ORACLE®

Oracle Text 詳細解説

日本オラクル株式会社
Oracle Direct

Agenda

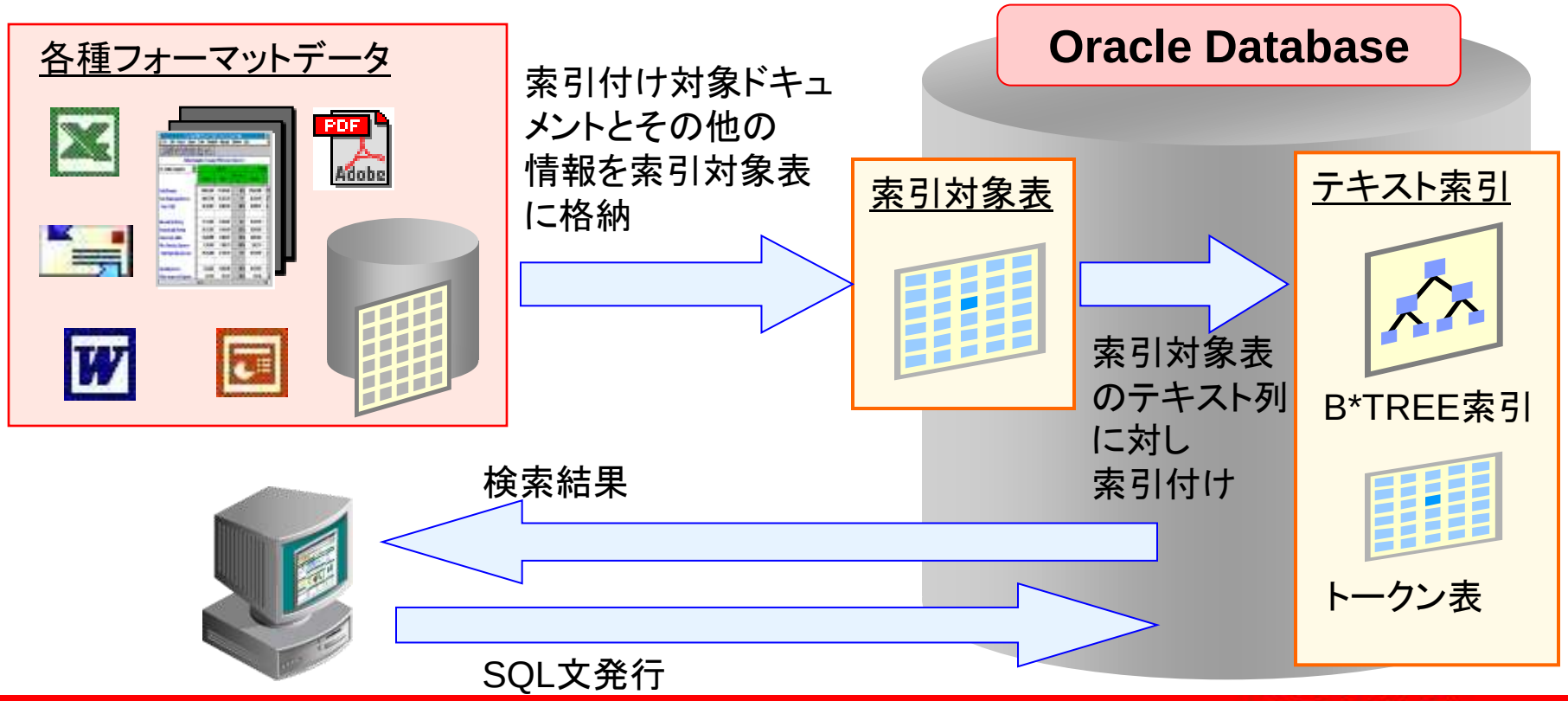
概要

Oracle Text とは？

- Oracleカーネルに組み込まれた、全文検索およびドキュメント分類のためのエンジン
- Oracle8i Database Release 8.1.6 以降で利用可能
  ※ Oracle8i 時点の製品名は「interMedia Text」。「Oracle Text」の呼称は Oracle9i Database Release 1（9.0.1）以降で使われている。interMedia Text と Oracle Text は、製品名が異なるだけで、その内容および利用方法は同一である。
- Oracle Database の全てのエディションで利用可能
  - Standard Edition One、Standard Edition、Enterprise Editionで利用可能
  - Oracle Text を利用するために追加のオプション・ライセンスは必要なし

Oracle Text 全体図

- Oracle Text はOracle Databaseカーネルに組み込まれており、データの一元管理が可能



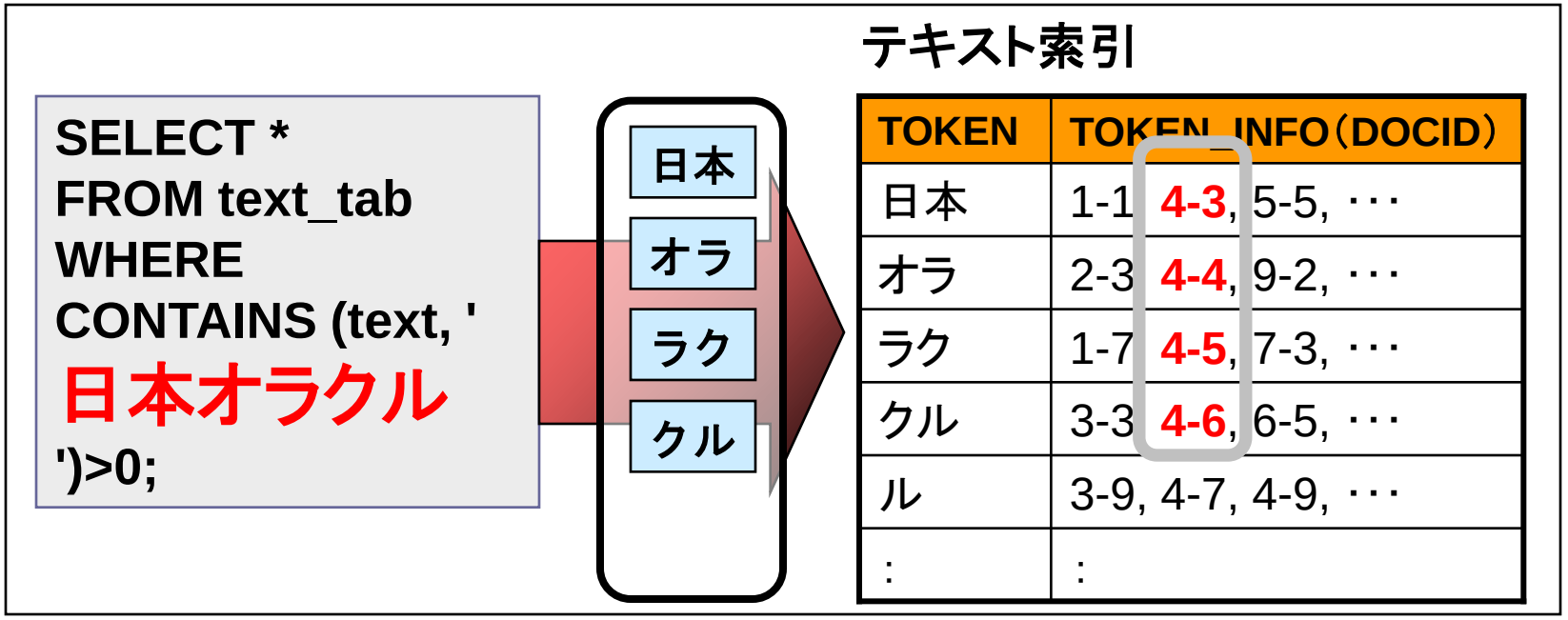
Oracle Text が提供する索引

- CONTEXT索引 ... 全文検索、あるいは全文検索＋定型検索(※1)を高速にするための索引。XML検索にも対応
  ※1　DB 11.1 以降では、コンポジット・ドメイン索引を利用することで、全文検索と定型検索の組み合わせを高速に処理可能。
- CTXCAT索引 ... 全文検索＋定型検索を高速にするための索引
- CTXRULE索引 ... ドキュメント分類を高速にするための索引

➔ この資料では、**CONTEXT索引**ついて解説する

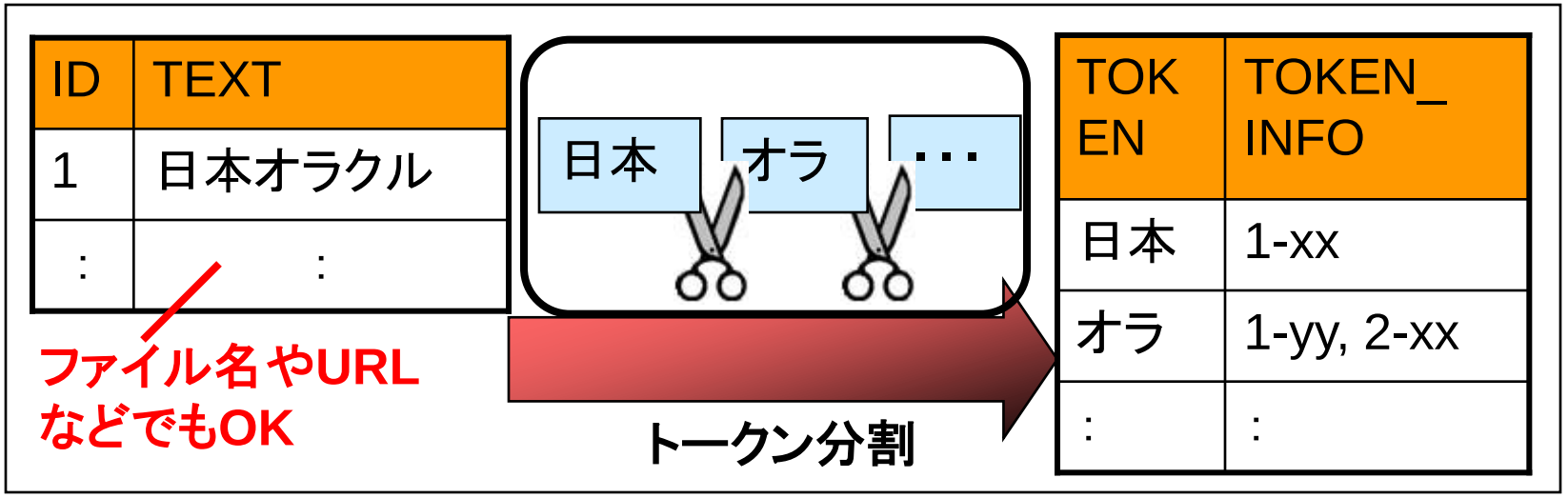
なぜOracle Textはそんなに速く結果を返すのか

- 全文検索専用のCONTEXT索引（以下、テキスト索引）を作成することで、検索対象ドキュメントをひとつひとつ検索するよりも高速に検索が行える



索引作成のメカニズム

- 文字列を「トークン」という細かい単位に区切り、出現位置情報を付けて「テキスト索引」に格納する
- WordやPDFをはじめ200種類以上のファイル形式、HTMLページなどにも索引付けできる



索引作成のメカニズム

①データストア
データの格納場所を指定

②フィルタ
プレーン・テキストを抽出

③セクショナ
タグを分離しセクション情報を理解

④レクサー
トークンへの分割

<H1>

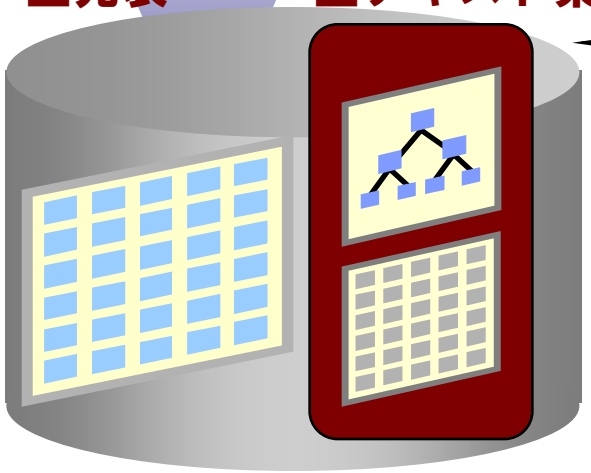
日本オラクル

日本 オラ

ラク クル ル

■元表

■テキスト索引



⑦ストレージ
テキスト索引の格納先を指定。

⑥ワードリスト
ファジー検索を行う際などに設定する。

⑤ストップリスト
a, theなど頻出するトークンを索引付けから除外する。

検索のメカニズム

1. 索引付け時と同様のアルゴリズムで、**検索文字列をトークン分割**する
2. テキスト索引を問い合わせ、**分割した各トークンの出現位置情報を取得**する
3. 取得した出現位置情報から、**「トークン分割順に連続する部分」**を割り出す

```
SELECT *
FROM text_tab
WHERE
CONTAINS (text, '
日本オラクル
')>0;
```

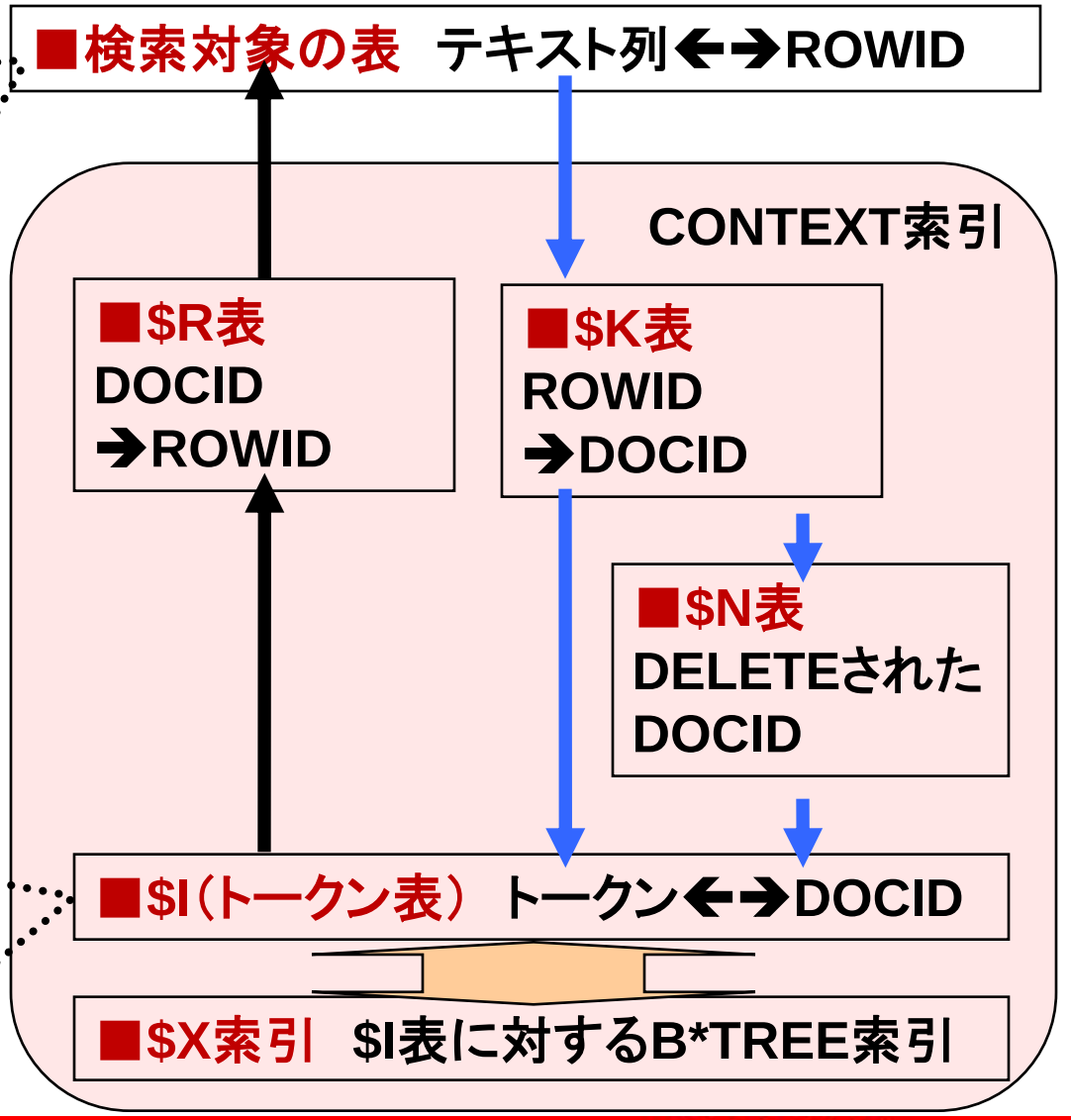
TOKEN	TOKEN_INFO（DOCID）
日本	1-1, **4-3**, 5-5, ・・・
オラ	2-3, **4-4**, 9-2, ・・・
ラク	1-7, **4-5**, 7-3, ・・・
クル	3-3, **4-6**, 6-5, ・・・
ル	3-9, **4-7**, 4-9, ・・・
:	:

