Session Report

BigQuery の性能を支える技術と
ユーザーでできる性能改善のヒントを解説

データ分析を加速する BigQuery 性能最適化のポイント

Google Cloud
カスタマー エンジニアリング
データ アナリティクス スペシャリスト
西村 哲徳

Google Cloud

Google Cloud

セッション レポート概要

BigQuery は特にチューニングをしなくても大規模なデータを高速に処理できるアーキテクチャを採用していますが、ユーザー側で最適化のポイントを理解して利用することで、さらに効率的なデータ分析が可能になり、素早い意思決定につなげることができます。今回は高速化と拡張性を実現しているアーキテクチャと最適化のポイントを紹介します。

プレゼンター紹介

Google Cloud
カスタマー エンジニアリング
データ アナリティクス スペシャリスト
西村 哲徳

複数のベンダーで主にデータ プラットフォーム製品、サービスのプリセールス エンジニアとして PoC やソリューションの提案などを経験して、現在はデータ アナリティクス スペシャリストとしてお客様のデータ分析基盤の課題を解決できるよう日々活動中です。

目次

BigQuery のアーキテクチャの概要

BigQuery はコンピュートとストレージの分離により、柔軟な拡張性を提供できるのが特長です。各コンピュートやストレージは Google のデータセンターにて巨大なクラスターとして管理されており、ユーザーは特に管理することなくデータを増やし続けても、問題なく処理できるだけの性能を誇ります。

ボトルネックになりやすいコンピュートとストレージの間のネットワークについても、ペタビット規模を誇り高いスケーラビリティを持ちます。つまり、ソフトウェアだけでなくハードウェアも最適化することで、高速かつ拡張性の高い仕組みを提供しています。

BigQuery のアーキテクチャ全体像

BigQuery のアーキテクチャ：コンピュート

BigQuery はどのようなアーキテクチャになっているか、まずはコンピュートについて、項目ごとに見ていきましょう。

・Borg（クラスター管理）

BigQuery は、クラスター管理の仕組みである「Borg」上で動いています。Borg は数十万台超のサーバー運用やジョブ管理が可能であり、BigQuery はタスク処理専用の何千もの CPU コアを使用できる非常に高い拡張性を有しています。Google によって管理されているので、ユーザーはメンテナンスによるダウンタイムを気にする必要はありません。

・Dremel（実行エンジン）

「Dremel」は、上記クラスター上で、SQL 文からクエリプランを生成する実行エンジンであり、クエリのサイズと複雑さにもとづいて最適なリソース量になるよう自動調整します。クエリを処理していくうちにデータ量が減った際にリソースを減らして結果を素早く返すというような、実行中の動的な調整も行います。

さらに、動的なプラン変更も行います。複数のテーブルを結合する場合、最適な順序で制御するのは難しいですが、BigQuery はデータ量に合わせて結合方式を変更していくため、大規模データを用いた複雑な処理の場合でも、非常に高速なレスポンスを実現できます。

Dremel における、レイテンシを削減する手法については、主に以下の技術が利用されています。

- **Speculative Execution（投機的実行）**

別のワーカーに同じタスクを振っておくことでレイテンシを削減します。

- **Multi-level execution trees**

root server、intermediate server、leaf server と複数レベルにて処理を実行できるツリー状の構造にすることで、リクエストを root から intermediate へ、さらに intermediate から leaf へ投げることで並列処理が可能になります。結果も全て並列化して返されます。

- **Adaptive query scaling**

複数レベルで実行するので、例えば、簡単なクエリは leaf で読み込みながら集計して最後に root で計算するという多段階の処理を実行します。少し複雑なものでは、intermediate を間に入れて、なるべく並列処理できるように動きます。

Dremel | レイテンシ削減する手法

Speculative Execution（投機的実行）

Multi-level execution trees

Adaptive query scaling

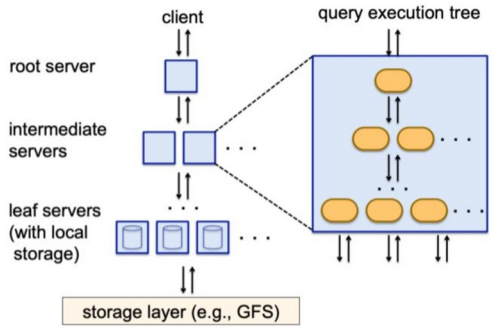


Figure 3: System architecture and execution inside a server node (Figure 7 in [32])

Dremel は複数レベルの実行可能なツリー構造で並列処理してレイテンシを削減

BigQuery のアーキテクチャ：ストレージ

ストレージに関しては、以下の「Colossus」と「Capacitor」が適切に機能することで非常に高いパフォーマンスを実現しています。

・Colossus（分散ストレージ）

巨大な分散ストレージである Colossus は、ユーザーへ一度に数千の専用ディスクを提供するのに十分なディスクであり、ペタバイト規模のアクセスでもスケールし、数千コアから読み込んでも十分なスループットが出るようにデータを配置します。速度だけでなく、高い耐久性のある格納方式やデータ複製、リカバリ自動化によって可用性にも優れます。

・Capacitor（カラムナー ストレージ）

BigQuery は、Capacitor と呼ばれるカラムナー ストレージのフォーマットでデータを保持しています。ネストやリピートなど半構造化データをサポートしているのが特長であり、JSON にようにネストが深い場合でも、各項目を別の形で格納することで高い格納効率を実現し、項目へのアクセスも効率的に行えるようになっています。

Capacitor のファイル ヘッダーには列の統計情報を持っています。例えば、無駄を省くための情報、例えば WHERE 句の複数条件をどの順序で評価すれば速くフィルタリングできるかという情報が含まれています。

