19c ORACLE Database

Oracleの索引キー（プリフィックス）圧縮と高度な索引圧縮

Oracleホワイト・ペーパー | 2019年7月

ORACLE

目次

免責事項

下記事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。マテリアルやコード、機能の提供をコミットメント（確約）するものではなく、購買を決定する際の判断材料になさらないでください。オラクルの製品に関して記載されている機能の開発、リリース、および時期については、弊社の裁量により決定されます。

はじめに

企業が保有するデータ量は指数関数的に増加していますが、このデータを管理するためのIT予算がほぼ同じ割合で増えているというわけではありません。この飛躍的なデータ増加が、企業にとって難しい課題となっています。ITは、オンライン・コンテンツの急激な増加、データ保存に関する政府の規制や、企業のビジネスが純粋に拡大したことなどによる、データ量の急速な増大に対応する必要があります。しかし、データベースの加速度的な増大により、予算内に収めながらパフォーマンスに関する要件に応え続けることが難しくなっている場合があります。重要なのは、システム・パフォーマンスの低下や追加コストを発生させずに、最小限の管理労力でデータの増加を管理することです。

ストレージのコストは大幅に下がってはいますが、エンタープライズ・クラスのストレージ・コストが同じペースで下がっているわけではありません。データ量の急増により、ほとんどのIT予算において、ストレージが最大のコスト要因の1つになっています。オラクルの革新的な圧縮技術が、大量のデータを管理するためのリソースとコストの削減に役立ちます。
Oracle Databaseには、これらの課題の解決を支援するための多くの機能とテクノロジーが搭載されています。
たとえば、表の圧縮、バックアップの圧縮、ネットワークの圧縮、列データの圧縮、LOBとファイルの圧縮、索引の圧縮などです。

索引は、リレーショナル表に格納されたデータへのさまざまなアクセス・パスを効率的にサポートできるため、OLTPおよび複合ワークロード環境で幅広く使用されています。索引は、その構造自体を維持するための書込みおよびストレージ領域を追加で使用して、データ取得操作のパフォーマンスを改善するデータ構造です。アプリケーションの多数のアクセス・パスをサポートするために、大量の索引が1つの表に作成されることがよくあります。そのため、ベース表単体のサイズと比べて、データベースのストレージ全体に占める索引の割合が大きくなることがあります。索引がデータベース領域全体の50 %を超える場合も多く、1つの表に20個を超える索引が含まれることも珍しくありません（これよりずっと多い場合もあります）。

表に作成された各追加索引は、特定の問合せの速度向上に役立つことはあっても、これらの索引を維持する必要があるDMLまたはデータ変更操作のオーバーヘッド増加の原因となります。ストレージおよび効率的なアクセスの観点から、これらの索引をできるだけ効率的に保存および管理することが非常に重要です。このホワイト・ペーパーでは、Oracle Databaseで使用可能な索引圧縮テクノロジーについて説明し、各索引圧縮オプションについて詳しく解説します。また、必要なディスク領域を最小限に減らしながら問合せのパフォーマンスを最大化するように、このテクノロジーを使用する方法とタイミングに関するガイドラインを示します。

索引キー圧縮

索引キー圧縮（別名索引プリフィックス圧縮）は、Oracle Database内のおそらくもっとも古い圧縮機能の1つであり、（9.2項の基本表圧縮より前に）Oracle Database 8.1.3でリリースされました。この機能によって、索引全体のサイズを大幅に減らせる可能性があり、複数列の一意索引と非一意索引の両方で役立ちます。そのため、索引キー圧縮は、索引による使用領域をDBAが効率的に管理するために使用できる、もっとも重要な索引最適化機能の1つです。

索引キー圧縮を使用すると、複数回繰り返される値をストレージに非効率的に保存することなく、索引セグメント（または索引構成表）中のキー値の部分を圧縮できます。索引キーを次の2つの部分に分けることで、データが圧縮されます。

