SYBASE TECHWAVE 2010

SQL ANYWHERE における MERGE 文とマテリアライズド・ビュー

ANIL K GOEL
DIRECTOR OF ENGINEERING
AUGUST 10, 2010

このプレゼンテーションの目的

- SQL Anywhere (SA) における MERGE 文のサポートを紹介する
  - 必要な理由 : ANSI SQL、SA の拡張
- マテリアライズド・ビューの配備について説明する
  - マテリアライズド・ビューとは
  - 複雑な DBMS の一部とすべき理由
  - 使用方法
  - どんなときに有効か、どんなときに回避すべきか
- SQL Anywhere におけるマテリアライズド・ビューのサポートを紹介する
  - SA 10 で導入、SA 11 と SA 12 で機能強化
  - 言語のサポート
  - 機能と制限事項
  - ベスト・プラクティス

謝辞

- 各リリースにおける優れた業績に対し、SQL Anywhere Query Processingチームに謝意を表します。
- SQL Anywhere は、まぎれもなく業界で最高のチーム!

- 本プレゼンテーションについては、Ani Nica 氏にご助力いただいたことを感謝いたします。

アジェンダ

- SQL Anywhere での MERGE 文
  - 必要な理由
  - ANSI SQL の MERGESA の拡張
- SQL Anywhere のマテリアライズド・ビュー
  - 定義と必要な理由
  - 長所と短所
  - 設計と作成の問題
  - 保守の方法と課題
  - クエリ・パフォーマンス向上のための配備
  - サーバのサポートと制限事項

MERGE 文の必要な理由

- 新しいデータのロードが必要
  - LOAD TABLE
  - INSERT [FROM SELECT]
  - 新しいローはすべて既存のテーブルに挿入される問題 1 : ロード前に新しいデータのクリーニングが必要な場合がある
  - 2 : 既存のテーブルのローが新しいデータで更新される場合がある
- 同じ文で INSERT 以上の柔軟な ELT (Extract-Load-Transform: 抽出-ロード-変換) 機能を使用する際に有効
  - 新規および既存のデータで複数の実行を回避
  - アプリケーションの複雑さを軽減
- 特定の入力ローについて、新しいローと既存ローのデータに応じて INSERT 以外のアクションを実行 (DELETE、UPDATE、SKIP など)
- SQL Anywhere 8で INSERT ON EXISTING のサポートが追加されたため、新しいローの挿入またはスキップ、既存ローの更新、エラーの生成が可能になった

MERGE 文の必要な理由

INSERT …

[ON EXISTING {

ERROR |

SKIP |

UPDATE …

}] …

- プライマリ・キーのあるテーブルに限定される
- 新しいローの挿入前に、同じプライマリ・キー値を持つローがすでにテーブルに存在するかどうかをサーバがチェックする
- 存在する場合、INSERT 以外のアクションを選択できる
- ANSI SQL の MERGE 文は、同じ機能の一部を、より一般的な形で提供

MERGE 文の必要な理由

- 同じ DML 文に ON EXISTING 以上の機能性を追加すると便利
  - INSERT に代わるアクションを提供する
  - 特定のローに対する挿入や更新を条件に基づいてスキップ
  - 既存ローの削除についてはどうか
  - 代替アクションに任意の条件を指定できる

MERGE 文の必要な理由

```
FOR each input row DO
    IF EXISTING THEN
        IF cond1 THEN SKIP
        ELSE IF cond2 THEN ERROR
        ELSE IF cond3 THEN DELETE
        ELSE UPDATE
    ELSE
        IF cond4 THEN SKIP
        ELSE IF cond5 THEN ERROR
        ELSE INSERT
```

MERGE 文の必要な理由

```
MERGE INTO T
USING (SELECT * FROM OPENSTRING( file 'c:¥techwave¥2010¥data1.txt')
        WITH( id int, qty int, cmt long varchar)  S) S
ON PRIMARY KEY
WHEN NOT MATCHED AND S.cmt = 'ignore' THEN SKIP
WHEN NOT MATCHED AND S.qty < 0  THEN RAISERROR
WHEN NOT MATCHED AND S.qty = 0 THEN INSERT VALUES( S.id, 1 )
WHEN NOT MATCHED THEN INSERT VALUES( S.id, S.qty )

WHEN MATCHED AND S.cmt = 'ignore' THEN SKIP
WHEN MATCHED AND S.cmt = 'new'  THEN RAISERROR
WHEN MATCHED AND T.qty + S.qty <= 0 THEN DELETE
WHEN MATCHED THEN UPDATE SET T.qty = T.qty + S.qty
```

ANSI SQL 2003 の MERGE 文

MERGE INTO target-object
USING source-object
ON merge-search-condition
merge-operation [merge-operation]

target-object: { target-table-name | target-view-name } [[AS] target-correlation]

source-object : { source-table-name | source-view-name } [[AS] source-correlation]
| (select-statement) [AS] source-correlation-name [using-column-list]

merge-search-condition : search-condition

merge-operation : WHEN MATCHED THEN match-action
| WHEN NOT MATCHED THEN not-match-action

match-action : UPDATE SET set-item, ...
not-match-action : INSERT [insert-column-list] VALUES (value, ...)

ANSI SQL 2003 の MERGE

- 文WHEN MATCHED と WHENNOT MATCHED のいずれか 1 つを指定する必要がある
- 実行に際して merge-operation の順序が重要
- **target-object のローごとに以下を行う**
  - 一致するローを検索する際、**source-object のローごとにマージ-検索**-条件が TRUE に評価されるかどうかを決定する
  - 一致する複数のローが見つかった場合にエラーを生成する
  - 一致するローが見つかった場合には、WHEN MATCHED 句 (存在する場合) で処理されるローのセットに、一致した 1 つのローを追加する
- 句が (存在する場合)、どの target-object ローにも一致しない source-object のローを処理する

ANSI SQL 2003 の MERGE 文

- 更新処理を実行する前に、一致するローと一致しないローのセットが生成される
- merge-operation が、指定した順序で処理される
- 各句は、独立した文として処理される
  - 該当するすべてのトリガが、対応する DML 更新文に対して通常どおり実行される
  - トリガ・アクションによって潜在的な副作用が発生する

ANSI SQL 2008 での MERGE の拡張

- 同じ文における複数の WHEN [NOT] MATCHED 句のサポートを追加
- 少なくとも 1 つの merge-operation が必要
- ただし、ANSI SQL 2003 と同じく、WHEN [NOT] MATCHED 句は任意に反復できる

MERGE INTO …

merge-operation [merge-operation …]

merge-operation :

WHEN MATCHED [AND search-condition] THEN match-action

| WHEN NOT MATCHED [AND search-condition] THEN not-match-action

ANSI SQL 2008 での MERGE の拡張

- WHEN [NOT] MATCHED を指定する順序が重要
  - 次の例で最後の 2 つの句は評価されない : 一致するローと一致しないローのセットが常に空になるため

WHEN MATCHED AND …. THEN …

WHEN NOT MATCHED AND … THEN …

WHEN MATCHED THEN …

WHEN NOT MATCHED THEN …

WHEN MATCHED AND *search-condition* THEN …

WHEN NOT MATCHED AND *search-condition* THEN …

ANSI SQL 2008 での MERGE の拡張

- ではどうするか
- WHEN MATCHED 句の AND 条件を評価して、一致するローを別々のサブセットに分割する
  - 各サブセットが異なる WHEN MATCHED ブランチによって処理される
  - 検索条件が TRUE に評価された最初の WHEN MATCHED 句が、一致する特定のローを処理する
  - AND 条件のいずれも TRUE に評価されない場合 (無条件の WHENMATCHED 句がない) や、複数の source-object ローが同じtarget-object ローに一致する場合には、一部に入力ローが破棄されることがある
- 同様に、一致しないローのセットは指定された WHEN NOT MATCHEDブランチ間で分割される

ANSI SQL 2008 での MERGE の拡張

- 同じ target-object ローが WHEN MATCHED 句のサブセットのいずれかに複数回出現する場合は、文の全体がエラーで失敗する
- 例 :

```
MERGE INTO     T
USING          S
ON   p0
WHEN MATCHED AND p1 THEN …
WHEN MATCHED AND p2 THEN …
WHEN MATCHED AND p3 THEN …
```

ANSI SQL 2008 での MERGE の拡張

S	T	p0	p1	p2	p3
s1	t1	T	Ⓣ	T	F
s2	t1	T	F	Ⓣ	T
s3	t1	T	Ⓣ	F	T

- カーディナリティ違反エラー

S	T	p0	p1	p2	p3
s1	t1	T	Ⓣ	T	F
s2	t1	T	F	Ⓣ	T
s3	t1	T	F	F	Ⓣ

オーケー → target-object ロー t1 は WHEN MATCHED AND p1 ブランチでのみ処理される

ANSI SQL 2008 での MERGE の拡張

- ローのサブセットが計算されると、指定された順序でブランチが処理される
- ブランチに指定されたアクションが、独立した文と同じように実行される
  - すべてのトリガは通常どおりに起動
  - ブランチの実行中に起動されるトリガが、以降のブランチの実行に影響する場合がある

SQL Anywhere での MERGE の拡張

- ANSI SQL の MERGE は INSERT ON EXISTING を一般化し、必要な機能を追加するが、
  - 冗長性が高いため、ブランチごとに INSERT と UPDATE のアクションを完全に、また別個に指定する必要がある
  - プライマリ・キーの検索条件がサポートされない
  - ERROR や SKIP など便利なアクションがなくなる
  - 便利なベルやホイッスルが追加される余地が残る

