12

ORACLE DATABASE 12c

SQL – 分析用の自然言語

Oracleホワイト・ペーパー | 2015年6月

ORACLE

目次

概要	1
はじめに	1
高機能なフレームワーク	1
簡単な最適化	7
絶え間ない進化	9
標準ベースであること	10
Oracle SQLが選ばれる理由	11
結論	12
その他の参考資料	12

概要

分析は、すべての企業のデータウェアハウスやビッグ・データ・プロジェクトの必須コンポーネントです。ビッグ・データの早期導入企業の多くが、MapReduce、Spark、Impalaなどの専用ツールを使用して、蓄積したデータの分析に対処しています。このアプローチの場合、データが独自のデータ・サイロ内に固定されるため、機能間やデータ・ストア間の分析が非常に難しくなるか、完全に不可能になります。ITチームは、進化する新しい分析要件に対応するように、複雑なサイロ固有のプログラム・コードを適用させようと苦心しています。この結果、プロジェクト・コストが増大し、各プロジェクト内のリスク・レベルも上がっています。

多くの企業は、機能豊富で堅牢性と生産性に優れた、標準対応の単一言語を求めています。この言語は、あらゆる種類のデータに統合的にアクセスし、機能豊富で高度な分析を実行し、プロジェクト・コストの削減と全体的なリスクの軽減に役立つものである必要があります。

このホワイト・ペーパーの目的は、SQLが急速にデータ分析のデフォルト言語になってきている理由と、SQLがどのようにしてデータ・サイロの作成を回避しながら両方のデータ・アクセスの成熟した包括的なフレームワークを提供し、幅広い高度な分析機能をサポートしているかについて説明することです。

はじめに

過去40年間で、データ処理は多くの変化と飛躍的な技術的進化を遂げてきました。ただし、生き残って進化してきたのは、Structured Query Language（SQL）という1つのテクノロジー、機能だけです。他にも多くのテクノロジーが現れては消えて行きましたが、SQLは常に存続してきました。実際、SQLは常に不変だったわけではなく、時間をかけて大幅に改善もされています。

SQLのどのような部分がそれほど優れているのでしょうか。SQLがもっとも成功した言語である理由は何でしょうか。SQLの持続的な成功は、次のようないくつかの重要かつ固有の要因によるものです。

- 高機能なフレームワーク
- 透過的な最適化
- 絶え間ない進化
- 標準ベースであること

このホワイト・ペーパーの以降の項では、SQLがデータ分析にとって優れた言語である理由となっている上記の重要な要因について、それぞれ詳しく説明します。

高機能なフレームワーク

データを収集する理由は、これらのデータに対して問合せを行う必要があるためです。このような問合せプロセスは通常、1つのデータ・ポイントやデータ行ではなく、一連のデータをまとめて実行します。1970年には、E.F.CoddがIBM在籍時に、問合せの構築と実行を管理する数学的ルール・セットを確立しました。これらのルールは関係代数（集合論）として知られ、次の4つの主要な演算子で構成されています。

- 射影
- フィルタ
- 結合
- 集計

以降の項では、これらのトピックについて詳しく説明します。

1. 射影

リレーショナル・モデルでは、データは行と列に整理されます。データセットに対して問合せを行う場合はまず、データセット内の対象となる列を特定します。これらの列は、列セット全体の射影と見なされます。射影は、SQL問合せのフレームワークの最初の部分（SELECT句）で指定します。

以下の例では、このセットはEMPという表に保存されている従業員データです。このデータセットから、必要な列を'選択'できます。

```
SELECT ename, job, hiredate
FROM emp;
```

```
SELECT
  ename,
  job,
  hiredate
FROM emp;
```

SQLは、拡張射影という概念もサポートしています。これは、算術演算によって、結果セット内に新しいデータ列が作成されるというものです。たとえば、SAL列とCOMM列の場合、SALをCOMMで割って新しい列COMM_RATEを作成することで、コミッションの割合を計算できます。

```
SELECT
  ename,
  job,
  hiredate
FROM emp;
```

SELECT句に、1つのサブセットでなくすべての列を含めることもできます。簡単なSQL構文で、セット内のすべての列を選択できます。

```
SELECT * FROM emp;
```