CONTEXT索引の構成要素

ID	TEXT
1	ORACLE
2	TEXT
3	ORACLE

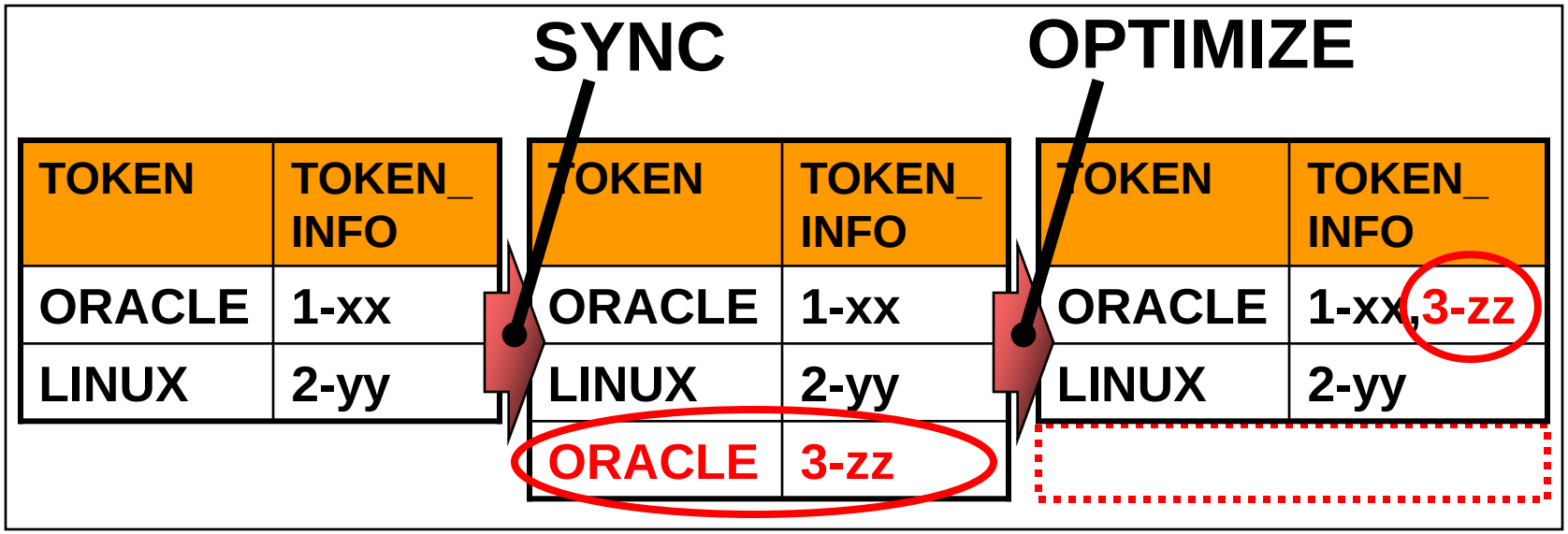
※ CONTEXT索引の実体は、右図のように4つの表と1つのB*TREE索引から構成される。元表の各行には、Oracle Textによって自動的にDOCIDが割り振られ、$I表にはDOCIDに基づくトークン情報が格納される。DOCIDとROWIDの対応付けは、$R表および$K表によって行われる。図の矢印は、検索時の処理の流れを示している。

トークン	出現位置
ORACLE	1-1 3-1
TEXT	2-1



索引メンテナンスのメカニズム

- 「SYNC」（同期化）を行うと、新たにINSERTされた行のトークンが索引の末尾に追加される
- 「OPTIMIZE」（最適化）を行うと、複数の同一トークン・エントリがマージされる



検索（1）～ CONTAINS関数

※ この章では、Oracle Textで利用可能な検索演算子について解説する

CONTAINS関数

- 全文検索で利用するSQL関数(CONTEXT索引)
- 返り値は、0以上100以下の整数
- 返り値が1以上のとき、全文検索の条件に一致することを示す
- このため、検索条件を、
「CONTAINS (<列名>, '<検索条件>') > 0」
のように指定する
- 使用例:

```
SELECT id FROM testtab WHERE CONTAINS (text, 'オラクル') > 0;
```

→ これで、text列に「オラクル」という文字列を含む全ての行のidが得られる

CONTAINS問合せ演算子（抜粋）

論理演算子　AND　OR　ACCUM
NOT　MINUS　MNOT

ワイルドカード　%　_

スコア関連　*　>

近傍　NEAR

等価　EQUIV

セクション
WITHIN
INPATH　HASPATH
MDATA　SDATA
NDATA

シソーラス
SYN　BT　NT

エスケープ文字
¥　{}

AND(&)

- 構文
  - ***term1 & term2*** ←「&」の前後に半角スペースはあってもなくてもいい
  - ***term1*** and ***term2*** ←「AND」、「And」、「aNd」等いずれも可

→ *term1* と*term2* の両方が含まれているドキュメントが返る

- スコア
  - 指定されたキーワードのうち、最小のスコアが返る
  - 例えば、「blue & black & red」という検索条件で、各キーワードのスコアが 10、20、30 であると仮定すると、全体のスコアは 10 になる

```
SELECT id FROM testtab
  WHERE CONTAINS (text, 'オラクル and 日本') > 0;
```

OR(|)

- 構文
  - ***term1*** | ***term2*** ←「|」の前後に半角スペースはあってもなくてもいい
  - ***term1*** or ***term2*** ←「OR」、「Or」、「oR」等のいずれも可

→ *term1* と*term2* のいずれかが含まれているドキュメントが返る

- スコア
  - 指定されたキーワードのうち、最大のスコアが返る
  - 例えば、「cats | dogs」という検索条件で、各キーワードのスコアが 30、40 であると仮定すると、全体のスコアは 40 になる

```
SELECT id FROM testtab
  WHERE CONTAINS (text, 'オラクル or 日本') > 0;
```

ACCUMulate(,)

※ accumulate＝累積する

- 構文
  - ***term1, term2*** ←「, 」の前後に半角スペースはあってもなくてもいい
  - ***term1*** **accum** ***term2*** ←「ACCUM」、「Accum」等のいずれも可

→ *term1* と*term2* のいずれかが含まれているドキュメントが返る

- スコア
  - 指定されたキーワードの全てが含まれているものに高いスコアが割り当てられる
  - スコア以外の部分は、OR演算子と同じ動作

```
SELECT id FROM testtab
  WHERE CONTAINS (text, 'オラクル accum 日本') > 0;
```

NOT(~)

- 構文
  - ***term1~term2*** ←「~」の前後に半角スペースはあってもなくてもいい
  - ***term1*** not ***term2*** ←「NOT」、「Not」、「nOt」等のいずれも可

→ *term1*が含まれるが、*term2*が含まれていないドキュメントが返る

- スコア
  - NOT演算子は、他の論理演算子が作成したスコアには影響を与えない

```
SELECT id FROM testtab
  WHERE CONTAINS (text, 'オラクル ~日本') > 0;
```

MINUS

- 構文
  - ***term1-term2*** ←「-」の前後に半角スペースはあってもなくてもいい
  - ***term1*** minus ***term2*** ←「MINUS」、「Minus」、「miNus」等のいずれも可

→ *term1*が含まれているドキュメントが返る。ただし、*term1*のスコアから*term2*のスコアを引いてスコアを計算し、正数のスコアを持つドキュメントのみが返る

- スコア
  - *term1*のスコアから*term2*のスコアを引いてスコアを計算する

```
SELECT id FROM testtab
  WHERE CONTAINS (text, 'オラクル -日本') > 0;
```

MNOT
（Oracle Database 11g Release 2（11.2）以降）

※ MNOT＝Mild Not

- 構文（使用例）
  - ***term1*** mnot ***term1 term2***

  → *term1*が含まれ、かつフレーズ *term1 term2* が含まれないドキュメントが返る（例えば「mexico MNOT new mexico」のように使う）
  - ***term1*** mnot ***term2***

  → *term1*が含まれるドキュメントが返る（単に「*term1*」と指定した場合と同じ動作）
- スコア
  - MNOTの左側に書かれた条件に一致し、かつ右側に書かれた条件に一致しない部分のみがスコアの加算対象となる
  - 例えば「*term1 term1 term2*」という文字列を含む文書を「*term1* mnot *term1 term2*」で検索した場合、この文書はヒットするが、スコアは最初の*term1*に対してのみ計算される

ワイルドカード(「%」、「_」)

- 2種類のワイルドカード

ワイルドカード文字	説明
%	任意のn文字との一致
_	任意の1文字との一致

- 英数字検索では、ワイルドカード文字を利用する
  - たとえば、文字列「Oracle」を含むドキュメントは、「WHERE CONTAINS (列名, '%ac%') > 0」という条件でヒットする
- 日本語検索では、ワイルドカード文字を付けてはいけない
  - たとえば、文字列「オラクル」を含むドキュメントは、「WHERE CONTAINS (列名, '%オラクル%') > 0」のように、ワイルドカード文字を付けて検索してもヒットしない。「WHERE CONTAINS (列名, 'オラクル') > 0」のように、ワイルドカード文字を付けない状態で検索をして初めて正常にヒットする

ワイルドカード使用例

- 右側切捨て問合せ
  - 右側切捨てでは、検索文字列の右側にワイルド・カードが置かれる
  - 例えば、パターンscalで始まるすべての語句を検索するには、「scal%」のように指定する
- 左側切捨て問合せ
  - 左側切捨てでは、検索文字列の左側にワイルド・カードが置かれる
  - 例えば、king、wing、singなどの語句を検索するには、「_ing」のように問合せを記述する
  - ingで終わるすべての語句を検索するには、「%ing」のように問合せを記述する
- 左右切捨て問合せ
  - 左側切捨て検索および右側切捨て検索を組み合せて、左右切捨て検索を実行することが可能
  - 例えば、サブストリング「benz」を含む全ての語句を検索するには、「%benz%」のように問合せを記述する

ワイルドカード問合せのパフォーマンス改善

- ワイルドカード問合せが左側切捨ておよび左右切捨ての場合・・・
  - サブストリング索引（p.166）を作成すると、問合せパフォーマンスが向上する
  - サブストリング索引によって、%ed、_ing、%benz%などの、あらゆる種類の左側切捨ておよび左右切捨てのワイルドカード検索の問合せパフォーマンスが向上する
- ワイルド・カード問合せが右側切捨ての場合・・・
  - プリフィックス索引（p.169）を作成すると、問合せパフォーマンスが向上する
  - プリフィックス索引によって、to%などのワイルドカード検索の問合せパフォーマンスが向上する
    【注】プリフィックス索引がなくても、右側切捨てワイルドカード問合せは、通常はある程度高速

ワイルドカード使用時の注意点

- エスケープされないワイルド・カード文字が含まれる、1つの問合せ内のすべてのワードのワイルド・カード拡張の合計数は、**BASIC_WORDLIST属性のWILDCARD_MAXTERMS**（p.172～176）で指定された拡張の最大数を超えないようにする

近傍検索（NEAR）

- 2つ以上の検索キーワードの出現位置の近さを基準にしたスコアが戻される
- ドキュメント内で互いに近接したキーワードには高いスコアが、互いに離れたキーワードには低いスコアが戻される

近傍検索(NEAR)

- 構文

NEAR((*word 1, word 2, . . . , word n*) [, max_span [, order]])

  - *word 1*～*word n*
    問合せ語句をカンマ区切りで指定
  - max_span
    上で指定した語句が何トークン以内に出現するかを、100以下の整数で指定(デフォルトは100)
  - order
    上で指定した語句が、指定した順序どおりで出現しているものを探すにはTRUE、そうでない場合はFALSEを指定
    (デフォルトはFALSE)

近傍検索(NEAR)使用例

- dogとcatが6ワード以内にある文書を検索するには・・・

```
SELECT id FROM testtab
  WHERE CONTAINS (text, 'NEAR((dog, cat), 6)') > 0;
```

- 3つの検索キーワード Monday、Tuesday、Wednesday が、20ワード以内に、この順で出現している文書を検索するには・・・

```
SELECT id FROM testtab
  WHERE CONTAINS (text,
'NEAR((Monday, Tuesday, Wednesday), 20, TRUE)') > 0;
```

近傍検索(NEAR)使用例

- 他の演算子との組合せ例

```
SELECT id FROM testtab
  WHERE CONTAINS (text,
  'NEAR((lion, tiger), 10) AND cheetah') > 0;
```

等価検索（EQUIV）

- 検索時に条件を満たすワードの置換を指定する
- 構文
  - *term1=term2*
  - *term1* equiv *term2*

→ *term1* と *term2* が互いに置換語として指定される

- たとえば、

```
German shepherds=alsatians are big dogs
```

という検索条件は、「alsatians are big dogs」を含む文書と「German shepherds are big dogs」を含む文書の両方にヒットする

重み付け(*)
WEIGHT

- 指定した係数をスコアに掛ける
- たとえば、問合せ「cat, dog*2」は、
  「catのスコア」と「dogのスコアの2倍」をベースに、
  全体のスコアが計算される
- スコアの最大値は100
- 構文
  - *term*n*　　※ *n* は、0.1～10の数値

→ *term*のスコアに *n* を掛けてスコアが計算される

- Tips:スコアに10より大きい数字を掛けるには？

(*term1*10)*10 or *term2*

→ これで、*term1* を含む文書のスコアは必ず100になる

重み付け(*)使用例
WEIGHT

- 前提
  - スポーツ記事の収集で、ブラジルのサッカーについての記事に関心があると仮定
  - 「soccer or Brazil」で通常の問合せをすると、USサッカーに関する多くの記事が高いランクで戻ってしまう
- このとき、ブラジルのサッカーについての記事のランクを上げるには・・・？

→ 例えば、次の問合せを実行する

```
soccer or Brazil*3
```

→ このときのスコアの変化(次ページ)

重み付け(*)使用例
WEIGHT

- 次の表は、重み付け演算子によって、サッカーの情報が含まれている3つのドキュメントA、B、Cのランクが変更されることを示している

スコア・サンプル

	soccer	Brazil	soccer or Brazil	soccer or Brazil*3
A	20	10	20	30
B	10	30	30	90
C	50	20	50	60

しきい値（>）
THRESHOLD

- しきい値の数値を下回るスコアのドキュメントを、検索結果から排除する
- 問合せ語句レベル、式レベルのいずれでも使用可能
- 構文
  - 問合せ語句レベル
    term > *n*

    → 式内で、スコアが *n* より大きい問合せ語句が含まれているドキュメントが返される
  - 式レベル
    expression > *n*

    → 結果セットで、スコアがしきい値 *n* を超えるドキュメントが返される

しきい値（>）使用例
THRESHOLD

- 問合せ語句レベル
  - lionのスコアが30よりも大きく、さらにtigerが含まれているドキュメントを検索するには・・・？

```
SELECT id FROM testtab
  WHERE CONTAINS (text, '(lion > 30) and tiger') > 0;
```

- 式レベル
  - relational databasesが含まれているドキュメントを検索し、スコアが75よりも大きいドキュメントのみを検索するには・・・？

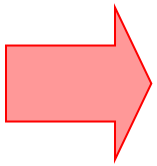
```
SELECT id FROM testtab
  WHERE CONTAINS(text, 'relational databases > 75')>0;
```

※ 式レベルでのしきい値指定は、CONTAINS関数の返り値を使って、
「WHERE CONTAINS(col, 'relational databases') > 75」のように指定してはいけない。
必ず「WHERE CONTAINS(col, 'relational databases > 75') > 0」の形で指定する。

