

ORACLE

# Oracle Partitioningを 最大限に活用する

実践ガイドとリファレンス

---

バージョン19c、2020年4月

## 免責条項

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

オラクルの将来の計画、予測、信念、意図、および見込みに関するこのプレゼンテーションの記述は“将来を見越した記述”であり、重大なリスクや不確実性を伴う可能性があります。弊社のビジネスに影響するこれらの要素や他のリスクに関する詳細については、直近のForm 10-KおよびForm 10-Qによる報告書の“リスク要素”セクションを含め、オラクルの米国証券取引委員会（SEC）への提出書類に記載されています。これらの書類は、SECのWebサイト、またはオラクルのWebサイト <http://www.oracle.com/investor>で参照できます。このプレゼンテーションのすべての情報は2019年9月現在のものであり、オラクルは、いずれの記述についても、新しい情報または将来のイベントを踏まえて更新する義務を負いません。

## 作業を開始する前に..

皆様のご意見をお聞かせください

- アイデアや課題はまだたくさんあります
- 提供される情報によって方向性が決まります

以下についての情報を求めています

- 興味深いユースケースと実装
- 機能強化のご要望
- 苦情

[dw-pm\\_us@oracle.com](mailto:dw-pm_us@oracle.com)宛てに情報をお寄せください



# Oracle Partitioning

[パーティション化の概要](#)

[パーティション化の概念](#)

[パーティション化の利点](#)

[パーティション化方法](#)

[パーティション化の拡張機能](#)

[パーティション化と外部データ](#)

[パーティション化と索引付け](#)

[パフォーマンス向上のためのパーティション化](#)

[パーティション化メンテナンス](#)

[パーティション化オブジェクトと非パーティション化](#)

[オブジェクトの相違点](#)

[パーティション化 - ヒント](#)

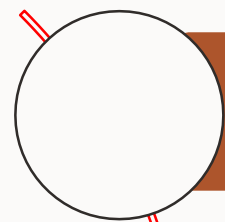
[属性クラスタリングとゾーン・マップ](#)

[ベスト・プラクティスとハウツー](#)

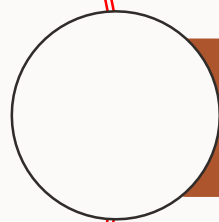
# パーティション化の概要

---

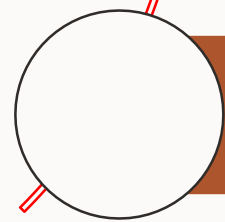
# Oracle Partitioningとは



オブジェクトをより小さな部分に論理的に分割する強力な機能



高パフォーマンスと高可用性を必要とする大規模データベースの主要な要件



ビジネス要件によって駆動

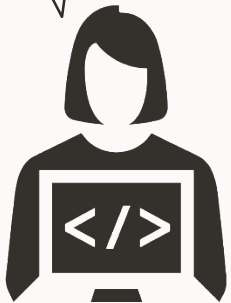
## Oracle Partitioningを使用する理由

- ↑ パフォーマンス - データ・アクセス時間を短縮
- ↑ 可用性 - 重要な情報へのアクセス性を向上
- ↓ コスト - 複数のストレージ層を利用
- ✓ 実装が容易 - アプリケーションと問合せに対する変更が不要
- ✓ 成熟した機能 - さまざまなパーティション化方式をサポート
- ✓ 十分な実績 - 何千ものオラクルのお客様が使用

# パーティション化の2種類のユーザー

## EVENTS

```
SELECT *  
FROM  
EVENTS;
```





MICRO


THERMO

1月

2月

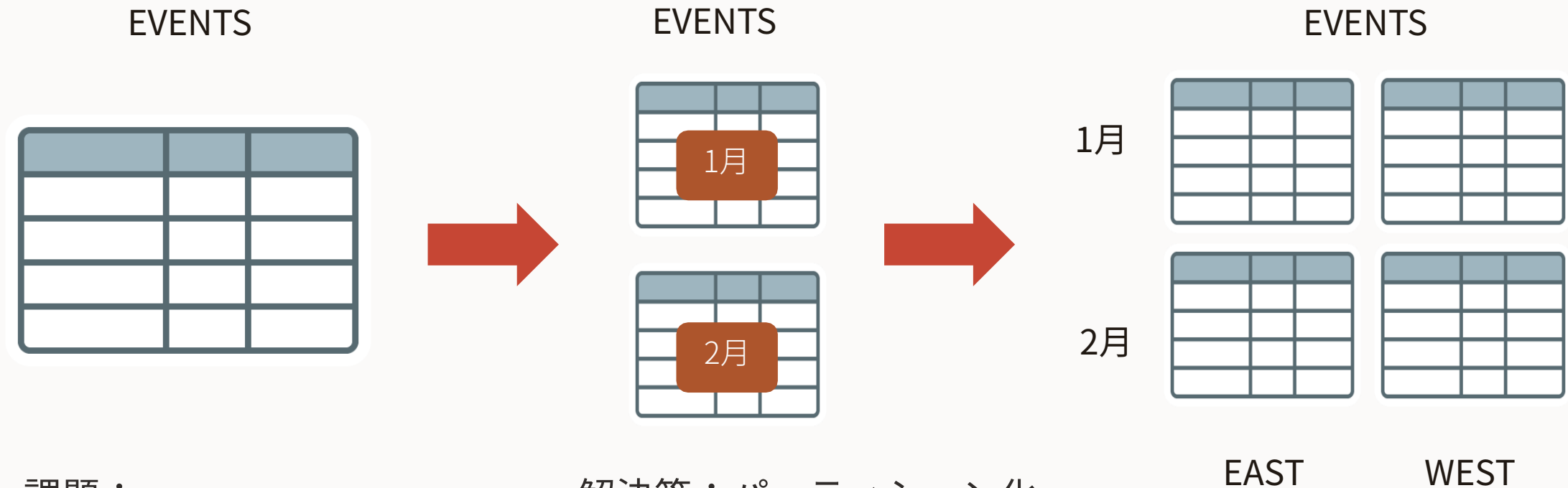
```
MOVE PARTITION  
COMPRESS  
READ ONLY;
```





# パーティション化の仕組み

大規模データベースと索引をより小規模で扱いやすい部分に分割可能



課題：  
大サイズの表は管理が  
困難

解決策：パーティション化

- 分割管理
- データ管理が容易
- パフォーマンスの向上

# パーティション化の概念

---

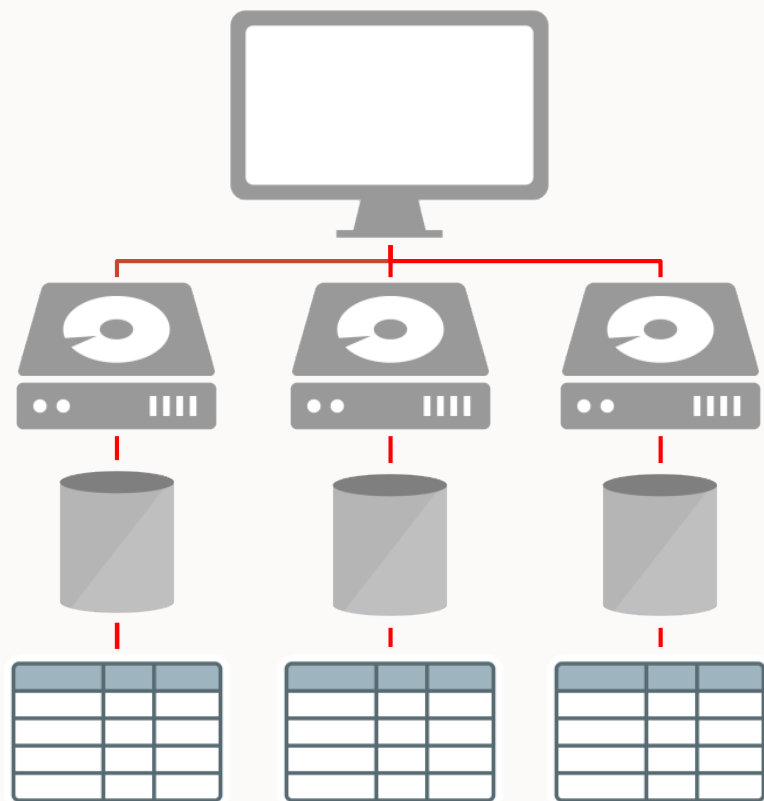
定義：パーティション

(何か) を部分に分割すること

“Merriam Webster Dictionary”

# 物理パーティション化

シェアード・ナッシング・アーキテクチャ



基本システムのセットアップ要件

- ノードがDBの一部を所有する
- 並列処理が可能

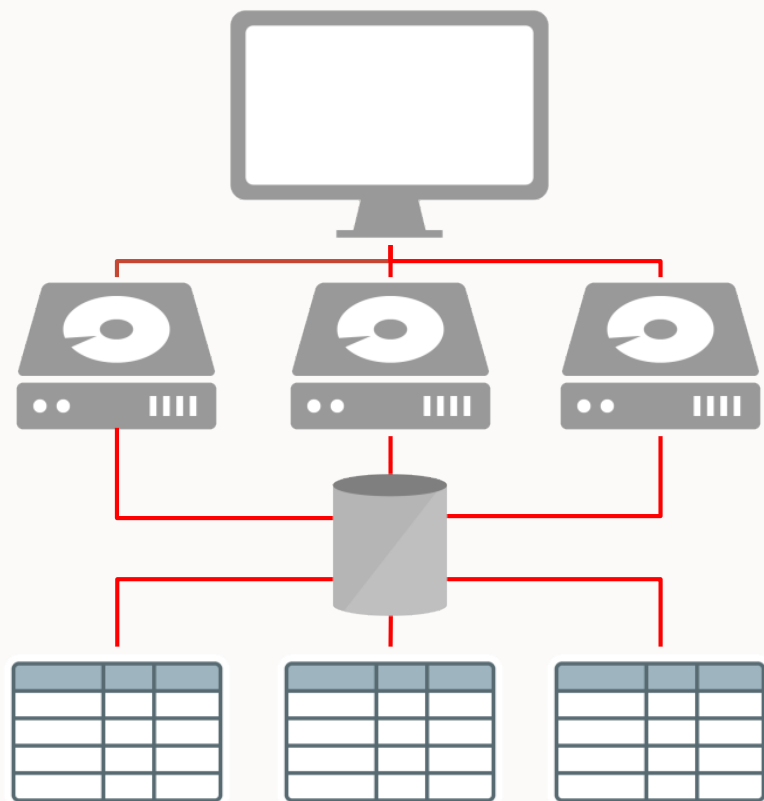
パーティション数が必要最小限の並列度と同等

- 常にハッシュまたはランダム分散が必要

適切なロードバランシングのために必要とされる、ノードあたり等サイズのパーティション

# 論理パーティション化

シェアード・エブリシング・アーキテクチャ - Oracle



根本的な制約なし

- SMP、MPP、クラスタ、グリッドの別を問わない

純粹にビジネス要件に基づく要件

- 可用性、管理性、パフォーマンス

すべての環境で有効

- もっとも包括的な機能を提供

# パーティション化の利点

---

# パフォーマンスの向上

## 関連するデータのみを処理

パーティション化により、次のようなデータ管理操作が可能...

- データのロード、結合、プルーニング
- 索引の作成と再構築
- オプティマイザ統計管理
- バックアップとリカバリ

... 表全体ではなくパーティション・レベルで操作

**結果：パフォーマンスが桁違いに向上**

# パフォーマンスの向上 - 例

## パーティション・プルーニング

5月1~2日の  
合計イベントは?



### EVENTS

5月5日
5月4日
5月3日
5月2日
5月1日
4月30日
4月29日

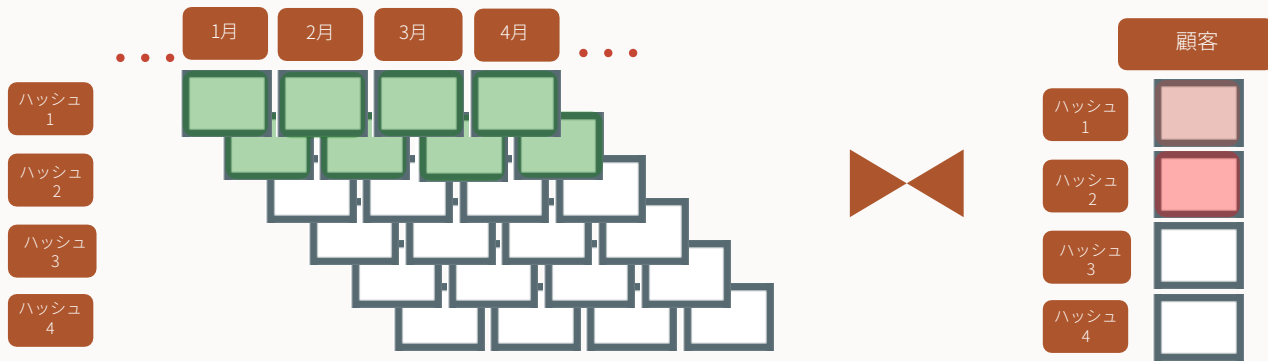
### パーティション・エリミネーション

- ストレージから取り込まれるデータ量を劇的に減少させる
- 関連するパーティションでのみ操作を実行する
- 問合せパフォーマンスを透過的に向上させ、リソース使用量を最適化する



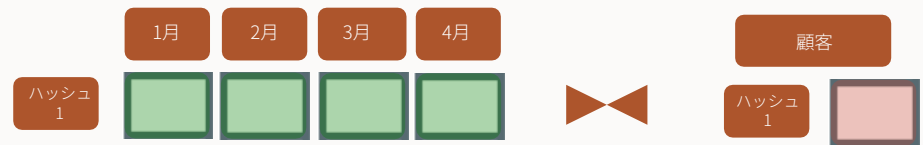
# パフォーマンスの向上 - 例

## パーティション・ワイズ結合



大規模結合は複数の小規模結合に分割され、並列実行される

- 結合するパーティションの数は、DOPの倍数であることが必須
- 両方の表は、結合列で同じ方法でパーティション化することが必要



# コストの削減

最適な方法でデータ保存

パーティション化では次の項目の間でバランスを取る...

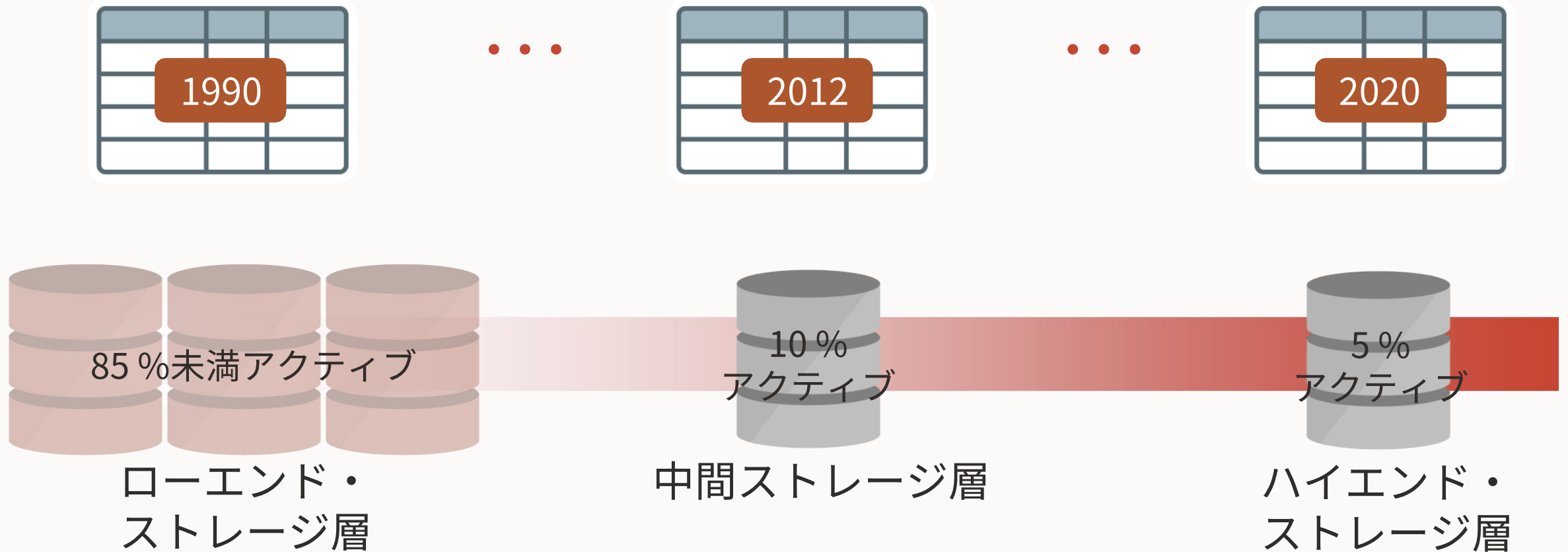
- データの重要性
- ストレージのパフォーマンス
- ストレージの信頼性
- ストレージの形式

...複数のストレージ層を利用できる

**結果：ストレージ・コストを2倍以上削減**

# コストの削減 - 例

層化ストレージ用のパーティション



# 高い可用性

個々のパーティションの管理性

パーティション化によって削減されるもの...

- メンテナンス期間
- スケジュールされた停止時間と障害の影響
- リカバリ時間

... 重要な表と索引がパーティション化されている場合

**結果：重要な情報へのアクセス性が向上**

# 可用性の向上 - 例

管理性/可用性を高めるためのパーティション



可視化できて使用可能な他のパーティション

## 実装が容易

アプリケーションに対して透過的

---

パーティション化ではアプリケーションと問合せに対する変更が不要

- パーティション化の利点をフル活用するには調整が必要になることもある

# 成熟して、十分に実証された機能

## 10年以上の開発期間

---

何万ものオラクルのお客様が使用  
さまざまなパーティション化方式をサポート

# 今日のOracle Partitioning

	おもな機能	パフォーマンス	管理性
Oracle 8.0	レンジ・パーティション化 ローカルおよびグローバル・レンジ索引	静的パーティション・プルーニング	基本的なメンテナンス：ADD、DROP、EXCHANGE
Oracle 8i	ハッシュ・パーティション化 レンジ-ハッシュ・パーティション化	パーティション・ワイズ結合 動的パーティション・プルーニング	メンテナンスの拡張：MERGE
Oracle 9i	リスト・パーティション化		グローバル索引メンテナンス
Oracle 9i R2	レンジ-リスト・パーティション化	高速パーティションSPLIT	
Oracle 10g	グローバル・ハッシュ索引		ローカル索引メンテナンス
Oracle 10g R2	表ごとに100万のパーティション	複数ディメンション・プルーニング	高速DROP TABLE
Oracle 11g	バーチャル・カラム・パーティション化 より多くの種類のコンポジット・パーティション 参照パーティション化		インターバル・パーティション化 Partition Advisor 増分統計管理
Oracle 11g R2	ハッシュ-*パーティション化 参照パーティション化の拡張	“AND”プルーニング	複数ブランチ実行（表拡張とも呼ばれる）
Oracle 12c R1	インターバル-参照パーティション化	複数パーティションでのパーティション・メンテナンス 非同期グローバル索引メンテナンス	オンライン・パーティションMOVE、 TRUNCATEのカスケード、部分索引付け
Oracle 12c R2	自動リスト・パーティション化 複数列リスト[サブ]パーティション化	オンラインでのパーティション・メンテナンス操作 表からパーティション表へのオンライン変換 DDLでのカーソル無効化の減少	フィルタ付きパーティション・メンテナ ンス操作 読取り専用パーティション 交換用の表の作成
Oracle 18c	パーティション外部表	並列パーティション・ワイズSQL操作 オンライン・パーティション・メンテナンスの完了 強化されたオンライン表変換	データ・コンテンツの検証
Oracle 19c	ハイブリッド・パーティション表		オブジェクト・ストレージ・アクセス*





# パーティション化の方法

---

# パーティション化が可能な要素

## 表

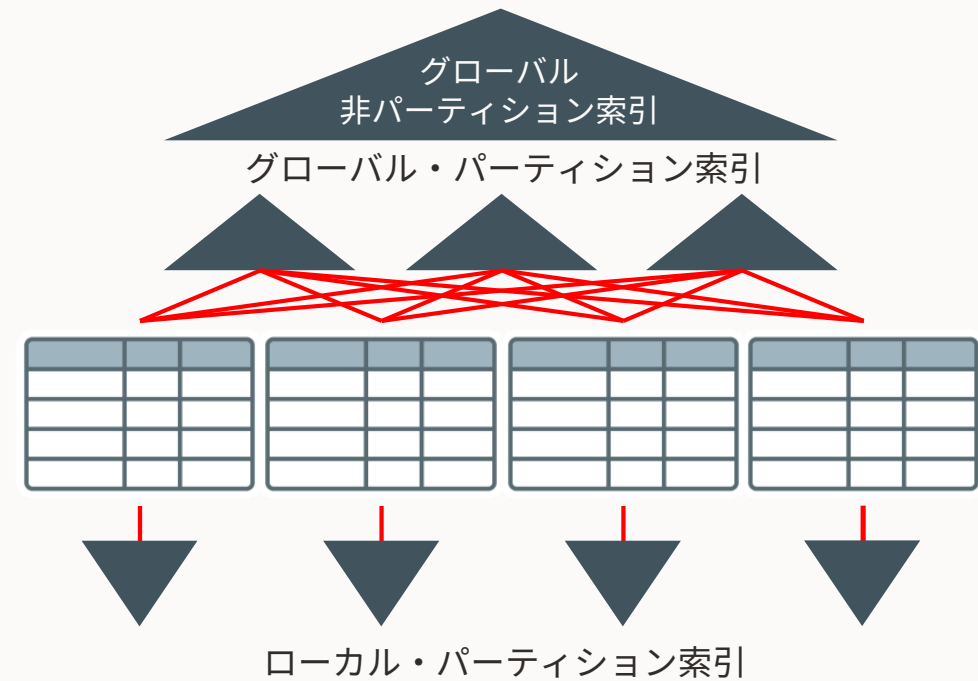
- ヒープ表
- 索引構成表

## 索引

- グローバル索引
- ローカル索引

マテリアライズド・ビュー

ハッシュ・クラスタ



# パーティション化の方法

## 単一レベル・パーティション化

- レンジ
- リスト
- ハッシュ

## コンポジット・レベル・パーティション化

- [レンジ | リスト | ハッシュ | インターバル] -  
[レンジ | リスト | ハッシュ]

## パーティション化の拡張機能

- インターバル
- 参照
- インターバル参照
- バーチャル・カラム・ベース
- 自動

# レンジ・パーティション化

Oracle 8.0で導入

## レンジ・パーティション化



データはレンジ単位で編成される

- 下側の境界は、先行するパーティションの上側の境界によって導出される
- 必要に応じて分割およびマージする
- ギャップなし

時系列データに理想的

# リスト・パーティション化

Oracle 9i (9.0) で導入

## リスト・パーティション化



データは値リスト単位で編成される

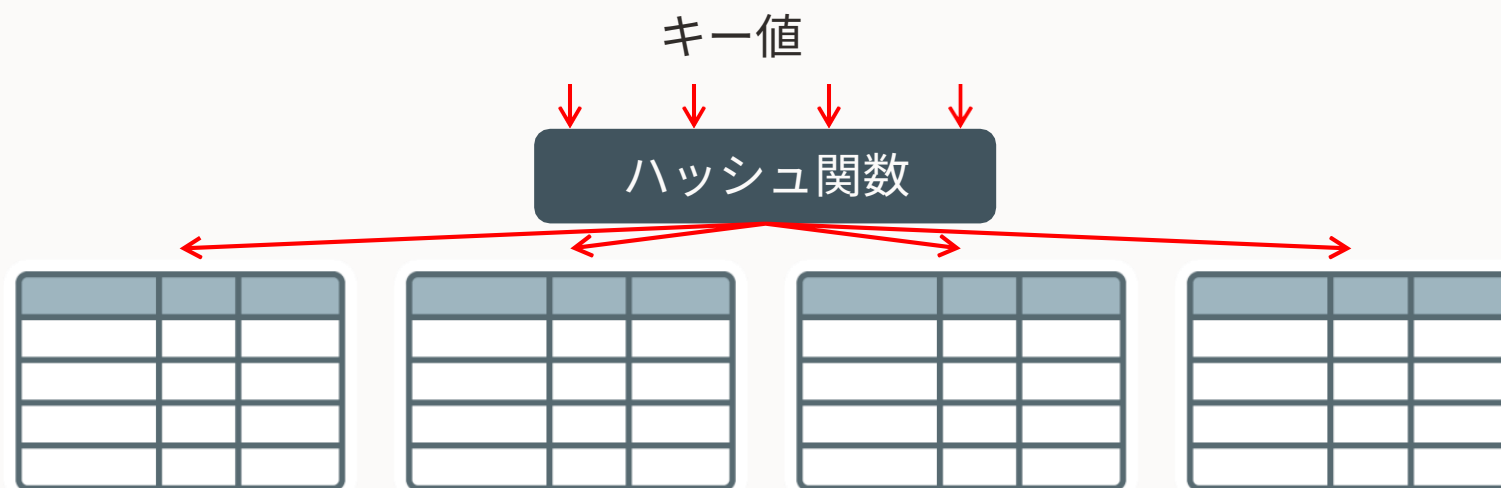
- リストあたり1つ以上の順序付けされていない個別値
  - DEFAULTパーティションの機能（すべての未指定値をまとめて取込み）
  - DEFAULTパーティションのコンテンツの確認 - 必要に応じて新規パーティションを作成
- リージョンなどの個別値のセグメント化に理想的

# ハッシュ・パーティション化

Oracle 8i (8.1) で導入



# ハッシュ・パーティション化



データはパーティション・キーのハッシュ値に基づいて配置される

- ハッシュ・バケットの数はパーティション数と等しい

均等なデータ分散に理想的

- 均等なデータ分散の場合のパーティション数は、2の累乗にする

## コンポジット・パーティション化

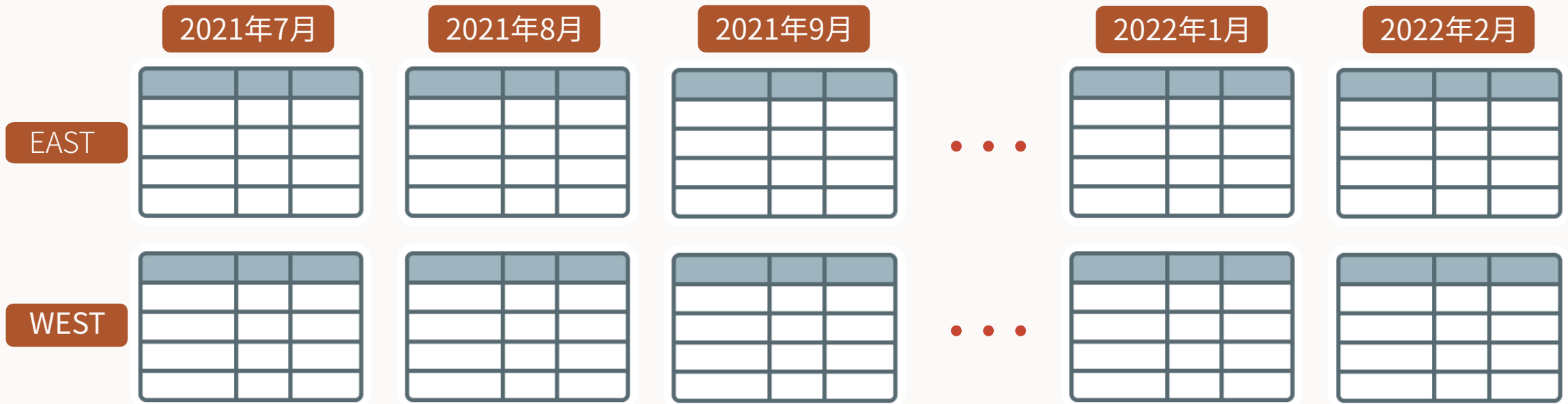
レンジ-ハッシュはOracle 8iで導入

レンジ-リストはOracle 9i Release 2で導入

[インターバル | レンジ | リスト | ハッシュ]-[レンジ | リスト | ハッシュ]はOracle 11g Release 1|2で導入

\*ハッシュ-ハッシュは11.2で導入

# コンポジット・パーティション化



データは2つのディメンションに編成される

- レコード配置はディメンションによって決定的に特定される  
- 例：レンジ-リスト

# コンポジット・パーティション化

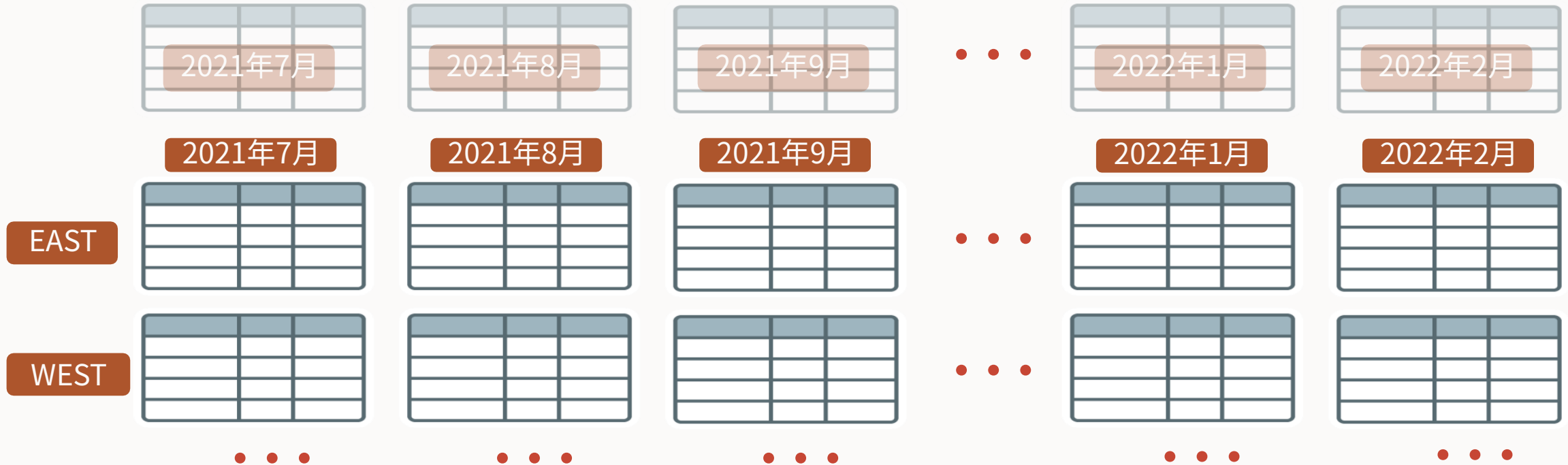
## 概念



```
CREATE TABLE EVENTS ..PARTITION BY RANGE (time_id)
```

# コンポジット・パーティション化

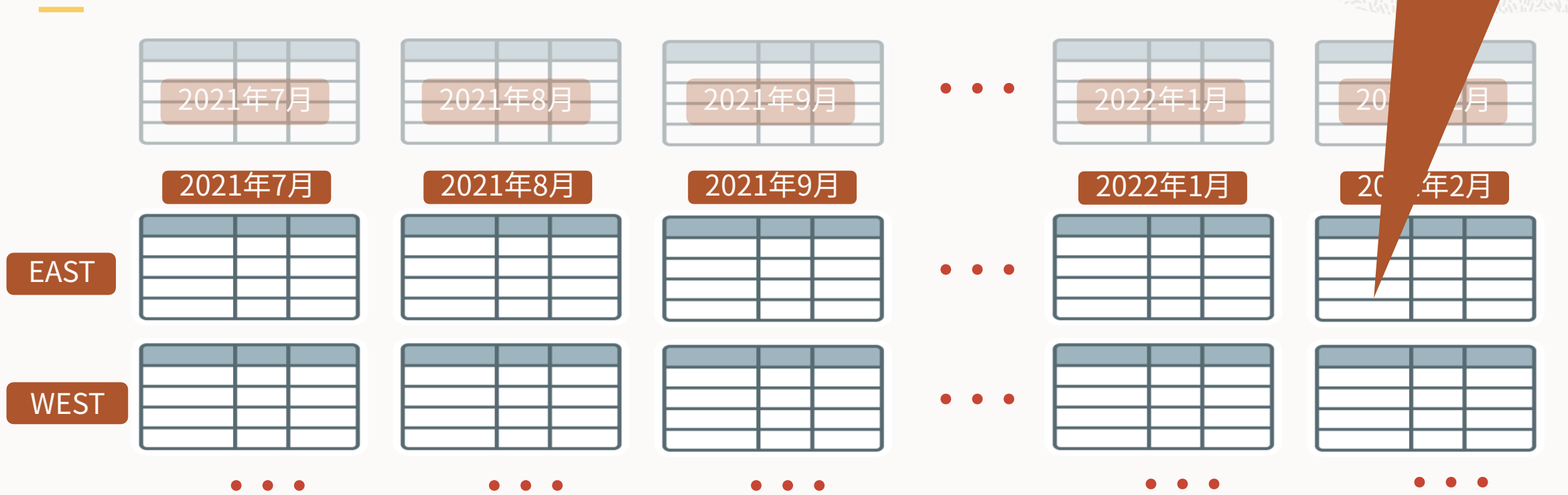
## 概念



```
CREATE TABLE EVENTS ..PARTITION BY RANGE (time_id)  
SUPARTITION BY LIST (region)
```

# コンポジット・パーティション化

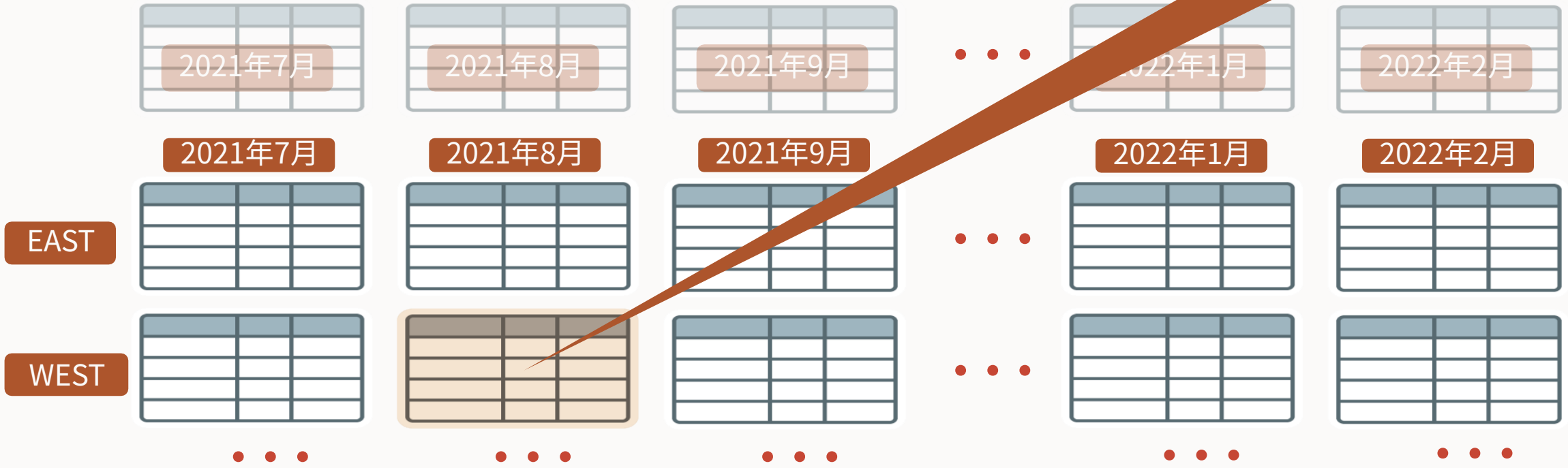
## 概念



```
CREATE TABLE EVENTS ..PARTITION BY RANGE (time_id)  
SUPARTITION BY LIST (region)
```

# コンポジット・パーティション化 概念

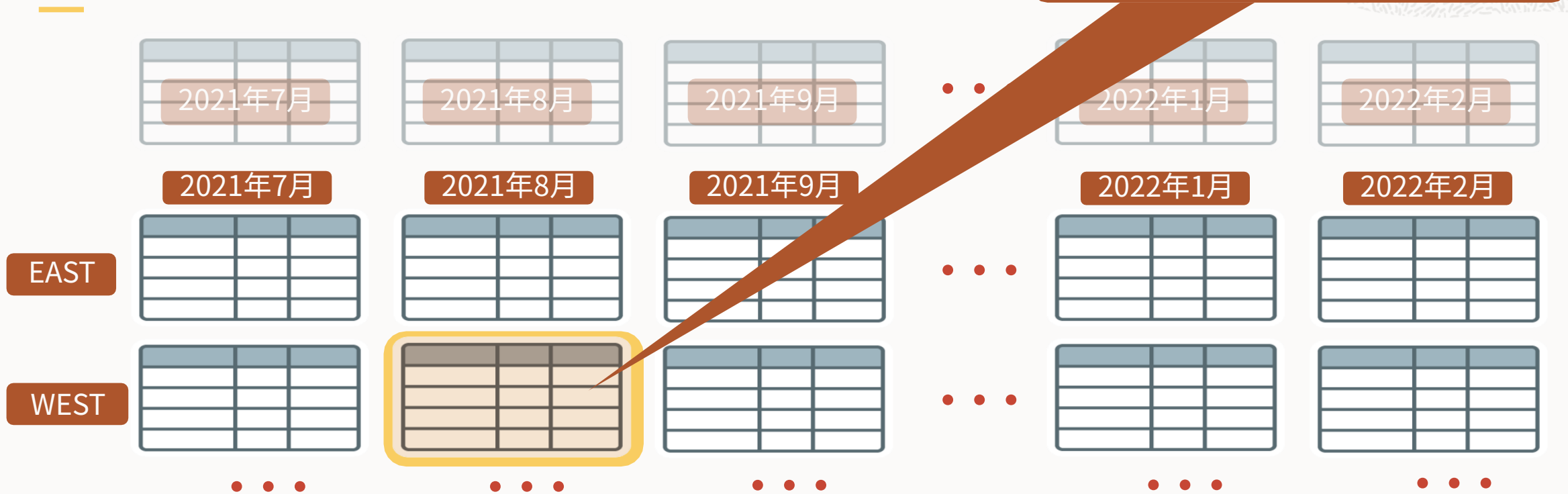
2021年8月のWESTの  
データ



```
CREATE TABLE EVENTS ..PARTITION BY RANGE (time_id)  
SUPARTITION BY LIST (region)
```