圧縮に関しても、単に深く圧縮するだけではなく、性能を考えた上で圧縮するような効率の良い圧縮方式をとっています。Colossus に Capacitor を書き込む際にも、数千のマシンから並列に読み込めるように最適な数にシャーディングして書き込むことによって、高いスループットを得ています。

BigQuery のアーキテクチャ：メモリとネットワーク

・分散インメモリ シャッフル

BigQuery を含む大規模なデータ処理では、データの再分割、いわゆる「シャッフル」という処理が行われています。シャッフルの中間結果の保存には、これまでローカル ディスクとメモリを組み合わせて行われていましたが、BigQuery では、2014 年頃から分散インメモリ型の方式を採用しており、それによって書き込みの断片化の防止や Read のスループット向上を実現。スケーラビリティのボトルネックを解消できるようになりました。

・Jupiter（ネットワーク）

BigQuery では Jupiter と呼ばれるネットワークを搭載しており、その総帯域幅は 6 Pb/秒を超えています。大量のデータを読み込むときのスキャンも、シャッフル処理におけるメモリのデータの移動も非常に高速に行えます。

このように BigQuery では、コンピュートとストレージとメモリを分離してそれぞれのスループットを向上させ、さらにネットワークがそれらを支えることで、特に大規模データに対する優れたパフォーマンスを実現しています。

BigQuery の最近の性能改善

BigQuery は継続的に性能改善を行っており、そのうち最近の代表的なものを紹介します。

- **スモールクエリのパフォーマンスの向上**

  テーブルのサイズに合わせて Capacitor のファイルのサイズを動的に変更します。これによりテーブルサイズが小さい場合の処理（クエリ）でも、並列で読み込めるためパフォーマンスが向上しました。

- **メタデータ管理によるクエリの高速化**

  BigQuery では、「Capacitor dynamic metadata (CMETA)」というメタデータ管理システムを利用しています。先ほど Capacitor ファイルのヘッダーに列の統計情報が入っていると述べましたが、ペタバイト規模のテーブルではそれを読み込むオーバーヘッドが大きくなります。そこでメタデータを別のテーブルに出して、SQL を投げたときにそれとセミ結合し、必要なファイルだけを読み込むことで高速化しています。

- **結合スキュー処理の手法でデータ分析の遅延を減少**

  並列分散処理では結合のキーが偏っていると、どれだけリソースが豊富でもそれをうまく活用できずボトルネックが生じるケースがあります。BigQuery はスキューの値を検出すると、それを自動的に分割して複数のワーカーで処理できる仕組みになっています。

性能最適化のポイント：効率的な SQL

ここまで、BigQuery が内部で行っている性能最適化を説明してきましたが、ここからはユーザー自身ができるチューニング ポイントやテーブル設計を解説します。

私自身は、性能最適化のために、なるべく少ないリソースで同じ結果が得られるように心がけています。例えば、なるべく読み込み量を減らせば、読み込み時間が短くなり I/O や通信量が減ります。

性能最適化のポイントの全体像

ここから性能最適化の方法を 1 つずつ見ていきます。

・必要な列のみクエリする

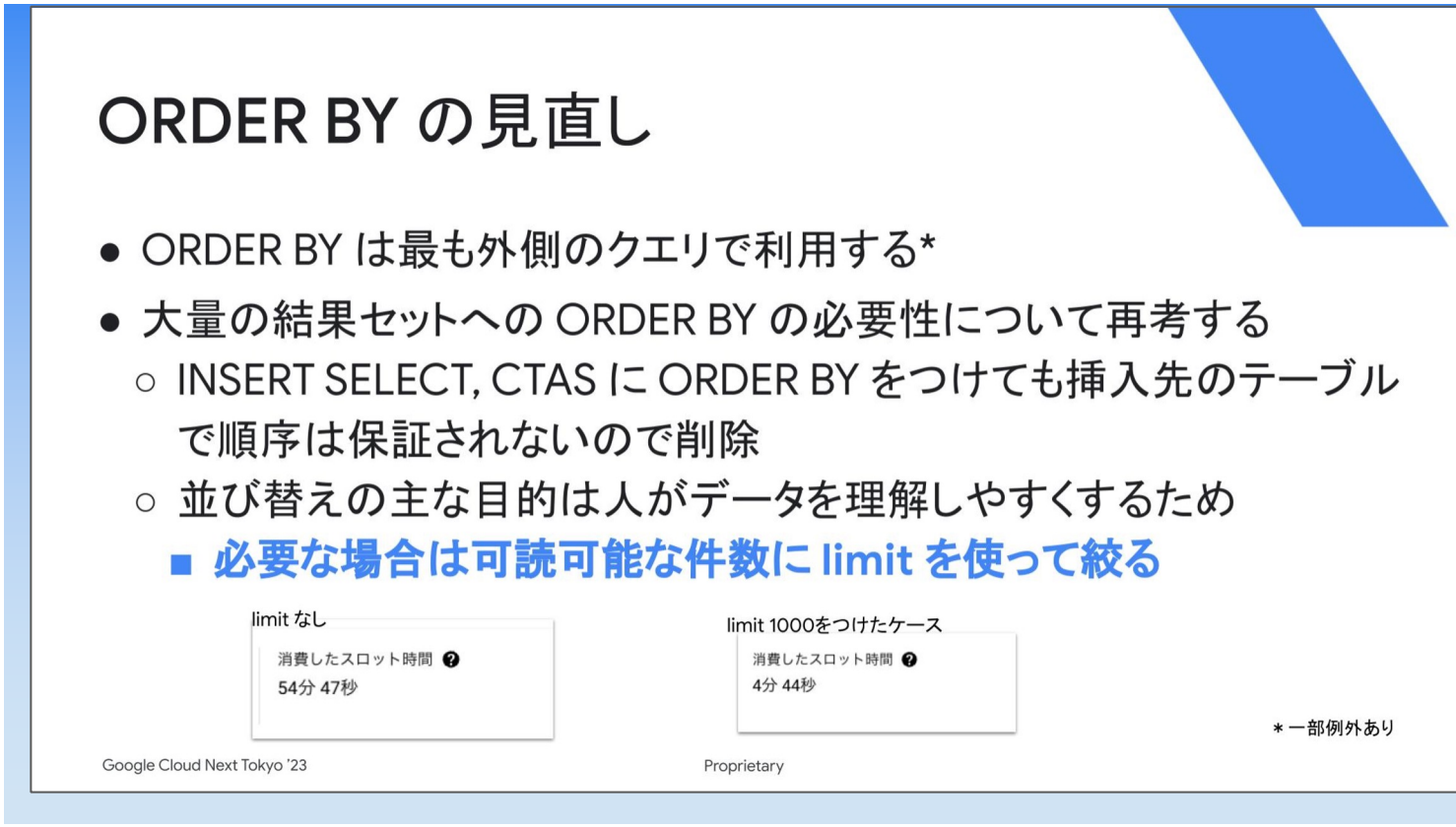
基本的なことですが、必要な列のみをクエリすることで、I/O やコンピュート リソース削減が可能です。コンソールなどに書くとき、ついつい面倒なので SELECT * と書いている方は気をつける必要があります。