シソーラス検索

- 意味上のあいまい検索が可能
  - 同義語検索 / 上位語検索 / 下位語検索
- 使用する演算子(一部抜粋)
  - 同義語検索(**SYN**: Synonym)
  - 上位語検索(**BT**: Broader Term)
  - 下位語検索(**NT**: Narrower Term)
    - 使用例

SYN(オラクル)
BT(日本)
NT(犬)



検索キーワードもしくは、定義された
同義語 / 上位語 / 下位語に
ヒットしたドキュメントを検索

（参考）シソーラス検索とは

- シソーラス展開や異表記語展開の後、各語について OR 検索することにより実現する全文検索のこと
- シソーラス検索の例
  - 狭義語　　「犬」と「番犬」
  - 広義語　　「犬」と「動物」
  - 異表記語　　「犬」と「イヌ」
  - 類義語　　「犬」と「猫」
  - 外来語　　「TEST」と「テスト」
「データベース」と「データーベース」
  - 省略形　　「Database」と「DB」
  - 慣用句　　「働く」と「額に汗する」
  - 表記の揺れ　　「行う」と「行なう」

シソーラス検索
シソーラスの設定手順

1. 同義語辞書を用意
   - 日本語シソーラス辞書は、データベース作成直後の段階では空の状態
2. 次のいずれかの方法でシソーラス情報をデータベースに読み込む
   - シソーラスローダー(ctxloadコマンド)を使ってバッチで読み込み
     - 予め、シソーラス情報をテキスト・ファイルとして用意する
       ※ このテキスト・ファイルのサンプルは次ページ
   - PL/SQLパッケージ(CTX_THES)を使って個別に読み込み
3. 検索を実行
   - シソーラスの演算子を利用

シソーラス・ローダー(ctxloadコマンド)による シソーラス情報の読み込み

- 使用する演算子(抜粋)
  - 同義語(SYN)
  - 上位語(BT)
  - 下位語(NT)
- 同義語辞書の例

thes.txt

```
オラクル
  SYN おらくる
  SYN oracle
日本
  SYN ニッポン
    BT アジア
    NT 東京
```

- 前提事項
  - シソーラスは、ISO-2788 および ANSI Z39.19 規格に基づいたフレームワークに対応

ctxloadコマンドの構文

- 構文

```
$ ctxload  -user username[/password][@sqlnet_address]
           -name object_name
           -file file_name
           [-thes]
           [-thescase y|n]
           [-thesdump]
           [-log file_name]
           [-trace]
           [-drop]
```

※ ctxload コマンドは、${ORACLE_HOME}/bin の下にある。

シソーラス・ローダー(ctxloadコマンド)による シソーラス情報の読み込み

- ctxloadコマンドの実行例

```
$ ctxload -user testctx/testctx -thes -name tech_thes2 -
thescase n -file thes.txt
Connecting...
Creating thesaurus tech_thes2...
Thesaurus tech_thes2 created...
Processing...
8 lines processed successfully
Beginning insert...8 lines inserted successfully
Disconnected
$
```

※ シソーラスは、上記コマンドで指定されたデータベース・ユーザーの所有となる
※複数のシソーラスを作成することも可能(検索時に、シソーラス演算子の引数で、どのシソーラス辞書を利用するのかを指定する)

CTX_THESパッケージによるシソーラス情報の読み込み

- ctxloadコマンドを利用する以外の方法として、CTX_THESパッケージによるシソーラス情報の読み込みが可能
- CTX_THESパッケージによるシソーラス情報の読み込みは次のステップで実行する
  1. CTX_THES.CREATE_THESAURUS プロシージャでシソーラスを作成
  2. CTX_THESパッケージの次のプロシージャを利用してシソーラス情報を個別に登録
     - CTX_THES.CREATE_RELATION
     - CTX_THES.CREATE_TRANSLATION

CTX_THESパッケージによるシソーラス情報の読み込み
シソーラスの作成

- CTX_THES.CREATE_THESAURUS プロシージャを利用してシソーラスを作成する

```
BEGIN
  CTX_THES.CREATE_THESAURUS
  (
    name      => 'thes',
    casesens => FALSE
  );
END;
/
```

作成するシソーラスで大/小文字を区別するかどうかを指定

※ シソーラスは、上記コマンドを実行するデータベース・ユーザーの所有となる
※複数のシソーラスを作成することも可能（検索時に、シソーラス演算子の引数で、どのシソーラス辞書を利用するのかを指定する）

CTX_THESパッケージによるシソーラス情報の読み込み
シソーラス情報の登録

- 次のリレーションを登録する

```
日本
  BT アジア
```

「『日本』の広義語は、『アジア』」と指定

このリレーションを追加するには、CTX_THES.CREATE_RELATION プロシージャの引数を次のように指定する

```
BEGIN
  CTX_THES.CREATE_RELATION (
    tname      => 'thes',
    phrase     => '日本',
    rel        => 'BT',
    relphrase => 'アジア'
  );
END;
/
```

シソーラス設定時の動作

- 広義語（Broader Term）
  - AをBの広義語に設定すると・・・
    ⇒ 自動的に、BはAの狭義語となる
- 狭義語（Narrower Term）
  - AをBの狭義語に設定すると・・・
    ⇒ 自動的に、BはAの広義語となる
- シノニム（同義語）（SYNonym）
  - AをBのシノニムに設定すると・・・
    ⇒ 自動的に、BはAのシノニムとなる

シソーラス検索用の演算子一覧

演算子	意味
BT、BTG、BTP、BTI	広義語(Broader Term)
NT、NTG、NTP、NTI	狭義語(Narrower Term)
SYN	シノニム(同義語)(SYNonym)
PT	優先語(Preferred Term)
RT	関連語(Related Term)
TT	最上位語(Top Term)
TR	翻訳語(Translation Term)
TRSYN	翻訳語シノニム(Translation Term Synonym)

シソーラス検索例

- 同義語検索(シソーラス名:DEFAULT)

```
SELECT id FROM testtab
  WHERE CONTAINS (text, 'SYN(オラクル)') > 0;
```

- 同義語検索(シソーラス名:thes を明示的に指定)

```
SELECT id FROM testtab
  WHERE CONTAINS (text, 'SYN(オラクル,thes)') > 0;
```

- 広義語検索(シソーラス名:DEFAULT)

```
SELECT id FROM testtab
  WHERE CONTAINS (text, 'BT(日本)') > 0;
```

シソーラス情報の参照

- CTX_USER_THESAURIビュー
（ユーザーが作成したシソーラスの一覧）

※ thesauri … thesaurusの複数形

```
SQL> SELECT * FROM ctx_user_thesauri;

THS_NAME
------------------------------
THES
```

- CTX_USER_THES_PHRASESビュー
（ユーザーが作成したシソーラスに登録されたフレーズ一覧）

```
SQL> SELECT * FROM ctx_user_thes_phrases;

THP_THESAURUS   THP_PHRASE   THP_QUALIFIER   THP_SCOPE_NOTE
--------------- ------------ --------------- ----------------
THES            アジア
THES            日本
```

シソーラス情報の参照

- ctxloadコマンドによるシソーラス情報のテキスト出力

```
$ ctxload -user test/test -thesdump -name thes -file out.txt
Connecting...
Writing thesaurus thes to file out.txt
5 lines processed successfully
Disconnected
$
```

out.txt

```
アジア
  NT  日本
日本
  BT  アジア
```

ファジー、ステミング

- ファジー(FAZZY演算子)
  →後述BASIC_WORDLISTのFUZZY_MATCH属性の項を参照(p.163～)
- ステミング(STEM($)演算子)
  →後述BASIC_WORDLISTのSTEMMER属性の項を参照(p.156～)

セクション検索

- セクション検索のための演算子
  - WITHIN(HTMLやXMLなどのタグ指定検索)
  - INPATH、HASPATH(XPath検索)
  - MDATA、SDATA、NDATA

→後述「セクション・グループ型」の項を参照(p.181～)

問合せ演算子のエスケープ

エスケープ文字	説明
{}	中カッコで、文字や記号から成る文字列をエスケープできる。中カッコで囲まれたもの全体が、エスケープ・シーケンスとみなされる。中カッコを使用して単一の文字をエスケープすると、エスケープされた文字が問合せ内の個別のトークンになる。
¥	バックスラッシュ文字を使用して、単一の文字または記号をエスケープできる。バックスラッシュの直後の文字のみがエスケープされる。たとえば、blue¥-green という問合せには、blue-green およびblue green が一致する。

ID	TEXT
1	Minato-ku

✕ `SELECT id FROM testtab WHERE CONTAINS(text,'Minato-ku')>0;`

○ `SELECT id FROM testtab WHERE CONTAINS(text,'Minato¥-ku')>0;`

○ `SELECT id FROM testtab WHERE CONTAINS(text,'{Minato-ku}')>0;`

CONTAINS関数における検索条件の指定方法

- 前述のCONTAINS問合せ演算子を利用して、検索条件を直接指定
- 前述のCONTAINS問合せ演算子と、後述（次ページ以降）の問合せテンプレートを組み合わせて、検索条件をカスタマイズして指定

問合せテンプレート（Query Template）
（Oracle9i Database Release 2（9.2）以降）

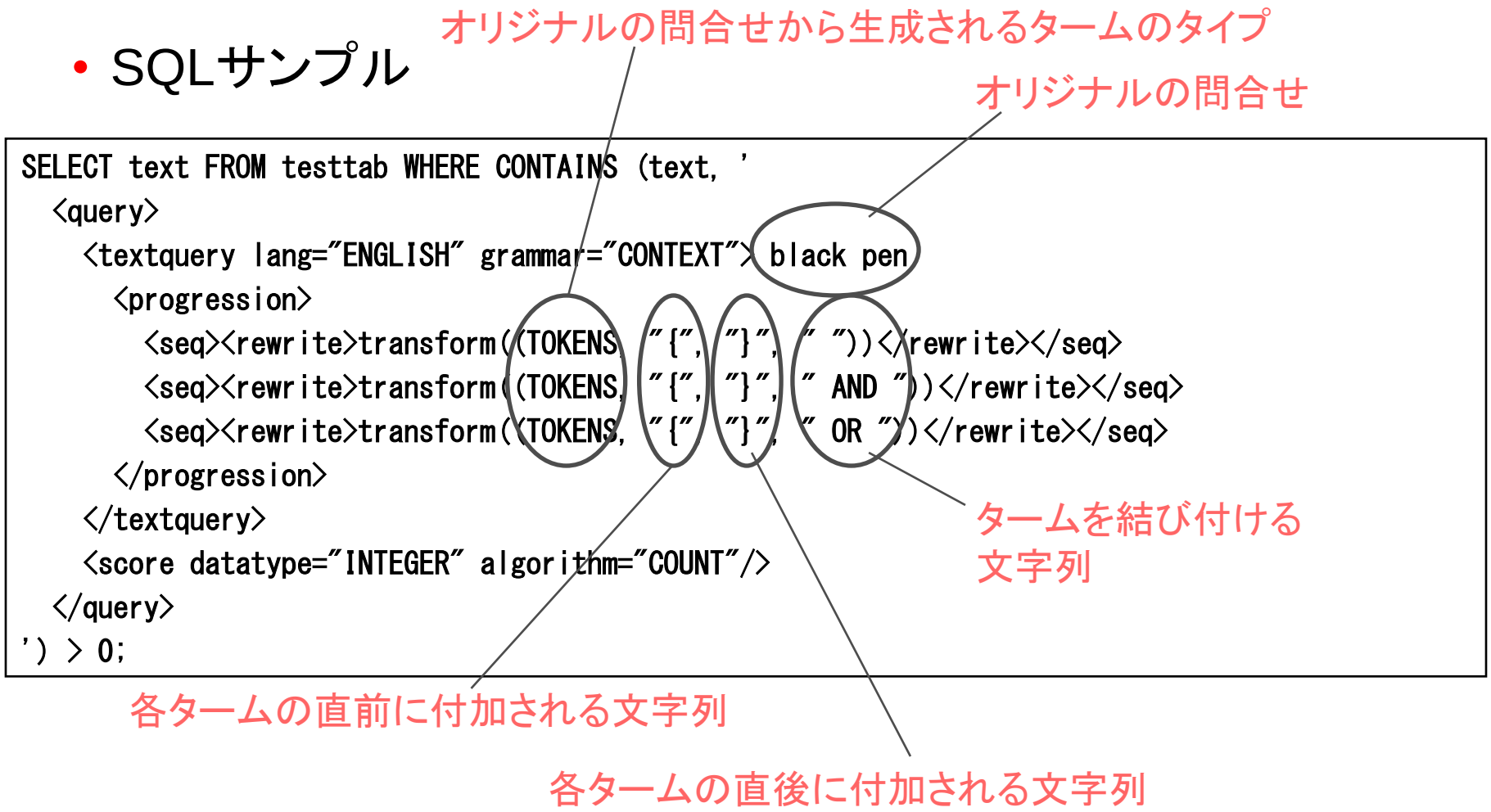
- 全文検索の条件をXML形式で記述する
  - このXMLは、CONTAINS関数の引数にそのまま渡される
- XMLの中で、以下を指定可能
  - 問合せリライト（Query Rewrite）
  - 問合せ緩和（Query Relaxation）
  - 問合せ言語（Query Language）
  - 代替スコア（Alternative Scoring）
  - 代替文法（Alternative Grammar）

問合せリライト(Query Rewrite)

- ユーザーの問合せ文字列を、別の問合せ文字列として解釈し直す
- 例えば、ユーザーの問合せ文字列が「kukui nut」の場合、検索アプリケーションではこの文字列を、「kukui nut」(フレーズ検索)、「kukui AND nut」(AND検索)、「kukui OR nut」(OR検索)のように解釈し直し、問合せを実行するケースが考えられる

問合せリライト ～ 使用例(1)

- SQLサンプル



問合せリライト ～ 使用例(2)

- 実行結果

```
TEXT
--------------------------------------------------------------------------------

Black Pen
Black XXX Pen
Pen Black
Pen
Black
```

※ 問合せリライトによって得られた問合せ結果に重複はない。

問合せ緩和(Query Relaxation)

- 問合せ結果を、検索条件の厳しいものから順に表示させる
- 次ページの例では、
  最も厳しい検索条件「{black} {pen}」(フレーズ検索)の次に、2番目に厳しい検索条件「{black} AND {pen}」、その次に、3番目に厳しい検索条件「{black} ACCUM {pen}」を、順に結果表示する。

問合せ緩和 ～ 使用例(1)

- SQLサンプル

```
SELECT text FROM testtab WHERE CONTAINS (text, '
  <query>
    <textquery lang="ENGLISH" grammar="CONTEXT">
      <progression>
        <seq>{black} {pen}</seq>
        <seq>{black} AND {pen}</seq>
        <seq>{black} OR {pen}</seq>
      </progression>
    </textquery>
    <score datatype="INTEGER" algorithm="COUNT"/>
  </query>
') > 0;
```

次第に条件が緩和される

※ 検索条件「{black} {pen}」は、「キーワード black、pen が、この順で、連続して出現する」という意味(フレーズ検索)

問合せ緩和 ～ 使用例(2)

- 実行結果

```
TEXT
--------------------------------------------------------------------------------
Black Pen
Black XXX Pen
Pen Black
Pen
Black
```

※問合せ緩和によって得られた問合せ結果に重複はない。

問合せ言語（Query Language）

- セッション言語とは異なる言語による検索を行うことが可能（多言語レクサー利用時に有効）
- 問合せテンプレートを利用しない場合、問合せ言語を設定するには、セッション言語（NLS_LANGUAGE）を明示的に変更する必要がある
- 問合せ言語は、textqueryタグのlang属性で指定
- 指定可能なlang値（一部抜粋）
  - JAPANESE（日本語）、ENGLISH（英語）、SIMPLIFIED（簡体字中国語）、TRADITIONAL（繁体字中国語）、KOREAN（韓国語）、FRENCH（フランス語）、GERMAN（ドイツ語）、DUTCH（オランダ語）など

    ※ 指定可能なlang値の一覧は、巻末付録「Oracleデータベースがサポートしている言語」を参照（p.280）

問合せ言語 ～ 使用例

- SQLサンプル

```
SELECT id FROM docs WHERE CONTAINS (text, '
  <query>
    <textquery lang="french">bon soir</textquery>
  </query>
')>0;
```

