ORACLE

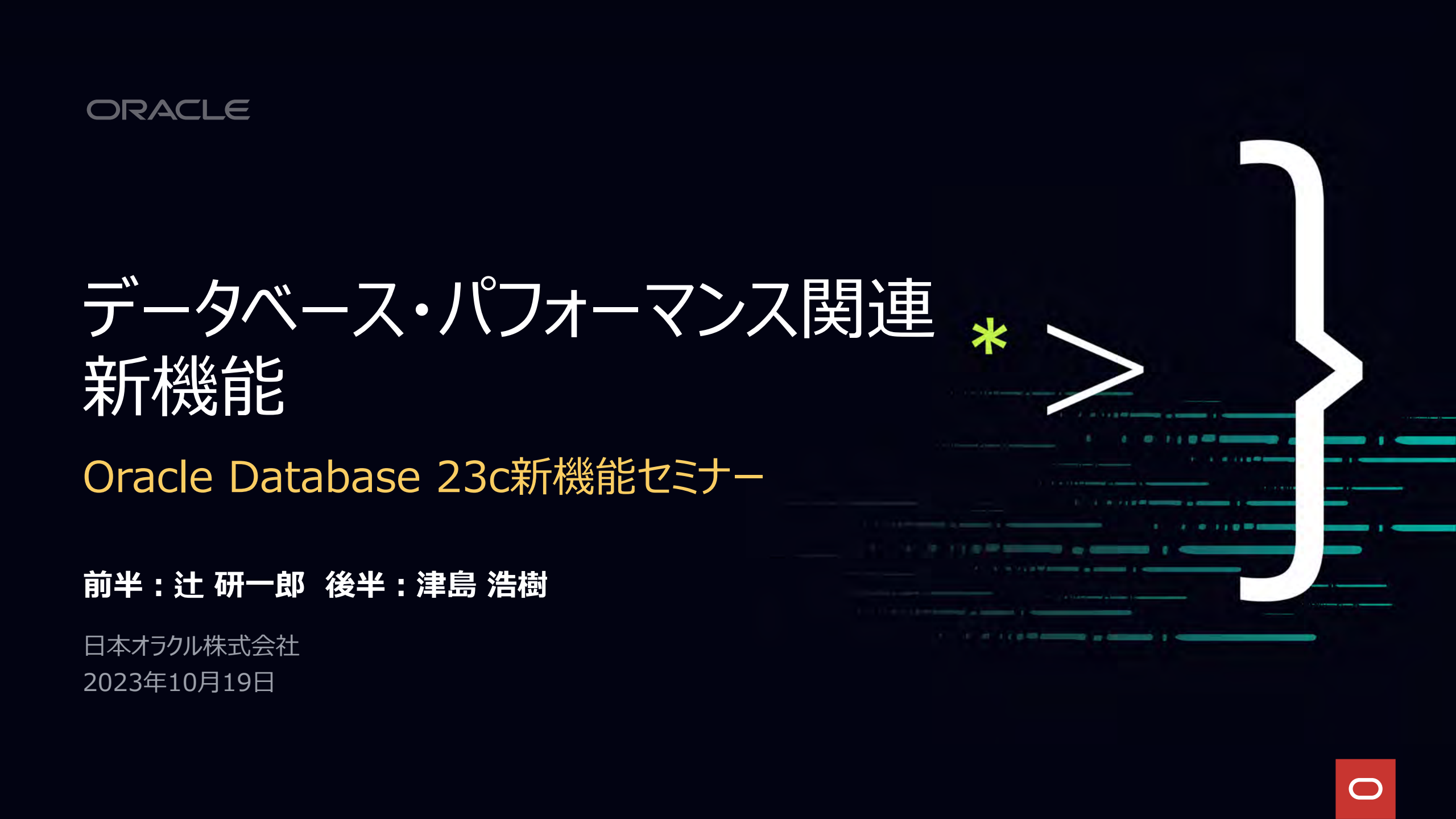
データベース・パフォーマンス関連 新機能

Oracle Database 23c新機能セミナー

前半：辻 研一郎　後半：津島 浩樹

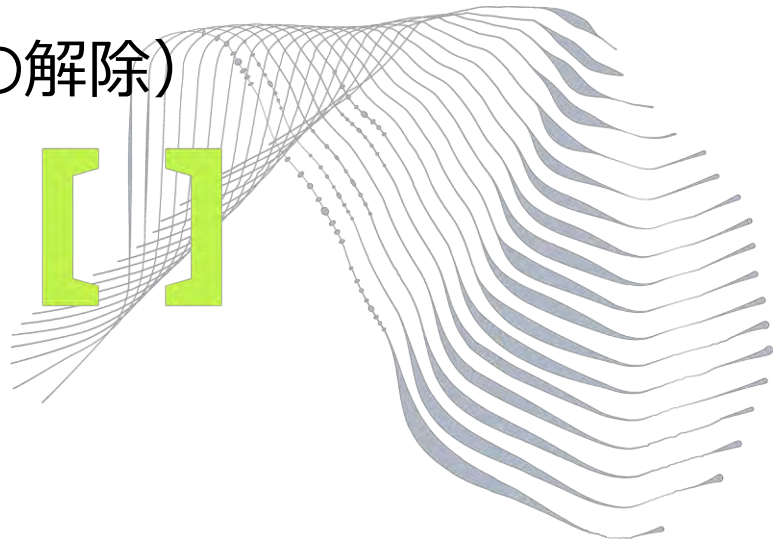
日本オラクル株式会社

2023年10月19日

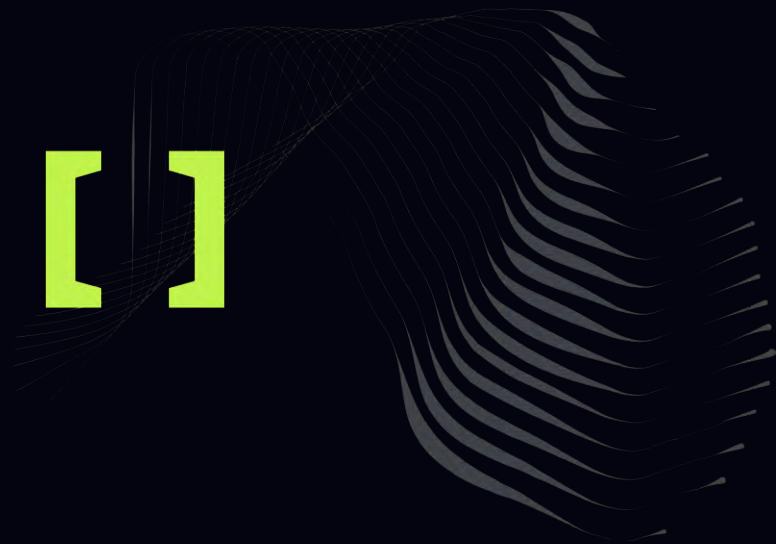


Agenda

1. ロックフリー列値の予約 (ロックフリー予約)
2. 自動トランザクションロールバック
3. 幅の広い表（表/ビューの最大列数の4096列への拡張）
4. 無制限のパラレルDML/ダイレクトロード（トランザクション制約の解除）
5. Oracle Database 23c におけるマテリアライズド・ビュー強化
6. Partitioning 23c 新機能



ロックフリー列値の予約(ロックフリー予約)



ロックフリー列値の予約

Lock-free Column Value Reservations

概要

- 人気商品の在庫管理表の在庫数など、数値列の加算、減算の更新に対するトランザクションでロック待ち競合を回避可能に
  - 加算、減算の結果を更新する代わりに差分をジャーナル表に保存
  - トランザクションのコミット時に行ロックを取得し差分を適用
- "ホット"リソースとなる数値集計データに有効
  - 例：在庫管理の在庫数、銀行口座残高、利用可能な座席数など

```
--減算のUPDATE文の例
○：UPDATE inventory SET qty_on_hand = qty_on_hand - 1
     WHERE item_id = ***
--値を更新するUPDATEには適用できない
×：UPDATE inventory SET qty_on_hand = 100
     WHERE item_id = ***
```

メリット

- ユーザー・エクスペリエンスと同時実行性の向上
  - 対象列に頻繁に複数の更新が同時発生するようなケース、待ちの回避でリソース効率も向上
- 幅広い範囲で利用可
  - ロックフリー予約の対象となる数値集計データはアプリケーションで利用が多い
- Sagaトランザクションの自動補正（暗黙的な補正トランザクションの自動発行）
  - Sagaトランザクションが中断された場合、ロックフリー予約列の自動補正、ロールバックを実行

→ こちらはDay2 Micro Serviceの回で扱います。(9/28 TechNight#71 - Oracle Database 23c 新機能#1 Microservices関連新機能でも扱っています)

従来の課題：行ロック待機によるトランザクションのブロック

時間

Txn1（SID 32）

```
SQL> select item_id, qty_on_hand, shelf_capacity from
inventory;

 ITEM_ID QTY_ON_HAND SHELF_CAPACITY
---------- ---------- --------------
       123          70            120
       456          50            100
       789          50             75

SQL> update inventory
     set qty_on_hand = qty_on_hand - 10
     where item_id = 123;

1 row updated.




select sid,event,blocking_session from v$session where
username='TEST'

SID    EVENT                                BLOCKING_SESSION
------ ---------------------------------- ----------------
    32 SQL*Net message to client
   227 enq: TX - row lock contention                     32
```

Txn2（SID 227）

```
SQL> update inventory
     set qty_on_hand = qty_on_hand + 20
     where item_id = 123;
```

Txn1のトランザクションが完了(コミット or ロールバック)するまで待機

ロックフリー列値の予約の使用手順

更新対象列を予約可能列とするため、RESERVABLE句を指定する

- 予約可能列の宣言例

```
create table inventory
( item_id             NUMBER          CONSTRAINT inv_pk PRIMARY KEY,
  item_display_name   VARCHAR2(100)   NOT NULL,
  qty_on_hand         NUMBER          RESERVABLE CONSTRAINT qty_ck CHECK (qty_on_hand >= 0) NOT NULL,
  shelf_capacity      NUMBER          NOT NULL,
  CONSTRAINT shelf_ck CHECK (qty_on_hand <= shelf_capacity)
);
```

- alter table add columnやalter table modify columnによる追加、変更も可。以下はmodifyの例

```
--MODIFYによる予約可能列の解除
ALTER TABLE inventory MODIFY (qty_on_hand NOT RESERVABLE);
--MODIFYによる予約可能列の再指定
ALTER TABLE inventory MODIFY (qty_on_hand RESERVABLE);
```

CHECK制約について

- 予約可能列にCHECK制約は必須ではない
- 表レベルのチェック制約に予約可能列、非予約可能列が含まれる場合は、非予約可能列の変更はコミット時に評価され、検証に失敗した場合はトランザクションは中断、ロールバックされる

予約ジャーナル表

予約ジャーナル表：SYS_RESERVJRNL_<ベース表のobject番号>

予約可能列を宣言した時に自動的に作成される表。予約可能列に対する更新情報が含まれる

- 予約ジャーナル表の例

```
SQL> desc SYS_RESERVJRNL_78065
 Name                          Null?    Type
 ----------------------------- -------- -------- ------------------------------------------------------
 ORA_SAGA_ID$                           RAW(16)       --SagaトランザクションのSaga ID
 ORA_TXN_ID$                            RAW(8)        --トランザクションID
 ORA_STATUS$                            VARCHAR2(11)  --Txn IDのステータス
 ORA_STMT_TYPE$                         VARCHAR2(6)   --DML文タイプ
 ITEM_ID                       NOT NULL NUMBER        --主キー列名
 QTY_ON_HAND_OP                         VARCHAR2(1)   --"予約可能列名"_OP。加算か減算の操作(+、-)
 QTY_ON_HAND_RESERVED                   NUMBER        --"予約可能列名"_RESERVED 予約された量
```

- 自トランザクションの更新情報のみ確認可であり、他トランザクションの更新情報は見えない
- 予約ジャーナル表へのDML、DDL操作は不可
- ユーザー表から予約可能列を削除すると、対応するロックフリー予約追跡列が予約ジャーナル表から削除され、最後の予約可能列がユーザー表から削除されると、予約ジャーナル表が削除される

予約ジャーナル表への加算、減算の結果の入り方と予約可能列のアップデート

```
SQL> select item_id, qty_on_hand, shelf_capacity
  2  from inventory where item_id=123;

   ITEM_ID QTY_ON_HAND SHELF_CAPACITY
---------- ----------- --------------
       123          80            120

SQL> update inventory
  2       set qty_on_hand = qty_on_hand - 10
  3       where item_id = 123;
1 row updated.

SQL> update inventory
  2       set qty_on_hand = qty_on_hand + 20
  3       where item_id = 123;
1 row updated.

SQL> select * from SYS_RESERVJRNL_100639;
ORA_SAGA_ID$                     ORA_TXN_ID$      ORA_STATUS$  ORA_STMT_TYPE$      ITEM_ID QTY_ON_ QTY_ON_HAND_RESERVED
-------------------------------- ---------------- ------------ ---------------- ---------- ------- --------------------
                                 03001200CE340000 ACTIVE       UPDATE                  123 -                        10
                                 03001200CE340000 ACTIVE       UPDATE                  123 +                        20

--予約可能列の値は自トランザクションであってもコミットするまで変わらない、コミットするとQTY_ON_HAND は 90になる
SQL> select item_id, qty_on_hand, shelf_capacity
  2  from inventory where item_id=123;

   ITEM_ID QTY_ON_HAND SHELF_CAPACITY
---------- ----------- --------------
       123          80            120
```

加算、減算の結果を更新する代わりに差分をジャーナル表に保存
自トランザクションの更新情報のみ確認可
コミット時に予約ジャーナル表の加算、減算を合算し、値を更新

予約可能列のチェック制約、表レベルのチェック制約の動作（1/2）

```
SQL> select item_id, qty_on_hand, shelf_capacity
  2  from inventory where item_id=123;

   ITEM_ID QTY_ON_HAND SHELF_CAPACITY
---------- ----------- --------------
       123          80            120

--予約可能列のチェック制約違反、即評価される
--他セッションのトランザクション確定前の情報を含めて評価され、チェック制約に違反したらエラーが返る
SQL> update inventory
  2       set qty_on_hand = qty_on_hand - 110
  3       where item_id = 123;

ORA-02290: check constraint (TEST.QTY_CK) violated
Help: https://docs.oracle.com/error-help/db/ora-02290/

--表レベルのチェック制約違反、予約可能列の値は即評価され、チェック制約に違反したらエラーが返る
SQL> update inventory
  2       set qty_on_hand = qty_on_hand + 120
  3       where item_id = 123;

ORA-02290: check constraint (TEST.SHELF_CK) violated
Help: https://docs.oracle.com/error-help/db/ora-02290/
```

予約可能列のチェック制約
CONSTRAINT qty_ck CHECK (qty_on_hand >= 0)
表レベルのチェック制約
CONSTRAINT shelf_ck CHECK (qty_on_hand <= shelf_capacity)
表レベルはコミット時に評価

予約可能列のチェック制約、表レベルのチェック制約の動作（2/2）

```
SQL> select item_id, qty_on_hand, shelf_capacity
  2  from inventory where item_id=123;

   ITEM_ID QTY_ON_HAND SHELF_CAPACITY
---------- ----------- --------------
       123          80            120
```

予約可能列のチェック制約
CONSTRAINT qty_ck CHECK (qty_on_hand >= 0)
表レベルのチェック制約
CONSTRAINT shelf_ck CHECK (qty_on_hand <= shelf_capacity)
表レベルはコミット時に評価