- SQL Anywhere の MERGE は、構文上使いやすい拡張機能を追加
  - アプリケーション開発者にとっての付加価値となる拡張機能を重視

SQL Anywhereでの MERGE の拡張

- target-object に SELECT 文を指定でき、ビュー名の場合と同様に動作する

target-object: …

| (select-statement) [AS] target-correlation-name

SQL Anywhereでの MERGE の拡張

- SA11 で強力かつ柔軟な ELT 機能を実装
  - クライアント・マシンからデータにアクセス
  - ソース・データは追加ソースから取得可能
    - プロシージャの結果セット
    - OPENSTRING() 機能を使用して外部ファイルから
  - MERGE の source-object として使用される前にデータを変換するための使いやすい構造体

source-object : …
| procedure

procedure :
procedure-name (procedure-syntax) [WITH (column-name data-type, ...)] [[AS] source-correlation-name]

SQL Anywhereでの MERGE の拡張

- ターゲット・カラムのリストを明示的に指定できる
  - source-object カラムに一致するようにカラムを並べ替えることができる

MERGE INTO target-object [into-column-list]

- WITH AUTO NAME を使用できる
  - 名前の比較に基づいてソース・カラムとターゲット・カラムを照合するときに便利

USING [WITH AUTO NAME] source-object

SQL Anywhere での MERGE の拡張

- 文の全体に適用されるクエリ･ヒントを指定できる

[OPTION (query-hint, ...)]

query-hint :
MATERIALIZED VIEW OPTIMIZATION option-value
| FORCE OPTIMIZATION
| option-name = option-value

SQL Anywhereでの MERGE の拡張

- merge-search-condition を指定してプライマリ・キー・カラムを基準に照合できる
  - スキーマの変更にアプリケーションを対応させることができる

merge-search-condition : …

| PRIMARY KEY

SQL Anywhereでの MERGE の拡張

- WHEN [NOT] MATCHED 句に便利なアクションを追加できる

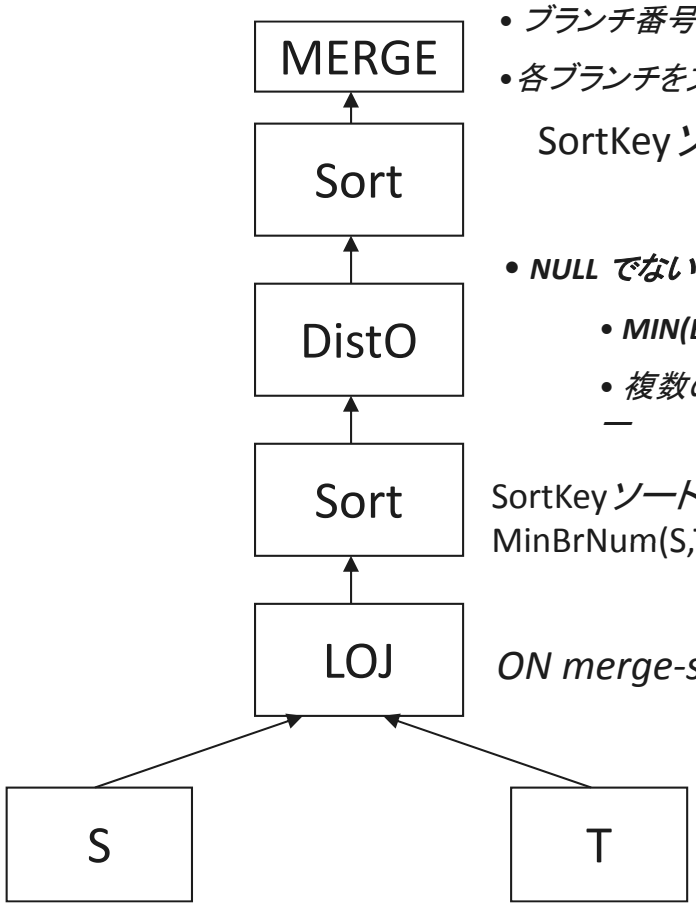
match-action : …
| DELETE
| RAISERROR [error-number]
| SKIP
| UPDATE [DEFAULTS { ON | OFF }]

not-match-action : …
| RAISERROR [error-number]
| SKIP
| INSERT

SQL Anywhereでの MERGE の実行

- 内部的な実行の詳細は変更に従う
  - 下位で使用して MERGE のセマンティックを説明する
  - グラフィカルなプランに表示
- WHEN [NOT] MATCHED の各ブランチに増分値を割り当てる
- 次の各タプルについて MinBrNum(S,T) を計算する
  - source-object S LEFT OUTER JOIN target-object T
    ON merge-search-condition
- merge-search-conditionが TRUE の場合は、
  - AND 条件が TRUE に評価された最初の WHEN MATCHED ブランチ番号を使用する
- ELSEそれ以外の場合は、
  - AND 条件が TRUE に評価された最初の WHEN NOT MATCHED ブランチ番号を使用する

SQL Anywhereでの MERGE の実行



- *ブランチ番号の昇順でローを処理*
- *各ブランチを文レベルの操作として扱う*

SortKey *ソート・キー*: BN

- ***NULL でない ROWID(T) ごとに***
  - ***MIN(BN) のみのローは維持***
  - *複数のローが残った場合にはカーディナリティ・エラー*

SortKey *ソート・キー*: ***BNとして、*** ROWID(T), MinBrNum(S,T) as BN

ON merge-search-condition

SQL AnywhereでのMERGEとトリガ

- merge-action に応じて通常どおりにトリガを起動
- トリガを作成する際、SA では同じトリガに複数のトリガ・イベント (INSERT, UPDATE, DELETE) を指定できる
- 1 つのトリガに複数のトリガ・イベントを指定することは、同じ本体でイベントごとに 1 つずつ複数のトリガを定義することに等しい
- したがって、merge-action リストに複数のアクション (INSERT/UPDATE/DELETE) が含まれている MERGE 文の中で同じトリガを複数回起動することができる

SQL Anywhere の MERGE と INSERT ON EXISTING

- INSERT ON EXISTING UPDATE も状況によっては引き続き有効だが、通常は MERGE で置き換えることができる
- SQL Anywhere の MERGE と INSERT ON EXISTING
  - SA の拡張では仕様から多くの冗長性が排除されていることに注意すること

SQL Anywhere の MERGE と INSERT ON EXISTING

```
// SA INSERT ON EXISTING (t2 には t2.c1 との重複がないと想定)
insert into t
on existing update
select * from t2

// ANSI 準拠の MERGE
merge into t
using t2
on t.c1 = t2.c1
when not matched then insert values( t2.c1, t2.c2 )
when matched then update set t.c2 = t2.c2

// SA 拡張の MERGE
merge into t
using t2
on primary key
when not matched then insert
when matched then update
```

SQL Anywhere の MERGE と INSERT ON EXISTING

```
// INSERT ... ON EXISTING
insert into t
on existing update
values(1, 11)

// ANSI 準拠の MERGE
merge into t
using (select 1 c1, 11 c2 ) as t2
on t.c1 = t2.c1
when not matched then insert values( t2.c1, t2.c2 )
when matched then update set t.c2 = t2.c2

// SA 拡張の MERGE
merge into t
using (select 1 c1, 11 c2) as t2
on primary key
when not matched then insert
when matched then update
```

Agendaアジェンダ

- SQL Anywhere での MERGE 文
  - Motivation必要な理由
  - ANSI SQL の MERGE
  - SA の拡張
- SQL Anywhere のマテリアライズド・ビュー
  - ➢ 定義と必要な理由
  - 長所と短所
  - 設計と作成の問題
  - 保守の方法と課題
  - クエリ・パフォーマンス向上のための配備
  - サーバのサポートと制限事項

マテリアライズド・ビュー: 定義

- ビュー: 一連のベース・テーブルに対して派生テーブルを定義する名前付きのクエリ
  - 非正規化、セキュリティなどさまざまな理由で使用され
  - 高いレベルでデータのビューが可能
  - サーバはビューをクエリする際にマクロ拡張 (インライン化とも言う) を実行する
  - 参照のたびに再計算する

マテリアライズド・ビュー: 定義

- マテリアライズド・ビュー: 結果セットがベース・テーブルとしてデータベースに格納されるビュー
  - 格納された結果セットを使用してクエリに応答できる
  - 参照のたびに再計算する必要がない
  - 基本的にはデータ・キャッシュであり、ベース・テーブルのデータのレプリカ
    - コストと効果はデータ・キャッシュに近い

マテリアライズド・ビュー: 必要な理由

- 例として使用する TPC-H スキーマ
  - 業界標準のアドホック、意思決定支援ベンチマーク
    - http://www.tpc.org/tpch/default.asp
  - 大量のデータに対する複雑なクエリ
- SQL Anywhere Server
  - SA 11.0.0.1441
    - Fixed 512M cache (“-ca 0 -c 512M”)
  - Dell Latitude 830 Laptop, Intel® Core™2 Duo 2.39GHz, 2G RAM

- 例として使用する TPCH SF1
  - 1GB 以下のデータとインデックス
  - テーブルのカーディナリティは以下のとおり:

LINEITEM	6,001,215
ORDERS	1,500,000
PARTSUPP	800,000
PART	200,000
CUSTOMER	150,000
SUPPLIER	10,000
NATION	25
REGION	5

マテリアライズド・ビュー: 必要な理由

- 各国のマネージャが国内で販売されている製品について知りたい場合
- それぞれが類似のクエリを発行する:

```
SELECT  DISTINCT p_name, p_size
FROM      Part, LineItem, Orders,
          Customer, Nation
WHERE   N_Name = 'Canada' AND
          n_nationkey = c_nationkey AND
          c_custkey = o_custkey AND
          o_orderkey = l_orderkey AND
          l_partkey = p_partkey
FOR READ ONLY;
```