Oracle Databaseではそのメタデータ・レイヤーを透過的に使用して、セットに関連付けられたすべての列名のリストを確立してから、問合せ文を内部的に拡張して、関連するすべての列名を含めます。

Oracle Databaseには、幅広いデータセットにわたる列の検出をサポートする包括的なメタデータ・レイヤーがあります。対象となるデータセットには、リレーショナル表/ビューだけでなく、XML文書、JSON文書、空間オブジェクト、画像スタイル・オブジェクト（BLOBやCLOB）、セマンティック・ネットワークなども含まれます。また、このメタデータ・モデルはOracle Database以外にも拡張されているため、他のリポジトリに保存されているデータ（NoSQLデータベース、JSON文書、HDFSに保存されているファイルなど）も対象となります。開発者は、この包括的なメタデータ・レイヤーを使用して、さまざまなデータ型やデータソースに基づく列射影を含むSQL文を非常に簡単に作成できます。

この高度なメタデータと文の自動拡張の組合せはSQLの中核となるものですが、多くのデータ分析言語には含まれていないため、開発者はカスタム・コードを追加して、このような基本要件に対応する必要があります。

2. フィルタ

問合せフレームワーク内の次の段階は、対象の行を指定することです。行は列と同じ方法（名前の使用など）では識別できないため、行ベースの射影には別のアプローチが使用されます。このアプローチでは、対象の行セットに関連付けられた属性を記述するフィルタを使用する必要があります。SQL言語内では、行フィルタリングはWHERE句構文の一部です（このようなフィルタはリレーショナル用語で条件と呼ばれる場合があります）。開発者は、結果セットに含める行を記述する具体的なフィルタリング条件を定義できます。たとえば従業員データセットを使用する場合、前の例を拡張して、問合せで返される行セットを'CLERK'という職種に関連付けられた行セットに限定できます。

```
SELECT
  ename,
  job,
  hiredate
FROM emp
WHERE job='CLERK';
```

SQL言語は、特定の行を識別するための、さまざまなフィルタリング比較演算子をサポートしています。たとえば、equal to、not equal to、greater than、greater than or equal to、less than、less than or equal toなどがあります。2つの値の間の範囲（これらの値を含む）を確認できます。たとえば、10,000～25,000ドルの給与の範囲に含まれるすべての従業員を検索できます。

```
SELECT
  ename,
  job,
  hiredate
FROM emp
WHERE sal BETWEEN 10000 AND 25000;
```

ワイルドカード記号を使った文字列検索や、特定の値リスト内での文字列検索がサポートされています。またnull[1]値の存在、およびそれらを含めるか除外するかも確認できます。これは非常に高度な機能であり、結合の概念と関連しています（次の項を参照）。

結果セットから特定のデータを除外することが必要な場合もあります。前の例を使用した場合、次のように非等価演算子!=を使用して、事務員（CLERK）を除くすべての職種を返すことができます。

```
SELECT
  ename,
  job,
  hiredate
FROM emp
WHERE job != 'CLERK';
```

[1] nullは、データ値が存在しないことを示すために使用されます。これは、関係代数に関するCoddのルール・セットの重要な要素です。これで、"見つからない"または"適用できない"データ・ポイントの概念がサポートされます。詳しくは、『SQL言語リファレンス』の「NULL」の項を参照してください。

上記の問合せによって、事務員以外の従業員のリストが取得されます。より複雑なビジネス要件をサポートするために、複数のフィルタが含まれる問合せを構築できます。AND演算子やOR演算子を使用してWHERE句を拡張できます。AND演算子を使用して2つの条件を組み合わせた場合、結果に含まれる行では、以下のように両方の条件がtrueと評価される必要があります。

```
SELECT
  ename,
  job,
  hiredate
FROM emp
WHERE sal BETWEEN 10000 AND
  25000 AND job != 'CLERK';
```

上記の例では、給与が10,000～25,000の範囲内で役職が'CLERK'でない行を含む結果セットが返されます。OR演算子を使用する場合、条件のうち1つがtrueを返す必要があるだけです。この問合せでは、給与が10,000～25,000の範囲内であるか、役職が'CLERK'でないすべての行が返されます。つまり、給与が10,000～25,000の範囲内の事務員は、結果セットに含まれます。