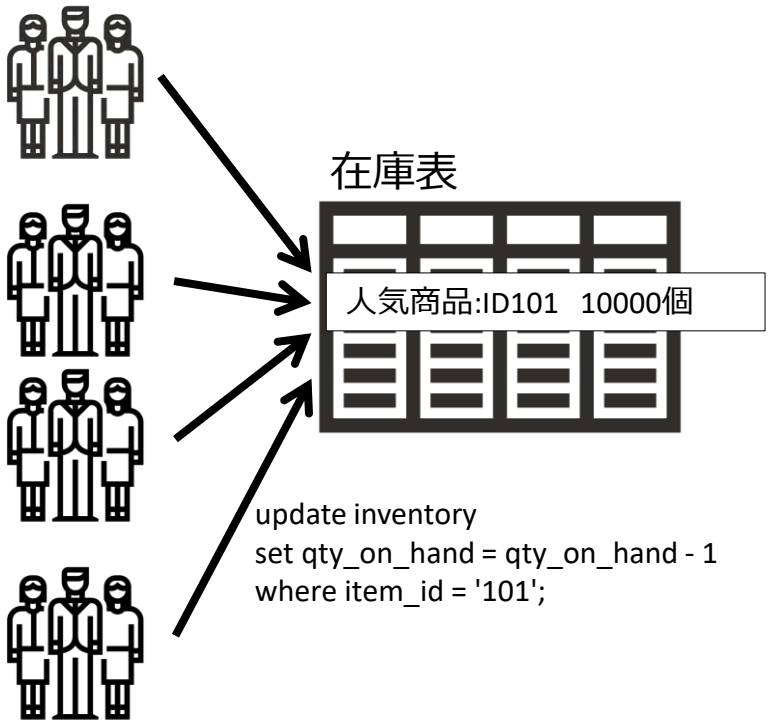
```
-- QTY_ON_HANDは現在 90 (=80-10+20)
SQL> select ORA_STMT_TYPE$,ITEM_ID,QTY_ON_HAND_OP,QTY_ON_HAND_RESERVED from SYS_RESERVJRNL_100639;

 ORA_STMT_TYPE$      ITEM_ID QTY_ON_ QTY_ON_HAND_RESERVED
 --------------- ---------- ------- --------------------
 UPDATE                 123 -                        10
 UPDATE                 123 +                        20

-- 非予約可能列SHELF_CAPACITYを表レベルのチェック制約違反となるように更新。この場合はこの時点では評価されない
SQL> update inventory
  2       set shelf_capacity  = 85
  3       where item_id = 123;
1 row updated.

--コミット時に、表レベルのチェック制約違反が評価され、ロールバックされる
SQL> commit;
commit
ORA-02290: check constraint (TEST.SHELF_CK) violated
Help: https://docs.oracle.com/error-help/db/ora-02290/
```

ロックフリー列値の予約性能検証

次の3パターンを検証

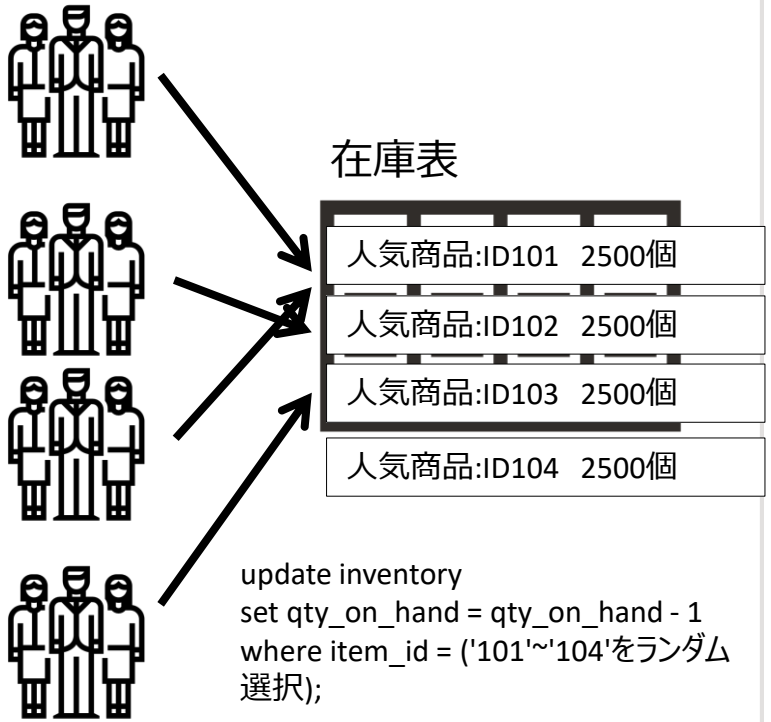
参考：23cでのロックフリー予約: アプリのスケーリング
https://blogs.oracle.com/oracle4engineer/post/ja-lockfree-reservation-in-23c-scale-your-apps
こちらのスクリプトを改変して検証をしています

１．非ロックフリー在庫分割なし



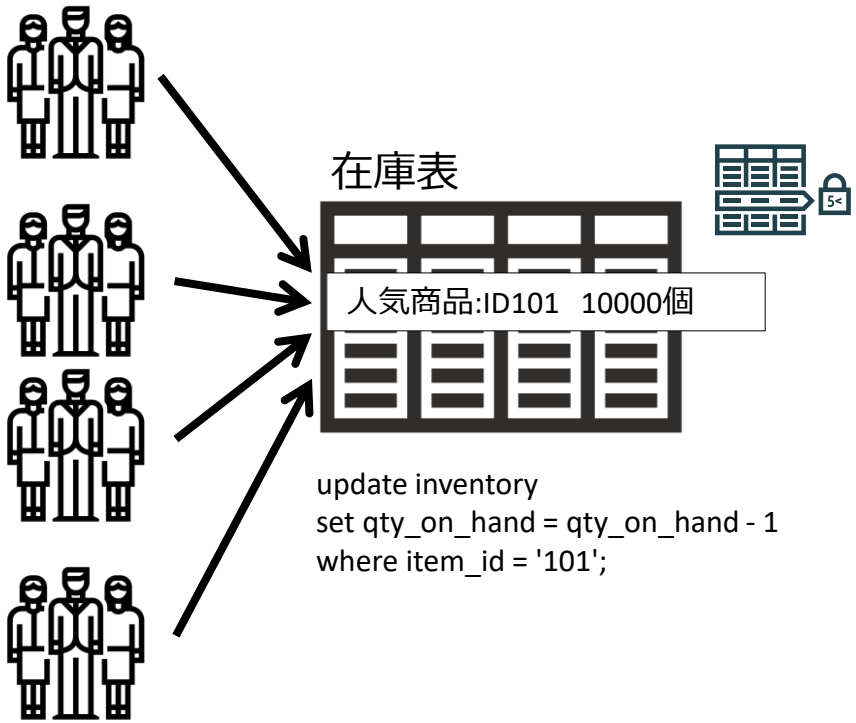
1つの行の更新に殺到するため、人の更新待ちが発生する
(enq: TX - row lock contention)

２．非ロックフリー在庫分割あり



アプリの改修を実施。同じ商品の管理レコードを増やして在庫を分割し、行アクセスを散らすことで同一行の更新の集中を回避

３．ロックフリー列値の予約を使用

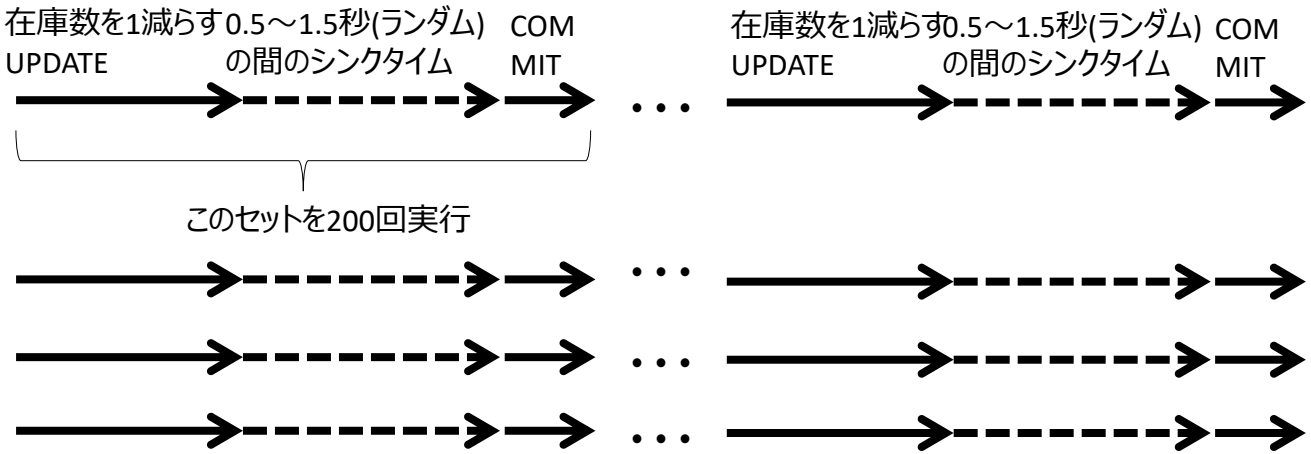


１と同じ状態だが、在庫数の列にロックフリー列値の予約を使用

ロックフリー列値の予約性能検証
検証内容説明

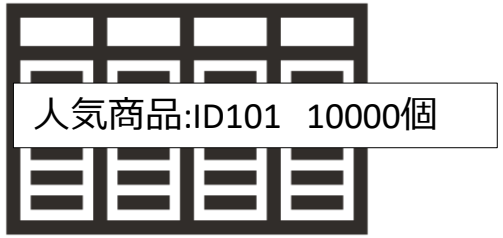
以下の通り検証を実施し、検証前後のAWRスナップショットから得られたAWRレポート中のenq: TX - row lock contentionを比較する

4セッション使用し、以下のように実行する



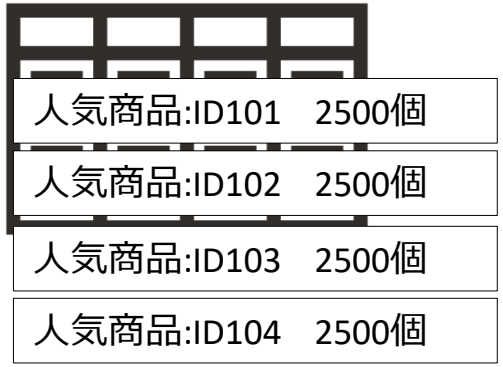
検証パターン１と３では１レコードにUPDATE集中
（３では予約可能列を指定）

在庫表



検証パターン2では以下のレコードを使用
UPDATE対象は101～104の中からランダム

在庫表



ロックフリー列値の予約性能検証

検証結果

ロックフリー列値の予約では、ほぼenq: TX - row lock contentionが見られなくなり、アプリケーションの改修なく人気商品への在庫数管理に対応可能なことが確認できました

enq: TX - row lock contention(他ユーザーの在庫数更新待ち)の値

	Waits	Total Wait Time (sec)	Avg Wait	% DB time
非ロックフリー在庫分割なし	3,288	3381.8	1028.52ms	99.9
非ロックフリー在庫分割あり	884	681	770.38ms	99.8
ロックフリー列値の予約	50	0	478.02us	1.0

※Top 10 Foreground Events by Total Wait Timeから抜粋

ロックフリー列値の予約の制約、ガイドライン

制約

- 予約可能列はOracle数値データ型（NUMBER,INTEGER,FLOAT）にのみ指定が可能
- 予約可能列をPrimary keyまたはIdentity列（または仮想列）にすることは不可
- ひとつの表あたり最大10個の予約可能列を含めることが可能
- 予約可能列が含まれるユーザー表には、Primary Keyが必要
- 予約可能列ではStorage句は非サポート
- 予約可能列では、索引、トリガーは非サポート
- 複合予約可能列は使用できない
- ブロック・チェーン表およびシャード表では、予約可能列は使用できない
- 予約可能列はCHECK制約式にのみ含めることが可。他のタイプの制約に含めることは不可
- 外部表、クラスタ表、IOT表および一時表では予約可能列の指定不可
- 予約可能列でのパーティション化は不可
- 予約可能列が含まれるユーザー表には、2フェーズのオンラインDDL最適化は提供されない
- 予約可能列を削除したり、列をUNUSEDとマークするには、保留中の予約があるトランザクションをファイナライズする必要がある

ロックフリー列値の予約の制約、ガイドライン

予約可能列に対するUpdate文のガイドライン

- 条件に主キーを指定した予約可能列の加算、減算のみ
  - `UPDATE <table_name> SET <reservable_column_name> = <reservable_column_name> +/- (<expression>) WHERE <primary_key_column> = <expression>`
- 複合主キーの場合はすべての主キー列を条件に指定する
- 1つのUpdate文で複数の予約可能列の更新可
- 1つのUpdate文で予約可能列と非予約可能列の更新は不可。DML returning句は非サポート
- 非予約可能列の更新によるロック済みの行での予約可能列の更新はロックフリー予約を取得する
- ジャーナル表に登録された予約は自分のトランザクションの予約のみ表示される。トランザクションの終了時に予約可能列に対して行われた変更を認識する。

ロックフリー列値の予約の制約、ガイドライン

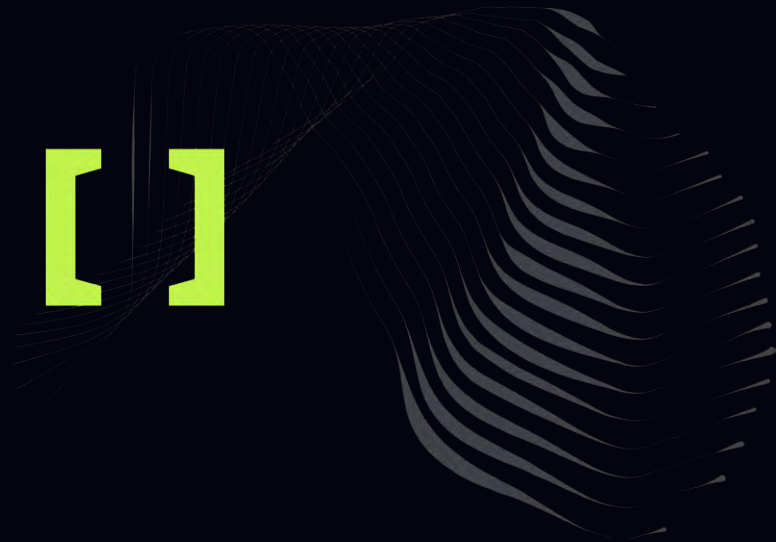
予約可能列を持つ表に対するINSERT/DELETE文のガイドライン

- トランザクションは動作を変更せず、予約可能列の値が含まれる行全体のINSERTが可能
- INSERTされた行はコミットされるまで他のトランザクションに表示されない
- 保留中の予約がある場合はその予約を含むトランザクションが完了しないとDELETEできない。（DELETEは内部的に30秒間隔で再試行される。タイムアウトした場合はリソース・ビジー・エラーが発生する）

予約可能列を持つ表に対する同時DDL文のガイドライン

- DDL文か進行中の場合は、DDLが完了するまで予約可能列に対する更新は不可
- 保留中の予約がある場合、DDLに主キー列または予約可能列に対する変更が含まれているDDL文は許可されない

自動トランザクションロールバック



自動トランザクション・ロールバック

- 概要
  - 行ロックによる待機が発生した場合に、行ロックを保持するトランザクションのセッションを、指定秒経過後、トランザクションの優先度に従って終了し、自動的にロールバックさせる機能
  - トランザクションの優先度は低（low）、中（medium）、高（high）（デフォルト）からセッション単位で指定
    - 高のトランザクションは終了されず行ロックを保持
  - 行ロック取得中の低優先度トランザクションを何秒待機するかは設定可能

- メリット
  - 優先度が高いトランザクションを他のトランザクションの行ロックによる待機から人手を介さず、開放することができる

自動トランザクション・ロールバックのイメージ

```
create table mycheck (value number);
insert into mycheck values (1);
commit;
alter system set txn_auto_rollback_high_priority_wait_target = 20;
```

時間

優先度lowのトランザクションTxn1	優先度lowのトランザクションTxn2	優先度Highのトランザクション Txn3
alter session set txn_priority=low	**alter session set txn_priority=low**	**--設定なし(txn_priority=high)**
update scott.mycheck set value=0;		
1 rows updated.	--Txn1のupdate実行後に実行	
--1レコードを更新	update scott.mycheck set value=10	--Txn2のupdate実行後に実行
	--Txn1の行ロックを待機	update scott.mycheck set value=1000;
		--txn_auto_rollback_high_priority_wait_target の20秒経過
(rollback、画面表示はなし) ←		--txn1をロールバック、セッション強制終了
	--Txn1のセッション終了後、更新される	
	1 rows updated	--txn_auto_rollback_high_priority_wait_target の20秒経過
	(rollback、画面表示はなし) ←	--txn2をロールバック、セッション強制終了
		--Txn2のセッション終了後、更新される
		1 rows updated.
-- セッションが強制終了されたためORA-3113が返される。実行してはじめてエラーがわかる。	--セッションが強制終了されたためORA-3113が返される。実行してはじめてエラーがわかる。	select * from scott.mycheck;
select * from scott.mycheck;	select * from scott.mycheck;	VALUE
ORA-03113: end-of-file on communication channel	ORA-03113: end-of-file on communication channel	----------
		1000