- ~22.7 seconds最速の実行時間: 22.7秒未満

マテリアライズド・ビュー: 必要な理由

- ベスト・プランでは Orders、LineItem、Part のジョインが必要
- <LineItem, Orders> のジョインを事前に計算して格納する
- キャッシュ・データを使用して元のクエリに応答できる

```
CREATE MATERIALIZED VIEW LineItem_Orders_mv AS
    SELECT   o_custkey, l_partkey
    FROM     Orders, LineItem
    WHERE    o_orderkey = l_orderkey
```

- 計算時間: 27.0 秒
- ストレージ: 8 バイトで 600 万ロー
- 事前計算された結果をどのように使用するか

マテリアライズド・ビュー: 必要な理由

- キャッシュされたジョインデータを使用するようにクエリを変更:

```
SELECT  DISTINCT p_name, p_size
FROM    Part, ~~LineItem, Orders ,~~ LineItem_Orders_mv,
        Customer, Nation
WHERE   N_Name = 'Canada' AND
                n_nationkey = c_nationkey AND
                c_custkey = o_custkey AND
                ~~o_orderkey = l_orderkey AND~~
                l_partkey = p_partkey
```

- 実行時間: 3.5 秒
- 元のクエリを変更することは望ましくない:
  - 時間と領域のかね合いは発展の過程で変化する
  - データ・キャッシュを再定義する柔軟性が必要
  - アプリケーションを変更したくない、または変更できない
- ではどんな方法があるか

マテリアライズド・ビュー: 必要な理由

- サーバは「ともかくも実行する」必要がある
  - キャッシュ・データを自動的に使用する
  - 指定されたクエリを既存のマテリアライズド・ビューとアプリケーション要件に照らして分析する
  - クエリの一部に、その部分を再計算するのではなくキャッシュ・データを使用して応答できるかどうかを確認する
  - コストベースの分析を実行し、キャッシュ・データを使用する方が有効かどうかを決定する
  - プロセスをアプリケーションに対して完全に透過的にする
- SQL Anywhereはこれを確実に実行
- 実行時間: 3.5 秒
  - サーバはマテリアライズド・ビューを利用する
  - クエリをリライトするときと同じ実行プラン

マテリアライズド・ビュー： 必要な理由

- ほかに事前計算の必要または可能なものはあるか
- Lineitem と Part のジョインを事前計算するか

```
CREATE MATERIALIZED VIEW LineItem_Part_mv AS
    SELECT   p_name, p_size, l_orderkey
    FROM     LineItem, Part
    WHERE    l_partkey = p_partkey;
```

- 計算時間： 38 秒
- ストレージ： 12 バイトで 600 万ロー
- 実行時間： 11.5 秒

マテリアライズド・ビュー： 必要な理由

- Lineitem、Orders、Part のジョインを事前計算するか

```
CREATE MATERIALIZED VIEW LineItem_Orders_Part_mv AS
    SELECT   p_name, p_size, o_custkey
    FROM     Orders, LineItem, Part
    WHERE    o_orderkey = l_orderkey AND
             l_partkey = p_partkey
```

- 計算時間： 35.6 秒
- ストレージ：可変サイズで 600 万ロー (8 バイト + 名前)
- 実行時間： 7.6 秒

マテリアライズド・ビュー： 必要な理由

- それぞれ異なる 3 種類の事前計算ビュー
  - 計算時間
  - ストレージ要件
  - 元のクエリに対する影響
  - これらのいずれを使用しても元のクエリに応答できるため、何も問題はない
  - 他のクエリに効果的なものは一部、またはなし
  - 最小のシステム負荷で総合的に最大の効果を得るには最小の種類のビューを事前計算することが望ましい
- 最終的な効果はどうか
  - マテリアライズド・ビューを使用するとクエリのパフォーマンスが大幅に向上
  - マテリアライズド・ビューの選択には慎重な分析が必要

アジェンダ

- SQL Anywhere での MERGE 文
  - 必要な理由
  - ANSI SQL の MERGE
  - SQL Anywhere の拡張
- SQL Anywhere のマテリアライズド・ビュー
  - 定義と必要な理由
  - ➢ 長所と短所
  - 設計と作成の問題
  - 保守の方法と課題
  - クエリ・パフォーマンス向上のための配備
  - サーバのサポートと制限事項

マテリアライズド・ビュー： 長所と短所

- マテリアライズド・ビューの長所
  - 事前計算された結果セットを使用してクエリ結果を全部または部分的に得るため、クエリのパフォーマンスが向上する
  - ベース・テーブルのローに対する競合が軽減されるため同時性の向上に有効
- マテリアライズド・ビューの短所
  - 必要なストレージが増える
  - ベース・テーブルの更新時に保守する必要がある
  - 格納されているデータが、基本となるベース・テーブルのデータと同期されないと古くなる
    - 同じクエリで異なる結果が返されることがある

マテリアライズド・ビュー： 長所と短所

- マテリアライズド・ビューを使用するシステムの設計には、コストと効果の分析が必要
- 以下のようなサンプルの命題を考慮すること:
  - マテリアライズド・ビューを作成して効果のあるクエリのセットは何か
  - マテリアライズド・ビュー用に格納されるデータが古くなってもよいか
  - データはどの程度まで古くなることが許容されるか
  - 同じクエリで異なる結果を返すことはできるか
  - クエリ・パフォーマンスが向上する可能性の方が、マテリアライズド・ビューによるストレージや保守のコストより重要か
    - 最新データの重要性はどのくらいか。クエリ結果の高速さはどうか
- これらに対する回答が設計上の決定に大きく影響する

アジェンダ

- SQL Anywhere での MERGE 文
  - 必要な理由
  - ANSI SQL の MERGE
  - SQL Anywhere の拡張
- SQL Anywhere のマテリアライズド・ビュー
  - 定義と必要な理由
  - 長所と短所
  - ➢ 設計と作成の問題
  - 保守の方法と課題
  - クエリ・パフォーマンス向上のための配備
  - サーバのサポートと制限事項

マテリアライズド・ビューの作成

- コストと効果の比を最適化するためにマテリアライズド・ビューを作成する
  - 「フリーサイズ」のような分析は行わない
- どんなビューを作成するか考慮すべき点:
  - クエリによるシステム負荷
    - クエリの定義と頻度。以下の点から
      - 頻繁に実行されるクエリと高負荷のクエリ
      - 応答時間の要件が不可欠である高負荷のクエリ
  - クエリ・パフォーマンスが向上する可能性
  - マテリアライズド・ビューとインデックスの領域要件
  - 複数のクエリに有効な共通のビューを見つける
    - 同じクエリで複数のビューを利用できる
    - 1 つのビューを複数のビューに分割すると他のクエリに効果的な場合がある

マテリアライズド・ビューの作成

- どんなビューを作成するか考慮すべき点:
  - 実際に利用できるビューに関するサーバ上の制限
  - ビューでの集計の複雑さ
    - 基本的な関数は実体化すべき
      - 例: SUM と COUNT から AVG を得られる
  - アプリケーションの更新パターン
    - 更新頻度の高いベース・テーブルに対してマテリアライズド・ビューを作成すると、保守コストが許容範囲を超えることがある
- マテリアライズド・ビューの作成が理想的なケース:
  - ベース・テーブルの更新頻度が低～中程度
  - ジョインが含まれる
  - 大量のデータに対して集計やグループ化を行う場合
  - 実時間で計算すると負荷が高すぎる場合

マテリアライズド・ビューの作成

- SQL Anywhere 12 でのマテリアライズド・ビューの制限
  - 定義に含められないもの:
    - 選択リストでの「*」
    - 他のビューに対するネストされた参照
      - » 今後の実装を予定
    - テンポラリ・テーブルまたはプロキシ・テーブルへの参照
    - CURRENT USER などの特殊な構造体を含む変数
      - » 式はすべて確定的でなければならない
    - ストアド・プロシージャとユーザ定義関数の呼び出し
    - SYS が所有しているシステム・オブジェクトへの参照
    - TSQL
    - XML 句
  - レプリケーション関係の構造体がない
  - RI およびその他の制約、トリガ、アーティクルがない
    - 非テキストのインデックスの作成は許可される

アジェンダ

- SQL Anywhere での MERGE 文
  - 必要な理由
  - ANSI SQL の MERGE
  - SQL Anywhere の拡張
- SQL Anywhere のマテリアライズド・ビュー
  - 定義と必要な理由
  - 長所と短所
  - 設計と作成の問題
  - ➢ 保守の方法と課題
  - クエリ・パフォーマンス向上のための配備
  - サーバのサポートと制限事項

マテリアライズド･ビューの保守

- マテリアライズド･ビューに関する一般的な概念
  - SQL Anywhere の実装に限定されない
- 最初は基本となるクエリを実行して計算する
- ベース･テーブルの更新に応じて保守が必要
  - データベースの変更状態と一致するようにマテリアライズド･ビューを更新する
- ビューの保守に関する 3 つの判断要因
  - 保守のタイミング
  - 保守の方法
  - 保守の主体

マテリアライズド・ビューの保守：タイミング

- ベース・テーブルのデータ変更をマテリアライズド・ビューのデータにいつ伝播するか
- 即時：同じトランザクションで伝播
- 延期：以降の別のトランザクションで伝播
  - 遅延：可能なかぎり、たとえばビューにクエリが発行されるまで遅延させる
    - 通常は現実的ではない
  - オンデマンド：定期的、バッチ処理
- アプリケーションの設計とセマンティックに大きく影響