代替スコア（Alternative Scoring）

- 異なるスコアリング・アルゴリズム、スコア精度を指定
- スコア・アルゴリズム
  - DEFAULT ... 0から100までの数値。サルトンの式（Salton's Formula）（後述）により算出される
  - COUNT ... 検索条件の出現回数を返す。
- スコア精度
  - INTEGER ... 整数
  - FLOAT ... 浮動小数点数

代替スコア ～ 使用例(1)

- SQLサンプル

```
SELECT text, score(1) "SCORE" FROM testtab
  WHERE CONTAINS (text, '
    <query>
      <textquery grammar="CONTEXT" lang="english">dog</textquery>
      <score datatype="float" algorithm="DEFAULT"/>
    </query>
  ', 1) > 0;
```

代替スコア ～ 使用例(2)

- 実行結果

```
TEXT                          SCORE
----------------------------  ---------
dog bird cat                    3.9031
dog cat bird                    3.9031
```

代替文法（Alternative Grammar）

- CONTEXT索引でCTXCAT文法を使用したり、逆にCTXCAT索引でCONTEXT文法を使用したりすることが可能
- textqueryタグのgrammar属性で指定
- 指定可能なgrammar値：
  - CONTEXT ... CONTEXT索引で使用される文法
  - CTXCAT ... CTXCAT索引で使用される文法

CTXCAT索引で使用可能な問合せ演算子一覧

操作	構文	操作の説明
論理AND	a b c	a、b、cの全てを含む行を戻す。
論理OR	a | b | c	a、b、cのうち、少なくとも1つを含む行を戻す。
論理NOT	a - b	aを含み、bを含まない行を戻す。
空白なしのハイフン	a-b	通常の文字列として処理される。例えば、ハイフンがskipjoin（後述）として定義されている場合、web-siteなどのワードは、単一の問合せ語句websiteとして処理される。同様に、ハイフンがprintjoin（後述）として定義されている場合、web-siteなどのワードは、そのままweb-siteとして処理される。
" "	"a b c"	句"a b c"を含む行を戻す。
( )	(A B) | C	グループ設定をカッコで囲む。左記の問合せは、CONTEXT文法の「(A & B) | C」に相当。
ワイルドカード	term*	ワイルド・カード文字は、0（ゼロ）個以上の文字と一致する。

代替文法 ～ 使用例

- SQL文

```
SELECT text FROM testtab WHERE CONTAINS (text, '
  <query>
    <textquery grammar="CTXCAT">San Diego</textquery>
    <score datatype="integer"/>
  </query>
')>0;
```

※CTXCAT文法では、検索文字列「San Diego」は「『San』と『Diego』をともに含む」(AND検索)という意味であるため、上記検索によって、「San xxx Diego」、「Diego San」等の文字列を含むドキュメントがヒットする。

問合せテンプレートのDTD

- 問合せテンプレートは、以下のDTDに従う

```
<!ELEMENT query (textquery, score?)>
<!ELEMENT textquery (#PCDATA|progression)*>
<!ELEMENT progression (seq)+>
<!ELEMENT seq (#PCDATA|rewrite)*>
<!ELEMENT rewrite (#PCDATA)>
<!ELEMENT score EMPTY>
<!ATTLIST textquery grammar (context | ctxcat) #IMPLIED>
<!ATTLIST textquery language CDATA #IMPLIED>
<!ATTLIST score datatype (integer | float) "integer">
<!ATTLIST score algorithm (default | count) "default">
```

CONTAINS関数 使用上の注意点

- CONTAINS関数は、必ずWHERE句の中で、「>0」の条件指定によって利用する
  - 「>1」、「>75」など、0より大きい値を指定してはならない
    - 代わりに、しきい値（THRESHOLD演算子（>）を利用する
    - 例えば、スコアが75より大きい文書を検索するには、「WHERE CONTAINS（col, '... > 75'） > 0」と指定する
  - 「=0」を指定してはならない
    - 代わりに、「WHERE CONTAINS（...） > 0」の条件と、SQL関数のMINUS演算子を組み合わせる
  - SELECTリストの中で、CONTAINS関数を指定してはならない
    - 代わりに、SCORE演算子（後述）を利用する

検索(2) ～ その他の機能

※ この章では、検索演算子以外の、Oracle Textの検索機能について解説する

SCORE関数

- CONTAINS問合せによって生成されたスコア値を戻す
- SCORE演算子では、次のように、CONTAINS句にSCOREラベル値(この例の場合は「1」)を参照するように指定する

```
SELECT SCORE(1), title FROM newsindex
  WHERE CONTAINS(text, 'oracle', 1) > 0
  ORDER BY SCORE(1) DESC;
```

スコアとソート

- スコア
  - 検索条件に対する一致度を0以上100以下の整数で返す
    - 検索条件に一致する箇所が1つでもあれば、0より大きいスコアが付く
    - スコアが大きいほど、検索条件に一致する箇所が多い
    - スコアの表示には、SCORE演算子を用いる（CONTAINS演算子を用いない）
- ヒット順表示
  - スコア順にソートして、結果表示

```
SELECT SCORE(1), title FROM newsindex
  WHERE CONTAINS(text, 'oracle', 1) > 0
  ORDER BY SCORE(1) DESC;
```

デフォルトのスコア・アルゴリズム

- デフォルトのスコアは、0から100までの整数値
- サルトンの式（Salton's Formula）により算出される
  ※ 別名「tf-idf法」

$$\mathrm{Min}\left(\left[3f\left(1+\log_{10}\left(\mathrm{N}/n\right)\right)\right], 100\right)$$

f ＝ ドキュメント内の検索キーワード出現回数
N ＝ 検索対象のドキュメント数（全体）
n ＝ 検索キーワードを含むドキュメント数

※ [ ] はガウス記号。$[x]$ は、x を超えない最大の整数。

【参考】スコアが100になるために必要な検索キーワード出現回数（全体で1ドキュメントのみがヒットした場合）

検索対象のドキュメント数（全体）	ドキュメント内の検索キーワード出現回数
1	34
5	20
10	17
50	13
100	12
500	10
1,000	9
10,000	7
100,000	6
1,000,000	5

検索キーワードが11回出現する場合のスコアは、
$3 \times 11 \times (1 + \log(100 / 1)) = 33 \times (1 + 2) = 99.$

検索キーワードが12回出現する場合のスコアは、
$3 \times 12 \times (1 + \log(100 / 1)) = 36 \times (1 + 2) = 108.$

よって、スコアが100になるために必要な検索キーワードの出現回数は12.

ヒット件数のカウント

- SQLの集計関数COUNTを利用する
  ※ WHERE句にCONTAINS関数の条件を記述し、SELECT句にCOUNT関数を用いる
  - COUNT(*) の場合、該当行の**DOCID取得＋ROWID取得**まで行われる

```
SELECT COUNT(*) FROM testtab WHERE CONTAINS (text, 'オラクル') > 0;
```

  - COUNT(<*column_name*>) の場合、**DOCID取得＋ROWID取得＋列値取得**まで行われる

```
SELECT COUNT(id) FROM testtab WHERE CONTAINS (text, 'オラクル') > 0;
```

- **【推奨】**Oracle Textが提供するCTX_QUERYパッケージを利用する
  - COUNT_HITS関数によって該当行の**DOCID取得**まで行われる

```
SELECT CTX_QUERY.COUNT_HITS('TESTIDX', 'オラクル') FROM DUAL;
```

  - 「exact」引数をFALSEに設定すると、前回同期化以降に更新および削除されたドキュメントの情報が反映されない（その分高速）

CTX_QUERY.COUNT_HITS関数
CTX_QUERY.COUNT_HITS_CLOB_QUERY関数

- COUNT_HITS関数の構文

```
CTX_QUERY.COUNT_HITS(
    index_name   IN VARCHAR2,                    ---索引名
    text_query   IN VARCHAR2,                    ---テキスト問合せ
    exact        IN BOOLEAN  DEFAULT TRUE,
    part_name    IN VARCHAR2 DEFAULT NULL
) RETURN NUMBER;
```

- COUNT_HITS_CLOB_QUERY関数の構文

```
CTX_QUERY.COUNT_HITS_CLOB_QUERY(
    index_name   IN VARCHAR2,                    ---索引名
    text_query   IN CLOB,                        ---テキスト問合せ
    exact        IN BOOLEAN DEFAULT TRUE,
    part_name    IN VARCHAR2 DEFAULT NULL
) RETURN NUMBER;
```

コンポジット・ドメイン索引

※ Composite Domain Index

（Oracle Database 11g Release 1（11.1）以降）

- 定型列の情報もOracle Textの索引内に持つことで、全文検索と定型列の処理を組み合わせた検索を高速に実行するためのもの
- コンポジット・ドメイン索引で高速化される処理:
  - WHERE句で、全文検索と定型検索の両方が指定されているSELECT文の実行
  - WHERE句で全文検索との条件が指定され、ORDER BY句で定型列が指定されているSELECT文の実行
  - 上記2つの組み合わせ

コンポジット・ドメイン索引 ~ FILTER BY

- CONTAINS問合せを含むSELECT文のWHERE句で、CONTAINS問合せとは別に、定型検索の条件として指定される可能性のある列を指定する
- 指定可能なデータ型
  - CHAR型、NUMBER型、DATE型、VARCHAR2(*n*)型、RAW(*n*)型
  - VARCHAR2 型と RAW 型は、最大長の指定が必須
  - 指定可能な最大長は249
  - *<関数名>*(*<列名>*) などの式や仮想列は指定不可
- 完全一致検索、範囲検索のいずれにも対応
  - 評価対象のリレーショナル演算子:
    <、<=、=、>=、>、BETWEEN、LIKE(VARCHAR2型)

コンポジット・ドメイン索引 ~ ORDER BY

- CONTAINS問合せを含むSELECT文のORDER BY句で指定される可能性のある列名を指定する
- 指定可能なデータ型
  - CHAR型、NUMBER型、DATE型、VARCHAR2(*n*)型、RAW(*n*)型
  - VARCHAR2 型と RAW 型は、最大長の指定が必須
  - 指定可能な最大長は249
  - *<関数名>*(*<列名>*) などの式や仮想列は指定不可

コンポジット・ドメイン索引 ~ ORDER BY

- コストベース・オプティマイザは、次の場合にソート処理をテキスト索引を利用して実行する
  - CREATE INDEX時に指定されたORDER BY句と、SELECT時のORDER BY句が完全に一致しているとき
  - CREATE INDEX時に指定されたORDER BY句の先頭に指定された列が、SELECT時のORDER BY句で指定されたとき
  - CREATE INDEX時に指定されたORDER BY句の先頭に指定された列が、SELECT時のORDER BY句で、スコアの次に指定されたとき
  - CREATE INDEX時に指定されたORDER BY句の先頭に指定された列が、SELECT時のORDER BY句で、スコアの前に指定されたとき

コンポジット・ドメイン索引 ~ ORDER BY

次の索引が作成されているとする。

```
CREATE INDEX foox ON foo(d)
  INDEXTYPE IS CTXSYS.CONTEXT
  FILTER BY b, c
  ORDER BY a, b DESC;
```

このとき、次のSELECT文で、コンポジット・ドメイン索引がソート処理で利用されるのは・・・？

```
SELECT a, SCORE(1) FROM foo
  WHERE CONTAINS(d, 'oracle',1)>0
  AND c>100
  ORDER BY col_list;
```

次のORDER BY句では
コンポジットドメイン索引が利用される

○

```
a
a, b desc
SCORE(1), a
SCORE(1), a, b DESC
a, SCORE(1)
a, b DESC, SCORE(1)
```

次のORDER BY句では
コンポジットドメイン索引が利用されない

×

```
b
b,a
SCORE(1), b
b, SCORE(1)
a, b, c
a, b ASC （または、単に a, b）
```

ハイライト、スニペット

- 検索条件に一致した箇所を強調表示するには・・・？
  → CTX_DOCパッケージのプロシージャ、およびファンクションを利用する（これにより、全文検索の条件にヒットした箇所を特定のテキスト文字列（たとえば「<B>」と「</B>」）で囲むことができる）
- **テキスト索引がある場合**には次を利用
  ※ 通常は、問合せの後で処理するドキュメントを特定してから使用する
  - HIGHLIGHT
  - MARKUP
  - **SNIPPET、SNIPPET_CLOB_QUERY**
- **テキスト索引がない場合**には次を利用
  ※ 通常は、問合せの後で処理するドキュメントを特定してから使用する
  - POLICY_HIGHLIGHT
  - POLICY_MARKUP
  - **POLICY_SNIPPET**

CTX_DOC.SNIPPET関数

- 問合せ文字列(text_query引数)がVARCHAR2型の場合に利用する
- 構文

```
CTX_DOC.SNIPPET(
  index_name                IN VARCHAR2,
  textkey                   IN VARCHAR2,
  text_query                IN VARCHAR2,
  starttag                  IN VARCHAR2 DEFAULT '<b>',
  endtag                    IN VARCHAR2 DEFAULT '</b>',
  entity_translation        IN BOOLEAN  DEFAULT TRUE,
  separator                 IN VARCHAR2 DEFAULT '<b>...</b>'
)
RETURN VARCHAR2;
```

CTX_DOC.SNIPPET_CLOB_QUERY関数

- 問合せ文字列(text_query引数)がCLOB型の場合に利用する
- 構文

```
CTX_DOC.SNIPPET_CLOB_QUERY(
   index_name                IN VARCHAR2,
   textkey                   IN VARCHAR2,
   text_query                IN CLOB,
   starttag                  IN VARCHAR2 DEFAULT '<b>',
   endtag                    IN VARCHAR2 DEFAULT '</b>',
   entity_translation        IN BOOLEAN  DEFAULT TRUE,
   separator                 IN VARCHAR2 DEFAULT '<b>...</b>'
)
RETURN VARCHAR2;
```

CTX_DOC.SNIPPET関数の使用例

- 準備(表作成、索引作成)

```
CREATE TABLE testtab (id NUMBER PRIMARY KEY, text VARCHAR2(4000));
INSERT INTO testtab VALUES (1,'日本オラクル株式会社');
COMMIT;
BEGIN
  CTX_DDL.CREATE_PREFERENCE('jvl','JAPANESE_VGRAM_LEXER');
END;
/
CREATE INDEX testidx ON testtab (text)
  INDEXTYPE IS CTXSYS.CONTEXT PARAMETERS ('LEXER jvl');
```

- CTX_DOC.SNIPPET 関数の実行

```
SQL> SELECT CTX_DOC.SNIPPET('testidx',1,'オラ|株式') result
  2    FROM testtab WHERE CONTAINS (text, 'オラ|株式') > 0

RESULT
--------------------------------------------------------------------------------
日本<b>オラ</b>クル<b>株式</b>会社
```

CTX_DOC.POLICY_SNIPPET関数

- 問合せ文字列(text_query引数)がVARCHAR2型の場合に利用する
- 構文

```
CTX_DOC.POLICY_SNIPPET(
  policy_name                IN VARCHAR2,
  document                   IN [VARCHAR2|CLOB|BLOB|BFILE],
  text_query                 IN VARCHAR2,
  language                   IN VARCHAR2 DEFAULT NULL,
  format                     IN VARCHAR2 DEFAULT NULL,
  charset                    IN VARCHAR2 DEFAULT NULL,
  starttag                   IN VARCHAR2 DEFAULT '<b>',
  endtag                     IN VARCHAR2 DEFAULT '</b>',
  entity_translation         IN BOOLEAN  DEFAULT TRUE,
  separator                  IN VARCHAR2 DEFAULT '<b>...</b>'
)
RETURN VARCHAR2;
```

CTX_DOC.POLICY_SNIPPET_CLOB_QUERY関数

- 問合せ文字列(text_query引数)がCLOB型の場合に利用する
- 構文

```
CTX_DOC.POLICY_SNIPPET_CLOB_QUERY(
  policy_name                IN VARCHAR2,
  document                   IN [VARCHAR2|CLOB|BLOB|BFILE],
  text_query                 IN CLOB,
  language                   IN VARCHAR2 DEFAULT NULL,
  format                     IN VARCHAR2 DEFAULT NULL,
  charset                    IN VARCHAR2 DEFAULT NULL,
  starttag                   IN VARCHAR2 DEFAULT '<b>',
  endtag                     IN VARCHAR2 DEFAULT '</b>',
  entity_translation         IN BOOLEAN  DEFAULT TRUE,
  separator                  IN VARCHAR2 DEFAULT '<b>...</b>'
)
RETURN VARCHAR2;
```