自動トランザクション・ロールバック

High、Lowの優先度の待機時間設定（秒）

TXN_AUTO_ROLLBACK_HIGH_PRIORITY_WAIT_TARGET = 1 ～2147483647（デフォルト）

TXN_AUTO_ROLLBACK_MEDIUM_PRIORITY_WAIT_TARGET = 1 ～2147483647（デフォルト）

- alter systemで指定

セッションの優先度の指定

TXN_PRIORITY = { HIGH （デフォルト） | MEDIUM | LOW}

- alter sessionで指定、alter systemでは指定できない

設定の確認はV$TRANSACTIONの列で可

- TX_PRIORITY
- TX_PRIORITY_WAIT_TARGET

自動トランザクション・ロールバックのモード指定（オプション）

txn_auto_rollback_mode = { ROLLBACK （デフォルト） | TRACK }

TRACKはROLLBACK（自動トランザクション・ロールバックの有効化）にする前の検証に有効。実際にロールバックされるわけではなく、次ページの統計情報にある統計値のみがカウントされる

- alter systemで指定

自動トランザクション・ロールバックの監視

待機イベント

通常の行ロックの待機イベント（enq: TX - row lock contention）の後に優先度が追加される

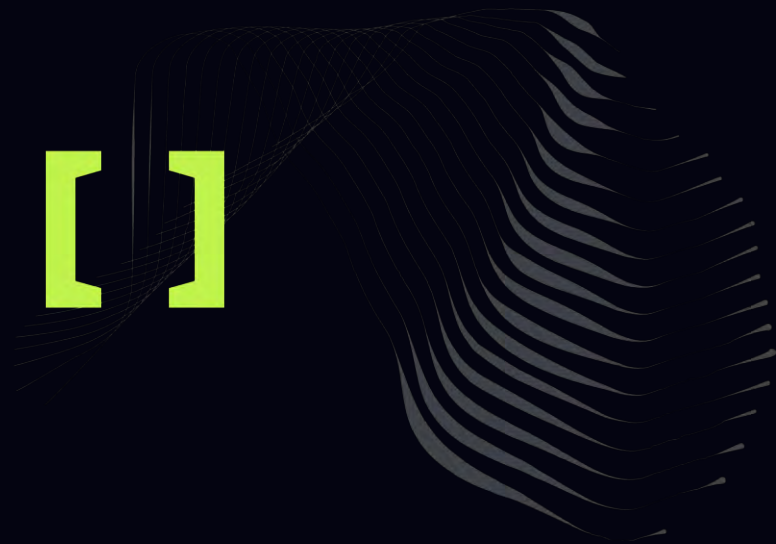
```
SQL> select SID, EVENT, SECONDS_IN_WAIT, BLOCKING_SESSION from v$session where event like '%enq%';
SID EVENT SECONDS_IN_WAIT BLOCKING_SESSION
---------- ---------------------------------------- -------------- ---------------
26 enq: TX - row lock contention (MEDIUM priority) 74 54
56 enq: TX - row lock contention (HIGH priority) 96 54
201 enq: TX - row lock contention (LOW priority) 4
```

アラートログ：終了させられたトランザクションの情報が出力される

統計情報

- ROLLBACKモードの場合は自動トランザクション・ロールバックが発生するたび以下の統計が増加
  - **txns rollback** txn_auto_rollback_high_priority_wait_target
  - **txns rollback** txn_auto_rollback_medium_priority_wait_target
- TRACKモードの場合は自動トランザクション・ロールバックが発生するたび以下の統計が増加
  - **txns track** txn_auto_rollback_high_priority_wait_target
  - **txns track** txn_auto_rollback_medium_priority_wait_target

幅の広い表（表/ビューの最大列数の4096列への拡張）



幅の広い表（表/ビューの最大列数の4096列への拡張）

- 概要
  - 表またはビューで使用できる列の最大数が従来の1000列から4096列に増加
  - 従来の1000列の制限を超える属性を1つのテーブルに格納できるアプリケーションを構築することが可能

- メリット
  - より多くの属性を1つの行に格納できるようになり、アプリケーションによっては設計・実装が簡略化される
  - 活用先として、さまざまな分析軸をもつDWH、メインフレームからの移行、機械学習のモデルが考えられる

幅の広い表（表/ビューの最大列数の4096列への拡張）

- 設定方法
  - 初期化パラメータにてMAX_COLUMNSをEXTENDEDに設定（デフォルトはSTANDARD）
    - COMPATIBLE初期化パラメータを23.0.0.0以上
  - PDBレベルで設定可能、再起動必要

```
SQL> alter system set max_columns=EXTENDED scope=spfile;

System altered.
SQL> shutdown immediate
Pluggable Database closed.
SQL> startup
Pluggable Database opened.
SQL> show parameter max_columns

NAME                                 TYPE        VALUE
------------------------------------ ----------- ------------------------------
max_columns                          string      EXTENDED
```

幅の広い表（表/ビューの最大列数の4096列への拡張）

- 初期化パラメータにてMAX_COLUMNSをSTANDARD戻すには、1001列以上のカラムをもつ表またはビューがない場合のみ

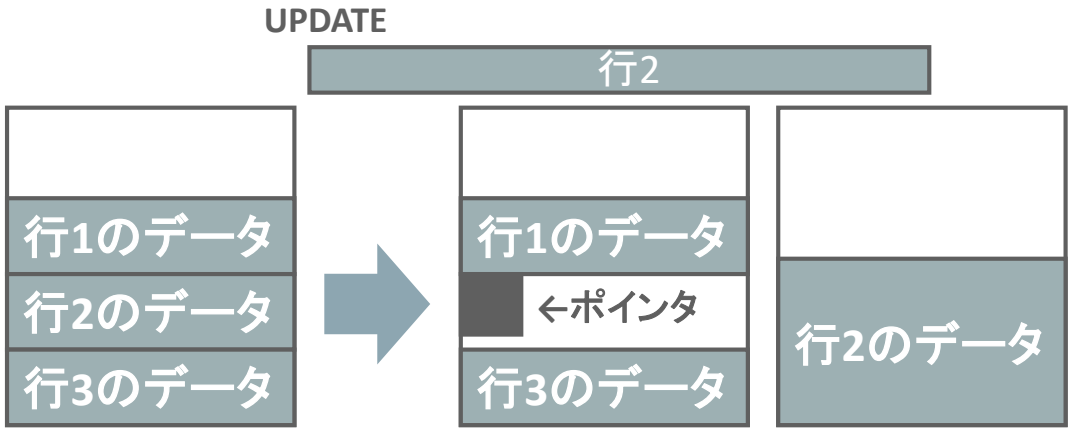
```
--1001列以上の表を持つ状態でmax_columns=STANDARDに戻そうとするとエラー
SQL> alter system set max_columns=STANDARD scope=spfile;
alter system set max_columns=STANDARD scope=spfile
*
ERROR at line 1:
ORA-32017: failure in updating SPFILE
ORA-60471: max_columns can not be set to STANDARD as there are one or more objects
with more than 1000 columns
Help: https://docs.oracle.com/error-help/db/ora-32017/
```

- 1001列以上の表またはビューへのアクセスする場合、クライアントは23c以上である必要があります
- ブロックサイズによらず1001列以上の表は作成可能。ブロックサイズに対してレコード長が長い場合、行移行、行連鎖が発生しやすくなる点には注意

ご参考：行移行、行連鎖

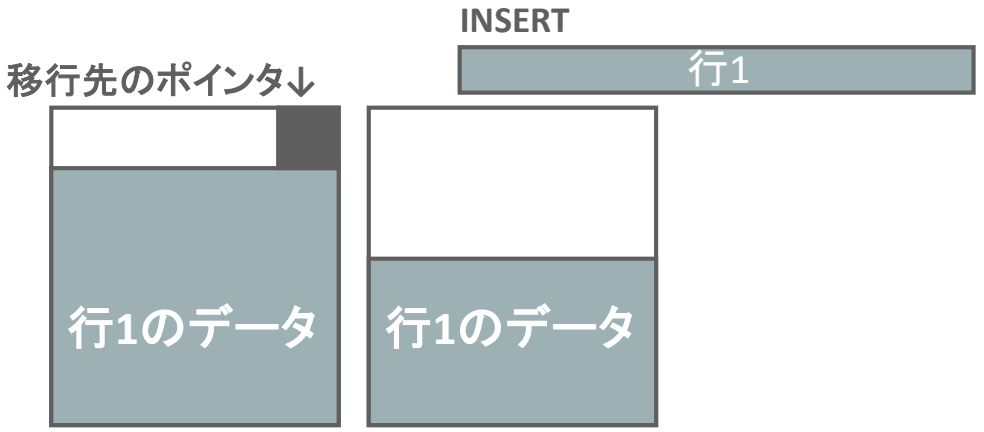
行移行(UPDATE/INSERTで発生)

　更新行や新規行がそのブロックに入りきらなくなった際、該当行のデータをすべて新規ブロックに移動

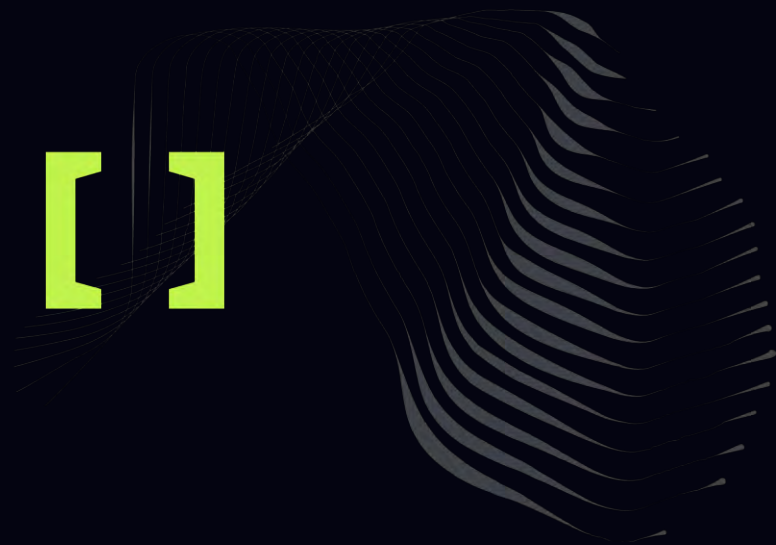


行連鎖(INSERTで発生)

　行が1ブロックに入りきらない場合に複数のブロックにまたがって格納



無制限のパラレルDML/ダイレクトロード（トランザクション制約の解除）



無制限のパラレルDML/ダイレクトロード

パラレルDML/ダイレクトロード実行後、コミット前に同一オブジェクトに対して読取や変更が可能に

- 概要
  - パラレルDMLや、ダイレクトロードを実行した表に対して、コミット/ロールバック前に、問合せまたは変更処理を実施すると、エラーとなる制約（※）があったが、その制約がなくなった

    ※同じテーブルに対するパラレルDMLは、トランザクションの最初でかつ唯一である必要があるという制約。ダイレクトロードに関してはシリアルDMLの後、コミット前にダイレクトロードを実行することは可能

- メリット
  - 従来では制約のために、本来意図したトランザクションを変更して、パラレルDMLやダイレクト・パス・インサートを実行前後に一度コミットする、また実行対象の表へのSQL処理はしないように制御する必要があったが、その制約がなくなり、活用の幅が広がった

無制限のパラレルDML
ユースケースと制限

- ユースケース

  以下、パラレルDML処理が活用できるケースにて活用の幅が広がります

  - DWHでの表のリフレッシュ
  - 中間サマリー表の作成
  - スコアリング・テーブルの使用
  - 履歴表の更新
  - バッチ・ジョブの実行

- 制限

  以下条件化では、無制限のパラレルDMLが有効になりません

  - 表がIOTまたはクラスタ化表
  - 表領域が自動セグメント領域管理（ASSM）管理ではない。一時表はASSM対象外
  - 表領域のエクステント管理がUNIFORMである場合

ご参考：パラレルDML設定/ダイレクト・パス・インサート設定

パラレルDMLの設定には、2通りの方法があります

- パラレルDMLのENABLE＋パラレル設定
  - ENABLE：ALTER SESSION ENABLE PARALLEL DML、またはENABLE_PARALLEL_DML SQLヒント
  - パラレル設定：表のPARALLEL属性の付与、またはPARALLELヒント
    問い合わせのパラレル化とは違い、パラレルDMLのセッションまたはヒントでのENABLE指定が必要
- もしくは、ALTER SESSION FORCE PARALLEL DMLを指定

- パラレルDMLにおいて、インサート実行では、ダイレクト・パス・インサートが実行されます。従来型インサートをパラレル実行したい場合は、NOAPPENDヒントで上書きすることで実現できます

```
SQL> INSERT /*+ NOAPPEND PARALLEL */ INTO sales_hist SELECT * FROM sales;
```

ダイレクト・パス・インサートの設定

- APPEND ヒント句をINSERT 文に付与することで使用できます

```
SQL> INSERT /*+ APPEND */ INTO sales_hist SELECT * FROM sales;
```

パラレルDML（従来での制約、エラー内容）

23cでなくなった制限の確認

- パラレルDMLの後にselectやDML実行は従来エラーで失敗していたが、23cでは可能に

```
SQL> ALTER SESSION FORCE PARALLEL DML PARALLEL 4 ;
SQL> INSERT INTO EMP SELECT * FROM EMP;
--従来はエラーで失敗
SQL> SELECT COUNT(*) FROM EMP;
ORA-12838: cannot read/modify an object after modifying it in parallel
--23cでは成功
SQL> SELECT COUNT(*) FROM EMP;
  COUNT(*)
----------
        28
```

- シリアルDML実行したあとにパラレルDML実行は従来エラーで失敗していたが、23cでは可能に

```
SQL> INSERT INTO EMP SELECT * FROM EMP;
--従来はエラーで失敗
SQL>  INSERT /*+ ENABLE_PARALLEL_DML PARALLEL(EMP 4) */ INTO EMP SELECT * FROM EMP;
ERROR:
ORA-12841: Cannot alter the session parallel DML state within a transaction
--23cでは成功
SQL> INSERT /*+ ENABLE_PARALLEL_DML PARALLEL(EMP 4) */ INTO EMP SELECT * FROM EMP;
28 rows created.
```

ダイレクトロード（従来での制約、エラー内容）

23cでなくなった制限

- 従来は、ダイレクトロードの後にSELECT/DMLでエラー

```
SQL> INSERT /*+ APPEND */ INTO EMP SELECT * FROM EMP;
SQL> SELECT COUNT(*) FROM EMP;
ERROR at line 1:
ORA-12838: cannot read/modify an object after modifying it in parallel

-- 23cだと成功
SQL> SELECT COUNT(*) FROM EMP;
  COUNT(*)
----------
        28
```

- DMLの後にダイレクトロードは従来から可能

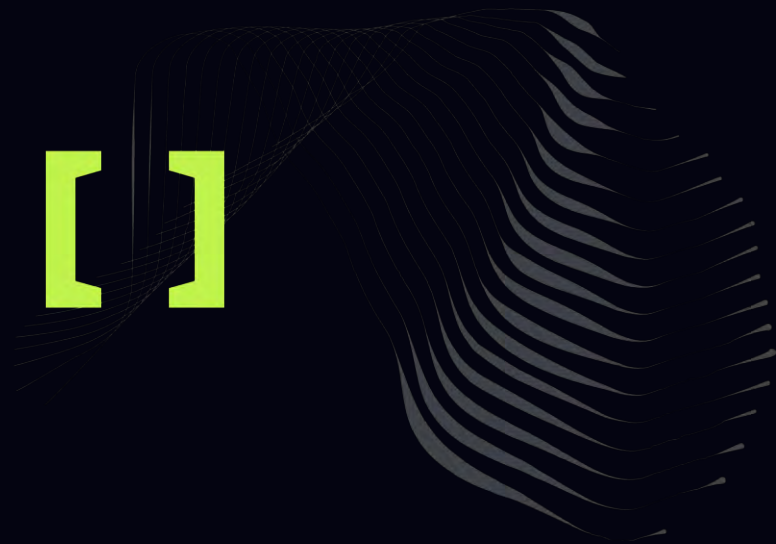
```
SQL> INSERT INTO EMP SELECT * FROM EMP;
14 rows created.

--こちらは従来から可能
SQL> INSERT /*+ APPEND */ INTO EMP SELECT * FROM EMP;
28 rows created.
```