SQL言語には、非常に豊富なフィルタリング手法が含まれています。これらは実装が簡単で、時間の経過による要件の進化に合わせて修正できます。

3. 結合

ほとんどの問合せ操作では、2つ以上のデータセットを使用する必要があるため、"結合"演算子が必要です。簡単なレベルでは、結合を使用して、2つ以上のデータセット内のフィールドを、さまざまなデータセット間の共通の論理リレーションシップに基づいて1つの問合せで結合します。ここで説明する簡単なデータ例では、従業員データセットのほかに、部門データセットがあるものとします。問合せで部門データと従業員データを'結合'して、両方のデータセットのデータを組み合わせた1つの結果を返すことができます。データセットを結合する方法はいくつかあります。

- Inner - 両方の表で1つ以上一致するすべての行が返されます。
- Full Outer - 両方の表からすべての行が返されます。両側で一致する行がある場合は、それを含みます。一致しない場合は、存在しない側にnullが含まれます。
- Outer left - 左の表からすべての行と、右の表から一致する行が返されます。
- Outer right - 右の表からすべての行と、左の表から一致する行が返されます。
- Cross - ソース・セットのデカルト積（右の表の各行について、左の表からすべての行）が返されます。

SQL結合の定義と実行のプロセスは簡単です。必要な結合の種類は、FROM句のANSI規格の構文を使用して実装します（この構文については、後で説明します）。

```
SELECT
 d.deptno,
 d.dname,
 e.empno,
 e.ename
FROM dept d
 INNER JOIN
 emp e
 ON (e.deptno = d.deptno);
```

2つのデータセットの実際の結合プロセスを記述する文は、このSQL問合せには含まれません。ビッグ・データで使用される多くの手続き型言語では、2つのデータセットを結合するプロセスは複雑です。結合アルゴリズムとともに、各結合構造を明示的にコーディングする必要があるためです。問合せに追加するデータセットが増えるにつれて複雑さが増し、データセットを組み合わせるためにさまざまな結合操作が必要となります（データセット内で見つからない値を考慮するなど）。

他の結合パターンの考慮が必要になると、結合プロセスはすぐに複雑になります。たとえば、2つの表（EMPとDEPT）を結合する場合に、従業員のいない部門が存在する可能性があります。内部結合を使用すると、従業員のいない部門は返されません。問合せで、（従業員のいない'空の'部門を含む）すべての部門の従業員数を返す必要がある場合、次のように外部結合が必要です。

```
SELECT
 d.deptno,
 count(e.empno)
FROM dept d
 LEFT OUTER JOIN
 emp e
 ON (e.deptno = d.deptno)
GROUP BY d.deptno;
```

SQLの結合コードが、わかりやすくコーディングしやすいことは明白です。開発者はセマンティック結合条件を指定するだけで、後の処理詳細（結合順序など）はSQLエンジンが実行してくれます。

4. 集計

集計は、データセットの分析プロセスにおける重要な手順です。運用、戦略、検出のためのほとんどの問合せでは、詳細なレベルのデータの要約が使用されます。業界アナリストは多くの場合、すべてのレポートの最大90%に何らかの集計情報が含まれると言っています。

データセットに適用される集計には、簡単な計算、合計、移動平均や、標準偏差などの統計分析まで、さまざまな種類があります。このため、データを簡単かつ効率的に集計する機能は、あらゆる分析的データ言語の重要な要件です。

手続き型言語（JavaベースのMapReduceなど）の機能は、データセットを取得、集計、または削減して、サマリー結果セットを提供するには十分です。このアプローチは、基本的な計算が含まれている、ほとんどのシンプルなデータ検出方式の問合せに適しています。ただし、より複雑な集計要件が追加されると、計算の管理に必要なコード量はすぐに増えてしまいます。

SQLには、すべての行を1セットにまとめて簡単に処理できる豊富なデータ集計機能があります。たとえば次のように、すべての行を1列で合計できます。

```
SELECT SUM(sal) AS total_salary
FROM emp;
```

この問合せを拡張して、新しい要件（従業員数と平均給与の計算など）に簡単に対応できます。

```
SELECT
  COUNT(empno) AS no_of_employees,
  SUM(sal) AS total_salary,
  AVG(sal) As average_salary
FROM emp;
```