マテリアライズド・ビューの保守：タイミング

- 即時保守
  - 更新トランザクションの一部としてビューを更新する
  - ベース・テーブルでの変更をビュー・データに伝播する
  - 基本となるベース・テーブルとの一貫性を維持する必要がある
  - ビュー・データが古くならない
  - ロック競合やデッドロックが発生する確率は高くなることがある
    - 更新トランザクションの同時性が損なわれる
  - アプリケーションへの影響：
    - クエリ結果が最新
    - クエリ実行が高速
    - ベース・テーブルの更新が低速

マテリアライズド・ビューの保守：タイミング

- 遅延 (ジャスト・イン・タイム) 保守
  - ビューは更新せずにベース・テーブルに変更を適用する
    - 増分更新が可能な場合には変更をログに記録できる
  - クエリ時にビューを使用して結果を提供できる:
    - 最新状態であればビューを即時に使用する
    - 最新状態でなければ、別の同期トランザクションを使用してログ上の変更を適用するかビューを再計算する。クエリの実行は待機する
  - アプリケーションへの影響:
    - クエリ結果が最新
    - クエリ実行は高速または低速
      - ビューを使用せずにクエリを実行するよりは遅くなる可能性が高い
    - ベース・テーブルの更新が高速
  - 問題が多く、頻繁には利用されない
    - トランザクション上の課題
    - クエリ実行が低速になることよりクエリ実行が一貫性を欠くことが問題

マテリアライズド・ビューの保守：タイミング

- 延期、オンデマンドの保守
  - ビューは更新せずにベース・テーブルに変更を適用する
    - 増分更新が可能な場合には変更をログに記録できる
  - 独立した非同期プロセスを使用してビューを更新する
    - 通常はビューを再計算する
    - ログに記録されたベース・テーブルの変更を使用できる
  - スケジュールを実装すれば更新を自動化できる
    - 更新の頻度は、リソースと最新さのバランスのかね合いで決まることが多い
  - アプリケーションへの影響:
    - クエリ結果は最新のことも古いこともある
      - 新しさはアプリケーション側で制御できるのが普通
    - クエリ実行が高速
    - ベース・テーブルの更新が高速
  - 古いクエリ結果が許容される場合にのみ有効

マテリアライズド・ビューの保守：方法

- マテリアライズド・ビューのデータを更新して、ベース・テーブルの新しい状態を反映する方法
- 再構築:
  - ビューを再計算する
- インクリメンタル:
  - 再計算せずに個々の更新を適用する
  - 使用できない場合もある
- 多くは効率上の問題
- 実際には、ビューの即時保守は常にインクリメンタル

マテリアライズド・ビューの保守：方法

- インクリメンタルなビューの保守
  - ベース・テーブルに対して行われた個々の変更を調べる
  - テーブルのスキーマとビュー定義を分析する
  - 既存の制約を考慮する
  - ベース・テーブルの変更を、再計算なしでビューに適用できる形に変換する公式を導出する
  - すべてのビューで可能とは限らない
    - ビュー定義に対して制限がかかる

マテリアライズド・ビューの保守：方法

- データベースに以下のマテリアライズド・ビューが存在し、Lineitemテーブルの現在の状態を反映すると想定する

```
CREATE MATERIALIZED VIEW mv_l AS
SELECT      l_partkey, l_suppkey, count( * ) num_orders
FROM  Lineitem
GROUP BY l_partkey, l_suppkey
```

- l_partkey と l_suppkey のどちらでも NULL は許可されない

マテリアライズド・ビューの保守：方法

```
SELECT *
FROM mv_l
WHERE l_suppkey in( 8315, 8759, 3287) AND
      l_partkey in (143286, 11255)
```

- 上記のクエリはビュー mv_l から次の 2 つのローを返す

l_partkey	l_suppkey	Num_orders
143286	3287	8
143286	8315	8

マテリアライズド・ビューの保守：方法

- lineitem に新しいローを挿入したい
- INSERT INTO lineitem(…, l_partkey, l_suppkey, …)
  VALUES (…, 1054181, 143286, 8315, 4, …)
- 再計算せずに、ビューへの影響を判断することは可能か
- 状況:
  - ビュー・クエリには、l_partkey,l_suppkey に対する GROUP BY が含まれる → Lineitem への挿入の影響を受けるのはビューの 1 つのローだけ
  - グループ (143286, 8315) はすでに存在する → カウントを増分

l_partkey	l_suppkey	Num_orders
143286	3287	8
143286	8315	8 → 9

マテリアライズド・ビューの保守：方法

- INSERT INTO lineitem(…, l_partkey, l_suppkey, …)
  VALUES (…, 1054181, 11255, 8579, 5, …)

- グループ (11255, 8579) は存在しない⇒カウント 1 で新しいローを挿入

l_partkey	l_suppkey	Num_orders
11255	8579	1
143286	3287	8
143286	8315	9

マテリアライズド・ビューの保守：方法

- DELETE FROM lineitem
  WHERE l_orderkey = 1054181 AND l_linenumber = 5
- 古い l_partkey, l_suppkey の値： 11255, 8579
- カウント 1 のグループが存在する□ローを削除

l_partkey	l_suppkey	Num_orders
~~11255~~	~~8579~~	~~1~~
143286	3287	8
143286	8315	9

マテリアライズド・ビューの保守：方法

- UPDATE lineitem SET l_suppkey = 3287
  WHERE l_orderkey = 1054181 AND l_linenumber = 4
- l_partkey, l_suppkey の値:
  - 旧: 143286, 8315
  - 新: 143286, 3287
- どちらのグループも存在する□カウントを調整する

l_partkey	l_suppkey	Num_orders
143286	3287	8 → 9
143286	8315	9 → 8

マテリアライズド・ビューの保守：主体

- マテリアライズド・ビュー・データの更新は何が始めるか

- サーバにより
  - 自動的に

- ユーザにより
  - 明示的に
    - オンデマンド
  - 暗黙的に
    - スケジュールに基づいて
    - ジャスト・イン・タイム

マテリアライズド・ビューの保守

- サーバは 1 つ以上のポリシーを実装する必要がある
  - 選択肢の組み合わせによっては不可能、非現実的、無効な場合がある
    - たとえば、ビューの再構築による即時保守は実行できない
- ポリシーはアプリケーションに影響する:
  - クエリ結果のセマンティック
    - 最新:　最新かつ一貫
    - 古い:同期されておらず、場合によっては矛盾
  - クエリのパフォーマンス
  - 更新のパフォーマンス
- アプリケーション設計ではマテリアライズド・ビューごとにポリシを指定する必要がある

	用語	*タイミング*	*方法*	*主体*
MS SQL Server	インデックス・ビュー	即時	インクリメンタル	システム
DB2	Summary Tables (staging tables)	即時 遅延	インクリメンタル 再構築	システム ユーザ
Oracle	マテリアライズド・ビュー	遅延	インクリメンタル 再構築	システム ユーザ
SQL Anywhere	マテリアライズド・ビュー	即時 遅延	インクリメンタル 再構築	システム ユーザ

マテリアライズド・ビューの保守：SQL Anywhere

- SQL Anywhere 10 から遅延保守を実装
  - 手動マテリアライズド・ビュー
  - 保守のタイミング：延期、オンデマンド
  - 保守の方法：再構築
    - 今後の予定：インクリメンタル (未定)
  - 保守の主体：ユーザ
- SQL Anywhere 11、SQL Anywhere 12 で即時保守のサポートを追加
  - 即時マテリアライズド・ビュー
  - 保守のタイミング：即時
  - 保守の方法：インクリメンタル
  - 保守の主体：サーバ

SQL Anywhere の即時ビュー : 保守

- 既存の即時ビュー (iMV) はベース・テーブル (T) を参照する

T:

```
CREATE TABLE T(
 id    int not null,
 quantity     int not null,
 … )
```

iMV:

```
CREATE MATERIALIZED VIEW iMV AS
SELECT id, count(*) count, sum( quantity ) qty, …
FROM T
GROUP BY id
```

```
CREATE UNIQUE INDEX imv_id ON iMV( id )
```