- プリフィックス・エントリ：列の先頭グループであり、複数のキー値で共有されている可能性があります
- サフィックス・エントリ：すべての索引キーにおいて一意であるサフィックス列です。

プリフィックスはブロック内の複数のキーで共有される可能性があるため、より適切に（つまり、ブロックあたり1回のみ）保存し、複数のサフィックス・エントリで共有することができ、索引データが圧縮されます。

索引キー圧縮は、Bツリー索引のリーフ・ブロックで実行されます。キーは索引のリーフ・ブロック内でローカルに圧縮されます。つまり、プリフィックス・エントリとサフィックス・エントリが同じブロック内に保存されます。
サフィックス・エントリによって、索引キーの圧縮表現が構成されます。これらの圧縮された各行は、同じブロックに保存されている、対応するプリフィックスを参照します。プリフィックスとサフィックスを同じブロックにローカルに保存することで、各索引ブロックが自己完結型となり、余分なブロックIOを発生させずに完全なキーを構築できます。キーの再構築は、非常に効率的な、メモリのみの操作です。

以下の図は、9個のキーが含まれる非一意索引のリーフ・ブロックを論理的に示したものです。左側のブロックは圧縮されていない表現です。すべての行において、すべてのキー列と、表で対応する行のROWIDが保存されています。データから明らかなように、先頭列（プリフィックス列）には多くの繰り返しがあります。これらはブロックで、より効率的に表現できます。右側のブロックは、同じ索引リーフ・ブロックの圧縮表現です。この場合、プリフィックス列は1回しか保存されておらず、各ユーザー行には対応するプリフィックスへの参照が保存されていて、索引データが圧縮される結果となっています。

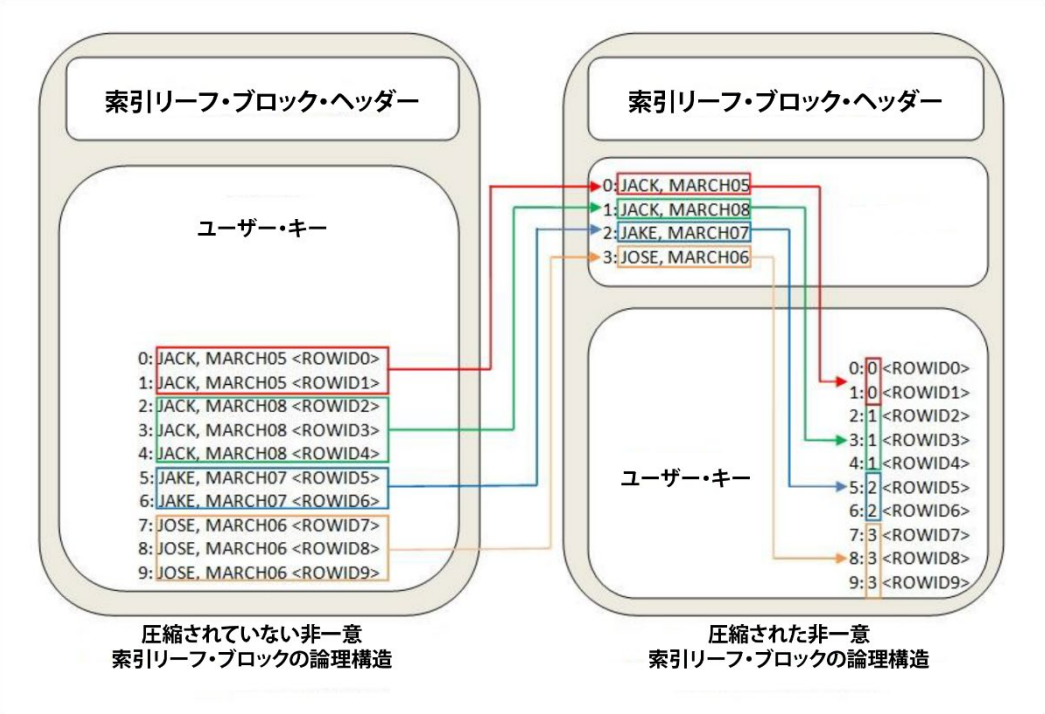

図1：索引キー圧縮を使った索引リーフ・ブロックの論理構造

索引キー圧縮の有効化

新しい索引と索引パーティションの場合、索引キー圧縮を有効にすることは簡単です。
CREATEで索引と索引パーティションを作成し、索引の圧縮句を指定するだけです。たとえば、次のような文を使用します。

```
CREATE INDEX idxname ON tabname(col1, col2, col3) COMPRESS;
```