これらの例をさらに利用して、GROUP BY句を使用して行を特定のカテゴリに簡単にグループ化できます。

集計関数とGROUP BY句のグループを使用してデータをグループ化してから、次のような1つの問合せで、特定の集計関数を使用して従業員数をカウントし、給与を合計して、部門ごとの平均給与を計算できます。

```
SELECT
   deptno,
   COUNT(empno) AS no_of_employees,
   SUM(sal) AS total_salary,
   AVG(sal) AS average_salary
FROM emp
GROUP BY deptno;
```

ANSI SQL:2003規格（後で詳しく説明します）では、分析関数の概念を取り入れることで、データ集計プロセスが拡張されています。これらの関数では、データセットがパーティションと呼ばれる行グループに分けられるため、各パーティションと、各パーティション内の行で集計を計算できます。追加のキーワードを使用すると、average、sum、min、maxなどの分析関数の評価方法を、パーティションごとに定義できます。前の例を使用して、以下の文で分析レポートの合計（各部門の給与の合計、各部門の給与の移動平均、平均給与など）を作成します。

```
SELECT
    d.deptno,
    d.dname,
    e.ename,
    e.sal AS sal,
    SUM(e.sal) OVER (ORDER BY e.deptno)) AS dept_sal,
    ROUND(AVG(e.sal) OVER (PARTITION BY e.deptno ORDER BY e.empno)) AS
    moving_avg_sal,
    ROUND(AVG(e.sal) OVER (ORDER BY e.deptno)) AS avg_dept_sal

FROM dept d
    LEFT OUTER JOIN
    emp e
    ON (e.deptno = d.deptno);
```

移動平均を計算する上記のSQLコードと、ブログ"Simple Moving Average, Secondary Sort, and MapReduce"[2]に含まれる同等のMapReduceコード（GithubのCloudera による投稿）とで、簡単さを比べてみてください。この"Simple（簡単）な"例は、移動平均計算を実行し、ジョブ・パラメータを管理し、関連するワークフローのフレームワークを指定するための12個のJavaプログラム・ファイルで構成されています。この"簡単な"プロジェクトのコード量を考えると、ビジネス要件の進化に合わせて計算を変更するのは非常に難しいです。

[2] http://blog.cloudera.com/blog/2011/04/simple-moving-average-secondary-sort-and-mapreduce-part-3/

SQLには、結果セットの一部として返される列を使用して制限を適用する、追加機能があります。HAVING句では、SELECT句での計算やGROUP BY処理による集計に基づいて、結果がフィルタリングされます。これは、WHEREフィルタリング句の適用方法と同じです。たとえば、次のようになります。

```
SELECT
    deptno AS department,s
    COUNT(empno) AS no_employees,
    AVG(sal) AS avg_sal,
    SUM(sal) AS tot_sal
FROM emp
    GROUP BY deptno
    HAVING avg(sal) > 2500;
```

開発者やDBAはSQLを使用することで簡単、便利、かつ効率的なデータ集計手法を利用でき、その他の分析的プログラミング言語を使用する場合と比べて、プログラム・コードを大幅に削減できます。SQLが提供する簡潔性によって、アプリケーション・コードの構築、管理、保守や、新しいビジネス要件への対応を、より簡単かつ迅速に実行できます。

簡単な最適化

SQLは宣言的な言語です。この宣言的なアプローチのおかげで、開発者は実際の制御フローを記述することなく、計算ロジックを示すだけで、必要なプログラムの構造と要素を簡単に構築できます。つまり、必要な結果を実際に得るための"方法"を記述するのではなく、"何"をプログラムつまり問合せによって達成するのかを記述するということです。処理のこの"方法"の部分は、基盤となるプラットフォームによって透過的に管理されます。これは、プログラミングの手続き型アプローチとは対照的です。手続き型アプローチでは、特定のタスクを変数、プロシージャ、関数、対応するデータ構造、および条件演算子のコレクションに落とし込むことを目的としています。手続き型アプローチでは、プロシージャとサブルーチンにまとめられた命令リストが使用され、コンピュータに対してステップ・バイ・ステップで処理が指示されます。現在の多くのビッグ・データ言語では、この手続き型アプローチが採用されています。

