ORACLE

ビジネス / 技術概要

Oracle DatabaseにおけるJSON：パフォーマンスに関する考慮事項

オンプレミスおよびクラウド（Autonomous Databaseを含む）、
SODA（Simple Oracle Document Access API）、
Oracle Database API for MongoDB

2022年11月、バージョン2.0

公開

ORACLE

目次

目的

本書は、Oracle Database内で保管および処理されるJavaScript Object Notation（JSON）をパフォーマンス・チューニングする際のベスト・プラクティスを説明したものです。開発者、DBA、アーキテクトは、ここに記載したベスト・プラクティスを適用することで、パフォーマンスの問題を事前予防的に回避し、設計するアプリケーションとシステムのピーク性能での動作を実現できます。

本書全体で使用しているハイパーリンクから、ドキュメント、追加情報、例、無償のハンズオン・トレーニングにアクセスできます。

JSON Developer Guide：

19c、21c

Oracle DatabaseにおけるJSONの概要

2014年にリリースされたOracle 12.1.0.2で、すべてのOracle Databaseエディションにネイティブ JSONサポートが追加されました。このリリースより前、JSONはNoSQLデータベースに格納されることが多かったため、機能性とデータ整合性モデルの欠如により、開発者はデータ整合性を確保するための追加コードを作成する必要がありました。

NoSQLの弱点を補うために、たとえば分析問合せを実行する場合は、リレーショナル・データベースまたはその他のデータ・ストレージ技術を追加で使用していました。2014年にネイティブJSONサポートが追加されたことで、こういった専用のデータ・ストレージ技術を追加する必要がなくなり、統合作業の解消、デプロイメントの簡素化、リスクとコストの低減により、大幅に迅速な開発が可能になっています。さらに、標準SQL演算子を使用してJSONを格納、処理、分析できることで、大幅に受け入れ時間が短くなり、必要なスキルも少ないため、開発者以外でも簡単にJSONデータを扱えるようになります。

Oracle DatabaseではネイティブJSONサポートが提供されています。各種オプション、Oracle管理パック、フレームワーク、アーキテクチャ、セキュリティを含むすべてのOracle Database機能でJSONを使用できます。また、Oracleデータベースに格納されたJSONは、Oracle Databaseのパフォーマンス、スケーラビリティ、可用性、拡張性、移植性、セキュリティを活用できます。Oracle Database内のJSONにアクセスする方法は、その他のデータベース・アクセス方法と同じで、OCI、.NET、JDBCを含みます。

Oracle DatabaseにおけるJSONに関する追加情報については、『JSON Developer's Guide』を参照してください。

パフォーマンス関連の機能とテクニック - 詳細

ここに示す機能については、ワークロードのセクションでより詳しく説明しています。

Oracle DatabaseにJSONデータを格納する際のパフォーマンス最適化

JSONを格納できるのは、データ型がVARCHAR2、CLOB、BLOB、またはJSONの列です。どのデータ型を使用する場合も、そのデータ型を持つその他のデータと同じように操作できます。

- Oracle 21cでは、問合せと効率的な（部分）更新が最適化された、ネイティブJSONデータ型の使用が推奨されています。JSON列に「IS JSON」チェック制約を定義すると、正しいJSON構文の使用を徹底できます。アプリケーションでJSONの正確性が保証されている場合は、この制約を無効化（削除ではない）できます。
- Oracle 19cでは、ネイティブBLOBデータ型の使用が推奨されています。BLOBでも問合せおよび効率的な更新が最適化されています。

- CLOBもサポートされていますが、エンコーディングがUCS2であり、通常2倍のストレージ容量（およびディスク読取り）が必要になるため、推奨されていません。
- VARCHAR2フィールドもサポートされています。JSONドキュメントの最大サイズが既知である、JSONドキュメントがすでにVARCHAR2フィールドに格納されている、VARCHAR2使用時の単純さが好まれている、といった場合は検討対象になります。VARCHAR2には最大32バイトの値を格納できます。