・WHERE 句で使用する列

クエリよりもテーブル設計に近い話ですが、WHERE 句でのフィルタリングや結合には、なるべく適切な型を使うことで効率化されます。STRING や BYTES よりも BOOL / INT / FLOAT または DATE が効率的です。特に日付は STRING ではなく DATE で持つことでサイズも小さくなります。

・ORDER BY の見直し

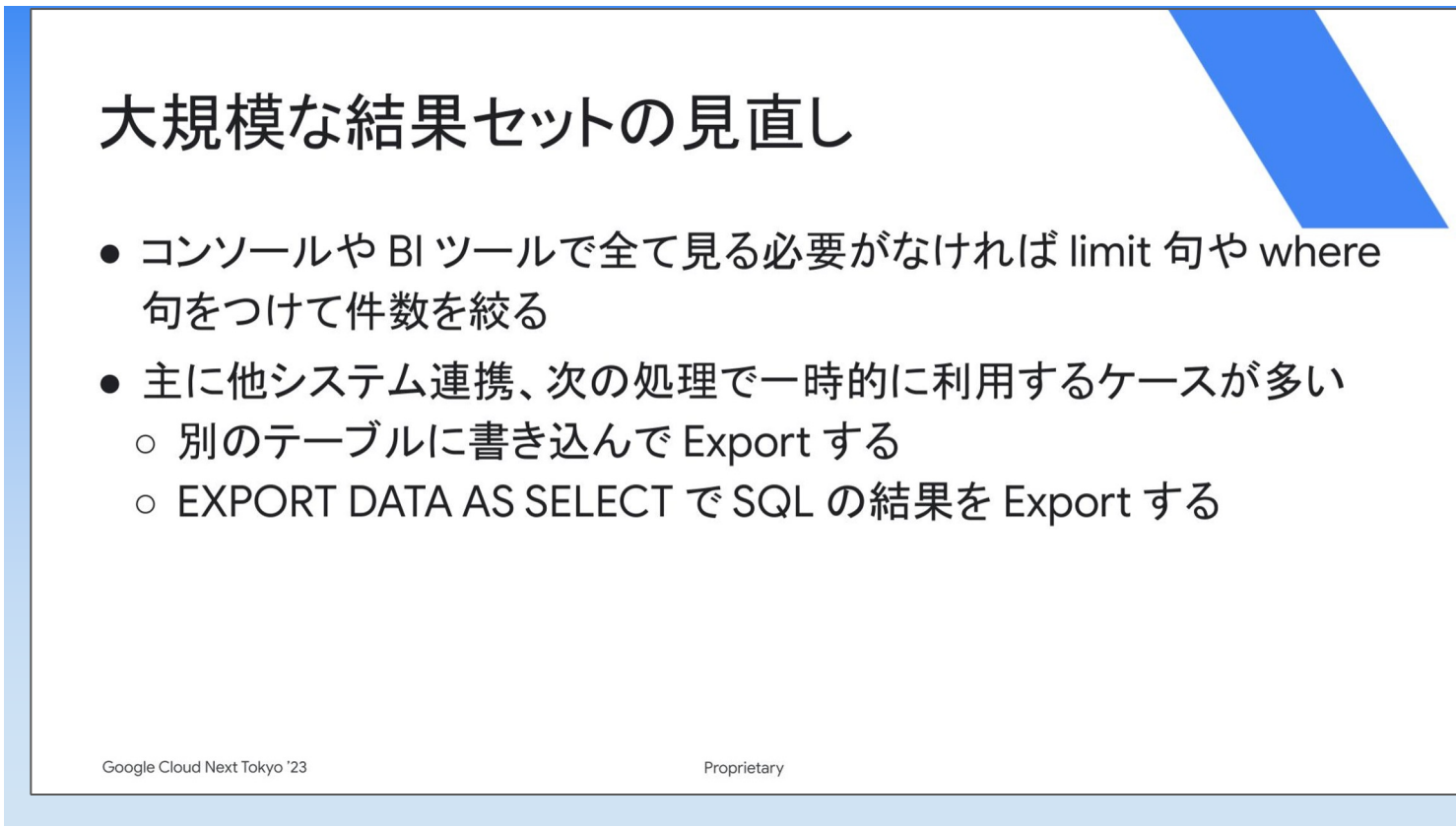
ORDER BY は最も外側のクエリで利用します。通常はそのように書くはずですが、他の人が書いた SQL を流用するとき、サブクエリに入れてしまい、それが残ったままとなりやすいため、気をつける必要があります。