ソートされた従業員リストを返す簡単な例では、SQLの宣言ベースのコードと、Javaなどのよりワークフロー指向の手続き型言語のおもな違いがよくわかります。

SQLの宣言的アプローチ

```
SELECT
  empno,
  ename,
  hiredate
FROM emp
ORDER BY empno, hiredate, ename
```

Javaの手続き型アプローチ[3]

```
public         class         Employee         implements
    Comparable<Employee>{ private int empno;
    private String ename;
    private int hiredate;

    public Employee(int empno, String name, int hiredate)
        { this.empno=empno;
        this.ename=ename;
        this.hiredate=hiredate;
    }

// Sort in descending order by Employee name.If name is same then sort in
descending order by age.
// If age is same then sort in descending order by Empid

    public int compareTo(Employee o){
                if (o.ename.equals(this.ename))
                        if (o.hiredate-this.hiredate == 0)
                                return o.empno - this.empno;
                        else return o.hiredate-this.hiredate;
                else return o.ename.compareTo(this.ename);
        }
      public int getHiredate(){
                return this.hiredate;
        }
      public int getEmpNo(){
                return this.empno;
        }
      public String getEname(){
                return this.ename;
        }
}

public class TestEmployeeSort {

        public static void main(String[] args)
                {       List       coll       =
                Util.getEmployees();
                Collections.sort(coll); // sort method
                printList(coll);
                System.out.println("---------------------");
                Collections.sort(coll,new SortEmployeeByName());
                printList(coll);
        }

        private static void printList(List<Employee> list)
                { System.out.println("EmpId\tName\tAge");
                for (Employee e: list) {
                        System.out.println(e.getEmpId() + "\t" +
e.getName() + "\t" + e.getAge());
                }
        }
}

public class SortEmployeeByName implements
        Comparator<Employee>{ public int compare(Employee o1,
        Employee o2){
                return o1.getName().compareTo(o2.getName());
        }
}
```

多くのビッグ・データ言語では手続き型アプローチが使用されており、開発者が最適化の手法を実装してワークロードをサポートします。開発者はそれぞれ担当するコード/プロセスを、全体的なワークロードから完全に切り離して最適化します。このように細かく制御することで柔軟性は大幅に向上しますが、類似した問合せを実行する複数のアプリケーションがある場合は、競合の原因となります。

ビッグ・データ・プロジェクトにおける最適化の手法は、初期の開発者中心の手法から、よりプラットフォーム中心のアプローチに移行しているところです。ビジネス・ユーザーやデータ・サイエンティストが必要とする分析が、ますます複雑化しているためです。

SQLなどの宣言的言語の場合、開発者はデータ・エンティティ間のリレーションシップ、最終出力、および出力のサポートに必要な変換を理解するだけで済みます。宣言的アプローチによって、よりコンパクトで大幅にシンプルな、保守や拡張が非常に簡単なコードを常に作成できます。宣言的コードでは、最終的な結果が示されるだけなので、データの処理方法の命令は含まれません。宣言的言語のこのような簡潔性が、SQLがビッグ・データ分析用の主要な言語になってきている理由の1つです。

[3] 比較演算子を使用した、Javaでのリストのソート方法の例

もちろん、SQLは簡潔性だけでなく拡張性やパフォーマンスにも優れています。Oracle Databaseには、透過的なSQL最適化手法が豊富に実装されています。オラクルの問合せオプティマイザはOracleデータベース・ソフトウェアの中核であり、

リクエストされたデータにSQL文でアクセスするためのもっとも効率的なメソッドを決定するものです。この問合せオプティマイザでは、特定のSQL文に最適な実行計画[4]が生成されます。オプティマイザでは、最適な計画の決定プロセス中に複数のアクセス・メソッド（全表スキャンや索引スキャンなど）が調べられます。また"最適な実行計画"を決定する前にさまざまな結合メソッド（ネステッド・ループやハッシュ結合など）が調べられます。データベース内には、多くの内部統計情報や内部ツールがあるため、

通常は開発者よりオプティマイザの方が、文の実行に最適なメソッドを決定しやすい状態にあります。

また、Oracle Databaseには高機能なSQLパラレル実行エンジンが搭載されているため、問合せでパラレル実行が可能であるかどうかやその方法を自律的に決定できます。パラレル実行の使用時期や選択する並列度の決定は、文のリソース要件と、問合せの送信時に使用可能なリソースの両方に基づきます。