CTX_DOC.POLICY_SNIPPET関数の使用例

- 準備(ポリシー作成)

```
BEGIN
  CTX_DDL.CREATE_PREFERENCE('jvl','JAPANESE_VGRAM_LEXER');
  CTX_DDL.CREATE_POLICY(
    policy_name => 'my_policy',
    lexer       => 'jvl'
  );
END;
/
```

- CTX_DOC.SNIPPET 関数の実行

```
SQL> SELECT CTX_DOC.POLICY_SNIPPET('my_policy','日本オラクル株式会社
','オラ|株式') result FROM DUAL;

RESULT
--------------------------------------------------------------------------------
日本<b>オラ</b>クル<b>株式</b>会社
```

問合せ文字列の解析

- 次のプロシージャを利用して、問合せ式の解析結果を参照できる
  - CTX_QUERY.EXPLAIN
    問合せ文字列（text_query引数）がVARCHAR2型の場合に利用する
  - CTX_QUERY.EXPLAIN_CLOB_QUERY
    問合せ文字列（text_query引数）がCLOB型の場合に利用する
- 問合せ文字列の解析結果は、次スライドの結果表に格納される
- 想定される利用シーン
  - レクサーによるトークン分割結果を確認したい時
  - ワイルドカード検索による拡張結果を確認したい時
  - シソーラス検索の展開結果を確認したい時　・・・等

問合せ文字列の解析
結果表

※ 問合せ文字列の解析結果は、この表に格納される。

- EXPLAIN結果表の構造

列名	データ型	説明
EXPLAIN_ID	VARCHAR2(30)	EXPLAINプロシージャ実行時に指定したexplain_id値。
ID	NUMBER	問合せ実行ツリーの各ノードに割り当てられた数字。
PARENT_ID	NUMBER	親ノードのID。ルート・オペレーション・ノード（ID=1）では、PARENT_ID=0。
OPERATION	VARCHAR2(30)	実行する内部操作の名前。
OPTIONS	VARCHAR2(30)	OPERATION列で示された操作のバリエーションを表す文字。
OBJECT_NAME	VARCHAR2(80)	セクション名、ワイルド・カード語句、重み、しきい値、索引で検索する語句。
POSITION	NUMBER	同じPARENT_IDを持つノードの処理順序。1から開始。
CARDINALITY	NUMBER	予備。将来の互換性のために必要。

問合せ文字列の解析
CTX_QUERY.EXPLAIN使用例

- 準備(結果表を作成)
  ※この表に問合せ文字列の解析結果が格納される

```
CREATE TABLE exptab (
  explain_id   VARCHAR2(30),
  id           NUMBER,
  parent_id    NUMBER,
  operation    VARCHAR2(30),
  options      VARCHAR2(30),
  object_name  VARCHAR2(80),
  position     NUMBER,
  cardinality  NUMBER
);
```

問合せ文字列の解析
CTX_QUERY.EXPLAIN使用例

- 準備(検索対象表作成、索引作成)

```
CREATE TABLE testtab (
  id    NUMBER  PRIMARY KEY,
  text  VARCHAR2(4000)
);
INSERT INTO testtab VALUES (1,'日本オラクル株式会社');
INSERT INTO testtab VALUES (2,'ora oracle oracular oracy');
COMMIT;
BEGIN
  CTX_DDL.CREATE_PREFERENCE('jvl','JAPANESE_VGRAM_LEXER');
END;
/
CREATE INDEX testidx ON testtab (text)
  INDEXTYPE IS CTXSYS.CONTEXT PARAMETERS ('LEXER jvl');
```

問合せ文字列の解析
CTX_QUERY.EXPLAIN使用例

問合せ文字列「オラクル」を解析

```
BEGIN
  CTX_QUERY.EXPLAIN(
    index_name    => 'TESTIDX',
    text_query    => 'オラクル',
    explain_table => 'exptab',
    sharelevel    => 1,
    explain_id    => '[Case1] オラクル',
    part_name     => NULL
  );
END;
/
```

問合せ文字列「ora%」を解析

```
BEGIN
  CTX_QUERY.EXPLAIN(
    index_name    => 'TESTIDX',
    text_query    => 'ora%',
    explain_table => 'exptab',
    sharelevel    => 1,
    explain_id    => '[Case2] ora%',
    part_name     => NULL
  );
END;
/
```

問合せ文字列の解析
CTX_QUERY.EXPLAIN使用例

- 結果表を参照

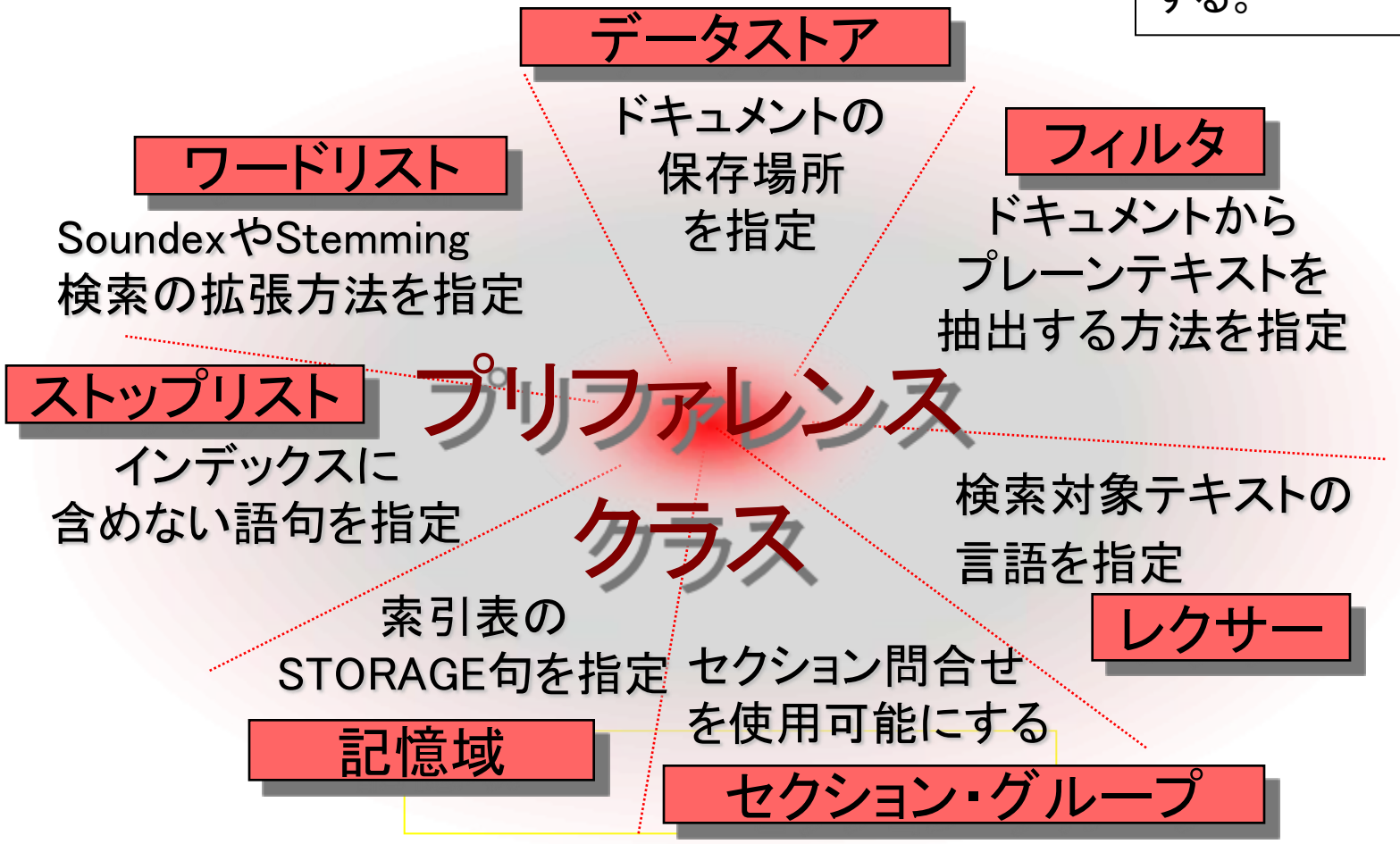
```
SQL> COLUMN EXPLAIN_ID FORMAT A20
SQL> COLUMN OPERATION FORMAT A20
SQL> COLUMN OPTIONS FORMAT A10
SQL> COLUMN OBJECT_NAME FORMAT A20
SQL> SELECT * FROM exptab ORDER BY explain_id, id;

EXPLAIN_ID           ID  PARENT_ID OPERATION       OPTIONS    OBJECT_NAME    POSITION CARDINALITY
-------------------- --- ---------- --------------- ---------- -------------- ---------- -----------
[Case1] オラクル       1          0 PHRASE                                            1
[Case1] オラクル       2          1 VGRAM0                     オラ                   1
[Case1] オラクル       3          1 VGRAM0                     ラク                   2
[Case1] オラクル       4          1 VGRAM0                     クル                   3
[Case2] ora%           1          0 EQUIVALENCE                ORA%                   1
[Case2] ora%           2          1 WORD                       ORA                    1
[Case2] ora%           3          1 WORD                       ORACLE                 2
[Case2] ora%           4          1 WORD                       ORACULAR               3
[Case2] ora%           5          1 WORD                       ORACY                  4
```

索引作成(1) ～ プリファレンス

※ この章では、プリファレンスによるテキスト索引のカスタマイズ方法について解説する

プリファレンスの種類と役割

任意のフィルタ、レクサなどを使用するために、プリファレンスを作成し、必要に応じて属性情報を設定する。



データストア型

- 検索対象のドキュメントが格納されている場所を指定する
- データベース内の表、マテリアライズド・ビューのほか、ファイルシステム上、インターネット上のリソースを指定可能
  - データベース表の(1つまたは複数の)列
    - 検索対象として指定可能なデータ型:
      CHAR、VARCHAR、VARCHAR2、BLOB、CLOB、BFILE、
      XMLType、URIType
  - ファイルシステム上のファイル
    - ローカル・ファイル・システム
    - リモート・マシンのファイル・システム(NFSマウント)
  - イントラネット上、あるいはインターネット上のファイル
    - URL で指定する(HTTP または FTP)
  - ユーザー定義
    - PL/SQLプロシージャの実行結果を検索対象とする

データストア型一覧

- **DIRECT_DATASTORE**
- **MULTI_COLUMN_DATASTORE**
- **FILE_DATASTORE**
- URL_DATASTORE
- DETAIL_DATASTORE
- NESTED_DATASTORE
- **USER_DATASTORE**

DIRECT_DATASTORE

- データベース表の単一列を検索対象とする場合に指定する
- 検索対象として指定可能なデータ型：
  CHAR、VARCHAR、VARCHAR2、BLOB、CLOB、BFILE、XMLType、URIType

※ BFILE、URIType については、参照先データに更新があっても、データベース表に更新がない限り、索引データは更新されない

MULTI_COLUMN_DATASTORE
（Oracle8i Database Release 8.1.7以降）

- データベース表の複数列を検索対象とする場合に指定する
- COLUMNS属性
  - 検索対象の列名をコンマ区切りで列挙する
  - NUMBER型、DATE型についても指定可能（ただし、文字列データとして索引が作成される。TO_CHAR関数でフォーマット指定可能）
  - RAW型についても指定可能、XMLType型は指定できない
- FILTER属性
  - COLUMNS属性に指定された各列にフィルタ処理を適用するか否かを指定する
  - テキスト形式の列とバイナリ形式の列が混在していても、列毎にFILTER属性を指定することが可能
- DELIMITER属性
  - COLUMN_NAME_TAG ... COLUMNS属性で指定された各列を、タグ付き文書として返す
    例：<列名a>列値a</列名a><列名b>列値b</列名b><列名c>列値c</列名c>...

    ※ これを BASIC_SECTION_GROUP (後述) と組み合わせることで、「特定の列に対象を限定した検索」、「全ての列を対象とした検索」の両者が可能となります。
  - NEWLINE ... COLUMNS属性で指定された各列を、改行コードで区切って返す

MULTI_COLUMN_DATASTORE 使用上の注意事項

- 「CREATE INDEX <*索引名*> ON <*表名*>(<*列名*>)」で指定される列は、表の中の1列を指定するが、実際には column_list 属性で指定された全ての列が検索対象となっている
  - 特に、CREATE INDEX文で指定された列が column_list に含まれていない場合、CREATE INDEX文で指定された列は検索対象にならない
- Oracle Text は、CREATE INDEX文で指定された列の更新状況のみを監視している
  - このため、CREATE INDEX文で指定された列以外で更新（UPDATE）が発生し、かつ、その更新を索引に反映したい場合には、CREATE INDEX文で指定された列についても同時にUPDATEする必要がある
    - この実装は、トリガーを使えばよい。このとき、CREATE INDEX文で指定された列に対する更新処理は、更新前と更新後のデータ内容が同一であっても問題ない
  - 行単位のDML処理（INSERT、DELETE等）については、特に考慮する必要なし

MULTI_COLUMN_DATASTORE 使用例

- プリファレンス作成例

```
BEGIN
  CTX_DDL.CREATE_PREFERENCE('my_multi','MULTI_COLUMN_DATASTORE');
  CTX_DDL.SET_ATTRIBUTE('my_multi', 'COLUMNS','foo,bar');
  CTX_DDL.SET_ATTRIBUTE('my_multi','FILTER','N,Y');
END;
/
```

bar列に対してのみフィルタ処理を実行

foo列とbar列を検索対象に指定

- 上記プリファレンスを利用して索引を作成すると、列データはタグ付けされ、各行に対して次のようなタグ付き文字列が生成される(このタグ付き文字列は、ユーザーが直接確認することはできない)

```
<FOO>foo contents</FOO><BAR>bar filtered contents</BAR>
```

FILE_DATASTORE

- データベース表の単一列に格納されたファイルパスに相当するファイルを検索対象とする場合に指定する
- ファイル本体の更新、削除は、データベース表に個別に反映する必要がある

  ※ Oracle Text は、CREATE INDEX文で指定された列の更新状況のみを監視している

FILE_DATASTORE 使用例

/mydocs
- first.txt
- second.txt

プリファレンス作成

```
BEGIN
  CTX_DDL.CREATE_PREFERENCE('common_dir','FILE_DATASTORE');
  CTX_DDL.SET_ATTRIBUTE('common_dir','PATH','/mydocs');
END;
/
```

表作成、データ格納

```
CREATE TABLE mytable(id NUMBER PRIMARY KEY, docs VARCHAR2(2000));
INSERT INTO mytable VALUES(111555,'first.txt');
INSERT INTO mytable VALUES(111556,'second.txt');
COMMIT;
```

索引作成

```
CREATE INDEX myindex ON mytable(docs)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('DATASTORE common_dir');
```

USER_DATASTORE

- PL/SQLプロシージャのIN OUT変数の返り値を検索対象とする

```
procedure (r IN ROWID, c IN OUT NOCOPY output_type)
```

- PL/SQLで記述可能な任意の仕組みでデータを収集可能
- プロシージャ名は任意
- *output_type* に指定可能なデータ型:
  CLOB(デフォルト)、BLOB、
  CLOB_LOC(CLOBロケータ)、BLOB_LOC(BLOBロケータ)、
  VARCHAR2
- CREATE INDEX文の列指定では、表の中の1列を指定するが、実際には必ずしもこの列が検索対象である必要はない。PL/SQLプログラムの中で、各ROWIDに対して、任意の動作をさせた結果が検索対象となる

USER_DATASTORE 使用例(CLOB)

表作成

```
CREATE TABLE articles(
  id       NUMBER,
  author   VARCHAR2(80),
  title    VARCHAR2(120),
  text     CLOB
);
```