ORDER BY の見直しのポイント

・大規模な結果セットの見直し

コンソールに書くときなど、WHERE 句や LIMIT 句を付け忘れやすくなりますが、これらを忘れずにつけて件数を絞ることで、リソースを抑えられる可能性があります。



大規模な結果セットの見直しポイント

・結合順序の指定

結合順序はオプティマイザが決定するため、細かく気にする必要はありませんが、ベスト プラクティスとしては、なるべくテーブルはサイズの大きい順で SQL 内に記述します。複雑な SQL は、データの中身を知っている人が順序を意識して書くようにします。また、外部結合を使いがちですが、内部結合で書けないかを意識しましょう。

・自己結合の回避

自己結合は概ねウィンドウ関数で書き換えられるため、どうしても自己結合が必要だと思ったときには、適切なウィンドウ関数がないか調べてみましょう。

・クロスジョインの回避

多くの人は直積を出さないように気をつけていると思いますが、よく本人が意図せず発生するケースがあります。例えば下図の青字の部分のように結合条件が等価結合でない場合（不等号、BETWEEN、LIKE など）に発生しがちです。

クロスジョインは意図せず発生する可能性がある

これは、そもそもデータの持ち方から見直しが必要なこともありますが、回避する方法もあります。例えば、結合条件を OR で連結しているようなケースでは、SQL を分解して UNION ALL や UNION にすることで内部結合が利用できます。ただし、スキャン量と結合の回数が増えるため、両方を試して高速な SQL を採用するとよいでしょう。

・COUNT（DISTINCT）の見直し

COUNT（DISTINCT）自体が重い処理ですが、気をつけたいのは、集計の粒度（GROUP BY の項目）が変わると再計算する必要があるということです。確定データではなく速報値を見るなど概算を知りたいといった用途では、近似集計の関数 APPROX_COUNT_DISTINCT を使えないかなど、軽くなる方法を検討します。

・ウィンドウ関数の見直し

ウィンドウ関数でトップ N のランクを出すようなケースは少なくありません。その場合はウィンドウ関数を使う前に、一度 LIMIT 句で絞ってから並べ替えると軽くできます。

・最新レコードを取得する集計関数の見直し

最新状態を取得する場合、RANK や ROW_NUMBER の代わりに使える、便利な集計関数に max_by や min_by があります。

性能最適化のポイント：処理内容の見直し

続いては処理内容を見直すことで性能を改善する方法の例を紹介します。

・中間テーブルの利用

次のようなケースでは、一度中間テーブルに処理結果を出力して、それを実行する方法が効率的です。

- 1 つの SQL内で JOIN が多いなどの複雑なクエリがある場合
- 複数の SQL から利用される共通のサブクエリや、WITH 句の定義を何度もクエリ内で参照するような共通した処理を行う場合