既存の索引や索引パーティションは、以下の構文を使用してREBUILT圧縮できます。

```
ALTER INDEX idxname REBUILD COMPRESS;
```

デフォルトでは、プリフィックスは非一意索引のすべての索引付き列と、一意索引のすべての索引付き列（最終列を除く）で構成されます。または、索引圧縮句の一部としてプリフィックス長（プリフィックス・エントリの列数）を指定することもできます。

```
CREATE INDEX idxname ON tabname(col1, col2, col3) COMPRESS 2;
```

COMPRESSキーワードの後の数値（プリフィックス列の長さ）は、圧縮する列の数を示します。
非一意索引の最大プリフィックス長は、索引キーの列の数であり、一意索引の場合はキー列の数から1を引いた数値になります。

プリフィックス・エントリは、索引ブロックにそのプリフィックスが含まれていない場合にのみ、その索引ブロックに書き込まれます。プリフィックス・エントリは、書き込まれるとすぐに複数のサフィックス・エントリ間で共有でき、最後に参照しているサフィックス・エントリがブロックから削除されるまで使用可能です。キー圧縮によって複数のエントリ間でキーの一部が共有されるため、索引で必要なストレージは減りますが、索引の検索またはスキャン中に、キー列値を再構築するためのCPUオーバーヘッドが若干発生します。これは、プリフィックスをローカルでブロックに保存することで最小限に抑えることができます。

索引キー圧縮によって索引を最適に表現できます。また、メンテナンス操作によるその後のオーバーヘッドなしで、索引を恒久的に圧縮しておくことができます。そのため、ストレージと領域を節減できます。また、キャッシュ効率の向上、リーフ・ブロックと深いツリーの減少などの2次的メリットにより、論理IOを減らして実行計画のコストを削減できる可能性もあります。多くの場合、完全なユーザー行を構築するためのオーバーヘッドは、より効率的なブロック表現、特定のブロックに含めることができるユーザー行の増加、索引行の読取りに必要なIOの削減、およびバッファ・キャッシュの効率向上によって埋め合わされ、アプリケーション全体のパフォーマンスが向上します。

索引キー圧縮の使用

索引キー圧縮は、さまざまな使用例において非常に便利な場合があります。以下にいくつか例を挙げます。

- 索引キー圧縮を非一意索引で使用する場合、ROWIDを追加してキーを一意にすることができます。キー圧縮を使用してこのような索引を圧縮した場合、索引ブロックには重複するキーが1回だけ、プリフィックス・エントリとして（ROWIDなしで）保存されます。残りの行は、ROWIDのみで構成されるサフィックス・エントリになります。

- 索引キー圧縮は、一意の複数列索引で使用できます（一意の単一列索引の場合、キー圧縮はできません。一意の要素はありますが、共有すべき、プリフィックスのグループ化要素がないためです）。

- 索引キー圧縮は、索引構成表でも使用できます。一意の複数列索引と同じ考慮事項が適用されます。

最適なプリフィックス列長の特定

適切な索引圧縮で重要なのは、圧縮でメリットを得られる索引を識別し、そのプリフィックス列長を適切に指定することです。そのためには、最適なプリフィックス列数を選択できるように、データについて深く理解する必要があります。
最適な圧縮率と、節約できるリーフ・ブロックのパーセンテージを見積もるには、ANALYZEで索引を分析してからINDEX_STATSビューを確認する必要があります。

```
ANALYZE INDEX index name VALIDATE structure;

SELECT name,
       height,
       blocks,
```

```
        opt_cmpr_count,
        opt_cmpr_pctsave
FROM    index_stats
WHERE   name = index name;
```