SQL Anywhere の即時ビュー：保守

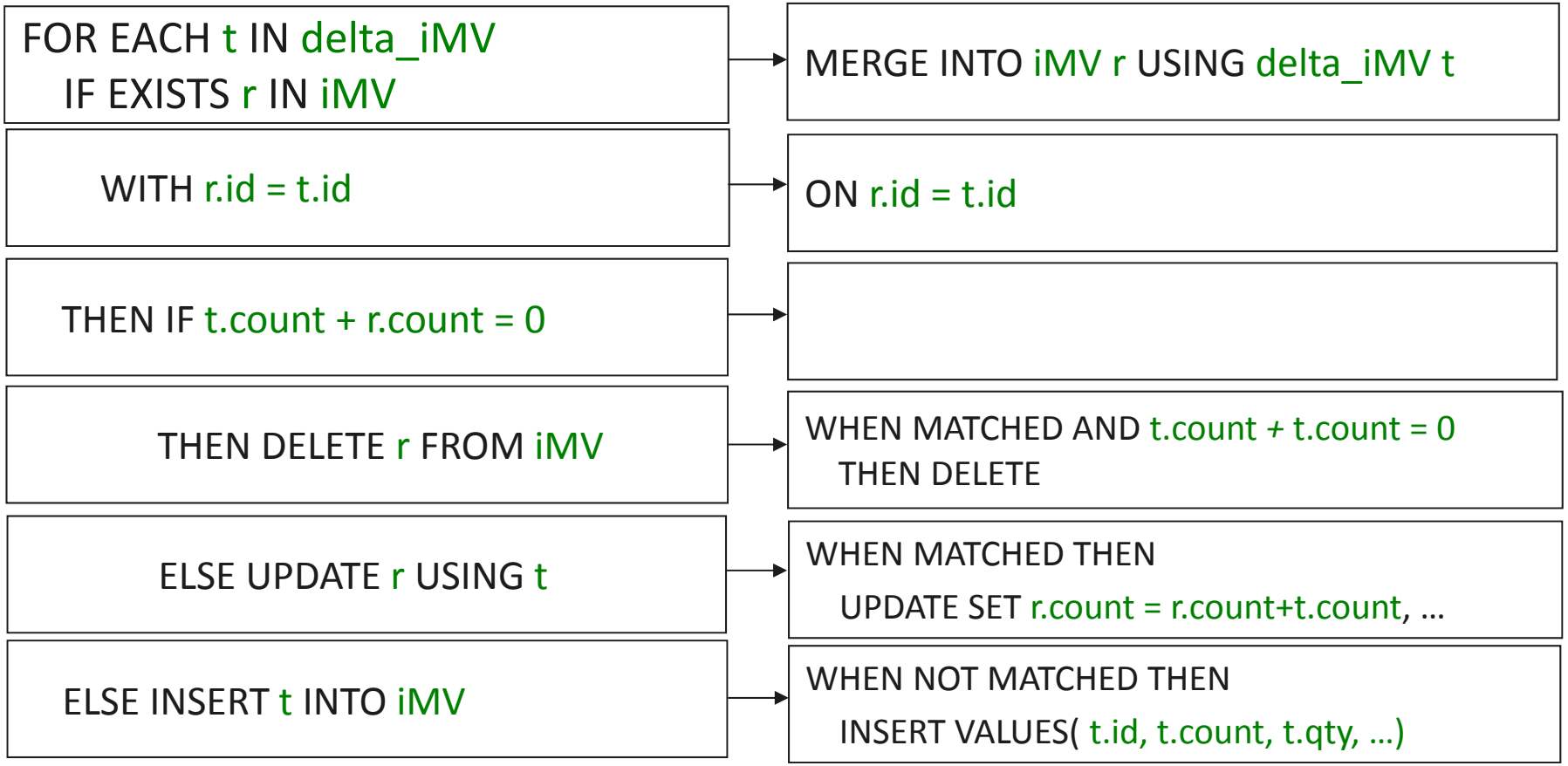
- ユーザは T に対して DML 更新文を実行する
- SQL Anywhere は自動的かつ即時に iMV を実行する必要がある
- 3 つのフェーズで処理される
- 更新フェーズ：T に対してユーザ更新を実行する
  - delta_T の計算：更新されたローの変更前と変更後のイメージ
- 伝播フェーズ：delta_T から delta_iMV を計算する
  - iMV には更新が必要
  - 各タプルに id、count (+/-)、qty(+/-)… が含まれる
- iMV適用フェーズ：delta_iMV を使用して iMV を更新する

SQL Anywhere の即時ビュー: 保守

- 適用フェーズ

```
FOR EACH t IN delta_iMV
    IF EXISTS r IN iMV
     WITH r.id = t.id
    THEN IF t.count + r.count = 0
     THEN    DELETE r FROM iMV
     ELSE UPDATE r USING t
    ELSE INSERT t INTO iMV
```

SQL Anywhere の即時ビュー：保守



SQL Anywhere の即時ビュー：保守

- T での DML 更新によりワーク・テーブルに *delta_T を移植*
  - T のローの前後に追加する

- 依存即時ビューを持つベース・テーブル T ごとに内部文レベルの AFTER がトリガされる
  - オンデマンドで透過的に保守される
  - 一時的に、メモリ内のみ
  - 動的：T に対する DML 更新の実行時に作成、維持される
  - 特定の DDL 操作中に破棄される
  - delta_T を delta_iMV に伝播し、T を参照する各 iMV に *delta_iMV*を適用する SQL 文

SQL Anywhere の即時ビュー：制限事項

即時ビューの作成には、4 段階のプロセスが必要

1. マテリアライズド・ビュー定義を作成する
   - 新規作成されるマテリアライズド・ビューはすべて手動
2. 少なくとも 1 つの一意なインデックスを次の条件でビューに作成する
   - 外部ジョインのない iMV に対してインデックス・カラムは NULLにできない
   - 外部ジョイン iMV に対し NULLS NOT DISTINCT を使用する一意なインデックス
   - グループ化されたクエリの場合に、インデックス・カラムは集計関数を参照しない
3. ALTER VIEW を使用し、ビューを即時として宣言する
4. ビューを最初に初期化すると、その後はサーバが自動的に保守する

SQL Anywhere の即時ビュー : 制限事項

- iMV 定義には 1 つのクエリ・ブロックを含める必要がある
  - SPJ, SPOJ
  - Grouped-SPJ, Grouped-SPOJ
    - No HAVING clause
    - 選択リストには COUNT(*) を含める必要がある
- ビュー定義に含められないもの : :
  - GROUPING SETS, CUBE, ROLLUP の各句
  - DISTINCT (代わりに GROUP BY を使用できる)
  - ローを制限する句
  - セルフジョインと再帰ジョイン
  - 集計関数に対する複雑な式

SQL Anywhere の即時ビュー：制限事項

- 依存即時ビューが存在する場合、ベース・テーブルに対する一部の更新文は使用できない
- TRUNCATE TABLE: 代替手段として DELETE FROM を使用できる
- LOAD TABLE: 代替手段として OPENSTRING() で INSERT または MERGE を使用できる
- 同じ即時ビューで参照される複数のベース・テーブルを更新する文
  - CREATE MATERIALIZED VIEW V … FROM T,R …
  - CREATE UNIQUE INDEX …
  - ALTER MATERIALIZED VIEW V REFRESH IMMEDIATE
  - REFRESH MATERIALIZED VIEW V
  - Not permitted: UPDATE T, R SET T.X = …, R.Y = …, …
- 即時ビューをトランケートし、禁止されている文を実行してビューを再初期化できる

SQL Anywhere の即時ビュー: 同時性

- 3ビュー保守の適用フェーズは分離レベル 3 で実行
  - iMV のローに更新
  - iMV で参照される他のベース・テーブルからローを読み取り
- 同時性の意味
- (内部)即時ビューの保守中にエラーが発生すると、トリガ側のDML更新が拒否される

アジェンダ

- SQL Anywhere での MERGE 文
  - Motivation必要な理由
  - ANSI SQL の MERGE
  - SQL Anywhere の拡張
- SQL Anywhere のマテリアライズド・ビュー
  - 定義と必要な理由
  - 長所と短所
  - 設計と作成の問題
  - 保守の方法と課題
  - ➢ クエリ・パフォーマンス向上のための配備
  - サーバのサポートと制限事項

マテリアライズド・ビューの使用：最適化

- クエリのパフォーマンス向上を図るために、オプティマイザでマテリアライズド・ビューを使用することができる
- クエリで直接マテリアライズド・ビューを参照できる
  - ビューがベース・テーブルのように扱われる
- クエリは変更されずに維持され、オプティマイザが
  - ビュー・マッチングを実行して、正規化したクエリの結果を全部または部分的に得ることができるマテリアライズド・ビュー候補を見つける
  - マテリアライズド・ビューありのプランと、なしのプランについてコストベース分析を行う

マテリアライズド・ビューの使用：最適化

- 特定のクエリでマテリアライズド・ビューが評価されるために満たすべき必要な条件
  - ビューがサーバによる使用と、最適化での使用に有効
  - ビューの結果セットがディスク上で実体化されている
  - ビューがマッチングに必要なオプティマイザ要件を満たしている
  - 特定のデータベース・オプションが適切な値である
  - ビューの現在の状態が古さの要件を満たしている
    - 即時ビューは常に最新状態

マテリアライズド・ビューの使用：最適化

SQL Anywhere でのビュー・マッチングで評価されるマテリアライズド・ビュー定義の制限

- 手動マテリアライズド・ビューの作成より制限が厳しい
  - クエリがマッチングに適さないビューを直接参照することがある
- 単一のクエリ・ブロック
- 含まれないも:
  - GROUPING SETS, CUBE, ROLLUP
    - » 今後の実装を予定
    - ローを制限する句 (TOP … [START AT …], FIRST)
      - » 今後、制限付きで実装を予定
    - DISTINCT
      - » GROUP BY を使用
    - 完全な外部ジョイン、セルフジョインと再帰ジョイン

マテリアライズド・ビューの使用：最適化

- SQL Anywhere でのビュー・マッチングの制限 (続き)
  - スナップショット・アイソレーションでは、クエリのトランザクションを開始する前に手動ビューが更新されている必要がある
    - ベース・テーブルと同様にバージョニングされるので、即時ビューでは問題にならない

マテリアライズド・ビューの使用：最適化

- マテリアライズド・ビューがクエリ・ブロック (の一部) を満たすことができるかどうか、ビュー・マッチングのアルゴリズムが判定する
- 2 つの最適化フェーズで実行されるビュー・マッチング
- 最適化前のフェーズ
  - クエリ・ブロックごとに候補のマテリアライズド・ビューを見つける
  - 以下のビューを探す
    - クエリ・ブロックでベース・テーブルの (最大限の) サブセットを対象とする
    - オプティマイザ要件を満たす
    - 対象のベース・テーブルから、クエリ・ブロックに必要なローの上位セットを含む
      - 対象のベース・テーブルに対して述部の範囲を提供するビュー
      - ビュー・カラムで対象のベース・テーブルに関する式をすべて計算できる
  - クエリ・ブロックはリライトされず、コストはまだ評価されない

マテリアライズド・ビューの使用：最適化

- 列挙フェーズ
  - コスト分析でベスト・プランを選択
  - 候補のマテリアライズド・ビューは直接参照されるベース・テーブルと同様に扱われる
  - マテリアライズド・ビューを評価する際、元のクエリのうちビューの対象となった一部を置き換える
    - ビューの対象となるベース・テーブルへの参照を縮小する
    - クエリ式を置き換えてビュー・カラムを参照する
    - 置き換えられるクエリ・ブロックによって返されるローのみを反映するようにビューの述部を調整する
      - 同一の述部は破棄できる
      - 含まれる述部を調整してビューに適用する
  - 詳細については、アプリケーション・プロファイリング、グラフィカルなプラン、インデックス・コンサルタントを参照