Oracle Database 23c におけるマテリアライズド・ビュー強化

マテリアライズド・ビューとは

Oracle Database 23c におけるマテリアライズド・ビュー強化

- マテリアライズド・ビューでのANSI結合のサポート
- 論理パーティション・チェンジ・トラッキング
- マテリアライズド・ビューの同時リフレッシュ

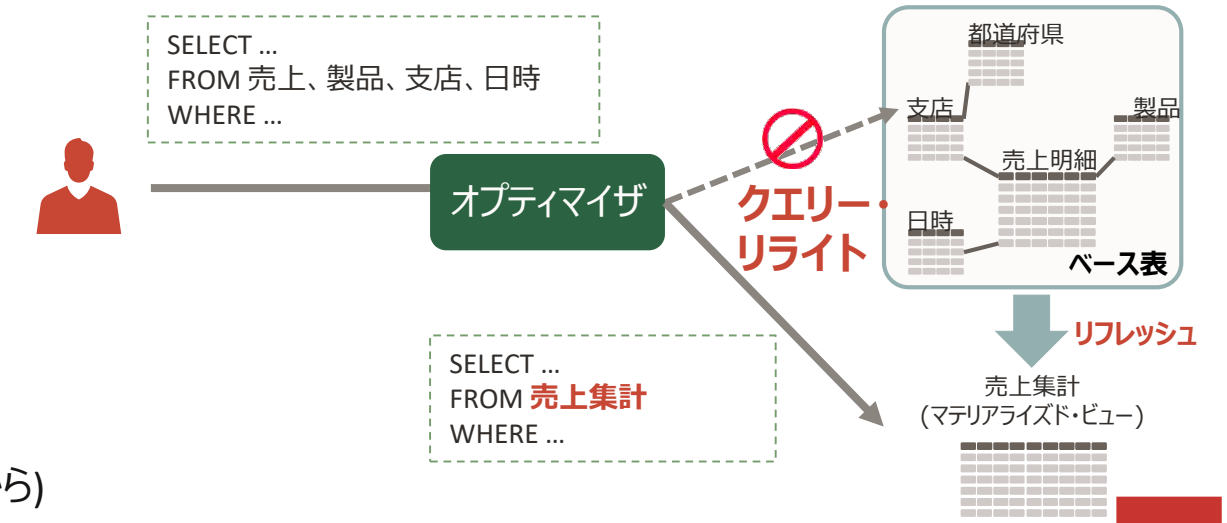


マテリアライズド・ビューとは

マテリアライズド・ビュー (MView) とは

クエリの結果セットを事前に保持し、分散環境でのレプリケーションやDWHでの検索処理時間短縮を図るスキーマ・オブジェクト (集計を含むMView、結合のみを含むMView、ネステッドMViewのタイプがある)

- クエリ・リライト
  - ベース表に対するクエリをMViewに対するクエリに書き換えて実行する (これでSQLチューニングとして使用可能)
    - デフォルトではコストベースでクエリ・リライトを行う (初期化パラメータQUERY_REWRITE_ENABLED=TRUE)
  - デフォルトではMViewが最新のデータでないとリライトしない (初期化パラメータQUERY_REWRITE_INTEGRITY)
    - PCT (Partition Change Tracking) リライトのときはパーティション単位に最新が識別される
    - リアルタイムMView(12.2から)により、失効したMViewでも**問合せ時計算**(MViewログの差分をマージ)で問合せ結果を生成
- リフレッシュ
  - MViewデータを最新の状態にする
  - リフレッシュ・タイプ (メソッド)
    - 完全リフレッシュ (すべてのデータを再構築する)
    - 高速リフレッシュ (変更部分だけをリフレッシュする)
      - ログベース、PCT (増分リフレッシュとも呼ぶ)
  - リフレッシュ・モード (動作するタイミング)
    - ON COMMIT、ON DEMAND、ON STATEMENT(12.2から)



マテリアライズド・ビュー (MView) とは
クエリ・リライト

問合せをリライトする場合の判断に使用する2つの方法

- **テキスト一致リライト**
  - 問合せとMView定義のテキストを比較して判断する
    - テキストの完全一致、部分一致 (FROM句以降のテキスト)
  - 最も簡単だが使用できる問合せ数は多くない
- **一般的なクエリ・リライト**
  - テキスト一致リライトが不可能な場合に行われる方法
  - 結合、データ列、グルーピング列および集計関数を比較してMViewから導出可能かのチェックが行われる
    - 結合 (結合互換性、後で説明)
    - データ列 (列が存在するか後戻り結合)
    - グルーピング列 (両方にGroup Byが存在する場合に実施、同じか上位レベル)
    - 集計関数 (集計可能性)
  - オプティマイザでは、主キー制約、外部キー制約、ディメンション・オブジェクトなど、依存可能なデータ関係を使用する

クエリ・リライトの整合性レベル

- 初期化パラメータQUERY_REWRITE_INTEGRITY
  - **ENFORCED（デフォルト）**
    - MViewが最新データで、全ての一貫性および整合性が施行され保証された場合のみがリライトされる
      - ENABLED VALIDATEDの主キー/外部キー制約に基づいたリレーションシップのみが使用される
  - **TRUSTED**
    - MViewが最新データで、ディメンションおよびRELY制約で宣言されたリレーションシップが正しいと信頼して、リライトされる
  - **STALE_TOLERATED**
    - 施行されないリレーションシップを使用してリライトできるようになり、MViewがベース表と矛盾する場合でもリライトされる

マテリアライズド・ビュー (MView) とは
クエリ・リライト（一般的なクエリ・リライト）

集計可能性 (Aggregate Computability)

- MViewに含まれている集計関数から、導出または計算可能なときのリライト (SUMとCOUNTのMViewに対してAVG集計)

集計ロールアップ (Aggregate Rollup)

- MViewに格納されているグルーピングから、ロールアップ (上位レベルのグルーピング) が可能なときのリライト

データのフィルタリング

- フィルタリングのデータ・サブセットを取得可能な (フィルター列が存在する) ときのリライト

ディメンションを使用したロールアップ

- MViewにない列を、ディメンション (CREATE DIMENSION) を使用してロールアップすることが可能なときのリライト
  - ディメンションのHIERARCHY句 (列の階層レベル) に定義されていれば、MViewにない列でもロールアップできる

後戻り結合 (Join Back)

- MViewにない列を、再結合して求めることが可能なときのリライト (結合済みの表を再結合することから後戻り結合)
  - 主キーまたはディメンションのDETERMINES句 (MViewにある列Aはない列Bを決定すると定義) を使用

複数のMViewを使用したリライト (Oracle10gから)

- 一つのMViewでは解決できない場合に、複数のMView (またはMViewとベース表) を使用したUNION ALLクエリから求めることが可能なときのリライト

パーティション・チェンジ・トラッキング (PCT) リライト

- パーティション単位に最新が識別され、最新のパーティションだけリライト (ハッシュ・パーティションは動作しない)
  - QUERY_REWRITE_INTEGRITY=STALE_TOLERATEDではPCTリライトは使用されない (データが最新かどうかが考慮されないため、最新でなくてもリライトする)

※「ANSI結合のサポート」で例を使用して解説

マテリアライズド・ビュー (MView) とは
クエリ・リライト（結合互換性チェック）

問合せの結合がMViewの結合と以下の比較を行う

- **共通結合** (両方に結合があり共通または導出可能か)
  - 内部結合やアンチ結合は外部結合 (結合のみのMView) から導出可能
  - セミ結合は結合のみの内部結合や外部結合から導出可能
- **問合せデルタ結合** (問合せだけに結合がある)
  - MViewにない結合を導出可能か
    - MViewに結合列が含んでいると導出可能
- **MViewデルタ結合** (MViewだけに結合がある)
  - 結合がなくなってもベース表と同じ結果になるか
    - ロスレス結合 (データ消失なし) と非重複結合が必要 (結合してもデータ消失なく、データ増加もない)
      - 内部結合では外部キーに外部キー制約とNOT NULL制約
        - 主キーと外部キー(NOT NULL制約あり)の関係によって、外部キー表の各行が主キー表の1行と正確に結合される
      - 外部結合 (結合のみのMView) では主キーに主キー制約
        - 外部結合のouter表は全ての行を保持、主キーで非重複結合

共通結合

- 内部結合の問合せを外部結合のMViewにリライト

```
CREATE MATERIALIZED VIEW join_sales_time_product_oj_mv
  ENABLE QUERY REWRITE AS
  SELECT p.prod_id, p.prod_name, t.time_id, t.week_ending_day,
         s.amount_sold
  FROM   sales s, products p, times t
  WHERE  s.time_id=t.time_id AND s.prod_id=p.prod_id(+);

SQL> SELECT p.prod_name, t.week_ending_day, SUM(s.amount_sold)
  2   FROM sales s, products p, times t
  3  WHERE s.time_id=t.time_id AND s.prod_id = p.prod_id
  4  GROUP BY p.prod_name, t.week_ending_day;

Execution Plan
----------------------------------------------------------------------------
| Id  | Operation                      | Name                             |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                                  |
|   1 |  HASH GROUP BY                 |                                  |
|*  2 |   MAT_VIEW REWRITE ACCESS FULL| JOIN_SALES_TIME_PRODUCT_OJ_MV|
----------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
    2 - filter("JOIN_SALES_TIME_PRODUCT_OJ_MV"."PROD_ID" IS NOT NULL)
```

マテリアライズド・ビュー (MView) とは
クエリ・リライト（結合互換性チェック）

- MViewデルタ結合のクエリ・リライト
  - MViewだけにある結合を無くしても同じ結果になるか
  - 外部キー(s.prod_id)に外部キー制約とNOT NULL制約があるので内部結合のMViewにリライトする

```
CREATE MATERIALIZED VIEW join_sales_time_product_mv
   ENABLE QUERY REWRITE AS
   SELECT p.prod_id, p.prod_name, t.time_id, t.week_ending_day,
          s.cust_id, s.amount_sold
   FROM sales s JOIN products p ON (s.prod_id = p.prod_id)
                JOIN times t ON (s.time_id = t.time_id);

SQL> SELECT t.week_ending_day, SUM(s.amount_sold)
  2    FROM sales s JOIN times t ON (s.time_id = t.time_id)
  3   GROUP BY week_ending_day;

Execution Plan
----------------------------------------------------------------------
| Id  | Operation                     | Name                         |
----------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                              |
|   1 |  HASH GROUP BY                |                              |
|   2 |   MAT_VIEW REWRITE ACCESS FULL| JOIN_SALES_TIME_PRODUCT_MV |
----------------------------------------------------------------------
```

- 問合せデルタ結合のクエリ・リライト
  - 問合せだけにある結合をMViewの結合列(s.cust_id)から導出する

```
SQL> SELECT p.prod_name, t.week_ending_day, c.cust_city,
  2         SUM(s.amount_sold)
  3    FROM sales s JOIN products p ON (s.prod_id = p.prod_id)
  4                 JOIN times t ON (s.time_id = t.time_id)
  5                 JOIN customers c ON (s.cust_id = c.cust_id)
  6   GROUP BY p.prod_name, t.week_ending_day, c.cust_city;

Execution Plan
----------------------------------------------------------------------
| Id  | Operation                     | Name                         |
----------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                              |
|   1 |  HASH GROUP BY                |                              |
|*  2 |   HASH JOIN                   |                              |
|   3 |    MAT_VIEW REWRITE ACCESS FULL| JOIN_SALES_TIME_PRODUCT_MV |
|   4 |    TABLE ACCESS FULL          | CUSTOMERS                    |
----------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("JOIN_SALES_TIME_PRODUCT_MV"."CUST_ID"="C"."CUST_ID")
```

マテリアライズド・ビュー (MView) とは
クエリ・リライトの制約事項

クエリ・リライトのタイプ	ディメンション	主キー/外部キー制約、 外部キーのNOT NULL制約	QUERY_REWRITE_INTEGRITY
テキスト一致	必要なし	必要なし	制限なし
後戻り結合 (ディメンションを使用) 後戻り結合 (主キーを使用)	必須 必要なし	RELY制約または必要なし 必要あり	TRUSTEDまたはSTALE_TOLERATED 制限なし
集計可能性	必要なし	必要なし	制限なし
集計ロールアップ	必要なし	必要なし	制限なし
ディメンションを使用したロールアップ	必須	RELY制約または必要なし	TRUSTEDまたはSTALE_TOLERATED
データのフィルタリング	必要なし	必要なし	制限なし
PCTリライト	必要なし	必要なし	ENFORCEDまたはTRUSTED
複数のMView	必要なし	必要なし	制限なし

マテリアライズド・ビュー (MView) とは
MViewのリフレッシュ

リフレッシュ・タイプ

- 完全リフレッシュ (COMPLETE)
  - すべてのデータを再構築する
- 高速リフレッシュ (FAST)
  ログベース、PCTの順番で実行する (増分リフレッシュとも呼ぶ)
  - ログベースの高速リフレッシュ
    - ベース表に対する変更は、MViewログまたはダイレクト・ローダー・ログに記録され、MViewを増分的にリフレッシュできるようにする
  - Partition Change Trackingリフレッシュ (FAST_PCT)
    - 実表がパーティション化している場合に、変更された実表のパーティションを使用して、MView内の影響を受けるパーティションまたはデータ部分をリフレッシュする
- 強制リフレッシュ (FORCE)
  - 高速リフレッシュが可能なときは高速リフレッシュを実行し、可能でないときは完全リフレッシュを実行する (これがデフォルト)

リフレッシュ・モード

- ON COMMITリフレッシュ・モード
  - トランザクションのコミットで更新するので、MViewが最新であることが保証される (コミット操作の実行時間が長くなる)
  - 高速リフレッシュが可能なときのみ
- ON DEMANDリフレッシュ・モード (デフォルト)
  - 実行したいときに、DBMS_MVIEWパッケージと12.1からのDBMS_SYNC_REFRESHパッケージを使用して行う
    - 12.1からDBMS_MVIEWパッケージで**ホーム外リフレッシュ** (外部の表をリフレッシュ後にMViewと切り替える) を各リフレッシュ・タイプで可能に (これまでのMViewを直接リフレッシュするものを**ホーム内リフレッシュ**と呼ぶ)
- 自動リフレッシュ(START WITH <DATE>/ NEXT <DATE>)
  - リフレッシュをSTART WITHから開始、NEXT間隔で継続する
- ON STATEMENTリフレッシュ・モード (12.2から)
  - SQL文ごとにリフレッシュが可能で、ロールバックするとMViewもロールバックされる (DML操作の実行時間が長くなる)
  - 高速リフレッシュが可能なとき (MViewログは必要ない)

Oracle Database 23c におけるマテリアライズド・ビュー強化

Oracle Database 23c におけるマテリアライズド・ビュー強化
ポイントとなる新機能

クエリ・リライトの機能拡張

- マテリアライズド・ビューでのANSI結合のサポート
  - ANSI結合文による完全なクエリ・リライトをサポート
- 論理パーティション・チェンジ・トラッキング
  - より効率的なMViewのリフレッシュと失効追跡のためのLPCTを追加（LPCTリライト）

リフレッシュの機能拡張

- 論理パーティション・チェンジ・トラッキング
  - より効率的なMViewのリフレッシュと失効追跡のためのLPCTを追加（LPCTリフレッシュ）
- マテリアライズド・ビューの同時リフレッシュ
  - 同時リフレッシュの提供（同じオンコミットMViewをシリアライズの必要ない同時リフレッシュが可能に）

マテリアライズド・ビューでのANSI結合のサポート

マテリアライズド・ビューでのANSI結合のサポート

ANSI結合構文による完全なクエリ・リライトをサポート

機能概要

- これまで、ANSI結合構文のMViewはテキスト一致クエリ・リライト (テキスト完全一致またはテキスト部分一致) のみであったが、すべてのクエリ・リライトが可能になった
  - MView側がOracle結合文であればクエリ側がANSI結合文でもすべてのクエリ・リライトが可能であった

メリット

- ANSI結合構文で作成したMViewのクエリ・リライト範囲が拡大されて使用しやすくなり、パフォーマンスも大幅に向上する (MViewの数を削減できる)
  - 多くの問合せ(特にSQLツールおよびレポートで生成される問合せ)では、ANSI結合構文が使用されている

利用イメージ (Oracle結合構文とANSI結合構文)

```
-- Oracle結合構文
CREATE MATERIALIZED VIEW cust_sales_mv
 PARALLEL
 BUILD IMMEDIATE
 REFRESH COMPLETE
 ENABLE QUERY REWRITE AS
 SELECT c.cust_last_name,
        SUM(amount_sold) AS sum_amount_sold
 FROM customers c, sales s WHERE s.cust_id = c.cust_id
 GROUP BY c.cust_last_name;
```

```
-- ANSI結合構文
CREATE MATERIALIZED VIEW cust_sales_mv
 PARALLEL
 BUILD IMMEDIATE
 REFRESH COMPLETE
 ENABLE QUERY REWRITE AS
 SELECT c.cust_last_name,
        SUM(amount_sold) AS sum_amount_sold
 FROM customers c JOIN sales s ON s.cust_id = c.cust_id
 GROUP BY c.cust_last_name;
```