“OPT_CMPR_COUNT”は、リーフ・ブロック内の領域を最大限節約するために圧縮する、索引中の列の数（プリフィックス列長）を示します。

“OPT_CMPR_PCTSAVE”は、このプリフィックス長を使用して索引を圧縮した場合に使用されるリーフ・ブロック領域の削減率を、パーセンテージで示します。

索引キー圧縮の制限事項

索引のプリフィックス列がリーフ・ブロック内で何回も繰り返される場合は、圧縮が非常に有益な可能性があります。
ただし、先頭列が非常に選択的な場合や、プリフィックス列に対応して繰り返される値がそれほど多くない場合は、索引プリフィックス圧縮が最善の解決策とはならない可能性もあります。このような場合でも、リーフ・ブロック内における圧縮列値の一意の組合せをすべて保存するプリフィックス・エントリが作成されます。索引行はプリフィックス・エントリを参照します。プリフィックス・エントリは（仮にそのようなことがあったとしても）他の索引行とは共有されません。
そのため、このような場合は圧縮は有益ではなく、すべてのプリフィックス・エントリを保存することでオーバーヘッドが発生し、結局は索引のサイズが増えてしまう可能性があります。

索引の圧縮を有益なものにするには、低カーディナリティの列が、連結索引内の先頭列となるようにします。
そうしないと、非圧縮のリーフ・ブロックより少ないキーしか保存できなくなる、負圧縮の状態となる危険性があります。
また、単一列の一意索引や、連結された複数列の一意索引中の各列を圧縮しても意味がありません。
このような場合、索引行ごとにプリフィックス・エントリがあることですべてにオーバーヘッドが発生するため、圧縮によって索引構造のサイズが減少せずにむしろ増えてしまいます（負圧縮）。

適切な索引圧縮で重要なのは、圧縮でメリットを得られる索引を識別し、プリフィックス列長を適切に指定することです。これまで説明してきた最適なプリフィックス列長の特定方法は役に立つ場合もありますが、次のようなマイナスの側面もあります。

- 最適なプリフィックス列数を選択するために、データについて深く理解する必要があります。
- 指定したプリフィックス列数によって、索引中のすべてのブロックで最適な圧縮率を得られるとは限りません。
- ANALYZE INDEXを実行して、最適なプリフィックス列数を取得し、全体で最適な索引数となるようにする必要があります。この粒度レベルはブロックではないため、最適な圧縮率にはならない可能性があります。

また、ANALYZE INDEXを実行すると表が排他ロックされ、この期間は表が事実上“オフライン”になります。

- 指定したプリフィックス列がブロック内で一意である場合などは、前述のように負圧縮となる可能性があります。

アプリケーション開発者とDBAは、圧縮する索引を慎重に選択し、これらの索引のプリフィックス列数を適切に設定する必要があります。オラクルは特定の明確な条件下でお客様を支援いたしますが、索引の適切な方法での圧縮は、お客様の責任において実施してください。

既存の非圧縮索引構成表（IOT）の圧縮

再編成を試みる前に、プリフィックス圧縮が有用かどうか、ならびに有用な場合は、どの程度の数のプリフィックス列を指定する必要があるかを判断します。このステップは索引キー圧縮に必須であり、データに関する知識がある程度必要です。繰り返しの先頭列がある場合は、一般的に、プリフィックス圧縮は有用です。