ワークロード・タイプとデータ・アクセス・パターン

データベース・ワークロードは、業務ワークロードと分析ワークロードに分類できます。オンライン・トランザクション処理システム（OLTP）とも呼ばれる業務ワークロードは、多くのユーザーを持つトランザクション指向で、即時応答（銀行のATMなど）を実現するように設計されています。OLTPシステムは、すべてのデータ操作タイプに対応しています。代表的な操作には、最小限の行を使用してデータを挿入または更新するトランザクションがあります。OLTPシステムのパフォーマンス目標は、トランザクション速度、スループット、データベースの同時実行性です。対照的に、オンライン分析処理（OLAP）、データウェアハウス、データ・レイクなどの分析ワークロードは、ユーザー数の少ないデータ分析向けであり、大量データを処理するように設計されています。代表的な操作では、多くのリソースを消費する複雑な問合せを使用して何百万もの行を処理し、多数の表をまたぐ結合やデータ集計を行います。OLAPシステムは問合せ向けに最適化されています。

「JSONのネイティブ・サポートは重要です。これまでは、専用のDBMSでより効率的なJSON管理を行うか、JSONデータと他のデータ（リレーショナル・データなど）を統合できるようにするかのどちらかを選択する必要がありました。しかし現在は、そのような選択を行う必要はありません。Oracle Databaseには、JSONの効率性と統合データ管理の両方が組み込まれているためです」

IDC、Carl W Olofson氏

bit.ly/nativeJSON_IDC

キーによるJSONドキュメントの取得（OLTP）

このケースのワークロードは、リレーショナル列（キー）に基づいて個別のJSONドキュメントを選択します。JSONデータは2番目の（ペイロード）列に格納されています。キー列に主キー制約を指定すると、一意のキー値が保証されるとともに、索引が作成されて検索が高速になります。キーがランダムではない場合（シーケンスまたはIdentity列を使用する場合など）、同時/後続の挿入で同じ索引ブロックが処理対象になるため、トランザクションの多いシステムでは索引がホット・スポットになる可能性があります。キー列の索引を**ハッシュ・パーティション化**すると、すべてのパーティション間で挿入が均等に分散されます。SODAコレクションおよびMongoDBコレクションには自動的に主キー列が設定されるため、キーベースでドキュメント検索するための追加作業は不要です。

フィールド値によるJSONドキュメントの取得（OLTP）

このケースでは、JSONドキュメント内のフィールド値によって、1つまたは複数のドキュメントが選択されます。JSON_VALUEまたはJSON_EXISTS演算子内のパス式により、値が定義されます。同じパス式が繰り返し使用される場合は、JSON_VALUEを使用した**ファンクションベースの索引**を推奨します。処理対象のフィールド値に索引を付けることで、データ取得時の全表スキャンが索引検索に置き換わるため、可能な限り最大のパフォーマンスが得られます。

JSONドキュメント内の1つのフィールドに索引を作成するのは容易ですが、配列の索引付けは難度が高くなります。ファンクションベースの索引を配列値に対して作成することはできませんが（JSONデータごとにファンクションから返される値は1つのみ）、Oracle 21cより前のリリースでは、代わりに**マテリアライズド・ビュー**を使用できます。マテリアライズド・ビューは、配列を展開して複数の行エントリを持つリレーショナル列に変え、通常の列としてこれらに索引を付けます。オラクルの包括的なクエリー・リライト・フレームワークは、JSONドキュメントを対象としたSQL文を自動的に書き換えて、マテリアライズド・ビューを使用することで高速なデータ取得を可能にします。Oracle 21cでは、このリリースで導入された新しい**複数値索引**機能を使用すると、JSON配列の値にネイティブで索引を作成できます。

全文検索によるJSONドキュメントの取得（OLTP、OLAP）

ワークロードによっては、その場限りの問合せや不定のドキュメントなど、JSONドキュメント内のフィールドに対するパス式が不明で、対象の値しかわからない場合があります。このようなワークロードのパフォーマンスを向上するために、**JSON検索索引**が提供されています。JSON検索索引では、SQL/JSON演算子であるJSON_TEXTCONTAINSを使用し、ワード・ステミングやファジー検索を含むテキスト検索条件に基づいて行を選択することができます。