マテリアライズド・ビューでのANSI結合のサポート
集計ロールアップとデータのフィルタリングのクエリ・リライト)

```
SQL> CREATE MATERIALIZED VIEW sum_sales_pscat_week_mv
  2    ENABLE QUERY REWRITE AS
  3    SELECT p.prod_subcategory, t.week_ending_day,
  4           SUM(s.amount_sold) AS sum_amount_sold,
  5           COUNT(s.amount_sold) AS count_amount_sold
  6    FROM sales s JOIN products p ON (s.prod_id = p.prod_id)
  7                 JOIN times t ON (s.time_id = t.time_id)
  8    GROUP BY p.prod_subcategory, t.week_ending_day;

SQL> SELECT t.week_ending_day, SUM(s.amount_sold) AS sum_amount
  2  FROM sales s JOIN products p ON (s.prod_id = p.prod_id)
  3               JOIN times t ON (s.time_id = t.time_id)
  4  WHERE p.prod_subcategory='Camera Media'
  5  GROUP BY t.week_ending_day ;

Execution Plan
----------------------------------------------------------------------
| Id  | Operation                     | Name                        |
----------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                             |
|   1 |  HASH GROUP BY                |                             |
|*  2 |   MAT_VIEW REWRITE ACCESS FULL| SUM_SALES_PSCAT_WEEK_MV |
----------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("SUM_SALES_PSCAT_WEEK_MV"."PROD_SUBCATEGORY"=
              'Camera Media')
```

- テキスト完全一致のクエリ・リライト

```
SQL> SELECT p.prod_subcategory, t.week_ending_day,
  2         SUM(s.amount_sold) AS sum_amount_sold,
  3         COUNT(s.amount_sold) AS count_amount_sold
  4  FROM sales s JOIN products p ON (s.prod_id = p.prod_id)
  5               JOIN times t ON (s.time_id = t.time_id)
  6  GROUP BY p.prod_subcategory, t.week_ending_day ;

Execution Plan
----------------------------------------------------------------------
| Id  | Operation                     | Name                        |
----------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                             |
|   1 |  MAT_VIEW REWRITE ACCESS FULL| SUM_SALES_PSCAT_WEEK_MV |
----------------------------------------------------------------------
```

- テキスト部分一致のクエリ・リライト

```
SQL> SELECT p.prod_subcategory, t.week_ending_day,
  2          AVG(s.amount_sold) AS avg_sales
  3  FROM sales s JOIN products p ON (s.prod_id = p.prod_id)
  4               JOIN times t ON (s.time_id = t.time_id)
  5  GROUP BY p.prod_subcategory, t.week_ending_day ;
```

実行計画は同じ

マテリアライズド・ビューでのANSI結合のサポート
ディメンションを使用したロールアップのクエリ・リライト

- ディメンションのHIERARCHY句 (列の階層レベル) を使用してロールアップする例
  - 以下のようなプロダクトのディメンション (カテゴリ、サブカテゴリ、プロダクトの階層) を作成
  - QUERY_REWRITE_INTEGRITYを'ENFORCED'以外にする必要がある

```
CREATE MATERIALIZED VIEW sum_sales_pscat_week_mv
  ENABLE QUERY REWRITE AS
  SELECT p.prod_subcategory, t.week_ending_day,
         SUM(s.amount_sold) AS sum_amount_sold,
         COUNT(s.amount_sold) AS count_amount_sold
  FROM sales s JOIN products p ON (s.prod_id = p.prod_id)
               JOIN times t ON (s.time_id = t.time_id)
  GROUP BY p.prod_subcategory, t.week_ending_day;

CREATE DIMENSION products_dim
  LEVEL product      IS (products.prod_id)
  LEVEL subcategory  IS (products.prod_subcategory)
  LEVEL category     IS (products.prod_category)
  HIERARCHY prod_rollup (product CHILD OF
                         subcategory CHILD OF
                         category)
  ATTRIBUTE subcategory DETERMINES products.prod_subcat_desc;
```

- MViewにないprod_categoryでグルーピングを実行した問合せ

```
SQL> SELECT p.prod_category, t.week_ending_day,
  2         SUM(s.amount_sold) AS sum_amount
  3   FROM sales s JOIN products p ON (s.prod_id = p.prod_id)
  4                JOIN times t ON (s.time_id = t.time_id)
  5  GROUP BY p.prod_category, t.week_ending_day;

Execution Plan
----------------------------------------------------------------------
| Id  | Operation                      | Name                       |
----------------------------------------------------------------------
|   0 | SELECT STATEMENT               |                            |
|   1 |  HASH GROUP BY                 |                            |
|*  2 |   HASH JOIN                    |                            |
|   3 |    MAT_VIEW REWRITE ACCESS FULL| SUM_SALES_PSCAT_WEEK_MV    |
|   4 |    VIEW                        |                            |
|   5 |     HASH UNIQUE                |                            |
|   6 |      TABLE ACCESS FULL         | PRODUCTS                   |
----------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("from$_subquery$_012"."PROD_SUBCATEGORY"=
              "SUM_SALES_PSCAT_WEEK_MV"."PROD_SUBCATEGORY")
```

マテリアライズド・ビューでのANSI結合のサポート
主キーを使用した後戻り結合のクエリ・リライト

- MViewにない列を、再結合して求めることが可能なときのリライト (主キーの使用例)
  - 結合済みであるが列がないので再結合 (後戻り結合)
    - MViewはproducts表と結合している
  - MViewにはproducts表の主キー (p.prod_id) がある

```
CREATE MATERIALIZED VIEW sum_sales_prod_week_mv
  ENABLE QUERY REWRITE AS
  SELECT p.prod_id, t.week_ending_day, s.cust_id,
         SUM(s.amount_sold) AS sum_amount_sold
  FROM sales s join products p on (s.prod_id=p.prod_id)
               join times t on (s.time_id=t.time_id)
  GROUP BY p.prod_id, t.week_ending_day, s.cust_id;
```

- 再度products表に主キーで結合している

```
SQL> SELECT p.prod_id, t.week_ending_day,
  2         SUM(s.amount_sold) AS sum_amount
  3    FROM sales s JOIN products p ON (s.prod_id = p.prod_id)
  4                 JOIN times t ON ( s.time_id = t.time_id)
  5   WHERE p.prod_subcat_desc LIKE '%Men'
  6   GROUP BY p.prod_id, t.week_ending_day;

Execution Plan
----------------------------------------------------------------------
| Id  | Operation                     | Name                  |
----------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                       |
|*  1 |  HASH JOIN                    |                       |
|*  2 |   TABLE ACCESS FULL           | PRODUCTS              |
|   2 |   MAT_VIEW REWRITE ACCESS FULL| SUM_SALES_PROD_WEEK_MV |
----------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("P"."PROD_ID"="SUM_SALES_PROD_WEEK_MV"."PROD_ID")
   2 - filter("P"."PROD_SUBCAT_DESC" LIKE '%Men')
```

マテリアライズド・ビューでのANSI結合のサポート
ディメンションのDETERMINES句を使用した後戻り結合のクエリ・リライト

- MViewにない列を、再結合して求めることが可能なときのリライト (ディメンションのDETERMINES句の使用例)
  - prod_subcategoryはprod_subcat_descを決定すると定義する
  - 重複値が含まれるなど主キーにできない列の場合

```
CREATE MATERIALIZED VIEW sum_sales_pscat_week_mv
  ENABLE QUERY REWRITE AS
  SELECT p.prod_subcategory, t.week_ending_day,
         SUM(s.amount_sold) AS sum_amount_sold,
         COUNT(s.amount_sold) AS count_amount_sold
  FROM sales s JOIN products p ON (s.prod_id = p.prod_id)
               JOIN times t ON (s.time_id = t.time_id)
  GROUP BY p.prod_subcategory, t.week_ending_day;

CREATE DIMENSION products_dim
  LEVEL product      IS (products.prod_id)
  LEVEL subcategory  IS (products.prod_subcategory)
  LEVEL category     IS (products.prod_category)
  HIERARCHY prod_rollup (product CHILD OF
                         subcategory CHILD OF
                         category)
  ATTRIBUTE subcategory DETERMINES products.prod_subcat_desc;
```

- products表にprod_subcategoryで結合している

```
SQL> SELECT p.prod_subcategory, t.week_ending_day,
  2         SUM(s.amount_sold) AS sum_amount
  3    FROM sales s JOIN products p ON (s.prod_id = p.prod_id)
  4                 JOIN times t ON ( s.time_id = t.time_id)
  5   WHERE p.prod_subcat_desc LIKE '%Men'
  6   GROUP BY p.prod_subcategory, t.week_ending_day;

Execution Plan
----------------------------------------------------------------------
| Id  | Operation                     | Name                       |
----------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                            |
|*  1 |  HASH JOIN                    |                            |
|   2 |   MAT_VIEW REWRITE ACCESS FULL| SUM_SALES_PSCAT_WEEK_MV |
|   3 |   VIEW                        |                            |
|   4 |    HASH UNIQUE                |                            |
|*  5 |     TABLE ACCESS FULL         | PRODUCTS                   |
----------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("from$_subquery$_011"."PROD_SUBCATEGORY"=
              "SUM_SALES_PSCAT_WEEK_MV"."PROD_SUBCATEGORY")
   6 - filter("PROD_SUBCAT_DESC" LIKE '%Men')
```

マテリアライズド・ビューでのANSI結合のサポート

ANSI結合構文MViewに対するPCT (リライト、リフレッシュ) はサポートされない

PCTリフレッシュでエラー、PCTリライトされない

```
CREATE TABLE sales_by_time
 (time_id, prod_id, amount_sold, quantity_sold)
 PARTITION BY RANGE (time_id)
 (PARTITION s2018   VALUES LESS THAN (TO_DATE('2019-01-01',...)),
  PARTITION s2019q1 VALUES LESS THAN (TO_DATE('2019-04-01',...)),
  PARTITION s2019q2 VALUES LESS THAN (TO_DATE('2019-07-01',...)),
  PARTITION s2019q3 VALUES LESS THAN (TO_DATE('2019-10-01',...)),
  PARTITION s2019q4 VALUES LESS THAN (TO_DATE('2020-01-01',...)),
  PARTITION s2020   VALUES LESS THAN (MAXVALUE) ) ;

CREATE MATERIALIZED VIEW sales_in_1999_mv
 ENABLE QUERY REWRITE AS
 SELECT s.time_id, s.prod_id, p.prod_name, SUM(quantity_sold)
 FROM sales_by_time s JOIN products p ON (p.prod_id = s.prod_id)
 WHERE s.time_id BETWEEN TO_DATE('1999-01-01','YYYY-MM-DD')
                     AND TO_DATE('1999-12-31','YYYY-MM-DD')
  GROUP BY s.time_id, s.prod_id, p.prod_name;

SQL> exec dbms_mview.refresh ('SALES_IN_1999_MV', method=> 'P');

ORA-12047: PCT FAST REFRESH cannot be used for materialized view
"SYSTEM"."SALES_IN_1999_MV"
```

```
MVIEW_NAME           DETAILOBJ_NAME  PART_NAM PART_NUM FRESHNE
-------------------- --------------- -------- -------- -------
...
SALES_IN_2019_MV     SALES_BY_TIME   S1999Q3         4 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S1999Q4         5 STALE
SALES_IN_2019_MV     SALES_BY_TIME   S2000           6 FRESH

SQL> SELECT s.time_id, p.prod_name, SUM(quantity_sold)
  2    FROM sales_by_time s JOIN
  3         products p ON (p.prod_id = s.prod_id)
  4   WHERE s.time_id BETWEEN TO_DATE('1999-07-01','YYYY-MM-DD')
  5                       AND TO_DATE('1999-07-31','YYYY-MM-DD')
  6   GROUP BY s.time_id, p.prod_name;

-------------------------------------------------------------------
| Id  | Operation                | Name          |Pstart|Pstop |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT         |               |      |      |
|   1 |  HASH GROUP BY           |               |      |      |
|*  2 |   HASH JOIN              |               |      |      |
|   3 |    PARTITION RANGE SINGLE|               |    5 |    5 |
|*  4 |     TABLE ACCESS FULL    | SALES_BY_TIME |    5 |    5 |
|   5 |    TABLE ACCESS FULL     | PRODUCTS      |      |      |
-------------------------------------------------------------------
```

マテリアライズド・ビューでのANSI結合のサポート

補足：Oracle結合構文のパーティション・チェンジ・トラッキング・リライト

```
CREATE TABLE sales_by_time (time_id, prod_id, amount_sold, quantity_sold)
  PARTITION BY RANGE (time_id)
  ( PARTITION s2018   VALUES LESS THAN (TO_DATE('2019-01-01','YYYY-MM-DD')),
    PARTITION s2019q1 VALUES LESS THAN (TO_DATE('2019-04-01','YYYY-MM-DD')),
    PARTITION s2019q2 VALUES LESS THAN (TO_DATE('2019-07-01','YYYY-MM-DD')),
    PARTITION s2019q3 VALUES LESS THAN (TO_DATE('2019-10-01','YYYY-MM-DD')),
    PARTITION s2019q4 VALUES LESS THAN (TO_DATE('2020-01-01','YYYY-MM-DD')),
    PARTITION s2020   VALUES LESS THAN (MAXVALUE) ) ;

CREATE MATERIALIZED VIEW sales_in_1999_mv
  ENABLE QUERY REWRITE AS
  SELECT s.time_id, p.prod_name, SUM(quantity_sold) FROM part_sales_by_time s, products p
  WHERE p.prod_id = s.prod_id AND s.time_id BETWEEN TO_DATE('1999-01-01', 'YYYY-MM-DD') AND TO_DATE('1999-12-31', 'YYYY-MM-DD')
  GROUP BY s.time_id, p.prod_name;

SQL> select MVIEW_NAME,DETAILOBJ_NAME,DETAIL_PARTITION_NAME PART_NAME, DETAIL_PARTITION_POSITION PART_NUM,FRESHNESS
  2    from DBA_MVIEW_DETAIL_PARTITION where mview_name='SALES_IN_1999_MV';

MVIEW_NAME           DETAILOBJ_NAME  PART_NAM PART_NUM FRESHNE
-------------------- --------------- -------- -------- -------
SALES_IN_2019_MV     SALES_BY_TIME   S1998           1 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S1999Q1         2 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S1999Q2         3 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S1999Q3         4 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S1999Q4         5 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S2000           6 FRESH
```

マテリアライズド・ビューでのANSI結合のサポート
補足：Oracle結合構文のパーティション・チェンジ・トラッキング・リライト

更新されていないパーティションはリライトが可能

- QUERY_REWRITE_INTEGRITY=STALE_TOLERATEDではPCTリライトは使用されない (データが最新かどうかが考慮されないため、最新でなくてもリライトする)

```
SQL> INSERT INTO sales_by_time
  2   VALUES (TO_DATE('1999-12-26','YYYY-MM-DD'),38920,2500,20);
SQL> commit;

SQL> select MVIEW_NAME, DETAILOBJ_NAME, ...
  2    from DBA_MVIEW_DETAIL_PARTITION
  3   where mview_name='SALES_IN_1999_MV';

MVIEW_NAME           DETAILOBJ_NAME  PART_NAM PART_NUM FRESHNE
-------------------- --------------- -------- -------- -------
...
SALES_IN_2019_MV     SALES_BY_TIME   S1999Q3         4 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S1999Q4         5 STALE
SALES_IN_2019_MV     SALES_BY_TIME   S2000           6 FRESH

SQL> SELECT s.time_id, p.prod_name, SUM(quantity_sold)
  2    FROM sales_by_time s JOIN
  3         products p ON (p.prod_id = s.prod_id)
  4   WHERE s.time_id BETWEEN TO_DATE('1999-07-01','YYYY-MM-DD')
  5                       AND TO_DATE('1999-07-31','YYYY-MM-DD')
  6   GROUP BY s.time_id, p.prod_name;
```

```
Execution Plan
----------------------------------------------------------
| Id  | Operation                     | Name             |
----------------------------------------------------------
|   0 | SELECT STATEMENT              |                  |
|*  1 |  MAT_VIEW REWRITE ACCESS FULL| SALES_IN_2019_MV |
----------------------------------------------------------

SQL> SELECT s.time_id, p.prod_name, SUM(quantity_sold)
  2    FROM sales_by_time s JOIN
  3         products p ON (p.prod_id = s.prod_id)
  4   WHERE s.time_id BETWEEN TO_DATE('1999-10-01','YYYY-MM-DD')
  5                       AND TO_DATE('1999-10-31','YYYY-MM-DD')
  6   GROUP BY s.time_id, p.prod_name;

Execution Plan
-----------------------------------------------------------------------
| Id  | Operation                | Name          |Pstart|Pstop |
-----------------------------------------------------------------------
|   0 | SELECT STATEMENT         |               |      |      |
|   1 |  HASH GROUP BY           |               |      |      |
|*  2 |   HASH JOIN              |               |      |      |
|   3 |    PARTITION RANGE SINGLE|               |    5 |    5 |
|*  4 |     TABLE ACCESS FULL    | SALES_BY_TIME |    5 |    5 |
|   5 |    TABLE ACCESS FULL     | PRODUCTS      |      |      |
-----------------------------------------------------------------------
```