たとえば、キーが以下のようであるとします。

A B C D
A C D B
A D B C

最初の列は繰り返しであるため、プリフィックスとして保存でき、圧縮によってサフィックス行を以下のように減らすことができます。

B C D
C D B
D B C

行が以下のような場合、選択はさらに難しくなります。

A B C D

A B D C

A C E F

A G H I

ここでは、圧縮するプリフィックス行を1行選択するのか2行選択するのかを特定するのは困難です。2行選択する場合は、以下のプリフィックス行を保存することになります。

A B
A C
A G

1行選択する場合は、"A"だけを保存することになります。"最適な"選択は、プリフィックス行とサフィックス行を合わせて領域をもっとも多く節約する選択です。

前述したように、ANALYZE INDEXで、圧縮する列数について最適な数が決まります。
たとえば、以下のようなDDLを使用するとします。

```
CREATE TABLE tiot (c1 number, c2 number, c3 number, constraint tiot_pk primary
key (c1, c2, c3)) ORGANIZATION INDEX;
```

この場合、ANALYZE INDEXを以下のように使用できます。

```
ANALYZE INDEX TIOT_PK VALIDATE STRUCTURE;

SELECT OPT_CMPR_COUNT, OPT_CMPR_PCTSAVE FROM index_stats;

OPT_CMPR_COUNT OPT_CMPR_PCTSAVE
-------------------- ---------------------- --------------- ----------------- ---------------
           1                      20
```

ANALYZE INDEXによって、プリフィックス数1の場合に領域が推定20 %節約されることが示されています。

以下のALTER TABLE MOVEコマンドをオンラインで使用し、圧縮を有効にして、IOTセグメントを再作成できます。
1列が圧縮に使用されることが示されています。

```
ALTER TABLE tiot MOVE COMPRESS 1 ONLINE;
```

以下に、既存の非圧縮IOTで索引キー圧縮を実行するためのSQLコマンド/例（およびIOTで無効なSQLコマンドの例）をいくつか示します。

- ALTER TABLE <table-name> MOVE COMPRESS [number of columns] [ONLINE] 例：
    - alter table tiot move compress online;
    - alter table tiot move compress 1 online;
    - alter table tiot move compress 2 online;

    - alter table tiot move compress 1

- ALTER TABLE <table-name> COMPRESS
    - これは、IOTで無効なコマンドです

- ALTER INDEX REBUILD COMPRESS
    - これは、IOTで無効なコマンドです。IOTの主キーを再作成しようとすると、以下の結果になります。
        - ORA-28650: Primary index on an IOT cannot be rebuilt

- 既存の非圧縮IOTを圧縮するには、DBMS_REDEFINITIONを使用できます。
  詳しくは、dbms_redefinitionのドキュメントを参照してください。

索引のプリフィックス圧縮を使用する場合、圧縮を有効化する手段としてパーティション交換を使用することはできません。パーティション化されたIOTは全体として移動できず、また、表（IOT）が全体として圧縮されていない場合は、圧縮対象のパーティションを移動できないためです。さらに、パーティションと表の両方に同じ圧縮属性が指定されている（つまり、両方がすでに圧縮されている、または両方とも圧縮されていない）場合を除き、パーティションを交換することはできません。

高度な索引圧縮

重複キーが多く含まれる索引エントリを圧縮すると、大規模な索引レンジ・スキャンや高速全スキャンのストレージ・オーバーヘッドとアクセス・オーバーヘッドを削減できます。索引のプリフィックス列がリーフ・ブロック内で何回も繰り返される場合は、プリフィックス圧縮が非常に有益な可能性があります。ただし、先頭列が非常に選択的な場合や、プリフィックス列に対応して繰り返される値がそれほど多くない場合は、索引プリフィックス圧縮が最善の解決策とはならない可能性もあります。

高度な索引圧縮は、索引圧縮を自動化すると同時に、索引圧縮率を大幅に改善します。高度な索引圧縮を使用すると、非常に優れたデータ圧縮率を実現でき、I/Oの減少による大幅なコスト削減およびパフォーマンス向上が企業にもたらされます。

高度な索引圧縮は、複数の圧縮レベル（低および高）を利用できるテクノロジーです。ストレージの節約量は、実行する圧縮のレベルによって異なりますが、平均で2～4倍です。高度な索引圧縮による相当なストレージ節約によって、ITマネージャーは数年間にわたるストレージの新規購入量を大幅に削減できます（多くの場合、完全に不要となります）。次に、次世代の索引圧縮テクノロジーとして、各圧縮レベルを詳しく説明します。