レポートおよび分析のためのJSON値の抽出（OLAP）

レポートまたは分析ユースケースでは、JSONデータをリレーショナル・モデルにマッピングし、その後の処理でSQLを使用できます。よく使用されるSQL操作には、結合（その他のJSONデータまたはリレーショナル・データと）、集計（合計、平均、ウィンドウ関数）、機械学習（分類、予測）があります。SQL/JSON演算子のJSON_TABLEを使用すると、JSONからリレーショナル・モデルへのマッピングを実行できます。可能な場合、Oracle Databaseは複数のJSON問合せ演算子を最適化して、1つのJSON_TABLE文に変換します（問合せの実行計画に表示されます）。

選択性の高い分析（フィールド値のフィルタ条件によってごく少数のJSONドキュメントだけが選択される）では、**索引**を使用してアクセスを最適化できます。アクセスされる行が多数で（ただし、すべての行ではない）、索引を使用しても十分少数に限定されない場合は、データを**パーティション化**して無関係なパーティションを問合せから除外することを検討してください。また、大量データを処理する場合は常に**パラレル実行**を活用することを推奨します。SQL/JSON演算子のJSON_TABLEは、制限なしで並列化できます。

日次レポートやダッシュボードの問合せなどで、JSONからリレーショナルへの同じ変換を繰り返し実行する場合があるとします。そのような場合は、**マテリアライズド・ビュー**を使用し、中間結果からマテリアライズド・ビューを作成すると、同じJSON_TABLEの変換を実行時に繰り返し実行せずに済みます。JSON_TABLEのマテリアライズド・ビューは高速リフレッシュが可能なので、挿入または更新後、効率的かつ自動的にリフレッシュされます。また、マテリアライズド・ビューを**Oracle Database In-Memory**と一緒に使用すると、インメモリ列圧縮と高速SIMDスキャンを利用できます。これにより、特に分析問合せのパフォーマンスが大幅に向上します。

JSONの生成（OLTP、OLAP）

Oracleデータベースには、リレーショナル・データや問合せ結果から新しいJSONデータを生成するためのSQL/JSON演算子が追加されています。一般的なユースケースとしては、特定のJSONドキュメントの形式を変更するか、分析問合せの結果をJSONデータ抽出として返す使用法があります。アクセスする行が非常に少ない場合は、索引により高速アクセスを実現できます。JSONが多数の行に基づいて生成される場合、**マテリアライズド・ビュー**の検討を推奨します。ただし、高速リフレッシュは限られたJSON生成ケースのみでサポートされる点に注意してください。

パフォーマンス関連の機能とテクニック - 詳細

ここからは、パフォーマンス関連について例を示しながら詳しく説明します。全般的に、**通常のSQLチューニング・テクニックが当てはまり**、既存のスキルを活用できます。このため、習熟期間は短く、Oracle DatabaseでのJSON採用に関してDBAやデータベース・マネージャーが抱く不安も解消されます。チューニング・テクニックの背景にあるおもな目的は、読取りと処理が必要なデータを減らすことです。

ファンクションベースの索引

ファンクションベースの索引は、特定のキーまたはキーの組合せに対して作成でき、同じキーに対してSQL/JSON演算子を使用する問合せ操作を最適化します。ファンクションベースの索引はJSON_VALUE演算子を使用して構築され、ビットマップ索引とBツリー索引の両方の形式をサポートします。

次の例は、サンプルJSONドキュメントのPONumberキーに対して、'$.PONumber'パス式によってアクセスされる（一意の）ファンクション索引を作成します。この例は、表'purchaseorder'の列'data'にJSONデータが格納されていることを前提としています。

```
create unique index PO_NUMBER_IDX on PURCHASEORDER po(
   json_value(po.DATA, '$.PONumber' returning number
                                          null on empty error on error));
```

ファンクション索引19c、21c

PONumberの値は数値として抽出（および索引付け）されます。これにより範囲問合せに影響があり（アルファベット順ではなく数値順になる）、数学的演算や比較を行う際、実行時のデータ型変換が回避されます。値が欠如している場合は、SQL NULL値として索引付けされます。

次の問合せでは、簡素化したJSON構文を使用しています。実行計画に表示されているとおり、'number()'アイテム・メソッドによって、データ取得にこの索引が使用されています。

```
select data from PURCHASEORDER p
o where po.data.PONumber.number() = 200;
-------------------------------------------------------------------------------
| Id  | Operation                            | Name             |
-------------------------------------------------------------------------------
|   0 | SELET STATEMENT                      |                  |
|   1 |   TABLE ACCESS BY INDEX ROWID        | PURCHASEORDER    |
| *  2 |     INDEX UNIQUE SCAN                | PO_NUMBER_IDX    |
-------------------------------------------------------------------------------
```