ただし、WITH 句を使う共通テーブル式（CTE）のケースは BigQuery 自体でも改善を行っており、重複で参照している場合は最適化するようになっています。

・シリアル処理の削減

データスキューの問題に関しては BigQuery で改善が行われているため、ユーザーが対応できることはあまり多くありませんが、次のような対応を検討します。

- 非正規化して結合数を減らす
- 必要なければ偏りのある値を早い段階でフィルタする
- 小さいテーブルであればブロードキャスト結合ができないか試す

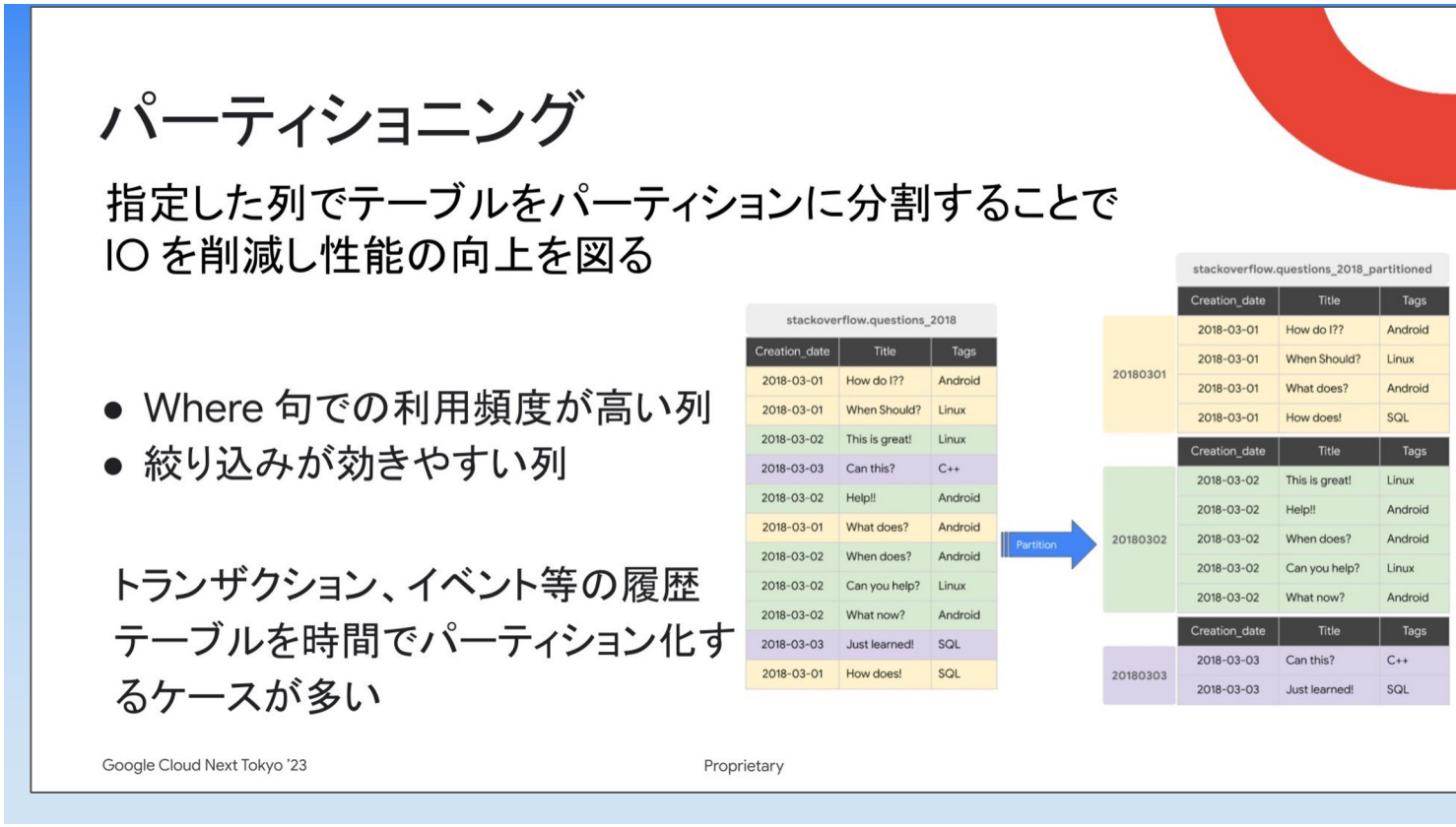
また、Salt で同じキーを分割して並列処理させるのも手です。

性能最適化のポイント：テーブル設計

最後にテーブル設計による性能最適化について見ていきます。テーブルは設計次第で良い効果と悪い影響どちらも与えうるものです。その意味で性能最適化における重要ポイントですが、やるべきことは難しくなくシンプルです。

・パーティショニング

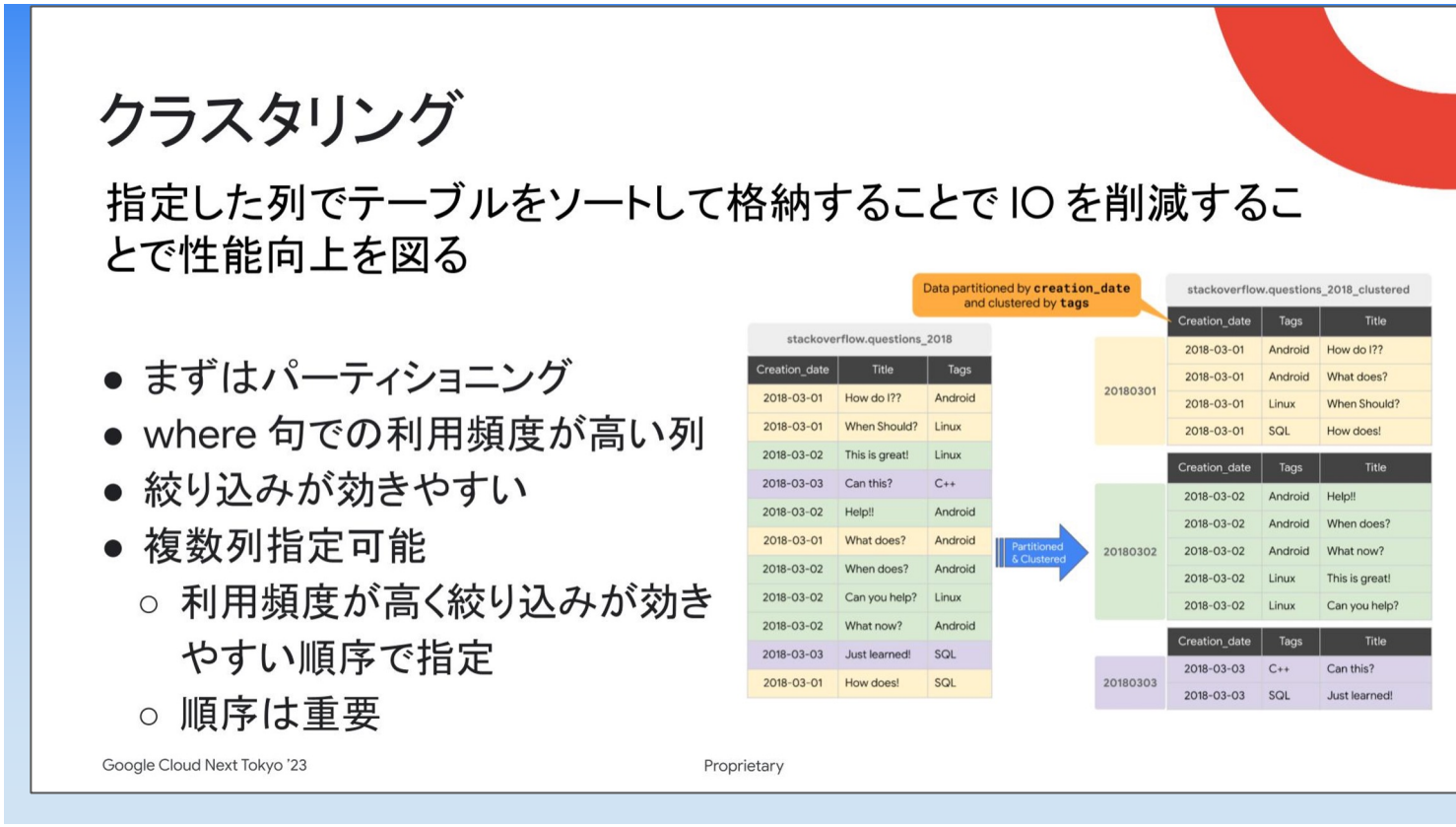
まず I/O を減らすために、指定した列でテーブルを内部的にパーティションに分割します。どの列をパーティションで選ぶべきかという点に関しては、WHERE 句での利用頻度が高く絞り込みが効きやすいものが望ましいです。