高度な索引圧縮（低）

高度な索引圧縮（低）は、索引キー圧縮を自動化します。圧縮対象の索引が自動的に決定され、圧縮された索引内のプリフィックス列数が計算されます。また、すべての索引リーフ・ブロックに静的なプリフィックス数が使用されるのではなく、索引内の索引リーフ・ブロックごとに最適なプリフィックス数が計算されます。

適切で最適なプリフィックス列数がブロック単位で自動的に計算されるため、可能な限り最適な圧縮率を得ることができます。索引リーフ・ブロックごとに、別々のプリフィックス列数で圧縮できるようになりました。また、繰り返しのプリフィックスがない場合は、まったく圧縮しないこともできます。

以下の図は、それぞれが圧縮に関して異なる状態にある、3つの連続する索引リーフ・ブロックの論理構造を示しています。左のブロックでは最適なプリフィックス列数が2で、プリフィックス内に含まれる、索引キーの最初の2列を使用してブロックが圧縮されています。中央のブロックでは先頭列に繰り返しがないため、ブロックは圧縮されないままです。
また、右のブロックでは最適なプリフィックス列数が1で、先頭の1つのプリフィックス列のみを使用してブロックが圧縮されています。ブロック単位で自動的にプリフィックス列数を計算する動的アルゴリズムによって、索引の圧縮によるメリットを最大化し、圧縮した索引セグメントのサイズが非圧縮の場合より大きくならないようにします。

図2：高度な索引圧縮を使った索引リーフ・ブロックの論理構造

高度な索引圧縮（低）の有効化

高度な索引圧縮（低）は、索引に対してCOMPRESSオプションを指定することで簡単に有効化できます。
圧縮時に新しい索引を自動的に作成できます。または、既存の索引を圧縮状態に再構築することもできます。

```
CREATE INDEX idxname ON tabname(col1, col2, col3) COMPRESS ADVANCED LOW;
```

高度な索引圧縮では、プリフィックス・エントリの列数を指定する必要がないことに注意してください。
リーフ・ブロックごとに自動的に計算されるためです。

高度な索引圧縮（高）

高度な索引圧縮（高）は、索引の圧縮率を大幅に高めることを目的としています。高度な索引圧縮（高）には多くの圧縮技術が追加で導入されており、圧縮率の大幅な改善と効率的なOLTPアクセスを同時に実現しています。

高度な索引圧縮（高）では、すべての索引リーフ・ブロックに圧縮行と非圧縮行を含めることができます。
圧縮された索引キー・エントリは、圧縮単位（ハイブリッド列圧縮と似た概念です）として物理的に保存されます。
この際、より複雑な圧縮アルゴリズムをより多くの索引キーに使用できる可能性があるため、圧縮率が改善されます。
一方、最近挿入されたキーと変更されたキーは、リーフ・ブロックの非圧縮領域に保存されます。

高度な索引圧縮では、高度な行圧縮と同様に、内部しきい値を使用してリーフ・ブロックの（再）圧縮をトリガーします。最近挿入された行はブロックのバッファに圧縮されない状態で保存され、ブロックの使用率がこのしきい値に近くなると圧縮されます。そのため、圧縮コストが複数のDML操作で償却され、すべての操作で圧縮オーバーヘッドが発生するというわけではありません。索引では、この内部しきい値によって索引ブロックの分割を防ぎ、索引構造への追加リーフ・ブロックの割り当ての必要性を低下させます。

高度な索引圧縮では、行圧縮によって完全な同時実行性と行レベル・ロックをサポートするため、デッドロックの解消と完全なアプリケーション透過性を実現できます。

前述のとおり、高度な索引圧縮では、複雑な一連の圧縮アルゴリズムによって優れた圧縮率を達成します。
高度な索引圧縮（高）で使用される圧縮技術には、たとえば次のようなものがあります。