「Yahoo! Cloud System Benchmarkに基づくJSONデータベースの独自ベンチマークにより、オラクルが（中略）すべての競合他社を上回る圧倒的なリーダーであることが明らかになりました」

Accenture技術レポート：

『Increase agility and cut development time with JSON and Oracle』（2021年版）
https://accntu.re/3lezy00

複数値索引

複数値索引は、パス式で選択される値が複数になる場合に推奨されます。これは、JSON配列内の値にアクセスするときには一般的なケースです。次の例では、サンプルJSONドキュメントのJSON配列'LineItems'に含まれるフィールド'UPCCode'に複数値索引を作成しています。値は文字列として索引付けされます。

```
create multivalue index UPCCODE_INDEX on PURCHASEORDER po(
po.data.LineItems.Part.UPCCode.string());
```

JSON複数値索引 21c

複数値索引ではBツリーも使用しますが、生成されるROWIDの重複排除が必要になるため、ファンクション索引よりも若干パフォーマンスが低下します。このため、パス式で返される値が1つ以下になるとわかっている場合は、ファンクションベースの索引を推奨します。複数値索引はOracle 21cで導入されました（上述の理由から、配列アクセスを高速にするためにマテリアライズド・ビューを使用できます）。次の問合せでは、複数値索引を使用しています。

```
select data from PURCHASEORDER po where
po.data.LineItems.Part.UPCCode.string() = '13131092705';

-------------------------------------------------------------------------------
| Id  | Operation                            | Name             |
-------------------------------------------------------------------------------
|   0 | SELET STATEMENT                      |                  |
|   1 |   TABLE ACCESS BY INDEX ROWID BATCHED| PURCHASEORDER    |
| *  2 |     INDEX RANGE SCAN (MULTI VALUE)   | UPCCODE_INDEX    |
-------------------------------------------------------------------------------
```

JSON検索索引

JSON検索索引：19c、21c
JSON検索索引ブログ
JSONデータ・ガイド：19c、21c

Oracle Databaseでは、検索索引を使用したJSONドキュメント全体への索引付けがサポートされています。検索索引は、オラクルの全文索引をベースとしています。検索索引は、すべての値だけでなくフィールド名も取り込んで、全文検索を可能にします。次の例は、'purchaseorder'に対するJSON検索索引を作成しています。

```
create search index PO_FULL_IDX on PURCHASEORDER po (po.data) for json
parameters('SYNC (EVERY "FREQ=SECONDLY; INTERVAL=1") DATAGUIDE OFF');
```

'parameters'句により、この索引が非同期で、毎秒同期されることが指定されています。トランザクションがコミットされるたびに索引を同期することもできますが、索引のメンテナンス・コストが大きくなり、同時実行DMLのスループットが低下します。JSON検索索引により、DML操作中のスキーマ変更を検出することもできます。*JSON*データ・ガイドと呼ばれる機能を使用すると、たとえば、JSON_Tableビューを自動生成できます。'DATAGUIDE OFF'句は、このスキーマ検出を無効にするため、DML操作中のJSON検索索引のコストが小さくなります。

JSON検索索引の基盤となるデータ構造はポスティング・リストで、通常はBツリー索引よりも遅くなります。JSON検索索引をファンクションベースの索引または複数値索引と併用した場合、可能な限り、ファンクションベースの索引または複数値索引がオプティマイザによって優先されます。これは、JSON索引検索がJSONデータ全体に索引付けするため、索引サイズがその他の索引よりも大幅に大きくなるからです。一般に、サイズは元のデータの20 %～30 %になります。JSON検索索引は、JSON配列内の値と全文検索操作をサポートしています。次の例は、'Description'フィールドに'Magic'と'Christmas'の両方の単語が含まれるドキュメントをすべて選択します。必要に応じて、'{and}'の代わりに'{near}'または'{not(...)}'も使用できます。JSON検索索引の詳しい機能については、該当するドキュメントを参照してください。

```
select data from PURCHASEORDER po
where JSON_TEXTCONTAINS(po.data, '$.LineItems.Part.Description', 'Magic
{and} Christmas');
```