マテリアライズド・ビューの使用：データベース設計

- マテリアライズド・ビューをクエリに使用するのは、通常のビューを使用する場合とは異なる
  - クエリにおける通常のビュー参照は拡張され、基本となるビュー定義に置き換えられる
  - ユーザ・クエリの一部は縮小され、サーバによってマテリアライズド・ビュー参照に置き換えられる
  - では、非正規化とアクセス制御は断念するか
    - 断念することはできない
    - 今までどおり通常のビューを定義して参照する
    - 拡張されたクエリに対してマテリアライズド・ビューを設計する
    - マテリアライズド・ビューで縮小と置き換えを行う前に、サーバはユーザ・クエリで通常のビュー参照を拡張する

マテリアライズド・ビューの使用：データベース設計

- マテリアライズド・ビューの 2 大グループ
  - Select-Project-Join (SPJ) ビュー、Select-Project-OuterJoin (SPOJ) ビュー
  - Grouped-SPJ ビュー、Grouped-SPOJ ビュー

- SPJ、SPOJ ビュー：
  - 前述の必要な理由で検討したジョインの例
  - ベース・テーブルで頻繁にアクセスされるパーティションを実体化
    - 例：特定の期間における全トランザクションのレコード
    - 残りのクエリで使用する小さいテーブル
    - 同時性の向上
    - 複数のディスク・ドライブの使用が改善される可能性

マテリアライズド・ビューの使用：データベース設計

- Grouped-SPJ、Grouped-SPOJ ビュー
  - パフォーマンスへの影響が大きい可能性がある
  - 頻繁または高負荷なクエリで集計に使用する
  - ビューで基本的な集計関数を使用する
    - AVG(*) の代わりに SUM(*) と COUNT(*) を使用する
    - オプティマイザはさらに複雑な集計を計算できる
    - 複数のクエリでビューの有効性が増す

- マテリアライズド・ビューに追加のインデックスを作成できる
  - 複数のテーブルからジョインされたデータもインデックスが可能

マテリアライズド・ビューの使用：データベース設計

TPCH Q19:

```
SELECT      sum(l_extendedprice* (1 - l_discount)) as revenue
FROM LineItem, Part
WHERE   (   l_partkey = p_partkey
        and p_brand = 'Brand#12'                        -- :1
        and p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        and l_quantity >= 1 and l_quantity <= 1 + 10-- :4
        and p_size between 1 and 5
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON‘
            )
       OR  (   l_partkey = p_partkey
        and p_brand = 'Brand#23‘                        -- :2
        and p_container in ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        and l_quantity >= 10 and l_quantity <= 10 + 10      -- :5
        and p_size between 1 and 10
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON‘
            )
        OR (   l_partkey = p_partkey                    -- :3
        and p_brand = 'Brand#34'
        and p_container in ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        and l_quantity >= 20 and l_quantity <= 20 + 10      -- :6
        and p_size between 1 and 15
        and l_shipmode in ('AIR', 'AIR REG')
        and l_shipinstruct = 'DELIVER IN PERSON'
            )
```

マテリアライズド・ビューの使用：データベース設計

- Q19 の実行時間：17.3 秒
- パフォーマンスを改善するマテリアライズド・ビューを設計したい
- マテリアライズド・ビューにはクエリで使用されるローの上位セットを含める必要がある
  - Q19 の計算に必要なローをすべて含める必要がある
  - Q19 でまったく不要なローは破棄できる
  - できるかぎり多くの集計を実行する
- ビュー定義は、Q19 で最も負荷の高い、2 つの大きいテーブルのジョイン操作から開始する必要がある
  - FROM LineItem, Part
    WHERE l_partkey = p_partkey

マテリアライズド・ビューの使用：データベース設計

- Q19 でまったく不要なローを削除する
  - AND l_shipmode in ('AIR', 'AIR REG')
  - AND l_shipinstruct = 'DELIVER IN PERSON’
  - AND p_container in (

    'SM CASE', 'SM BOX', 'SM PACK', 'SM PKG',

    'MED BAG', 'MED BOX', 'MED PKG', 'MED PACK',

    'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG‘)
- Q19 には選択したローのセットに対する SUM が含まれる
  - 状況：ローのセットに対する SUM は、特定のセットについて切断。サブセットの任意のセットに対する SUM から計算できる
  - 残りのローをグループに分割し、そのグループに対して SUM を事前計算することができる

マテリアライズド・ビューの使用：データベース設計

- どのグループか
- Q19 は p_brand と l_quantity でパラメータ化される
  - これらのカラムの値は*推論的*にはわかっていない
  - 実体化されたデータはこれら 2 つのカラムをフィルタリングできる必要がある
  - GROUP BY p_brand, l_quantity
- (p_brand, l_quantity) の 3 つのペアごとに：
  - 論理和述部が p_size と p_container で異なる値を選択
  - 実体化されたデータに対してこのフィルタリングを可能にするには、これら 2 つのカラムを GROUP BY に追加する必要がある
  - GROUP BY p_brand, l_quantity, p_size, p_container

マテリアライズド・ビューの使用：データベース設計

```
CREATE MATERIALIZED VIEW v19 AS
SELECT      sum(l_extendedprice* (1 - l_discount)) AS revenue,
        p_brand,
        l_quantity,
        p_size,
        p_container
FROM        LineItem, Part
WHERE       l_partkey = p_partkey
        AND p_container in ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG',
                    'MED BAG', 'MED BOX', 'MED PKG', 'MED PACK',
                    'LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        AND l_shipmode in ('AIR', 'AIR REG')
        AND l_shipinstruct = 'DELIVER IN PERSON'
GROUP BY
        p_brand, l_quantity, p_size, p_container
```

マテリアライズド・ビューの使用：データベース設計

- 元のクエリ・プラン：
  - GrByS[HF[Part<seq>] *JHx LineItem<seq>]
  - Part と Lineitem のスキャンが必要：620 万ロー
  - 実行時間：17.3 秒
- v19 を使用できるクエリ・プラン：
  - GrByS[v19<seq>]
  - ビューの 1 回のスキャン：60,691 ロー
  - 実行時間：0.07 秒
  - 実行時間には、ビュー・マッチングとコストベース分析による追加のオーバーヘッドが含まれる

マテリアライズド・ビューの使用：データベース設計

- さらに向上を図ることができるか
- ビューをベース・テーブルとして扱って同じ分析を適用し、基本となるデータへのアクセスを高速化する
- 効果的なインデックスを作成する
- Q19 は p_brand の 3 つの値について v19 からローを選択する
- CREATE INDEX v19_brand ON v19(p_brand)
  - Plan: GrByS[IN JNL v19<v19_brand>]

マテリアライズド・ビューの使用：データベース設計

- Q19 はさらに l_quantity でローのフィルタリングを行う
- CREATE INDEX v19_brand_quantity ON
  v19(p_brand, l_quantity)
  - Plan: GrByS[IN JNL v19<v19_brand_quantity>]
- tablesv19_brand_quantity は 2 種類のベース・テーブルからデータをインデックス化することに注意
- インデックス・コンサルタントの推奨事項:
  - v19< p_brand, l_quantity >
- インデックスはデータ量が多くなるほど重要になる

マテリアライズド・ビューの使用：データベース設計

- マテリアライズド・ビューは、同じクエリの複数のブロックとマッチングできる
- Q15：ユーザが指定した日付から始まる 3 か月間の上位サプライヤはどこか
- ビューには全サプライヤのリストと、この 3 か月における売上高が載っていると想定する

```
CREATE VIEW revenue (supplier_no, total_revenue) AS
    SELECT   l_suppkey, sum(l_extendedprice * (1 - l_discount))
    FROM     LineItem
    WHERE  l_shipdate >= date( '1996-01-01‘ ) AND          -- :1
                l_shipdate < DATEADD( month, 3, '1996-01-01' )  -- :1
    GROUP BY    l_suppkey
```

マテリアライズド・ビューの使用: データベース設計

- Q15:

```
SELECT s_suppkey, s_name, s_address, s_phone,
       total_revenue
FROM  Supplier, revenue
WHERE s_suppkey = supplier_no AND
    total_revenue = ( select MAX( total_revenue )
                      FROM revenue )
```

- 売上高のクエリは 2 回実行する必要がある:
  - 売上高の最高値を計算するサブクエリ
  - 上位サプライヤを検索する Supplier とのジョイン
- 売上高のクエリは事前計算できない
  - 開始の日付が不明

マテリアライズド・ビューの使用：データベース設計

- 各サプライヤの日次の売上高を計算するマテリアライズド・ビュー:
  - 日付がわかった時点で 3 か月分のローをスキャン

```
CREATE MATERIALIZED VIEW viewtpch15 AS
  SELECT      sum(l_extendedprice * (1 - l_discount)),
              l_suppkey, l_shipdate
  FROM        Lineitem
  GROUP BY    l_suppkey,  l_shipdate
```

- 日付がわかった時点で 3 か月分のローをスキャン:

Work[GrByH[viewtpch15<viewtpch15_shipdate>]
JNL SUPPLIER<Supplier>]
: GrByS[GrByH[viewtpch15<viewtpch15_shipdate>]]