- 列内のプリフックス置換

  列内のプリフックス置換アルゴリズムでは、索引行をキー順にソートすると、各キーのプリフィックスが前のキーの対応するプリフィックスと一致する可能性が（サブ・キー列レベルでも）高いという事実を利用しています。
  各行の一致するプリフィックスを、対応する記号への参照に置き換えると、圧縮率が改善されます。
  また、記号表の索引のカーディナリティが低く、多数の索引キーのプリフィックスが一致する場合は、記号表の参照をビット・エンコーディングすると、圧縮率がさらに改善される可能性があります。

- 長さバイトの圧縮

  列長が短い索引に多数の行が含まれることは非常に多くあります。そのため、（非圧縮の索引やプリフィックスを圧縮した索引と同様に）これらの長さを1バイト未満にエンコードして、領域を節約できます。
  また、ブロック内におけるすべてのキー列の長さが同じ場合は、ブロック・レベルの固定長を保存できます。

- 重複キーの削除

  索引ブロックに多数の重複が含まれる場合は、キーに続いて、そのキーに関連付けられたROWIDをソートしたリストを付けて、それらを1回だけ保存すれば、領域を大幅に節約できます。次に、この変換された表現に対して列内のプリフィックス圧縮を重ねて適用すれば、一意となったキー・セットをさらに圧縮できます。

- ROWIDリスト圧縮

  ROWIDリスト圧縮は、一意の索引キーごとの一連のROWIDで実行され、これらのROWIDを圧縮形式で表す独立した変換です。圧縮されたROWID表現がROWIDの順序で論理的に保持されるため、ROWIDベースの効率的な検索が可能となります。

- 行ディレクトリ圧縮

  行ディレクトリ圧縮の背後にある一般的な考えは、索引ブロックの256バイト領域内のそれぞれにおいて、圧縮した行を、オフセットが増加する順序で連続的にレイアウトすることです。これで、ベース・オフセット（256バイトあたり1回）と、圧縮した行あたり1バイトの相対オフセットを維持できます。

- フラグおよびロック・バイトの圧縮

  一般的に、索引行はロックされておらず、索引ブロック内におけるすべての行のフラグは似ています。
  ディスク上のこのようなロックとフラグのバイトにアクセスして変更できれば、より効率的な表現が可能になります。
  フラグ・バイトやロック・バイトを変更するには、これらが圧縮されていないことが必要です。

高度な索引圧縮（高）の有効化

高度な索引圧縮（高）は、索引に対してCOMPRESSオプションを指定することで簡単に有効化できます。
圧縮時に新しい索引を自動的に作成できます。または、既存の索引を圧縮状態に再構築することもできます。

```
CREATE INDEX idxname ON tabname(col1, col2, col3) COMPRESS ADVANCED HIGH;
```

高度な索引圧縮で使用する圧縮方法を指定する必要はないことに注意してください。索引によっては、適用できない圧縮方法があります。索引に適用できる圧縮アルゴリズムはリアルタイムに決定され、索引やブロックによって異なる可能性があります。

高度な索引圧縮の使用

高度な索引圧縮は、サポートされるすべての索引（プリフィックス・キー圧縮に適さない索引を含む）で問題なく使用できます。高度な索引圧縮を使用して索引を作成すると、すべての一意（または非一意）索引のサイズを減らすことができます（または、少なくとも負圧縮によるサイズ増大を防ぐことができます）。また同時に、索引に効率的にアクセスしながら圧縮率を大幅に改善することができます。

次のグラフは、SAP環境で高度な索引圧縮を使用している2種類の顧客について、圧縮率のサンプルを示しています。これらのワークロードでは、索引のストレージ・フットプリントが大幅に減少し、システム全体のパフォーマンスも大幅に向上しました。