論理パーティション・チェンジ・トラッキング

論理パーティション・チェンジ・トラッキング

MViewのリフレッシュおよび失効追跡のための論理パーティション・チェンジ・トラッキング

機能概要

- 論理パーティション・チェンジ・トラッキング (LPCT) を作成することで、新しいディクショナリ表にある定義済の論理パーティション内の変更点が追跡され、手軽に高速リフレッシュや失効追跡が可能になる
  - MViewの失効が論理パーティションの粒度で追跡される（物理パーティション化キーと同じキーは指定できない）
    - アプリケーションに対する問合せリライトの適用性を大幅に拡大する（LPCTリライト）
    - 失効した論理パーティションのみを対象としたリフレッシュが可能なため、リフレッシュ時間が短縮される（LPCTリフレッシュ）
  - 論理パーティション・チェンジ・トラッキングを生成するSQL文が追加された
    - CREATE/DROP LOGICAL PARTITION TRACKING 文
    - 単一の列、RANGEまたはINTERVAL論理パーティションに限定
- 変更された行が比較的大きい場合にログベースのリフレッシュより優れていてる
  - 最新データのためにMViewログ全体をスキャンしベース表と結合する必要がない

メリット

- ベース表のパーティション化が強制されずに高速リフレッシュや失効追跡が可能になり、MViewの管理がしやすくなった
  - パーティション・プルーニングはできないが、MViewの高速リフレッシュや失効追跡のみを可能に
  - 物理パーティションと組み合わせるとよりきめ細かく失効範囲を特定できる
- PCTとは異なり表作成に関係なく、いつでもベース表で作成、変更または削除できる

論理パーティション・チェンジ・トラッキング
利用イメージ

- CREATE LOGICAL PARTITION TRACKING文を使用して論理パーティションを作成する
- リフレッシュ状態はDBA_MVIEW_DETAIL_LOGICAL_PARTITIONビューのFRESHNESS列で確認する
- リフレッシュはdbms_mview.refresh(method => 'L')で行う
  - 高速リフレッシュ(method => 'F')や強制リフレッシュ(method => '?')でも動作する

```
SQL> CREATE LOGICAL PARTITION TRACKING ON promotions
  2   partition by range (promo_id) interval(100)
  3   (partition p1 values less than(100)) ;

SQL> select MVIEW_NAME, DETAILOBJ_NAME, DETAIL_LOGICAL_PARTITION_NAME LPARTNAME,
  2         DETAIL_LOGICAL_PARTITION_NUMBER LPART#, FRESHNESS
  3  from DBA_MVIEW_DETAIL_LOGICAL_PARTITION where DETAILOBJ_NAME='PROMOTIONS';

MVIEW_NAME           DETAILOBJ_NAME  LPARTNAME  LPART#   FRESHNE
-------------------- --------------- ---------- -------- -------
MV                   PROMOTIONS      P1                0 FRESH
…

SQL> exec dbms_mview.refresh ('mv', method=> 'L');

SQL> select MVIEW_NAME,LAST_REFRESH_TYPE from DBA_MVIEWS where mview_name='MV';

MVIEW_NAME           LAST_REF
-------------------- --------
MV                   FAST_LPT
```

```
SQL> exec dbms_mview.refresh ('mv', method=> 'F');

SQL> select ... from DBA_MVIEWS where mview_name='MV';

MVIEW_NAME           LAST_REF
-------------------- --------
MV                   FAST_LPT

SQL> exec dbms_mview.refresh ('mv', method=> '?');

SQL> select ... from DBA_MVIEWS where mview_name='MV';

MVIEW_NAME           LAST_REF
-------------------- --------
MV                   FAST_LPT
```

論理パーティション・チェンジ・トラッキング
利用イメージ（論理パーティション・チェンジ・トラッキングの作成）

```
SQL> CREATE MATERIALIZED VIEW  sales_in_2019_mv
  2    ENABLE QUERY REWRITE AS
  3    SELECT s.time_id, p.prod_name, SUM(quantity_sold) FROM sales_by_time s, products p
  4    WHERE p.prod_id = s.prod_id AND s.time_id BETWEEN TO_DATE('2019-01-01','YYYY-MM-DD') AND TO_DATE('2019-12-31','YYYY-MM-DD')
  5    GROUP BY s.time_id, p.prod_name;

SQL> CREATE LOGICAL PARTITION TRACKING ON sales_by_time
  2    PARTITION BY RANGE (time_id)
  3    ( PARTITION s2018 VALUES LESS THAN (TO_DATE('2019-01-01', 'YYYY-MM-DD')),
  4      PARTITION s2019q1 VALUES LESS THAN (TO_DATE('2019-04-01', 'YYYY-MM-DD')),
  5      PARTITION s2019q2 VALUES LESS THAN (TO_DATE('2019-07-01', 'YYYY-MM-DD')),
  6      PARTITION s2019q3 VALUES LESS THAN (TO_DATE('2019-10-01', 'YYYY-MM-DD')),
  7      PARTITION s2019q4 VALUES LESS THAN (TO_DATE('2020-01-01', 'YYYY-MM-DD')),
  8      PARTITION s2020 VALUES LESS THAN (MAXVALUE) ) ;

SQL> select MVIEW_NAME, DETAILOBJ_NAME, DETAIL_LOGICAL_PARTITION_NAME LPARTNAME, DETAIL_LOGICAL_PARTITION_NUMBER LPART#, FRESHNESS
  2    from DBA_MVIEW_DETAIL_LOGICAL_PARTITION
  3   where MVIEW_NAME='SALES_IN_2019_MV';

MVIEW_NAME           DETAILOBJ_NAME  LPARTNAME    LPART# FRESHNE
-------------------- --------------- ---------- -------- -------
SALES_IN_2019_MV     SALES_BY_TIME   S2018             0 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S2019Q1            1 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S2019Q2            2 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S2019Q3            3 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S2019Q4            4 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S2020              5 FRESH
```

論理パーティション・チェンジ・トラッキング
利用イメージ（ベース表の更新）

- 更新されていない論理パーティションはリライトが可能 (PCTリライトと同じように)
  - QUERY_REWRITE_INTEGRITY=STALE_TOLERATEDではLPCTリライトは使用されない

```
SQL> INSERT INTO sales_by_time
  2   VALUES (TO_DATE('2019-12-26','YYYY-MM-DD'),38920,2500, 20);
SQL> commit;

SQL> select MVIEW_NAME, DETAILOBJ_NAME, ...
  2    from DBA_MVIEW_DETAIL_LOGICAL_PARTITION
  3   where MVIEW_NAME='SALES_IN_2019_MV';

MVIEW_NAME           DETAILOBJ_NAME  LPARTNAME    LPART# FRESHNE
-------------------- --------------- ---------- -------- -------
...
SALES_IN_2019_MV     SALES_BY_TIME   S2019Q3           3 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S2019Q4           4 STALE
SALES_IN_2019_MV     SALES_BY_TIME   S2020             5 FRESH

SQL> SELECT s.time_id, p.prod_name, SUM(s.quantity_sold)
  2    FROM sales_by_time s, products p
  3   WHERE p.prod_id = s.prod_id
  4     AND s.time_id BETWEEN TO_DATE('2019-07-01','YYYY-MM-DD')
  5                       AND TO_DATE('2019-07-31','YYYY-MM-DD')
  6   GROUP BY s.time_id, p.prod_name;
```

```
Execution Plan
----------------------------------------------------------
| Id  | Operation                     | Name             |
----------------------------------------------------------
|   0 | SELECT STATEMENT              |                  |
|*  1 |  MAT_VIEW REWRITE ACCESS FULL| SALES_IN_2019_MV |
----------------------------------------------------------

SQL> SELECT s.time_id, p.prod_name, SUM(s.quantity_sold)
  2    FROM sales_by_time s, products p
  3   WHERE p.prod_id = s.prod_id
  4     AND s.time_id BETWEEN TO_DATE('2019-12-01','YYYY-MM-DD')
  5                       AND TO_DATE('2019-12-31','YYYY-MM-DD')
  6   GROUP BY s.time_id, p.prod_name;

Execution Plan
---------------------------------------------------- -----
| Id  | Operation                      | Name          |
----------------------------------------------------------
|   0 | SELECT STATEMENT               |               |
|   1 |  HASH GROUP BY                 |               |
|   2 |   NESTED LOOPS                 |               |
|   3 |    NESTED LOOPS                |               |
|*  4 |     TABLE ACCESS FULL          | SALES_BY_TIME |
|*  5 |     INDEX UNIQUE SCAN          | PROD_PK       |
|   6 |    TABLE ACCESS BY INDEX ROWID| PRODUCTS      |
----------------------------------------------------------
```

論理パーティション・チェンジ・トラッキング
利用イメージ（リフレッシュ）

- 更新された論理パーティションだけリフレッシュされる

```
SQL> exec dbms_mview.refresh ('mv4', method=> 'L');

SQL> select mview_name, DETAILOBJ_NAME, ...
  2    from DBA_MVIEW_DETAIL_LOGICAL_PARTITION
  3   where mview_name='SALES_IN_2019_MV';

MVIEW_NAME           DETAILOBJ_NAME  LPARTNAME    LPART# FRESHNE
-------------------- --------------- ---------- -------- -------
SALES_IN_2019_MV     SALES_BY_TIME   S2018             0 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S2019Q1           1 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S2019Q2           2 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S2019Q3           3 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S2019Q4           4 FRESH
SALES_IN_2019_MV     SALES_BY_TIME   S2020             5 FRESH

SQL> select MVIEW_NAME, LAST_REFRESH_TYPE
  2    from DBA_MVIEWS where mview_name='SALES_IN_2019_MV';

MVIEW_NAME           LAST_REF
-------------------- --------
SALES_IN_2019_MV     FAST_LPT
```

```
SQL> SELECT s.time_id, p.prod_name, SUM(s.quantity_sold)
  2    FROM sales_by_time s, products p
  3   WHERE p.prod_id = s.prod_id
  4     AND s.time_id BETWEEN TO_DATE('2019-12-01','YYYY-MM-DD')
  5                       AND TO_DATE('2019-12-31','YYYY-MM-DD')
  6   GROUP BY s.time_id, p.prod_name;

Execution Plan
----------------------------------------------------------
| Id  | Operation                     | Name             |
----------------------------------------------------------
|   0 | SELECT STATEMENT              |                  |
|*  1 |  MAT_VIEW REWRITE ACCESS FULL| SALES_IN_2019_MV |
----------------------------------------------------------
```

論理パーティション・チェンジ・トラッキング
リフレッシュ時間（完全リフレッシュとLPCTリフレッシュの比較）

```
CREATE TABLE test(a number primary key, b number);
begin
  for i in 1..100000 loop
    insert into test values(i,i);
  end loop;
  commit;
end;
/
CREATE LOGICAL PARTITION TRACKING ON test
  PARTITION BY RANGE(b)
  (PARTITION p1 VALUES LESS THAN (250),
   PARTITION p2 VALUES LESS THAN (500),
   PARTITION p3 VALUES LESS THAN (750),
   PARTITION p4 VALUES LESS THAN (MAXVALUE));
CREATE MATERIALIZED VIEW mv ENABLE QUERY REWRITE AS
  select * from test;

SQL> select mview_name, DETAILOBJ_NAME, ...
  2    from DBA_MVIEW_DETAIL_LOGICAL_PARTITION
  3   where mview_name='MV';

MVIEW_NAME           DETAILOBJ_NAME  LPARTNAME    LPART# FRESHNE
-------------------- --------------- ---------- -------- -------
MV                   TEST            P1                0 FRESH
MV                   TEST            P2                1 FRESH
MV                   TEST            P3                2 FRESH
MV                   TEST            P4                3 FRESH
```

```
SQL> INSERT INTO test VALUES (100001, 100001);
SQL> commit;

SQL> select mview_name, DETAILOBJ_NAME, ...
  2    from DBA_MVIEW_DETAIL_LOGICAL_PARTITION
  3   where mview_name='MV';

MVIEW_NAME           DETAILOBJ_NAME  LPARTNAME    LPART# FRESHNE
-------------------- --------------- ---------- -------- -------
MV                   TEST            P1                0 FRESH
MV                   TEST            P2                1 FRESH
MV                   TEST            P3                2 FRESH
MV                   TEST            P4                3 STALE

SQL> set timing on
SQL> exec dbms_mview.refresh('MV','L');

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.07

SQL> exec dbms_mview.refresh('MV','C');

PL/SQL procedure successfully completed.

Elapsed: 00:00:04.14
```

論理パーティション・チェンジ・トラッキング
PCTとの組み合わせとログベースとの組み合わせのクエリ・リライト確認

PCTと組み合わせテスト

- 更新
  - 物理パーティション(P2)と論理パーティション(LP1)の更新

```
update test set c=c+1 where a=10000 and b=0;
```

- 問合せ

```
select sum(c) from test where a=10000 and b=1;
select sum(c) from test where a=10001 and b=0;
select sum(c) from test where a=20000 and b=0;
select sum(c) from test where a=10000 and b=500;
```

ログベースと組み合わせテスト

- 更新
  - 論理パーティション(LP3)の更新

```
update test set c=c+1 where b=500;
```

- 問合せ

```
select sum(c) from test where b=500;
select sum(c) from test where b=250;
```

```
create table test(a number primary key, b number, c number)
 partition by range(a)
 (partition p1 values less than(10000),
  partition p2 values less than(20000),
  partition p3 values less than(30000),
  partition p4 values less than(MAXVALUE));

create logical partition tracking on test
 partition by range(b)
 (partition lp1 values less than(250),
  partition lp2 values less than(500),
  partition lp3 values less than(750),
  partition lp4 values less than(MAXVALUE));

create materialized view mv
 refresh fast
 enable query rewrite as
 select * from test;
```

```
CREATE TABLE test(a number primary key, b number, c number);

CREATE MATERIALIZED VIEW LOG ON test WITH PRIMARY KEY;

create logical partition tracking on test
 partition by range(b)
 (partition lp1 values less than(250), ... );

create materialized view mv ... ;
```

論理パーティション・チェンジ・トラッキング

PCTとの組み合わせでのクエリ・リライト確認 (LPCTとPCTのどちらでもリライトする)

```
SQL> update test set c=c+1 where a=10000 and b=0;
SQL> commit;

MVIEW_NAME           DETAILOBJ_NAME  PART_NAM PART_NUM FRESHNE
-------------------- --------------- -------- -------- -------
MV                   TEST            P1              1 FRESH
MV                   TEST            P2              2 STALE
MV                   TEST            P3              3 FRESH
MV                   TEST            P4              4 FRESH

MVIEW_NAME           DETAILOBJ_NAME  LPARTNAME    LPART# FRESHNE
-------------------- --------------- ---------- -------- -------
MV                   TEST            LP1             0 STALE
MV                   TEST            LP2             1 FRESH
MV                   TEST            LP3             2 FRESH
MV                   TEST            LP4             3 FRESH
```

- P2とLP1(どちらも失効)への問合せ

```
SQL> select sum(c) from test where a=10000 and b=1;
SQL> select sum(c) from test where a=10001 and b=0;
-------------------------------------------------------------------
| Id  | Operation                          | Name        |Pstart|
-------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |             |      |
|   1 |  SORT AGGREGATE                    |             |      |
|*  2 |   TABLE ACCESS BY GLOBAL INDEX ROWID| TEST        |    2 |
|*  3 |    INDEX UNIQUE SCAN               | SYS_C008418 |      |
-------------------------------------------------------------------
```

```
SQL> select sum(c) from test where a=20000 and b=0;
-------------------------------------------------------------------
| Id  | Operation                                | Name        |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |             |
|   1 |  SORT AGGREGATE                          |             |
|*  2 |   MAT_VIEW REWRITE ACCESS BY INDEX ROWID| MV          |
|*  3 |    INDEX UNIQUE SCAN                     | SYS_C008419 |
-------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("MV"."B"=0)
   3 - access("MV"."A"=20000)

SQL> select sum(c) from test where a=10000 and b=500;
-------------------------------------------------------------------
| Id  | Operation                                | Name        |
-------------------------------------------------------------------
|   0 | SELECT STATEMENT                         |             |
|   1 |  SORT AGGREGATE                          |             |
|*  2 |   MAT_VIEW REWRITE ACCESS BY INDEX ROWID| MV          |
|*  3 |    INDEX UNIQUE SCAN                     | SYS_C008419 |
-------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("MV"."B"=500)
   3 - access("MV"."A"=10000)
```