問合せの実行計画に、JSON検索索引'Domain Index'が表示されています。

```
-----------------------------------------------------------------------------
| Id  | Operation                     | Name          |
-----------------------------------------------------------------------------
|   0 | SELET STATEMENT               |               |
|   1 |   TABLE ACCESS BY INDEX ROWID | PURCHASEORDER |
| * 2 |     DOMAIN INDEX              | PO_FULL_IDX   |
-----------------------------------------------------------------------------
```

DML操作の多いワークロードでは、多数のファンクション索引および複数値索引を使用するよりも1つのJSON検索索引を使用する方が、索引メンテナンス（DML操作後の索引の同期）が少なくなるため、有益な場合があります。その他の最適化戦略については、右側に記載した参照先のブログを参照してください。

マテリアライズド・ビュー

JSON_Table
マテリアライズド・ビュー：19c、21c

マテリアライズド・ビューを使用すると、多数行にアクセスする高頻度の問合せのパフォーマンスを上げることができます（索引駆動のキーベース検索を除く）。マテリアライズド・ビューは問合せの結果を永続化します。このため、後続の問合せが、マテリアライズド・ビューの問合せと部分的（または全体的）に一致する場合、元の問合せを再実行しなくてもマテリアライズド・ビューのデータにアクセスできます（スペースとスピードのトレードオフ）。

本書では、おもにJSON_TABLEマテリアライズド・ビューに重点を置いて説明します。次の例で作成するマテリアライズド・ビューには、サンプルJSONドキュメントの'LineItems'配列内の値が含まれています。前述したように、マテリアライズド・ビューを使用すると、複数値のJSON索引を使用できないOracle 19cでも、JSONの配列値に索引を作成できます。

```
create materialized view PO_MV build immediate
  refresh fast on statement with primary key as
    select po.id, jt.*
      from PURCHASEORDER po,
           json_table(po.data, '$' error on error null on empty
           columns (
                po_number          NUMBER           PATH '$.PONumber',
                userid             VARCHAR2(10)     PATH '$.User',
                NESTED                              PATH '$.LineItems[*]'
                columns (
                     itemno          NUMBER          PATH '$.ItemNumber',
                     description     VARCHAR2(256)   PATH '$.Part.Description',
                     upc_code        NUMBER          PATH '$Part.UPCCode',
                     quantity        NUMBER          PATH '$.Quantity',
                     unitprice       NUMBER          PATH '$.Part.UnitPrice')))
jt;
```

配列の値がマテリアライズド・ビュー内の複数の行に変換されるため、次のように、JSON配列のフィールドに追加（2番目）の索引を作成できます。

```
CREATE INDEX mv_idx ON PO_MV(upc_code, quantity);
```

ベース表に対するSQL/JSON問合せは透過的に書き換えられ、可能な場合は常にマテリアライズド・ビューとその索引が使用されます。次の問合せは、Oracleデータベースが自動的に問合せを書き換えている例であり、実行計画からわかるとおり、マテリアライズド・ビューとその2番目の索引が使用されています。

```
select data from PURCHASEORDER po
where JSON_EXISTS(po.data, '$.LineItems[*]?(@.Part.UPCCode == 1234)');
-------------------------------------------------------------------------------------------
| Id  | Operation                                | Name       |
•••
|   4 |    MAT_VIEW ACCESS BY INDEX ROWID BATCHED | PO_MV      |
|*  5 |      INDEX RANGE SCAN                    | MV_IDX     |
-------------------------------------------------------------------------------------------
```

マテリアライズド・ビュー（MV）と基盤データの同期状態を維持するため、マテリアライズド・ビューは'on statement'（DML更新後即時）での高速リフレッシュが可能になっています。これにより、リフレッシュ・プロセスが自動化され、マテリアライズド・ビューと元表データの整合性が常に維持されます。マテリアライズド・ビューの各種リフレッシュ・メカニズムの詳しい説明は、本書の対象外です。詳細については、該当するドキュメントを参照してください。

「Oracle Autonomous Databaseは、分析負荷からトランザクション負荷まで、企業の重要なデータベース負荷をすべて自律的に実行できることに加え、ML、グラフ、IoT、JSONなどをサポートしており、現在のデータベースの市場において他の製品とは一線を画しています。それぞれが独自のセキュリティ・プロファイルと管理学習曲線を持つ9つの専用データベースと、あらゆる種類のデータセットを使用して自律的に稼働する単一のデータベースのどちらを所有したいですか」