# コンポジット・パーティション化

## 概念



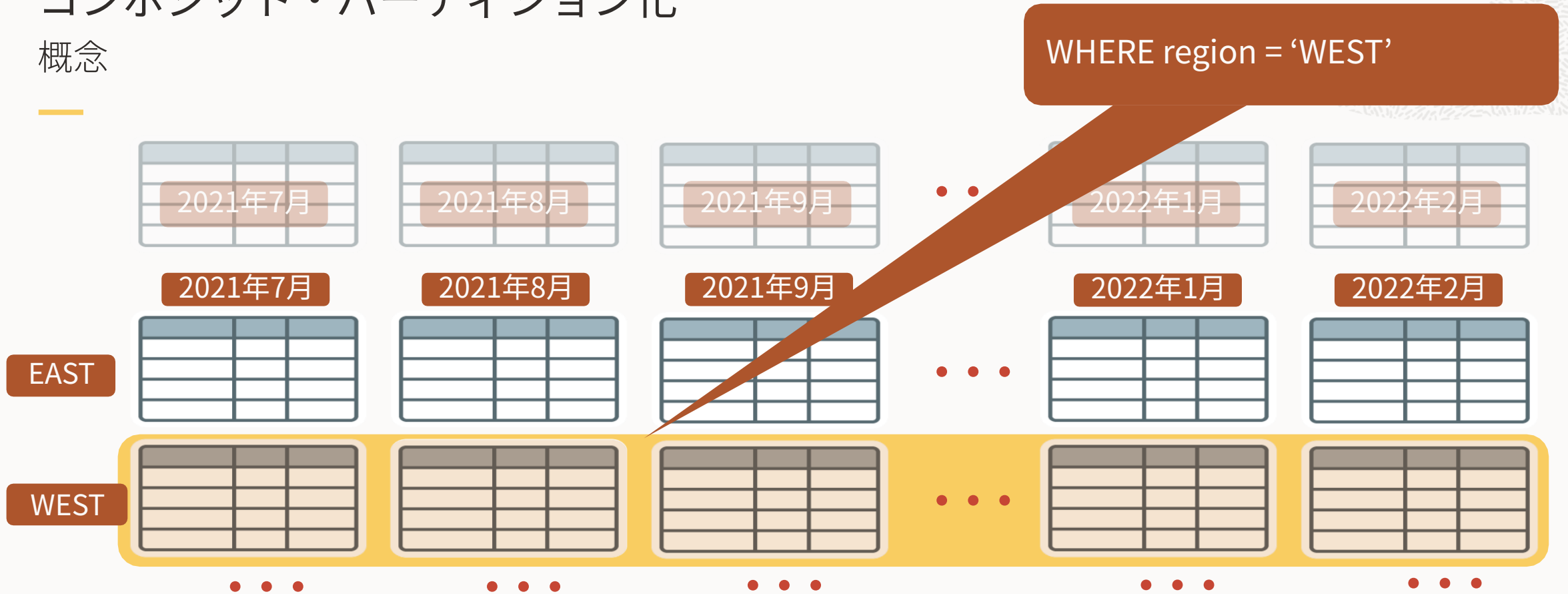
パーティション・プルーニングはコンポジット順序に依存しない

- 一方または両方のディメンションに沿ったプルーニング
- レンジ-リストおよびリスト-レンジで同じプルーニング



# コンポジット・パーティション化

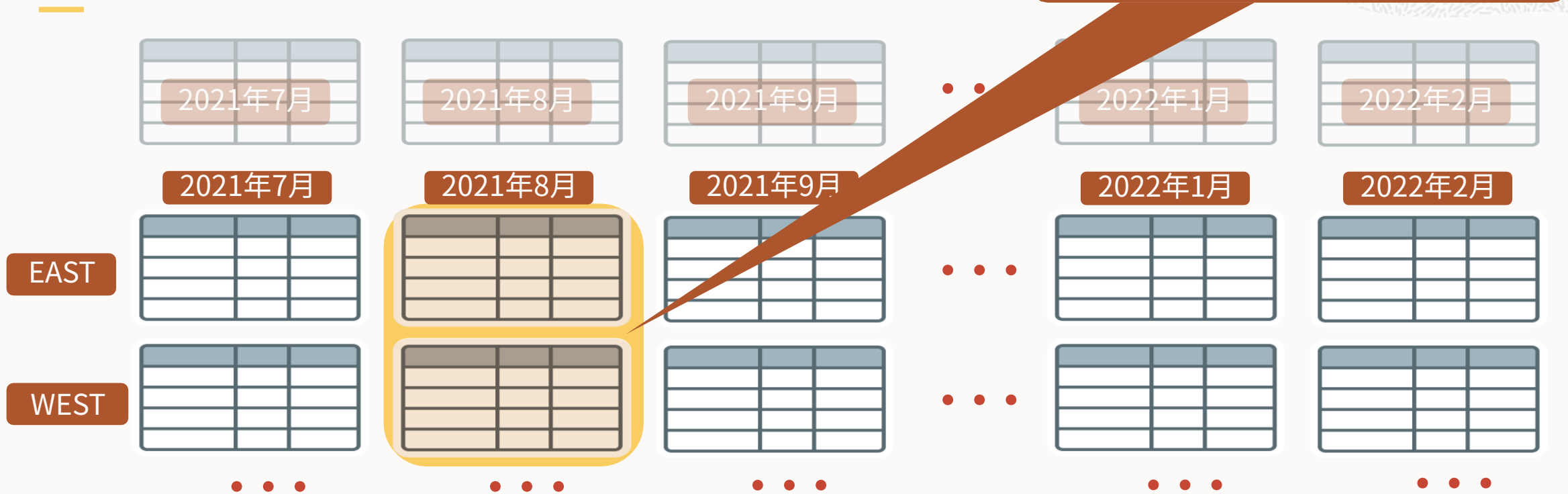
## 概念



パーティション・プルーニングはコンポジット順序に依存しない

- 一方または両方のディメンションに沿ったプルーニング
- レンジ-リストおよびリスト-レンジで同じプルーニング

# コンポジット・パーティション化 概念

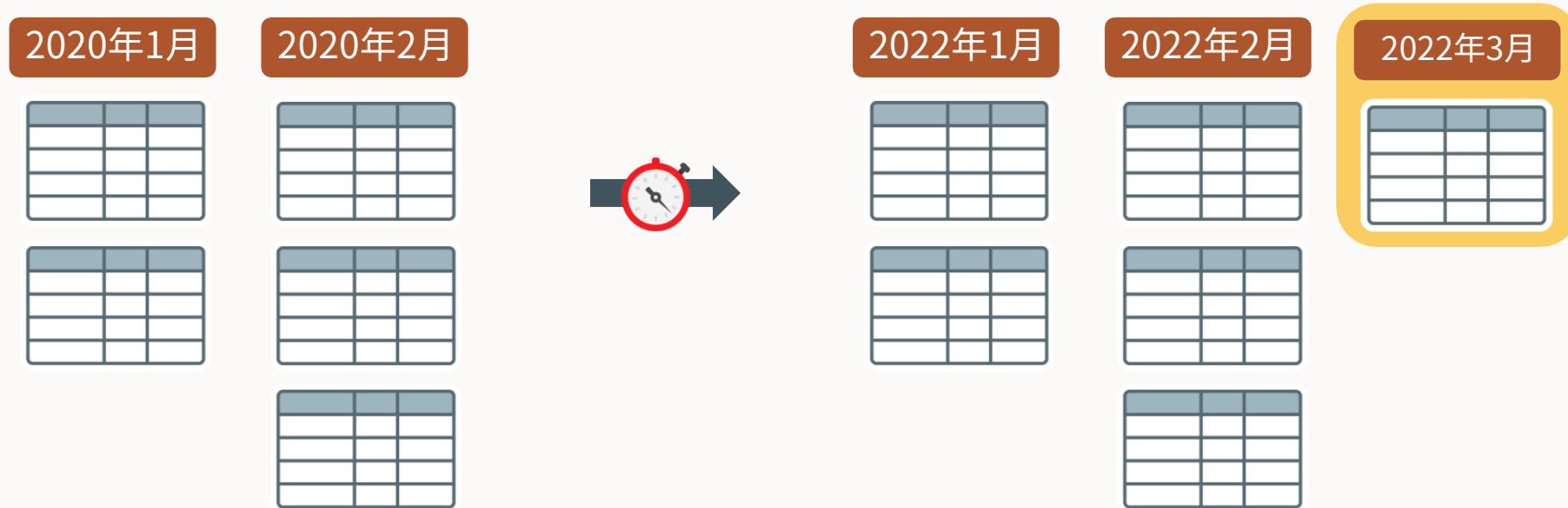


パーティション・プルーニングはコンポジット順序に依存しない

- 一方または両方のディメンションに沿ったプルーニング
- レンジ-リストおよびリスト-レンジで同じプルーニング

# コンポジット・インターバル・パーティション化

## パーティションの追加



サブパーティション・テンプレートなしの場合は、サブパーティションが1つだけ作成される

- レンジ：MAXVALUE
- リスト：DEFAULT
- ハッシュ：1つのハッシュ・バケット

# コンポジット・インターバル・パーティション化

## サブパーティション・テンプレート

サブパーティション・テンプレートにより将来のサブパーティションを定義する

- 任意の時点で追加または変更可能
- 既存の[サブ]パーティションへの影響なし

サブパーティションの物理属性も制御する

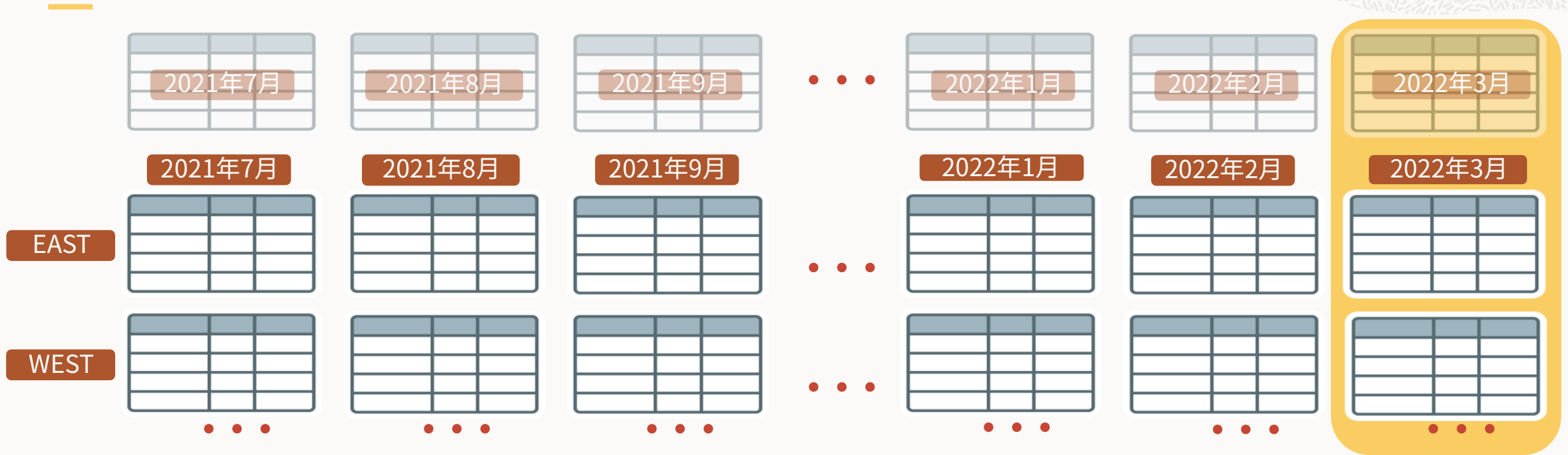
- パーティションの場合のパーティション表のデフォルト設定と同様

インターバル・パーティション化とレンジ・パーティション化の相違

- ネーミング・テンプレートはレンジ・パーティション化の場合のみ
- システム生成名はインターバル・パーティション化の場合

# コンポジット・パーティション化

## パーティションの追加

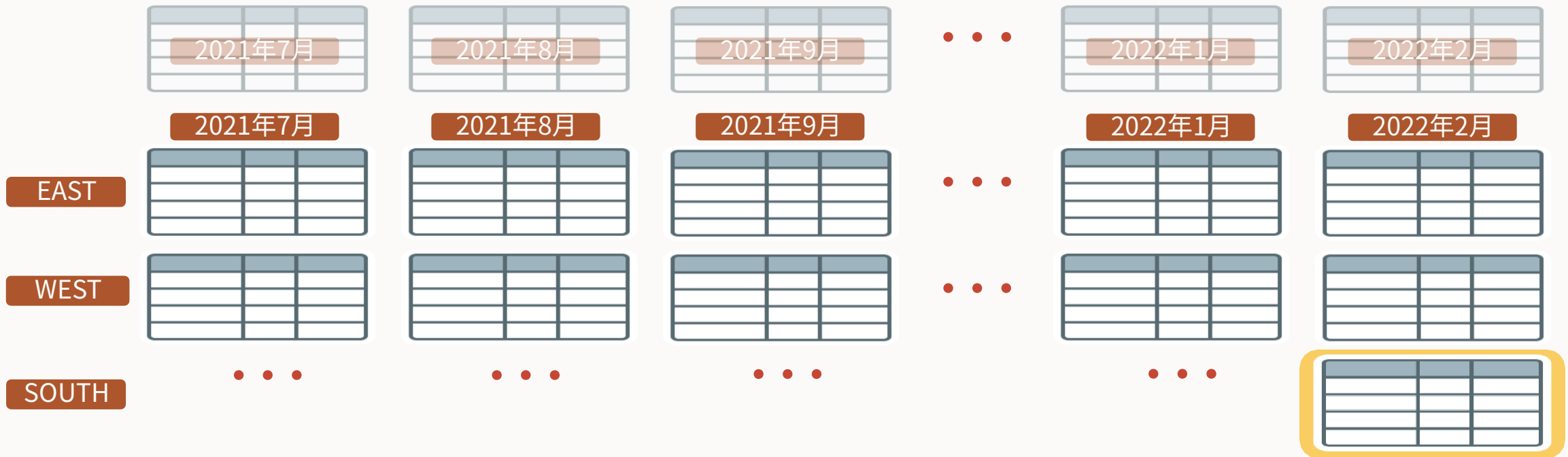


ADD PARTITIONは常に最上位レベルのディメンション

- すべての新規追加サブパーティションの場合に同一
  - レンジ-リスト：新しいtime\_idレンジ
  - リスト-レンジ：新しいリージョン値リスト

# コンポジット・パーティション化

## サブパーティションの追加

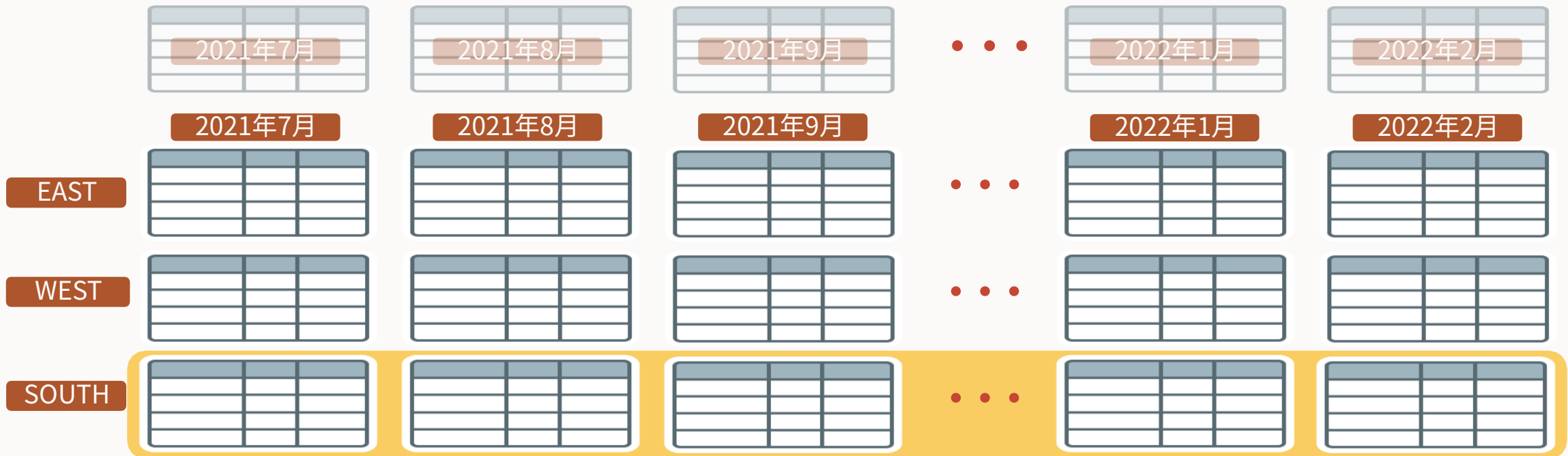


ADD SUBPARTITIONは1つのパーティションの場合のみ

- 非対称、サブパーティション・レベルでのみ可能
- パーティション・ワイズ結合への影響

# コンポジット・パーティション化

## サブパーティションの追加

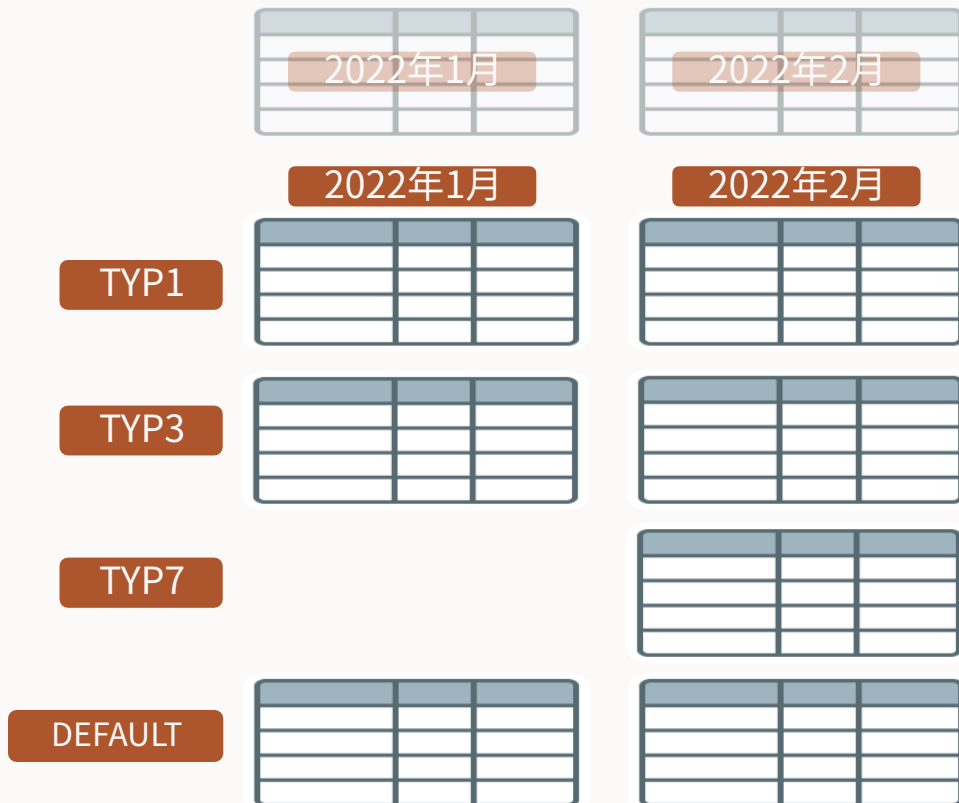


すべてのパーティションの場合にADD SUBPARTITION

- N回の操作が必要（既存のパーティションごと）
- 将来のパーティションのサブパーティション・テンプレートを調整

# コンポジット・パーティション化

## 非対称サブパーティション



サブパーティションの数は個々のパーティションごとに異なる

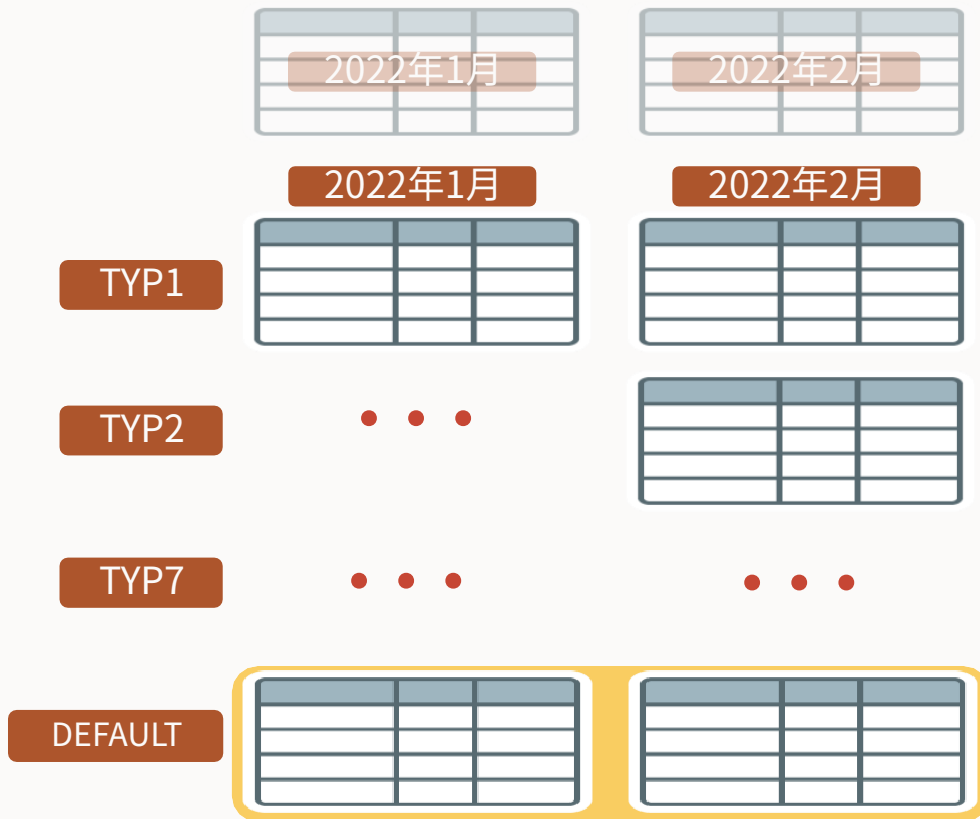
- リスト・サブパーティション戦略でもっとも一般的

```
CREATE TABLE EVENTS..  
PARTITION BY RANGE (time_id)  
SUPARTITION BY LIST (model)
```



# コンポジット・パーティション化

## 非対称サブパーティション



サブパーティションの数は個々のパーティションごとに異なる

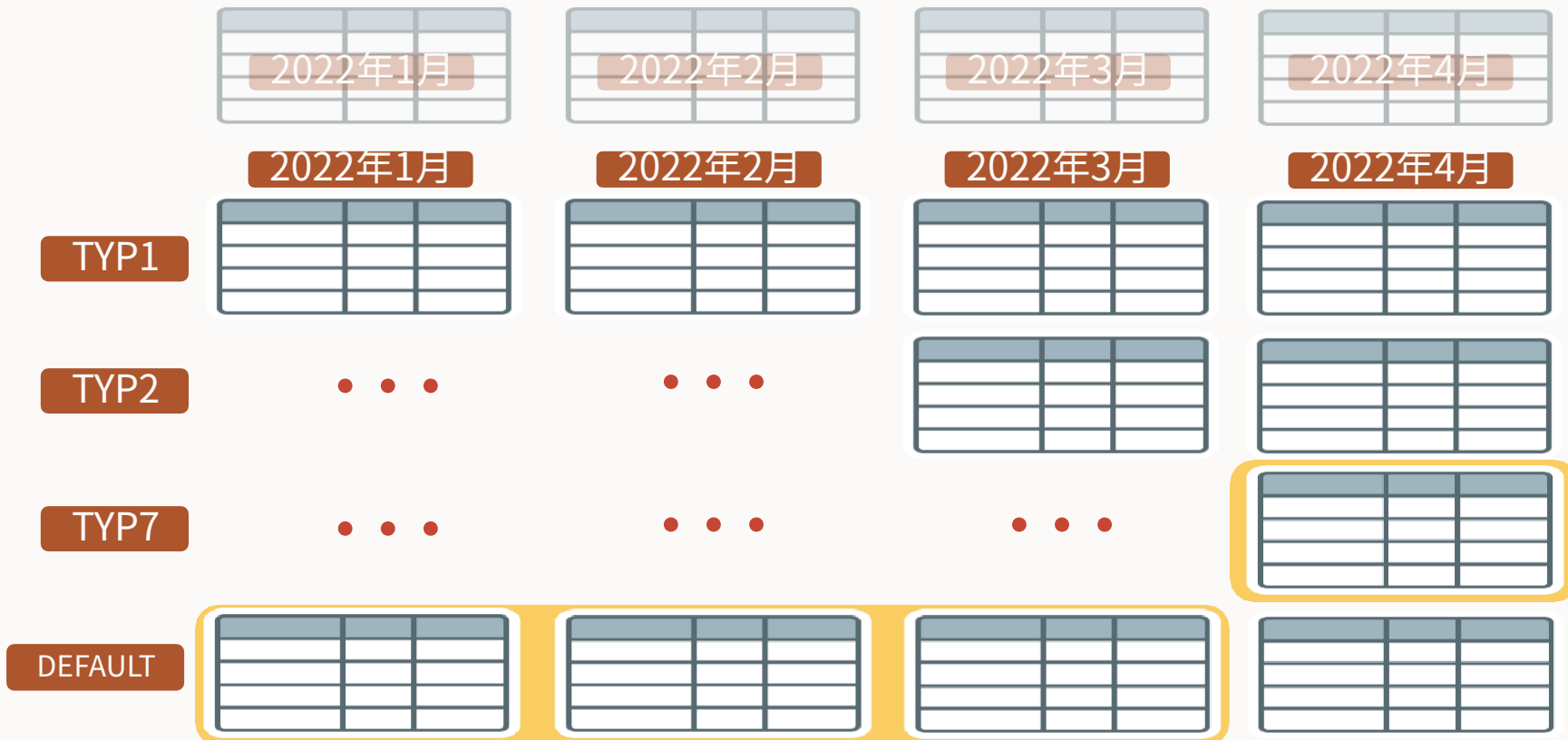
- リスト・サブパーティション戦略でもっとも一般的

パーティション・プルーニング機能への影響なし

```
SELECT ..FROM events  
WHERE model = 'TYP7';
```

# コンポジット・パーティション化

## 非対称サブパーティション



```
SELECT ..FROM events  
WHERE model = 'TYP7';
```

# コンポジット・パーティション化

## 常に適切なコンポジット戦略を使用する

おもに管理性の向上のために選択されるトップレベル・ディメンション

- 例：時間範囲の追加および削除

パフォーマンスまたは管理性の向上のために選択されるサブレベル・ディメンション

- 例：load\_id、customer\_id

非対称にはメリットがあるが、熟考することが必要

- 例：異なるリージョンごとに異なる時間粒度
- 非対称コンポジット・パーティション化の影響を忘れない

# パーティション化と索引付け

---

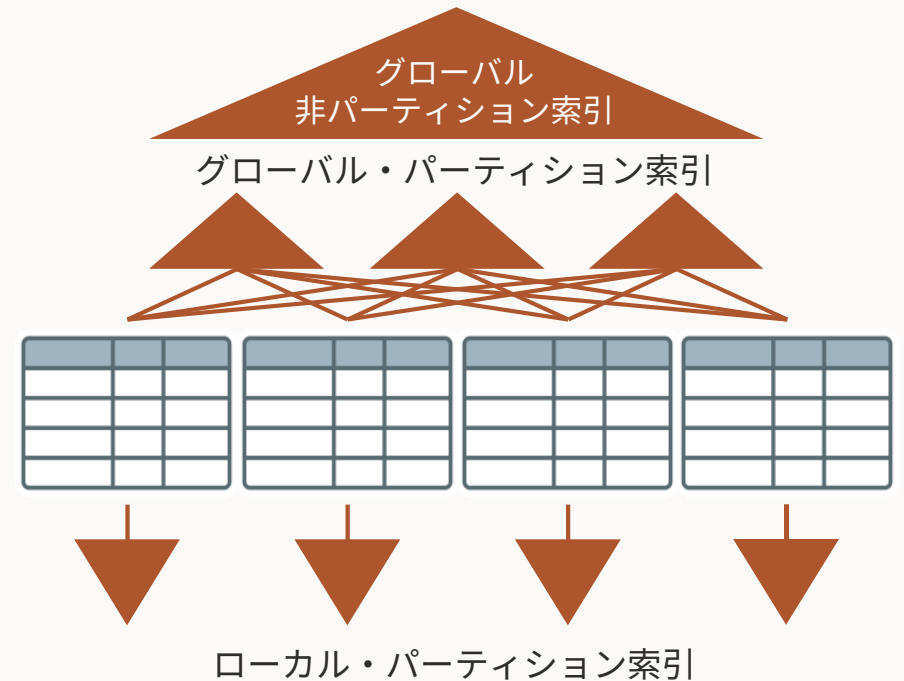
## パーティション表の索引付け

グローバル索引は任意のパーティションの行を参照する

- 索引はパーティション化してもしなくても構わない

ローカル索引は、表と同様にパーティション化される

- 索引パーティション化キーは、索引キーと異なる場合がある



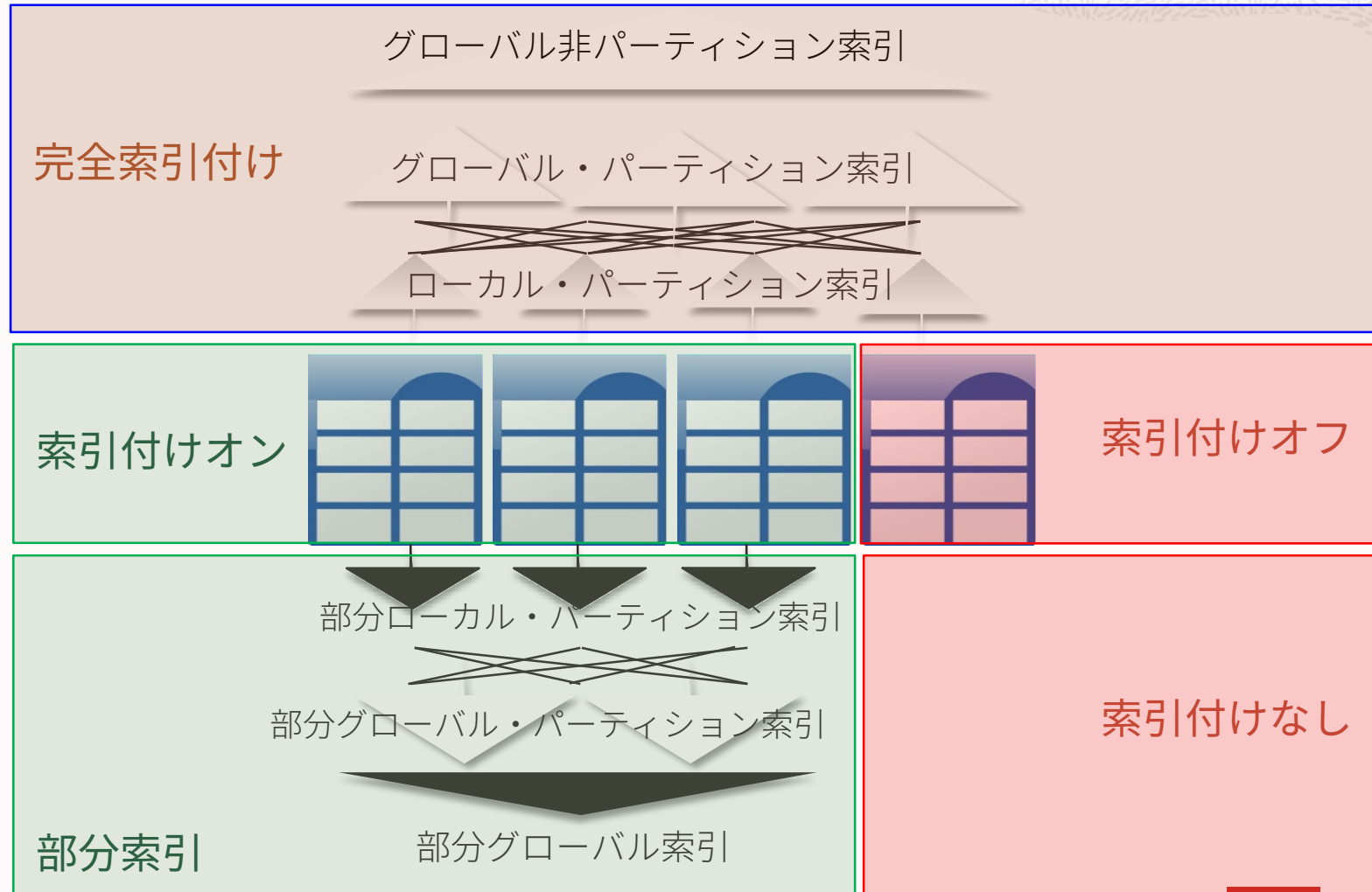
# パーティション表の索引付け

部分索引の対象は一部のパーティションのみ

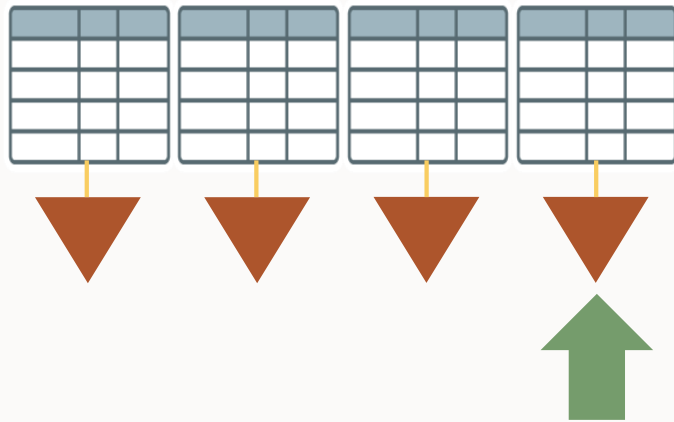
ローカル索引とグローバル索引に適用可能

完全索引を補完

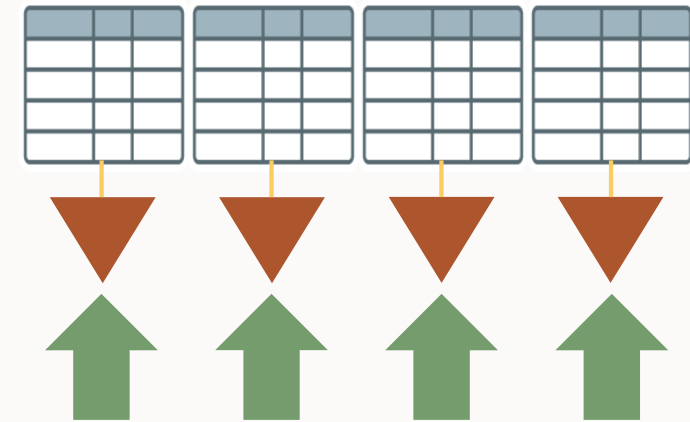
オンライン索引メンテナンスのフル・サポート



# データ・アクセス - ローカル索引とグローバル・パーティション索引



単一パーティション・プルーニングありの  
パーティション索引アクセス



パーティション・プルーニングなしの  
パーティション索引アクセス

# データ・アクセス - ローカル索引とグローバル・パーティション索引

アクセスされたパーティションの数と同一の索引プローブの数

- すべての索引パーティションをプローブするパーティション・プルーニングなし

OLTP環境向けに最適化されていない

- 常にパーティション・プルーニングがあるという保証はない
- 例外：DML競合緩和のためのグローバル・ハッシュ・パーティション索引
  - ごく一般的に少数のパーティション

索引接頭辞に基づくグローバル・パーティション索引でのプルーニング

- 索引の先頭キーと同一の索引接頭辞



# ローカル索引

索引は、表（データ）パーティションと同じ境界に沿ってパーティション化される

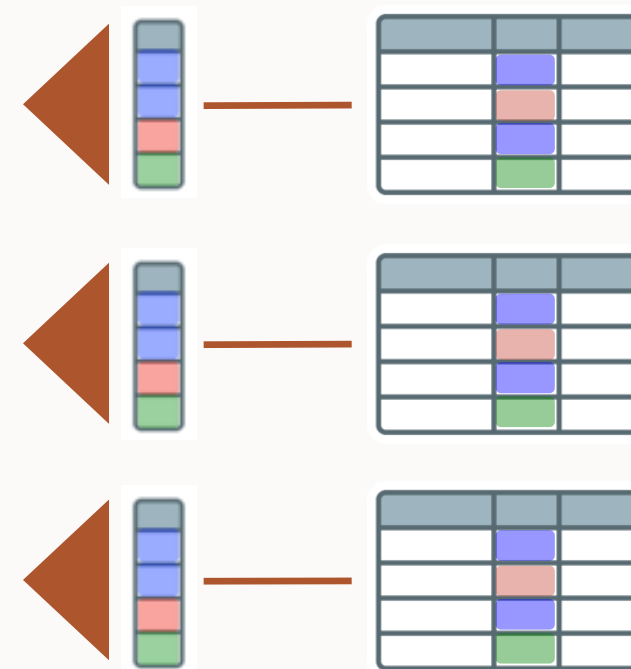
- Bツリーまたはビットマップ

## 長所

- 管理が容易
- パラレル索引スキャン

## 短所

- 少量データを取得する場合には効率が低下する（所定のパーティション・プルーニングなし）



# グローバル非パーティション索引

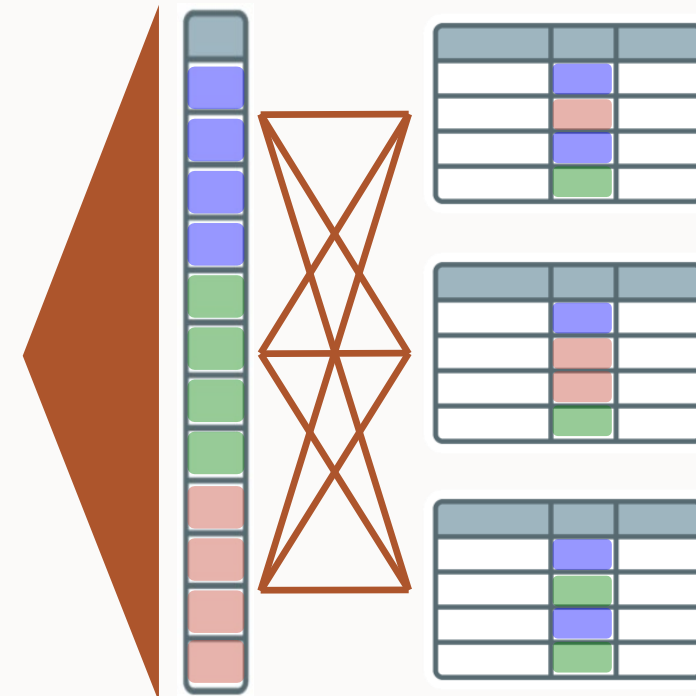
すべてのパーティションを対象とする1つの索引Bツリー構造

## 長所

- 任意の個々のレコードに効率よくアクセスできる

## 短所

- パーティション・メンテナンスには索引メンテナンスが常に関係する



# グローバル・パーティション索引

索引はデータに関係なくパーティション化される

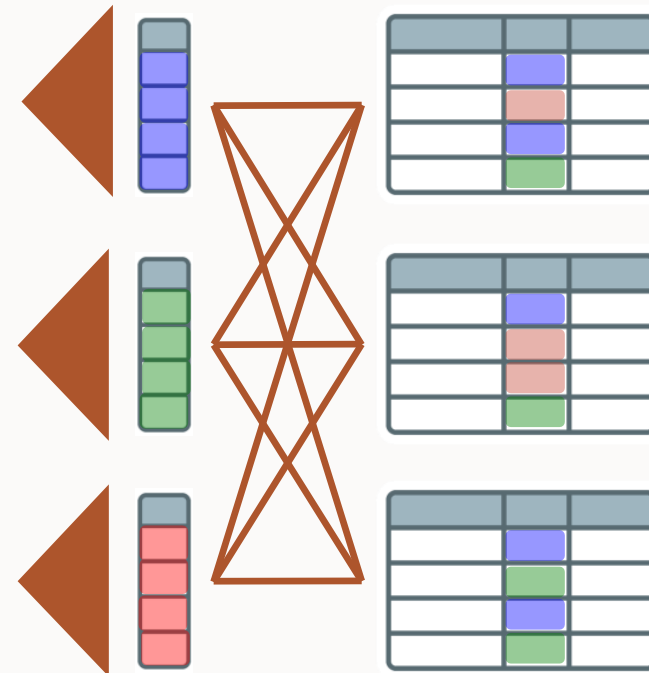
- 各索引構造で任意またはすべてのパーティションを参照できる

長所

- 可用性と管理性

短所

- パーティション・メンテナンスには索引メンテナンスが常に関係する



## 索引メンテナンスとパーティション・メンテナンス

オンライン索引メンテナンスは、グローバルとローカルの**両方**の索引で実行可能

- グローバル索引メンテナンスはOracle 9i以降、ローカル索引メンテナンスはOracle 10g以降

DROPとTRUNCATEの場合、ローカルとグローバルの**両方**の索引で高速索引メンテナンス

- Oracle 12c Release 1には非同期グローバル索引メンテナンスを追加

索引メンテナンスは、他のすべてのパーティション・メンテナンス操作の場合にローカルとグローバルの**両方**の索引で必須

## 索引メンテナンスとパーティション・メンテナンス

オンライン索引メンテナンスは、グローバルとローカルの両方の索引で実行可能

- グローバル索引メンテナンスはOracle 9i以降、ローカル索引メンテナンスはOracle 10g以降

DROPとTRUNCATEの場合、ローカルとグローバルの両方の索引で高速索引メンテナンス

- Oracle 12c Release 1には非同期グローバル索引メンテナンスを追加

索引メンテナンスは、他のすべてのパーティション・メンテナンス操作の場合にローカルとグローバルの両方の索引で必須

索引メンテナンスによるパーティション・メンテナンスでの判断事項は、いつでもパフォーマンスと可用性のバランスである

- 索引の再構成は常にデータの5%~10%以上が関係する場合に速い

新旧両方のデータで部分索引付けを検討する

- 最初にすべてのデータを索引付けする必要はない

# 一意制約と主キーの索引付け

---

## 一意制約/主キー

一意制約は一意索引によって適用される

- 主キー制約によってNOT NULLが列に追加される
- 表には主キー ("一意の識別子") を1つだけ指定できる

パーティション表には2つのタイプの索引がある

- ローカル索引
- グローバル索引、パーティションと非パーティションの両方

どちらを選択すべきか

- そもそも選択肢があるか

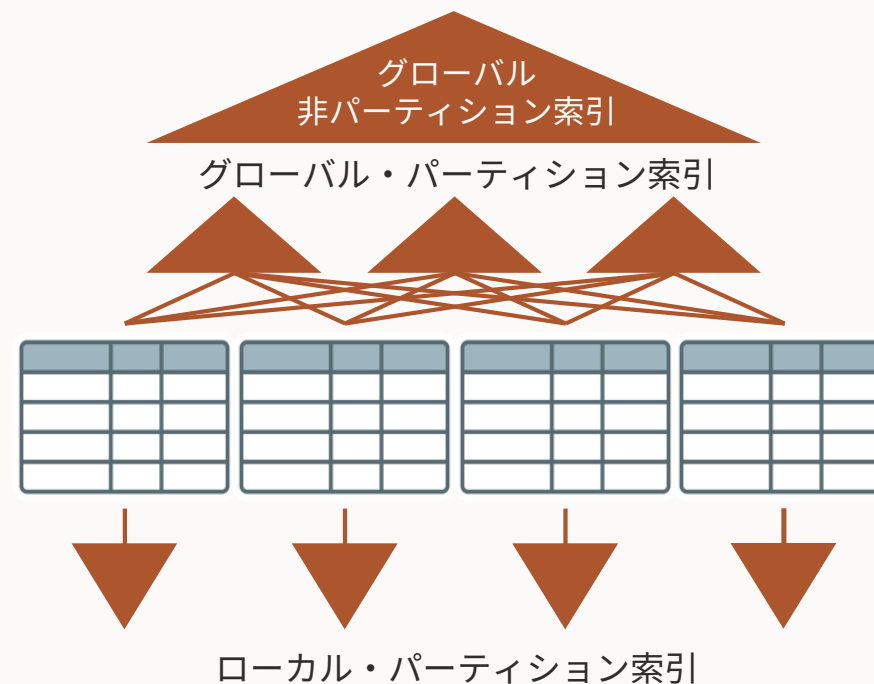
# 索引のパーティション化

グローバル索引はすべてのパーティションの行を参照する

- 索引はパーティション化してもしなくても構わない
- パーティション・メンテナンスは索引全体に影響する

ローカル索引は1つのパーティションの行を参照する

- 索引は、表と同様にパーティション化される
- 索引パーティション化キーは、索引キーと異なる場合がある
- 索引パーティションは別々にメンテナンス可能





## 一意制約/主キー

### ローカル索引の適用可能性

ローカル索引は、表によって等間隔パーティション化される

- 表パーティションの自律性概念に従う
  - “自分のことだけ考える”

一意性を適用するためのローカル索引の要件

- パーティション・キー列を一意キーのサブセットにする

## 一意制約/主キー（続き）

### ローカル索引の適用可能性

ローカル索引は、表によって等間隔パーティション化される

- 表パーティションの自律性概念に従う
  - “自分のことだけ考える”

一意性を適用するためのローカル索引の要件

- パーティション・キー列は、一意キーのサブセットでなければならない



Three tables are shown, each with 4 columns and 4 rows. The first two columns of each table are shaded blue. A large green checkmark is placed to the right of the third table.

PARTITION BY (col1), PK(col1)



Three tables are shown, each with 4 columns and 4 rows. The first two columns of each table are shaded blue. A large red prohibition sign (a circle with a diagonal slash) is placed to the right of the third table.

PARTITION BY (col1), PK(col2)

## 一意制約/主キー（続き）

### グローバル索引の適用可能性

グローバル索引と表のパーティションとの間にリレーションはない

- 定義により、グローバル索引にはすべてのパーティションからのデータが格納される
- パーティションと非パーティションの両方のグローバル索引に当てはまる

グローバル索引は、一意性を適用するためにいつでも使用可能



PARTITION BY (col1), PK(col1)



PARTITION BY (col1), PK(col2)

# 部分索引付け

Oracle 12c Release 1 (12.1) で導入

# Oracle Partitioningによる索引付け機能の強化

Oracle Database 12cより前の索引付け

ローカル索引

非パーティションまたはパーティションのグローバル索引

使用可能または使用不可の索引セグメント

- 索引の非永続ステータス、表とのリレーションなし

# Oracle Partitioningによる索引付け機能の強化

## Oracle Database 12cでの索引付け

### ローカル索引

非パーティションまたはパーティションのグローバル索引

使用可能または使用不可の索引セグメント

- 索引の非永続ステータス、表とのリレーションなし

### 部分ローカル索引と部分グローバル索引

- 部分索引付けで、表と[サブ]パーティション・レベルのメタデータを導入
- ローカル・パーティション索引の使用可能/使用不可状態を利用
- 部分索引付けのポリシーは上書き可能

# パーティション表の索引付け機能の強化

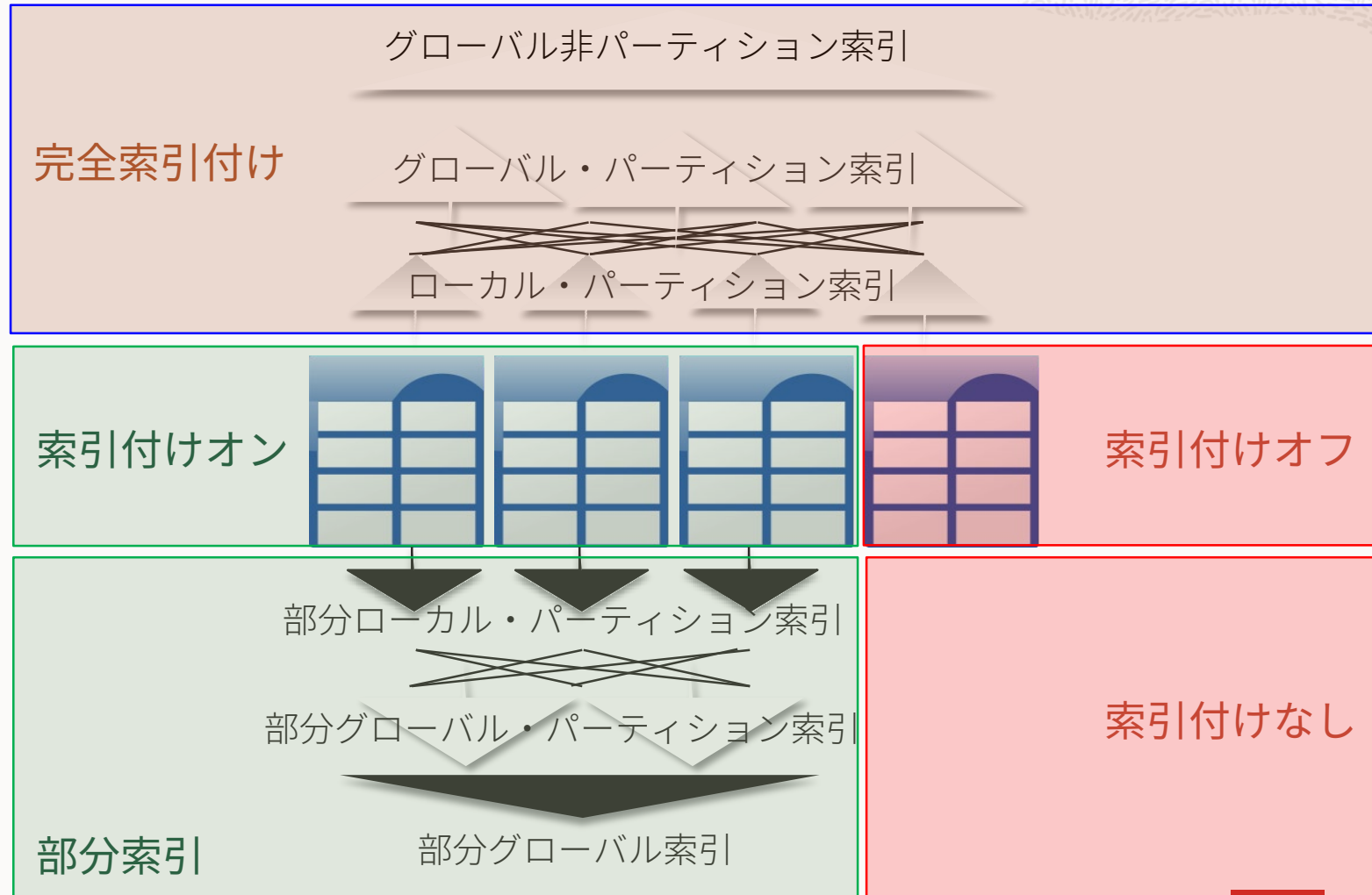
## 部分ローカル索引と部分グローバル索引

部分索引の対象は一部のパーティションのみ

ローカル索引とグローバル索引に適用可能

完全索引を補完

オンライン索引メンテナンスのフル・サポート



# Oracle Partitioningによる索引付け機能の強化

## 部分ローカル索引と部分グローバル索引

### 変更前

```
SQL> create table pt (col1, col2, col3, col4)
  2  indexing off
  3  partition by range (col1)
  4  interval (1000)
  5  (partition p100 values less than (101) indexing on,
  6  partition p200 values less than (201) indexing on,
  7  partition p300 values less than (301) indexing on);
```

Table created.

```
SQL> REM partitions and its indexing status
SQL> select partition_name, high_value, indexing
  2  from user_tab_partitions where table_name='PT';
```

PARTITION_NAME	HIGH_VALUE	INDEXING
P100	101	ON
P200	201	ON
P300	301	ON
SYS_P1256	1301	OFF

### 変更後

```
SQL> REM local indexes
SQL> create index i_l_partpt on pt(col1) local indexing partial;
SQL> create index i_l_pt on pt(col4) local;
```

```
SQL> REM global indexes
SQL> create index i_g_partpt on pt(col2) indexing partial;
SQL> create index i_g_pt on pt(col3);
```

```
SQL> REM index status
SQL> select index_name, partition_name, status, null
  2  from user_ind_partitions where index_name in ('I_L_PARTPT','I_L_PT')
  3  union all
  4  select index_name, indexing, status, orphaned_entries
  5  from user_indexes where index_name in ('I_G_PARTPT','I_G_PT');
```

INDEX_NAME	PARTITION_NAME	STATUS	ORPHAN
I_L_PARTPT	P100	USABLE	
I_L_PARTPT	P200	USABLE	
I_L_PARTPT	P300	USABLE	
I_L_PARTPT	SYS_P1257	UNUSABLE	
I_L_PT	P200	USABLE	
I_L_PT	P300	USABLE	
I_L_PT	SYS_P1258	USABLE	
I_L_PT	P100	USABLE	
I_G_PT	FULL	VALID	NO
I_G_PARTPT	PARTIAL	VALID	NO

10 rows selected.



# Oracle Partitioningによる索引付け機能の強化

## 部分ローカル索引と部分グローバル索引

パーティション4を除外した部分グローバル索引

```
SQL> explain plan for select count(*) from pt where col2 = 3;
```

Explained.

```
SQL> select * from table(dbms_xplan.display);
```

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		1	22	54 (12)	00:00:01		
1	SORT AGGREGATE		1	22				
2	VIEW	VW_TE_2	2		54 (12)	00:00:01		
3	UNION-ALL							
* 4	TABLE ACCESS BY GLOBAL INDEX ROWID BATCHED	PT	1	26	2 (0)	00:00:01	ROWID	ROWID
* 5	INDEX RANGE SCAN	I_G_PARTPT	1		1 (0)	00:00:01		
6	PARTITION RANGE SINGLE		1	26	52 (12)	00:00:01	4	4
* 7	TABLE ACCESS FULL	PT	1	26	52 (12)	00:00:01	4	4

Predicate Information (identified by operation id):

```
4 - filter("PT"."COL1"<301)
5 - access("COL2"=3)
7 - filter("COL2"=3)
```

# 使用不可索引と部分索引

---

## 使用不可索引

使用不可索引パーティションは、一般に高速ロード要件がある環境で使用される

- データ挿入時の索引メンテナンスの時間を“節約”
- 使用不可索引セグメントは領域を消費しない（11.2） 使用不可索引はオプティマイザによって無視される

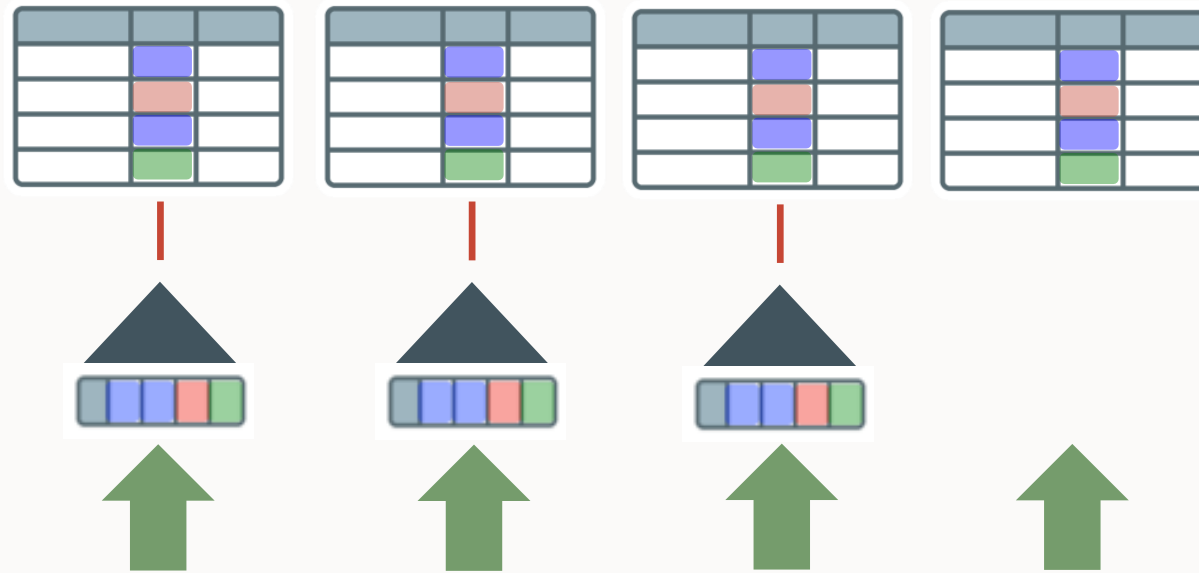
パーティション索引は、一部のパーティションが使用不可でもオプティマイザで使用できる

```
SKIP_UNUSABLE_INDEXES = [TRUE | FALSE]
```

- 11.2より前は、静的プルーニングと、使用可能索引パーティションのアクセスのみが必須
- 11.2では、UNION ALLを使用して問合せのインテリジェント・リライト

# 表OR拡張

複数のSQLブランチが生成および実行される



部分的に使用不可の索引が存在する場合のインテリジェントUNION ALL拡張

- 透過的な内部リライト
- 使用可能な索引パーティションが使用される
- 使用不可索引パーティションの場合のフル・パーティション・アクセス

# 表OR拡張

サンプル計画 - 複数のSQLブランチが生成および実行される

```
select count(*) from toto where name = 'FOO' and rn between 1300 and 1400
```

Plan hash value: 2830852558

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				27M(100)			
1	SORT AGGREGATE		1	21				
2	VIEW	VW_TE_2	2		27M (3)	92:15:22		
3	UNION-ALL							
4	PARTITION RANGE SINGLE		1	20	2 (0)	00:00:01	14	14
5	TABLE ACCESS BY LOCAL INDEX ROWID	TOTO	1	20	2 (0)	00:00:01	14	14
* 6	INDEX RANGE SCAN	I_TOTO	1		1 (0)	00:00:01	14	14
7	PARTITION RANGE SINGLE		1	22	27M (3)	92:15:22	15	15
* 8	TABLE ACCESS FULL	TOTO	1	22	27M (3)	92:15:22	15	15

Predicate Information (identified by operation id):

```
6 - access("NAME"='FOO')
8 - filter(("NAME"='FOO' AND "TOTO"."RN">=1400))
```

27 rows selected.

