.tstamp)) < '01:00:00.00'))
);
```

コンプライアンス・チームは、アプリケーション・ログを使用して、顧客口座の不正な変更を示すパターンを探します。たとえば税務署員が、個人の税金記録の大規模変更（税の減額のための未認可の手動調整など）の前に、一連の細かい変更を行う場合があるかもしれません。

SQLベースのパターン・マッチングは、コンプライアンスのような要件に最適です。変更やその管理がずっと簡単であるためです。このような種類のパターンは、時間をかけて進化します。犯罪者は常にコンプライアンス・チームの先手を打とうとするためです。SQLベースのパターン・マッチングは、所定の検索パターンを迅速かつ簡単に変更して、わずかに異なるイベントや新しいイベントを探す機能です。

オンライン顧客の行動の監視

パターン・マッチングでもっとも使用されるユースケースの1つは、Webサイトでのユーザー・アクティビティの分析です。パターン・マッチングを使用して一連のクリックでの各セッションを識別し、通常は複数のイベントが含まれるユーザー・アクティビティを追跡します。たとえばクリック動作、ページAからページBへの移動、お勧めへの反応などです。この分析のデータを、図3のようなレポートにプッシュできます。この図は、Webサイト中のトラフィック・フローを示しています。

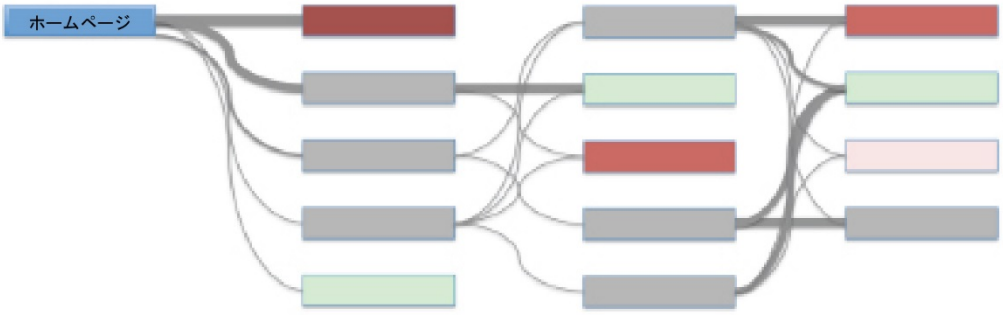

図3：Webサイトの使用状況に関する、A-Bのテストのリアルタイムな結果

全体的な目的は、一部のパスが失敗し（プロモーションを表示しても購入やクリックにつながらない）、一部のパスが成功する（プロモーションが購入やクリックにつながる）理由を把握することです。

詳細レベルでのセッション番号の割当ては、分析プロセスの始まりにすぎません。このような分析にパターン・マッチングを使用するビジネス上の本当の価値は、セッションごとにデータを集計して、全体図が判明したときに発揮されます。一般的な集計にはセッション単位、およびセッション期間全体のイベントの要約が含まれます。

この分析での最初の手順は、セッションの定義です。セッションとは、タイムスタンプ間の時間間隔が指定したしきい値より短い、同じパーティション・キーが含まれる一連の時系列の行です。以下の例では、セッション時間のしきい値を10秒と設定しています。

```
SELECT time_stamp, user_id, session_id
FROM Events
MATCH_RECOGNIZE (PARTITION
  BY User_ID
    ORDER BY Time_Stamp
    MEASURES
        MATCH_NUMBER() AS session_id,
        COUNT(*) AS no_of_events,
        FIRST(time_stamp) AS start_time, LAST(time_stamp) -
        FIRST(time_stamp) AS duration
    PATTERN (b s*)
    DEFINE
        s AS (s.Time_Stamp - PREV(Time_Stamp) <= 10)
    )
    ORDER BY user_id, session_id;
```

行のタイムスタンプ間の時間差が指定したしきい値より大きい場合、これらの行は別のセッションにあると見なされます。セッションを識別したら、次の内容の定量化などの、より詳しい分析を実行できます。

- 一般的なセッションで、各ユーザーが訪問するページ数
- 各ユーザーが、次のページに移動する前に各ページに費やす時間

オンライン顧客の行動を把握することは、サービスの提供、向上、価格設定、マーケティング・キャンペーンなどの決定に役立つため、組織にとって大きなメリットになります。

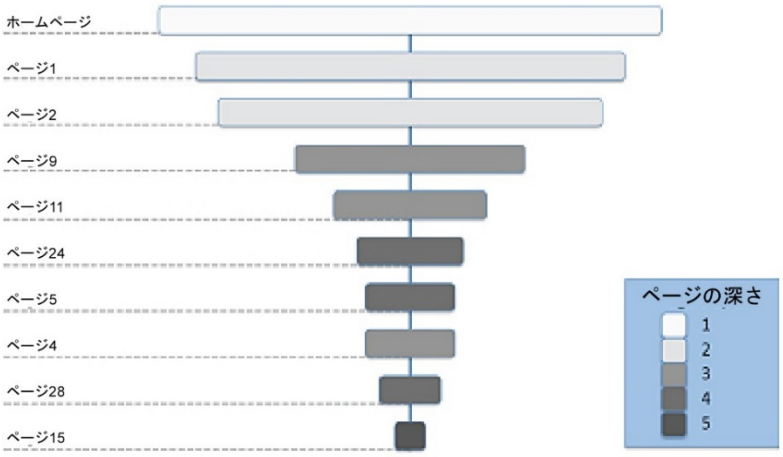