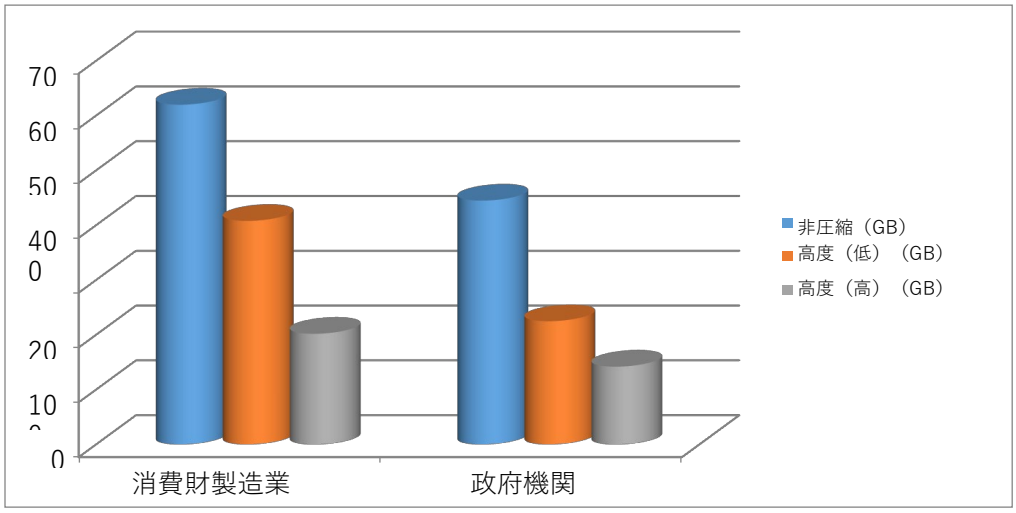

図3：高度な索引圧縮による圧縮率のサンプル（サイズはGB単位）

パーティション索引での高度な索引圧縮

パーティション索引の場合、索引全体またはパーティション単位で圧縮句を指定できます。
そのため、一部の索引パーティションのみを圧縮し、他は圧縮しないままにすることができます。

次に、パーティション索引の圧縮属性を組み合わせた場合の例を示します。

```
CREATE INDEX my_test_idx ON test(a, b) COMPRESS ADVANCED HIGH local
```

```
(PARTITION p1 COMPRESS ADVANCED LOW,
 PARTITION p2 COMPRESS,
 PARTITION p3,
 PARTITION p4 NOCOMPRESS);
```

以下の例では、親索引が圧縮されていないパーティションにおける高度な索引圧縮のサポートを示します。

```
CREATE INDEX my_test_idx ON test(a, b) NOCOMPRESS local
    (PARTITION p1 COMPRESS ADVANCED LOW,
     PARTITION p2 COMPRESS ADVANCED HIGH,
     PARTITION p3);
```

高度な索引圧縮の制限事項

- ビットマップ索引では、高度な索引圧縮がサポートされません。
- 索引構成表（IOT）では、高度な索引圧縮がサポートされません。
- ファンクション索引の圧縮では、高度な索引圧縮がサポートされません。

結論

企業は、データ量の急増という重大な問題に直面しています。企業は、収益に影響を与えることなく変化の激しいビジネス状況に迅速に適応する必要があります。ITマネージャーは、既存のインフラストラクチャを効率的に管理してコストを制御しながら、優れたアプリケーション・パフォーマンスを提供し続けていく必要があります。

高度な索引圧縮によって、すべてのBツリー索引を圧縮できるようになりました。このとき、圧縮でメリットを得られるすべての索引リーフ・ブロックが自動的に圧縮される一方で、ブロックごとに最適なプリフィックス列長が計算されます。
そのため、索引圧縮がブロック・レベルの真にローカルなものになり、圧縮プリフィックス表の作成とリーフ・ブロックの圧縮方法の決定が、ブロックごとにローカルで行われます。さらに、索引に効率的にアクセスしながら、索引セグメント全体にとって最適な圧縮率の実現を目指します。

企業は、高度な索引圧縮とOracle Advanced Compressionの他の機能を組み合わせて利用することにより、必要量が増大しているデータを最小限の管理作業で効率的に管理でき、最高レベルのアプリケーション・パフォーマンスを実現しながらデータベース・ストレージのコストを最小限に抑えられます。

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