# パーティション化の拡張機能

---

# インターバル・パーティション化

Oracle 11g Release 1 (11.1) で導入

# インターバル・パーティション化

レンジ・パーティション化の拡張機能

同サイズ・レンジ・パーティションの完全自動化

パーティションはメタデータ情報としてのみ作成される

- 開始パーティションは永続化される

新規データを受け取るとすぐにセグメントが割り当てられる

- 新規パーティションの作成は不要
- ローカル索引も作成およびメンテナンスされる

**パーティション管理は一切不要**



# インターバル・パーティション化

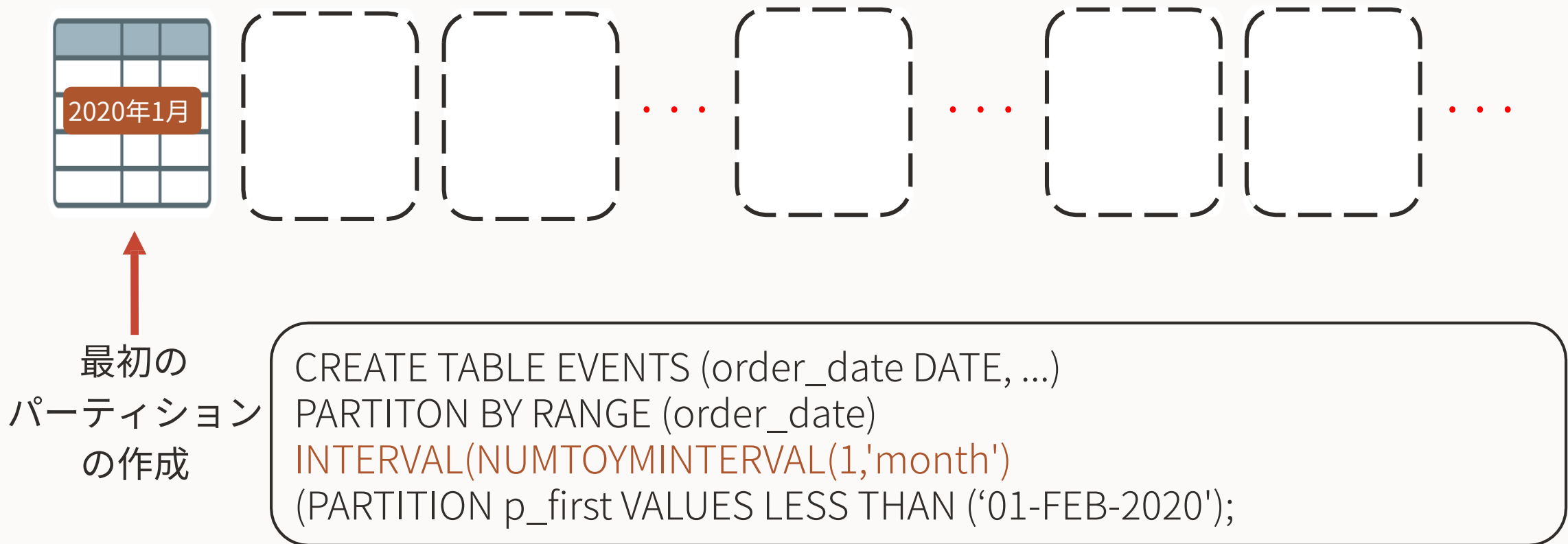


データを受信すると、パーティションを自動的に作成する

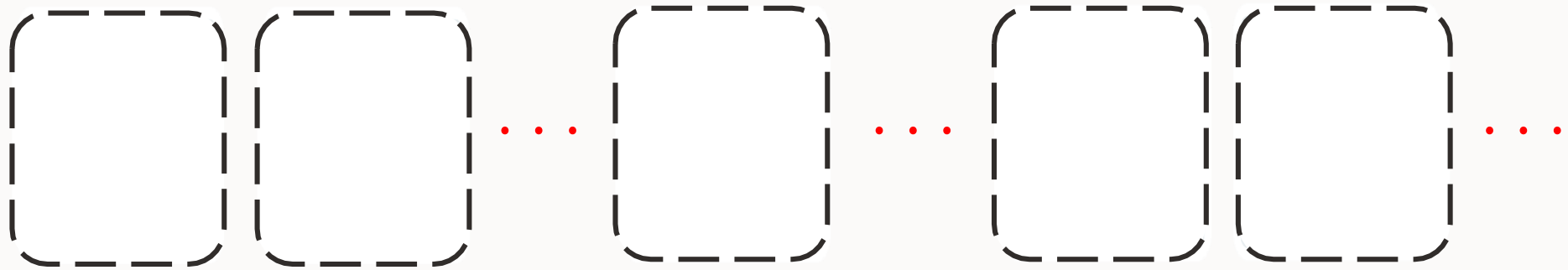
- レンジ・パーティション化の拡張機能

# インターバル・パーティション化

## 非常に簡単



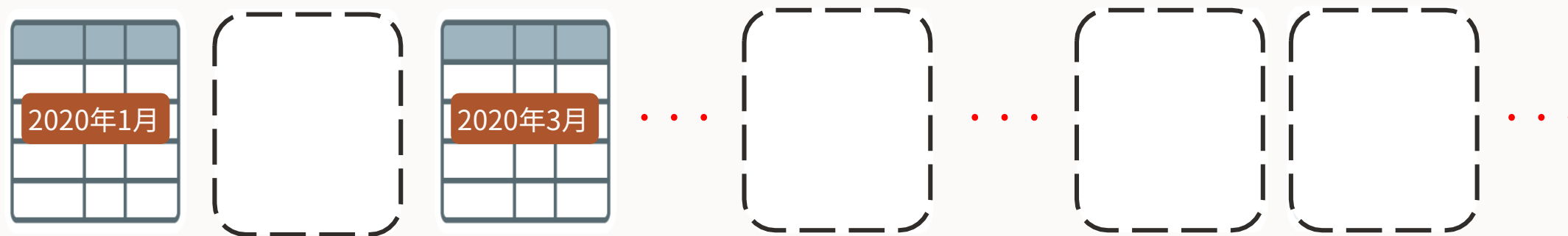
# インターバル・パーティション化 非常に簡単



その他のパーティションは表メタデータにのみ存在

# インターバル・パーティション化

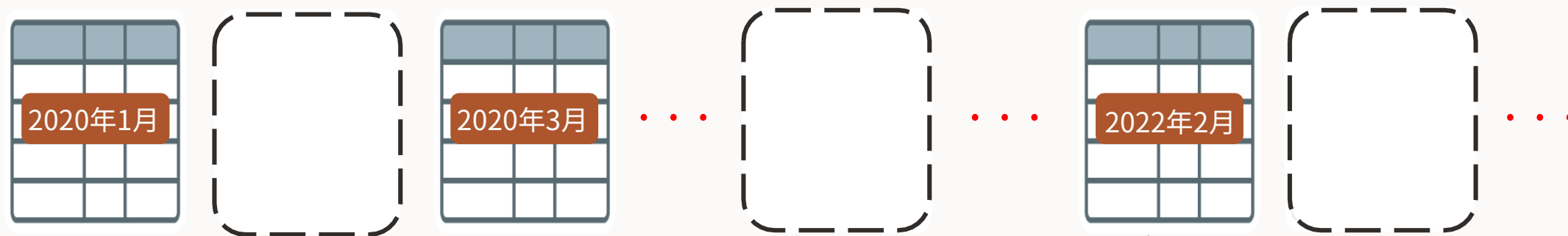
## 非常に簡単



新しいパーティションは自動的にインスタンス化

```
INSERT INTO EVENTS (order_date DATE, ...)  
VALUES ('15-MAR-2020',...);
```

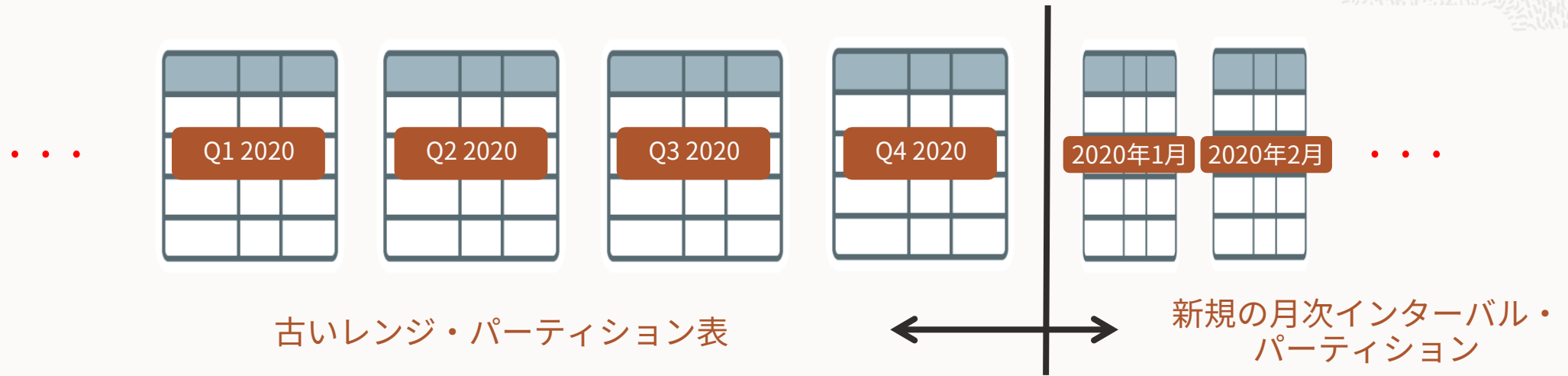
# インターバル・パーティション化 非常に簡単



新しいパーティションデータを受け取った時にはいつでも

```
INSERT INTO EVENTS ( order_date DATE, ...)  
VALUES ('04-FEB-2022',...);
```

# インターバル・パーティション化

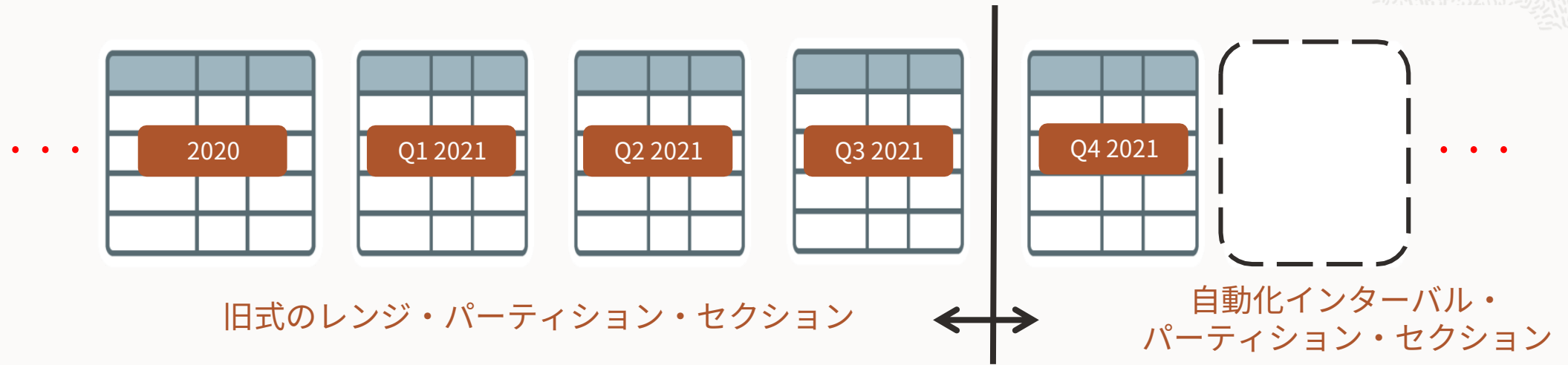


レンジ・パーティション表は、インターバル・パーティション表に拡張可能

- 簡単なメタデータ・コマンド
- 投資保護

```
ALTER TABLE EVENTS  
SET INTERVAL(NUMTOYMINTERVAL(1,'month');
```

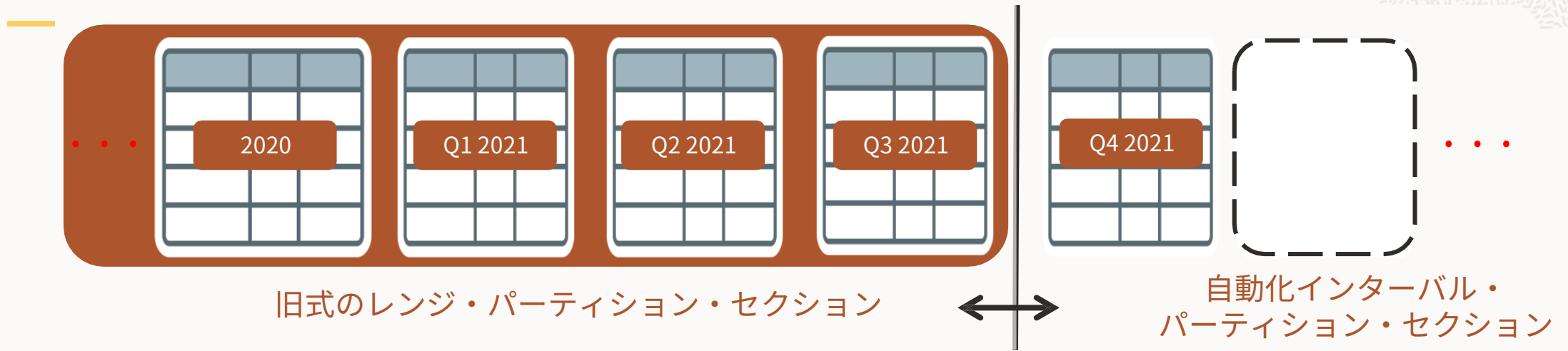
# インターバル・パーティション化



インターバル・パーティション表には、旧式のレンジ・インターバル・セクションと自動化インターバル・セクションがある

- 自動化された新規パーティション管理機能と追加のフル・パーティション・メンテナンス機能："双方の利点を最大活用"

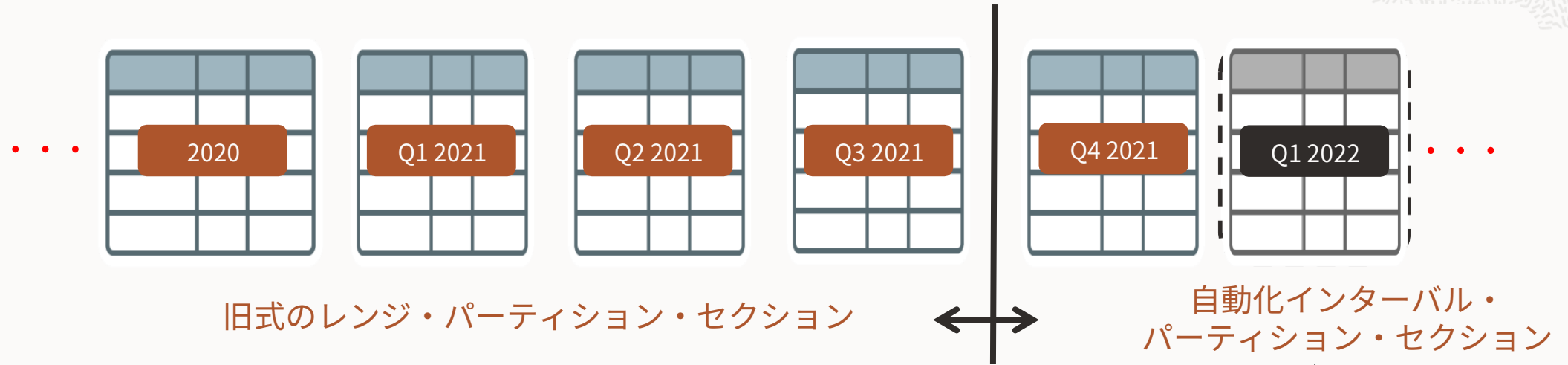
# インターバル・パーティション化



1. ILMの旧パーティションをマージおよび移動する



# インターバル・パーティション化



1. ILMの旧パーティションをマージおよび移動する
2. 新規データを挿入する
  1. 自動パーティション・インスタンス化

# 遅延セグメント作成とインターバル・パーティション化

## インターバル・パーティション化

- パーティションの最大数は100万個に事前定義されている
  - 明示的に定義されたパーティションとインターバルベース・パーティションの合計
- データなしのパーティションにはセグメントが割り当てられない
  - 新規のレコード挿入によりセグメント作成がトリガーされる
- "成長し続ける"表の場合に理想的

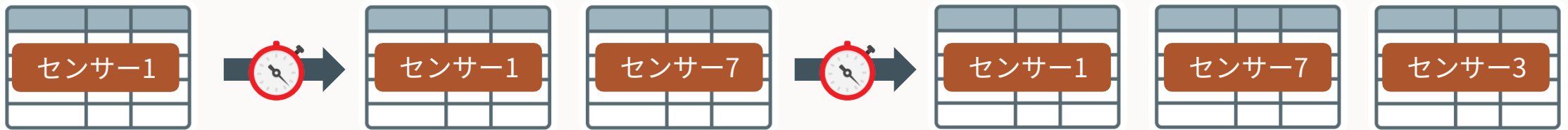
## 遅延セグメント作成による"標準"パーティション化

- 明示的に定義されたパーティションのみが存在する
  - 新しいパーティションはDDLによって追加される
- データなしのパーティションにはセグメントが割り当てられない
  - データが事前定義パーティションに適合する場合は、新規のレコード挿入によりセグメント作成がトリガーされる
- 移入データの少ない事前定義表の場合に理想的

# 自動リスト・パーティション化

Oracle Database 12.2で導入

## 自動リスト・パーティション化



データを受信すると、パーティションを自動的に作成する

- リスト・パーティション化の拡張機能
- すべての個別パーティション・キー値が別々のパーティションに格納される

## 自動リスト戦略の詳細

パーティションあたり1つの値を含む新しいリスト・パーティションを自動的に作成

- 12.2.0.1での最上位パーティション化戦略としてのみ使用可能

DEFAULTパーティションは指定できない

自動的に作成されたパーティションに対してシステム生成されたパーティション名

- 決定的な[サブ]パーティションの識別にFOR VALUES句を使用する

リスト・パーティション化を自動リスト・パーティション化にすることが可能

- DEFAULTパーティションを設けないことが唯一の要件
- スキーマへの投資の保護

# 自動リスト・パーティション表

## 構文例

```
CREATE TABLE EVENTS( sensor_type      VARCHAR2(50),  
                      channel          VARCHAR2(50), ...)  
PARTITION BY LIST (sensor_type) AUTOMATIC  
( partition p1 values ('GYRO'));
```

# 自動リストはリスト+DEFAULTと同等ではない

さまざまなユースケース・シナリオ

DEFAULTパーティション化によるリスト

- 対象は複数のサイズの大きな個別リストの値と"ノイズ"

自動リスト・パーティション化

- パーティション・キー値あたり'レコードのクリティカルマス'があると想定
- リスト+DEFAULTを使用する場合の先行状態として使用可能

# 自動リストはリスト+DEFAULTと同等ではない

さまざまなユースケース・シナリオ

DEFAULTパーティション化によるリスト

- 対象は複数のサイズの大きな個別リストの値と"ノイズ"

自動リスト・パーティション化

- 値ごとに'レコードのクリティカルマス'があると想定
- リスト+DEFAULTを使用する場合の先行状態として使用可能

..また、両者は機能的に競合しており、併用できない

- 新しいパーティション・キー値用に新しいパーティションを取得するか、
- ..または、すべてをまとめたバケットで"ダンプ"するかのいずれか



# バーチャル・カラム・ パーティション化

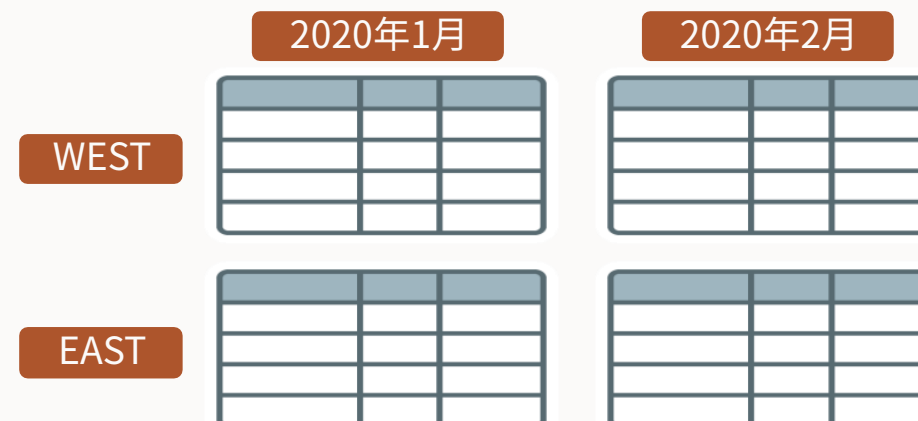
Oracle 11g Release 1 (11.1) で導入

# バーチャル・カラム・パーティション化

## EVENTS

EVENT_ID	EVENT_DATE	SENSOR_ID	...REGION AS (SUBSTR(EVENT_ID,6,2))
9834	GY 14 12-JAN-2020	65920	GY
8300	TH 97 14-FEB-2020	39654	T
3886	TH 02 16-JAN-2020	4529	H
2566	GY 94 19-JAN-2020	15327	GY
3699	GY 63 02-FEB-2020	18733	T
			H

REGIONにストレージは不要  
ORDER\_DATE、REGION別に  
パーティション化



# バーチャル・カラム 例

すべての属性を持つベース表...

```
CREATE TABLE accounts  
(acc_no      number(10)    not null,  
 acc_name    varchar2(50) not null, ...
```

12500	Adams	
12507	Blake	
12666	King	
12875	Smith	

# バーチャル・カラム

## 例

すべての属性を持つベース表...

- ...バーチャル（派生）カラムによって拡張される

```
CREATE TABLE accounts
(acc_no      number(10)      not null,
 acc_name    varchar2(50)    not null, ...
 acc_branch  number(2)       generated always as
              (to_number(substr(to_char(acc_no),1,2))))
```

12500	Adams	12
12507	Blake	12
12666	King	12
12875	Smith	12

# バーチャル・カラム

## 例

すべての属性を持つベース表...

- ...バーチャル（派生）カラムによって拡張される
- ...また、バーチャル・カラムがパーティション化キーとして使用される

```
CREATE TABLE accounts  
(acc_no      number(10)    not null,  
 acc_name    varchar2(50)  not null, ...  
 acc_branch  number(2)     generated always as  
             (to_number(substr(to_char(acc_no),1,2)))  
 partition by list (acc_branch) ...
```

12500	Adams	12
12507	Blake	12
12666	King	12
12875	Smith	12

...

32320	Jones	32
32407	Clark	32
32758	Hurd	32
32980	Kelly	32

# バーチャル・カラム

## パーティション・プルーニング

概念モデルではバーチャル・カラムを可視属性および使用属性とみなす

パーティション・プルーニングは現在、バーチャル・カラム（パーティション・キー）  
自体で条件付きでのみ機能する

- 過渡的条件なし

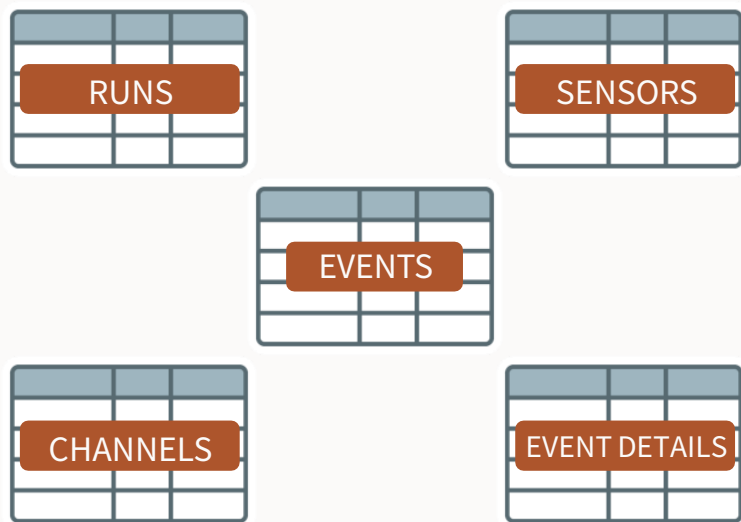
今後のリリース（直近ではない）で機能強化を計画

# 参照パーティション化

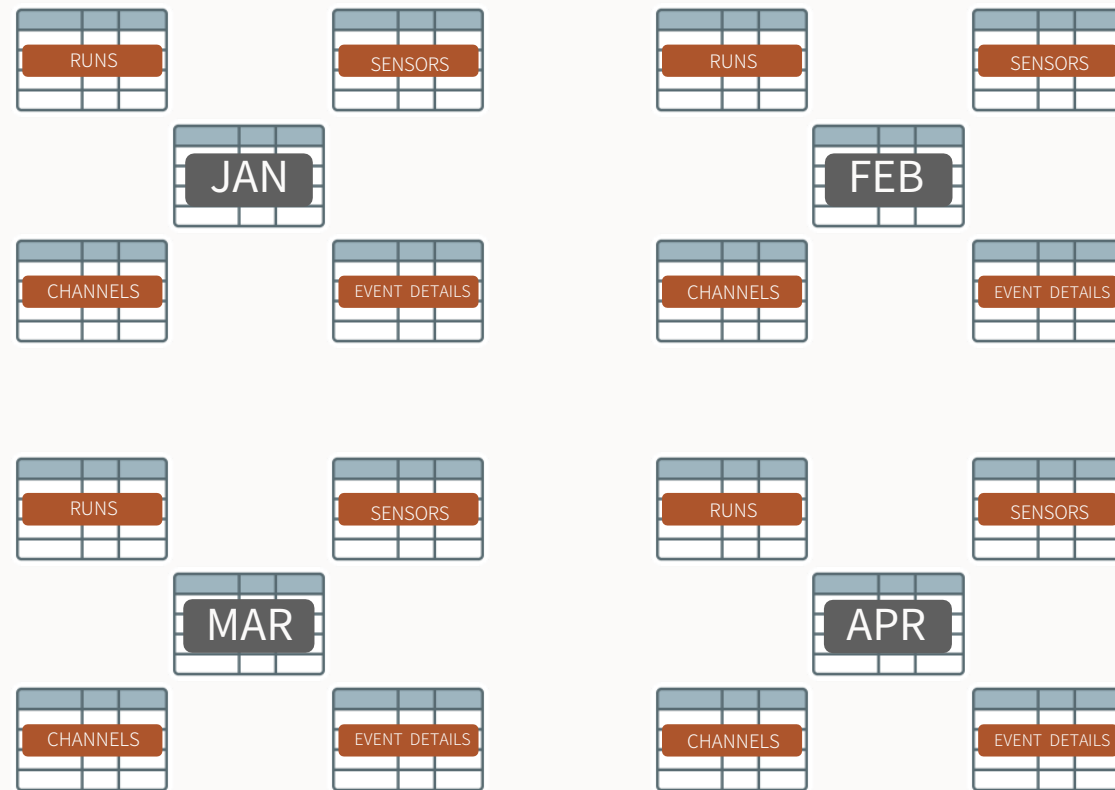
Oracle 11g Release 1 (11.1) で導入

# 参照パーティション化

## パーティション化戦略の継承



パーティション  
EVENTS  
日付別





# 参照パーティション化

## ビジネス上の問題

同じパーティション化戦略の恩恵を受ける関連表

- サンプル3NF注文入力データ・モデル

同じ情報の冗長ストレージにより問題を解決

- データとメンテナンスのオーバーヘッド

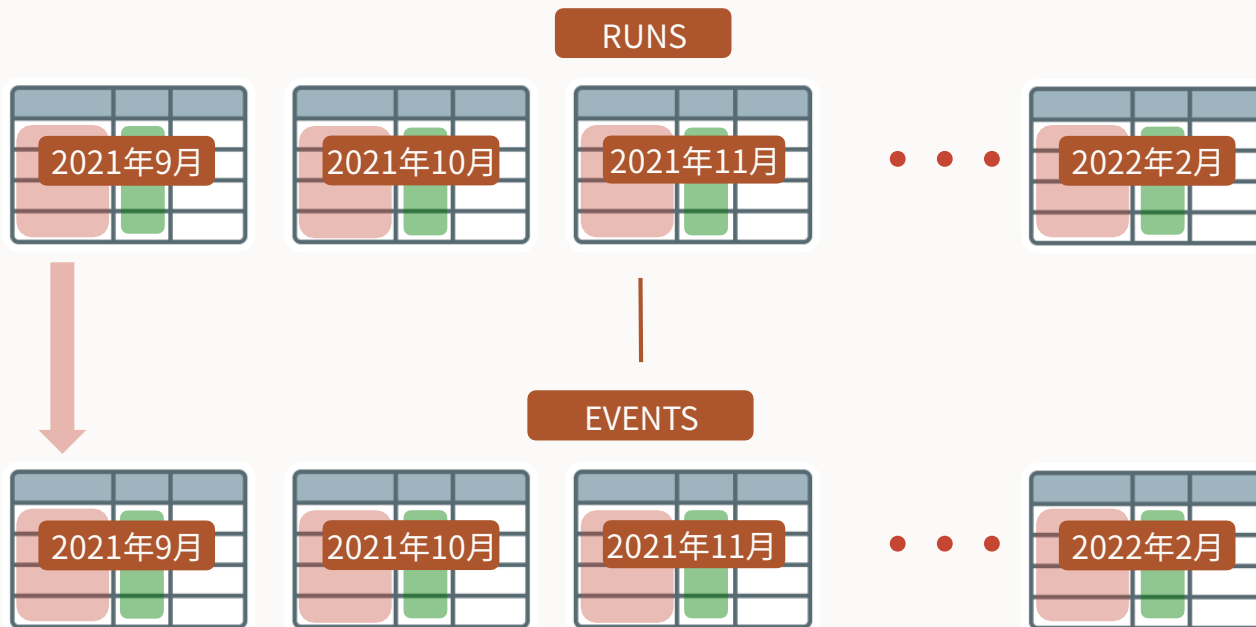
## 解決策

Oracle Database 11gで参照パーティション化を導入

- PK-FKにより、子表が親表のパーティション化戦略を継承
- 直感的なモデリング

パフォーマンスと管理性の向上

# 主キー - 参照パーティション化なしの外部キー

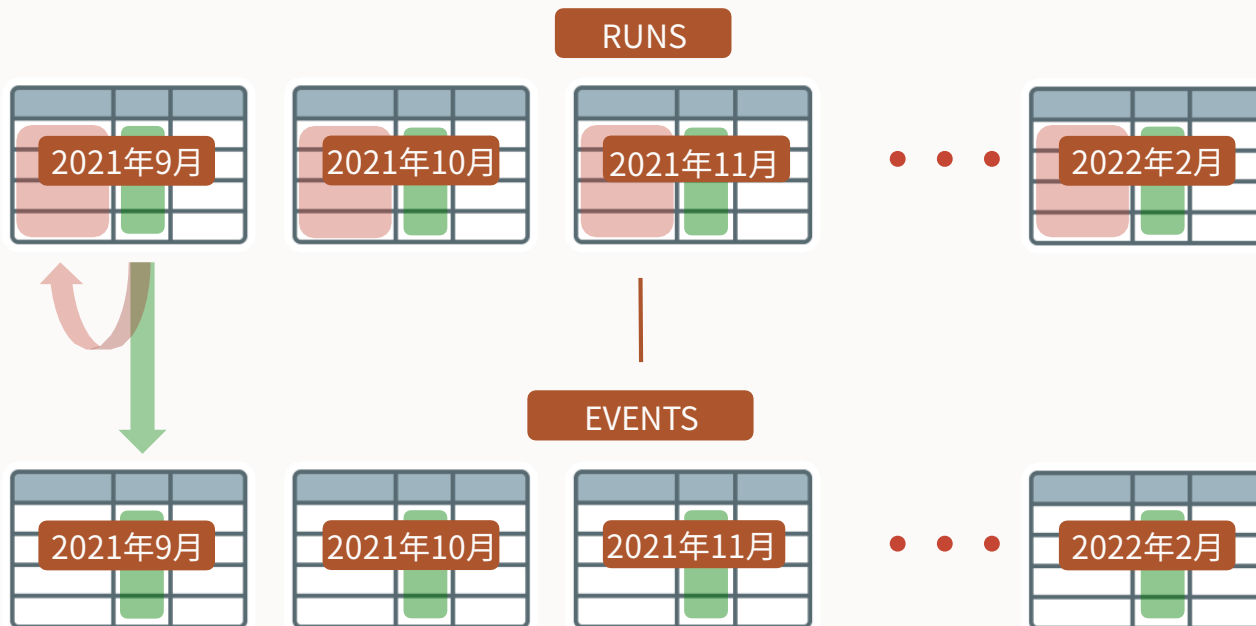


RANGE(run\_date)  
Primary key run\_id

- 冗長ストレージ
- 冗長メンテナンス

RANGE(run\_date)  
Foreign key run\_id

# 主キー - 参照パーティション化ありの外部キー



RANGE(run\_date)  
Primary key run\_id

- PK-FKリレーションシップによって継承したパーティション化キー

RANGE(run\_date)  
Foreign key run\_id

# 参照パーティション化

## ユースケース

---

### 従来のリレーショナル・モデル

- 主キーはすべてのレベルの子に継承され、（伸長した）主キー定義の一部となる

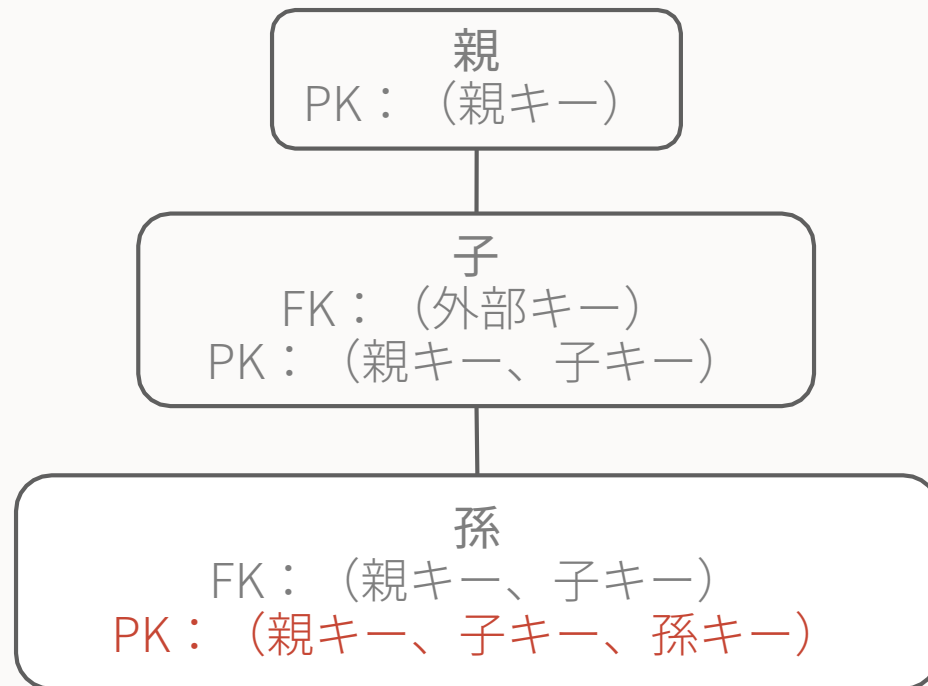
### オブジェクト指向のようなモデル

- 数レベルの主-外部キー・リレーションシップ
- 各レベルの主キーは主キー+”オブジェクトID”

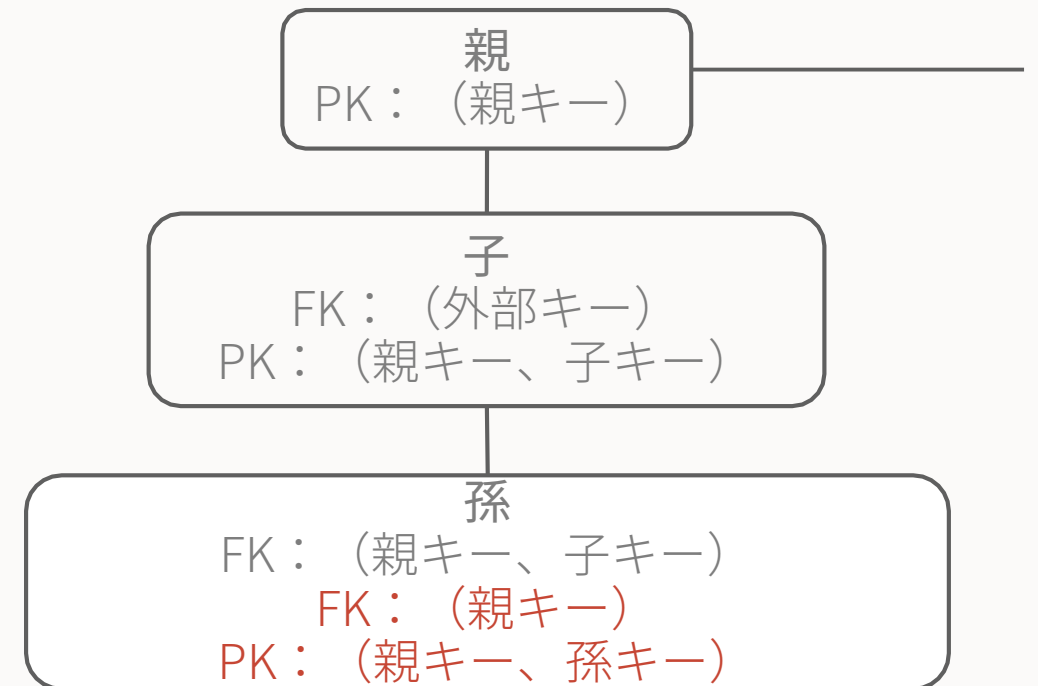
参照パーティション化は両方のモデル化手法に対処するのに最適

# 参照パーティション化

## リレーショナル・モデル



## “オブジェクトのような”モデル



# 参照パーティション化

例

```
create table project (project_id number not null,  
                    project_number varchar2(30),  
                    project_name varchar2(30), ...  
                    constraint proj_pk primary key (project_id))  
  
partition by list (project_id)  
(partition p1 values (1),  
 partition p2 values (2),  
 partition pd values (DEFAULT));
```

```
create table project_customer (project_cust_id number not null,  
                              project_id number not null,  
                              cust_name varchar2(30),  
                              constraint pk_proj_cust primary key  
                                (project_id, project_cust_id),  
                              constraint proj_cust_proj_fk foreign key  
                                (project_id) references project(project_id))  
  
partition by reference (proj_cust_proj_fk);
```

# 参照パーティション化

例（続き）

```
create table proj_cust_address (project_cust_addr_id number not null,  
project_cust_id number not null, project_id  
number not null, cust_address varchar2(30),  
constraint pk_proj_cust_addr primary key  
    (project_id, project_cust_addr_id),  
constraint proj_c_addr_proj_cust_fk foreign key (project_id,  
project_cust_id)  
    references project_customer  
        (project_id, project_cust_id))  
partition by reference (proj_c_addr_proj_cust_fk);
```

# 参照パーティション化

メタデータ・サンプル

## 表情報

```
SQL> SELECT table_name, partitioning_type, ref_ptn_constraint_name
       FROM   user_part_tables
       WHERE  table_name IN ('PROJECT','PROJECT_CUSTOMER','PROJ_CUST_ADDRESS');
```

TABLE_NAME	PARTITION	REF_PTN_CONSTRAINT_NAME
PROJECT	LIST	
PROJECT_CUSTOMER	REFERENCE	PROJ_CUST_PROJ_FK
Pro*Ada	REFERENCE	PROJ_C_ADDR_PROJ_FK

## パーティション情報

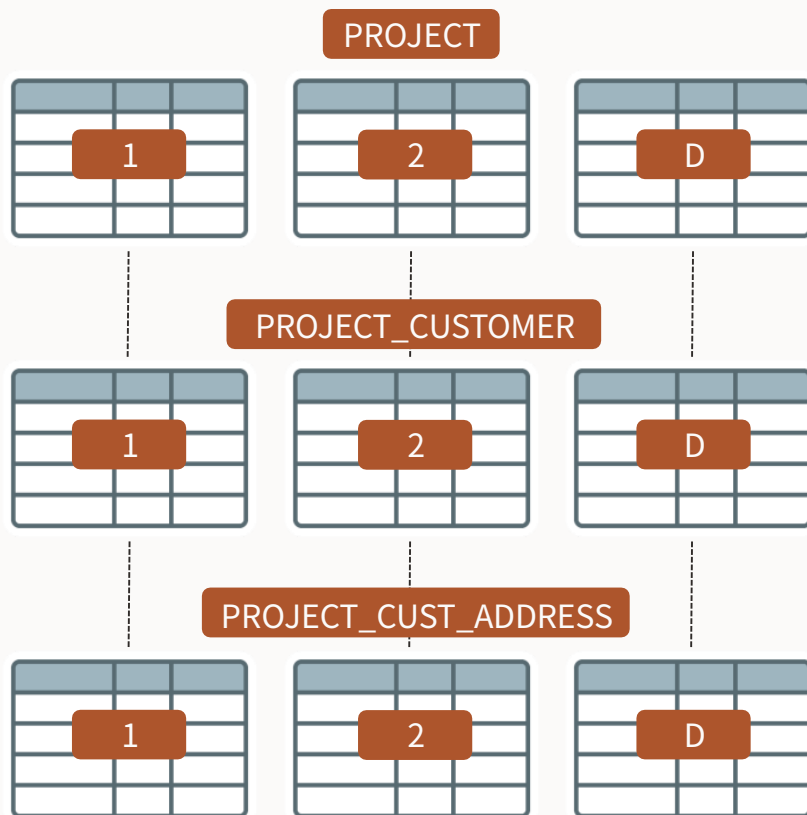
```
SQL> SELECT table_name, partition_name, high_value
       FROM   user_tab_partitions
       WHERE  table_name in ('PROJECT','PROJECT_CUSTOMER')
       ORDER BY table_name, partition_position;
```

TABLE_NAME	PARTITION_NAME	HIGH_VALUE
PROJECT	P1	1
PROJECT	P2	2
PROJECT	PD	DEFAULT
PROJECT_CUSTOMER	P1	
PROJECT_CUSTOMER	P2	
PROJECT_CUSTOMER	PD	



# 参照パーティション化

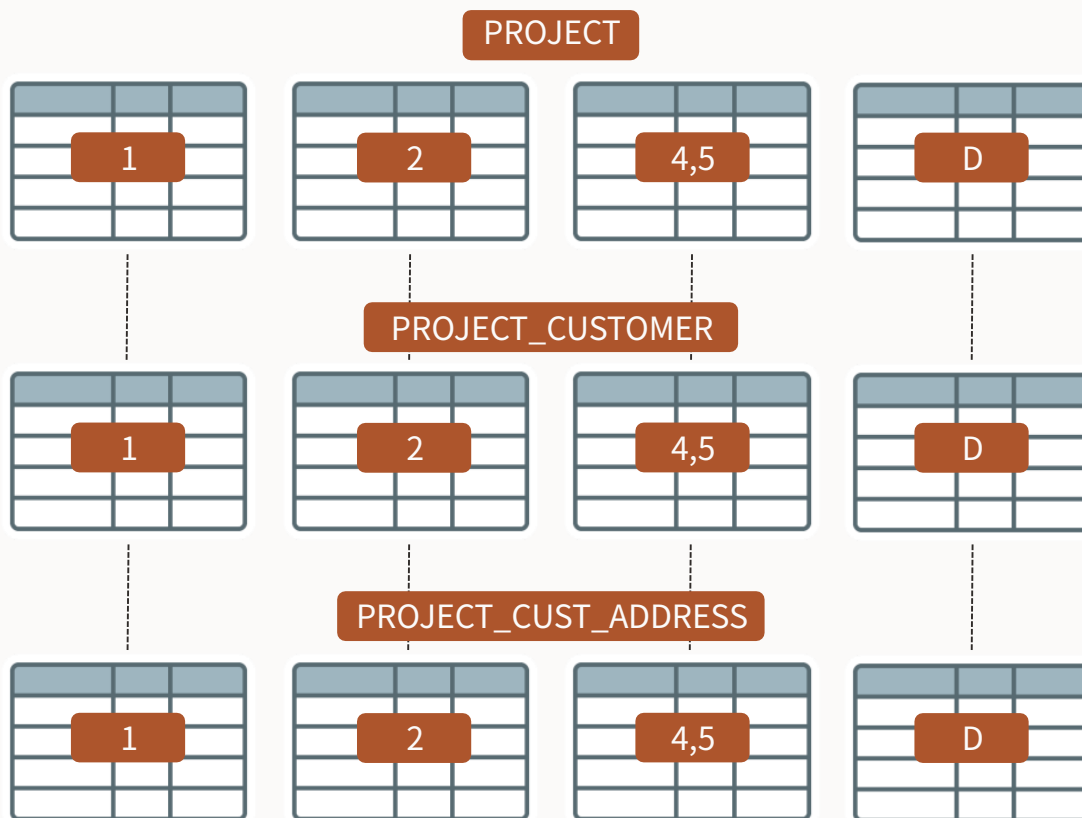
## パーティション・メンテナンス



```
ALTER TABLE project  
SPLIT PARTITION pd VALUES (4,5)  
INTO  
(PARTITION pd, PARTITION p45);
```