図4 – Webサイトの分析：訪問者ナビゲーションのページの深さ

図4のようなデータ可視化や、Oracle Enterprise Rのサポートに使用されるような分析ツールを使用すると、ビジネス・ユーザーがこの新しいパターン・マッチング機能を使用して、ページごとのクリック回数や合計滞在時間を迅速かつ簡単に特定できます。次に、これらの結果を使用して、さらに多くのレイヤーで分析できます（ページごとの最長、最短、平均滞在時間など）。

コール・サービス品質の追跡

電気通信会社は、自社のネットワーク・ログのサービス品質分析を行って、接続の切断やユーザーのリダイヤルが含まれる電話セッションのセッション特性を把握します。

このネットワーク・ログの分析は、他のユースケースと比べて少し複雑です。通話が終了したことが明らかでも、実際にユーザーが意図的に通話を終了したとは限らないためです。電話接続の切断時に、ユーザーが携帯電話を使用していた場合を考えてみましょう。通常は、このユーザーはリダイヤルして通話を継続しようとします。このような場合、同じ電話番号のペアを含む複数の通話を、1つの電話セッションの一部と見なす必要があります。

次のコード・スニペットでは、通話間の間隔が60秒のしきい値以内であった場合に2者間の通話が1つのセッションにまとめられているセッションが検索されます。このしきい値は、`DEFINE`句で指定されます。`MEASURES`句では、各通話の次のような情報が返されます。

- 1つのセッションで通話が再開された回数
- 有効な通話の合計時間
- 中断の合計時間

```
SELECT Caller,
        Callee,
        Start_Time,
        Effective_Call_Duration,
        (End_Time - Start_Time) - Effective_Call_Duration AS
                    Total_Interruption_Duration,
        No_Of_Restarts,
        Session_ID
FROM my_cdr
MATCH_RECOGNIZE
       PARTITION BY Caller, Callee
       ORDER BY Start_Time MEASURES
              A.Start_Time AS callstart_time,
              End_Time AS call_end_time,
              SUM(End_Time - Start_Time) AS call_duration,
              COUNT(B.*) AS no_of_dropped_calls MATCH_NUMBER()
              AS pm_reference
       PATTERN (A B*)
       DEFINE
              B AS B.Start_Time - PREV(B.end_Time) < 60
       );
```

これによりサービス品質が判明することが多いため、重要なメトリックです。顧客は電話サービスの品質が悪いと契約を終了するか、または更新しないことが多く、これがコストの増大につながります。

マネー・ロンダリングの追跡

多くの企業は、不正行為を簡単かつより効率的に特定、追跡する方法を求めています。金融サービスや電気通信会社などの業界では、その商品やサービスがマネー・ロンダリングに利用されていることが発覚しています。疑わしい財務パターンを検索する機能は主要な要件になりつつあります。オラクルのSQLパターン・マッチングは処理が高速かつ効率的であるため、この用途に最適なデータベース内SQLメソッドです。