マテリアライズド・ビューの使用：データベース設計

- オプティマイザは同じクエリに対して複数の種類のマテリアライズド・ビューを使用できる
- 単純なクエリと、キャッシュされたプランによるクエリが最適化プロセスをバイパス
  - ビュー・マッチングは実行されない
  - OPTION(FORCE OPTIMIZATION) を使用してこれらのクエリを最適化し、マテリアライズド・ビューの効果を得る
    - プラン・キャッシュの効果を比較する
- 更新可能なテーブルに対してビュー・マッチングは実行されない:
  - グループ化されたクエリでない場合には READ ONLY を宣言する
- マテリアライズド・ビューの作成に効果的とは限らない

マテリアライズド・ビューの使用: SQL Anywhere での OUTER JOIN ビューのサポート

- マテリアライズド・ビュー:

```
CREATE MATERIALIZED VIEW VV1 ( quantity, size, partkey ) AS
SELECT l_quantity, p_size, p_partkey
FROM Lineitem KEY LEFT OUTER JOIN Part
```

- クエリ:

```
SELECT l_quantity, p_size, p_partkey
FROM Lineitem KEY JOIN Part
WHERE p_size > 10
```

- ビューを使用してリライトされたクエリ:

```
SELECT *
FROM VV1
WHERE VV1.quantity > 10 AND VV1.partkey IS NOT NULL;
```

- 外部ジョインのある場合とない場合のクエリ、WINDOW 関数を使用するクエリ、その他に対する OUTER JOIN ビュー・マッチングについては、SQL Anywhere ドキュメントのサンプルを参照のこと

マテリアライズド・ビューの使用：古さの制御

- 延期マテリアライズド・ビューには古いデータが含まれる場合がある
  - ベース・テーブルに対する更新が即時に伝播されない
- 古いマテリアライズド・ビューを使用して返された結果セットには、データベースの最新状態が反映されない
- 古い結果が許容されるかどうかはアプリケーションで決まる
- サーバは新しさの要件を受け入れる
  - ビュー・マッチングの際にサーバによるマテリアライズド・ビューの使用に影響する

マテリアライズド・ビューの使用：古さの制御

- 接続レベル:
  - SET OPTION … Materialized_view_optimization = '…'
- 新しい文 (SELECT, MERGE, UPDATE, …) OPTION 句:
  - SELECT … OPTION(MATERIALIZED VIEW OPTIMIZATION '…')
- 使用可能な値:
  - **Disabled**: マテリアライズド・ビューを評価しない
  - **Fresh**: 新しいマテリアライズド・ビューのみを評価する
  - **Stale**: 古さにかかわらず、すべてのマテリアライズド・ビューを評価する
  - **N units**: 直前の N 時間単位内に更新されている場合にのみ古いマテリアライズド・ビューを評価する
    - 例: 5 分 - 古さが 5 分以内の場合にのみ古いビューを使用する
- デフォルト値: Stale

アジェンダ

- SQL Anywhere での MERGE 文
  - 必要な理由
  - ANSI SQL の MERGE
  - SQL Anywhere の拡張
- SQL Anywhere のマテリアライズド・ビュー
  - 定義と必要な理由
  - 長所と短所
  - 設計と作成の問題
  - 保守の方法と課題
  - クエリ・パフォーマンス向上のための配備
  - ➢ サーバのサポートと制限事項

SQL Anywhere のマテリアライズド・ビュー: DDL

- **CREATE MATERIALIZED VIEW**
  [*owner*.]*materialized-view-name* [**(** *column-name*, ... **)**]
  [**IN** *dbspace-name*]
  AS *select-statement*
  [**CHECK { IMMEDIATE | MANUAL } REFRESH**]

- **ALTER MATERIALIZED VIEW** [*owner*.]*materialized-view-name* {
  SET HIDDEN
  | { **ENABLE** | **DISABLE** }
  | { **ENABLE** | **DISABLE** } **USE IN OPTIMIZATION**
  | { **ADD PCTFREE** *percent-free-space* | **DROP PCTFREE** }
  | [**NOT**] **ENCRYPTED**
  | { IMMEDIATE | MANUAL } REFRESH
  }

SQL Anywhere のマテリアライズド・ビュー: DDL/DML

- **REFRESH MATERIALIZED VIEW** *view-list*
  [**WITH** { **ISOLATION LEVEL** *isolation-level*
  | { **EXCLUSIVE | SHARE** } **MODE** }]
  [**FORCE BUILD**]

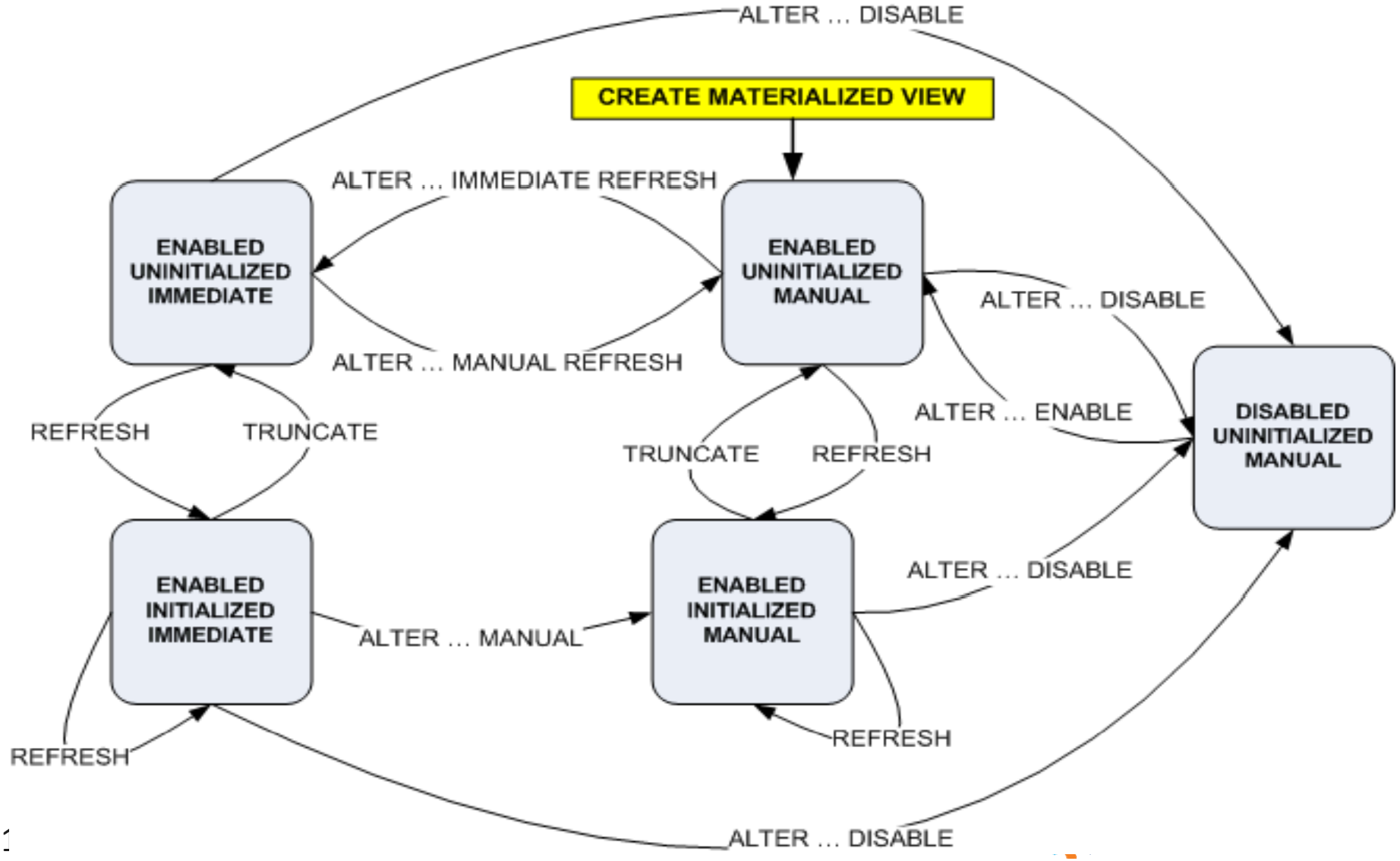
- **TRUNCATE MATERIALIZED VIEW** *[owner.]view-name*
  - *Supported starting with SA11SA11 からサポートを開始*

- マテリアライズド・ビューをターゲットにできる句:
  - UNLOAD
  - VALIDATE
  - COMMENT ON
  - DROP

SQL Anywhere のマテリアライズド・ビュー: ステータス

- **Enabled:** 正常にコンパイルされ、使用可能な状態。初期化解除が可能
- **Disabled:** ユーザが明示的に無効化。ビューを無効化すると、データ(未初期化) とインデックスが破棄され、リフレッシュ・タイプが手動に戻る
- **Initialized:** 有効化され、データがある
- **Uninitialized:** 有効か無効のいずれかで、データがない
  - 新規のビューは、手動でも即時でも明示的にリフレッシュするまで使用も保守もできない
- 初期化後に、TRUNCATE 文および DISABLE 文でビューのデータを
- 破棄すると初期化が解除される

SQL Anywhere のマテリアライズド・ビュー: ステータス



SQL Anywhere のマテリアライズド・ビュー：オプション

- マテリアライズド・ビューを作成する際、一部の接続オプションには必須の
- 設定がある

オプション	必須の値
Ansi_integer_overflow (SA10 only)	ON
Ansinull	ON
Conversion_error	ON
Divide_by_zero_error	ON
Float_as_double (SA10 only)	OFF
Sort_collation	Internal
String_rtruncation	ON