Oracle Database Resource Manager（Oracle DBRM）を使用すると、リソースへのアクセスを制御するワークロードの優先順位をデータベース管理者が決定できます。このため、さまざまなジョブに対して、その優先順位に基づいてCPU時間を割り当てて、優先順位の高いタスクが優先順位の低い作業の影響を受けないようにすることができます。

データの配置先のソースでデータベースによる最適化が実行されるようにすると、幅広いフロントエンド・システム（BIレポート、ダッシュボード、アプリケーションなど）全体でチューニングの改善を共有できます。このとき各システムは、特に固有の問合せに対してだけでなく、データベース・システム全体のワークロード全般に対しても最適化されます。

絶え間ない進化

SQLは分析的言語として数十年間にわたり使用されてきました。SQLの中核は前述のように汎用のデータ処理手法に基づいていますが、SQLは新しい処理手法、新しい計算、および新しいデータ型へのアクセス機能によって拡張されてきたためです。オラクルは長年にわたり、Oracle Database内に高度なSQLベースの分析を組み込んできました。

ビッグ・データの分析における主要な分析機能となっているウィンドウ関数はOracle8i Database（1999年）に初めて導入され、複雑なビッグ・データ要件に対処するために多くの開発者によって使用されています。Oracle Database 10g（2003年）では、SQL Model句が導入されました。SQL Model句はビジネス・ユーザー向けのスプレッドシート・タイプのwhat-ifモデリング・フレームワークを備えており、データセット内で行と列の両方とやり取りして、特定のセット内に新しいレコードを作成できます。

Oracle Database 12cには、新しいビッグ・データ関連の分析関数がいくつか含まれており、これらの構造と構文は既存のSQL関数と共通です。

分析的要件の進化の好例は、近似処理回答に対するニーズです。場合によっては、分析的問合せで必要となる精度のレベルを、下げることができます。つまり、必要十分なレベルで完璧な回答になります。たとえば、実際の値の1%以内となる近似処理回答で十分な場合があります。オラクルは、SQLのHyperLogLogアルゴリズム（'重複を除く概算カウント'操作用）を実装することで、Database 12cの最新リリースにこの機能を追加しました。

[4] "実行計画"には、SQL文の実行の推奨メソッドが記述されます。計画ごとに、SQL文の実行時にOracle Databaseで使用する必要がある手順の組み合わせが詳しく記述されます。

SQLは、分析のサポートに必要なデータ型の範囲の増加に合わせて拡張され、より幅広いデータソース（リレーショナル表/ビュー、XML文書、JSON文書、Hive表、HDFSファイル、空間オブジェクト、画像スタイル・オブジェクト（BLOBやCLOB）、セマンティック・ネットワークなど）がサポートされるようになりました。このため、プロジェクト・チームがさまざまなデータ型やデータソースにわたって機能豊富な単一の分析的言語を使用でき、全体的な複雑さやリスクが軽減されるため、機能拡張や新しい要件をさらに簡単に組み込めるようになりました。

標準ベースであること

ビッグ・データ・プロジェクトの多くの課題の1つは、どの分析的言語を開発者が使用できるかを決定することです。最新レポートによると、ある大手コンサルタント組織[5]が、分析標準についてプロジェクト・チームで真剣に検討する必要があると述べています。

このようなオープン・ソースのビッグ・データ言語の多くが、1セットの業界標準で関連付けられていないため、複数のデータ保管場所にわたって問合せを統合する場合に課題が生じています。各言語が独自の構文、分析関数セット、データ型をサポートしています。このような言語間の一貫性の欠如のために、プラットフォーム間で大規模なデータ移動が必要となることがよくあります。これは、一部のソース・プラットフォームに特定の分析関数が含まれないためです。名前付け、構文、処理での一貫した表記規則がないために、開発者やビジネス・ユーザーにとっては機能が使いにくくなっています。

現在、業界標準に従わなかった場合に生じる影響について、わかりやすく的確な例があります。初期のビッグ・データ・プロジェクトでは、MapReduceの2段階の手続き型フレームワークが使用され、コーディングに多大なコストがかかっていました。これは今では、パフォーマンス向上を目的とし、Apache Sparkプロジェクトに取って代わられています。ただし、Sparkのフレームワークは基本的に別のものであるため、プロジェクト・チームは既存のMapReduceコードを書き直す必要があります。