開発者はこの豊富なSQL機能を使用して、業界固有、または特定の環境で、"異常"で"疑わしい"行動パターンを定義できます。たとえば、送金時の疑わしい行動を検索する必要があるとします。30日以内に小口送金（2000ドル未満）が3回以上行われた後、最後の小口送金から10日以内に大口送金（1,000,000ドル超）が行われた場合は、典型的な"疑わしい"送金パターンとして定義される可能性があります。

Pattern句を使用すれば、まず小口送金を識別し、その後の大口送金にフラグを付け、小口送金が指定した時間内（この場合は30日間）に発生した場合に識別し、最終的にこの後、最後の小口送金の指定した時間枠（この場合は10日間）内に発生した大口送金を捕捉する一連の文を定義することは、比較的簡単です。

つまり、このようなパターンの構文は次のようになります。

```
SELECT userid,
        first_t,
        last_t,
        amount
FROM (SELECT * FROM event_log WHERE event = 'transfer')
MATCH_RECOGNIZE
       (PARTITION BY userid
       ORDER BY time MEASURES
              FIRST(x.time) first_t,
              y.time last_t, y.amount
              amount
       PATTERN ( x{3,} y )
       DEFINE
              x AS (event='transfer' AND amount < 2000),
              y AS (event='transfer' AND amount >= 1000000
              AND LAST(x.time) - FIRST(x.time) < 30
              AND y.time - LAST(x.time) < 10)
       );
```

この新しいSQLベースのデータベース内パターン・マッチング機能を使用すると、Oracle Databaseの簡易性、パフォーマンス、スケーラビリティを利用してリアルタイムに不正検出処理を行い、マネー・ロンダリング防止の要件に対応できます。

侵入者の検出

コンピュータ・システムへのハッキングや不正アクセスの増加によって、堅牢かつスケーラブルでパフォーマンスの高いパターン・マッチング・プロセスへのニーズがかつてないほど高まっています。ほとんどのコンピュータ・システムはエラー・メッセージと認証チェックを生成するように構成されています。これらを分析すれば、セキュリティ上の問題や、関連するその他の問題があるかどうかを特定できます。

ほとんどの場合、ログ結合プロセスでは、所定の回数の連続的な認証失敗が調べられ、発信元IPアドレスなどの他の属性は無視されます。パターン・マッチングを使用すれば、このようなパターンの検索を簡単に定義できます。たとえば、コードは次のようになります。

```
...MATCH_RECOGNIZE(
PARTITION BY error_type ORDER BY time_stamp
MEASURES
       S.time_stamp AS START_T
       W.time_stamp AS END_T
       W.ipaddr AS IPADDR COUNT(*)
       AS CNT MATCH_NUMBER() AS
       MNO
ONE ROW PER MATCH
AFTER MATCH SKIP TO LAST W
PATTERN ( S W{2,} )
DEFINE
       S AS S.message LIKE '%authenticat%', W
       AS W.message = PREV (W.message) AND
       W.ipaddr = PREV (W.ipaddr)
) MR_S3
...