# 参照パーティション化

## パーティション・メンテナンス



```
ALTER TABLE project  
SPLIT PARTITION pd VALUES (4,5) INTO  
(PARTITION pd, PARTITION p45);
```

PROJECTパーティションPDは分割される

- "Default"と(4,5)

PROJECT\_CUSTOMERはその依存パーティションを分割する

- PROJECTの同等の親レコードとコロケーション
- (4,5)の親レコードは(4,5)の子レコードを意味する

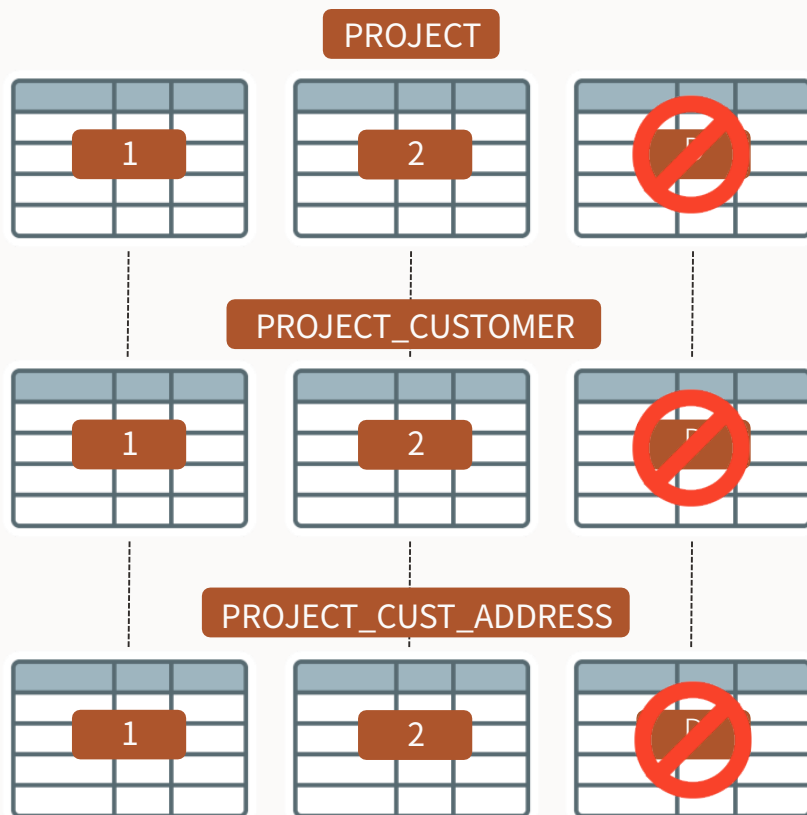
PROJECT\_CUST\_ADDRESSはその依存パーティションを分割する

- PROJECT\_CUSTOMERの同等の親レコードとコロケーション

両方の配置で1レベルのルックアップが必要

# 参照パーティション化

## パーティション・メンテナンス



```
ALTER TABLE project_cust_address  
DROP PARTITION pd;
```

PROJECTパーティションPDは削除される

- PK-FKは違反されないことが保証される

PROJECT\_CUSTOMERはその依存パーティションを削除する

PROJECT\_CUST\_ADDRESSはその依存パーティションを削除する

"通常の"パーティション表と異なり、PK-FKリレーションシップは有効のままになる

- PK-FKリレーションシップのPKでパーティションを自由裁量で削除または切り捨てできない

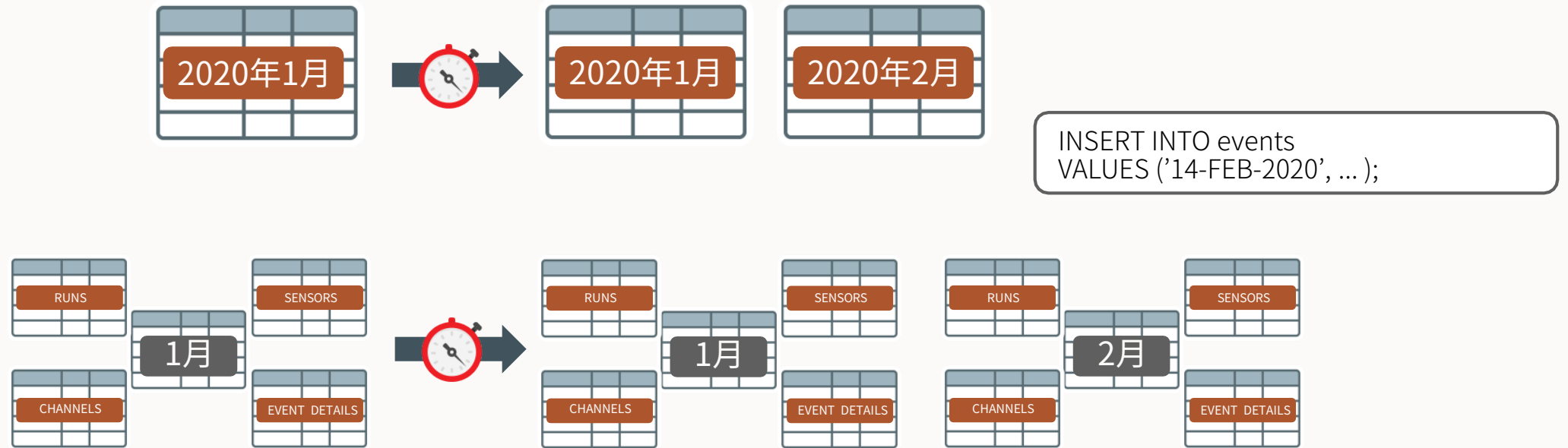
TRUNCATEの場合にも当てはまる

- ボトムアップ操作

# インターバル参照パーティション化

Oracle 12c Release 1 (12.1) で導入

# インターバル-参照パーティション化



新しいデータを受け取ると新しいパーティションが自動的に作成される

すべての子表は自動的にメンテナンスされる

ビジネスのモデル化を改善するため、2つの有効なパーティション化戦略を組み合わせ

## インターバル-参照パーティション化

```
SQL> REM create some interval-referenced tables ..
SQL> create table intRef_p (pkcol number not null, col2 varchar2(200),
 2          constraint pk_intref primary key (pkcol))
 3 partition by range (pkcol) interval (10)
 4 (partition p1 values less than (10));

Table created.

SQL>
SQL> create table intRef_c1 (pkcol number not null, col2 varchar2(200), fkcol number not null,
 2          constraint pk_c1 primary key (pkcol),
 3          constraint fk_c1 foreign key (fkcol) references intRef_p(pkcol) ON DELETE CASCADE)
 4 partition by reference (fk_c1);

Table created.

SQL>
SQL> create table intRef_c2 (pkcol number primary key not null, col2 varchar2(200), fkcol number not null,
 2          constraint fk_c2 foreign key (fkcol) references intRef_p(pkcol) ON DELETE CASCADE)
 3 partition by reference (fk_c2);

Table created.
```

# インターバル-参照パーティション化

新しいパーティションはデータを受け取った場合にのみ作成される

- 参照ツリー全体に対する自動パーティション・インスタンス化は実行されない
- 移入データの少ない参照パーティション表向けに最適化

パーティション名は既存のパーティションから継承

- 直接的な関連パーティションから名前を継承
- 親パーティションがp100の場合、子パーティションもp100となる
- 親パーティションがp100で、子パーティションがc100の場合、孫パーティションはc100となる

# 複数列リスト・パーティション化

Oracle Database 12.2で導入



## 複数列リスト・パーティション化



データは複数値のリスト（複数列）に編成される

- 個々のパーティションには複数値のセットを格納できる
- DEFAULTパーティションの機能（未指定値をまとめて取込み）

個別値タプル（sensor\_type、channel、...など）のセグメント化に理想的

## 複数列リスト戦略の詳細

複数の列をパーティション化キーとして指定できる

- 最大16個のパーティション・キー列
- パーティション・キーの各セットは一意にする必要がある

1つのDEFAULTパーティションの通知

機能面のサポート

- パーティション戦略とサブパーティション戦略の両方としてサポートされる
- ヒープ表をサポートする
- 外部表をサポートする
- 参照パーティション化と自動リストでサポートされる

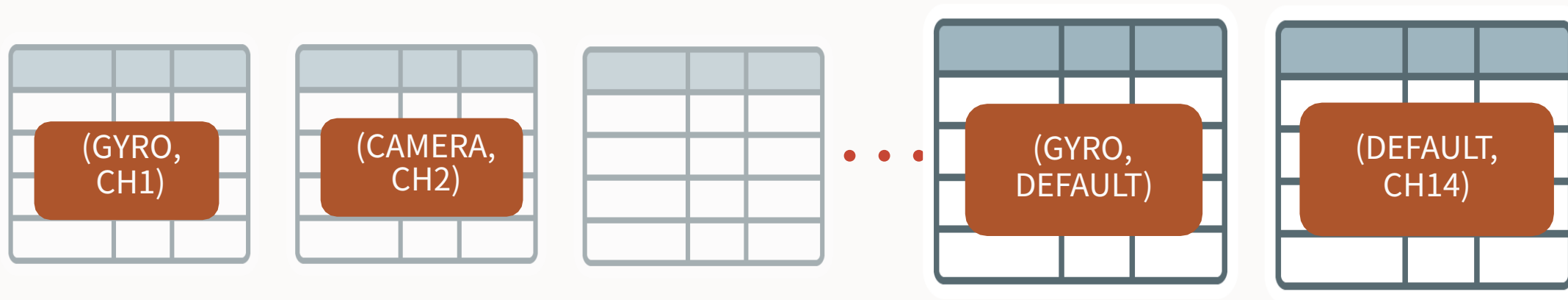
# 複数列リスト・パーティション表

## 構文例

```
CREATE TABLE EVENTS( sensor_type          VARCHAR2(50),
                      channel              VARCHAR2(50), ...)
PARTITION BY LIST (sensor_type, channel)
( partition p1 values ('GYRO','CH1'),
  partition p2 values ('GYRO','CH2'),
  partition p3 values ('CAMERA','CH4'),
  ...
  partition p44 values (('THERMO','CH8'),
                      ('THERMO','CH14')),
  partition p45 values (DEFAULT)
);
```

# 複数列リスト・パーティション化

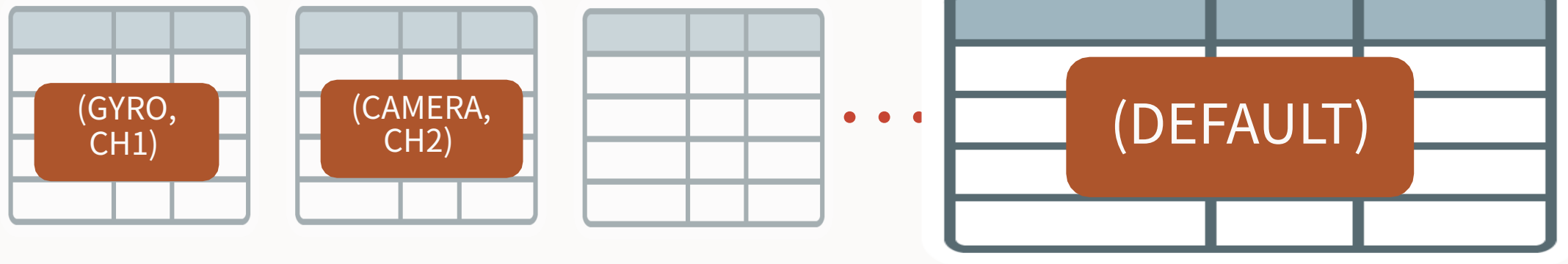
列あたりDEFAULTが1つある場合はどうするか。



(GYRO, CH14)はどこに保存するか?

# 複数列リスト・パーティション化

列あたりDEFAULTが1つある場合はどうするか。



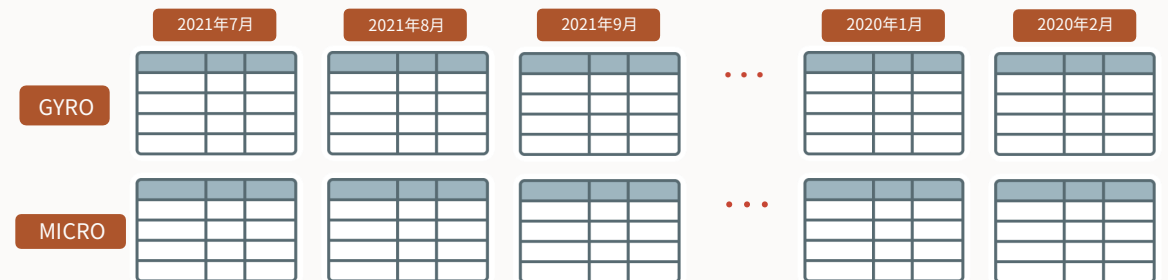
(GYRO, CH12)はどこに保存するか?

- 唯一のDEFAULTパーティション内

## 12.2より前の複数列リスト・パーティション化

### リスト - リスト・パーティション化

- ほぼ同等
- キーは2列のみ (2レベル)
- 概念上は対称



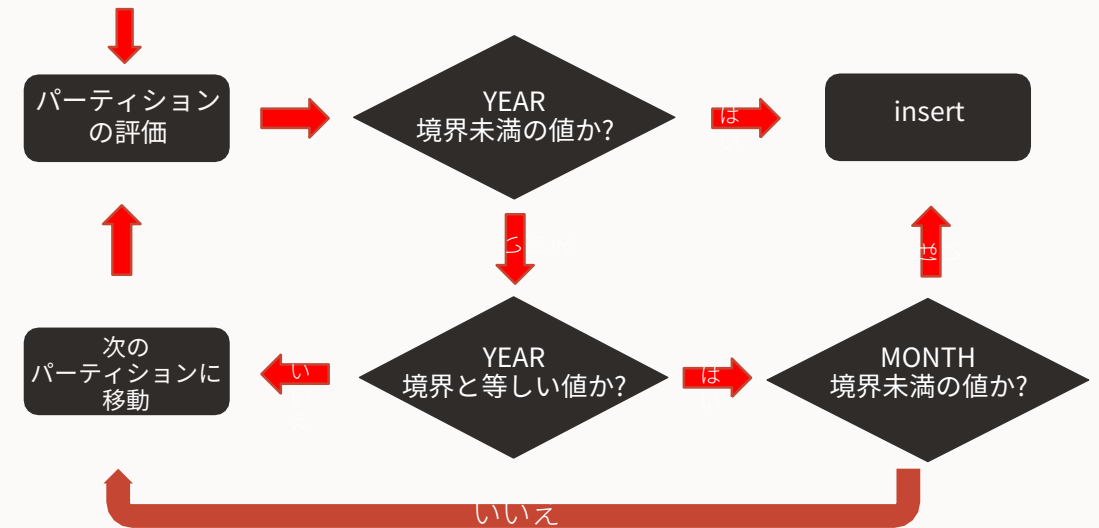
## 12.2より前の複数列リスト・パーティション化

### リスト - リスト・パーティション化

- ほぼ同等
- キーは2列のみ (2レベル)
- 概念上は対称

### 複数列レンジ・パーティション化

- 同等ではない
- 曖昧性が失われた場合にのみ条件を階層的に評価

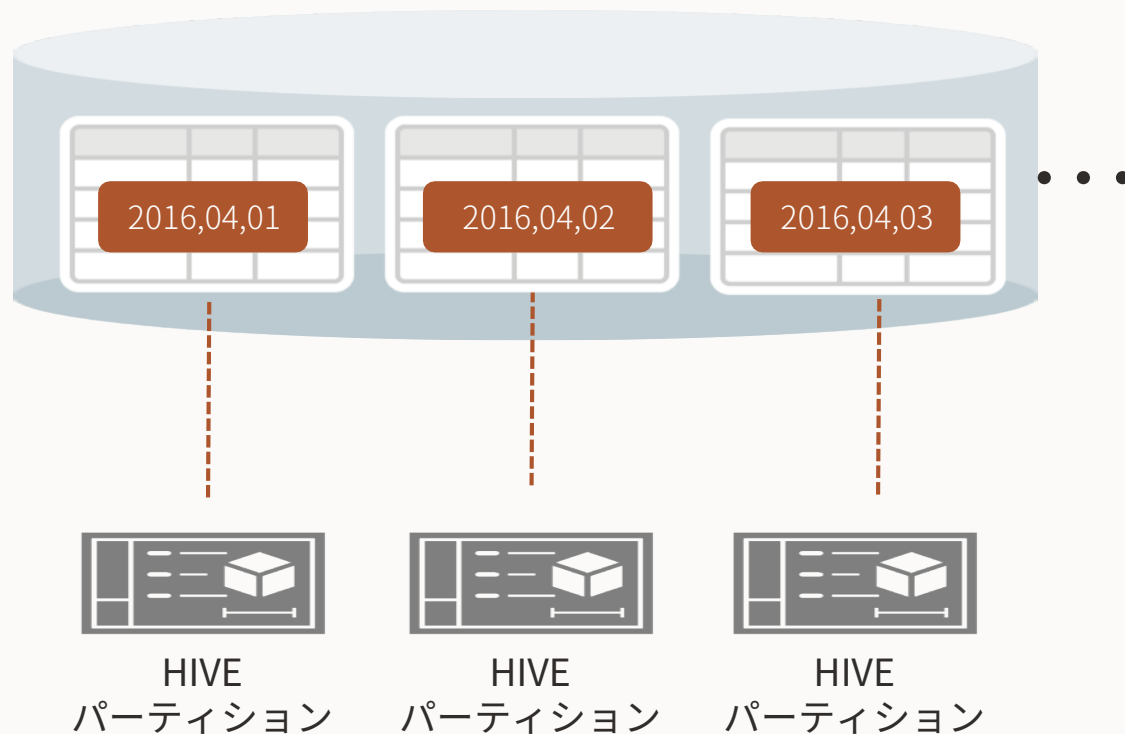


# パーティション化と外部データ

---



# パーティション外部表



すべてのデータがデータベース外

- ファイル・システム内のファイル
- パーティション化されたHive表とHDFS表

Oracle Partitioningの機能で外部データを処理する

- パーティション・プルーニング
- パーティション・メンテナンス

桁違いに高速の問合せを有効化

パフォーマンスと強化されたデータ・メンテナンス

# パーティション外部表

初期作成

```
CREATE TABLE orders ( order_id number,  
                        order_date DATE,      ... )  
ORGANIZATION EXTERNAL  
(   TYPE oracle_loader DEFAULT DIRECTORY data_dir  
    ACCESS PARAMETERS (.)  
) REJECT LIMIT unlimited  
PARTITION BY RANGE(order_date)  
(  partition q1_2015 values less than ('2014-10-01')  
   DEFAULT DIRECTORY old_data_dir LOCATION ('q1_2015.csv'),  
   partition q2_2015 values less than ('2015-01-01')  
   LOCATION ('q2_2015.csv'),  
   partition q3_2015 values less than ('2015-04-01')  
   LOCATION ('q3_2015.csv'),  
   partition q4_2015 values less than ('2015-07-01')  
);
```

# パーティション外部表

初期作成

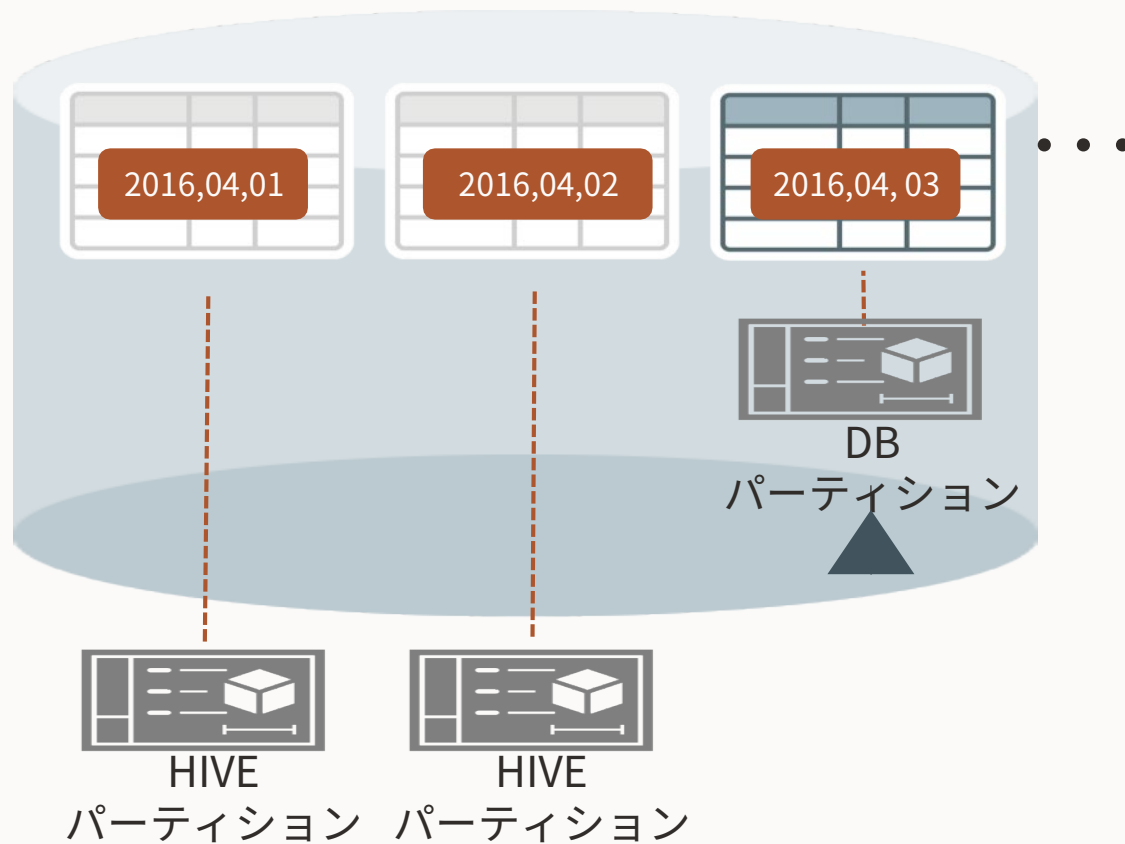
```
CREATE TABLE orders ( order_id number,  
                        order_date DATE,      ... )  
ORGANIZATION EXTERNAL  
(   TYPE oracle_loader DEFAULT DIRECTORY data_dir  
    ACCESS PARAMETERS (..) )  
REJECT LIMIT unlimited  
PARTITION BY RANGE(order_date)  
(  partition q1_2015 values less than ('2014-10-01')  
   DEFAULT DIRECTORY old_data_dir LOCATION ('q1_2015.csv'),  
   partition q2_2015 values less than ('2015-01-01')  
   LOCATION ('q2_2015.csv'),  
   partition q3_2015 values less than ('2015-04-01')  
   LOCATION ('q3_2015.csv'),  
   partition q4_2015 values less than ('2015-07-01')  
);
```

# パーティション外部表

初期作成

```
CREATE TABLE orders ( order_id number,  
                        order_date DATE,      ... )  
ORGANIZATION EXTERNAL  
(   TYPE oracle_loader DEFAULT DIRECTORY data_dir  
    ACCESS PARAMETERS (.)  
) REJECT LIMIT unlimited  
PARTITION BY RANGE(order_date)  
(  partition q1_2015 values less than ('2014-10-01')  
   DEFAULT DIRECTORY old_data_dir LOCATION ('q1_2015.csv'),  
   partition q2_2015 values less than ('2015-01-01')  
   LOCATION ('q2_2015.csv'),  
   partition q3_2015 values less than ('2015-04-01')  
   LOCATION ('q3_2015.csv'),  
   partition q4_2015 values less than ('2015-07-01')  
);
```

# ハイブリッド・パーティション表



1つの表に内部（RDBMS）と外部の両方のパーティションを格納する

- 部分索引付け、部分読取り専用、制約、マテリアライズド・ビューなどのフル機能をサポートする

最適化されたハイブリッド処理

- RDBMSと外部の両方の処理機能をフル活用

情報ライフサイクル管理のためのパーティション・メンテナンス

- 現在の制限ありサポート
- 進行中の拡張機能

# ハイブリッド・パーティション表

## 初期作成

```
CREATE TABLE orders ( order_id number,  
                        order_date DATE,      ... )  
EXTERNAL PARTITION ATTRIBUTES  
( TYPE oracle_loader DEFAULT DIRECTORY data_dir ACCESS  
  PARAMETERS (..)REJECT LIMIT unlimited  
)  
PARTITION BY RANGE(order_date)  
( partition q1_2015 values less than ('2014-10-01')  
  EXTERNAL LOCATION ('order_q1_2015.csv'),  
  partition q2_2015 values less than ('2015-01-01'), partition  
  q3_2015 values less than ('2015-04-01'), partition q4_2015 values  
  less than ('2015-07-01')  
);
```

# ハイブリッド・パーティション表

初期作成

```
CREATE TABLE orders ( order_id number,  
                        order_date DATE,      ... )  
EXTERNAL PARTITION ATTRIBUTES  
( TYPE oracle_loader DEFAULT DIRECTORY data_dir ACCESS  
  PARAMETERS (..)REJECT LIMIT unlimited  
)  
PARTITION BY RANGE(order_date)  
( partition q1_2015 values less than ('2014-10-01')  
  EXTERNAL LOCATION ('order_q1_2015.csv'),  
  partition q2_2015 values less than ('2015-01-01'), partition  
  q3_2015 values less than ('2015-04-01'), partition q4_2015 values  
  less than ('2015-07-01')  
);
```

# ハイブリッド・パーティション表

初期作成

```
CREATE TABLE orders ( order_id number,  
                        order_date DATE,      ... )  
EXTERNAL PARTITION ATTRIBUTES  
( TYPE oracle_loader DEFAULT DIRECTORY data_dir ACCESS  
  PARAMETERS (..)REJECT LIMIT unlimited  
)  
PARTITION BY RANGE(order_date)  
( partition q1_2015 values less than ('2014-10-01')  
  EXTERNAL LOCATION ('order_q1_2015.csv'),  
  partition q2_2015 values less than ('2015-01-01'), partition  
  q3_2015 values less than ('2015-04-01'), partition q4_2015 values  
  less than ('2015-07-01')  
);
```



## ハイブリッド・パーティション表への進化

```
ALTER TABLE orders
ADD EXTERNAL PARTITION ATTRIBUTES
(
  TYPE oracle_loader DEFAULT
  DIRECTORY data_dir ACCESS
  PARAMETERS
    (records delimited by newline badfile
    'cdxt_%a_%p.bad' logfile
    'cdxt_%a_%p.log'
    fields terminated by ','
    missing field values are null
    )
  REJECT LIMIT unlimited
);
```

# ハイブリッド・パーティション表

## ライフサイクル管理のサポート

EXCHANGEによる外部と内部のストレージ間のライフサイクル管理の初期サポート

- MOVEまたは他の高度な機能（SPLIT、MERGE）なし
- カスタマー/アプリケーションによって実施されるデータ移動

現在、外部と内部のストレージ間のライフサイクル管理はサポートされていない

- 機能はOracle Database 19c、Release 19.7に組み込まれる
  - 内部パーティションを外部表と交換する（バグ28876926）
  - 外部パーティションを内部表と交換する（バグ30172925）

# オブジェクト・ストアのデータへのアクセス

任意のオブジェクト・ストア内のデータにアクセス可能

- Oracle Object Store、AWS S3、またはAzure

明示的認証または認証前URI

(明らかに特定のパーティション化機能ではないが、それでも優れる)



任意のオブジェクト・ストレージ

## ファイル・システム・アクセスとオブジェクト・ストレージ

```
CREATE TABLE orders ( order_id number,  
                        order_date DATE,      ... )  
ORGANIZATION EXTERNAL  
(   TYPE oracle_loader DEFAULT DIRECTORY data_dir  
    ACCESS PARAMETERS ( )  
) REJECT LIMIT unlimited  
PARTITION BY RANGE(order_date)  
(  partition q1_2015 values less than ('2014-10-01')  
   LOCATION ('q1_2015.csv'),  
   partition q2_2015 values less than ('2015-01-01')  
   LOCATION ('q2_2015.csv'),  
   partition q3_2015 values less than ('2015-04-01')  
   LOCATION ('q3_2015.csv'),  
   partition q4_2015 values less than ('2015-07-01')  
);
```

## ファイル・システム・アクセスとオブジェクト・ストレージ

```
CREATE TABLE orders ( order_id number,  
                      order_date DATE,      ... )  
ORGANIZATION EXTERNAL  
(   TYPE oracle_loader DEFAULT DIRECTORY data_dir  
    ACCESS PARAMETERS (  
      CREDENTIAL 'OSS_ACCESS' )  
) REJECT LIMIT unlimited  
PARTITION BY RANGE(order_date)  
(   partition q1_2015 values less than ('2014-10-01')  
    LOCATION ('https://swiftobjectstorage.us-ashburn-1 ...'),  
    partition q2_2015 values less than ('2015-01-01')  
    LOCATION ('...'),  
    partition q3_2015 values less than ('2015-04-01')  
    LOCATION ('...'),  
    partition q4_2015 values less than ('2015-07-01')  
);
```

# データ配置検証

---

内部パーティション化によりデータが適切に配置される

- ここでも1つの例外がある

外部パーティション化はファイル内のデータがパーティションに正しくマッピングされることに依存する

# データ配置検証

内部パーティション化によりデータが適切に配置される

- ここでも1つの例外がある

外部パーティション化はファイル内のデータがパーティションに正しくマッピングされることに依存する

パーティション外部表によって追加される新機能によってデータ配置を検証する

- `ORA_PARTITION_VALIDATION(rowid)`
- データ配置が正しい場合には1を、正しくない場合には0を返す

## データ配置検証

```
SQL> SELECT hpto.*,  
           ORA_PARTITION_VALIDATION(rowid) AS correct_partition  
FROM hpto;
```

DEPTNO	DNAME	LOC	CORRECT_PARTITION
12	dept_12	xp1_15	1
16	dept_16	dept_loc_16	1
17	dept_17	dept_loc_17	1
29	dept_29	xp2_30	1
31	dept_31	dept_loc_31	1
32	dept_32	dept_loc_32	1
9999	dept_50	xp_wrong	0



# パフォーマンス向上のための パーティション化

# パフォーマンス向上のためのパーティション化

パーティション化はパフォーマンスを改善するために透過的に活用される

## パーティション・プルーニング

- パーティション化メタデータを使用して関心対象のパーティションにのみアクセスする

## パーティション・ワイズ結合

- 等間隔パーティション表を最小限のリソース消費で結合する
- Oracle RAC環境のコロケーション機能进行处理する

## パーティション交換ロード

- メタデータ操作によって新しいデータを"ロード"する

# パフォーマンス向上のためのパーティション化

## パーティション・プルーニング

5月1日から2日のEVENTの合計はいくつか?



### EVENTS

<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>										5月5日
<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>										5月4日
<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>										5月3日
<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>										5月2日
<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>										5月1日
<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>										4月30日
<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>										4月29日
<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>										4月28日
<table border="1"><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr><tr><td></td><td></td><td></td></tr></table>										4月27日

### パーティション・エリミネーション

- ストレージから取り込まれるデータ量を劇的に減少させる
- 関連するパーティションでのみ操作を実行する
- 問合せパフォーマンスを透過的に向上させ、リソース使用量を最適化する

# パーティション・プルーニング

単純および複雑なSQL文の場合に有効

すべてのアプリケーションに対して透過的

2種類のプルーニング

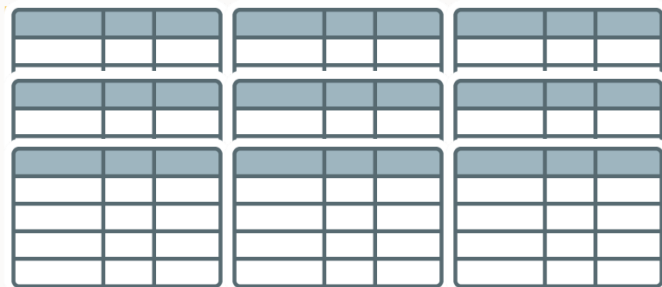
- コンパイル時の静的プルーニング
- 実行時の動的プルーニング

Oracle Exadata Storage Serverを補完

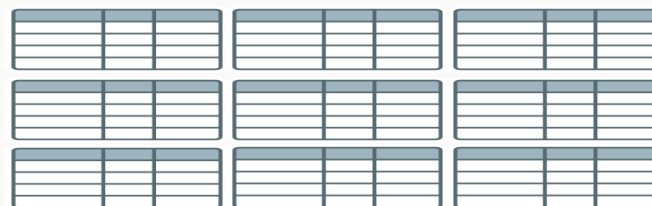
- パーティション化では、パーティション・エリミネーションによって論理的にプルーニングする
- Oracle Exadataでは、ストレージ索引によって物理的にプルーニングする
  - フィルタ処理と投影によってさらにデータを削減

# 利点を倍増するパフォーマンスの特長

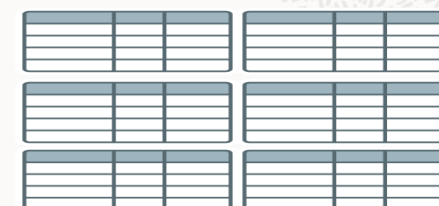
例



ユーザー・データ100 TB



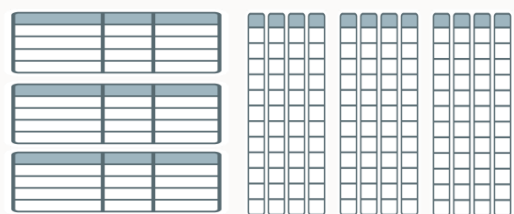
ユーザー・データ10 TB



ユーザー・データ2TB

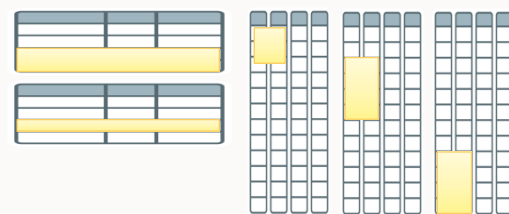
10倍圧縮の場合

パーティション・プルーニングの場合



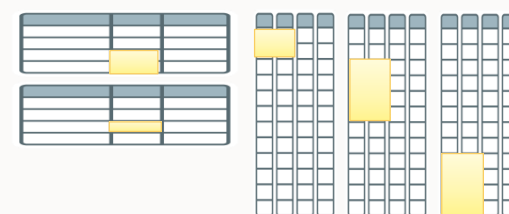
ユーザー・データ2 TB

ディスク上に1 TB、  
メモリ内に1 TB



ユーザー・データ100 GB

ストレージ索引と  
ゾーン・マップあり



ユーザー・データ30 GB

Smart Scanあり



1秒未満でのスキャン

索引なし



## 静的パーティション・プルーニング

```
SELECT avg( luminosity ) FROM EVENTS  
WHERE times_id  
BETWEEN '01-MAR-2021' and '31-MAY-2021';
```

2021年1月

2021年2月

2021年3月

2021年4月

2021年5月

2021年6月

関連パーティションはコンパイル時に判明する

- 計画のPSTART/PSTOP列で実際の値を探す

オプティマイザにはSQL文のもっとも正確な情報がある

# 静的プルーニング

## サンプル計画

```
SELECT avg( luminosity )  
FROM atlas.EVENTS s, atlas.times t  
WHERE s.time_id = t.time_id  
AND s.time_id between TO_DATE('01-JAN-2021', 'DD-MON-YYYY')  
and TO_DATE('01-JAN-2020', 'DD-MON-YYYY')
```

Plan hash value: 2025449199

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				3 (100)			
1	SORT AGGREGATE		1	12				
2	PARTITION RANGE ITERATOR		313	3756	3 (0)	00:00:01	9	13
* 3	TABLE ACCESS FULL	EVENTS	313	3756	3 (0)	00:00:01	9	13

Predicate Information (identified by operation id):

```
3 - filter("S"."TIME_ID" <= TO_DATE(' 2020-01-01 00:00:00', 'syyy-mm-dd hh24:mi:ss'))
```

22 rows selected.

# 静的プルーニング

## サンプル計画

```
SELECT avg( luminosity )
FROM atlas.EVENTS s, atlas.times t
WHERE s.time_id = t.time_id
AND s.time_id between TO_DATE('01-JAN-2021', 'DD-MON-YYYY')
and TO_DATE('01-JAN-2020', 'DD-MON-YYYY')
```

Plan hash value: 2025449199

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				3 (100)			
1	SORT AGGREGATE		1	12				
2	PARTITION RANGE ITERATOR		313	3756	3 (0)	00:00:01	9	13
* 3	TABLE ACCESS FULL	EVENTS	313	3756	3 (0)	00:00:01	9	13

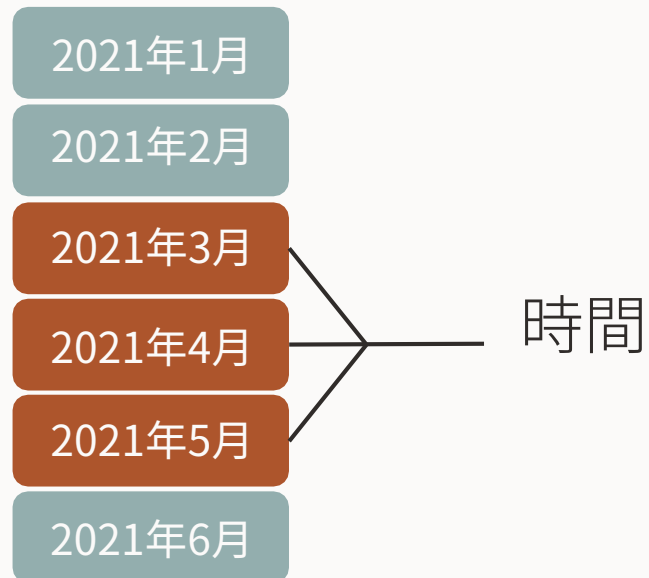
Predicate Information (identified by operation id):

```
3 - filter("S"."TIME_ID" <= TO_DATE(' 2020-01-01 00:00:00', 'yyyy-mm-dd hh24:mi:ss'))
```

22 rows selected.



## 動的パーティション・プルーニング



```
SELECT avg( luminosity ) FROM  
EVENTS s, times t  
WHERE t.time_id = s.time_id  
AND t.calendar_month_desc IN  
('MAR-2021', 'APR-2021', 'MAY-2021');
```

複雑な問合せのための高度なプルーニングの仕組み  
関連パーティションは実行時に決定される

- 計画のPSTART/PSTOP列でワード'KEY'を探す

# 動的パーティション・プルーニング

## サンプル計画 - ネステッド・ループ

```
SELECT avg( luminosity )  
FROM atlas.EVENTS s, atlas.times t  
WHERE s.time_id = t.time_id  
AND t.calendar_month_desc in ('MAR-2021', 'APR-2021', 'MAY-2021')
```

Plan hash value: 2025449199

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				13 (100)			
1	SORT AGGREGATE		1	28				
2	NESTED LOOP		2	56	13 (0)	00:00:01		
* 3	TABLE ACCESS FULL	TIMES	2	32	13 (8)	00:00:01		
4	PARTITION RANGE ITERATOR		2	24	0 (0)		KEY	KEY
* 5	TABLE ACCESS FULL	EVENTS	2	24	0 (0)		KEY	KEY

Predicate Information (identified by operation id):

3 - filter(("T"."CALENDAR\_MONTH\_DESC"='MAR-2021' OR "T"."CALENDAR\_MONTH\_DESC"='APR-2021'  
OR "T"."CALENDAR\_MONTH\_DESC"='MAY-2021'))

5 - filter("T"."TIME\_ID"="S"."TIME\_ID")

26 rows selected.

# 動的パーティション・プルーニング

## サンプル計画 - ネステッド・ループ

```
SELECT avg( luminosity )
FROM atlas.EVENTS s, atlas.times t
WHERE s.time_id = t.time_id
AND t.calendar_month_desc in ('MAR-2021', 'APR-2021', 'MAY-2021')
```

Plan hash value: 2025449199

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				13 (100)			
1	SORT AGGREGATE		1	28				
2	NESTED LOOP		2	56	13 (0)	00:00:01		
* 3	TABLE ACCESS FULL	TIMES	2	32	13 (8)	00:00:01		
4	PARTITION RANGE ITERATOR		2	24	0 (0)		KEY	KEY
* 5	TABLE ACCESS FULL	EVENTS	2	24	0 (0)		KEY	KEY

Predicate Information (identified by operation id):

3 - filter(("T"."CALENDAR\_MONTH\_DESC"='MAR-2021' OR "T"."CALENDAR\_MONTH\_DESC"='APR-2021'  
OR "T"."CALENDAR\_MONTH\_DESC"='MAY-2021'))

5 - filter("T"."TIME\_ID"="S"."TIME\_ID")

26 rows selected.

# 動的パーティション・プルーニング

## サンプル計画 - 副問合せプルーニング

```
SELECT avg( luminosity )
FROM atlas.EVENTS s, atlas.times t
WHERE s.time_id = t.time_id
AND t.calendar_month_desc in ('MAR-2021', 'APR-2021', 'MAY-2021')
```

Plan hash value: 2025449199

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				2000K(100)			
1	SORT AGGREGATE		1	28				
* 2	HASH JOIN		24M	646M	2000K(100)	06:40:01		
* 3	TABLE ACCESS FULL	TIMES	2	32	43 (8)	00:00:01		
4	PARTITION RANGE SUBQUERY		10G	111G	1166K(100)	03:53:21	KEY(SQ)	KEY(SQ)
5	TABLE ACCESS FULL	EVENTS	10G	111G	1166K(100)	03:53:21	KEY(SQ)	KEY(SQ)

Predicate Information (identified by operation id):

2 - access("S"."TIME\_ID"="T"."TIME\_ID")

3 - filter(("T"."CALENDAR\_MONTH\_DESC"='MAR-2021' OR "T"."CALENDAR\_MONTH\_DESC"='APR-2021'  
OR "T"."CALENDAR\_MONTH\_DESC"='MAY-2021'))

26 rows selected.

# 動的パーティション・プルーニング

## サンプル計画 - ブルーム・フィルタ・プルーニング

```
SELECT avg( luminosity )  
FROM atlas.EVENTS s, atlas.times t  
WHERE s.time_id = t.time_id  
AND t.calendar_month_desc in ('MAR-2021', 'APR-2021', 'MAY-2021')
```

Plan hash value: 2025449199

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				19 (100)			
1	SORT AGGREGATE		1	28				
* 2	HASH JOIN		2	56	19 (100)	00:00:01		
3	PART JOIN FILTER CREATE	:BF0000	2	32	13 (8)	00:00:01		
* 4	TABLE ACCESS FULL	TIMES	2	32	13 (8)	00:00:01		
5	PARTITION RANGE JOIN-FILTER		960	11520	5 (0)	00:00:01	:BF0000	:BF0000
6	TABLE ACCESS FULL	EVENTS	960	11520	5 (0)	00:00:01	:BF0000	:BF0000

Predicate Information (identified by operation id):

2 - access("S"."TIME\_ID"="T"."TIME\_ID")

4 - filter(("T"."CALENDAR\_MONTH\_DESC"='MAR-2021' OR "T"."CALENDAR\_MONTH\_DESC"='APR-2021'  
OR "T"."CALENDAR\_MONTH\_DESC"='MAY-2021'))26 rows selected.

27 rows selected.

## “AND”プルーニング

```
FROM events s, times t ...  
WHERE s.time_id = t.time_id .  
AND t.fiscal_year in (2021,2020)  
AND s.time_id  
  between TO_DATE('01-JAN-2021','DD-MON-YYYY')  
  and TO_DATE('01-JAN-2022','DD-MON-YYYY')
```

動的プルーニング

静的プルーニング

パーティション・キーのすべての条件がプルーニングで使用される

- 動的条件と静的条件を組み合わせるようになった

例：

- FACT表とディメンションの両方でプルーニング条件を指定してスター変換

# “AND”プルーニング

## サンプル計画

Plan hash value:552669211

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT		1	24	17 (12)	00:00:01		
1	SORT AGGREGATE		1	24				
* 2	HASH JOIN		204	4896	17 (12)	00:00:01		
3	PART JOIN FILTER CREATE	:BF0000	185	2220	13 (8)	00:00:01		
* 4	TABLE ACCESS FULL	TIMES	185	2220	13 (8)	00:00:01		
5	PARTITION RANGE AND		313	3756	3 (0)	00:00:01	KEY (AP)	KEY (AP)
* 6	TABLE ACCESS FULL	EVENTS	313	3756	3 (0)	00:00:01	KEY (AP)	KEY (AP)

Predicate Information (identified by operation id):

2 – access(“S”.TIME\_ID=“T”.TIME\_ID)

4 – filter(“T”.TIME\_ID<=TO\_DATE(‘ 2020-01-01 00:00:00’, ‘yyyy-mm-dd hh24:mi:ss’) AND  
 (“T”.FISCAL\_YEAR=2021 OR “T”.FISCAL\_YEAR=2020) AND “T”.TIME\_ID>=TO\_DATE(‘ 2021-01-01  
00:00:00’, ‘yyyy-mm-dd hh24:mi:ss’))

6 – filter(“S”.TIME\_ID<=TO\_DATE(‘ 2020-01-01 00:00:00’, ‘yyyy-mm-dd hh24:mi:ss’))

22 rows selected.

# 確実なパーティション・プルーニング

パーティション・キーのフィルタ条件で関数を使用しない

```
SELECT avg(luminosity)
FROM atlas.EVENTS s, atlas.times t
WHERE s.time_id = t.time_id
AND TO_CHAR(s.time_id, 'YYYYMMDD') between '20210101' and '20220101'
```

Plan hash value: 672559287

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				6 (100)			
1	SORT AGGREGATE		1	12				
2	PARTITION RANGE ALL		2	24	6 (17)	00:00:01	1	16
* 3	TABLE ACCESS FULL	EVENTS	2	24	6 (17)	00:00:01	1	16

Predicate Information (identified by operation id):

```
3 - filter((TO_CHAR(INTERNAL_FUNCTION("S"."TIME_ID"), 'YYYYMMDD') >= '20210101' AND
            TO_CHAR(INTERNAL_FUNCTION("S"."TIME_ID"), 'YYYYMMDD') <= '20220101'))
```

23 rows selected.