SQL Anywhere のマテリアライズド・ビュー: オプション

- ビューの作成時に保存される一部のオプション設定:
  - Date_format, Date_order, Default_timestamp_increment, First_day_of_week, Nearest_century, Precision, Scale, Time_format, Timestamp_format, Timestamp_zone_format, Uuid_has_hyphens
  - SA12 only: st_geometry_asbinary_format, st_geometry_astext_format, st_geometry_asxmly_format, st_geometry_on_invalid
  - クエリの結果に影響する必須のオプション
- ビューの保守では、ビューで保存されたすべての必須オプション値を暗黙的に想定する
  - 現在の接続設定は無視される
- 最適化の際のビュー・マッチングでマテリアライズド・ビューが評価されるのは、保存されたオプション値が現在の接続のオプション値に一致する場合のみ。

SQL Anywhere のマテリアライズド・ビュー: 診断

- **sa_materialized_view_info(** [*view_name* [**,** *owner_name*]] **)**
- 現在の接続のコンテキストで、既存のマテリアライズド・ビューに関する情報を示す
- ステータス:
  - **D:** ユーザによりビューが無効化されている
  - **E:** ビューが有効
- DataStatus:
  - **N:** 未初期化。作成後にリフレッシュされていないか、トランケートまたは無効化された
  - **E:** 未初期化。前回リフレッシュを試行したときエラーが発生した
  - **F:** 最新
  - **S:** 古い。基本となるデータが前回のリフレッシュ後に変更された

SQL Anywhere のマテリアライズド・ビュー: 診断

- ViewLastRefreshed: 前回のリフレッシュ時刻
- DataLastModified: 古いビューで、基本となるデータの変更がわかっていた最後の時刻
- AvailableForOptimization: オプティマイザがマッチングの際にビューを考慮するかどうか。考慮しない場合は、その理由
  - **Y:** オプティマイザで使用可能
  - **D**: オプティマイザによる使用が無効
  - **N**: 未初期化
  - **I:** マッチングの制限に違反
  - **O:** オプション値が現在の接続に一致しない
- RefreshType: ビューの保守ポリシー
  - **I:** サーバによる即時、自動、インクリメンタルの保守
  - **M:** ユーザによる手動の延期保守

SQL Anywhere のマテリアライズド・ビュー: 診断

- **sa_materialized_view_can_be_immediate(** *view_name*, *owner_name* **)**
- マテリアライズド・ビューが即時ビューとして宣言される要件をすべて満たしているかどうかを検証する
- ビューがすべての要件を満たしている場合にはローなし
- ビューを即時と宣言できない潜在的な理由があれば、その 1 つごとに 1 つのローが結果セットで返される
  - SQLStateVal: エラーの SQLSTATE
  - ErrorMessage: エラー・メッセージ

SQL Anywhere のマテリアライズド・ビュー: 診断

```
CREATE MATERIALIZED VIEW view10 AS
SELECT c_name, sum( l_extendedprice ) as revenue
FROM Customer, Orders, Lineitem
WHERE c_custkey = o_custkey AND o_orderkey = l_orderkey
GROUP BY c_custkey, c_name;

REFRESH MATERIALIZED VIEW view10;
```

SQL Anywhere のマテリアライズド・ビュー: 診断

SELECT SQLStateVal, ErrorMessage FROM
sa_materialized_view_can_be_immediate('view10', 'DBA')

SQLStateVal	ErrorMessage
42WC3	The materialized view view10 cannot be changed to immediate because it has already been initialized.マテリアライズド・ビュー view10 は、すでに初期化されているため即時に変更できません)
42WCA	The materialized view view10 cannot be changed to immediate because it does not have a unique index on non-nullable columns. (マテリアライズド・ビュー view10は、NULL 不可のカラムに一意のインデックスがないため即時に変更できません)
42WC6	The materialized view cannot be changed to immediate because COUNT(*) is required to be part of the SELECT list. (COUNT(*) を SELECT リストの一部にする必要があるため、マテリアライズド・ビューは即時に変更できません)
42WC7	The materialized view cannot be changed to immediate because it does not have a unique index on non-aggregate non-nullable columns. (集計以外の NULL 不可のカラムに一意のインデックスがないため、マテリアライズド・ビューは即時に変更できません)

SQL Anywhere のマテリアライズド・ビュー: インデックス・コンサルタントと アプリケーション・プロファイリング

- インデックス・コンサルタントは、次のいずれの場合にもマテリアライズド・ビューのインデックスを推奨する
  - ワークロードで直接使用される - 他のベース・テーブルと類似
  - ワークロードで直接使用されないが、最適化プロセスのビュー・マッチングで評価される
- アプリケーション・プロファイリングでは、マテリアライズド・ビューをベース・テーブルのように扱う

SQL Anywhere のマテリアライズド・ビュー: インデックス・コンサルタント

- マテリアライズド・ビューを作成して初期化する:

```
CREATE MATERIALIZED VIEW mv_lps AS
SELECT p_name, s_name, count( * )
FROM Lineitem, Part, Supplier
WHERE l_partkey = p_partkey
    AND l_suppkey = s_suppkey
GROUP BY p_name, s_name
```

SQL Anywhereのマテリアライズド・ビュー: インデックス・コンサルタント

- クエリを実行する:

```
SELECT p_name, s_name, count( * ) as num_orders
FROM Lineitem, Supplier, Part
WHERE l_partkey = p_partkey
   AND l_suppkey = s_suppkey
   AND s_name = 'Supplier#000008315'
   AND p_name like 'white%'
GROUP BY p_name, s_name;
```

- 実行時間 : 4.816 秒
- ビューはコスト計算されるが使用されない

SQL Anywhereのマテリアライズド・ビュー: インデックス・コンサルタント

- クエリに対してインデックス・コンサルタントを実行する
- 推奨のインデックス <s_name,p_name>
- 推奨されたインデックスを作成する
- クエリはマテリアライズド・ビューを使用する
- 実行時間: 0.058 秒

SQL Anywhereのマテリアライズド・ビュー: グラフィカルプラン

- *[Advanced Details] ペインには、オプティマイザが評価したマテリアライズド・ビューに関する情報が表示される*
- まったく評価されなかったビューは表示されない
  - 診断に**sa_materialized_view_info() を使用する**
- 詳細には、ビューごとにオプティマイザが実行した内容が含まれる:
  - 最高のクエリ実行プランで使用された
  - コストを分析したが、他のプランの方が負荷が小さかった
  - コストベース分析でビューが評価されなかった理由
    - 「述部の不一致」、「選択リストの不一致」、「マッチングしきい値」、「アクセス不可」など

SQL Anywhereのマテリアライズド･ビュー: グラフィカルプラン

- 推奨されたクエリの *[Advanced Details] のサンプル*:

評価されたマテリアライズド･ビュー	結果	照合名
viewtpch14	Select list mismatch	LINEITEM
…		
lineitem_part_mv	Costed	LINEITEM
lineitem_orders_part_mv	Costed	LINEITEM
lineitem_orders_mv	View used	LINEITEM
viewtpch1	Select list mismatch	LINEITEM
…		
viewtpch3	Select list mismatch	LINEITEM ORDERS
viewtpch7	Predicate mismatch	LINEITEM ORDERS
…		
viewtpch10	Select list mismatch	LINEITEM ORDERS CUSTOMER
viewtpch22	Select list mismatch	CUSTOMER

SQL Anywhereのマテリアライズド・ビュー：パフォーマンスに対する影響

- クエリで最適なパフォーマンスを達成するために作成されたマテリアライズド・ビュー
  - 前述の例で使用したのと同様に分析
- マテリアライズド・ビューが常に効果的とは限らない
- 多くの TPC-H クエリは、マテリアライズド・ビューを推奨しないように設計されている：アドホック・ベンチマーク
  - Q15 は、SF1 でビューを使用するが SF10 では使用しない
- 適切な状況ではパフォーマンス上のメリットは大きい
- マテリアライズド・ビューのは並行プランにも効果的

結論として

- SQL Anywhereの MERGE は強力な ETL 機能を果たす
- マテリアライズド・ビューの効果
  - 領域と時間のかね合い
  - 計算上のコストを軽減する
- ただし、通常のビューの代替になるわけではない
- マテリアライズド・ビューを設計するアクションのプラン
- システム負荷から、例えば以下のようなクエリのサブセットを選択することから開始する
  - 応答時間の要件が不可欠である高負荷のクエリ
  - 頻繁に実行される、重要なクエリ
  - 結果の最新状態が必要以上に重要ではないクエリ
  - 静的なデータ (大部分) に対するクエリ:
    - スナップショット・クエリ／履歴クエリ:「前営業日の決算時の集計」など

結論として

- 選択したクエリごとにマテリアライズド・ビューの設計を検討する:
  - オプティマイザがマッチングに利用できる構造体を使用する
  - このプレゼンテーションの例を参考にする
  - マテリアライズド・ビューはクエリよりも汎用に保つ
    - クエリで必要とされるより高い粒度
  - アプリケーションがクエリに求める最新さの要件を検討し、適切なリフレッシュ戦略を配慮して設計する:
    - マテリアライズド・ビューを更新するタイミングと方法
    - 即時、インクリメンタルの保守は適切か
  - ストレージと保守の要件を意識する
  - クエリごとに、有効と考えられるマテリアライズド・ビューを複数検討する
  - マテリアライズド・ビューのパーティショニングを検討する

結論として

- 設計した一連のマテリアライズド・ビューを一緒に検討し、冗長なビューは排除する
  - 複数のビューで共通するパーツの分割を考慮する
    - オプティマイザは 1 つのクエリに複数のマテリアライズド・ビューを使用できる
- 最終的なマテリアライズド・ビューのセットに対してインデックスを設計する
  - ベース・テーブルで採用するのと類似の分析
  - インデックス・コンサルタントを活用する
- 選択したマテリアライズド・ビューを実装し、選択したクエリについてパフォーマンスの改善点を測定する
  - 必要に応じてビューに調整を加える

THANK YOU

QUESTIONS?