プロシージャ作成

```
CREATE PROCEDURE myproc(rid IN ROWID, tlob IN OUT CLOB NOCOPY) is
BEGIN
  FOR c1 IN (SELECT author, title, text FROM articles WHERE ROWID = rid)
  LOOP
    DBMS_LOB.WRITEAPPEND(tlob, LENGTH(c1.title) , c1.title );
    DBMS_LOB.WRITEAPPEND(tlob, LENGTH(c1.author), c1.author);
    DBMS_LOB.WRITEAPPEND(tlob, LENGTH(c1.text)  , c1.text  );
  END LOOP;
END;
/
```

プリファレンス作成

```
BEGIN
  CTX_DDL.CREATE_PREFERENCE('myud', 'USER_DATASTORE');
  CTX_DDL.SET_ATTRIBUTE('myud', 'PROCEDURE', 'myproc');
  CTX_DDL.SET_ATTRIBUTE('myud', 'OUTPUT_TYPE', 'CLOB');
END;
/
```

USER_DATASTORE 使用例(BLOB_LOC)

プロシージャ作成

```
CREATE PROCEDURE myds(rid IN ROWID, dataout IN OUT NOCOPY BLOB)
IS
  l_dtype  VARCHAR2(10);
  l_pk     NUMBER;
BEGIN
  SELECT dtype, pk INTO l_dtype, l_pk FROM mytable WHERE ROWID = rid;
  IF (l_dtype = 'MOVIE') THEN
    SELECT movie_data INTO dataout FROM movietab WHERE fk = l_pk;
  ELSIF (l_dtype = 'SOUND') THEN
    SELECT sound_data INTO dataout FROM soundtab WHERE fk = l_pk;
  END IF;
END;
/
```

```
BEGIN
  CTX_DDL.CREATE_PREFERENCE('myud', 'USER_DATASTORE');
  CTX_DDL.SET_ATTRIBUTE('myud', 'PROCEDURE', 'myds');
  CTX_DDL.SET_ATTRIBUTE('myud', 'OUTPUT_TYPE', 'BLOB_LOC');
END;
/
```

プリファレンス作成

フィルタ型

- 検索対象のドキュメントからテキスト文字列を抽出する方法を指定する
- 検索対象がテキスト形式の場合にはこの部分はスキップ
- 標準付属のフィルタ
  - 200種類以上のドキュメントをフィルタ処理可能
- ユーザー定義
  - OSコマンドの実行結果を検索対象とする
  - PL/SQLプロシージャの実行結果を検索対象とする

フィルタ型一覧

- **AUTO_FILTER**
- CHARSET_FILTER
- **NULL_FILTER**
- MAIL_FILTER ※日本語を扱えない
- **USER_FILTER**
- **PROCEDURE_FILTER**

AUTO_FILTER
（Oracle Database 10g Release 2（10.2）以降）

- 200種類以上のファイル形式に対応
  - PDF、MS Office（Word、Excel、PowerPoint）、StarOffice、OpenOffice、一太郎、Lotus 1-2-3、等
- 利用されているフィルタ・モジュール
  - DB 10.2.0.1～10.2.0.4、DB 11.1.0.1～11.1.0.6
    → Autonomy（旧Verity）KeyView Export
  - DB 10.2.0.5～、DB 11.1.0.7～、DB 11.2～
    → Oracle（旧Stellent）Outside in HTML Export
- Oracle Database 10g Release 1（10.1）まで提供されていた INSO_FILTERに代わるもの

NULL_FILTER

- フィルタ処理を行う必要がない場合に指定する
  - 検索対象のデータ型が、CHAR、VARCHAR、VARCHAR2、CLOB、XMLType のいずれかである場合
  - 上記以外のデータ型であっても、検索対象がテキスト形式（HTML、XMLなどを含む）の場合
    - 検索対象のキャラクタセットが、データベース・キャラクタセットと異なる場合には、CREATE INDEX 文で CHARSET COLUMN 属性を指定することで、キャラクタセット変換が行える（後述）

USER_FILTER

- 外部フィルタを呼び出す場合に指定する
- COMMAND属性に、外部フィルタの実行コマンドを指定する
  - 指定できる外部フィルタの実行コマンドは1つのみ（この1つのコマンドで、全ての文書のフィルタ処理を行えなければならない）
  - 実行コマンドは、次の引数で動作する必要がある
    - 第1引数 ... フィルタ適用対象の入力ファイル名
    - 第2引数 ... フィルタ適用後の出力ファイル名
  - 指定されるコマンドの実行ファイルは、$ORACLE_HOME/ctx/bin（UNIXの場合）%ORACLE_HOME%¥ctx¥bin（Windowsの場合）に置かれている必要がある

※ Oracle Database 10g Release 2（10.2）以前のWindowsプラットフォームでは、%ORACLE_HOME%¥bin に置かれている必要がある。

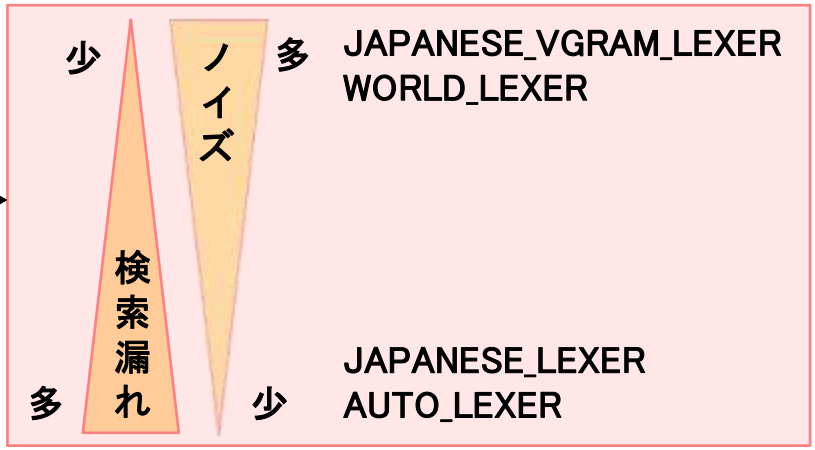
PROCEDURE_FILTER
（Oracle8i Database Release 8.1.7以降）

- ストアド・プロシージャによるフィルタ処理を実行する場合に指定する
- 指定可能なストアド・プロシージャは、
  - 入力データ型：BLOB、CLOB、VARCHAR2、FILEのいずれか
  - 出力データ型：CLOB、VARCHAR2、FILEのいずれか
    ※ FILEを指定する場合、ストアド・プロシージャのデータ型としては、VARCHAR2型となる
- ストアド・プロシージャのシグネチャ例

```
PROCEDURE ( IN BLOB     , IN OUT NOCOPY CLOB     )
PROCEDURE ( IN CLOB     , IN OUT NOCOPY CLOB     )
PROCEDURE ( IN VARCHAR2 , IN OUT NOCOPY CLOB     )
PROCEDURE ( IN BLOB     , IN OUT NOCOPY VARCHAR2 )
PROCEDURE ( IN CLOB     , IN OUT NOCOPY VARCHAR2 )
PROCEDURE ( IN VARCHAR2 , IN OUT NOCOPY VARCHAR2 )
```

レクサー型

- 検索対象のテキスト文字列からトークン（索引情報の最小単位）を生成する方法を指定する
  - トークンの生成方法は、検索条件に対するヒットや、索引作成パフォーマンス、検索パフォーマンス、索引サイズなどに影響を与える
- 日本語対応レクサー
  - JAPANESE_VGRAM_LEXER
  - JAPANESE_LEXER
  - WORLD_LEXER（多言語レクサー）
  - AUTO_LEXER（多言語レクサー）
  - MULTI_LEXER（多言語レクサー）
    ※ 使用するレクサーを言語毎に明示的に指定する
  - USER_LEXER（ユーザー定義レクサー）



- 日本語非対応のレクサー（一部抜粋）
  - BASIC_LEXER（全ての「スペース区切り言語」に対応。西ヨーロッパ系言語である英語、フランス語、ドイツ語、イタリア語、スペイン語などを含む。日本語は扱えない）

レクサー型一覧

- **AUTO_LEXER**
- **BASIC_LEXER**
- **MULTI_LEXER**
- CHINESE_VGRAM_LEXER
- CHINESE_LEXER
- **JAPANESE_VGRAM_LEXER**
- **JAPANESE_LEXER**
- KOREAN_MORPH_LEXER
- **USER_LEXER**
- **WORLD_LEXER**

BASIC_LEXER

- 全ての「スペース区切り言語」に対応
  - 西ヨーロッパ系言語(英語、フランス語、ドイツ語、イタリア語、スペイン語など)を含む
  - 日本語には非対応
- スペース区切りでトークンを生成する
  - 例:文字列「Oracle Text」は、BASIC_LEXERによって、「ORACLE」「TEXT」という2つのトークンに分割される
  - 例:文字列「aaa123 bbb456 789ccc」は、BASIC_LEXERによって、「AAA123」「BBB456」「789CCC」という3つのトークンに分割される
- 全てのデータベースキャラクタセットで利用可能

BASIC_LEXERの属性

- **continuation**
- numgroup
- numjoin
- **printjoins**
- punctuations
- **skipjoins**
- startjoins
- endjoins
- **whitespace**
- newline
- base_letter
- base_letter_type
- override_base_letter
- **mixed_case**
- composite
- index_stems
- index_themes
- index_text
- prove_themes
- theme_language
- alternate_spelling
- new_german_spelling

BASIC_LEXERの属性

- continuation
  - ワードが次の行に続き、そのワードを1つのトークンとして索引付けする必要があることを示す文字を指定する
  - 最も一般的な連結文字はハイフン「-」および円記号「¥」
- printjoins
  - 英数字以外の文字を指定する
  - この文字は、ワード内（先頭、中程、末尾）にあれば英数字として処理され、テキスト索引にトークンとともに組み込まれる
  - 連続しているprintjoinsも同様に処理される
  - たとえば、ハイフン「-」とアンダースコア「_」がprintjoinsとして定義されている場合、「pseudo-intellectual」や「_file_」などの語句は、そのまま「pseudo-intellectual」および「_file_」としてテキスト索引に格納される

BASIC_LEXERの属性

- skipjoins
  - 英数字以外の文字を指定する
  - この文字がワード内で使用されている場合は、そのワードを単一のトークンとして識別する
  - ただし、その文字はテキスト索引内にトークンとともに格納されない
  - たとえば、ハイフン「-」がskipjoinsとして定義されている場合、ワード「pseudo-intellectual」は、テキスト索引に、「pseudointellectual」として格納される

【注】printjoinsおよびskipjoinsは相互に排他的。同じ文字を両方の属性に指定することはできない。

BASIC_LEXERの属性

- whitespace
  - whitespaceの事前定義のデフォルト値は、半角スペース文字とタブ文字（これらの値は変更できない）
  - whitespaceとして文字を指定すると、指定された文字がデフォルト値に追加される
- mixed_case
  - テキスト索引に対して大・小文字の区別をするか否かを指定する
  - デフォルトは「NO」（トークンはすべて大文字に変換され、大・小文字の区別をしない）
  - 「YES」に設定する（＝大・小文字の区別を有効にする）と、その索引に対する問合せでは常に大文字と小文字が区別される（区別しない検索は不可能になる）

JAPANESE_VGRAM_LEXER

- 日本語レクサーの1つ
- 日本語文字は基本的に2文字ずつ(英数字部分はBASIC_LEXER と同様にスペース区切り)で、それぞれトークンを生成する
  - 例:文字列「オラクルJapan」は、「オラ」「ラク」「クル」「ル」「JAPAN」のようにトークン分割される
- 次のデータベースキャラクタセットで利用可能
  - AL32UTF8、UTF8
  - JA16SJIS、JA16SJISTILDE、JA16SJISYEN
  - JA16EUC、JA16EUCTILDE、JA16EUCYEN

JAPANESE_VGRAM_LEXER

N-Gram インデックス

- 文書に現れるすべての文字をある固定の長さNの連続する文字列として扱い、索引登録と検索を行う方式

文書1：新宿区役所へ行く

文書2：私は新宿区民です

テキスト索引

トークン	文書番号と出現位置	
新宿	1-1	2-3
宿区	1-2	2-4
区役	1-3	
役所	1-4	
所へ	1-5	
へ行	1-6	
行く	1-7	
私は	2-1	
は新	2-2	
区民	2-5	
民で	2-6	
です	2-7	

JAPANESE_VGRAM_LEXER

N-Gram インデックス

「新宿区民」で全文検索をした場合の内部動作

① ２文字ずつトークンに切り分ける。

新宿　宿区　区民

② トークンを索引と照らし合せ、出現文書が同じで、かつそれらのトークンが連続して出現している文書番号を探す。

トークン「新宿」は索引から [1-1] と [2-3] に出現していることが分かる。
トークン「宿区」は索引から [1-2] と [2-4] に出現していることが分かる。
トークン「区民」は索引から [2-5] に出現していることが分かる。

③ 検索語句「新宿区民」は文書2の3番目から6番目（「区民」が5番目に始まるので、それに1加えた値）に出現していることがわかる。

JAPANESE_VGRAM_LEXER

Oracle Text のアルゴリズム

- V-Gramインデックス
  - N-Gramインデックスを改良した方法
  - 隣接する文字によって、トークン長を**可変**に

Variable

例えば「ジャイブ」をトークンに分けると・・・

N=2の場合、「ジャ」「ャイ」「イブ」

検索時に「ャ」で始まる語句を指定することはないので、

トークンから「ャイ」を除き、「ジャ」を「ジャイ」まで拡張

「ジャイ」「イブ」「ブ」

最後の一文字は「ブ」一文字での検索に対応するために必要

トークンの数を減らし、検索効率を向上

JAPANESE_VGRAM_LEXER
トークンの分割例

英数字はシングルバイトの大文字に統一されてトークンとして登録される

OracleText → ORACLETEXT

Oracle□Text → ORACLE | TEXT

※シングルバイトに挟まれた空白は単語の区切りと認識

マルチバイト文字は漢字・カタカナ・平仮名でトークンが区切られる

日本オラクル
日本□オラクル
→ 日本 | 本 | オラ | ラク | クル | ル

※マルチバイトに挟まれた空白は無視されます

連結文字（を、ん、長音 etc）が含まれる場合

りんご → りんご | ご

インターフェース → インタ | ターフ | フェース | ス

プリン製造 → プリ | リン製 | 製造 | 造

JAPANESE_VGRAM_LEXER

検索語句に関する注意事項（1）

ワイルド・カード検索に関して

言語	全文検索例	結果
英語	Oracle%	使用可能で「OracleText」はヒットする
	%Oracle	BASIC_WORDLIST オブジェクトのSUBSTRING_INDEX 属性を使用することを推奨
	%Oracle%	
日本語	オラクル%	日本語に対してワイルドカードは使用不可
	%オラクル	
	%オラクル%	

本文「製品番号EEE00AA0001が生産開始となった」に対して“00AA”で全文検索を行ってもヒットしない。

JAPANESE_VGRAM_LEXER

検索語句に関する注意事項（2）

日本語の場合、内部的に Like 検索が行われる場合がある

一文字検索の場合

本 **海** **愛**

日本海 → **日本** **本海** **海**

「日本海」は上記のように分割される。「本」の検索で「日本海」をヒットさせるため、「本%」として検索される

語尾が長音や「ャ」「ュ」「ョ」などで終る語句

キャッシュ

コンピューター

シューマン → **シューマ** **マン**

長音や「ャ」などは次の文字まで連結して切り出される。「シュ」の検索で「シューマン」をヒットさせるために「シュ%」として検索される

JAPANESE_VGRAM_LEXER

内部的な LIKE 検索

検索語句	内部で行われる テキスト索引($I 表)への検索
キャッシュ	キャッシ と シュ に分割 … where token_text = 'キャッシ' … where token_text like 'シュ%'
カー	これ以上分割できない … where token_text like 'カー%'
人	これ以上分割できない … where token_text like '人%'

内部的にワイルドカードを用いた
LIKE 検索が行われる

JAPANESE_VGRAM_LEXER

何故 LIKE 検索が行われるのか？

語句 **カーレース** にテキスト索引を作成すると...

トークン **カーレ** **レース** **ス** に分割

テキスト索引（$I 表）

token_text	....
カーレ	...
レース	...
ス	...