新しいビッグ・データ・プラットフォームにおける新しい処理言語の進化は、SQLの進化とは完全に対照的です。SQLは、1986年に米国規格協会（ANSI）の標準となり、それ以来最新版のANSI 2011まで進化してきました。この標準化には、おもに2つのメリットがあります。

まず、標準によって、さまざまなデータベース・システムにわたり、大きなコード変更なしでアプリケーションを移植しやすくなります。データウェアハウスの分野では、BIツールによって、複数の種類のSQLデータベースを簡単かつ効率的にサポートできます。

次に、SQL標準によって、アプリケーション開発の継続性を確保できます。30年前に記述されたSQL文は、SQLコードの変更なしで今でも実行できます。現在のSQL文のおもな違いは、SQLの実行速度が大幅に向上していることです。データベースでSQL文が透過的に最適化され、最新のパフォーマンス向上機能が使用されるためです。

一部のビッグ・データ言語では、SQLにより似るように試みられており、ANSI規格の特定の要素が採用されています。これらの言語の多くでは、SQL 92標準の一部または全部が採用されています。ただし、ほとんどのプロジェクト・チームが求めるレベルの移植性はまだ提供されていません。この好例がHiveです。HiveはImpalaなどの新しいSQL実装への置換えが進んでいます。HiveもImpalaも、以前のバージョンのANSI規格にある程度準拠していると言われていますが、Impalaでは多くのもっとも一般的なHiveQL機能（集計関数、ラテラル・ビュー、問合せあたり複数のDISTINCT句など）がサポートされていません。このような違いのために、アプリケーション・ロジックの再コーディングに多大なリソースを投資せずに、プロジェクト間でコードを移行するのは非常に難しいです。標準に準拠しないと、プロジェクト内の全体的なリスク・レベルが上がります。

[5] Deloitte Analytics：Analytics Trends 2015：A Below-the-Surface Look

SQL 92標準の発行以来、より高度な分析をサポートするために、非常に多くのデータウェアハウス機能が追加されてきました。SQL 99では多次元集計機能（ROLLUP、CUBE、GROUPING SETSなど）が、SQL 2003では分析ウィンドウ関数とOLAP機能が、SQL 2011では一時的なデータ型と分析のサポートが追加されました。

業界標準のSQLを実装するデータベースは、顧客にとってメリットがあります。業界標準のSQLを使用すると、分析プロセスを最大限に再利用および共有でき、これらのプロセスが古くなるリスクを最小限に抑えられるためです。

Oracle SQLが選ばれる理由

オラクルの分析的SQLは、ANSIのSQL 2011標準を完全にサポートしており、次バージョンの標準に組み込まれる予定の新しい機能（SQLパターン・マッチングや重複を除く概算カウントなど）も含まれています。このため、これらの機能を幅広くサポートし、オラクルのパートナー・ネットワークやその他の独立系ソフトウェア・ベンダーからのアプリケーションやツール全般で、新しく導入された機能を迅速に採用できます。オラクルは、多くのパートナーと継続的に連携し、拡大する分析関数ライブラリをパートナーが活用できるように支援しています。

すでに多くの独立系ソフトウェア・ベンダーが、Database 12cの新しいデータベース内分析関数のサポートを自社製品に統合しています。

Oracle Exadataプラットフォームのリリースによって、既存のSQLコードを変更せずに移行して実行しても（1980年代に作成されたコードであっても）、パフォーマンスが大幅に向上します。Oracle Exadata Storage Serverに組み込まれているストレージ・レベルのパフォーマンス最適化機能（ストレージ索引、Hybrid Columnar Compression、Smart Scanなど）は、SQLコードに対して完全に透過的です。このため、このプラットフォームではすべてのアプリケーションとSQL文を変更せずに実行でき、最適化されたエンジニアド・プラットフォームで実行するメリットを享受できます。

オラクルはその継続的投資の一環として、SQL問合せのパフォーマンスをさらに高めるために、インメモリ処理をリリースしました。インメモリ・データ構造の管理とライフ・サイクル、および操作の種類（OLTP中心なのか分析中心なのか）に応じた各種の圧縮の使用は、問合せに対して透過的です。開発者やDBAが、この機能を利用するためにアプリケーション・コードを書き直す必要はありません。Exadataと同様に、既存のすべての問合せでこれらの利点を自動的に継承できます。