論理パーティション・チェンジ・トラッキング

ログベースとの組み合わせでのクエリ・リライト確認 (LPCTリライトが動作する)

```
SQL> update test set c=c+1 where b=500;
SQL> commit;

MVIEW_NAME           DETAILOBJ_NAME  LPARTNAME    LPART# FRESHNE
-------------------- --------------- ---------- -------- -------
MV                   TEST            LP1               0 FRESH
MV                   TEST            LP2               1 FRESH
MV                   TEST            LP3               2 STALE
MV                   TEST            LP4               3 FRESH
```

- LP3への問合せ

```
SQL> select sum(c) from test where b=500;
-----------------------------------
| Id  | Operation          | Name |
-----------------------------------
|   0 | SELECT STATEMENT   |      |
|   1 |  SORT AGGREGATE    |      |
|*  2 |   TABLE ACCESS FULL| TEST |
-----------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
 2 - filter("B"=500)
```

LP3以外への問合せ

```
SQL> select sum(c) from test where b=250;
-----------------------------------------------------
| Id  | Operation                     | Name        |
-----------------------------------------------------
|   0 | SELECT STATEMENT              |             |
|   1 |  SORT AGGREGATE               |             |
|*  2 |   MAT_VIEW REWRITE ACCESS FULL| MV          |
-----------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
   2 - filter("MV"."B"=250)
```

論理パーティション・チェンジ・トラッキング
PCTとの組み合わせとログベースとの組み合わせのリフレッシュ確認

PCTと組み合わせテスト

- 常にPCTリフレッシュになる

```
SQL> exec dbms_mview.refresh('MV', 'L');

SQL> select MVIEW_NAME, LAST_REFRESH_TYPE
  2    from DBA_MVIEWS where mview_name='MV';

MVIEW_NAME           LAST_REF
-------------------- --------
MV                   FAST_PCT

SQL> exec dbms_mview.refresh('MV', 'F');

SQL> select MVIEW_NAME,LAST_REFRESH_TYPE from DBA_MVIEWS ... ;

MVIEW_NAME           LAST_REF
-------------------- --------
MV                   FAST_PCT
```

ログベースと組み合わせテスト

- 効果的に行うには手動で使い分ける必要がある
  - (method => 'F')ではログベースのリフレッシュが動作
  - (method => 'L')ではLPCTリフレッシュが動作

```
SQL> exec dbms_mview.refresh('MV', 'F');

SQL> select MVIEW_NAME, LAST_REFRESH_TYPE from DBA_MVIEWS
  2   where mview_name='MV';

MVIEW_NAME           LAST_REF
-------------------- --------
MV                   FAST

SQL> exec dbms_mview.refresh('MV', 'L');

SQL> select MVIEW_NAME,LAST_REFRESH_TYPE from DBA_MVIEWS ... ;

MVIEW_NAME           LAST_REF
-------------------- --------
MV                   FAST_LPT
```

- リフレッシュはログベースで行い、クエリ・リライトはLPCTリライトが可能になる

マテリアライズド・ビューの同時リフレッシュ

マテリアライズド・ビューの同時リフレッシュ

オンコミットMViewの同時リフレッシュが追加された

機能概要

- 同時リフレッシュにより、複数セッションが同じオンコミットMViewを同時にコミットすると、シリアライズされずに同時リフレッシュすることが可能になった
  - ベース表に対してDMLを実行するトランザクションを同時実行したときのリフレッシュが高速になった
  - 同時実行セッション数の制限はない
- 同時リフレッシュを実行できるかどうかは、以下の条件で判断される
  - 同時リフレッシュが有効である (ON COMMITリフレッシュMViewのみ有効にできる)
  - すべての同時DMLセッションは同じベース表を更新する
  - 異なるリフレッシュ・セッションで更新されたMViewの行が重なっていない

メリット

- OLTPでコミット時にMViewのリフレッシュを行う場合や、多くの同時実行DMLトランザクションがファクト表のみを更新する場合などに有効
- この機能とLPCTにより、アプリケーションにおけるMViewの適用範囲が広がり、アプリケーション開発の簡素化に貢献する

マテリアライズド・ビューの同時リフレッシュ
利用イメージ

- MViewに{ ENABLE | DISABLE } CONCURRENT REFRESHを指定する（デフォルトはDISABLE）
  - ON COMMITと一緒に指定する必要がある
- 同時リフレッシュが有効かどうかはDBA_MVIEWSビューのCONCURRENT_REFRESH_ENABLEDで確認する

```
SQL> CREATE TABLE test (a NUMBER PRIMARY KEY, b NUMBER);
SQL> CREATE MATERIALIZED VIEW LOG ON test WITH PRIMARY KEY;
SQL> CREATE MATERIALIZED VIEW mv1
  2    REFRESH FAST ON COMMIT
  3    ENABLE CONCURRENT REFRESH
  4    ENABLE QUERY REWRITE AS
  5    SELECT * FROM test ;

SQL> select MVIEW_NAME, CONCURRENT_REFRESH_ENABLED
  2   from DBA_MVIEWS where MVIEW_NAME = 'MV1';

MVIEW_NAME      C
--------------- -
MV1             Y

SQL> CREATE MATERIALIZED VIEW mv1
  2    REFRESH FAST
  3    ENABLE CONCURRENT REFRESH
  4    ENABLE QUERY REWRITE AS
  5    SELECT * FROM test ;

ORA-32383: unsupported materialized view for concurrent refresh
```

```
SQL> CREATE TABLE test2 (a NUMBER PRIMARY KEY, b NUMBER);
SQL> CREATE MATERIALIZED VIEW LOG ON test2 WITH PRIMARY KEY;
SQL> CREATE MATERIALIZED VIEW mv2
  2    REFRESH FAST ON COMMIT
  3    ENABLE QUERY REWRITE AS
  4    SELECT * FROM test2 ;

SQL> select MVIEW_NAME, CONCURRENT_REFRESH_ENABLED
  2   from DBA_MVIEWS where MVIEW_NAME = 'MV2';

MVIEW_NAME      C
--------------- -
MV2             N
```

マテリアライズド・ビューの同時リフレッシュ
同時リフレッシュ時間

- 同時リフレッシュ有効の場合（二つのセッションで同時コミット）

```
SQL> begin
      for i in 1..10000 loop
       insert into test values(i,i);
      end loop;
     end;
     /
SQL> commit;

Commit complete.

Elapsed: 00:00:02.20
```

```
SQL> begin
      for i in 10001..20000 loop
       insert into test values(i,i);
      end loop;
     end;
     /
SQL> commit;

Commit complete.

Elapsed: 00:00:01.37
```

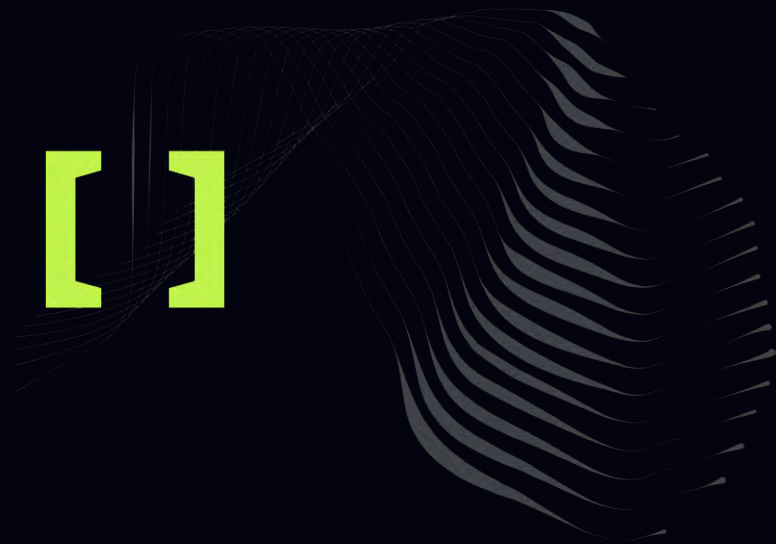
- 同時リフレッシュ無効の場合（二つのセッションで同時コミット）

```
SQL> begin
      for i in 1..10000 loop
       insert into test2 values(i,i);
      end loop;
     end;
     /
SQL> commit;

Commit complete.

Elapsed: 00:00:01.98
```

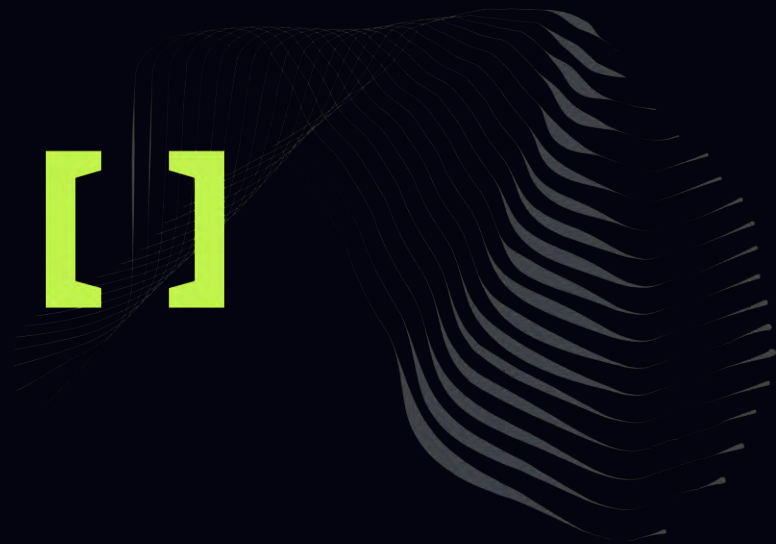
```
SQL> begin
      for i in 10001..20000 loop
       insert into test2 values(i,i);
      end loop;
     end;
     /
SQL> commit;

Commit complete.

Elapsed: 00:00:20.04
```

Partitioning 23c 新機能

時間隔および自動リスト・パーティション化によるハイブリッド・パーティション表
拡張されたパーティション化のメタデータ



時間隔および自動リスト・パーティション化によるハイブリッド・パーティション表



時間隔および自動リスト・パーティション化によるハイブリッド・パーティション表

インターバル・パーティションおよび自動リスト・パーティションをサポート

機能概要

- ハイブリッド・パーティションとして、インターバル・パーティションおよび自動リスト・パーティションをサポート
  - レンジやリストと同様に少なくとも 1 つの内部パーティションが必要
  - アクセスドライバーとしては、ORACLE_LOADER、ORACLE_DATAPUMP、ORACLE_HDFS、ORACLE_BIGDATAが選択可能、ORACLE_HIVEは未対応
  - Oracle Database In-Memoryでも利用可能

メリット

- ハイブリッド・パーティション表に使いやすいパーティション化方法を提供する

インターバル・パーティションの例(月単位の分割)

```
CREATE TABLE INTERVAL_TEST(
…
 PARTITION BY RANGE (AAA) INTERVAL(NUMTOYMINTERVAL(1, 'MONTH'))
(
 PARTITION P0 VALUES LESS THAN (TO_DATE('01-01-2018', 'DD-MM-YYYY')),
 PARTITION P1 VALUES LESS THAN (TO_DATE('01-01-2019', 'DD-MM-YYYY'))
 EXTERNAL LOCATION('interval_hypt.csv'),
 PARTITION P2 VALUES LESS THAN (TO_DATE('01-07-2020', 'DD-MM-YYYY'))
);
```

自動リスト・パーティションの例

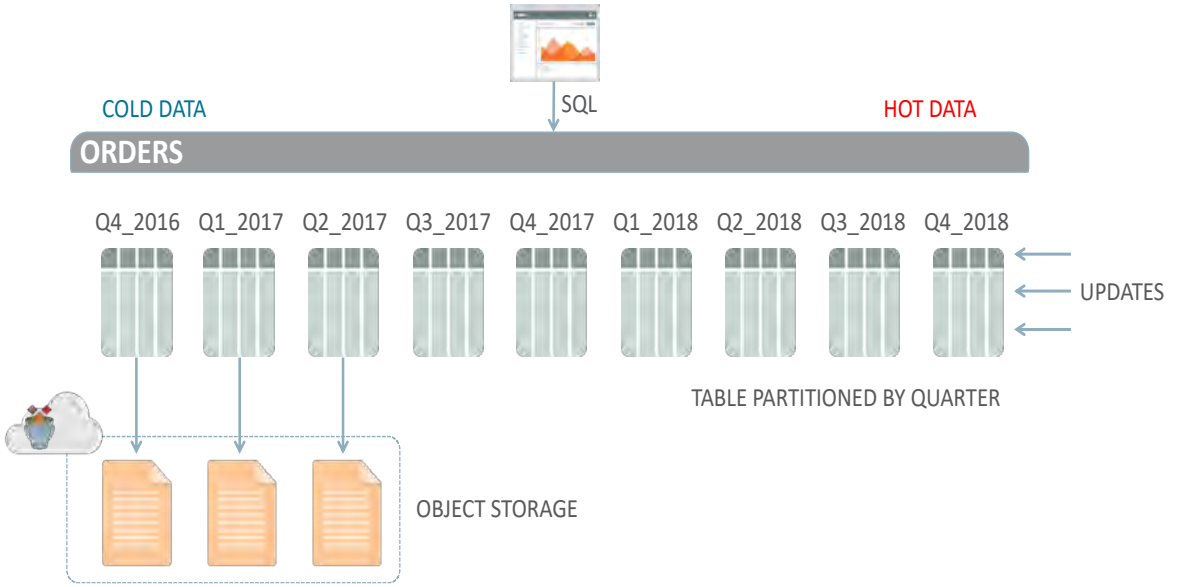
```
CREATE TABLE AUTO_TEST(
…
 PARTITION BY LIST(STATE) AUTOMATIC
(
 PARTITION P_TOKYO VALUES ('TOKYO'),
 PARTITION P_OSAKA VALUES ('OSAKA')
 EXTERNAL LOCATION('p_osaa.csv')
);
```

時間隔および自動リスト・パーティション化によるハイブリッド・パーティション表

ハイブリッド・パーティション表(19c～)

```
CREATE TABLE hybrid_partition_orders (
    prod_id       NUMBER         NOT NULL,
    cust_id       NUMBER         NOT NULL,
    time_id       DATE           NOT NULL,
    channel_id    NUMBER         NOT NULL,
    promo_id      NUMBER         NOT NULL,
    quantity_sold NUMBER(10,2)   NOT NULL,
    amount_sold   NUMBER(10,2)   NOT NULL
)
EXTERNAL PARTITION ATTRIBUTES (
  TYPE ORACLE_LOADER
  DEFAULT DIRECTORY sales_data …
)
PARTITION BY RANGE (time_id) (
   PARTITION sales_2022
     VALUES LESS THAN ('01-01-2023'),
   PARTITION sales_2021
     VALUES LESS THAN ('01-01-2022')
       EXTERNAL LOCATION ('sales2021_data.txt'),
   PARTITION sales_2020
     VALUES LESS THAN ('01-01-2021')
       EXTERNAL LOCATION ('sales2020_data.txt')
```

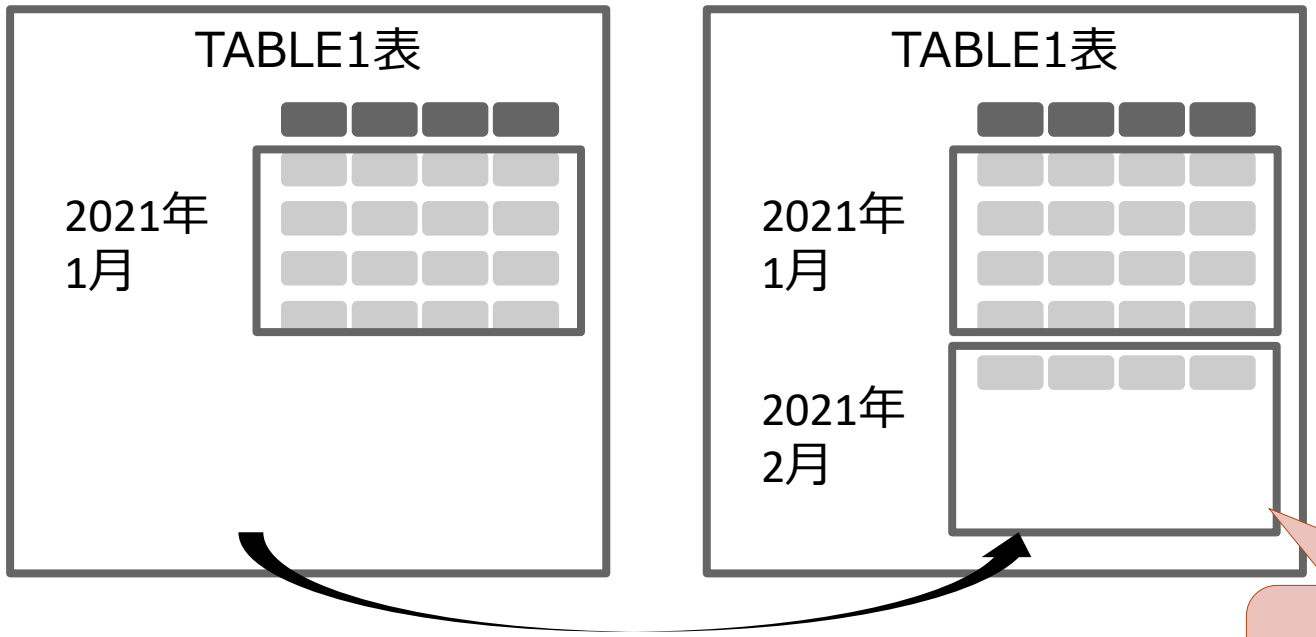
- 外部表を含めたパーティション表の作成
- 古くなったパーティションをストレージに外だしするが、DBからも参照できるようにする
- 単一のレンジおよびリスト・パーティションのみ対応
- 外部パーティションについてはREAD ONLYになる
- 外部パーティションに対してSPLIT、MERGE、MOVEメンテナンス操作はできない