# 確実なパーティション・プルーニング パーティション・キーのフィルタ条件で関数を使用しない

```
SELECT avg( luminosity )
FROM atlas.EVENTS s, atlas.times t
WHERE s.time_id = t.time_id
AND TO_CHAR(s.time_id, 'YYYYMMDD') between '20210101' and '20220101'
```



```
SELECT avg( luminosity )
FROM atlas.EVENTS s, atlas.times t
WHERE s.time_id = t.time_id
AND s.time_id between TO_DATE('20210101','YYYYMMDD') and TO_DATE('20220101','YYYYMMDD')
```



Plan hash value: 2025449199

Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time	Pstart	Pstop
0	SELECT STATEMENT				3 (100)			
1	SORT AGGREGATE		1	12				
2	PARTITION RANGE ITERATOR		313	3756	3 (0)	00:00:01	9	13
* 3	TABLE ACCESS FULL	EVENTS	313	3756	3 (0)	00:00:01	9	13

Pstart	Pstop
1	16
1	16

Predicate Information (identified by operation id):

```
3 - filter("S"."TIME_ID"<=TO_DATE(' 2020-01-01 00:00:00', 'syyy-mm-dd hh24:mi:ss'))
```

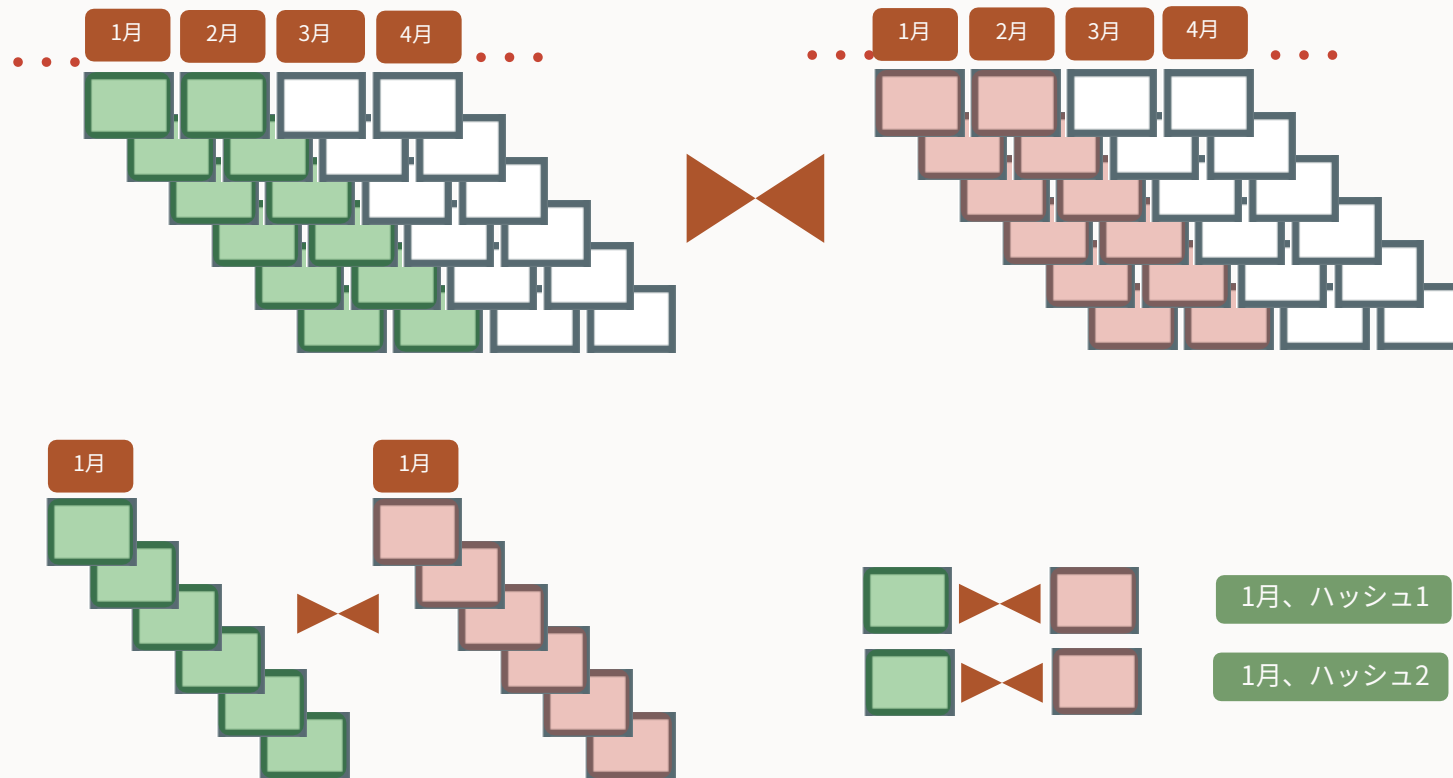
22 rows selected.

101' AND  
01'))



# パーティション・ワイズ結合

パーティション・プルーニングとPWJの"実際"

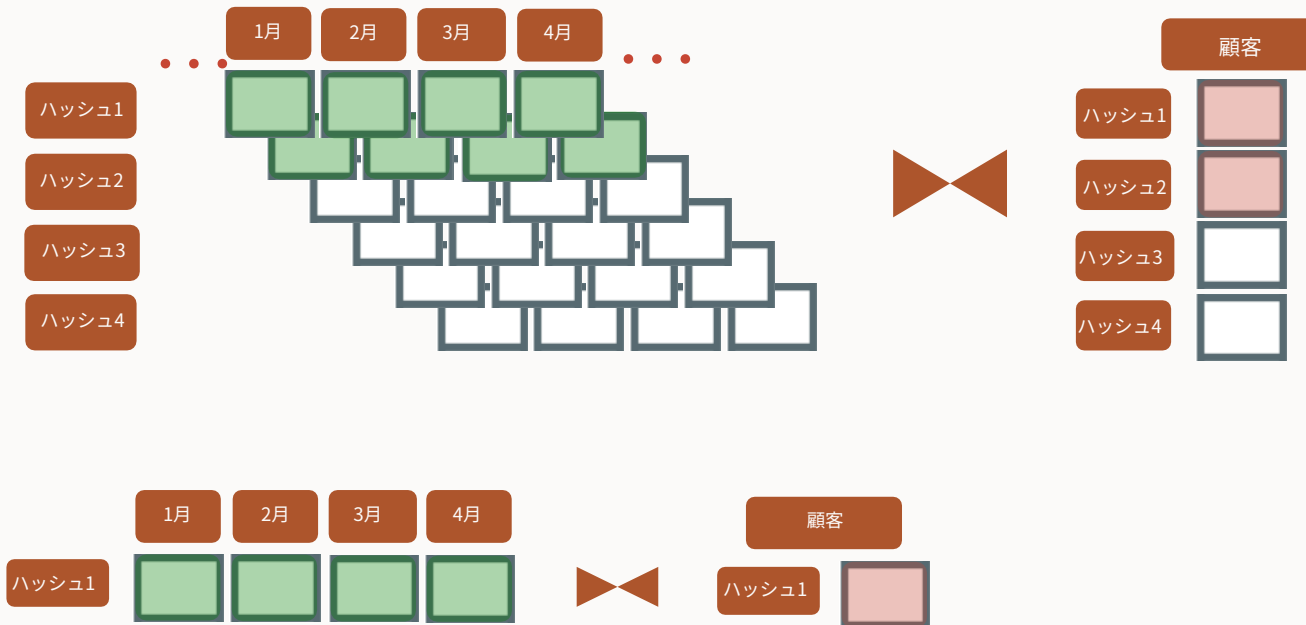


大規模結合は複数の小規模結合に分割され、並列実行される

- 結合するパーティションの数は、DOPの倍数であることが必須
- 両方の表は、結合列で同じ方法でパーティション化することが必要

# パーティション・ワイズ結合

## パーティション・プルーニングとPWJの"実際"



大規模結合は複数の小規模結合に分割され、並列実行される

- 結合するパーティションの数は、DOPの倍数であることが必須
- 両方の表は、結合列で同じ方法でパーティション化することが必要

## パーティション・ページとロード

メタデータ限定操作としてデータを削除および追加する

- パーティションのメタデータを交換する

スタンドアロン表を任意の単一パーティションと交換する

- データ・ロード：スタンドアロン表にロード中の新規データが格納される
- データ・ページ：データが格納されているパーティションが空の表と交換される

ページの代替としてパーティションを削除する

- データは完全に消去される

"空"



### EVENTS表

2021年5月18日

2021年5月19日

2021年5月20日

2021年5月21日

2021年5月22日

2021年5月23日

2021年5月24日

# パーティション化のメンテナンス

---

# パーティション・メンテナンス

## 成功のための基本概念

パフォーマンスがもっとも可視化しやすいが、その他の領域も忘れてはならない

- パーティション化では、パフォーマンス、管理性、可用性のすべてのビジネス関連領域に対処しなければならない

パーティションの自律性が不可欠

- すべてのパーティション・メンテナンス操作の基本的な要件
- パーティションをデータ・ディクショナリ内のメタデータとして認識する

# パーティション・メンテナンス

## 成功のための基本概念

完全なパーティション自律性を実現する

- 可能な場合にはローカル索引を使用する
- パーティションで、パーティションの全表レベル操作（TRUNCATE、MOVE、COMPRESSなど）を使用可能にする

データベース管理のためにパーティションを可視化できて使用可能にする

- 使用しやすくするためのパーティション名前付け

メンテナンス操作はパーティション認識でなければならない

- 索引の場合にも当てはまる

メンテナンス操作はパーティション表のオンライン使用と干渉してはならない

# データ管理の側面

パーティション・メンテナンス操作で対処可能

## データの高速入力

- EXCHANGE
- パーティション単位のダイレクト・パス・ロード

## データの高速削除

- DROP、TRUNCATE、EXCHANGE

## データの高速再編成

- MOVE、SPLIT、MERGE



# パーティション・メンテナンス

## 表のパーティション・メンテナンス操作

ALTER TABLE ADD PARTITION(S)  
ALTER TABLE DROP PARTITION(S)  
ALTER TABLE EXCHANGE PARTITION  
ALTER TABLE MODIFY PARTITION  
    [PARALLEL][ONLINE]  
ALTER TABLE MOVE PARTITION [PARALLEL][ONLINE]  
ALTER TABLE RENAME PARTITION  
ALTER TABLE SPLIT PARTITION [PARALLEL][ONLINE]  
ALTER TABLE MERGE PARTITION(S) [PARALLEL]  
    [ONLINE]  
ALTER TABLE COALESCE PARTITION [PARALLEL]  
ALTER TABLE ANALYZE PARTITION  
ALTER TABLE TRUNCATE PARTITION(S)  
Export/Import [by partition]  
Transportable tablespace [by partition]

## 索引メンテナンス操作

ALTER INDEX MODIFY PARTITION  
ALTER INDEX DROP PARTITION(S)  
ALTER INDEX REBUILD PARTITION  
ALTER INDEX RENAME PARTITION  
ALTER INDEX RENAME  
ALTER INDEX SPLIT PARTITION  
ALTER INDEX ANALYZE PARTITION

すべてのパーティションがいつでも  
使用可能な状態を続ける

ONLINE操作でのDMLロックなし

OFFLINEモードで影響を受ける  
パーティションでのDMLロック

# 複数パーティションでの パーティション・メンテナンス

Oracle 12c Release 1 (12.1) で導入

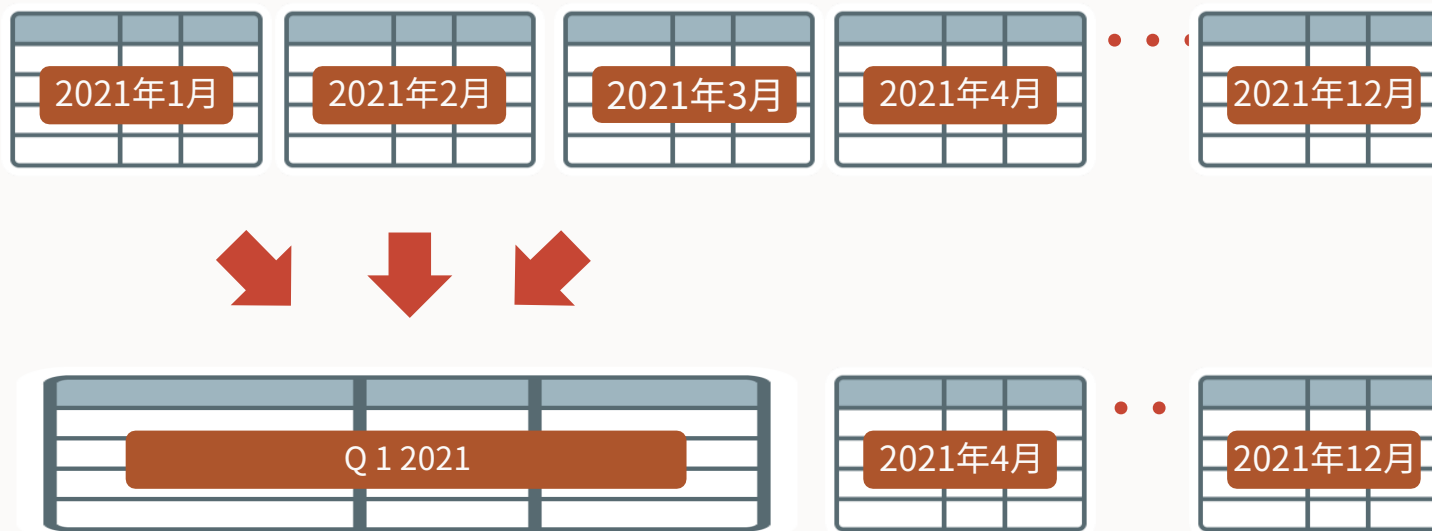
# 強化されたパーティション・メンテナンス操作

複数パーティションで操作する

1回の操作により、複数パーティションでのパーティション・メンテナンス

完全並列処理

ローカル索引とグローバル索引の透過的メンテナンス



```
ALTER TABLE events
MERGE PARTITIONS Jan2021, Feb2021, Mar2021
INTO PARTITION Q1_2021 COMPRESS FOR ARCHIVE HIGH;
```

## 強化されたパーティション・メンテナンス操作

複数パーティションで操作する

複数のパーティションを順に指定する

```
SQL > alter table pt merge partitions part05, part15, part25 into partition p30;  
Table altered.
```

パーティションのレンジを指定する

```
SQL > alter table pt merge partitions part10 to part30  
      into partition part30; Table  
altered.
```

```
SQL > alter table pt split partition p30 into  
2   (partition p10 values less than (10),  
3   partition p20 values less than (20),  
4   partition p30);  
Table altered.
```

すべてのPMOPSで有効に機能する

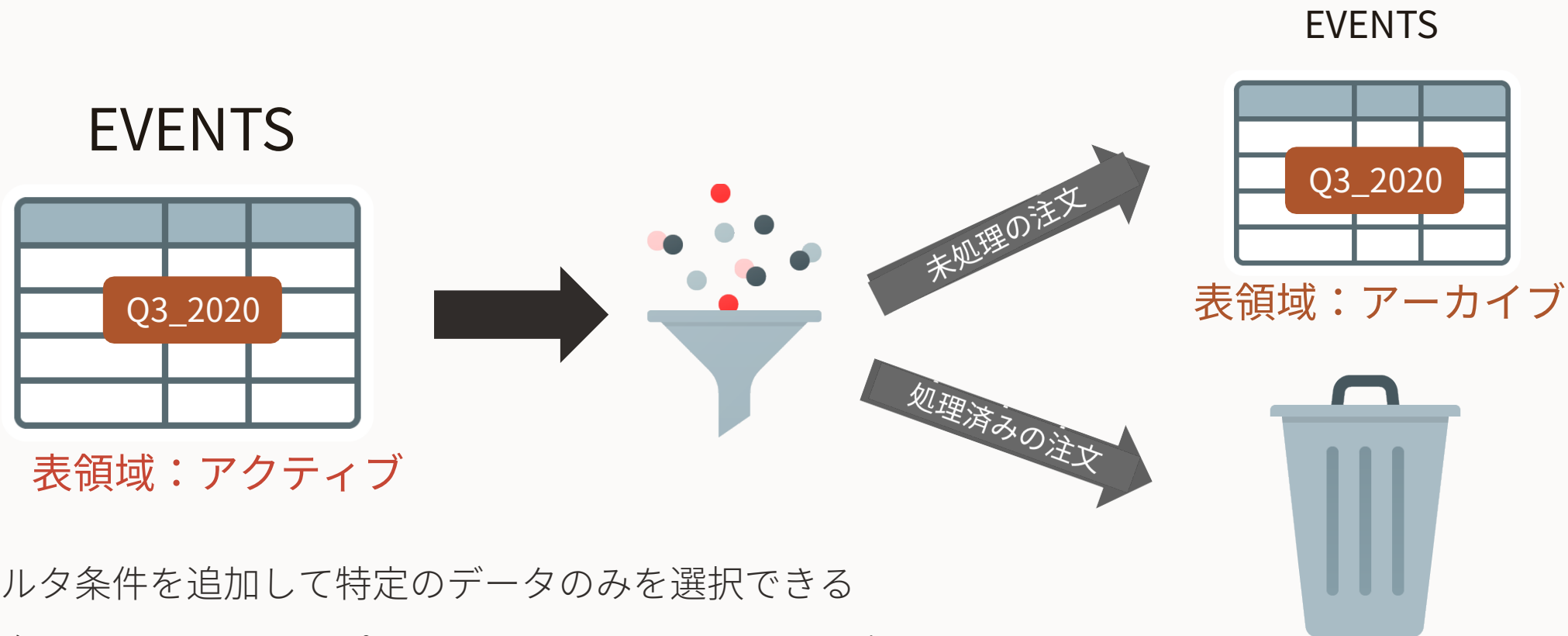
高速分割のように最適化をサポートする

# フィルタ付きパーティション・ メンテナンス操作

Oracle Database 12.2で導入

# フィルタ付きパーティション・メンテナンス操作

## パーティションの移動例



フィルタ条件を追加して特定のデータのみを選択できる  
データ・メンテナンスとパーティション・メンテナンスを  
組み合わせる

## フィルタ付きパーティション・メンテナンス操作の詳細

MOVE、SPLIT、MERGE操作に単一表のフィルタ条件を指定できる

- 指定条件はパーティション・メンテナンス全体で一貫していなければならない
- 指定条件では、対象のデータを明確に指定することが必要

指定条件は、さまざまな新しいパーティションやサブパーティションのセグメントを作成するために、繰り返し生成されるCTASコマンドに追加される

フィルタ条件はオフラインと新規オンラインの両方のPMOPに適用される

# フィルタ付きパーティション・メンテナンス操作

## パーティションの移動構文の例

---

```
ALTER TABLE orders MOVE PARTITION q3_2020  
TABLESPACE archive  
INCLUDING ROWS WHERE order_state = 'open';
```



# フィルタ付きパーティション・メンテナンス操作

## パーティションの移動構文の例

```
ALTER TABLE orders MOVE PARTITION q3_2020  
TABLESPACE archive online  
INCLUDING ROWS WHERE order_state = 'open';
```

.. オンラインにすると?

# フィルタ付きパーティション・メンテナンス操作

## オンライン操作でのDML動作

フィルタ条件は進行中の同時DMLには適用されない

```
INCLUDING ROWS WHERE order_state = 'open'
```

# フィルタ付きパーティション・メンテナンス操作

## オンライン操作でのDML動作

フィルタ条件は進行中の同時DMLには適用されない

```
INCLUDING ROWS WHERE order_state = 'open'
```

挿入は必ず実行される

```
INSERT VALUES(order_state = 'closed')
```

# フィルタ付きパーティション・メンテナンス操作

## オンライン操作でのDML動作

フィルタ条件は進行中の同時DMLには適用されない

```
INCLUDING ROWS WHERE order_state = 'open'
```

挿入は必ず実行される

```
INSERT VALUES(order_state = 'closed')
```

含まれているデータでの削除は必ず実行される

```
DELETE WHERE order_state = 'open'
```

削除されたデータでの削除は無効

```
DELETE WHERE order_state = 'closed'
```

# フィルタ付きパーティション・メンテナンス操作

## オンライン操作でのDML動作

フィルタ条件は進行中の同時DMLには適用されない

```
INCLUDING ROWS WHERE order_state = 'open'
```

挿入は必ず実行される

```
INSERT VALUES(order_state = 'closed')
```

含まれているデータでの削除は必ず実行される

```
DELETE WHERE order_state = 'open'
```

削除されたデータでの削除は無効

```
DELETE WHERE order_state = 'closed'
```

含まれているデータでの更新は必ず実行される

```
UPDATE set order_status = 'closed'  
WHERE order_state = 'open'
```

除外されたデータでの更新は無効

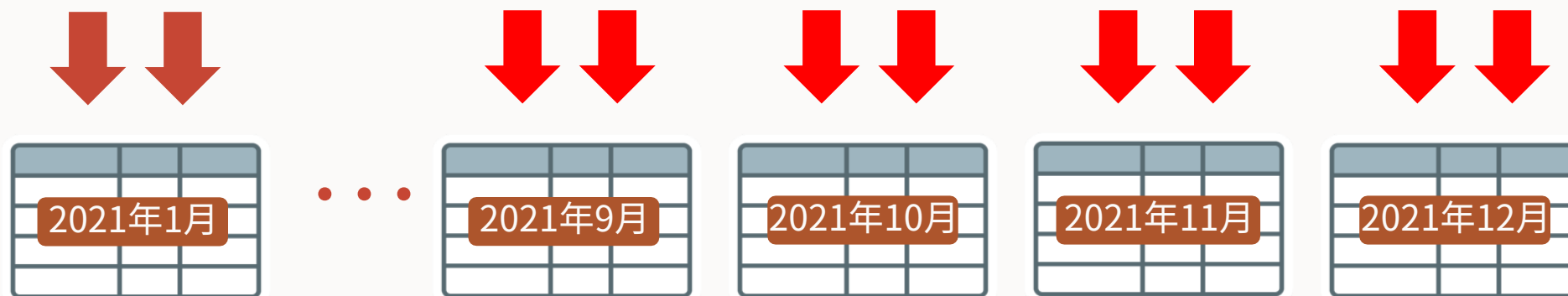
```
UPDATE set order_status = 'open'  
WHERE order_state = 'closed'
```

# パーティションのオンライン移動

Oracle 12c Release 1 (12.1) で導入

# 強化されたパーティション・メンテナンス操作

オンライン・パーティション移動



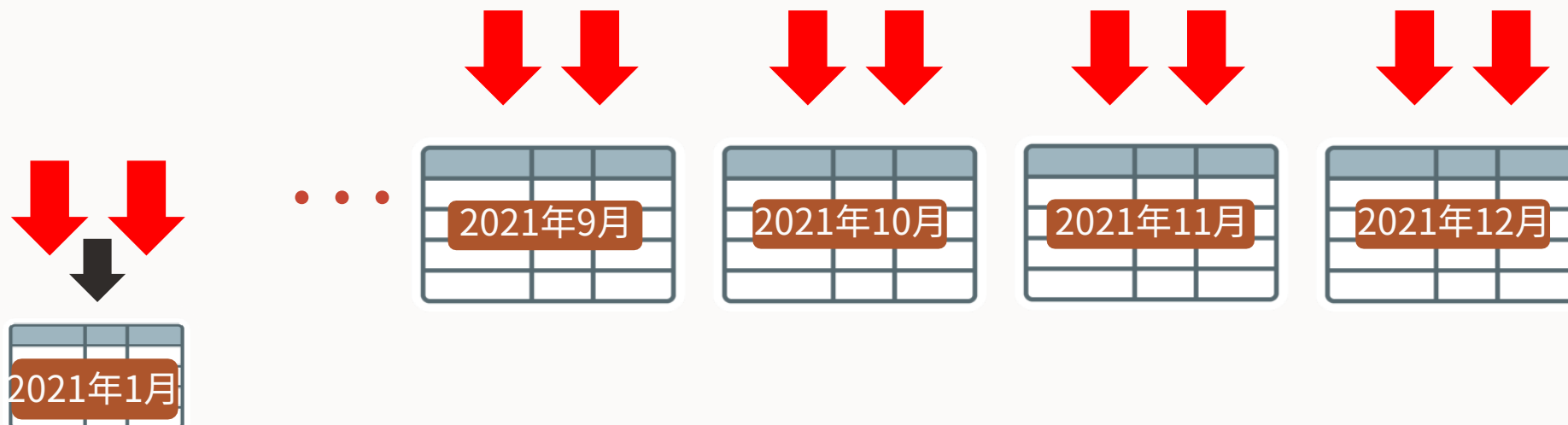
透過的なMOVE PARTITION ONLINE操作

同時DMLと問合せ

ローカル索引とグローバル索引の索引メンテナンス

# 強化されたパーティション・メンテナンス操作

## オンライン・パーティション移動



透過的なMOVE PARTITION ONLINE操作

同時DMLと問合せ

ローカル索引とグローバル索引の索引メンテナンス



# 強化されたパーティション・メンテナンス操作

## オンライン・パーティション移動 - ベスト・プラクティス

可能な場合は、同時実行DML操作を最小限に抑える

- ジャーナリング用に追加のディスク領域とリソースが必要
- ジャーナルは初期一括移動後に再帰的に適用される
- ジャーナルが大きいほど実行時間が長くなる

同時DMLは圧縮効率に影響する

- 初期一括移動による最良の圧縮比

# 非同期グローバル索引メンテナンス

Oracle 12c Release 1 (12.1) で導入

# 非同期グローバル索引メンテナンス

DROPおよびTRUNCATE PARTITIONの実行後に、索引メンテナンスを行わずに使用可能なグローバル索引

- 影響を受けるパーティションは、データ・アクセス時に内部で判別され、フィルタされる

DROPおよびTRUNCATEは高速なメタデータのための操作となる

- 大幅な高速化と初期リソース消費の削減

グローバル索引メンテナンスの遅延実行

- ALTER INDEX REBUILD|COALESCEによるメンテナンスの遅延実行
- スケジュール・ジョブを使用した自動クリーンアップ

# 非同期グローバル索引メンテナンス

## 変更前

```
SQL> select count(*) from pt partition for (9999);
```

COUNT(*)
25341440

```
Elapsed: 00:00:01.00
SQL> select index_name, status, orphaned_entries from user_indexes;
```

INDEX_NAME	STATUS	ORPHANED_ENTRIES
I1_PT	VALID	NO

```
Elapsed: 00:00:01.04
SQL>
SQL> alter table pt drop partition for (9999) update indexes;
```

Table altered.

Elapsed: 00:02:04.52

```
SQL>
SQL> select index_name, status, orphaned_entries from user_indexes;
```

INDEX_NAME	STATUS	ORPHANED_ENTRIES
I1_PT	VALID	NO

```
Elapsed: 00:00:00.10
```

## 変更後

```
SQL> select count(*) from pt partition for (9999);
```

COUNT(*)
25341440

```
Elapsed: 00:00:00.98
SQL> select index_name, status, orphaned_entries from user_indexes;
```

INDEX_NAME	STATUS	ORPHANED_ENTRIES
I1_PT	VALID	NO

```
Elapsed: 00:00:00.33
SQL>
SQL> alter table pt drop partition for (9999) update indexes;
```

Table altered.

Elapsed: 00:00:00.04

```
SQL>
SQL> select index_name, status, orphaned_entries from user_indexes;
```

INDEX_NAME	STATUS	ORPHANED_ENTRIES
I1_PT	VALID	YES

```
Elapsed: 00:00:00.05
```

# 非同期グローバル索引メンテナンス

メンテナンス・パッケージの初期実装

- 常にINDEX COALESCE CLEANUPを使用する
- 索引の並列処理に依存する

最新リリースに強化機能を追加

- INDEX COALESCE CLEANUPまたは“旧式の”索引クリーンアップを選択可
- メンテナンス操作で並列処理を選択可

より頻繁な索引クリーンアップの場合に推奨される旧式のクリーンアップ

- より一般的な顧客ユースケースになっており、したがって新しいデフォルト

バグ24515918に対処するため12.1で使用可能な機能

# パーティション表へのオンライン 表変換

Oracle Database 12.2/18.1で導入（パーティション間）

# オンライン表変換

EVENTS



EVENTS















完全ノンブロッキング（オンライン）DDL

# オンライン表変換

## 構文例

```
CREATE TABLE EVENTS ( sensor_grp      VARCHAR2 (50),  
                      channel          VARCHAR2 (50), ... );
```

```
ALTER TABLE EVENTS MODIFY  
PARTITION BY LIST ( sensor_grp )  
  (partition p1 values ('GYRO_GRP'),  
   partition p2 values ('CAMERA_GRP'),  
   partition p3 values ('THERMO_GRP'),  
   partition p4 values (DEFAULT))
```

```
UPDATE INDEXES ONLINE;
```



# オンライン表変換

## 索引付け

索引は変換プロセスを通じてオンラインで変換および保持される

索引として十分な柔軟性、今日のルールに従う

アクセス変更動作を最小限またはゼロに抑えるためのデフォルトの索引付けルール

- グローバル・パーティション索引は元のパーティション化形態を保持する
- 非同一キー索引はグローバル非パーティション索引となる
- 同一キー索引はローカル・パーティション索引に変換される
- ビットマップ索引はローカル・パーティション索引となる

# パーティション表のオンライン表変換

すべての人が大きいことを考え、小さい表から始めるのではない...

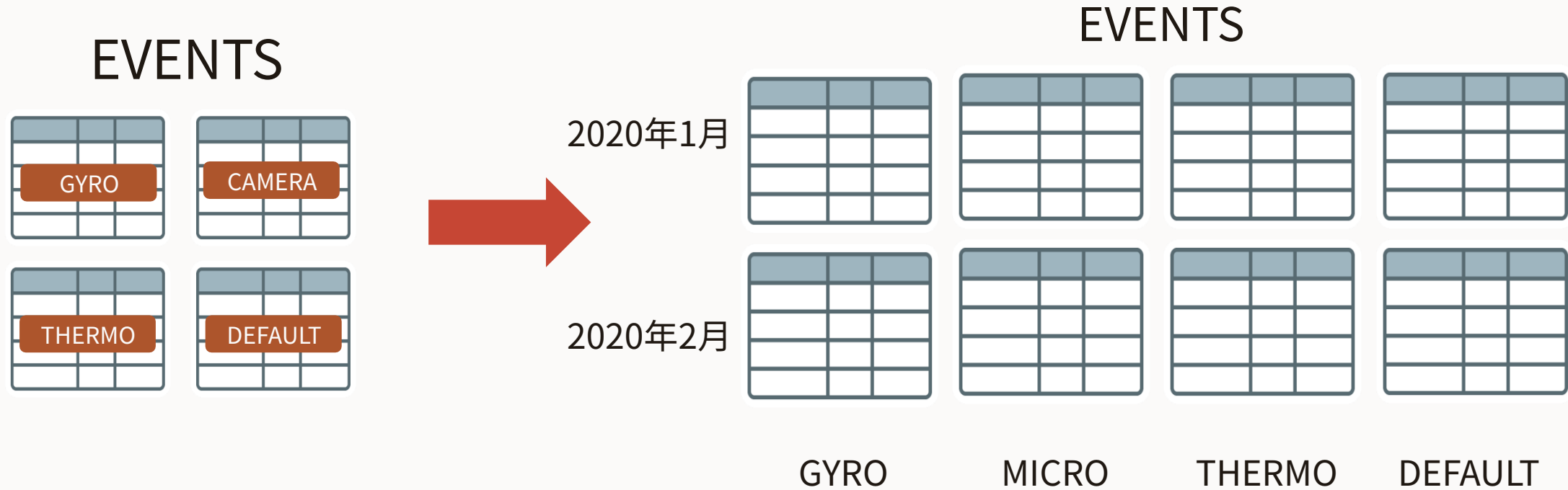
- ... 表は非パーティション表として小サイズから開始できる
- ... 表が成長を続ける
- ... 表が予期されていた方法とは異なる方法で使用される
- ... 表のメンテナンスが問題となる
- ... パフォーマンスに影響が出る

そのような表を停止時間なしでどのように変換するのか?

パーティション化を採用できる...

- ... しかし、"間違った"タイプ/細分度を選択した (理由が何であれ)

# パーティション表のオンライン表変換



表および索引用の完全ノンブロッキング（オンライン）DDL

## パーティション表のオンライン表変換

索引は変換プロセスを通じてオンラインで変換および保持される

アクセス変更動作を最小限またはゼロに抑えるためのデフォルトの索引付けルール

- 非パーティションの変換のルールと比べてほとんど同じ
- 相違点：
  - ローカル索引は、2つのディメンションのパーティション・キーのいずれかが含まれる場合、ローカルのまま留まる
  - グローバル同一キー・パーティション索引は、ローカル・パーティション索引に変換される

索引として十分な柔軟性、今日のルールに従う

- 変更中の見たいものを何であれオーバーライドする

## パーティション表のオンライン表変換

```
CREATE TABLE EVENTS ( run_id          NUMBER,
                      sensor_type VARCHAR2 (50), ... )
PARTITION BY LIST ( ... )

ALTER TABLE EVENTS MODIFY
PARTITION BY RANGE      ( run_id          )
SUBPARTITION BY LIST ( sensor_type )...

UPDATE INDEXES
  (i1_run_id GLOBAL,
   i2_sensor LOCAL,
   i3 GLOBAL PARTITION BY RANGE ( ... )
     (PARTITION p0100 VALUES LESS THAN (100000),
      PARTITION p1500 VALUES LESS THAN (1500000),
      PARTITION pmax  VALUES LESS THAN (MAXVALUE)))

ONLINE;
```

# 交換用の表の作成

Oracle Database 12.2で導入

# 交換用の表の作成

簡単なDDLコマンド

セマンティックと内部表形態の両方を確実に同一にして、パーティション変換コマンドが常に成功するようにする

特別なCREATE TABLE AS SELECT操作のように操作する

必ず空の表を作成する

# 交換用の表の作成

## 構文例

---

```
CREATE TABLE events_cp TABLESPACE ts_boson  
FOR EXCHANGE WITH events;
```

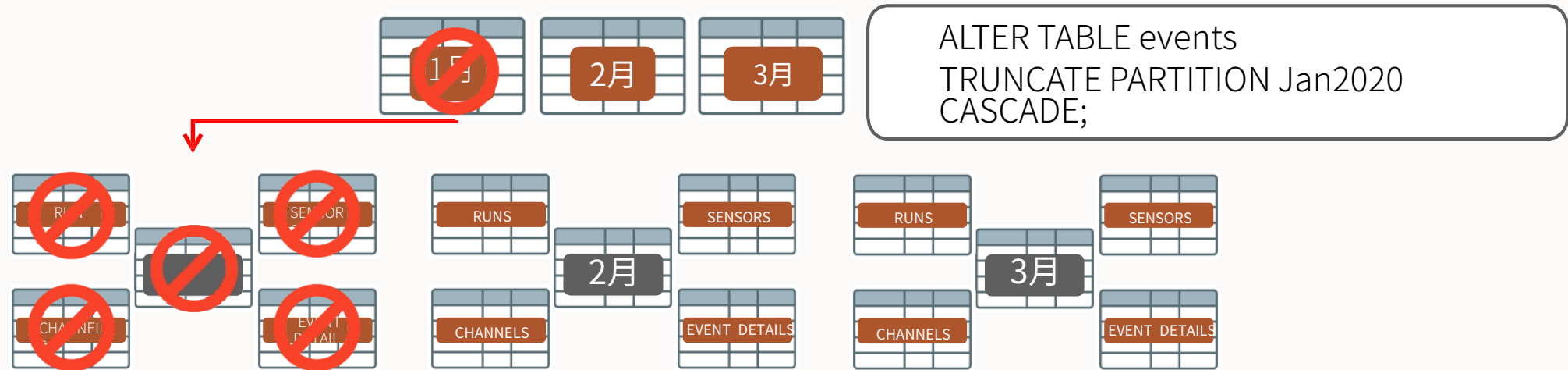


# 参照パーティション化のための 切捨てと交換のカスケード

Oracle 12c Release 1 (12.1) で導入

# 高度なパーティション化のメンテナンス

## TRUNCATEとEXCHANGE PARTITIONのカスケード

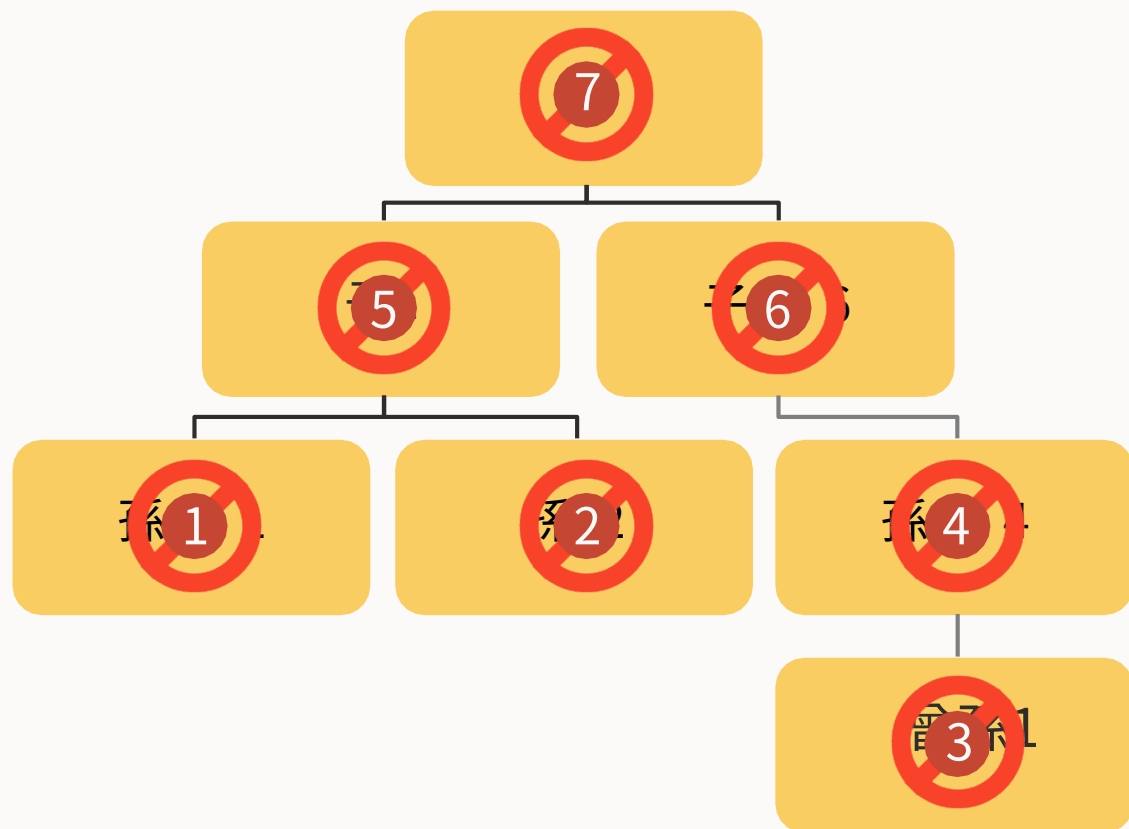


ビジネス継続性を向上させるためのTRUNCATEとEXCHANGEのカスケード

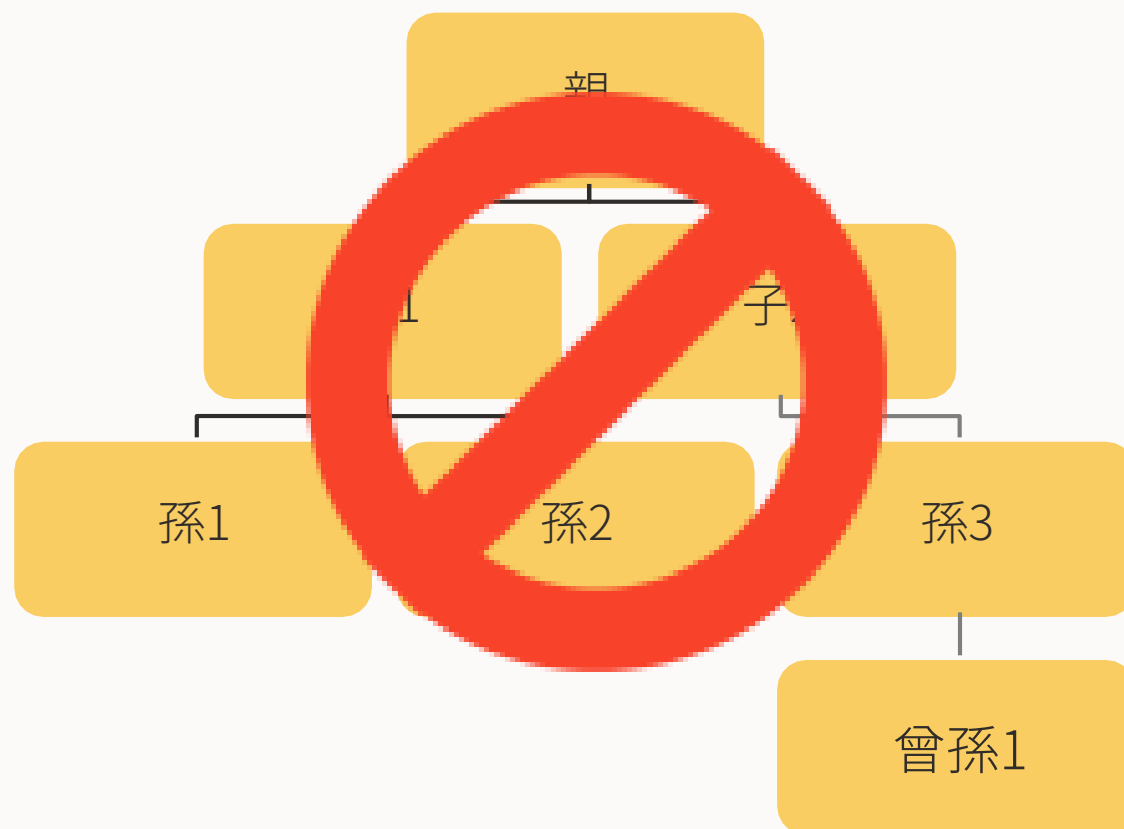
1つの原子トランザクションによってデータの整合性を維持

簡素化されエラーの入り込みにくいコード開発

# TRUNCATE PARTITIONのカスケード



適切なボトムアップ処理が必要  
7つの個別切捨て操作



1つの切捨て操作

## TRUNCATE PARTITIONのカスケード

```
SQL> create table intRef_p (pkcol number not null, col2 varchar2(200),
 2                          constraint pk_intref primary key (pkcol))
 3  partition by range (pkcol) interval (10)
 4  (partition p1 values less than (10));
```

Table created.

```
SQL>
SQL> create table intRef_c1 (pkcol number not null, col2 varchar2(200), fkcol number not null,
 2                          constraint pk_c1 primary key (pkcol),
 3                          constraint fk_c1 foreign key (fkcol) references intRef_p(pkcol) ON DELETE CASCADE)
 4  partition by reference (fk_c1);
```

Table created.