Oracle SQLによって、プロジェクト・チームは機能豊富な単一の分析的言語で作業できるため、全体的な複雑さを軽減し、データ・サイロの作成を防ぐことができます。またITチームは、機能拡張やビジネス・チームからの新しい要件を簡単に組み込むことができます。

結論

SQLは事実上、データ分析用の言語として登場しました。これは単にSQLが技術的に優れており、この言語が提供するメリットが大きいからです。

詳細データの収集量が増え、アプリケーションやダッシュボードをリアルタイムに最新情報で更新する必要があり、ビジネス要件が分単位で絶え間なく進化する状況において、ITチームは新しい課題に直面しています。ITチームはこれらの課題を解決するために、データ分析用に柔軟性の高い高機能なフレームワークを必要としています。

SQLはその柔軟性と機能性によって、すべてのデータ分析プロジェクトの必須ツールとなっており、SQLを分析用の主要言語として使用するITチームは、かつてない勢いで増えています。これは、SQLの次のような重要かつ独自の要素によるものです。

- 高機能なフレームワークを提供している
- 処理の最適化が完全に透過的である
- 現在の厳しい要件に適合できるよう、絶え間なく進化している
- 国際的なANSI規格に準拠している

すでに多くの企業が、OracleとSQLを使用して、俊敏性の高い開発エコシステムの一部として、あらゆるデータセットで高度な分析を実行しています。これはOracle SQLが、データ・サイロの作成を防ぐデータ・アクセス機能と、特殊な処理エンジンにデータを渡さなくて済む豊富な分析機能の両方に対応する、成熟した包括的なフレームワークを提供するためです。

ビジネス・ユーザーやビッグ・データ開発者は、Oracle SQLを使用することで、非常に複雑なデータ検出要件やビジネス・インテリジェンス・レポートの要件を簡単にサポートできます。Oracle SQLはANSI 2011などの最新の業界標準、業界をリードする最適化をサポートしており、絶え間ない技術革新をコミットしているため、あらゆるデータ型やデータソースに対応するデフォルトの分析言語となっています。

つまり、Oracle SQLを使用することで、プロジェクト・チームは全体的なプロジェクト・コストを削減し、リスクを軽減できます。Oracle SQLを使用して分析を行うことで、製品のイノベーションを検討し、新しい市場を見いだすことができます。またもっとも重要なのは、競争上大きなメリットを得られる可能性があるということです。

その他の参考資料

Oracle Databaseの以下の機能に関する参考資料です。

1. Oracle Database SQL言語リファレンス - Oracleと標準SQL
2. Oracle SQLの分析機能と分析関数 - SQLを使用してアクセスできる、一連の優れた分析機能と分析関数
3. Oracle Statistical Functions - 外部システムに移行やステージングしなくても、統計的分析を実行
4. Oracle Database 12c問合せ最適化 - 計画の実行と安定性での進化を実現
5. Oracle Big Data SQL - 1つの高速な問合せで、すべてのデータを処理
6. Oracle Database In-Memory - リアルタイムな分析、ビジネス・インテリジェンス、レポートをサポート

次のオラクル・ホワイト・ペーパーとデータ・シートは必読です。

1. 『Effective Resource Management Using Oracle Database Resource Manager』
2. 『Best Practices for Gathering Optimizer Statistics with Oracle Database 12c』
3. 『Parallel Execution and Workload Management for an Operational Data Warehouse』
4. 『Oracle Database 12cの管理性』
5. 『Best Practices for Workload Management of a Data Warehouse on the Oracle Exadata Database Machine』

上記のホワイト・ペーパーへのリンクやその他の情報については、Oracle Technology NetworkのWebページ"Oracle Data Warehousing Best Practice"を参照してください。

http://www.oracle.com/technetwork/jp/database/focus-areas/bi-datawarehousing/dbbi-tech-info-best-prac-154753-ja.html

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

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

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together

Copyright © 2015, 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、ここに記載されてい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。

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。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。UNIXは、The Open Groupの登録商標です。0215