時間隔および自動リスト・パーティション化によるハイブリッド・パーティション表

インターバル・パーティション(11gR1～)

レンジ・パーティションにて、既存のパーティションに存在しない内容のレコードの生成があった場合、指定ルールに従って新規パーティションを自動作成する機能




```
CREATE TABLE table1 (
  id            NUMBER,
  created_date DATE
)
PARTITION BY RANGE (created_date)
INTERVAL (NUMTOYMINTERVAL(1,'MONTH'))
(
   PARTITION part_01 values LESS THAN
(TO_DATE(20210101, 'YYYYMMDD'))
);
```

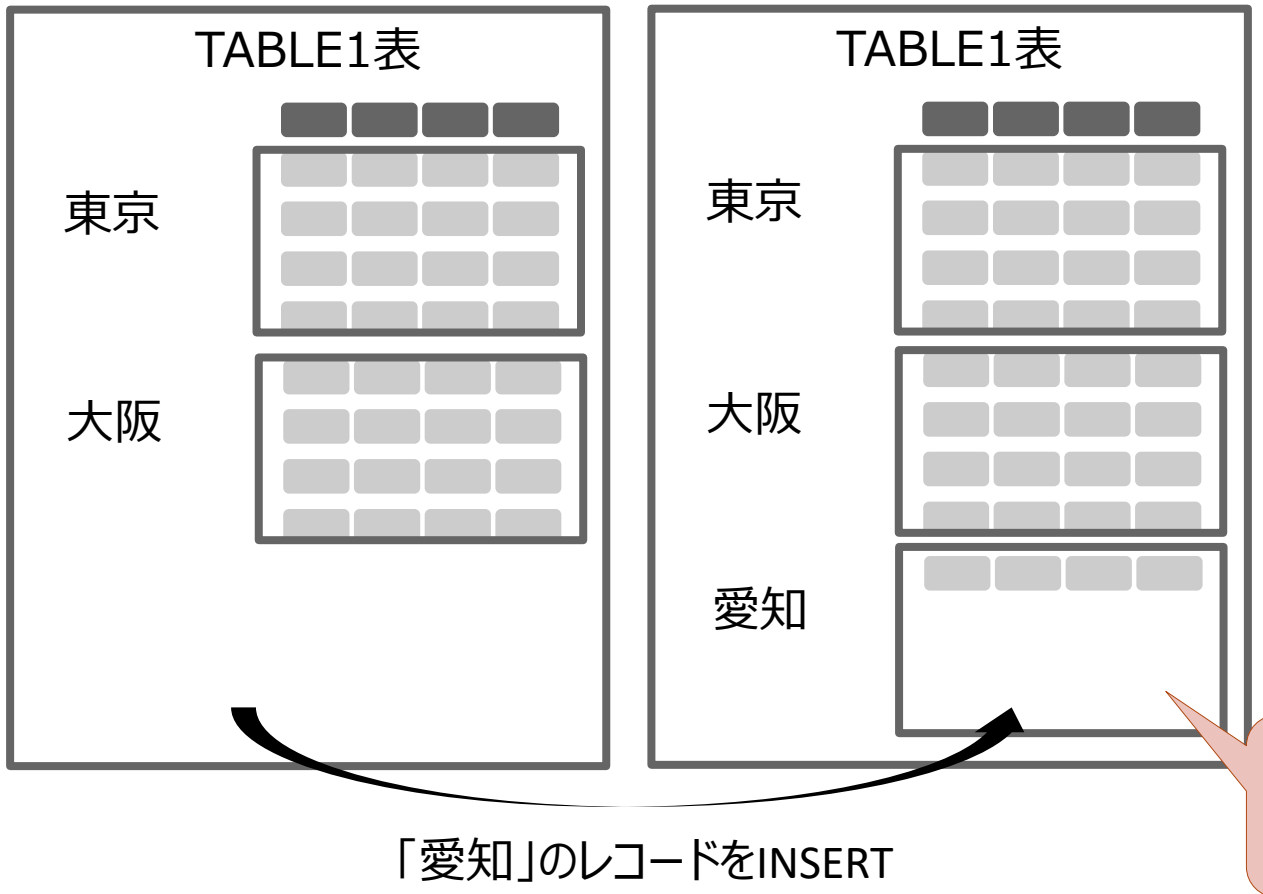
追加されるパーティションの名称はシステム生成

最低一つはパーティションを作成する必要あり

時間隔および自動リスト・パーティション化によるハイブリッド・パーティション表

自動リスト・パーティション(12cR2～)

リストに存在しない値が挿入された場合、そのリスト値を格納する新規パーティションを作成

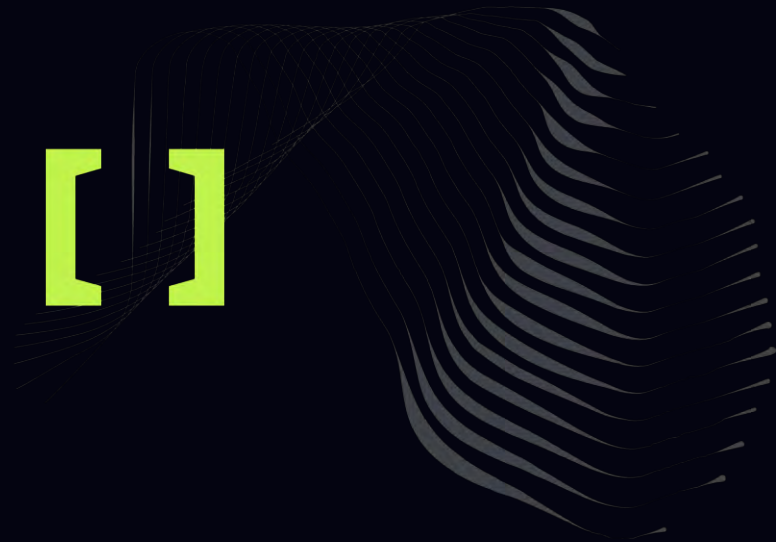



```
CREATE TABLE table1 (
  id              NUMBER,
  prefecture VARCHAR2(9)
)
PARTITION BY LIST (prefecture)
AUTOMATIC
(
   PARTITION TOKYO VALUES('東京'),
   PARTITION OSAKA VALUES('大阪')
);
```

最低一つはパーティションを作成する必要あり

追加されるパーティションの名称はシステム生成

拡張されたパーティション化のメタデータ



拡張されたパーティション化のメタデータ

パーティション化メタデータにパーティション境界情報が追加された

機能概要

- パーティション関連のメタデータ (データ・ディクショナリ・ビュー) にパーティションの境界情報(High Value)の列(JSON/CLOB形式)が追加された (これまではLONG型のHIGH_VALUE列)
  - HIGH_VALUE_CLOB
    - パーティション/サブパーティションの境界情報をCLOB形式で格納 (LONG型ではサポートされない、SUBSTR関数などが使用できる)
  - HIGH_VALUE_JSON
    - パーティション/サブパーティションの境界情報をJSON形式で格納、複雑なHigh Value式の定義 (複数の値を持つリスト・パーティションなど) に役立つ
- 利用可能なデータ・ディクショナリ・ビュー
  - ALL/DBA/USER/CDB_TAB_PARTITIONS, xxx_TAB_SUBPARTITIONS, xxx_IND_PARTITIONS, xxx_IND_SUBPARTITIONS

メリット

- スキーマ検索やライフサイクル管理操作のために、プログラム (SQL) でより簡単に使用できるようになった

```
CREATE TABLE test(a number primary key, ... )
 PARTITION BY RANGE (a)
 (PARTITION p1 VALUES LESS THAN(10000),
  PARTITION p2 VALUES LESS THAN(20000),
  PARTITION p3 VALUES LESS THAN(30000),
  PARTITION p4 VALUES LESS THAN(MAXVALUE));

SQL> select TABLE_NAME,PARTITION_NAME,HIGH_VALUE_CLOB
  2    from USER_TAB_PARTITIONS;

TABLE_NAME      PARTITION_NAME  HIGH_VALUE_CLOB
--------------- --------------- ------------------
TEST            P1              10000
TEST            P2              20000
TEST            P3              30000
TEST            P4              MAXVALUE

SQL> select TABLE_NAME,PARTITION_NAME,HIGH_VALUE_JSON
  2    from USER_TAB_PARTITIONS;

TABLE_NAME      PARTITION_NAME  HIGH_VALUE_JSON
--------------- --------------- --------------------
TEST            P1              {"high_value":10000}
TEST            P2              {"high_value":20000}
TEST            P3              {"high_value":30000}
TEST            P4              {"is_maxvalue":true}
```

拡張されたパーティション化のメタデータ

利用イメージ（DATEのレンジ・パーティション）

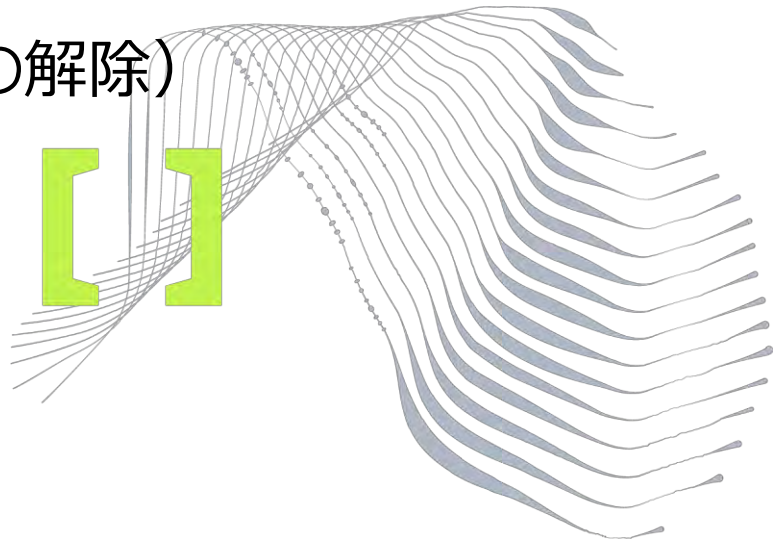
```
CREATE TABLE part_sales_by_time ( ... )
  PARTITION BY RANGE (time_id)
  ( PARTITION s1998 VALUES LESS THAN (TO_DATE('1999-01-01', 'YYYY-MM-DD')),
    PARTITION s1999q1 VALUES LESS THAN (TO_DATE('1999-04-01', 'YYYY-MM-DD')),
    PARTITION s1999q2 VALUES LESS THAN (TO_DATE('1999-07-01', 'YYYY-MM-DD')),
    PARTITION s1999q3 VALUES LESS THAN (TO_DATE('1999-10-01', 'YYYY-MM-DD')),
    PARTITION s1999q4 VALUES LESS THAN (TO_DATE('2000-01-01', 'YYYY-MM-DD')),
    PARTITION s2000 VALUES LESS THAN (MAXVALUE) );

SQL> set linesize 140 long 100 longchunksize 100
SQL> select TABLE_NAME,PARTITION_NAME,HIGH_VALUE_CLOB from USER_TAB_PARTITIONS where TABLE_NAME='PART_SALES_BY_TIME';

TABLE_NAME           PARTITION_NAME  HIGH_VALUE_CLOB
-------------------- --------------- --------------------------------------------------------------------------------
PART_SALES_BY_TIME   S1998           TO_DATE(' 1999-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
PART_SALES_BY_TIME   S1999Q1         TO_DATE(' 1999-04-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
PART_SALES_BY_TIME   S1999Q2         TO_DATE(' 1999-07-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
PART_SALES_BY_TIME   S1999Q3         TO_DATE(' 1999-10-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
PART_SALES_BY_TIME   S1999Q4         TO_DATE(' 2000-01-01 00:00:00', 'SYYYY-MM-DD HH24:MI:SS', 'NLS_CALENDAR=GREGORIAN')
PART_SALES_BY_TIME   S2000           MAXVALUE

SQL> select TABLE_NAME,PARTITION_NAME,HIGH_VALUE_JSON from USER_TAB_PARTITIONS where TABLE_NAME='PART_SALES_BY_TIME';

TABLE_NAME           PARTITION_NAME  HIGH_VALUE_JSON
-------------------- --------------- --------------------------------------------------------------------------------
PART_SALES_BY_TIME   S1998           {"high_value":"1999-01-01T00:00:00"}
PART_SALES_BY_TIME   S1999Q1         {"high_value":"1999-04-01T00:00:00"}
PART_SALES_BY_TIME   S1999Q2         {"high_value":"1999-07-01T00:00:00"}
PART_SALES_BY_TIME   S1999Q3         {"high_value":"1999-10-01T00:00:00"}
PART_SALES_BY_TIME   S1999Q4         {"high_value":"2000-01-01T00:00:00"}
PART_SALES_BY_TIME   S2000           {"is_maxvalue":true}
```

拡張されたパーティション化のメタデータ
利用イメージ（リスト・パーティション）

```
CREATE TABLE q1_sales_by_region (deptno number, ... state varchar2(2))
  PARTITION BY LIST (state)
  ( PARTITION q1_northwest VALUES ('OR', 'WA'),
    PARTITION q1_southwest VALUES ('AZ', 'UT', 'NM'),
    PARTITION q1_northeast VALUES  ('NY', 'VM', 'NJ'),
    PARTITION q1_southeast VALUES ('FL', 'GA'),
    PARTITION q1_northcentral VALUES ('SD', 'WI'),
    PARTITION q1_southcentral VALUES ('OK', 'TX'));

SQL> set linesize 140 long 100 longchunksize 100
SQL> select TABLE_NAME,PARTITION_NAME,HIGH_VALUE_CLOB from USER_TAB_PARTITIONS where TABLE_NAME='Q1_SALES_BY_REGION';

TABLE_NAME            PARTITION_NAME  HIGH_VALUE_CLOB
--------------------- --------------- --------------------------------------------------------------------------------
Q1_SALES_BY_REGION    Q1_NORTHWEST    'OR', 'WA'
Q1_SALES_BY_REGION    Q1_SOUTHWEST    'AZ', 'UT', 'NM'
Q1_SALES_BY_REGION    Q1_NORTHEAST    'NY', 'VM', 'NJ'
Q1_SALES_BY_REGION    Q1_SOUTHEAST    'FL', 'GA'
Q1_SALES_BY_REGION    Q1_NORTHCENTRAL 'SD', 'WI'
Q1_SALES_BY_REGION    Q1_SOUTHCENTRAL 'OK', 'TX'

SQL> select TABLE_NAME,PARTITION_NAME,HIGH_VALUE_JSON from USER_TAB_PARTITIONS where TABLE_NAME='Q1_SALES_BY_REGION';

TABLE_NAME           PARTITION_NAME  HIGH_VALUE_JSON
--------------------- --------------- --------------------------------------------------------------------------------
Q1_SALES_BY_REGION    Q1_NORTHCENTRAL {"high_value":["SD","WI"]}
Q1_SALES_BY_REGION    Q1_NORTHEAST    {"high_value":["NY","VM","NJ"]}
Q1_SALES_BY_REGION    Q1_NORTHWEST    {"high_value":["OR","WA"]}
Q1_SALES_BY_REGION    Q1_SOUTHCENTRAL {"high_value":["OK","TX"]}
Q1_SALES_BY_REGION    Q1_SOUTHEAST    {"high_value":["FL","GA"]}
Q1_SALES_BY_REGION    Q1_SOUTHWEST    {"high_value":["AZ","UT","NM"]}
```

まとめ

1. ロックフリー列値の予約(ロックフリー予約)
2. 自動トランザクションロールバック
3. 幅の広い表（表/ビューの最大列数の4096列への拡張）
4. 無制限のパラレルDML/ダイレクトロード（トランザクション制約の解除）
5. Oracle Database 23c におけるマテリアライズド・ビュー強化
6. Partitioning 23c 新機能



ありがとうございました