## TRUNCATE PARTITIONのカスケード

```
SQL> create table intRef_p (pkcol number(2) constraint pkc1 primary key,
3 partition by range (pkcol) into (partition p1 values less than
4 (partition p2 values less than
```

Table created.

```
SQL>
SQL> create table intRef_c1 (pkcol number(2) constraint pkc2 primary key,
3 constraint fkc1 foreign key (pkcol) references intRef_p (pkcol),
4 partition by reference (fk_c1);
```

Table created.

```
SQL> select * from intRef_p;
```

PKCOL	COL2
333	data for truncate - p
999	data for truncate - p

```
SQL> select * from intRef_c1;
```

PKCOL	COL2	FKCOL
1333	data for truncate - c1	333
1999	data for truncate - c1	999

```
SQL> alter table intRef_p truncate partition for (999) cascade update indexes;
```

Table truncated.

```
SQL> select * from intRef_p;
```

PKCOL	COL2
333	data for truncate - p

```
SQL> select * from intRef_c1;
```

PKCOL	COL2	FKCOL
1333	data for truncate - c1	333

# TRUNCATE PARTITIONのカスケード

CASCADEは参照ツリー全体に適用される

- 1つの原子トランザクション、オール・オア・ナッシング
- 複雑さや深さは問題ではない
- 参照パーティション表の任意のレベルで指定可能
- すべての外部キーの場合にON DELETE CASCADEが必要

TRUNCATEのカスケードは非パーティション表でも使用可能

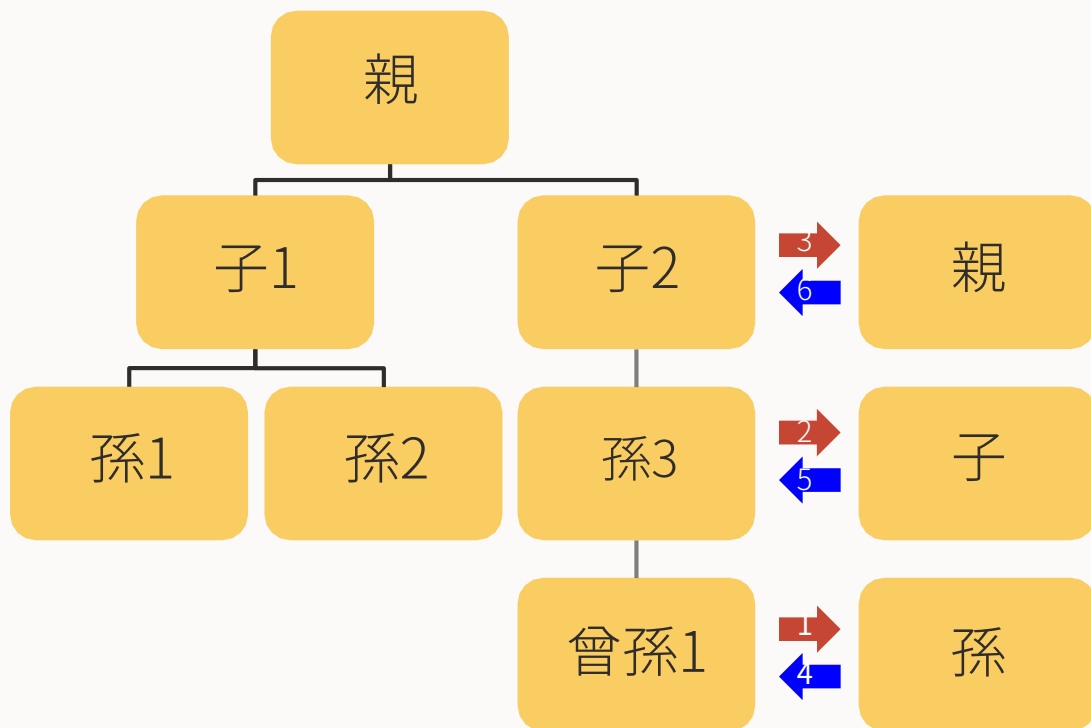
- 非パーティション表の依存性ツリーは、無効な外部キーの制約によって中断される可能性がある

参照パーティション階層は、交換対象のターゲットと表に適合する必要がある

同じレベルに複数の子が存在する複雑なツリーの場合、所定のレベルの各子は親表の別のキーを参照する必要がある

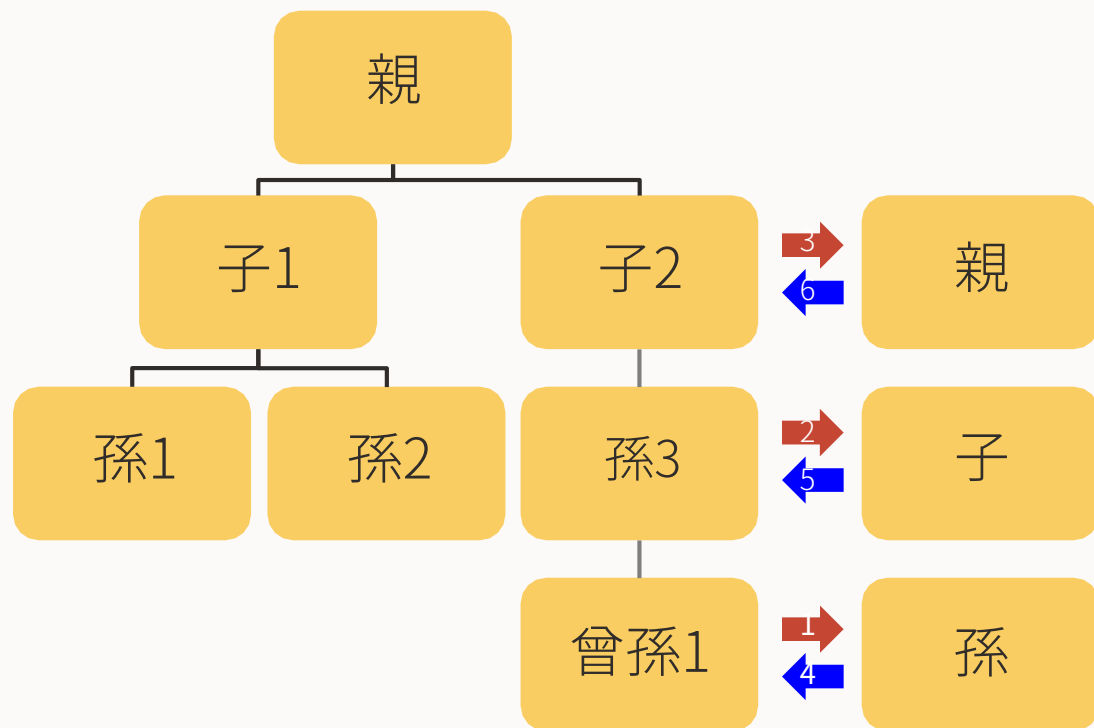
- 階層ツリー内の表を明確にペアにするために必要

# EXCHANGE PARTITIONのカスケード

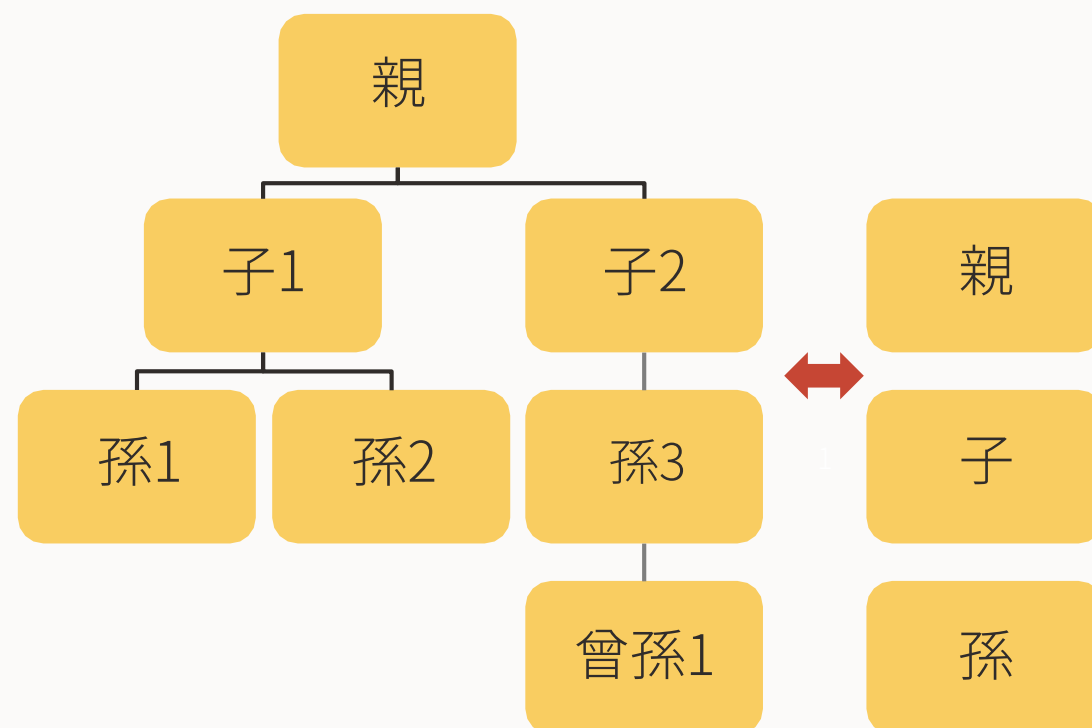


ターゲットからのボトムアップ交換（クリア）  
ターゲットへのトップダウン交換（移入）

# EXCHANGE PARTITIONのカスケード



ターゲットからのボトムアップ交換（クリア）  
ターゲットへのトップダウン交換（移入）



完全な階層ツリーの交換  
1つの交換操作



## EXCHANGE PARTITIONのカスケード

```
SQL> create table intRef_p (pkcol number not null, col2 varchar2(200),
 2          constraint pk_intref primary key (pkcol))
 3 partition by range (pkcol) interval (10)
 4 (partition p1 values less than (10));

SQL> create table intRef_c1 (pkcol number not null, col2 varchar2(200), fkcol number not null,
 2          constraint pk_c1 primary key (pkcol),
 3          constraint fk_c1 foreign key (fkcol) references intRef_p(pkcol) ON DELETE CASCADE)
 4 partition by reference (fk_c1);

SQL> create table intRef_gc1 (col1 number not null, col2 varchar2(200), fkcol number not null,
 2          constraint fk_gc1 foreign key (fkcol) references intRef_c1(pkcol) ON DELETE CASCADE)
 3 partition by reference (fk_gc1);
```

## EXCHANGE PARTITIONのカスケード

```
SQL> REM create some PK-FK equivalent table construct for exchange
SQL> create table XintRef_p (pkcol number not null, col2 varchar2(200),
  2                          constraint xpk_intref primary key (pkcol));

SQL> create table XintRef_c1 (pkcol number not null, col2 varchar2(200), fkcol number not null,
  2                          constraint xpk_c1 primary key (pkcol),
  3                          constraint xfk_c1 foreign key (fkcol) references XintRef_p(pkcol) ON DELETE CASCADE);

SQL> create table XintRef_gc1 (col1 number not null, col2 varchar2(200), fkcol number not null,
  2                          constraint xfk_gc1 foreign key (fkcol) references XintRef_c1(pkcol) ON DELETE CASCADE);
```

## EXCHANGE PARTITIONのカスケード

```
SQL> select * from intRef_p;
```

PKCOL	COL2
-------	------

333	p333 - data BEFORE exchange - p
999	p999 - data BEFORE exchange - p

```
SQL> select * from intRef_c1;
```

PKCOL	COL2
-------	------

FKCOL
-------

1333	p333 - data BEFORE exchange - c1	333
1999	p999 - data BEFORE exchange - c1	999

```
SQL> select * from intRef_gc1;
```

COL1	COL2
------	------

FKCOL
-------

1333	p333 - data BEFORE exchange - gc1	1333
1999	p999 - data BEFORE exchange - gc1	1999

```
SQL> select * from XintRef_p;
```

PKCOL	COL2
-------	------

333	p333 - data AFTER exchange - p
-----	--------------------------------

```
SQL> select * from XintRef_c1;
```

PKCOL	COL2
-------	------

FKCOL
-------

1333	p333 - data AFTER exchange - c1	333
------	---------------------------------	-----

```
SQL> select * from XintRef_gc1;
```

COL1	COL2
------	------

FKCOL
-------

1333	p333 - data AFTER exchange - gc1	1333
------	----------------------------------	------

## EXCHANGE PARTITIONのカスケード

```
SQL> alter table intRef_p exchange partition for (333) with table XintRef_p cascade update indexes;  
Table altered.
```

# EXCHANGE PARTITIONのカスケード

```
SQL> select * from intRef_p;
```

```
PKCOL COL2
```

```
-----  
333 p333 - data AFTER exchange - p  
999 p999 - data BEFORE exchange - p
```

```
SQL> select * from intRef_c1;
```

```
PKCOL COL2
```

```
FKCOL
```

```
-----  
1333 p333 - data AFTER exchange - c1 333  
1999 p999 - data BEFORE exchange - c1 999
```

```
SQL> select * from intRef_gc1;
```

```
COL1 COL2
```

```
FKCOL
```

```
-----  
1333 p333 - data AFTER exchange - gc1 1333  
1999 p999 - data BEFORE exchange - gc1 1999
```

```
SQL> select * from XintRef_p;
```

```
PKCOL COL2
```

```
-----  
333 p333 - data BEFORE exchange - p
```

```
SQL> select * from XintRef_c1;
```

```
PKCOL COL2
```

```
FKCOL
```

```
-----  
1333 p333 - data BEFORE exchange - c1 333
```

```
SQL> select * from XintRef_gc1;
```

```
COL1 COL2
```

```
FKCOL
```

```
-----  
1333 p333 - data BEFORE exchange - gc1 1333
```

# パーティション化 - ヒント

---

# レンジとインターバルの相違点

---

# インターバル・パーティション化

同サイズ・レンジ・パーティションの完全自動化

パーティションはメタデータ情報としてのみ作成される

- 開始パーティションは永続化される

新規データを受け取るとすぐにセグメントが割り当てられる

- 新規パーティションの作成は不要
- ローカル索引も作成およびメンテナンスされる

インターバル・パーティション化は、レンジ・パーティション化の透過的な拡張とほぼ同じ

- ..ただし、インターバル実装はいくつかの点で微妙に異なる



# インターバル・パーティション化とレンジ・パーティション化

## パーティション・バウンド

- インターバル・パーティションには下限と上限がある
  - 無限上限なし (MAXVALUES)
- レンジ・パーティションには上限のみがある
  - 下限は前のパーティションによって導出される
  - 上限は無限 (MAXVALUES)

## パーティションの名前付け

- インターバル・パーティションに事前に名前付けすることはできない
  - PARTITION FOR (<value>)句を使用する
- レンジ・パーティションには名前を付ける必要がある

## インターバル・パーティション化とレンジ・パーティション化（続き）

### パーティション・マージ

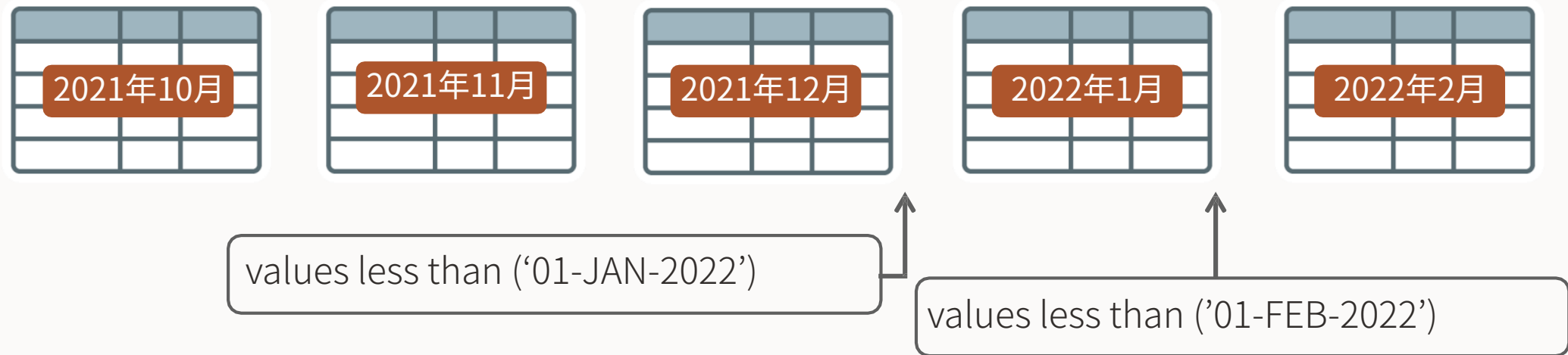
- 複数の実在しないインターバル・パーティションは暗黙的にマージされる
- 任意の時点で隣接する2つのレンジ・パーティションのみをマージ可能

### パーティション数

- インターバル・パーティション表のパーティション数は常に100万個
  - 実在しないパーティションはINTERVAL句によって"存在する"
  - インターバル・パーティション化用のMAXVALUE句はない
    - 最大値はパーティション数とINTERVAL句によって定義する
- レンジ・パーティション化の最大パーティション数は100万個
  - MAXVALUE句によって最上位パーティションを定義する

# インターバル・パーティション化とレンジ・パーティション化

## レンジ・パーティション化のパーティション・バウンド

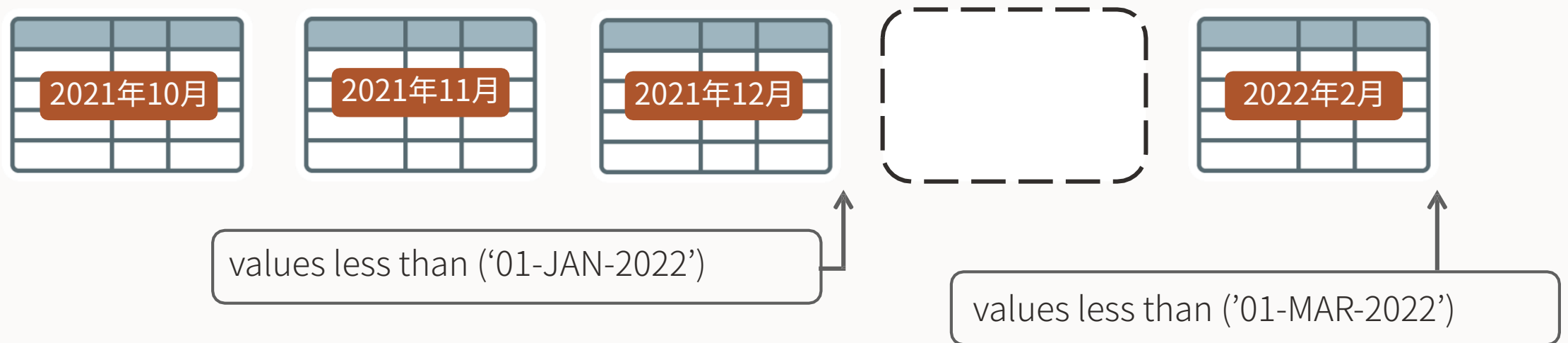


パーティションには上限のみがある

- 下限は前のパーティションの上限から導出される

# インターバル・パーティション化とレンジ・パーティション化

## レンジ・パーティション化のパーティション・バウンド

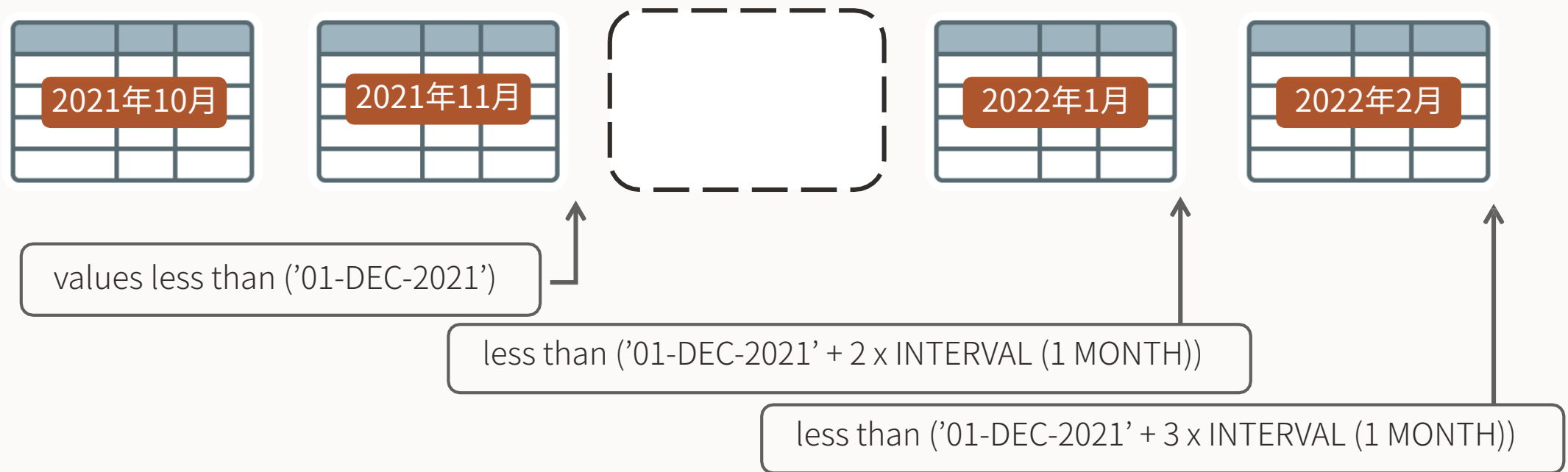


前のパーティションが削除されると下限に移動する

- これにより、“Feb 2022”で01-JAN-2022から28-FEB-2022までが生成される

# インターバル・パーティション化とレンジ・パーティション化

## インターバル・パーティション化のパーティション・バウンド

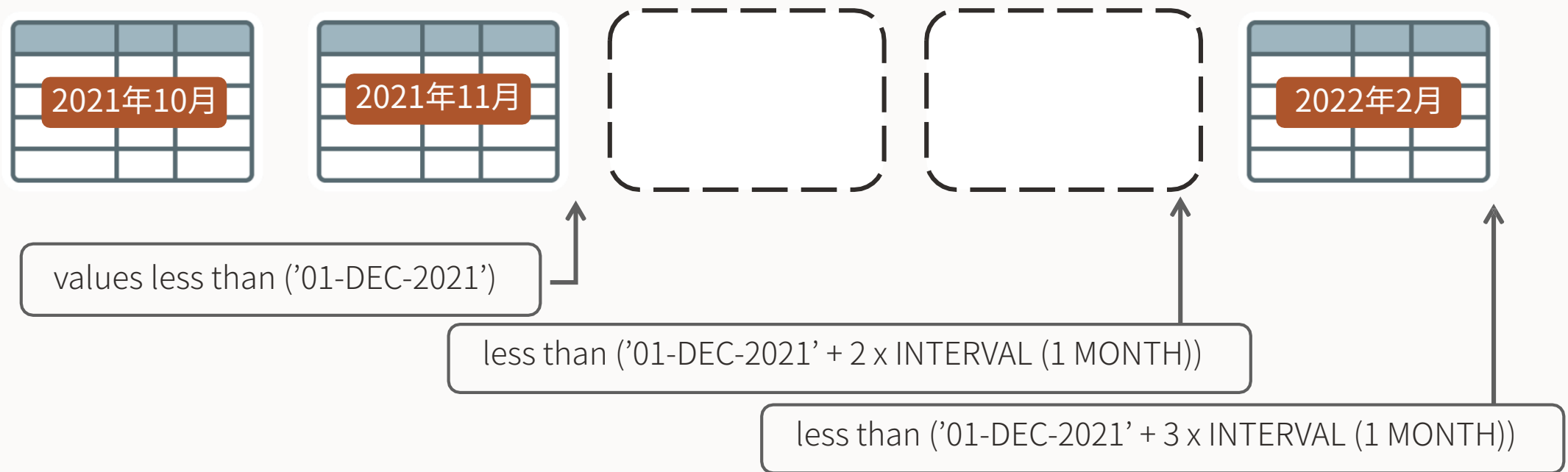


パーティションには上限と下限がある

- INTERVAL関数と最後のレンジ・パーティションによって導出される

# インターバル・パーティション化とレンジ・パーティション化

## インターバル・パーティション化のパーティション・バウンド



削除はパーティションの境界に影響しない

- “Feb 2022”により引き続き01-FEB-2022から28-FEB-2022が生成される

# インターバル・パーティション化とレンジ・パーティション化

## パーティションの名前付け

レンジ・パーティションには名前を付けることができる

- 指定しない場合はシステム生成の名前を使用

```
SQL> alter table t add partition values less than(20); Table altered.  
SQL> alter table t add partition P30 values less than(30);  
Table altered.
```

インターバル・パーティションに名前を付けることはできない

- 常にシステム生成の名前を使用

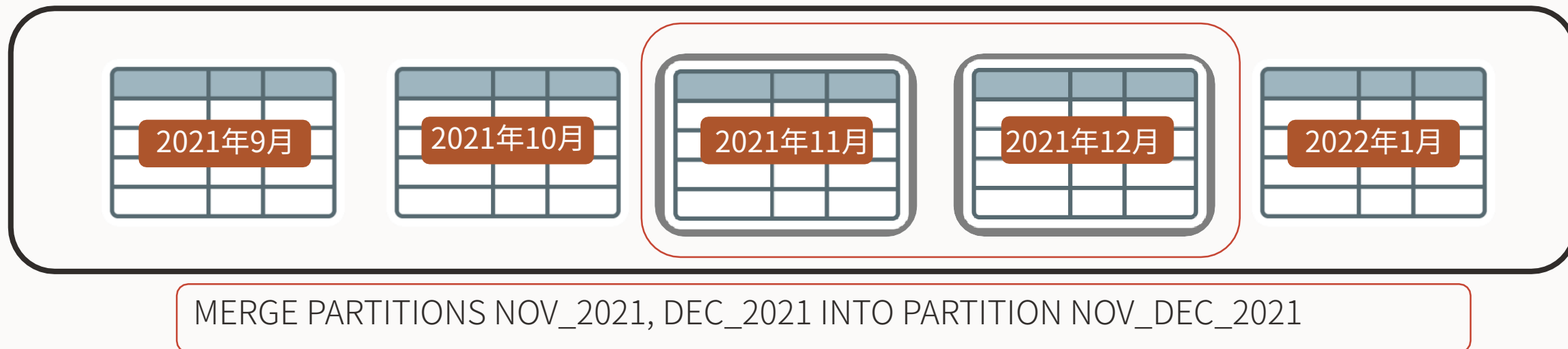
```
SQL> alter table t add partition values less than(20);  
ERROR at line 1:ORA-14760:ADD PARTITION is not permitted  
on Interval partitioned objects
```

新しい決定的PARTITION FOR ()拡張子を使用する

```
SQL> alter table t1 rename partition for (9) to p_10;  
Table altered.
```

# インターバル・パーティション化とレンジ・パーティション化

## パーティション・マージ - レンジ・パーティション化



レンジ・パーティション化で2つの隣接するパーティションをマージする

- 上位パーティションの上限が新しい上限
- 下限は前のパーティションの上限から導出される



# インターバル・パーティション化とレンジ・パーティション化

## パーティション・マージ - レンジ・パーティション化

2021年9月		

2021年10月		

NOV_DEC_2021		

2022年1月		

```
MERGE PARTITIONS NOV_2021, DEC_2021 INTO PARTITION NOV_DEC_2021
```

マージされたパーティションの新規セグメントが作成される

- 表の残りの部分には影響しない

# インターバル・パーティション化とレンジ・パーティション化

## パーティション・マージ - インターバル・パーティション化



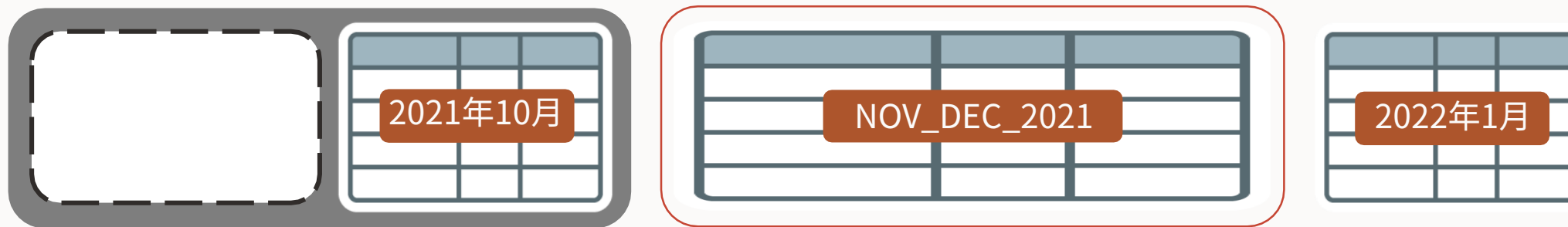
```
MERGE PARTITIONS NOV_2021, DEC_2021 INTO PARTITION NOV_DEC_2021
```

インターバル・パーティション化で2つの隣接するパーティションをマージする

- 上位パーティションの上限が新しい上限
- 下限は最初のパーティションの下限から導出される

# インターバル・パーティション化とレンジ・パーティション化

## パーティション・マージ - インターバル・パーティション化



```
MERGE PARTITIONS NOV_2021, DEC_2021 INTO PARTITION NOV_DEC_2021
```

マージされたパーティションの新規セグメントが作成される

- 最上位の非インターバル・パーティションの前の空白も暗黙的に"マージ"される
  - インターバルは最上位非インターバル・パーティションを越えたパーティションでのみ有効

# 複数列レンジ・パーティション化

Oracle 8i (8.1) で導入

# 複数列レンジ・パーティション化

## 概念

パーティション化キーは複数列で構成され、後続の列によって先行する列よりも高い粒度を定義する

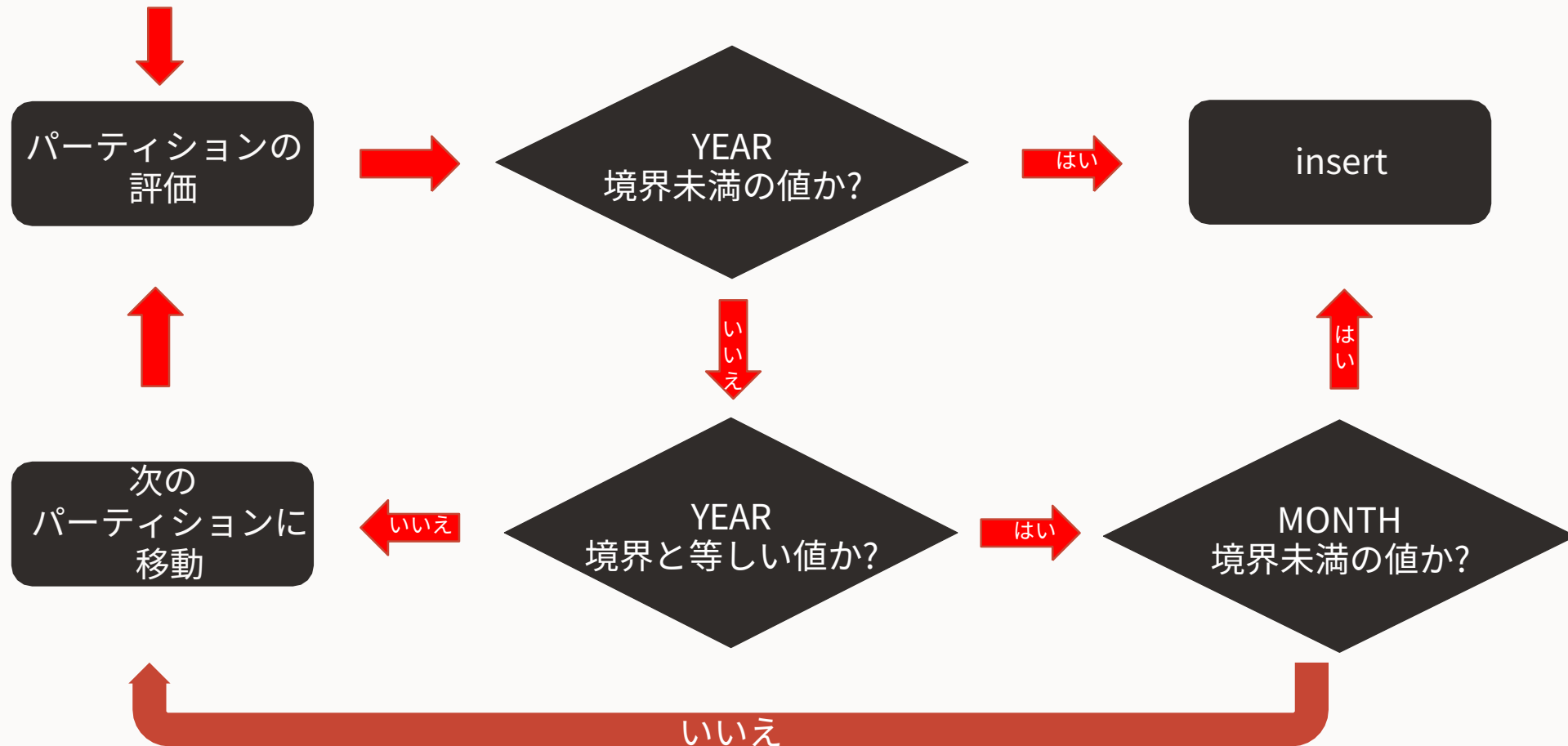
- 例： (YEAR, MONTH, DAY)
- nディメンションのパーティション化ではない

おもな注目点は、パーティション境界の評価方法の違い

- 簡単なレンジの場合、境界は (排他) 未満
- 複数列レンジの境界は、以下
  - n番目の列は、複数列キーの先行するすべての (n-1) の値がパーティションの (n-1) バウンドと完全に一致する場合にのみ調査される

# 複数列レンジ・パーティション

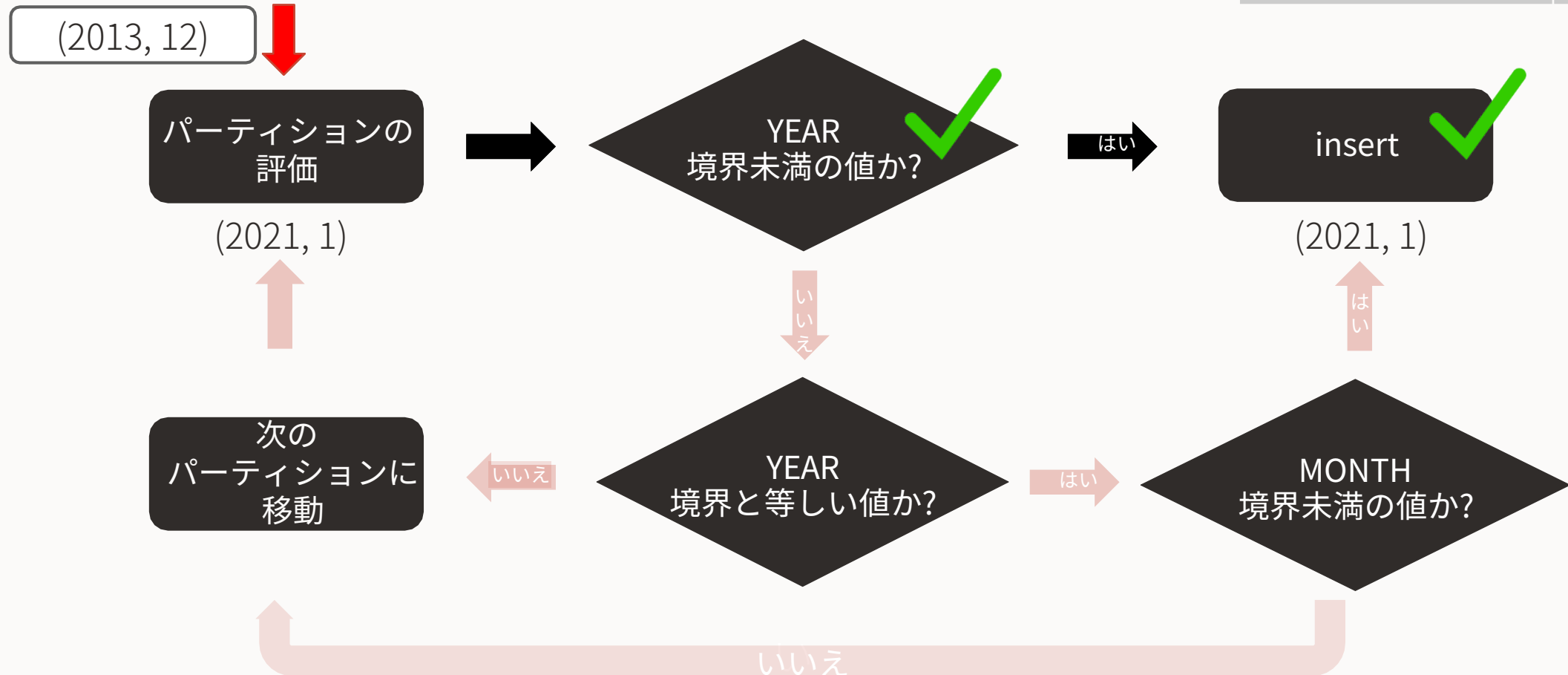
サンプル・ディシジョン・ツリー (YEAR、MONTH)



# 複数列レンジ・パーティション

## 例

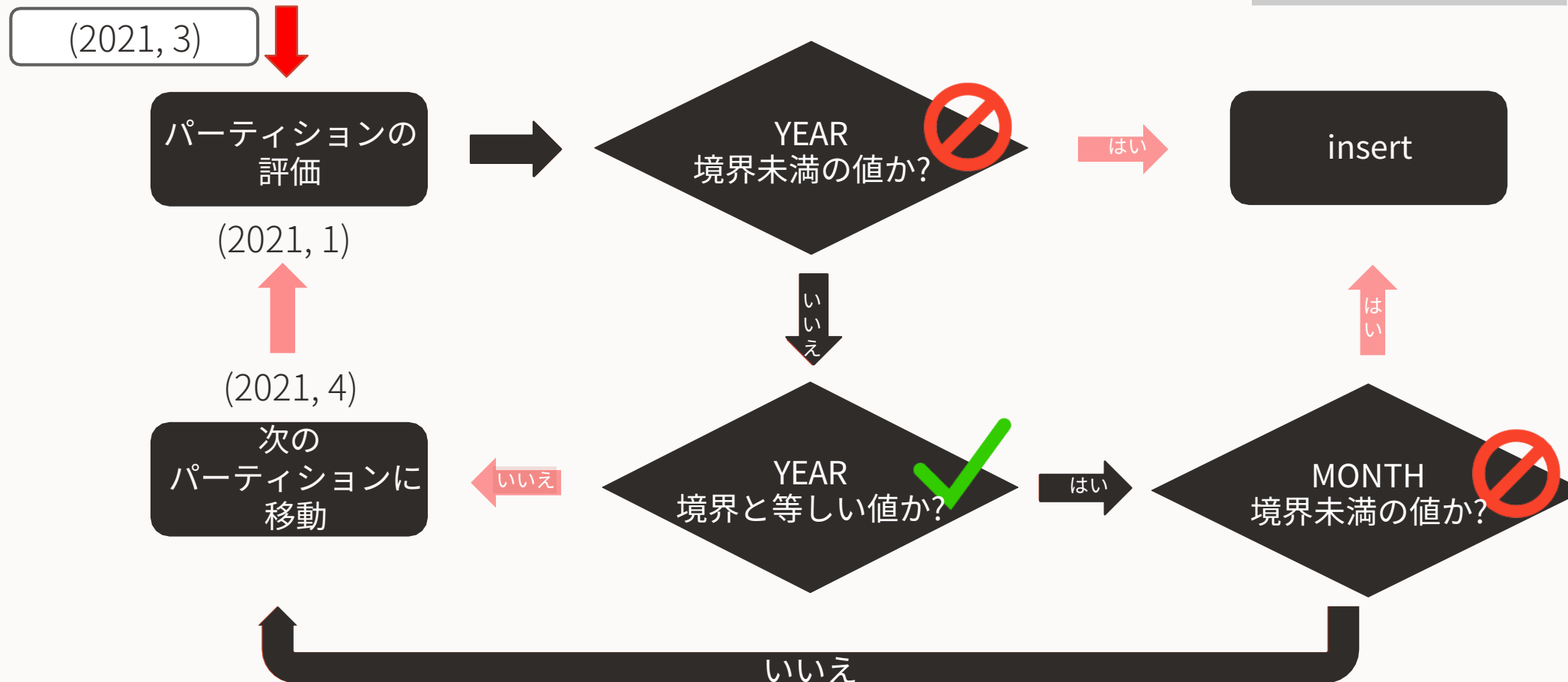
(YEAR、MONTH) 境界	値
(2021,1)	(2013, 12)
(2021,4)	
(2021,7)	
(2021,10)	
(2022,1)	
(MAXVALUE,0)	



# 複数列レンジ・パーティション

## 例 (続き)

(YEAR、MONTH) 境界	値
(2021,1)	(2013, 12)
(2021,4)	
(2021,7)	
(2021,10)	
(2022,1)	
(MAXVALUE,0)	

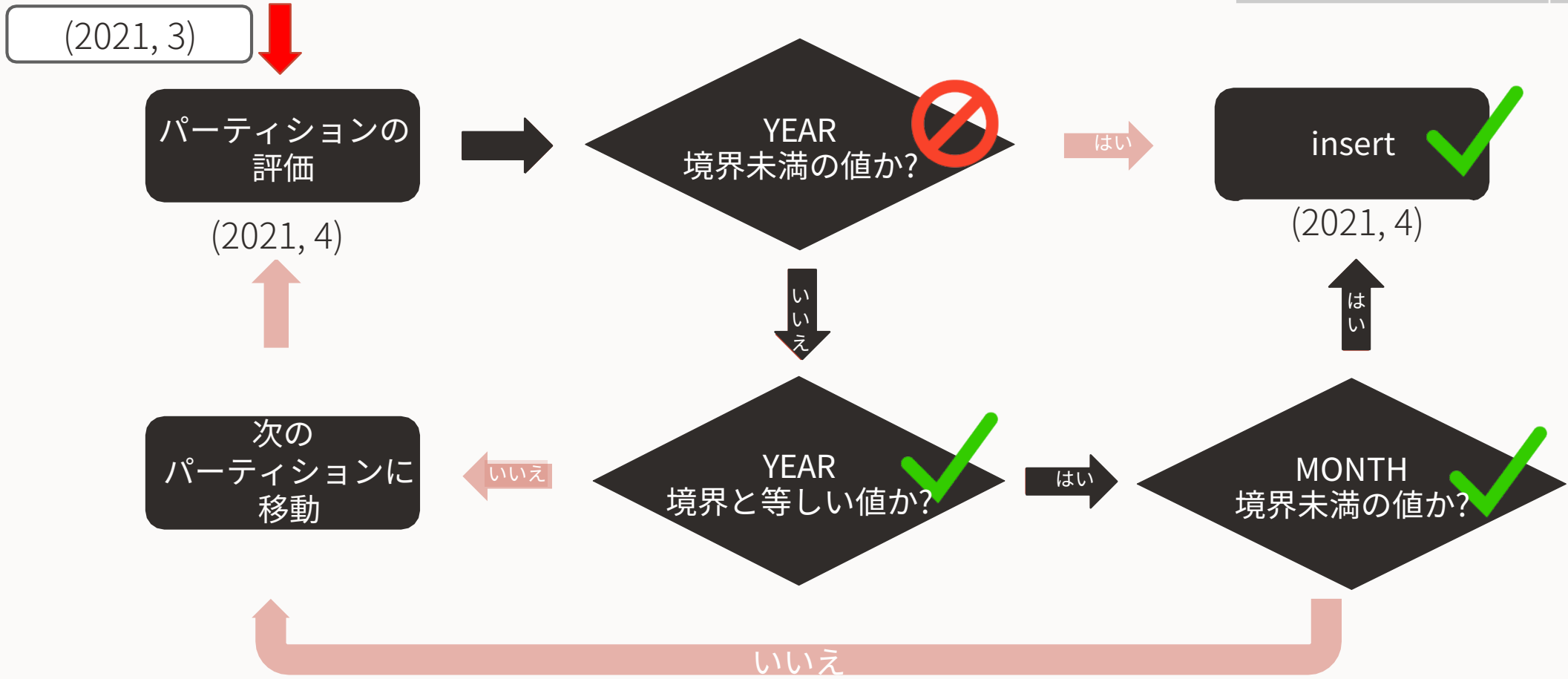




# 複数列レンジ・パーティション

## 例 (続き)

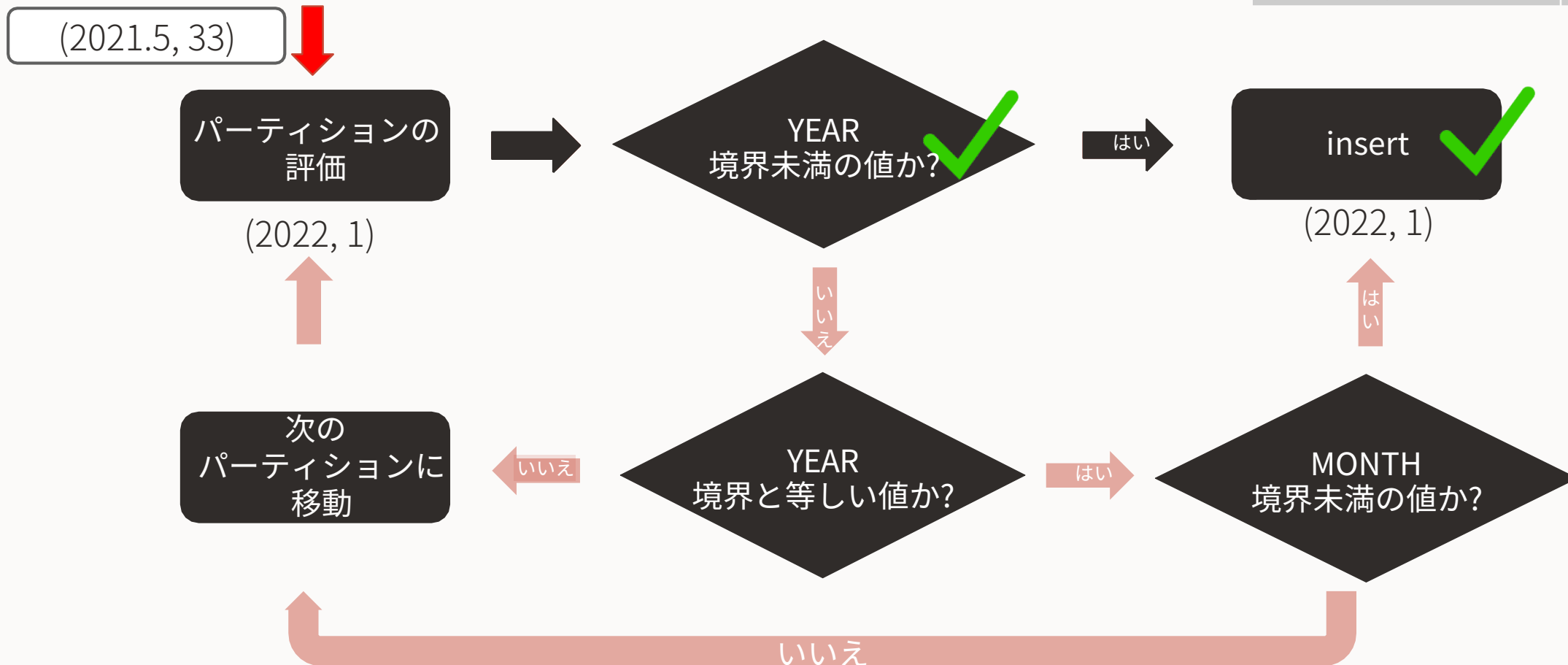
(YEAR、MONTH) 境界	値
(2021,1)	(2013, 12)
(2021,4)	
(2021,7)	
(2021,10)	
(2022,1)	
(MAXVALUE,0)	



# 複数列レンジ・パーティション

## 例（続き）

(YEAR、MONTH) 境界	値
(2021,1)	(2013, 12)
(2021,4)	
(2021,7)	
(2021,10)	
(2022,1)	
(MAXVALUE,0)	



# 複数列レンジ・パーティション化

## 留意事項

- ✓ 3番目（以上）のディメンションを追加するための強力なパーティション化メカニズム
  - より小さなデータ・パーティション

プルーニングは、先頭の列をフィルタすることなく、後続の列条件にも有効

- △ 境界はパーティション定義によって適用されない
  - レンジは連続的

論理的なADDパーティションは、表の中間でのSPLITパーティションを意味することもある

# 複数列レンジ・パーティション

わずかに異なる現実のシナリオ

複数列レンジは、3番目の（数値以外の）ディメンションの導入に使用される

```
CREATE TABLE events (event_id number, site_id CHAR(2),start_date date)
PARTITION BY RANGE (site_id, start_date)
SUBPARTITION BY HASH (event_id) SUBPARTITIONS 16
(PARTITION I1_2020 VALUES LESS THAN ('L1',to_date('01-JAN-2021','dd-mon-yyyy')),
PARTITION I1_2021 VALUES LESS THAN ('L1',to_date('01-JAN-2022','dd-mon-yyyy')),
PARTITION I2_2020 VALUES LESS THAN ('L2',to_date('01-JAN-2021','dd-mon-yyyy')),
PARTITION I3_2020 VALUES LESS THAN ('L3',to_date('01-JAN-2021','dd-mon-yyyy')),
PARTITION x3_2021 VALUES LESS THAN ('X1',to_date('01-JAN-2022','dd-mon-yyyy')),
PARTITION x4_2020 VALUES LESS THAN ('X4',to_date('01-JAN-2021','dd-mon-yyyy'))
);
```

文字SITE\_IDは順番に定義しなければならない

# 複数列レンジ・パーティション

わずかに異なる現実のシナリオ

複数列レンジは、3番目の（数値以外の）ディメンションの導入に使用される

```
CREATE TABLE events (event_id number, site_id CHAR(2),start_date date)
PARTITION BY RANGE (site_id, start_date)
SUBPARTITION BY HASH (event_id) SUBPARTITIONS 16
(PARTITION l1_2020 VALUES LESS THAN ('L1',to_date('01-JAN-2021','dd-mon-yyyy')),
PARTITION l1_2021 VALUES LESS THAN ('L1',to_date('01-JAN-2022','dd-mon-yyyy')),
PARTITION l2_2020 VALUES LESS THAN ('L2',to_date('01-JAN-2021','dd-mon-yyyy')),
PARTITION l3_2020 VALUES LESS THAN ('L3',to_date('01-JAN-2021','dd-mon-yyyy')),
PARTITION x3_2021 VALUES LESS THAN ('X1',to_date('01-JAN-2022','dd-mon-yyyy')),
PARTITION x4_2020 VALUES LESS THAN ('X4',to_date('01-JAN-2021','dd-mon-yyyy'))
);
```

未定義のSITE\_IDはLESS THANプローブに従い、常に定義済みSITE\_IDの最下位パーティションで終わる

# 複数列レンジ・パーティション

わずかに異なる現実のシナリオ

複数列レンジは、3番目の（数値以外の）ディメンションの導入に使用される

```
CREATE TABLE events (event_id number, site_id CHAR(2),start_date date)
PARTITION BY RANGE (site_id, start_date)
SUBPARTITION BY HASH (event_id) SUBPARTITIONS 16
PARTITION I1_2020 VALUES LESS THAN ('L1',to_date('01-JAN-2021','dd-mon-yyyy')),
PARTITION I1_2021 VALUES LESS THAN ('L1',to_date('01-JAN-2022','dd-mon-yyyy')),
PARTITION I2_2020 VALUES LESS THAN ('L2',to_date('01-JAN-2021','dd-mon-yyyy')),
PARTITION I3_2020 VALUES LESS THAN ('L3',to_date('01-JAN-2021','dd-mon-yyyy')),
PARTITION x3_2021 VALUES LESS THAN ('X1',to_date('01-JAN-2022','dd-mon-yyyy')),
PARTITION x4_2020 VALUES LESS THAN ('X4',to_date('01-JAN-2021','dd-mon-yyyy'))
);
```

将来の日付は常に、次に上位のSITE\_IDの最下位パーティションに入るか、または拒否される

# 複数列レンジ・パーティション

わずかに異なる現実のシナリオ

複数列レンジは、3番目の（数値以外の）ディメンションの導入に使用される

```
create table events(prod_id number, site_id CHAR(2),start_date date)
partition by range (site_id, start_date)
subpartition by hash (prod_id) subpartitions 16
(partition below_L1      values less than ('L1',to_date('01-JAN-1492','dd-mon-yyyy')),
partition l1_2013      values less than ('L1',to_date('01-JAN-2014','dd-mon-yyyy')), values less than
partition l1_2021      ('L1',to_date('01-JAN-2020','dd-mon-yyyy')), values less than ('L1',MAXVALUE),
partition l1_max
partition below_x1     values less than ('X1',to_date('01-JAN-1492','dd-mon-yyyy')),
...
partition x4_max       values less than ('X4',MAXVALUE),
partition pmax         values less than (MAXVALUE,MAXVALUE));
```

ダミーの'BELOW\_...!'パーティションを導入して  
"下位の"未定義SITE\_IDを捕捉する

# 複数列レンジ・パーティション

わずかに異なる現実のシナリオ

複数列レンジは、3番目の（数値以外の）ディメンションの導入に使用される

```
create table events(prod_id number, site_id CHAR(2),start_date date)
partition by range (site_id, start_date)
subpartition by hash (prod_id) subpartitions 16
(partition below_l1          values less than ('L1',to_date('01-JAN-1492','dd-mon-yyyy')),
 partition l1_2020          values less than ('L1',to_date('01-JAN-2021','dd-mon-yyyy')), values less than
 partition l1_2021          ('L1',to_date('01-JAN-2022','dd-mon-yyyy')),
 partition l1_max           values less than ('L1',MAXVALUE),
 partition below_x1         values less than ('X1',to_date('01-JAN-1492','dd-mon-yyyy')),
 ...
 partition x4_max           values less than ('X4',MAXVALUE),
 partition pmax             values less than (MAXVALUE,MAXVALUE));
```

MAXVALUE 'X\_FUTURE'パーティションを導入して将来の日付を捕捉する



# 複数列レンジ・パーティション

わずかに異なる現実のシナリオ

複数列レンジは、3番目の（数値以外の）ディメンションの導入に使用される

```
create table events(prod_id number, site_id CHAR(2),start_date date)
partition by range (site_id, start_date)
subpartition by hash (prod_id) subpartitions 16
(partition below_l1      values less than ('L1',to_date('01-JAN-1492','dd-mon-yyyy')),
 partition l1_2020      values less than ('L1',to_date('01-JAN-2021','dd-mon-yyyy')), values less than
 partition l1_2021      ('L1',to_date('01-JAN-2022','dd-mon-yyyy')), values less than ('L1',MAXVALUE),
 partition l1_max        values less than ('X1',to_date('01-JAN-1492','dd-mon-yyyy')),
 partition below_x1
 ...
 partition x4_max        values less than ('X4',MAXVALUE),
 partition pmax          values less than (MAXVALUE,MAXVALUE));
```

必要に応じて、オープンエンドのSITE\_IDを捕捉する（先頭のキー列）

# パーティション化オブジェクトと 非パーティション化オブジェクト の相違点

---

# 物理属性と論理属性

---

# 物理属性と論理属性

## 論理属性

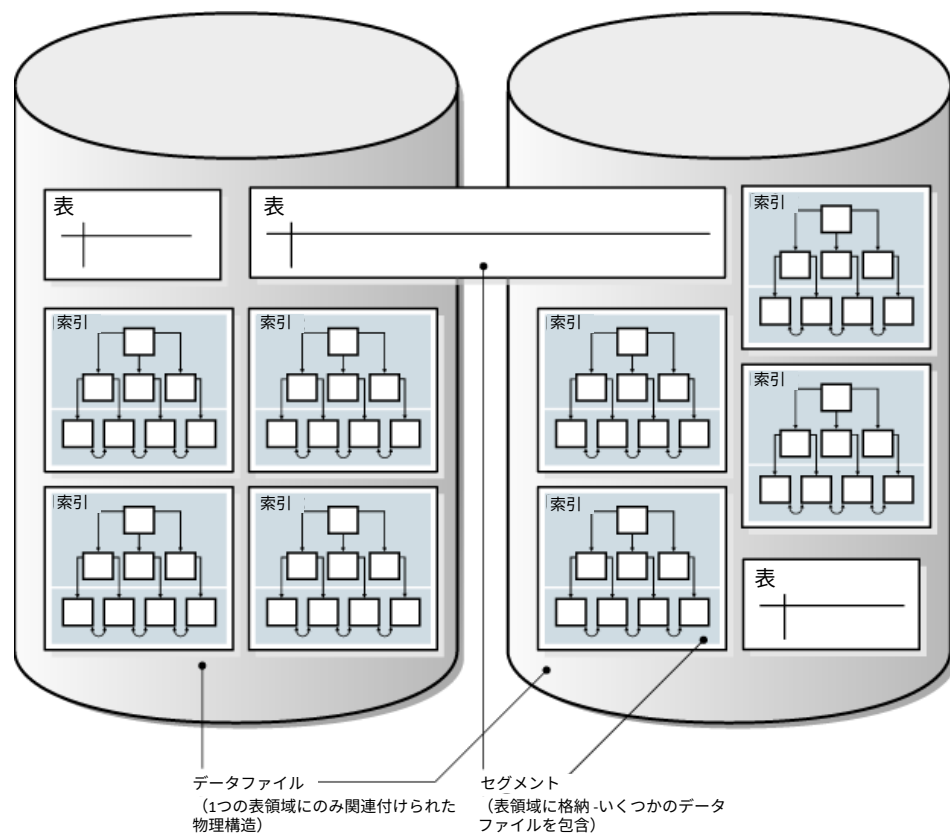
- パーティション化のセットアップ
- 索引付けと索引メンテナンス
- 読取り専用（表領域分離との組合せ）

## 物理属性

- データ配置
- 一般のセグメント・プロパティ

# 非パーティション表

## 物理属性と論理属性



### 論理表プロパティ

- 列とデータ型
- 制約
- 索引、...