語尾が長音や小さい文字（ャ、ュ、ョ）の場合、一文字検索の場合には、LIKE 検索を実行しないとヒットしなくなってしまう。

LIKE 検索の場合、ヒット数が多くなるためパフォーマンス劣化

JAPANESE_VGRAM_LEXER
属性

- **delimiter**
  ※ Oracle Database 10g Release 1（10.1）以降で利用可能
- **mixed_case_ASCII7**
  ※ Oracle Database 10g Release 2（10.2）以降で利用可能

JAPANESE_VGRAM_LEXER
属性

- delimiter

  ※ Oracle Database 10g Release 1（10.1）以降で利用可能

  - 全角のスラッシュや全角の中黒など、日本語の特定の空白文字を削除した上でトークンを生成する場合には「NONE」を指定する
  - 空白文字を考慮せずに、空白文字を含めてそのままトークンを生成する場合には「ALL」を指定する
  - デフォルトは「NONE」

JAPANESE_VGRAM_LEXER
属性

- mixed_case_ASCII7
  ※ Oracle Database 10g Release 2（10.2）以降で利用可能
  - BASIC_LEXER の mixed_case 属性と同等の機能
  - テキスト索引に対して大・小文字の区別をするか否かを指定する
  - デフォルトは「NO」（トークンはすべて大文字に変換され、大・小文字の区別をしない）
  - 「YES」に設定する（＝大・小文字の区別を有効にする）と、その索引に対する問合せでは常に大文字と小文字が区別される（区別しない検索は不可能になる）

JAPANESE_LEXER
（Oracle9i Database Release 1（9.0.1）以降）

- 日本語レクサーの1つ
- レキシコン（lexicon）と呼ばれる単語辞書に基づいてトークンを生成する。レキシコンに登録された単語に一致する部分については、単語毎にトークンを生成する。レキシコンに一致しない部分については、JAPANESE_VGRAM_LEXER と同じ方法でトークンを生成する
- 次のデータベースキャラクタセットで利用可能
  - AL32UTF8、UTF8
  - JA16SJIS、JA16SJISTILDE、JA16SJISYEN
  - JA16EUC、JA16EUCTILDE、JA16EUCYEN
- レキシコンは、デフォルトの状態で、約15万語（148,129語）のエントリーを含む（Oracle Database 11.2.0.2 の場合）
- レキシコンは、カスタマイズ可能　※Oracle Database 10g Release 1（10.1）以降
  - レキシコン情報の参照・登録・更新・削除には、レキシコン・コンパイラ（ctx|cコマンド）を用いる（詳細はp.136～）

JAPANESE_LEXER
属性

- **delimiter**
  ※ Oracle Database 10g Release 1（10.1）以降で利用可能
- **mixed_case_ASCII7**
  ※ Oracle Database 10g Release 2（10.2）以降で利用可能

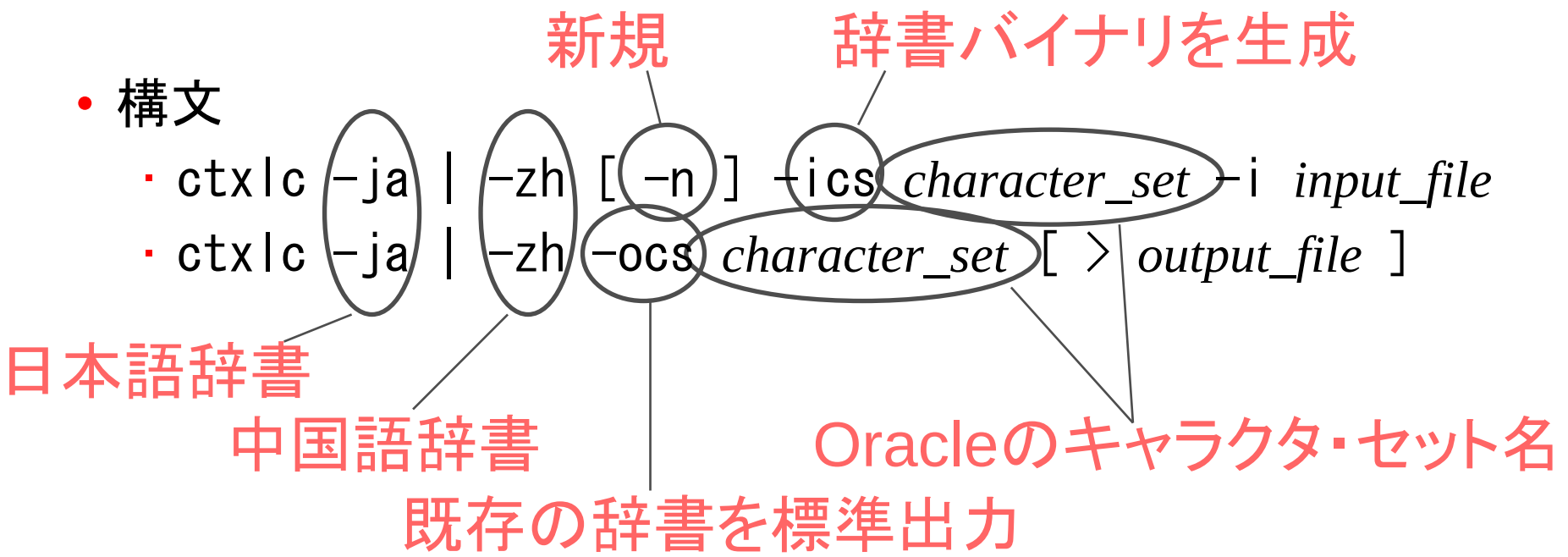
利用方法は、JAPANESE_VGRAM_LEXER の同名の属性と同じ

JAPANESE_LEXER

ctxlc ～ レキシコン・コンパイラ

※ ctxlcコマンドはOracle Database 10g Release 1（10.1）以降で利用可能。

- JAPANESE_LEXER が内部的に参照する日本語辞書（バイナリ・ファイル）の参照・編集に使用されるコマンド
- 構文
  - ctxlc -ja | -zh [-n] -ics *character_set* -i *input_file*
  - ctxlc -ja | -zh -ocs *character_set* [> *output_file*]



JAPANESE_LEXER

ctxlc 使用例

※ ctxlcコマンドはOracle Database 10g Release 1(10.1)以降で利用可能。

- 既存辞書内容を UTF-8 キャラクタ・セットでファイル jadict.txt.ORG に出力する

```
$ ctxlc -ja -ocs UTF8 > jadict.txt.ORG
```

- UTF-8 キャラクタ・セットで書かれたトークン・リスト・ファイル jadict.txt から辞書バイナリを生成する(既存辞書への追加)

```
$ ctxlc -ja -ics UTF8 -i jadict.txt
```

JAPANESE_LEXER

※ ctxlcコマンドはOracle Database 10g Release 1（10.1）以降で利用可能。

レキシコンの作成から検索まで（1）

1. 辞書ファイルの作成

トークン・リスト・ファイルをテキスト形式で作成する。トークン・リスト・ファイルには、1行に1トークンを記入する。

jadict.txt（トークン・リスト・ファイルのサンプル）

オラクルワールド

※ ctxlcコマンドはOracle Database 10g Release 1（10.1）以降で利用可能。

JAPANESE_LEXER レキシコンの作成から検索まで（2）

2. 辞書バイナリの作成

ctxlc コマンドを使用して、辞書バイナリを作成する。

```
$ ctxlc -ja -ics UTF8 -i jadict.txt
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
DRG-52118: Writing index file for terms
DRG-52117: Writing index file for IDs
DRG-52116: Done writing all terms
DRG-52115: Writing new terms in lexicon to files
DRG-52114: Writing lexicon to files
```

JAPANESE_LEXER

※ ctxlcコマンドはOracle Database 10g Release 1(10.1)以降で利用可能。

レキシコンの作成から検索まで(3)

3. 既存辞書ファイルのバックアップ

$ORACLE_HOME/ctx/data/jalx ディレクトリ以下にある4つのファイル droldJA.dat、droliJA.dat、drolkJA.dat、drolsJA.dat のバックアップを取得する。

```
$ cd $ORACLE_HOME/ctx/data/jalx
$ cp -p droldJA.dat droldJA.dat.ORG
$ cp -p droliJA.dat droliJA.dat.ORG
$ cp -p drolkJA.dat drolkJA.dat.ORG
$ cp -p drolsJA.dat drolsJA.dat.ORG
```

※ ctxlcコマンドはOracle Database 10g Release 1(10.1)以降で利用可能。

JAPANESE_LEXER
レキシコンの作成から検索まで(4)

4. 既存辞書ファイルの上書き

ファイル名 drold.dat、droli.dat、drolk.dat、drols.dat を、それぞれ droldJA.dat、droliJA.dat、drolkJA.dat、drolsJA.dat に変更して $ORACLE_HOME/ctx/data/jalx 以下にコピーする。

```
$ cp drold.dat $ORACLE_HOME/ctx/data/jalx/droldJA.dat
$ cp droli.dat $ORACLE_HOME/ctx/data/jalx/droliJA.dat
$ cp drolk.dat $ORACLE_HOME/ctx/data/jalx/drolkJA.dat
$ cp drolt.dat $ORACLE_HOME/ctx/data/jalx/droltJA.dat
```

※ ctxlcコマンドはOracle Database 10g Release 1(10.1)以降で利用可能。

JAPANESE_LEXER
レキシコンの作成から検索まで(5)

5. 索引作成

JAPANESE_LEXER を使用してCONTEXT索引を作成する。

```
CREATE TABLE testtab (text VARCHAR2(4000));
INSERT INTO testtab VALUES ('オラクルワールド');
COMMIT;
BEGIN
  CTX_DDL.CREATE_PREFERENCE('my_lexer', 'JAPANESE_LEXER');
END;
/
CREATE INDEX testidx ON testtab(text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('LEXER my_lexer');
```

※ ctxlcコマンドはOracle Database 10g Release 1（10.1）以降で利用可能。

JAPANESE_LEXER
レキシコンの作成から検索まで(6)

6. 検索実行

CONTAINS演算子を使用した全文検索を実行する。

```
SQL> SELECT token_text FROM dr$testidx$i;

TOKEN_TEXT
--------------------------------
オラクルワールド
```

JAPANESE_LEXER
注意事項

※ ctxlcコマンドはOracle Database 10g Release 1（10.1）以降で利用可能。

- レキシコンの変更は、CREATE INDEX 後は行うことができない
  - 索引作成時のレクサー動作と検索時のレクサー動作が同一である必要があるため
- レキシコンは、1つの ORACLE_HOME 上に1つだけ定義可能
  - 日本語、中国語で各1つずつ
- レキシコンには、100万語まで登録可能
  - デフォルトのレキシコンには約15万語（148,129語）が登録されている

MULTI_LEXER

- 多言語レクサーの1つ
- 言語とレクサー(SUB_LEXER)の対応付けを管理者が行う
- 各レコード(行)と言語の対応付けを明示的に行う(言語列が必要)
  - 1つのレコードに複数の言語が含まれていている場合、言語列で指定された言語に相当する部分のみが索引に格納される
- 日本語を扱える
  - 日本語用レクサーとしては、以下を指定可能
    - JAPANESE_VGRAM_LEXER(WORLD_LEXER)
      ※ 日本語文書に対する SUB_LEXER として、JAPANESE_VGRAM_LEXER を指定しても、WORLD_LEXER を指定しても、動作としては同じ
    - JAPANESE_LEXER(DB 9.0.1～)
    - AUTO_LEXER(DB 11.1～)
    - USER_LEXER(DB 9.2～)

WORLD_LEXER
（Oracle Database 10g Release 1（10.1）以降）

- 多言語レクサーの1つ
- テキスト文字列を、そのコードポイントにより言語を自動判別してトークン分割を行う
  - 1つのレコードに複数の言語が含まれていても、すべての言語を正しく索引付け可能
  - MULTI_LEXER とは異なり、言語列を必要とせず、また、SUB_LEXER の設定も必要としない
  - Unicode 5.0 標準で定義されるほとんどの言語で動作（次スライドに対応言語一覧）
- 日本語を扱える
  - 日本語部分は JAPANESE_VGRAM_LEXER と同一の方法でトークンが生成される

WORLD_LEXER の対応言語一覧

言語グループ	含まれる言語
アラビア語	アラビア語、ファルシ語、クルド語、パシュトー語、シンド語、ウルドゥー語
アルメニア語	アルメニア語
ベンガル語	アッサム語、ベンガル語
Bopomofo	客家（ハッカ）語、ビンナン語
キリル語	ベラルーシ語、ブルガリア語、マケドニア語、モルダビア語、ロシア語、セルビア語、セルビア・クロアチア語、ウクライナ語を含む50以上の言語
デーヴァナーガリー文字	ボジュプリー語、ビハール語、ヒンディー語、カシミール語、マラーティー語、ネパール語、パーリ語、サンスクリット語
エチオピア語	アムハラ語、ゲーズ語、ティグリニャ語、ティグレ語
グルジア語	グルジア語
ギリシャ語	ギリシャ語
グジャラート語	グジャラート語、カッチ語
グルムキー語	パンジャブ語
ヘブライ語	ヘブライ語、ラディノ語、イディッシュ語
カガンガ文字	レジャン語
カンナダ語	カナラ語、カンナダ語
韓国語	韓国語、ハンジャ・ハングル語
ラテン語	アフリカーンス語、アルバニア語、バスク語、ブルトン語、カタロニア語、クロアチア語、チェコ語、デンマーク語、オランダ語、英語、エスペラント 語、エストニア語、フェロー語、フィジー語、フィンランド語、フラマン語、フランス語、フリジア語、ドイツ語、ハワイ語、ハンガリー語、アイスランド語、 インドネシア語、アイルランド語、イタリア語、ラップ語、古典ラテン語、ラトビア語、リトアニア語、マレー語、マルタ語、中国標準語（ピンイン表記）、マ オリ語、ノルウェー語、ポーランド語、ポルトガル語、プロヴァンス語、ルーマニア語、サモア語、ゲール語（スコットランド）、スロバキア語、スロベニア 語、ソルビア語、スペイン語、スワヒリ語、スウェーデン語、タガログ語、トルコ語、ベトナム語、ウェールズ語
マラヤーラム語	マラヤーラム語
モンゴル語	モンゴル語
オリヤー語	オリヤー語
シンハラ語	パーリ語、シンハラ語
シリア語	アラム語、シリア語
タミル語	タミル語
テルグ語	テルグ語
ターナ文字	ディベヒ語、モルディブ語
中国語	広東語、中国標準語、ピンイン表音文字
日本語	日本語（ひらがな、漢字、カタカナ）
クメール語	カンボジア語、クメール語
ラオ語	ラオ語
ミャンマー語	ビルマ語
タイ語	タイ語
チベット語	ゾンカ語、チベット語

http://otndnld.oracle.co.jp/document/products/oracle11g/111/doc_dvd/text.111/E05789-02/amultlng.htm#CEGBAFHC

AUTO_LEXER
（Oracle Database 11g Release 1（11.1）以降）

- 多言語レクサーの1つ
- テキスト文字列を、形態素解析に基づいてトークンに分割する
  - MULTI_LEXER とは異なり、言語列を必要とせず、また、SUB_LEXER の設定も必要としない
- 日本語を扱える
  - 日本語部分に関しても、形態素解析エンジンによってトークン分割が行われる

AUTO_LEXER の対応言語一覧

アラビア語
カタロニア語
中国語（簡体字）
クロアチア語
チェコ語
デンマーク語
オランダ語
英語
フィンランド語
フランス語
ドイツ語
ギリシャ語
ヘブライ語
ハンガリー語
イタリア語
ノルウェー語（ブークモール）
日本語
韓国語
中国語（繁体字）
ポーランド語
ポルトガル語
ルーマニア語
ロシア語
セルビア語
スロバキア語
スロベニア語
スペイン語
スウェーデン語
タイ語
トルコ語
ノルウェー語（ニーノシュク）
ペルシア語

http://otndnld.oracle.co.jp/document/products/oracle11g/111/doc_dvd/text.111/E05789-02/cdatadic.htm#BHCDCAIB

USER_LEXER
（Oracle9i Database Release 2（9.2）以降）

- ユーザー定義レクサー
  - レクサーのアルゴリズムをPL/SQLで記述する
- 日本語を含む全ての言語を扱える
- 2種類のプロシージャを同時に定義する必要がある
  - 索引付けプロシージャ（索引作成用）
    PROCEDURE（IN CLOB, IN OUT CLOB, IN BOOLEAN）
    または、
    PROCEDURE（IN VARCHAR2, IN OUT VARCHAR2, IN BOOLEAN）
  - 問合せプロシージャ（検索用）
    PROCEDURE（IN VARCHAR2, IN CTX_ULEXER.WILDCARD_TAB, IN OUT VARCHAR2）