Constellation、Holger Mueller氏
bit.ly/ADB_Constellation

Oracle Partitioning

JSONのパーティション化：19c、21c
パーティション化の概念：19c、21c

ドキュメントを含む表を、通常と同じようにパーティション化してパフォーマンスを上げることができます。パーティション化により、表と索引を、パーティションと呼ばれる個別の小さな物理オブジェクトに分割できます。パーティション化された表内のデータ位置は、パーティション化キーによって識別されます。このキーは、リレーショナル列またはJSONデータのフィールドに指定できます。アプリケーションから見ると、パーティション表と非パーティション表はまったく同じです。

次の例は、JSON_VALUEに基づく仮想列'po_num_vc'を使用して、列'data'に格納されたJSONドキュメントから抽出したパーティション・キーを持つレンジ・パーティション表を作成しています。

```
CREATE TABLE part_j (id VARCHAR2 (32) NOT NULL PRIMARY KEY,
                                 data JSON,
                                 po_num_vc NUMBER GENERATED ALWAYS AS
                                (json_value (data, '$.PONumber' RETURNING NUMBER)))
PARTITION BY RANGE (po_num_vc)
    (PARTITION p1 VALUES LESS THAN (1000),
    PARTITION p2 VALUES LESS THAN (2000));
```

問合せをJSONフィールド'$.PONumber'（仮想列のパーティション化キーとして使用されるJSONフィールド）でフィルタリングすることで、透過的にOracle Partitioningのメリットを活用できます。パーティション・プルーニングと呼ばれる最適化テクニックにより、無関係なパーティション（問合せに関係するデータを含まないと判明しているパーティション）が自動的に除外されます。

次のサンプル問合せでは、問合せの等価条件で一致レコードが見つかるのは最初のパーティション内だけなので、アクセスが必要になるのはこのパーティションだけです。これは実行計画で、Pstart列とPstop列がともに1になっていることからわかります。

```
select data from part_j
where json_value (data, '$.PONumber' RETURNING NUMBER) = 500;

-------------------------------------------------------------------------------------------
| Id  | Operation                | Name    | Time     | Pstart | Pstop |
-------------------------------------------------------------------------------------------
|   0 | SELET STATEMENT          |         | 00:00:01 |        |       |
|   1 |   PARTITION RANGE ALL    |         | 00:00:01 |      1 |     1 |
| * 2 |     TABLE ACCESS FULL    | PART_J  | 00:00:01 |      1 |     1 |
-------------------------------------------------------------------------------------------
```

Oracle Partitioningは、表をパーティション化するためのさまざまなメカニズムを提供していますが、ここではスペース上の理由から省略します。詳細については、該当するドキュメントを参照してください。JSONドキュメントが大きい場合（平均>32 kb）、たいていはパーティション化キーとしてリレーショナル列を使用する方が、JSON_VALUE仮想列を使用するよりもDML操作中のパフォーマンスが高くなります。これは、後者の場合、適切なパーティションに書き込む前にJSONからパーティション化キーを抽出する必要があるからです。

パラレル実行

パラレル実行：19c、21c

複数のプロセスを使用してJSONドキュメントを処理することで、JSON操作（問合せやバルク更新など）を並列化することができます。並列化はハードウェア・リソースの使用を効率化するもので、大規模データ処理における鍵となります。

大規模なデータウェアハウスでは、優れたパフォーマンスを得るために、常にパラレル実行を使用する必要があります。OLTPアプリケーションの特定の処理（バッチ処理など）でも、パラレル実行によって大きなメリットが得られます。

パラレル実行は、問合せとDML（挿入、更新）の両方に対応しています。パラレル実行を有効にして設定する方法はいくつかあります。たとえば、Oracle Autonomous Databaseでは、接続に対して選択されたコンシューマ・グループに応じて自動的に並列処理が選択されます。並列化を手動で管理するデータベースの場合、セッション・レベルで並列処理を有効化するか、オブジェクトごとに指定できます。次の例では、表'purchaseorder'に対して8という並列度を有効にしています。

```
alter table PURCHASEORDER parallel 8;
```