### 物理表プロパティ

- セグメントと同様の表
- 表領域
- 圧縮、[Logging | nologging]、...
- インメモリ
- セグメント・レベルで管理および変更されるプロパティ



# パーティション表

## 物理属性と論理属性

表はメタデータのみで、将来のパーティションのディレクティブ

- 表レベルの物理セグメントはない
- 物理属性は、指定されていれば、新しいパーティションのディレクティブとなる

## 単一レベルのパーティション表

- パーティションはセグメントと同等
- 物理属性はパーティション・レベルで管理および変更される

## コンポジット・レベルのパーティション表

- パーティションはメタデータのみで、将来のサブパーティションのディレクティブ
- サブパーティションはセグメントと同等

## パーティション表でのデータ配置

各パーティションまたはサブパーティションは別々のオブジェクト

個々のレベルでストレージ属性を指定する

- 下位レベルの配置ポリシーとして
- 個々の[サブ]パーティションごとに

ストレージ属性を指定していない場合は標準の階層継承が作動する





# パーティション表でのデータ配置

## 特殊なケースのインターバル・パーティション化

インターバル・パーティション化ではすべてのパーティションを"事前に作成する"

- 100万の[サブ]パーティションすべてが論理的に存在する

物理ストレージも（ほとんど）決定される

### パーティション配置

- 表レベルから継承
- STORE IN ()句によりラウンドロビンでパーティション配置

### サブパーティション配置

- サブパーティション・テンプレートの使用
- STORE IN句は現在ノーオペレーション



# パーティション表でのデータ配置

## サブパーティション・テンプレート

将来のパーティションのサブパーティションの事前定義を許可する  
データ・ディクショナリ内のメタデータとして格納される

- 構文（マクロ）シュガーだけではない

```
CREATE TABLE stripe_regional_EVENTS (deptno
    number, item_no varchar2(20),
    txn_date date, txn_amount number, state varchar2(2))
PARTITION BY RANGE (txn_date)
SUBPARTITION BY LIST (state)
SUBPARTITION TEMPLATE
(SUBPARTITION northwest VALUES ('OR', 'WA') TABLESPACE tbs_1
SUBPARTITION southwest VALUES ('AZ', 'UT', 'NM') TABLESPACE tbs_2 SUBPARTITION
northeast VALUES ('NY', 'VM', 'NJ') TABLESPACE tbs_3 SUBPARTITION southeast
VALUES ('FL', 'GA') TABLESPACE tbs_4 SUBPARTITION midwest VALUES ('SD', 'WI')
TABLESPACE tbs_5 SUBPARTITION south VALUES ('AL', 'AK') TABLESPACE tbs_6
SUBPARTITION south VALUES (DEFAULT) TABLESPACE tbs_7
)
(PARTITION q1_2021 VALUES LESS THAN ( TO_DATE('01-APR-2021', 'DD-MON-YYYY')), (PARTITION
q2_2021 VALUES LESS THAN ( TO_DATE('01-JUL-2021', 'DD-MON-YYYY')), (PARTITION q3_2021 VALUES
LESS THAN ( TO_DATE('01-OCT-2021', 'DD-MON-YYYY')), (PARTITION q4_2021 VALUES LESS THAN
( TO_DATE('01-JAN-2020', 'DD-MON-YYYY')),
);
```

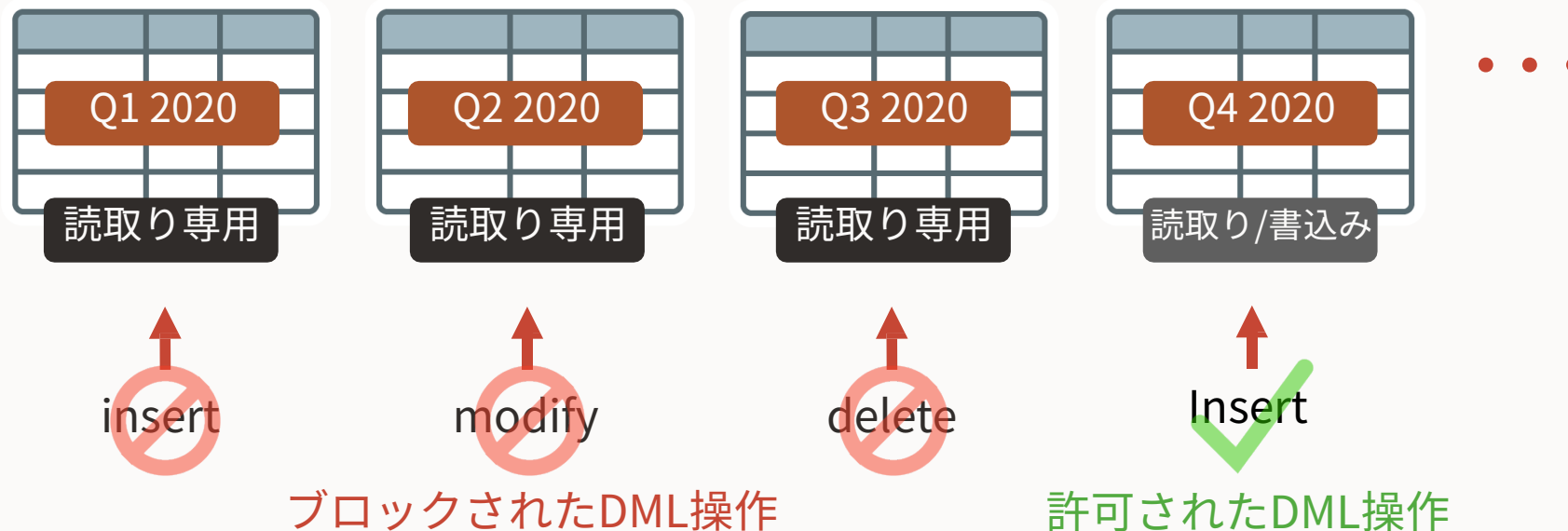
将来のパーティション  
すべてのサブパーティ  
ション定義

サブパーティション  
はすべてのパーティ  
ションに適用される

# 読取り専用パーティション

Oracle Database 12.2で導入

# 読取り専用パーティション



パーティションとサブパーティションは、読取り専用または読取り/書込みに設定することが可能  
読取り専用パーティションのデータを変更しようとする、エラーが発生する  
ユーザーまたはトリガーによる意図しないDMLからデータを保護する場合に理想的

## 読取り専用パーティションの詳細

読取り専用属性によりデータの不変性を保証

- “SELECT <column\_list> FROM <table>”は、表または[サブ]パーティションが読取り専用設定された後に、必ず同じデータセットを返す

指定しない場合、各パーティションとサブパーティションは最上位の親から読取り専用プロパティを継承する

- 下位レベルの読取り専用プロパティを変更すると、上位レベルのプロパティをオーバーライドする
- 表領域の変更は最高の優先度で、上書きできない

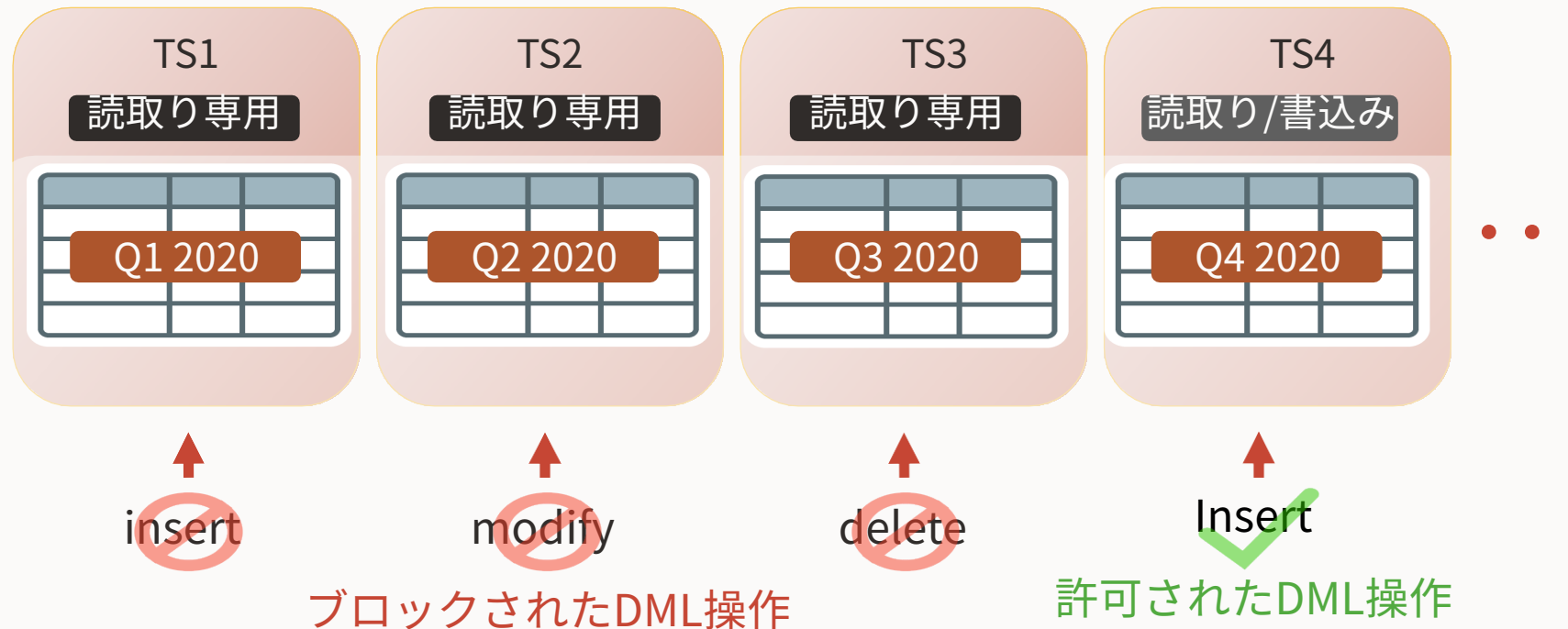
データの不変性によって、表のすべてのDDLが防止されるわけではない

- ADDおよびMODIFY COLUMNは許可されており、既存データのデータ不変性に違反しない
- DROP/RENAME/SET UNUSED COLUMNのような他の操作は禁止されている
- DROP [読取り専用] PARTITIONも禁止されており、表のデータ不変性に違反する

## 読取り専用パーティション

```
CREATE TABLE events ( event_id number,  
                       evnt_date DATE,      ... ) read only  
  
PARTITION BY RANGE(event_date)  
( partition    q1_2020 values less than ('2020-04-01'),  
  partition    q2_2020 values less than ('2020-07-01'),  
  partition    q3_2020 values less than ('2020-10-01'),  
  partition    q4_2020 values less than ('2021-01-01') read write  
);
```

## 読取り専用の表領域とパーティション



パーティションとサブパーティションは、読取り専用表領域に配置可能  
読取り専用表領域でデータを変更しようとする、エラーが発生する

# 読取り専用パーティション



パーティションとサブパーティションは、読取り専用または読取り/書込みに設定することが可能  
読取り専用パーティションでデータを変更しようとする、エラーが発生する



## 読取り専用オブジェクトと読取り専用表領域

読取り専用表領域は、物理ストレージを更新から保護する

- ストレージに影響しないDDL操作は許可される
  - 例：ALTER TABLE SET UNUSED、DROP TABLE
- データの不変性は保証されない

読取り専用オブジェクトは、データを更新から保護

- ‘データの不変性’
- ストレージでの変更を阻止しない
  - 例：ALTER TABLE MOVE COMPRESS, ALTER TABLE MERGE PARTITIONS

# 読取り専用パーティション

読取り専用属性によりデータの不変性を保証

- “SELECT <column\_list> FROM <table>”は、表または[サブ]パーティションが読取り専用設定された後に、必ず同じデータセットを返す

データの不変性によって、表のすべてのDDLが防止されるわけではない

- ADDおよびMODIFY COLUMNは許可されており、既存データのデータ不変性に違反しない
- DROP/RENAME/SET UNUSED COLUMNのような他の操作は禁止されている
- DROP [読取り専用] PARTITIONも禁止されており、表のデータ不変性に違反する

# DDLでのカーソル無効化の減少

Oracle Database 12.2で導入

## DDLでのカーソル無効化の減少

DDLによるハード解析の数を削減する

- ハード解析を避けられない場合、ワークロードは時間の経過とともに分散される
- いくつかのDDLの場合の新しいオプション句“[ DEFERRED | IMMEDIATE ] INVALIDATION”
- DEFERREDの場合、可能であれば、Oracleでは依存カーソルの無効化を回避する
- IMMEDIATEの場合は、Oracleでは依存カーソルを直ちに無効化する
- どちらでもない場合、CURSOR\_INVALIDATIONパラメータによってデフォルト動作を制御する

サポートされるDDL：

- 索引の作成、削除、変更
- 表列の変更操作
- 表セグメントの変更操作
- 表の削除

# DDLでのカーソル無効化の減少 構文例

```
DROP INDEX meas_campaign DEFERRED INVALIDATION;
```

# パーティション化の統計管理

---

# 統計収集

オプティマイザ統計を収集しなければならない

- 動的サンプリングの使用は適切な解決策ではない
- グローバルおよびパーティション・レベルの統計が推奨される
  - サブパーティション・レベルはオプション

空の表に対してすべての問合せを実行して、列の使用状況を取り込む

- これにより、作成されるヒストグラムを自動的に取得する列を識別できる

オプティマイザ統計は、データがロードされた後、かつ索引が作成される前に収集する必要がある

- Oracleでは、索引の作成時にその統計を自動的に収集する

# 統計収集

デフォルトでは、DBMS\_STATSが表ごとに次の統計を収集する

- グローバル（表レベル）、パーティション・レベル、サブパーティション・レベル

問合せに2つ以上のパーティションが関係する場合、オプティマイザはグローバル統計を使用する

問合せでパーティション・エリミネーションが実行され、  
問合せに答える必要のあるパーティションが1つだけの場合、オプティマイザはパーティション統計を使用する

- 問合せに2つ以上のパーティションが関係する場合、オプティマイザはグローバルとパーティション・レベルの統計の組合せを使用する

問合せでパーティション・エリミネーションが実行され、1つのサブパーティションが問合せに答える必要がある場合にのみ、  
オプティマイザはサブパーティション・レベルの統計を使用する



# 効率的な統計管理

## AUTO\_SAMPLE\_SIZEを使用する

- 新しい効率的な統計収集を可能にする唯一の設定
- ハッシュ・ベースのアルゴリズム、表全体のスキャン
  - サンプルング速度、計算の正確さ

## 増分グローバル統計収集を可能にする

- 単一パーティションの変更後の全パーティションのスキャンを回避する
  - 11.1より前、グローバル統計では全パーティションのスキャンを必要とした
- 表レベル単位で管理される
  - 統計設定
- 非パーティション表を交換するためのシノプシスを作成する (Oracle Database 12c)

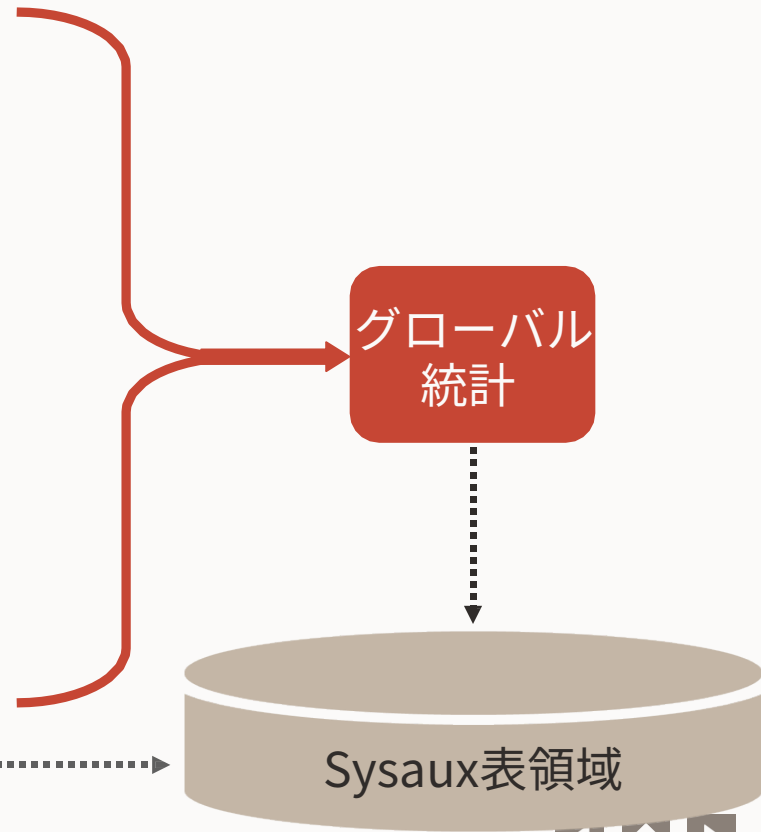
# 増分グローバル統計



1.パーティション・レベルの統計が収集され、シノプシスが作成される



2.パーティション・シノプシスを収集することによって生成されるグローバル統計



## 増分グローバル統計 (続き)

### EVENTS表

2021年5月18日

2021年5月19日

2021年5月20日

2021年5月21日

2021年5月22日

2021年5月23日

2021年5月24日

3.新しいパーティションが表に追加され、データがロードされる

4.新しいパーティションのパーティション統計を収集する



S7

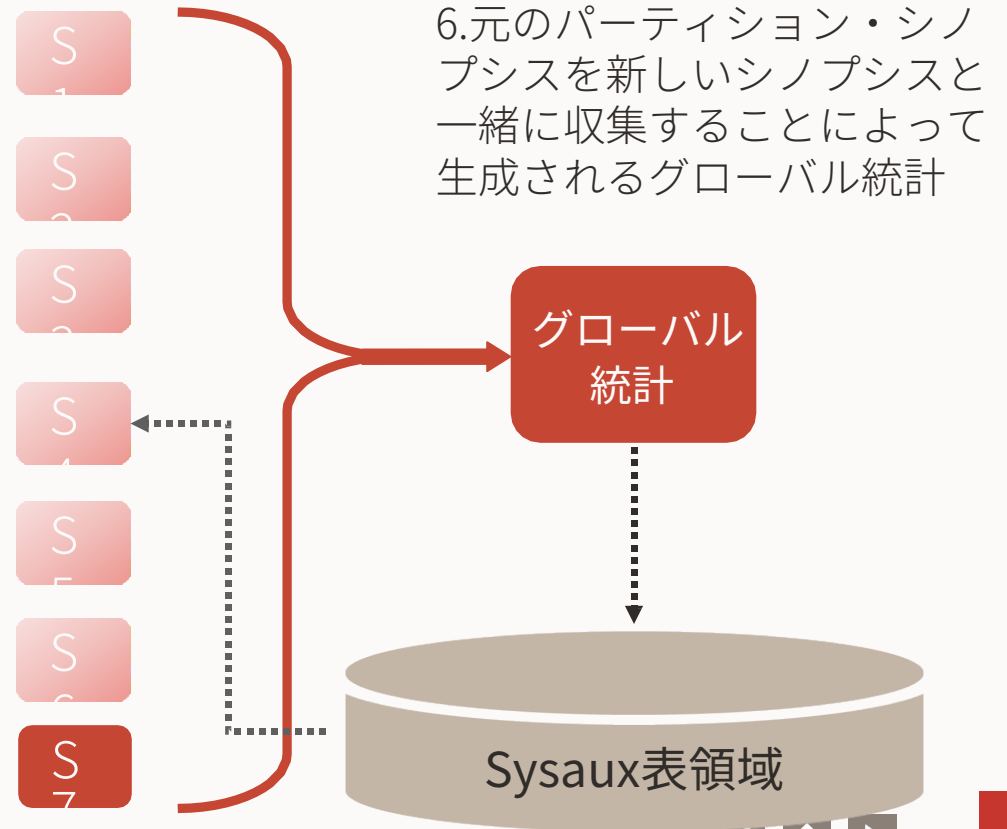


Sysaux表領域

## 増分グローバル統計 (続き)



5.Sysauxから他の各パーティションのシノプシスを取得する



## 正確な統計を収集するために必要な手順

表の増分機能をオンにする

```
EXEC DBMS_STATS.SET_TABLE_PREFS('ATLAS','EVENTS','INCREMENTAL','TRUE');
```

ロード後に、GATHER\_TABLE\_STATSを使用して表統計を収集する

- パラメータを指定する必要はない

```
EXEC DBMS_STATS.GATHER_TABLE_STATS('ATLAS','EVENTS');
```

このコマンドにより、パーティションの統計が収集され、パーティション・レベルの統計とシナプシスに基づいてグローバル統計が更新される

すべての表の場合に増分をtrueに設定可能

- 既存の表でのみ有効

```
EXEC DBMS_STATS.SET_GLOBAL_PREFS('INCREMENTAL','TRUE');
```

# ベスト・プラクティスとハウツー

---

# パーティション化戦略について

---

# パーティション化戦略の選択

## 考慮事項

- データ
- 使用状況

## パーティション化に期待すること

- 問合せパフォーマンスの利点
- ロード（またはページ）パフォーマンスの利点
- データ管理の利点



# パーティション化戦略の選択

## データの論理形態

システムへのデータの挿入方法は?

システム内でのデータのメンテナンス方法は?

システム内のデータへのアクセス方法は?

# パーティション化戦略の選択

## データの論理形態

---

システムへのデータの挿入方法は?

- 時間、場所、テナント、ビジネス・ユーザー、...
- 範囲、関連性のない値リスト、"とにかく量が多い場合"、...

システム内でのデータのメンテナンス方法は?

システム内のデータへのアクセス方法は?

# パーティション化戦略の選択

## データの論理形態

システムへのデータの挿入方法は?

- 時間、場所、テナント、ビジネス・ユーザー、 ...
- 範囲、関連性のない値リスト、"とにかく量が多い場合"、 ...

システム内でのデータのメンテナンス方法は?

- アクティブなデータの時間枠の移動、法的要件、"永続"データ、 ...
- 未定

システム内のデータへのアクセス方法は?

# パーティション化戦略の選択

## データの論理形態

システムへのデータの挿入方法は?

- 時間、場所、テナント、ビジネス・ユーザー、...
- 範囲、関連性のない値リスト、"とにかく量が多い場合"、...

システム内でのデータのメンテナンス方法は?

- アクティブなデータの時間枠の移動、法的要件、"永続"データ、...
- 未定

システム内のデータへのアクセス方法は?

- 常に全表、一般的FILTER条件を使用、常に索引アクセス、...
- 未定

# パーティション化戦略の選択

## パフォーマンスの改善

### 問合せの高速化

- パーティション・エリミネーション
- パーティション・ワイズ結合

### DMLの高速化

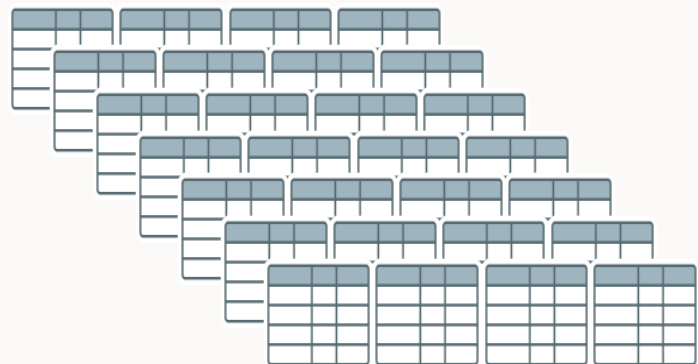
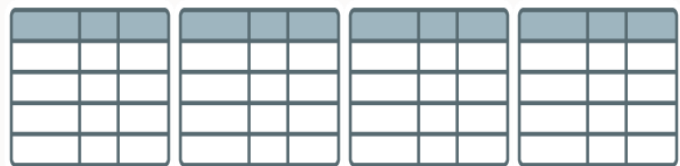
- 競合ポイントの緩和

### データ・メンテナンス

- DMLではなくDDL

# パーティション化戦略の選択

## データ・アクセス - 全表アクセス



I/Oの削減量はプルーニングされるパーティションの数に比例する

- 1/10 : IOが10倍減少
- 1/100 : IOが100倍減少

実行時間の短縮要因

- CPUの動作に対するIOの相対的な貢献割合
- 後続の操作への潜在的な影響

# パーティション化戦略の選択

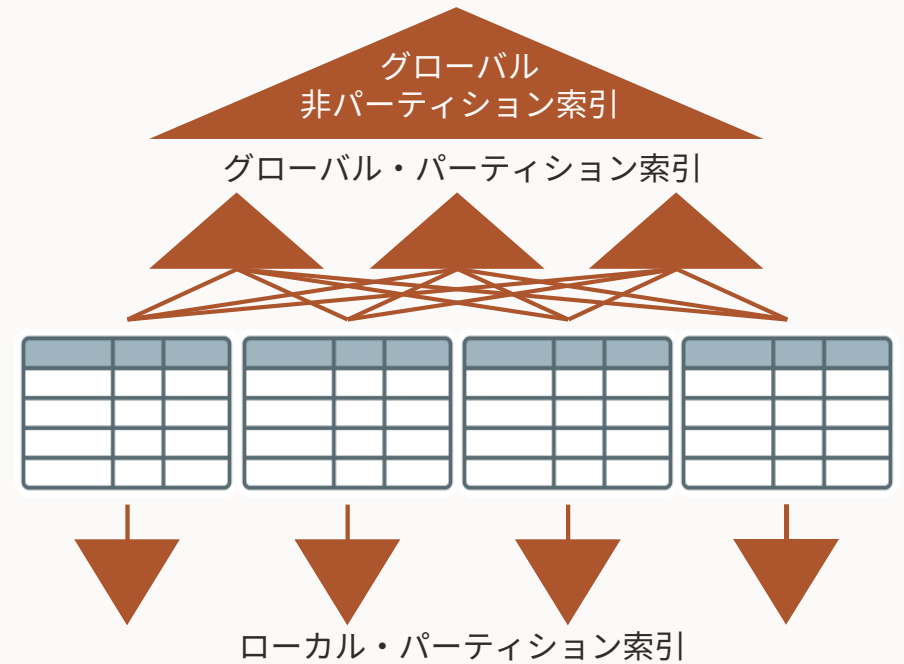
## パーティション表の索引付け

グローバル索引は任意のパーティションの行を参照する

- 索引はパーティション化してもしなくても構わない

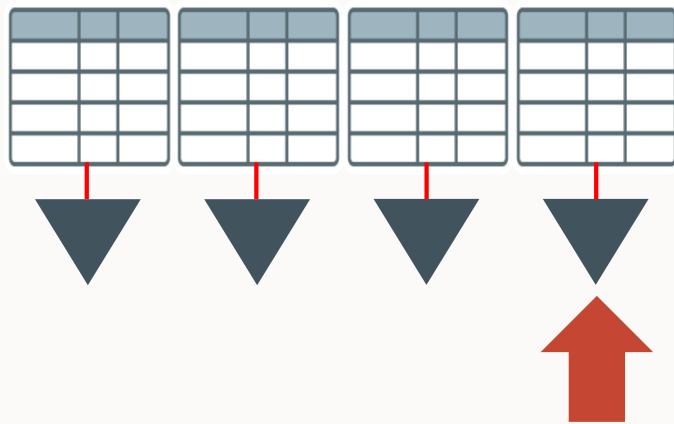
ローカル索引は、表と同様にパーティション化される

- 索引パーティション化キーは、索引キーと異なる場合がある

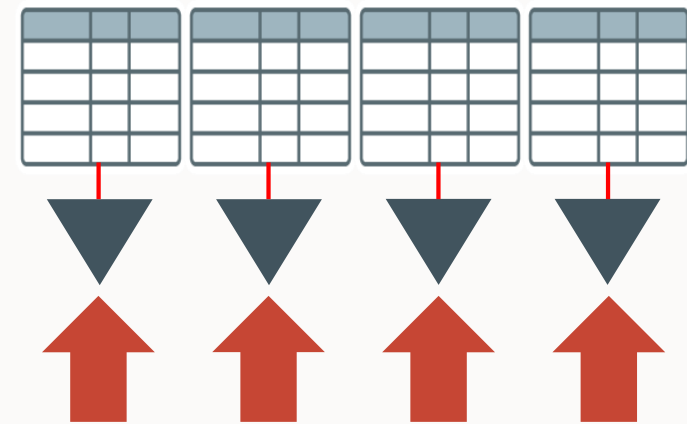


# パーティション化戦略の選択

データ・アクセス - ローカル索引とグローバル・パーティション索引



単一パーティション・プルーニングありの  
パーティション索引アクセス



パーティション・プルーニングなしの  
パーティション索引アクセス



# ローカルとグローバルのパーティション索引

## データ・アクセス

アクセスされたパーティションの数と同一の索引プローブの数

- すべての索引パーティションをプローブするパーティション・プルーニングなし

OLTP環境向けに最適化されていない

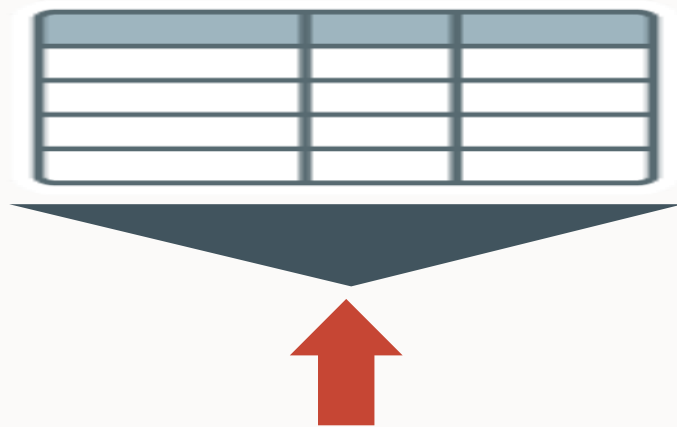
- 常にパーティション・プルーニングがあるという保証はない
- 例外：DML競合緩和のためのグローバル・ハッシュ・パーティション索引
  - ごく一般的に少数のパーティション

索引接頭辞に基づくグローバル・パーティション索引でのプルーニング

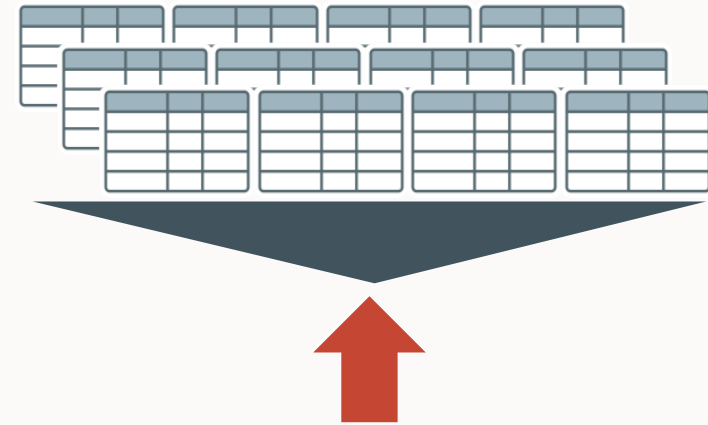
- 索引の先頭キーと同一の索引接頭辞

# パーティション化戦略の選択

## グローバル非パーティション索引

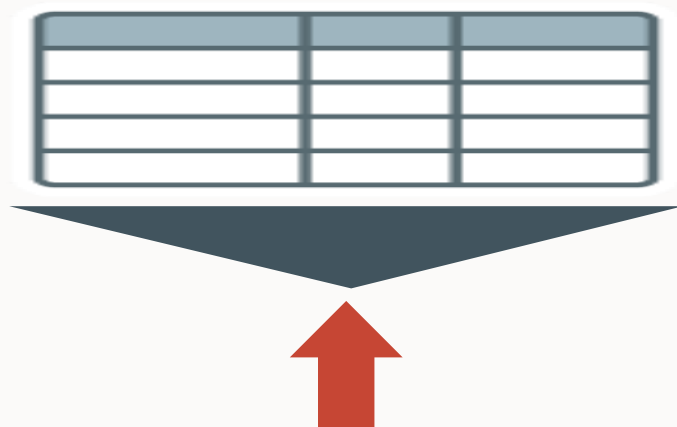


違いは何か?

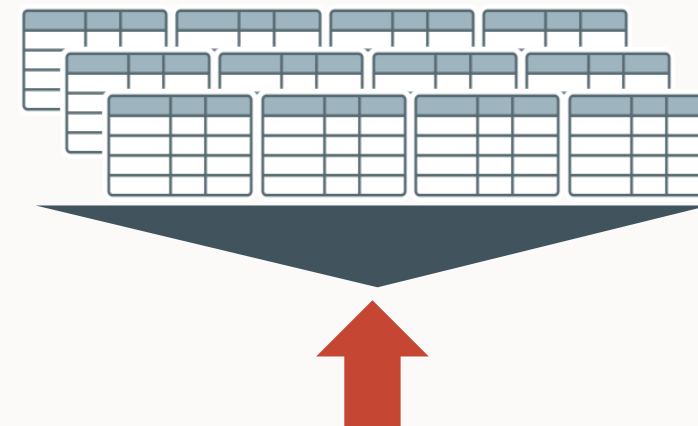


# パーティション化戦略の選択

## グローバル非パーティション索引



違いは何か?  
ほとんどない\*



\* rowidが大きいため、索引サイズに多少の違いがある

# グローバル索引

## データ・アクセス

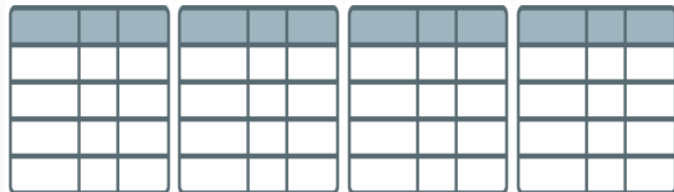
非パーティション索引ではプルーニングなし

- 常に1つの索引セグメント内をプローブする

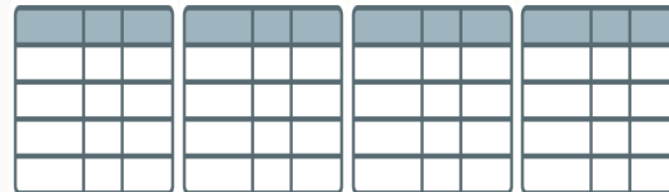
グローバル・パーティション索引接頭辞は、索引の先頭キーと同一

- パーティション・キー列ではなく、索引接頭辞でプルーニング

OLTP環境でもっとも一般的



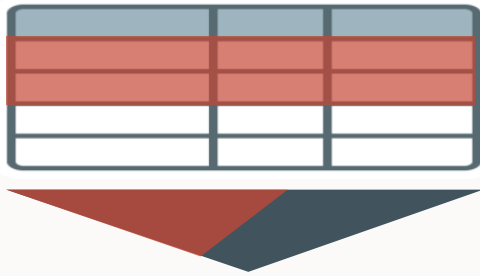
PARTITION BY (col1), idx(col1)



PARTITION BY (col1), idx(col2)

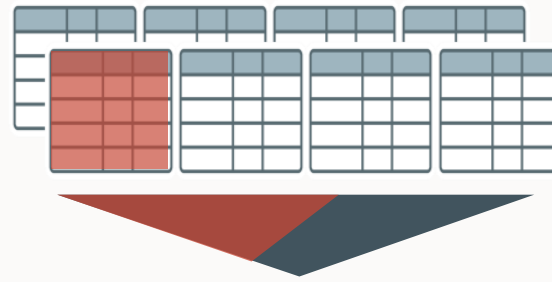
# パーティション化戦略の選択

## データ・メンテナンス



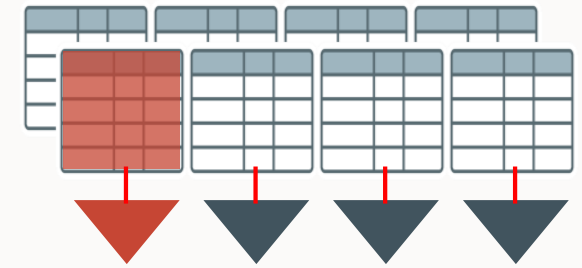
DELETE FROM ...  
WHERE ...

- レコードが削除される
  - 索引メンテナンス
  - 元に戻すとやり直し



ALTER TABLE ...DROP PARTITION ...

- パーティションが削除される
  - 高速なグローバル索引メンテナンス (12c)
  - 最小限の元に戻す



- パーティションが削除される
  - ローカル索引が削除される
  - 最小限の元に戻す

# ローカル索引

## データ・メンテナンス

### 増分索引作成が可能

- 最初は使用不可索引の作成、個々のパーティションの再構築

パーティションが1つだけ関係するすべてのパーティション・メンテナンス操作で高速索引メンテナンス

- 交換、削除、切捨て

複数のパーティションが関係するパーティション・メンテナンスでは、索引メンテナンスが必要

- マージと分割によって新しいデータ・セグメントを作成する
- 新しい索引セグメントも作成される

# グローバル索引

## データ・メンテナンス

増分索引作成は、不可能ではないものの難しい

削除と切捨での場合の"高速"索引メンテナンスはOracle Database 12c以降導入されている

- 高速索引メンテナンスは実際には遅延索引メンテナンスを意味する

削除と切捨を除いたパーティション・メンテナンスでは索引メンテナンスを必要とする

- 従来の索引メンテナンスは、PMOPを表すDML操作と同等

# 必要なパーティションの数は?

その決定要因



# データ量とパーティション数

---

100 TBの表の場合...

- パーティション数が100万であれば、各パーティションのサイズは100 MB

10 TBの表の場合...

- パーティション数が100万であれば、各パーティションのサイズは10 MB

1 TBの表の場合...

- パーティション数が100万であれば、各パーティションのサイズは1 MB

# データ量とパーティション数

100 TBの表の場合...

- パーティション数が100万であれば、各パーティションのサイズは100 MB

10 TBの表の場合...

- パーティション数が100万であれば、各パーティションのサイズは10 MB

1 TBの表の場合...

- パーティション数が100万であれば、各パーティションのサイズは1 MB

システムで1 MBを読み取るのに要する時間は？

- Oracle Exadataの全表スキャン速度は、数十から数百GB/秒...

# データ量とパーティション数

多ければ良いとは限らない

- すべてのパーティションは、ディクショナリ内のメタデータを表す
- すべてのパーティションのSGAで、メタデータ・フットプリントが増大する

パーティション数とその平均サイズの間で、自分にとって最適なバランスを見つける

- "通常システム"のセグメントの場合は、GBの値が1桁でも問題ない
- セグメント・サイズが5 GB以上のパーティションを増やすことを検討する

# パーティション化戦略の選択

## 顧客の使用状況パターン

レンジ（インターバル）は、引き続きもっとも一般的なパーティション化戦略

- ほとんどいつでも、何らかの時間依存性を示す

リストはますます一般化している

- 興味深いことに、多くの場合に時間にも基づく
- 多くの場合、サブパーティション化戦略として

ハッシュは、パフォーマンス（PWJ、DML競合）のために使用されるだけではない

- データ配置を制御することはできないが、ある程度理解している
- 2の累乗ルールを忘れないこと

# パーティション化戦略の選択

## 拡張パーティション化戦略

---

インターバル・パーティション化 - 成長著しい新しいパーティション化戦略

- レンジ・パーティション化に対する管理性拡張機能

参照パーティション化

- データ・モデルでPK/FK制約を利用する

インターバル参照パーティション化 (Oracle Database 12cでの新機能)

バーチャル・カラム・ベースのパーティション化

- アプリケーション変更がほとんどまたは全くない派生属性

上記のすべてのバリエーション

# 柔軟性には代価が伴う

---

# Oracle Partitioningでの柔軟性

100万個のパーティション - 多ければ良いか?

オンライン操作 - 聖杯か?

PMOPはいつでもDMLで実行されるか?

# データ量とパーティション数

---

100 TBの表の場合...

- パーティション数が100万であれば、各パーティションのサイズは100 MB

10 TBの表の場合...

- パーティション数が100万であれば、各パーティションのサイズは10 MB

1 TBの表の場合...

- パーティション数が100万であれば、各パーティションのサイズは1 MB



# データ量とパーティション数

100 TBの表の場合...

- パーティション数が100万であれば、各パーティションのサイズは100 MB

10 TBの表の場合...

- パーティション数が100万であれば、各パーティションのサイズは10 MB

1 TBの表の場合...

- パーティション数が100万であれば、各パーティションのサイズは1 MB

システムで1 MBを読み取るのに要する時間は？

- Oracle Exadataの全表スキャン速度は、数十から数百GB/秒...