日単位でパーティショニングするイメージ図

・クラスタリング

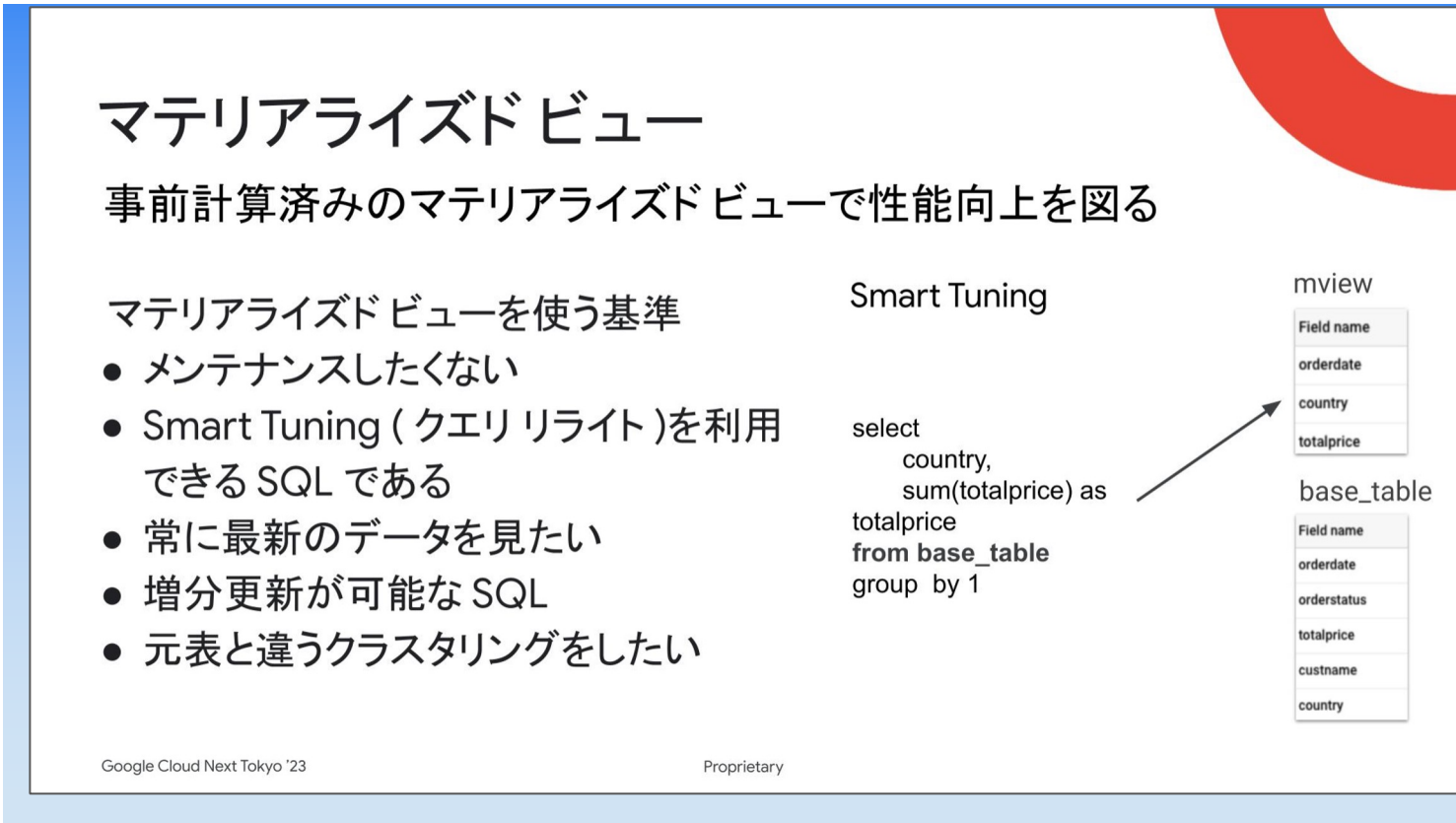
クラスタリングでは、指定した列でテーブルをソートして格納することで I/O を削減します。まずはパーティショニングを行った上で使用すると良いでしょう。GEOGRAPHY 型のようなパーティショニングで切れない列にも使えるため、地理情報の分析で使用されるケースをよく見かけます。



クラスタリングの要点とイメージ図

・マテリアライズド ビュー

事前集計済みのデータを入れたり、重い処理をあらかじめ入れておいたり、データのサブセットを入れたりすると、I/O やコンピュートを減らせます。別のテーブルに集計済みのデータを入れてそれを参照する方法もありますが、マテリアライズド ビューならではの利点として、例えばメンテナンス フリーであること、クエリリライトしてくれるケースがあること、増分更新も可能なことなどが挙げられます。