パラレル実行が使用されている場合、実行計画の行に'*PX*'が表示されます。

```
|   1 | PX COORDINATOR                |
|   2 |       PX SEND QC (ORDER)      |
```

オラクルのインメモリ列形式ストレージ

インメモリでのJSON：19c、21c
インメモリ・ガイド：19c、21c

問合せパフォーマンスを上げるために、JSONデータをインメモリ列ストア（IM列ストア）に格納することができます。サイズが32 KBまでのJSONデータは、インメモリに直接ロードし、他のリレーショナル列と一緒に処理することができます。JSONドキュメントには、分析問合せとは関係ない値が含まれることも少なくありません。そのような場合、関連のあるJSONフィールドだけを個別にメモリ内に移動することで、メモリの使用効率を上げることができます。このとき、仮想列または中間マテリアライズド・ビューを使用します。

次の例では、'purchaseorder'表に仮想列を1つ追加して、注文の住所から'zipCode'フィールドを抽出しています。仮想列が追加され、表のインメモリ処理が有効化されています。

```
alter table PURCHASEORDER add (ZIP varchar2(4000) generated always as
(JSON_VALUE(data, '$.ShippingInstructions.Address.zipCode.number()')));

alter table PURCHASEORDER inmemory;
```

次のサンプル分析問合せは、zipCode別に注文数をカウントしています。高速インメモリ処理を利用していることが実行計画からわかります。

```
select zip, count(1) from PURCHASEORDER group by zip ;

-------------------------------------------------------------------------------
| Id  | Operation                          | Name          |
-------------------------------------------------------------------------------
|   0 | SELET STATEMENT                    |               |
|   1 |   HASH GROUP BY                    |               |
|   2 |     TABLE ACCESS INMEMORY FULL     | PURCHASEORDER |
-------------------------------------------------------------------------------
```

「当社では、外部APIやカスタムのユーザー拡張子から予測不可能なデータが取得されたときは常に、JSONを多用します。JSONやブロックチェーンに対するSQL分析もサポートするOracle Databaseをドキュメント・ストアとして使用することを決めました」

Retraced CTO、Peter Merkert氏

www.retraced.co

Oracle Exadata Database Machine

Exadataは、表スキャンと索引スキャンを使用する問合せで、データ検索および取得処理をExadata Storage Serverにオフロードすることで、JSONのパフォーマンスを引き上げます。このオフロードは、JSON演算子（JSON_VALUE、JSON_EXISTSなど）が問合せのWHERE句内で使用されている場合、透過的かつ自動的に実施されます。Exadata Storage Serverにオフロードできるのは、4 KBまでのJSONドキュメントです。これよりも大きいドキュメントはデータベース内で処理されます。

実行計画の*STORAGE*という用語が、オフロードの実行を示しています。

```
-------------------------------------------------------------------------------
| Id  | Operation                          | Name          |
-------------------------------------------------------------------------------
|   3 |     TABLE ACCESS STORAGE FULL      | PURCHASEORDER |
-------------------------------------------------------------------------------
```

Oracle Exadata Database Machine：ドキュメント

Oracle Real Application Clusters

Oracle Real Application Clusters（Oracle RAC）を使用すると、複数のサーバーで1つのOracle Databaseを実行できるため、共有ストレージにアクセスしながら、可用性を最大化し、水平スケーラビリティを実現できます。

Oracle Real Application Clustersドキュメント：19c、21c

JSONドキュメントの処理では、Oracle Real Application Clustersの使用を意識する必要はなく、どのようなSQL/JSON処理でも自動的にメリットが得られます。

Oracle Sharding

Oracle Shardingも水平スケーリング技術の1つですが、Oracle RACとは異なり、シェアード・ナッシング・アーキテクチャを使用します。JSONドキュメントのシャーディングにより、膨大なデータ量およびトランザクション数に対応するスケーラビリティがもたらされ、データ主権にも対応できます。JSONドキュメントは、シャーディング・キーに従って個別のデータベース表シャードに分散されます。シャーディング・キーには、1つのリレーショナル列またはJSONフィールドを使用できます。