```

このシンプルな例を拡張して、発信元IPアドレスに関係なく、4回以上の連続的な認証失敗のログを検索できます。このような結果セットを使用すると、この上にさらにその他の分析機能を積み重ねて、より高いレベルの考察を得るのに役立ちます。IP識別サービスを使用すれば、空間地図を使用して、各インシデントの発生場所を表示できます。このため、セキュリティ・チームが結果に含まれる他の“パターン”を探すことができます。

結論

12cの新しいパターン・マッチング機能のリリースは、SQL開発者にもビジネス・ユーザーにも多大なメリットがあります。

SQL開発者は、PerlやJavaなどの言語によるパターン処理の柔軟性と、SQLの宣言的、分析的機能を組み合わせる方法を求めています。新しいMATCH_RECOGNIZE構文を使用すれば、データベース開発者の既存機能と統合できるSQLベースのデータベース内処理を利用して、アプリケーション・コードを簡素化できます。このような既存機能としては、たとえば、高度な処理が必要な場合に結果セットをリアルタイムにストリームするPL/SQL表の機能や、JavaやC++が開発言語としてよく使用されるOracle Data Cartridge Interfaceなどがあります。

このアプローチの場合、アプリケーションに組み込まれてクライアント側か中間層アプリケーション・サーバー内で実行されるカスタム・コードを使い続ける場合と比べて、パフォーマンス、管理性、スケーラビリティが大幅に向上します。また、開発者は独自のパターン・マッチング・ロジックをOracle Database 12cの新しいSQLベースのパターン・マッチング機能に切り替えることで、簡単なコードを使用でき、処理効率とパフォーマンスを上げることができます。

ビジネス・ユーザーとデータ・サイエンティストにとって、データベース内パターン・マッチングの主な利点は、豊富なパターンを既存の分析ワークフローに組み込めることです。このとき、ネットワーク・トラフィックやデータ待機時間の増加につながる複雑な外部コード・ルーチンを使用する必要はありません。

BIユーザーは、パターン・マッチングを初期問合せに組み込むことで、分析を簡単に積み重ねて、他の結果セットと同様に、一面/多面的にあらゆる方向から細かく分析できます。初期パターンが発見されれば、他の分析レイヤー（多次元、空間、データ・マイニングなど）を初期の結果セットに適用して、より深い考察を得ることができます。

全体的には、新しいデータベース内パターン・マッチングによってデータの複雑さ、データの移動、待機時間を減らすと同時に、データ・セキュリティの改善とパフォーマンスの大幅な向上を実現できます。SQLパターン・マッチングを使用すれば、Oracle Databaseテクノロジーへの投資をさらに最大限に活用できます。

Patterns everywhere – Find them fast!
Oracle Database 12cのSQLパターン・マッチング
2013年6月

著者：Keith Laker
Senior Principal Product Manager、
Data Warehousing and Big Data

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

海外からのお問い合わせ窓口：
電話：+1.650.506.7000
ファクシミリ：+1.650.506.7200

www.oracle.com

Oracle is committed to developing practices and products that help protect the environment

Copyright © 2013, Oracle and/or its affiliates.All rights reserved.
本文書は情報提供のみを目的として提供されており、ここに記載される内容は予告なく変更されることがあります。本文書は一切間違いがないことを保証するものではなく、さらに、口述による明示または法律による黙示を問わず、特定の目的に対する商品性もしくは適合性についての黙示的な保証を含み、いかなる他の保証や条件も提供するものではありません。オラクル社は本文書に関するいかなる法的責任も明確に否認し、本文書によって直接的または間接的に確立される契約義務はないものとします。本文書はオラクル社の書面による許可を前もって得ることなく、いかなる目的のためにも、電子または印刷を含むいかなる形式や手段によっても再作成または送信することはできません。

OracleおよびJavaはOracleおよびその子会社、関連会社の登録商標です。その他の名称はそれぞれの会社の商標です。

AMD、Opteron、AMDロゴおよびAMD Opteronロゴは、Advanced Micro Devicesの商標または登録商標です。IntelおよびIntel XeonはIntel Corporationの商標または登録商標です。すべてのSPARC商標はライセンスに基づいて使用されるSPARC International, Inc.の商標または登録商標です。UNIXはX/Open Company, Ltd.によってライセンス提供された登録商標です。0110

Hardware and Software, Engineered to Work Together