# 100万個のパーティション - 多ければ良いか?

## 多ければ良いとは限らない

- すべてのパーティションは、ディクショナリ内のメタデータを表す
- すべてのパーティションのSGAで、メタデータ・フットプリントが増大する
- パーティションの数が大量だと、カタログ・ビューのパフォーマンスに影響する可能性がある

## パーティション数とその平均サイズの間で、自分にとって最適なバランスを見つける

- "通常システム"のセグメントの場合は、GBの値が1桁でも問題ない
- セグメント・サイズが5 GB以上のパーティションを増やすことを検討する

## 表およびパーティションのオンライン（データ移動）操作

パーティション・メンテナンス操作（PMOP）はオンライン

- 移動：場所属性とストレージ属性を変更する
- マージ：多くのパーティションが1つになる
- 分割：1つのパーティションが多くのパーティションになる

表変換操作はオンライン

- パーティション表となるように非パーティション表に変更を加える
- パーティション表の形態を変更する

すべてのオンライン操作が索引メンテナンスに対応する

# 表およびパーティションのオンライン（データ移動）操作

可能な範囲で最適な時間枠を計画する

オンライン操作によってアプリケーションの透過性を維持し、ビジネスへの影響を最小限に抑える

- アプリケーションのワークフローと設計について考えることを止めるために導入されたのではない

## オンライン操作のコストは同時実行性に伴って増大する

可能な場合は、同時実行DML操作を最小限に抑える

- ジャーナリング用に追加のディスク領域とリソースが必要
- ジャーナルは初期一括移動後に再帰的に適用される
- ジャーナルが大きいほど実行時間が長くなる

同時DMLは圧縮効率に影響する

- 初期一括移動による最良の圧縮比

## PMOPはいつでもDMLで実行されるか?

パーティション・メンテナンス操作は、データをロードまたはアンロードするための高速で効率的な方法

...しかし、代価が伴う

- パーティション・メタデータを更新するための再帰的DML
  - 関係するパーティション（表および索引）の数にもっとも一般的に比例する、例外あり
- カーソル無効化
  - よりきめ細かい無効化と増分メタデータ無効化/リフレッシュを行うことに努力する

## PMOPはいつでもDMLで実行されるか?

パーティション・メンテナンス操作は、データをロードまたはアンロードするための高速で効率的な方法

...しかし、代価が伴う

- パーティション・メタデータを更新するための再帰的DML
  - 関係するパーティション（表および索引）の数にもっとも一般的に比例する、例外あり
- カーソル無効化
  - よりきめ細かい無効化と増分メタデータ無効化/リフレッシュを行うことに努力する

DMLは実行可能な代替手段

- 特にデータ量が少ない場合

# パーティション化を使用した ホットスポットの発生回避

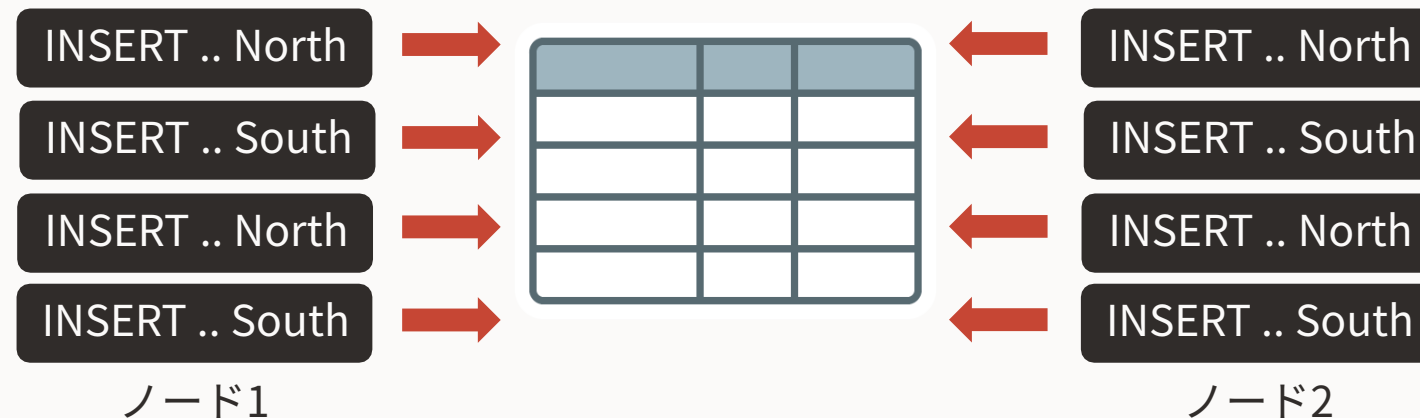
---

# パーティション化を使用したホットスポットの発生回避

## 非パーティション表

Oracle RACでは、高DMLワークロードにより大きなキャッシュ・フュージョン・トラフィックが発生する

- Oracleではこのブロックをpingと呼ぶ





# パーティション化を使用したホットスポットの発生回避

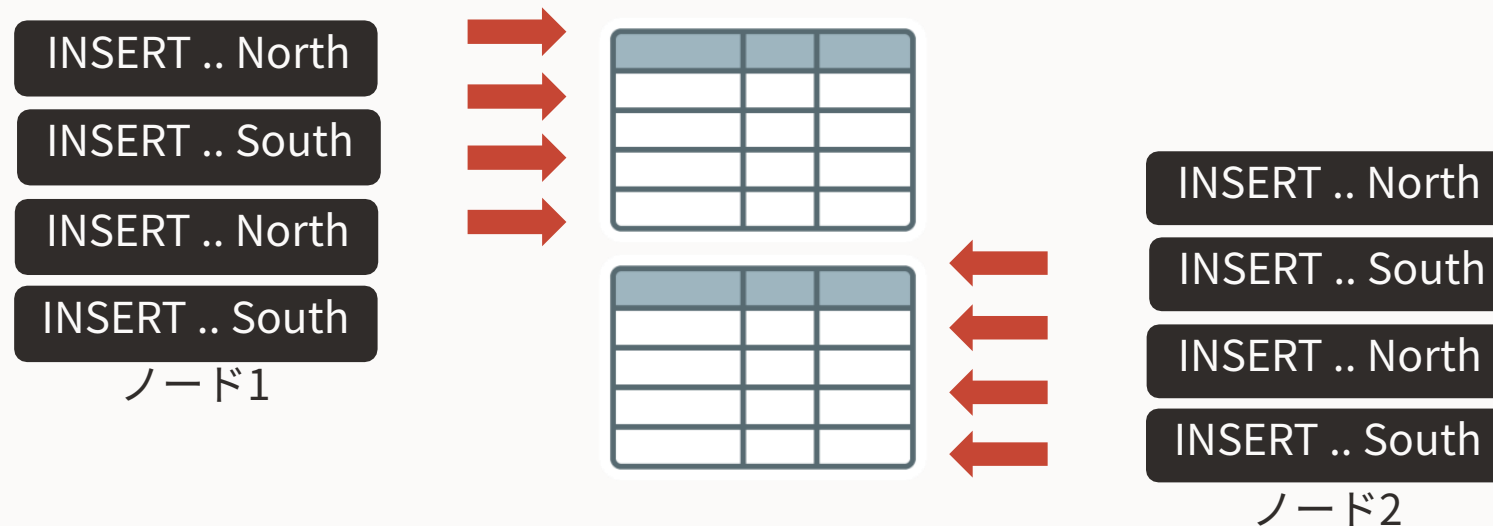
## ハッシュ・パーティション表

Oracle RACでは、高DMLワークロードにより大きなキャッシュ・フュージョン・トラフィックが発生する

- Oracleではこのブロックをpingと呼ぶ

ハッシュ（またはリスト）パーティション表によってこの状況を緩和できる

- 注意事項：通常、ある種の"アプリケーション・パーティション化"または"アプリケーションOracle RAC認識"が必要

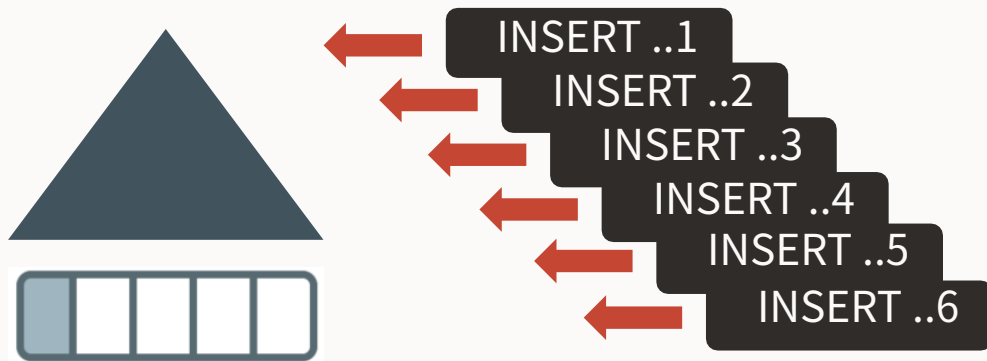


# パーティション化を使用したホットスポットの発生回避

## ハッシュ・パーティション索引

高DMLワークロードにより、索引ブロックでホットスポット（競合）が作成される可能性がある

- 例：人為的（右上がり）主キー索引



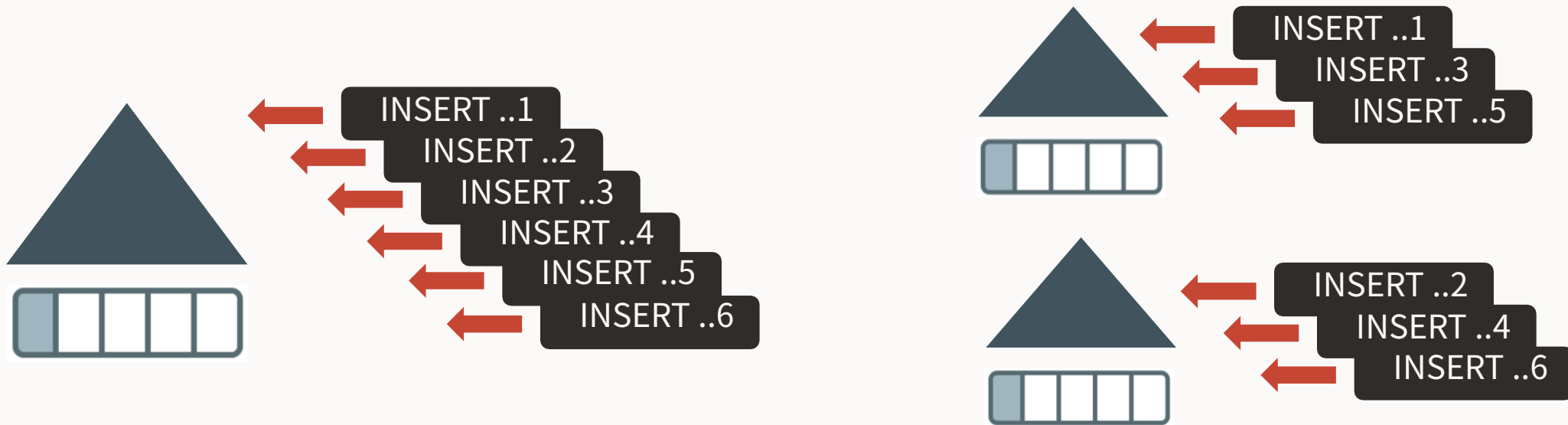
# パーティション化を使用したホットスポットの発生回避

## ハッシュ・パーティション索引

高DMLワークロードにより、索引ブロックでホットスポット（競合）が作成される可能性がある

- 例：人為的（右上がり）主キー索引

ハッシュ・パーティション索引により、ウォーム・スポットが発生する



# ホットスポットの発生回避 - ユースケース

## 課題

オブジェクト・リレーショナル・マッピングを使用する小売アプリケーション

"一般的な"データベース機能のみが使用される

1行ごとにすべて1回のトランザクションで更新する必要がある

一括インポートは一切できない

千単位の小さなSQL文が発行される

ユーザー・アクセスにおける突然の高負荷ピーク

- 例：サイバー・マンデー、クリスマス・セール、特別セール、...

散発的競合の発生

# ホットスポットの発生回避 - ユースケース

アプリケーション・コードの変更なしでのパフォーマンス

## PoC (SKUデータ・ロード) の結果

リファレンス・システム：1秒あたり120 SKU

Exadataマシン (単一ノード・ロード)

- 1秒あたり2,500 SKU (20倍高速)

ExadataマシンX3-2 (2ノード・ロード、パーティション化なし)

- 1秒あたり"わずか"1,900 SKU (シングル・ノード・ロードより低速)

ExadataマシンX3-2 (2ノード・ロード、適切なパーティション化あり)

- 1秒あたり4,800 SKU (40倍高速)

適切なパーティション化により線形スケールリングが可能

# ホットスポットの発生回避 - ユースケース

方法（代替方法A、ストアIDでのハッシュ・パーティション化）

ハッシュ・パーティション化では表内に<n>個のエントリ・ポイントが作成される

```
CREATE TABLE <table_name> (  
  ID          NUMBER(10) NOT NULL,  
  Cn          ...)  
PARTITION BY HASH(ID) PARTITIONS <n>  
TABLESPACE <tablespace_name> STORAGE ( ... );  
  
CREATE UNIQUE INDEX <index_name> ON <table_name>  
(ID) LOCAL TABLESPACE <tablespace_name> STORAGE ( ... );  
  
INSERT INTO <table_name> (ID, ...)  
SELECT SEQ_ID.nextval, ... ;
```

# ホットスポットの発生回避 - ユースケース

方法（代替方法B、インスタンス#でのリスト・パーティション化）

シーケンスSEQ\_IDにより、各パーティションでIDが強制的に一意の値にされる  
リスト・パーティション化では、インスタンスごとにエン트리・ポイントを完全に分離する

```
CREATE TABLE <table_name> (  
  ID          NUMBER(10) NOT NULL,  
  Cn          ...  
  INSTANCE_NUMBER NUMBER(1) DEFAULT sys_context('USERENV','INSTANCE') NOT NULL)  
PARTITION BY LIST (INSTANCE_NUMBER)  
( PARTITION P1 VALUES(1),  
  PARTITION P2 VALUES(2),  
  ...  
  PARTITION Pn VALUES(n))  
TABLESPACE <tablespace_name> STORAGE (...);  
  
CREATE UNIQUE INDEX <index_name> ON <table_name>  
(ID, INSTANCE_NUMBER) LOCAL TABLESPACE <tablespace_name> STORAGE (...);  
  
INSERT INTO <table_name> (ID, ...)SELECT SEQ_ID.nextval, ... ;
```

# ホットスポットの発生回避 - ユースケース

方法（強化代替方法B、インスタンス#でのハッシュ・パーティション化）

シーケンスSEQ\_IDにより、各パーティションでIDが強制的に一意の値にされる

```
CREATE TABLE <table_name> (  
  ID          NUMBER(10) NOT NULL,  
  Cn          ...  
  INSTANCE_NUMBER NUMBER(1) DEFAULT sys_context('USERENV','INSTANCE') NOT NULL)  
PARTITION BY LIST (INSTANCE_NUMBER)  
SUBPARTITION BY HASH (ID) SUBPARTITIONS <m>  
( PARTITION P1 VALUES(1),  
  PARTITION P2 VALUES(2),  
  ...  
  PARTITION Pn VALUES(n))  
TABLESPACE <tablespace_name> STORAGE ( ... );  
CREATE UNIQUE INDEX <index_name> ON <table_name>  
(ID, INSTANCE_NUMBER) LOCAL TABLESPACE <tablespace_name> STORAGE ( ... );  
INSERT INTO <table_name> (ID, ...)SELECT SEQ_ID.nextval, ... ;
```



## 最適技術の発見

すべてのインスタンスにおいて、高負荷の並列挿入操作でスケーリングする

### 逆キー索引

レンジ・スキャンは使用不可に

### ハッシュ・パーティション索引

右上がり索引のホットスポットを緩和する

引き続き、表ブロックでの同時実行性と索引ブロックのブロックping

### ローカル索引ありのハッシュ・パーティション表

大幅な改善、ただし引き続きXインスタンス挿入での同時実行性

### インスタンス別のコンポジット・リストとローカル索引ありのハッシュ・サブパーティション

最適ソリューション、同時実行性を"排除"し、ロード・ジョブを線形にスケーリング

# スマートな部分パーティション交換

強化された"フィルタ付きパーティション・メンテナンス"

## ロードとページのためのパーティション交換

メタデータ限定操作としてデータを削除および追加する

- パーティションと表のメタデータを交換する

データ・ロード：スタンドアロン表にはロード中の新規データが格納されるのに対し、交換用のパーティションは通常空

データ・ページ：データが格納されているパーティションが空の表と交換される

ページの代替としてパーティションを削除する

- データは完全に消去される

<TABLE>



EVENTS 表
2021年5月18日
2021年5月19日
2021年5月20日
2021年5月21日
2021年5月22日
2021年5月23日

# スマートな部分パーティション交換

簡単そうだが...

パーティション境界が100%そろっていない場合はどうするか?

- "部分ページ"

ユースケース

- 日をまたぐ通話
- 未払いの古い注文
- 未配達 of 古い注文
- その他の"レコードありで完了していない"シナリオ

# スマートな部分パーティション交換

## 部分パーティション

パーティションをパーティション中にロックする

```
LOCK TABLE ...PARTITION ...
```

### EVENTS 表

2021年5月18日

2021年5月19日

2021年5月20日

2021年5月21日

2021年5月22日

2021年5月23日

# スマートな部分パーティション交換

## 部分パーティション

パーティションをパーティション中にロックする

```
LOCK TABLE ...PARTITION ...
```

残りのデータセットが格納されている表を作成する

- 条件が複雑になり、複数の表が関係する可能性がある

```
CREATE TABLE ...AS SELECT WHERE ...
```

“残り”



EVENTS  
表

2021年5月18日

2021年5月19日

2021年5月20日

2021年5月21日

2021年5月22日

2021年5月23日

# スマートな部分パーティション交換

## 部分パーティション

パーティションをパーティション中にロックする

```
LOCK TABLE ...PARTITION ...
```

残りのデータセットが格納されている表を作成する

- 条件が複雑になり、複数の表が関係する可能性がある

```
CREATE TABLE ...AS SELECT WHERE ...
```

必須の索引があれば作成する

“残り”



EVENTS  
表

2021年5月18日

2021年5月19日

2021年5月20日

2021年5月21日

2021年5月22日

2021年5月23日

# スマートな部分パーティション交換

## 部分パーティション

パーティションをパーティション中にロックする

```
LOCK TABLE ...PARTITION ...
```

残りのデータセットが格納されている表を作成する

- 条件が複雑になり、複数の表が関係する可能性がある

```
CREATE TABLE ...AS SELECT WHERE ...
```

必須の索引があれば作成する

パーティションを交換する

```
ALTER TABLE ...EXCHANGE PARTITION ...
```

2021年5月18日



EVENTS  
表

2021年5月18日

2021年5月19日

2021年5月20日

2021年5月21日

2021年5月22日

2021年5月23日



# 一意キーと主キーの制約が 存在する中での交換

---

# 一意制約/主キー

一意制約は一意索引によって適用される

- 主キー制約によってNOT NULLが列に追加される
- 表には主キー ("一意の識別子") を1つだけ指定できる

パーティション表には2つのタイプの索引がある

- ローカル索引
- グローバル索引、パーティションと非パーティションの両方

# パーティション交換


パーティションのロードとページとも呼ばれる

メタデータ限定操作としてデータを削除および追加する

- パーティションのメタデータを交換する

両方の表で論理形態が同じであることが正常な交換のための必須の前提条件

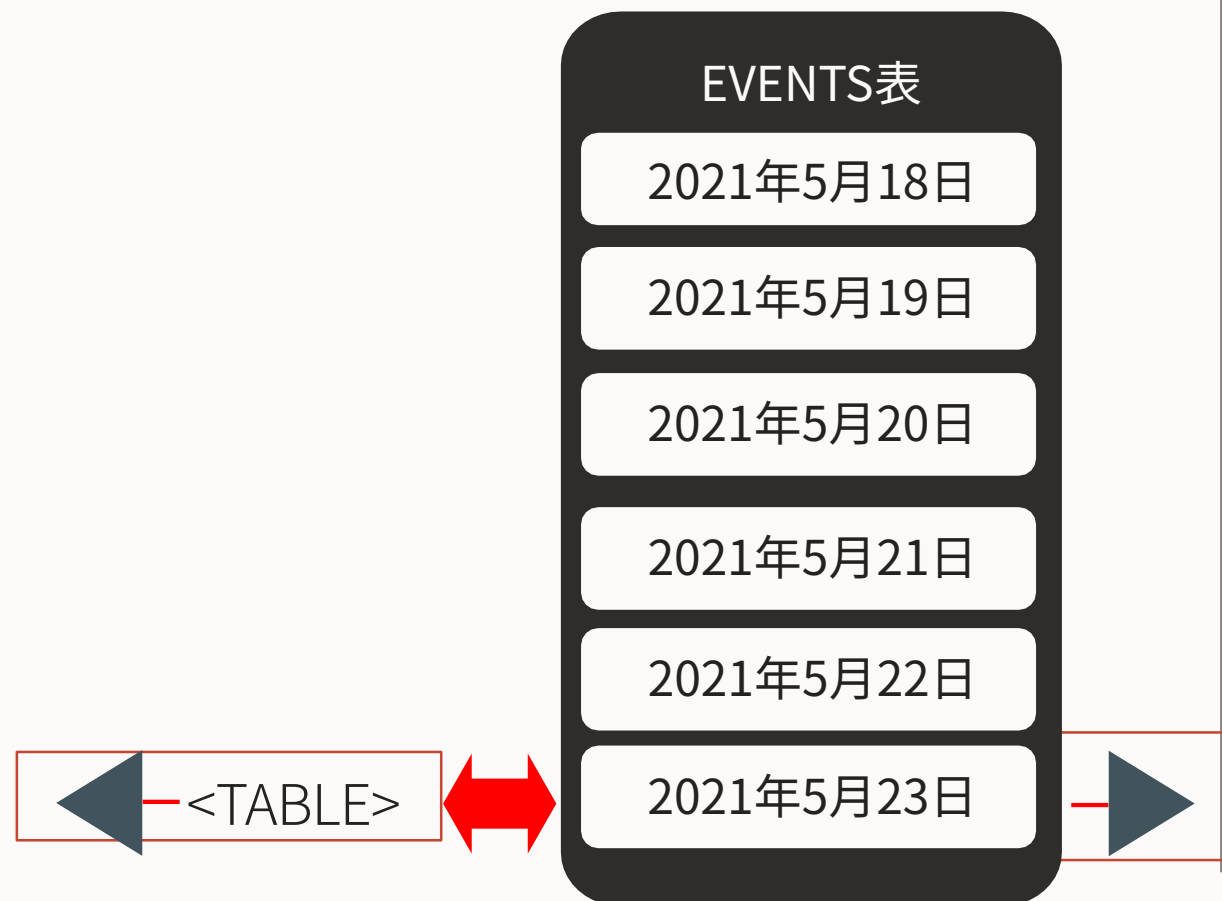
- 列の数とデータ型が同じ
  - 列名は問題ではない
- 同じ制約
- 同じ数とタイプの索引

交換表  
空または新規データ 

EVENTS表
2021年5月18日
2021年5月19日
2021年5月20日
2021年5月21日
2021年5月22日
2021年5月23日
2021年5月24日

# パーティション交換

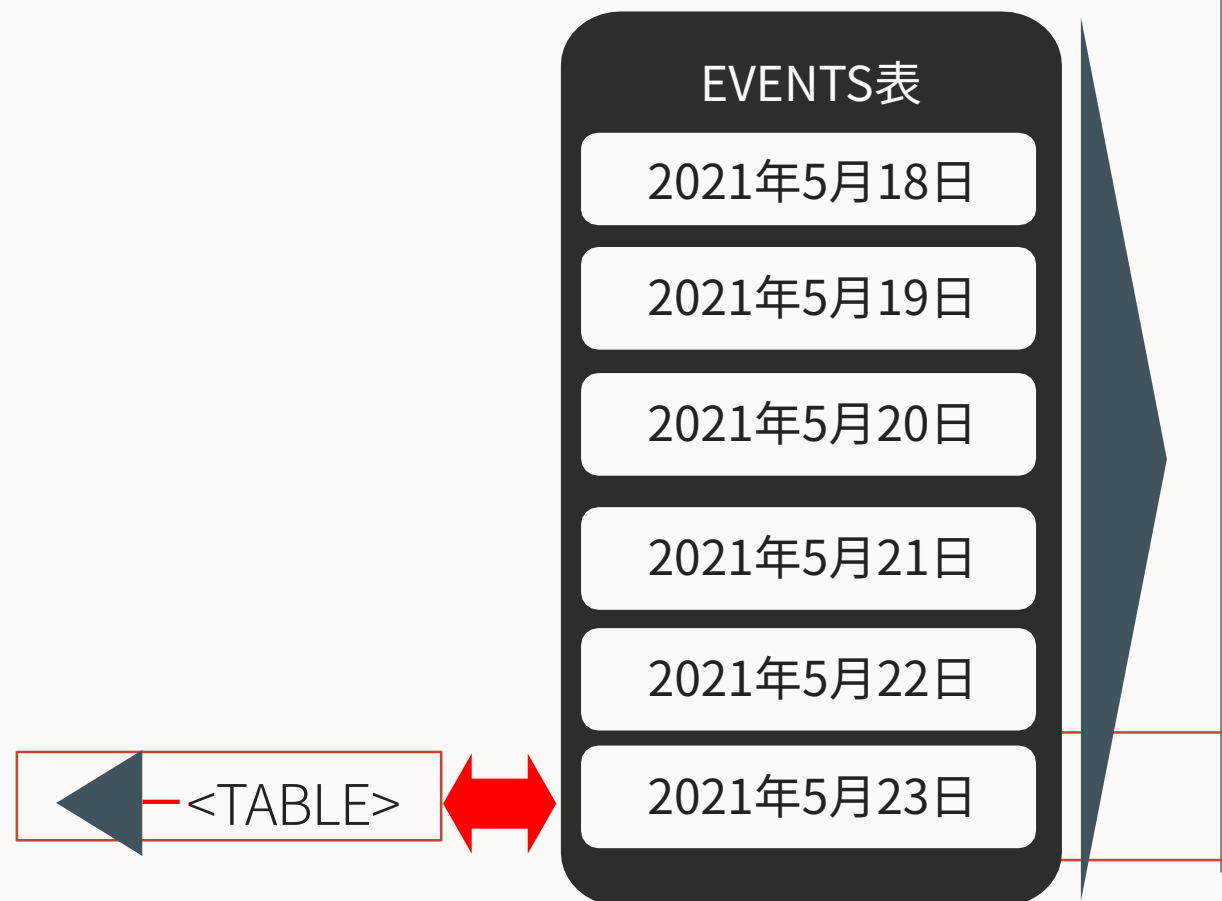
## ローカル索引



交換表の任意の索引がローカル・パーティション索引と同等

# パーティション交換

## ローカル索引



交換表の任意の索引がローカル・パーティション索引と同等

パーティション表のPK索引でグローバル索引を適用する必要がある場合はどうするか?

- 論理的に同等であるという要件を忘れないこと
- ...

# パーティション交換とPK/一意制約

ジレンマ

グローバル索引はパーティション表のためだけに存在する

- しかし、一意性を持たせるために交換表の索引が必要...

# パーティション交換とPK/一意制約

実はジレンマではない

グローバル索引はパーティション表のためだけに存在する

- しかし、一意性を持たせるために交換表の索引が必要...

一般的には正しくない

- 一意索引はenabled状態の制約の場合にのみ必要とされる
- 新規または変更されたデータは索引プローブによって適用される

# パーティション交換とPK/一意制約

実はジレンマではない

グローバル索引はパーティション表のためだけに存在する

- しかし、一意性を持たせるために交換表の索引が必要...

一般的には正しくない

- 一意索引はenabled状態の制約の場合にのみ必要とされる
- 新規または変更されたデータは索引プローブによって適用される
- disabled状態の制約によってデータ挿入を阻止する

```
SQL> alter table tt add(constraint x unique (col1) disable validate);
```

```
Table altered.
```

```
SQL> insert into tt values(1,2); insert into  
tt values(1,2);  
*
```

```
ERROR at line 1;  
ORA-25128:No insert/update/delete on table with constraint (SCOTT.X)  
disabled and validated
```



# パーティション交換とPK/一意制約 解決策

## パーティション化ターゲット表

- グローバル索引（パーティションまたは非パーティション）によって適用されるPKまたは一意制約

## 交換されるスタンドアロン表（"交換表"）

- 同等のdisabled validated状態の制約
- 制約適用のための索引なし、交換問題なし

# パーティション交換とPK/一意制約

## 簡便な例

```
SQL > CREATE TABLE tx_simple
2      (
3      TRANSACTION KEY          NUMBER,
4      INQUIRY_TIMESTAMP        TIMESTAMP(6),
5      RUN_DATE                  DATE
6      )
7      PARTITION BY RANGE (RUN_DATE)
8      (
9      PARTITION TRANSACTION_202105 VALUES LESS THAN (TO_DATE('20210601', 'yyyymmdd')),
10     PARTITION TRANSACTION_202106 VALUES LESS THAN (TO_DATE('20210701', 'yyyymmdd')),
11     PARTITION TRANSACTION_202107 VALUES LESS THAN (TO_DATE('20210801', 'yyyymmdd')),
12     PARTITION TRANSACTION_202108 VALUES LESS THAN (TO_DATE('20210901', 'yyyymmdd')),
13     PARTITION TRANSACTION_202109 VALUES LESS THAN (TO_DATE('20211001', 'yyyymmdd')),
14     PARTITION TRANSACTION_202110 VALUES LESS THAN (TO_DATE('20211101', 'yyyymmdd')),
15     PARTITION TRANSACTION_MAX VALUES LESS THAN (MAXVALUE)
16     )
17     /
```

Table created.

# パーティション交換とPK/一意制約

## 簡便な例

```
SQL > CREATE TABLE tx_simple
 2  (
 3  TRANSACTION KEY          NUMBER,
 4  INQUIRY_TIMESTAMP        TIMESTAMP(6),
 5  RUN_DATE                  DATE
 6  )
 7  PARTITION BY RANGE (RUN_DATE)
 8  (
 9  PARTITION TRANSACTION_202105 VALUES LESS THAN (TO_DATE('20210601', 'yyyymmdd')),
10  PARTITION TRANSACTION_202106 VALUES LESS THAN (TO_DATE('20210701', 'yyyymmdd')),
11  PARTITION TRANSACTION_202107 VALUES LESS THAN (TO_DATE('20210801', 'yyyymmdd')),
12  PARTITION TRANSACTION_202108 VALUES LESS THAN (TO_DATE('20210901', 'yyyymmdd')),
13  PARTITION TRANSACTION_202109 VALUES LESS THAN (TO_DATE('20211001', 'yyyymmdd')),
14  PARTITION TRANSACTION_202110 VALUES LESS THAN (TO_DATE('20211101', 'yyyymmdd')),
15  PARTITION TRANSACTION_MAX VALUES LESS THAN (MAXVALUE)
16  )
17  /
```

Table created.

```
SQL > INSERT into tx_simple (
 2  select object_id, LAST_DDL_TIME,
 3  add months(TO_DATE('20210501', 'yyyymmdd'), mod(OBJECT_ID,
4  12))
 4  from DBA_OBJECTS
 5  where object_id is not null)
 6  /
```

73657 rows created.

# パーティション交換とPK/一意制約

## 簡便な例

```
SQL > CREATE TABLE tx_simple
 2  (
 3      TRANSACTION_KEY          NUMBER,
 4      INQUIRY_TIMESTAMP        TIMESTAMP(6),
 5      RUN_DATE                 DATE
 6  )
 7  PARTITION BY RANGE (RUN_DATE)
 8  (
 9      PARTITION TRANSACTION_202105 VALUES LESS THAN (TO_DATE('20210601', 'yyyymmdd')),
10      PARTITION TRANSACTION_202106 VALUES LESS THAN (TO_DATE('20210701', 'yyyymmdd')),
11      PARTITION TRANSACTION_202107 VALUES LESS THAN (TO_DATE('20210801', 'yyyymmdd')),
12      PARTITION TRANSACTION_202108 VALUES LESS THAN (TO_DATE('20210901', 'yyyymmdd')),
13      PARTITION TRANSACTION_202109 VALUES LESS THAN (TO_DATE('20211001', 'yyyymmdd')),
14      PARTITION TRANSACTION_202110 VALUES LESS THAN (TO_DATE('20211101', 'yyyymmdd')),
15      PARTITION TRANSACTION_MAX VALUES LESS THAN (MAXVALUE)
16  )
```

```
SQL > INSERT into tx_simple (
 2      select object_id, LAST_DDL_TIME,
 3      add_months(TO_DATE('20210501', 'yyyymmdd'), mod(OBJECT_ID,
12))
```

```
SQL > CREATE UNIQUE INDEX tx_simple_PK ON tx_simple (TRANSACTION_KEY) nologging
 2      GLOBAL PARTITION BY RANGE (TRANSACTION_KEY) (
 3      PARTITION P_Max VALUES LESS THAN (MAXVALUE)
 4      )
 5  /
```

Index created.

```
SQL > ALTER TABLE tx_simple ADD ( CONSTRAINT tx_simple_PK PRIMARY KEY (TRANSACTION_KEY)
 2      USING INDEX nologging);
```

Table altered.

# パーティション交換とPK/一意制約

## 簡便な例（続き）

```
SQL > create table DAILY_ETL_table
2     as
3     select * from tx_simple partition (TRANSACTION_202107);
```

Table created.

```
SQL > alter table daily_etl_table add ( constraint pk_etl primary key (transaction_key) disable validate);
```

Table altered.

```
SQL > alter table tx_simple
2     exchange partition TRANSACTION_202107
3     with table daily_ETL_table
4     including indexes
5     --excluding indexes
6     WITHOUT VALIDATION
7     UPDATE GLOBAL INDEXES
8     /
```

Table altered.

# 属性クラスタリングと ゾーン・マップ

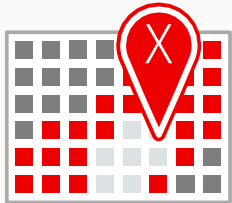
Oracle 12c Release 1 (12.1.0.2) で導入  
Oracle ExadataおよびOracle Cloudのみ

# 属性クラスタリングによるゾーン・マップ



## 属性クラスタリング

列の値が一緒にディスクに格納されるようにデータを配列



## ゾーン・マップ

ゾーン単位で指定された列の最小/最大値を格納

問合せ実行時に不必要なデータをフィルタするために使用

## 組み合わせることの利点

問合せのパフォーマンスと同時実行性の向上

- 物理データ・アクセスの削減
- 選択性の高い操作のIO処理を大幅に削減

領域の使用効率を最適化

- 索引の必要性を軽減
- データ・クラスタリングによる圧縮比の改善

完全なアプリケーションの透過性

- すべてのアプリケーションにとってメリットがある

# 属性クラスタリング

## 概要

選択した列の値："属性"に基づいて、  
緊密に近接するようにデータを順序付けする

属性は、単一の表や複数の表から取り込むことが  
可能

- 例：ファクト表やディメンション表

## 利点

ゾーン・マップで使用する場合にIOを著しくプルー  
ニング

索引レンジ・スキャンでの表参照でブロックIOを削減

ソートと集計の問合せで事前順序付けデータによる  
メリットを活用できる

圧縮比を改善できる

- 順序付けされたデータは、順序付けされない  
データよりも圧縮率が高くなりやすい



# ゾーン・マップのための属性クラスタリング 順序付けされた行

```
ALTER TABLE EVENTS  
ADD CLUSTERING BY  
LINER ORDER (category);  
  
ALTER TABLE EVENTS  
MOVE;
```

Category	Country
BOYS	AR
BOYS	JP
BOYS	SA
BOYS	US
GIRLS	AR
GIRLS	JP
GIRLS	SA
GIRLS	US
MEN	AR
MEN	JP
MEN	SA
MEN	US
WOMEN	AR
WOMEN	JP
WOMEN	SA
WOMEN	US

カテゴリ値BOYS、GIRLS、MENが入力されている順序付けされた行。

ゾーン・マップは、特定の列値レンジが入力された行またはゾーンのカタログ領域を示す。

- デフォルトでは、各ゾーンは最大で1024ブロック。

たとえば、カテゴリ"GIRLS"を検索する場合は、このゾーンをスキャンするだけでよい。他のすべてのゾーンはスキップできる。

# 属性クラスタリング

## 基本

---

### 2種類の属性クラスタリング

- LINEAR ORDER BY
  - 旧式の順序付け
- INTERLEAVED ORDER BY
  - 複数ディメンションの順序付け

### 単一表での簡単な属性クラスタリング

### 結合属性クラスタリング

- 複数の表の結合によって派生する属性のクラスタ
  - 最大で4つの表
  - 非重複結合（結合表でのPKまたはUKが必要）

# 属性クラスタリング 例

LINEAR ORDER (category, country)

Category	Country
BOYS	AR
BOYS	JP
BOYS	SA
BOYS	US
GIRLS	AR
GIRLS	JP
GIRLS	SA
GIRLS	US
MEN	AR
MEN	JP
MEN	SA
MEN	US
WOMEN	AR
WOMEN	JP
WOMEN	SA
WOMEN	US

対

INTERLEAVED ORDER (category, country)

	Country			
	10 AR WOMEN	11 JP WOMEN	14 SA WOMEN	15 US WOMEN
8	AR MEN	9 JP MEN	12 SA MEN	13 US MEN
2	AR GIRLS	3 JP GIRLS	6 SA GIRLS	7 US GIRLS
0	AR BOYS	1 JP BOYS	4 SA BOYS	5 US BOYS

# 属性クラスタリング

## 基本

---

表レベルで指定されるクラスタリング・ディレクティブ

- ALTER TABLE ...ADD CLUSTERING ...

ディレクティブは新規データとデータ移動に適用される

ダイレクト・パス操作

- INSERT APPEND、MOVE、SPLIT、MERGE
- 従来のDMLには適用されない

オンデマンドで有効化および無効化が可能

- ヒントまたは特定の構文

# ゾーン・マップ

## 概念と基本

---

指定された列の最小値と最大値を格納する

- 情報はゾーン単位で格納される
- マルチディメンションのパーティション・プルーニングのためのパーティション表の [サブ]パーティション・レベルのロールアップ情報

疎索引構造に類似

- 索引よりはるかにコンパクト
- ゾーン・マップで不要なものをフィルタして除外し、索引で必要なものを検索する

完全なアプリケーションの透過性による著しいパフォーマンス上の利点

- 表自体で、または結合ゾーン・マップ ("階層ゾーン・マップ"とも呼ばれる) を使用する結合表であっても、条件付きの表スキャンでIOを削減

順序付けデータの場合に最大の利点が得られる

- 属性クラスタリングまたは自然に順序付けされたデータと組み合わせて使用される

# ゾーン・マップ

## 基本

---

表に合わせて構築された独立アクセス構造

- ある種のマテリアライズド・ビューを使用して実装
- パーティション表と非パーティション表の場合

表あたり1つのゾーン・マップ

- パーティション表のゾーン・マップには、[サブ]パーティション単位の集計エントリが取り込まれている

透過的に使用される

- 問合せの変更またはヒントは不要

暗黙的または明示的な作成と列選択

- 属性クラスタリングの場合：CREATE TABLE ... CLUSTERING
- CREATE MATERIALIZED ZONEMAP ... AS SELECT ...

# ゾーン・マップでの属性クラスタリング

## CLUSTERING BY LINEAR ORDER (category, country)

ゾーン・マップの利点は順序付けデータの場合に最大化

- 順序付け列で条件が指定されている場合にのみプルーニングする
- 順序付け列をスキップする場合はプルーニングなし

Category	Country
BOYS	AR
BOYS	JP
BOYS	SA
BOYS	US
GIRLS	AR
GIRLS	JP
GIRLS	SA
GIRLS	US
MEN	AR
MEN	JP
MEN	SA
MEN	US
WOMEN	AR
WOMEN	JP
WOMEN	SA
WOMEN	US

## プルーニングのコマンド：

```
SELECT ..  
FROM table  
WHERE category =  
  'BOYS';
```

```
SELECT ..  
FROM table WHERE  
category =  
  'BOYS';  
AND country = 'US';
```

# ゾーン・マップでの属性クラスタリング

## CLUSTERING BY INTERLEAVED ORDER (category, country)

ゾーン・マップの利点は順序付けデータの場合に最大化

- すべての順序付け列が対象だとプルーニングの効率が低下する
- 後続の順序付け列でのプルーニング

		Country						
Category	10	AR WOMEN	11	JP WOMEN	14	SA WOMEN	15	US WOMEN
	8	AR MEN	9	JP MEN	12	SA MEN	13	US MEN
	2	AR GIRLS	3	JP GIRLS	6	SA GIRLS	7	US GIRLS
	0	AR BOYS	1	JP BOYS	4	SA BOYS	5	US BOYS

## プルーニングのコマンド：

```
SELECT ..  
FROM table WHERE category =  
'BOYS';
```

```
SELECT ..  
FROM table  
AND country = 'US';
```

```
SELECT ..  
FROM table WHERE category =  
'BOYS'  
AND country = 'US';
```



# ゾーン・マップ

## 失効

DML操作とパーティション操作により、ゾーン・マップが完全または部分的に失効する可能性がある

- ダイレクト・パス挿入によってゾーン・マップが失効することはない

## 単一表の'ローカル'ゾーン・マップ

- 更新および挿入により、影響を受けたゾーンが失効（および任意の集計されたパーティション・エントリ）としてマークされる
- 削除するゾーン・マップには影響しない

## 結合ゾーン・マップ

- ファクト表でのDMLは、単一表のゾーン・マップと同等の動作
- デイメンション表でのDMLは、依存ゾーン・マップを完全に失効させる

# ゾーン・マップ

## リフレッシュ

DMLによる必要に応じて、増分リフレッシュと完全リフレッシュ

- ゾーン・マップ・リフレッシュにはマテリアライズド・ビューのログが必須
  - MVをリフレッシュするため、失効ゾーンのみがスキャンされる
- 結合ゾーン・マップの場合
  - ファクト表でのDML：増分リフレッシュ
  - ディメンション表でのDML：完全リフレッシュ

ゾーン・マップ・メンテナンスで使用するコマンド

- DBMS\_MVIEW.REFRESH()
- ALTER MATERIALIZED ZONEMAP <xx> REBUILD;

## 例 - デイメンション階層

ORDERS

id	product_id	location_id	amount
1	3	23	2.00
2	88	55	43.75
3	31	99	33.55
4	33	62	23.12
5	21	11	38.00
6	33	21	5.00
7	44	71	10.99

注：ゾーンには通常、ここで示すよりもはるかに多くの行があります。これは、説明目的のほんの一部に過ぎません。

LOCATIONS

location_id	State	county
23	California	Inyo
102	New Mexico	Union
55	California	Kern
1	Ohio	Lake
62	California	Kings

```
CREATE TABLE orders (...)  
CLUSTERING orders  
JOIN locations ON (orders.location_id = locations.location_id)  
BY INTERLEAVED ORDER (locations.state, locations.county)  
WITH MATERIALIZED ZONEMAP ...
```

## 例 - デイメンション階層

ORDERS

id	product_id	location_id	amount
1	3	23	2.00
2	88	55	43.75
3	31	99	33.55
4	33	62	23.12
5	21	11	38.00
6	33	21	5.00
7	44	71	10.99

Scan  
Zone



LOCATIONS

location_id	State	county
23	California	Inyo
102	New Mexico	Union
55	California	Kern
1	Ohio	Lake
62	California	Kings

注：ゾーンには通常、ここで示すよりもはるかに多くの行があります。これは、説明目的のほんの一部に過ぎません。

```
SELECT SUM(amount)
FROM orders
JOIN locations ON (orders.location.id = locations.location.id)
WHERE state = 'California';
```

## 例 - ディメンション階層

ORDERS

id	product_id	location_id	amount
1	3	23	2.00
2	88	55	43.75
3	31	99	33.55
4	33	62	23.12
5	21	11	38.00
6	33	21	5.00
7	44	71	10.99

Scan  
Zone



LOCATIONS

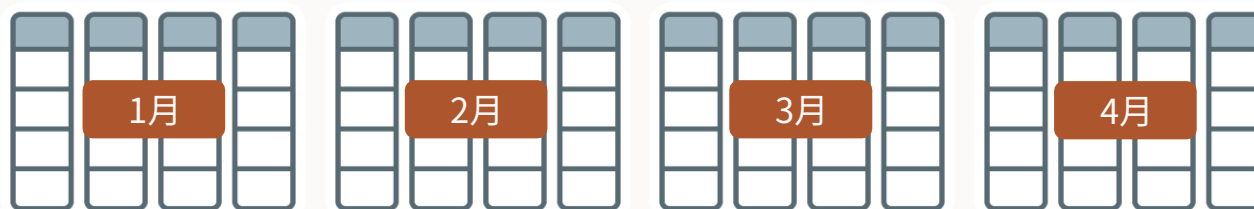
location_id	State	county
23	California	Inyo
102	New Mexico	Union
55	California	Kern
1	Ohio	Lake
62	California	Kings

注：ゾーンには通常、ここで示すよりもはるかに多くの行があります。これは、説明目的のほんの一部に過ぎません。

```
SELECT SUM(amount)
FROM orders
JOIN locations ON (orders.location.id = locations.location.id)
WHERE state = 'California'
AND county = 'Kern';
```

# ゾーン・マップとパーティション化

パーティション・キー：  
ORDER\_DATE



ゾーン・マップ列  
SHIP\_DATE  
パーティション・キー  
ORDER\_DATEと相関

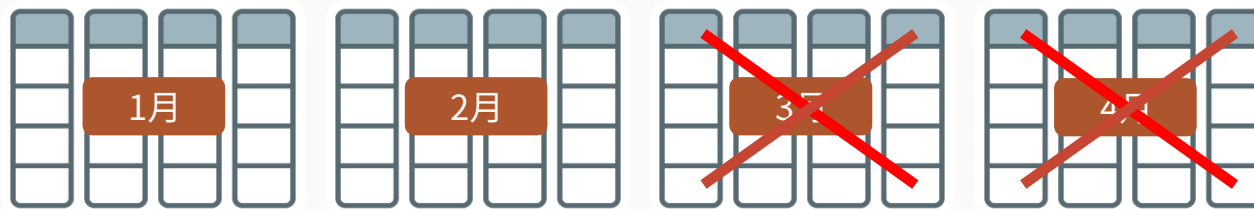
ゾーン・マップ：  
SHIP\_DATE



ゾーン・マップでは、パーティション（またはサブパーティション）キーに含まれていない列のパーティションをプルーニングできる

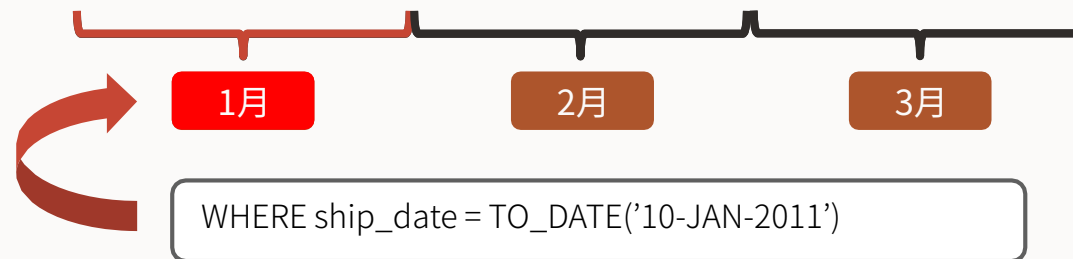
# ゾーン・マップとパーティション化

パーティション・キー：  
ORDER\_DATE



3月と4月のパーティションがプルーニングされる

ゾーン・マップ：  
SHIP\_DATE



```
WHERE ship_date = TO_DATE('10-JAN-2011')
```

ゾーン・マップでは、パーティション（またはサブパーティション）キーに含まれていない列のパーティションをプルーニングできる

## ゾーン・マップとストレージ索引

属性クラスタリングとゾーン・マップは、Exadataのストレージ索引と透過的に連携動作する

- Exadataのストレージ索引の利点は、引き続きフルに活用される

加えて、ゾーン・マップにより（属性クラスタリングと併用するときに）以下の機能が提供される

- 著しいIO最適化を追加で可能にする
  - 特に大規模表において索引の代替手段を提供する
  - 結合とファクト-ディメンション問合せ、ディメンション階層検索を含む
  - スター・スキーマとスノーフレイク・スキーマで特に有用
- パーティションとサブパーティションの全体をプルーニングできる
- ダイレクト・パスと従来式パスの両方の読取りで効果的
- 結合と索引レンジ・スキャンの最適化を含む
- 物理データベース設計の部分：DBAが明示的に作成および制御する



当社のミッションは、人々が新たな方法  
でデータを参照し、インサイトを発見し、  
無限の可能性を解き放つことができるよ  
う支援することです。