Oracle Shardingドキュメント：19c、21

JSONコレクションのシャーディング：ドキュメント

シャーディングされたJSONドキュメントの処理で、Oracle Shardingの使用を意識する必要はありません。多くの操作で、シャーディングされたドキュメントの処理は特定のドキュメント・シャードを持つデータベース上のみで実行されますが、クロス・シャード問合せでは、すべての関連シャードから透過的に結果データが収集および集計されます。

Oracle DatabaseにおけるJSONのパフォーマンス・チューニング機能についての説明は以上です。この後は、JSONドキュメント・ストアAPI（MongoDBコレクションおよびSODAコレクション）のパフォーマンス関連トピックについて簡単に説明します。

SODAコレクションのパフォーマンスに関するヒント

Oracle Databaseは、コレクションとしてJSONデータにアクセスできるAPIを提供しており、*Oracle Database API for MongoDB*と*Simple Oracle Document Access API*（*SODA*）の2種類があります。概念的に言うと、JSONコレクションは、自動生成した表にJSONデータ（ドキュメントと呼ばれる）を格納します（よって、SQLアクセスも可能）。SODAは、通常の表と同じストレージ・オプションをJSONデータに対してサポートします。また、Oracle 19cではBLOBを使用し、Oracle 21cではネイティブJSONタイプを使用するという同じ推奨事項が当てはまります。

SODA API

Oracle Database API for MongoDB

ユーザーがJSONコレクションを使用するとき、通常はSODA for JavaやSODA for Pythonなどのネイティブ言語ドライバを使用します。一般に、SODAネイティブ言語ドライバの方が、RESTドライバ（SODA for REST）よりもスループット（1秒あたりの操作数）が高くなります。

SODAドライバ設定時の推奨事項は以下のとおりです。

- **SODAメタデータ・キャッシュ**を有効化する

  SODAドライバ側で、各JSONコレクションのメタデータ（列名、タイプなど）を把握する必要があります。メタデータ・キャッシュを有効化することで、データベースへのラウンドトリップが回避され、待機時間とスループットが向上します。

- **ステートメント・キャッシュ**を有効化する

  ステートメント・キャッシュを有効にすると、ループや何度も呼び出されるメソッドなどで繰り返し使用される実行可能文がキャッシングされるため、パフォーマンスが向上します。Javaの場合は、JDBCを使用してステートメント・キャッシュを有効化します。

- ロードバランスされたシステムでは**DNSキャッシュをオフにする**

  ロードバランシングは、SODAの操作を複数ノード間で分散します。DNSキャッシュがオンになっていると、すべての接続が同じノードを使用するため、ロードバランシングが無効になります。Javaの場合は、次のようにシステム・プロパティを設定します。inet.addr.ttl=0

データベース・パフォーマンスのチューニング・テクニックはSODAにも当てはまります。たとえば、**SODAコレクションのパーティション化やシャーディングが可能**であり、**索引やマテリアライズド・ビュー**を使用して問合せを高速化することができます。SODAの操作は自動的に同等のSQL操作に変換されます。たとえば、SODA問合せは、WHERE句内でJSON_EXISTS演算子を使用するSELECT文に置き換えられます。

SQL操作は、v$sqlデータベース・ビューから取得できます。または、直接**SODAドライバのロギングを有効にする**方法もあります。Javaの場合、ロギング用の標準パッケージが使用されますが、以下の方法でSODAに対して有効化できます。

```
java -classpath "..." -Doracle.soda.trace=true -
      Djava.util.logging.config.file=logging.properties <program>
```

- 'oracle.soda.trace=true'と指定すると、SQL文のロギングが有効になります。
- 'logging.java.util.logging.config.file'には、java.util.logging構成ファイルへのパスを定義します。ここでは各種のロギング・レベルを指定でき、FINESTがもっとも詳細なロギング・レベルになります。

追加情報 - リンク

Oracle XE

Oracle Standard Edition

Oracle Enterprise Edition

Oracle Exadata Cloud Service

Oracle Exadata Cloud at Customer

Oracle Exadata Database Machine

Oracle Database Cloud Service

Oracle Autonomous JSON

Oracle Autonomous Transaction Processing

Oracle Autonomous Data Warehouse

Connect with us

+1.800.ORACLE1までご連絡いただくか、**oracle.com**をご覧ください。北米以外の地域では、**oracle.com/contact**で最寄りの営業所をご確認いただけます。

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