マテリアライズドビューを使う基準

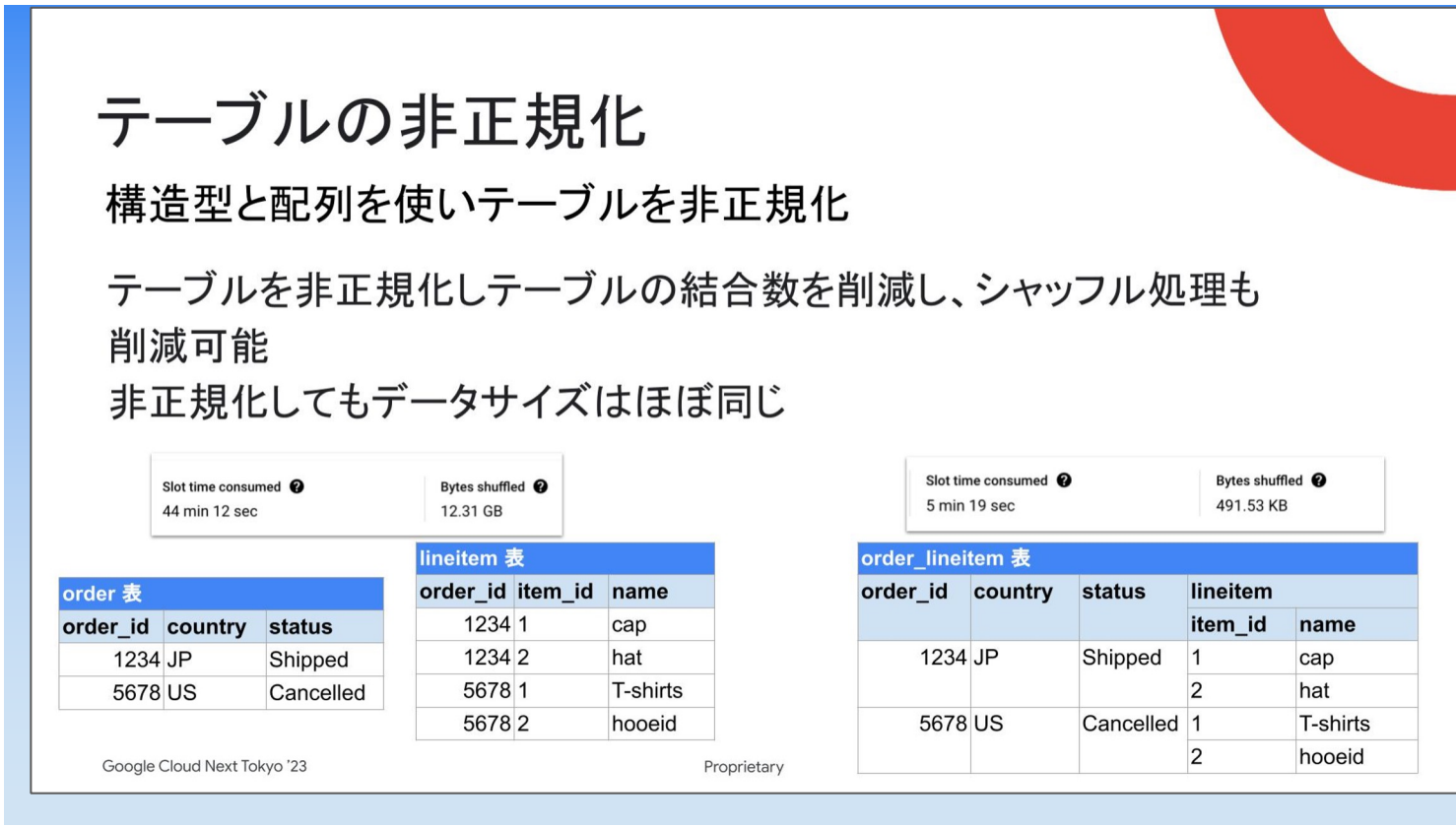
・SEARCH INDEX

読み込み量やスロット利用量の削減に効果があるテキスト インデックスです。STRING や NATIVE JSON に入っている文字列をトークン化し、そのトークンがどの位置にあるのかを持つインデックスであるため、カーディナリティの高い列に対する検索の高速化が可能です。また、トークン化して持つためログ分析のニーズにも対応します。

現在は以前と異なり、Where 句に SEARCH 関数を書かなければインデックスを使えないということはなくなりました。今は、＝、IN、LIKE、START_WITH にも利用でき、遅い SQL があれば SEARCH INDEX を作ることで、クエリを変更することなく改善できる可能性があります。

・テーブルの非正規化

BigQuery では構造型や配列もサポートしているため、テーブルを非正規化しながらもデータ量があまり変わらず格納できるメリットがあります。



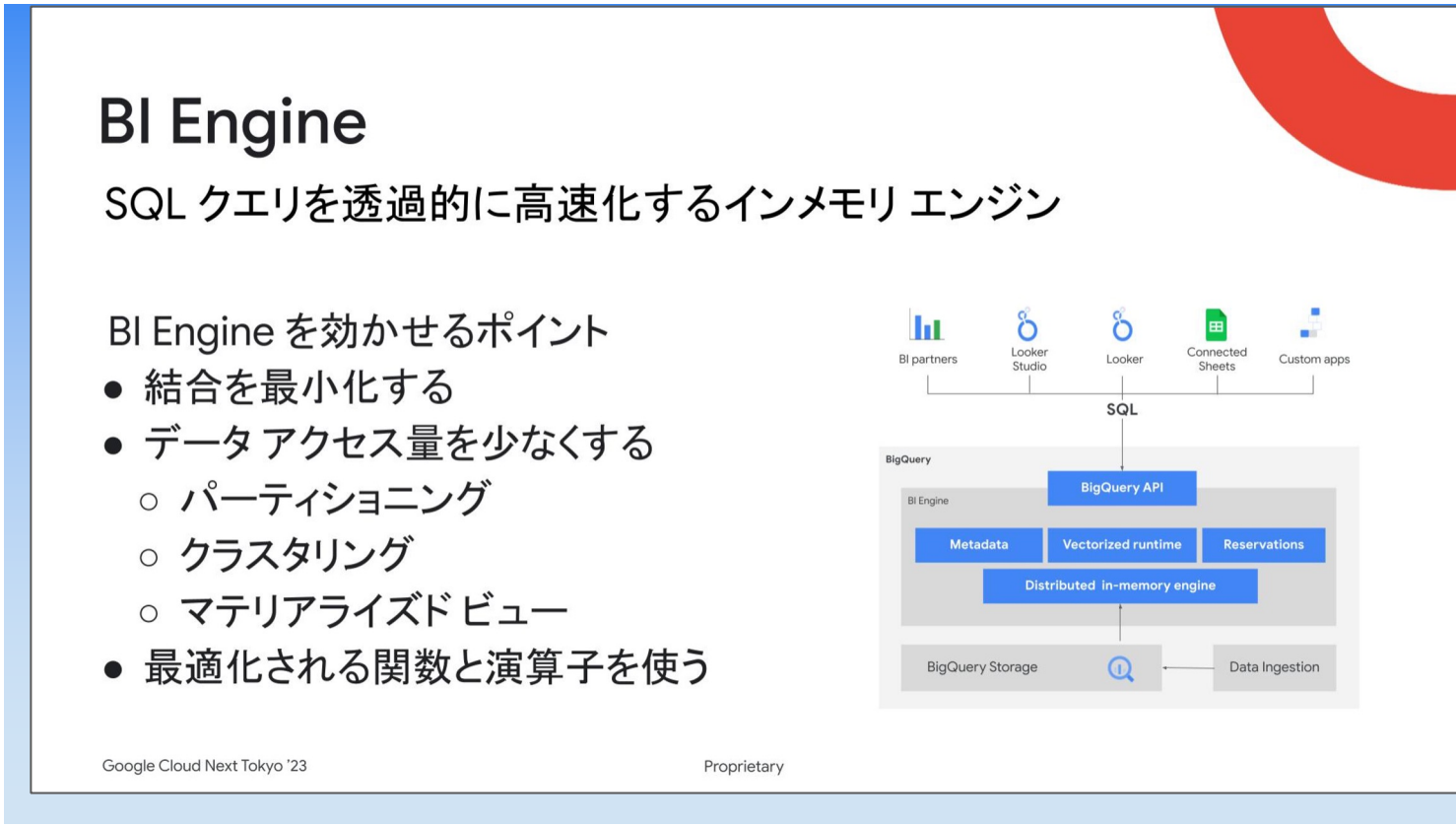
テーブルの非正規化の特徴とメリット

・主キーと外部キーで結合を最適化

BigQuery では最近、主キーと外部キーを定義できるようになりました。これはクエリ オプティマイザがより良いクエリプランを生成するために利用されます。内部結合解除、外部結合解除、結合順序変更といった効果があります。

- **BI Engine**

BigQuery にはインメモリの SQL エンジンである「BI Engine」があります。メモリのサイズを設定するだけでスケールを変えずに、クエリを透過的に高速化することが可能です。今遅いと感じているクエリが、設定だけで高速化できる可能性があります。もし使用して効果がない場合には、メモリを削除すればそれ以上はコストがかかりませんし、手軽に始めて手軽に止められるので試してみる価値があります。



BI Engine を効かせるポイント

・**Metadata Caching**

先ほど CMETA というメタデータ管理システムの話をしましたが、「Metadata Caching」は、BigLake テーブル用の CMETA ようなものととらえるとよいでしょう。BigLake テーブルは、「Cloud Storage」などのオブジェクト ストレージにあるファイルを BigQuery の表として利用する、いわゆる外部表と呼ばれるものです。

BigLake テーブルを使うと性能面のボトルネックが出るケースがあります。その中で、Metadata Caching では、ファイルをあらかじめキャッシングしておき、さらに Capacitor のように列の統計情報も取れるため、本当に必要なデータのみにアクセスすることで高速化できます。

現在、外部テーブルを使っていてファイル数が多くて重いという課題は、BigLake テーブルに変えて、Metadata Caching をオンにすることで解消できる可能性があります。

ここまで触れてきたように BigQuery は今でもバックエンドで性能改善を進めていますが、性能を意識して SQL を書き、テーブルを設計することで、さらに快適に利用でき、リソースも抑えられます。

もちろん、今回紹介した方法だけが全てではありませんが、いずれにしても「なるべく少ないリソースで結果にたどりつける」という考えを常に意識しながら書き方を心がけることで、さまざまなボトルネック解消につながっていくはずです。

参照リンク

1. BigQuery 製品紹介ページ
2. データ分析を加速する BigQuery 性能最適化のポイント アーカイブ動画視聴ページ

製品、サービスに関するお問い合わせ

goo.gl/CCZL78

Google Cloud の詳細については、上記 URL もしくは QR コードからアクセスしていただくか、同ページ「お問い合わせ」よりお問い合わせください。

Google Cloud