※ 索引作成時と、検索時では、トークン分割の仕組みが異なるため、2種類のプロシージャを同時に定義する（詳細は次ページ）

※ どちらのプロシージャも、トークン分割の結果をXML文書として出力する

USER_LEXER
索引付けプロシージャと問合せプロシージャの違い

- 索引付けプロシージャと問合せプロシージャでは、トークンの分割方法が異なる
- 例えば USER_LEXER で 2-Gram レクサーを実装する場合・・・

- 検索対象ドキュメントに「**オラクル**」という文字列が単独で含まれる場合
  → 索引付けプロシージャは「**オラ**」「**ラク**」「**クル**」「**ル**」のようにトークン分割
- 検索キーワードとして「**オラクル**」が指定された場合
  → 問合せプロシージャは「**オラ**」「**ラク**」「**クル**」のようにトークン分割

- 検索対象ドキュメントに「**オ**」という文字が単独で含まれる場合
  → 索引付けプロシージャはそのまま「**オ**」というトークンを生成
- 検索キーワードとして「**オ**」が指定された場合
  → 問合せプロシージャは「**オ%**」というワイルドカード検索に変換
  ※ このようにしないと、「オ」で始まる2文字のトークン（「オラ」など）を見つけることができない

USER_LEXER
索引付けプロシージャと問合せプロシージャの違い

- 索引付けプロシージャの出力例

「オラクル」のトークン分割結果

```
<tokens>
  <word>オラ</word>
  <word>ラク</word>
  <word>クル</word>
  <word>ル</word>
</tokens>
```

「オ」のトークン分割結果

```
<tokens>
  <word>オ</word>
</tokens>
```

- 問合せプロシージャの出力例

「オラクル」のトークン分割結果

```
<tokens>
  <word>オラ</word>
  <word>ラク</word>
  <word>クル</word>
</tokens>
```

「オ」のトークン分割結果

```
<tokens>
  <word wildcard="1">オ%</word>
</tokens>
```

USER_LEXER
利用時の注意点

- ユーザー定義レクサーでは、トークン情報をXML形式で記述する
- このため、「<」「>」「&」「″」「’」の5文字は、実体参照を利用して以下のように記述する必要がある
  - < → <
  - > → >
  - & → &
  - ″ → "
  - ’ → '

ワードリスト型

- ファジー検索や、(英単語の)ワイルドカード検索などをさらに高速にする場合に指定する
  - 日本語でも利用可能
- ワードリスト型は、BASIC_WORDLIST の1種類のみ

BASIC_WORDLIST の属性一覧

- **STEMMER**
- **FUZZY_MATCH**
- FUZZY_SCORE
- FUZZY_NUMRESULTS
- **SUBSTRING_INDEX**
- **PREFIX_INDEX**
- PREFIX_MIN_LENGTH
- PREFIX_MAX_LENGTH
- **WILDCARD_MAXTERMS**
- NDATA_BASE_LETTER
- NDATA_ALTERNATE_SPELLING
- NDATA_THESAURUS
- NDATA_JOIN_PARTICLES

BASIC_WORDLIST
STEMMER属性

- STEM演算子「$」による語幹検索を行う場合に指定する
- 語幹検索とは、名詞の単複、動詞の活用などを同一視した検索を指す
- 日本語でも使用可
  - ただし、JAPANESE_LEXER、AUTO_LEXER 利用時のみ使用可
  - JAPANESE_VGRAM_LEXER、WORLD_LEXER 利用時には使用不可
- 指定可能な属性値:
  - NULL(語幹検索を行わない)
  - AUTO(セッション言語を、語幹検索を行う言語として採用する。ただし、日本語の自動検出は行えない)
  - ENGLISH(英語)、DERIVATIONAL(派生語(英語))※
  - DUTCH(オランダ語)、FRENCH(フランス語)、GERMAN(ドイツ語)、ITALIAN(イタリア語)、SPANISH(スペイン語)
  - JAPANESE(日本語)

※派生語を含めた語幹検索を可能にする(英語)。ここで言う派生語とは、「形容詞happy→名詞happiness」、「形容詞modern→動詞modernize」、「動詞write→動詞rewrite」等の操作を指す。

BASIC_WORDLIST

STEMMER属性(使用例)

```
-- 表作成
CREATE TABLE testtab (text VARCHAR2(4000));

-- 1件のデータをINSERT
INSERT INTO testtab VALUES ('喜び');
COMMIT;

-- プリファレンス作成（レクサーとワードリスト）
BEGIN
  CTX_DDL.CREATE_PREFERENCE('my_lexer', 'JAPANESE_LEXER');
  CTX_DDL.CREATE_PREFERENCE('my_wordlist', 'BASIC_WORDLIST');
  CTX_DDL.SET_ATTRIBUTE('my_wordlist', 'STEMMER', 'JAPANESE');
END;
/

-- 索引作成
CREATE INDEX testidx ON testtab(text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('LEXER my_lexer WORDLIST my_wordlist');
```

BASIC_WORDLIST

STEMMER属性(使用例)

```
SQL> -- 検索実行
SQL> SELECT text FROM testtab
  2    WHERE CONTAINS (text, '$喜ぶ') > 0;

TEXT
--------------------------------------------------------------------------------
喜び
```

「喜ぶ」という検索条件で、
「喜び」がヒットしている

BASIC_WORDLIST
STEMMER属性

※ この動作状況は、
CTX_QUERY.EXPLAIN
プロシージャで確認できる
（次ページにサンプル）

- STEMMER属性で「JAPANESE」を指定した場合の動作
  - 検索文字列「**$喜ぶ**」は、内部的に、

```
(喜ぶ) EQUIV (喜ば) EQUIV (喜び) EQUIV (喜べ) EQUIV
(喜べば) EQUIV (喜ぼう) EQUIV (喜んだ) EQUIV (喜んで)
```

  のように書き換えられ、検索が実行される※
  - 検索文字列「**$美しい**」は、内部的に、

```
(美しい) EQUIV (美しかった) EQUIV (美しかろう) EQUIV (
美しく) EQUIV (美しけれ) EQUIV (美しさ)
```

  のように書き換えられ、検索が実行される※
  - 検索文字列「**$walk**」は、内部的に、

```
(WALK) EQUIV (WALKED) EQUIV (WALKING) EQUIV (WALKS)
```

  のように書き換えられ、検索が実行される※

BASIC_WORDLIST
STEMMER属性

```
CREATE TABLE testtab (text VARCHAR2(10));
BEGIN
  CTX_DDL.CREATE_PREFERENCE('my_lexer', 'JAPANESE_LEXER');
END;
/
BEGIN
  CTX_DDL.CREATE_PREFERENCE('my_wordlist', 'BASIC_WORDLIST');
  CTX_DDL.SET_ATTRIBUTE('my_wordlist', 'STEMMER', 'JAPANESE');
END;
/
CREATE INDEX testidx ON testtab(text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('
    LEXER my_lexer
    WORDLIST my_wordlist
    STOPLIST ctxsys.empty_stoplist
  ');

CREATE TABLE exptab (
  explain_id   VARCHAR2(30),
  id           NUMBER,
  parent_id    NUMBER,
  operation    VARCHAR2(30),
  options      VARCHAR2(30),
  object_name  VARCHAR2(80),
  position     NUMBER,
  cardinality  NUMBER
);
```

JAPANESE_LEXERを指定

STEMMER属性にJAPANESE（日本語）を指定

CTX_QUERY.EXPLAINプロシージャの実行結果は、explain table と呼ばれる結果表に格納される。ここでは、その結果表を作成。

CTX_QUERY.EXPLAIN プロシージャの実行

```
BEGIN
  CTX_QUERY.EXPLAIN(
    index_name    => 'TESTIDX',
    text_query    => '$walk',
    explain_table => 'exptab',
    sharelevel    => 1,
    explain_id    => 'JL$walk',
    part_name     => NULL
  );
  CTX_QUERY.EXPLAIN(
    index_name    => 'TESTIDX',
    text_query    => '$(喜ぶ)',
    explain_table => 'exptab',
    sharelevel    => 1,
    explain_id    => 'JL$(喜ぶ)',
    part_name     => NULL
  );
  CTX_QUERY.EXPLAIN(
    index_name    => 'TESTIDX',
    text_query    => '$(美しい)',
    explain_table => 'exptab',
    sharelevel    => 1,
    explain_id    => 'JL$(美しい)',
    part_name     => NULL
  );
END;
/

COLUMN EXPLAIN_ID FORMAT A20
COLUMN OPERATION FORMAT A20
COLUMN OPTIONS FORMAT A10
COLUMN OBJECT_NAME FORMAT A20
SELECT * FROM exptab ORDER BY explain_id, id;
```

結果表の参照

BASIC_WORDLIST
STEMMER属性

前ページ最後のSELECT文の実行結果

```
SQL> SELECT * FROM exptab ORDER BY explain_id, id;
```

EXPLAIN_ID	ID	PARENT_ID	OPERATION	OPTIONS	OBJECT_NAME	POSITION	CARDINALITY
JL$(喜ぶ)	1	0	EQUIVALENCE	($)	喜ぶ	1	
JL$(喜ぶ)	2	1	WORD		喜ば	1	
JL$(喜ぶ)	3	1	WORD		喜び	2	
JL$(喜ぶ)	4	1	WORD		喜ぶ	3	
JL$(喜ぶ)	5	1	WORD		喜べ	4	
JL$(喜ぶ)	6	1	WORD		喜べば	5	
JL$(喜ぶ)	7	1	WORD		喜ぼう	6	
JL$(喜ぶ)	8	1	WORD		喜んだ	7	
JL$(喜ぶ)	9	1	WORD		喜んで	8	
JL$(美しい)	1	0	EQUIVALENCE	($)	美しい	1	
JL$(美しい)	2	1	WORD		美しい	1	
JL$(美しい)	3	1	WORD		美しかった	2	
JL$(美しい)	4	1	WORD		美しかろう	3	
JL$(美しい)	5	1	WORD		美しく	4	
JL$(美しい)	6	1	WORD		美しけれ	5	
JL$(美しい)	7	1	WORD		美しさ	6	
JL$(美しい)	8	1	WORD		臂	7	
JL$walk	1	0	EQUIVALENCE	($)	WALK	1	
JL$walk	2	1	WORD		WALK	1	
JL$walk	3	1	WORD		WALKED	2	
JL$walk	4	1	WORD		WALKING	3	
JL$walk	5	1	WORD		WALKS	4	

「$(喜ぶ)」の展開結果

「$(美しい)」の展開結果

「$walk」の展開結果

BASIC_WORDLIST
STEMMER属性（参考）

- STEMMER属性で「ENGLISH」を指定した場合の動作
  - 検索文字列「$think」は、内部的に次のように書き換えられ、検索が実行される

```
(THINK) EQUIV (THINKING) EQUIV (THINKS) EQUIV (THOUGHT)
```

- STEMMER属性で「DERIVATIONAL」を指定した場合の動作
  - 検索文字列「$think」は、内部的に次のように書き換えられ、検索が実行される

```
(NONTHINKER) EQUIV (NONTHINKERS) EQUIV (NONTHINKING) EQUIV (NONTHINKINGLY) EQUIV
(NONTHINKINGNESS) EQUIV (NONTHINKINGS) EQUIV (NONTHOUGHT) EQUIV (NONTHOUGHTLY) EQUIV
(NONTHOUGHTNESS) EQUIV (OUTTHINK) EQUIV (OUTTHINKABILITIES) EQUIV (OUTTHINKABILITY) EQUIV
(OUTTHINKABLE) EQUIV (OUTTHINKABLENESS) EQUIV (OUTTHINKABLENESSES) EQUIV (OUTTHINKABLY)
EQUIV (OUTTHINKER) EQUIV (OUTTHINKERS) EQUIV (OUTTHINKING) EQUIV (OUTTHINKINGLY) EQUIV
(OUTTHINKINGNESS) EQUIV (OUTTHINKINGS) EQUIV (OUTTHINKS) EQUIV (OUTTHOUGHT) EQUIV
(OUTTHOUGHTLY) EQUIV (OUTTHOUGHTNESS) EQUIV (RETHINK) EQUIV (RETHINKER) EQUIV (RETHINKERS)
EQUIV (RETHINKING) EQUIV (RETHINKINGLY) EQUIV (RETHINKINGNESS) EQUIV (RETHINKINGS) EQUIV
(RETHINKS) EQUIV (RETHOUGHT) EQUIV (RETHOUGHTLY) EQUIV (RETHOUGHTNESS) EQUIV (THINK) EQUIV
(THINKABILITIES) EQUIV (THINKABILITY) EQUIV (THINKABLE) EQUIV (THINKABLENESS) EQUIV
(THINKABLENESSES) EQUIV (THINKABLY) EQUIV (THINKER) EQUIV (THINKERS) EQUIV (THINKING) EQUIV
(THINKINGLY) EQUIV (THINKINGNESS) EQUIV (THINKINGS) EQUIV (THINKS) EQUIV (THOUGHT) EQUIV
(THOUGHTLY) EQUIV (THOUGHTNESS) EQUIV (UNTHINKER) EQUIV (UNTHINKERS) EQUIV (UNTHINKING)
EQUIV (UNTHINKINGLY) EQUIV (UNTHINKINGNESS) EQUIV (UNTHINKINGS) EQUIV (UNTHOUGHT) EQUIV
(UNTHOUGHTLY) EQUIV (UNTHOUGHTNESS)
```

BASIC_WORDLIST
FUZZY_MATCH属性

- ファジー検索（FUZZY演算子を利用）を有効化する場合に指定する
- 日本語でも使用可（以下の表記ゆれの同一視のみ）
  - バ行とヴァ行の置換（バイオリン／ヴァイオリン）
  - {チ,テ,ツィ,ティ}の相互置換（チター／ツィター）
  - {ツ,トゥ}の相互置換（ツール／トゥール）
  - {ヂョ,ジョ}の相互置換（アダーヂョ／アダージョ）
  - {ヂ,デ,ジ,ヂィ,ディ}の相互置換（ディーゼル／ディーゼル）
  - {ず,づ}の相互置換（まづ／まず）
  - {じ,ぢ}の相互置換（いいじゃないか／いいぢゃないか）
  - "ッ","ヵ","ヶ"のありなし（漢字）（四ッ谷／四谷）
- 指定可能な属性値：
  - AUTO（セッション言語から自動的に言語を導出）
  - GENERIC（デフォルト）
  - ENGLISH、DUTCH、FRENCH、GERMAN、ITALIAN、SPANISH
  - JAPANESE_VGRAM
  - CHINESE_VGRAM（ダミー）、KOREAN（ダミー）
  - OCR (Optical Character Recognition)

※ 中国語と韓国語については、ファジー検索の機能は提供されない。左記の属性値を指定しても、実際にはファジー検索は行われない。

BASIC_WORDLIST

FUZZY_MATCH属性(使用例)

```
-- 表作成
CREATE TABLE testtab (text VARCHAR2(4000));

-- 1件のデータをINSERT
INSERT INTO testtab VALUES ('ヴァイオリン');
COMMIT;

-- プリファレンス作成（レクサーとワードリスト）
BEGIN
  CTX_DDL.CREATE_PREFERENCE('my_lexer', 'JAPANESE_VGRAM_LEXER');
  CTX_DDL.CREATE_PREFERENCE('my_wordlist', 'BASIC_WORDLIST');
  CTX_DDL.SET_ATTRIBUTE('my_wordlist', 'FUZZY_MATCH', 'JAPANESE_VGRAM');
END;
/

-- 索引作成
CREATE INDEX testidx ON testtab(text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('LEXER my_lexer WORDLIST my_wordlist');
```

BASIC_WORDLIST

FUZZY_MATCH属性(使用例)

```
SQL> -- 検索実行
SQL> SELECT text FROM testtab
  2    WHERE CONTAINS (text, 'FUZZY(バイオリン,,,N)') > 0;

TEXT
--------------------------------
ヴァイオリン
```

「バイオリン」という検索条件で、
「ヴァイオリン」がヒットしている

BASIC_WORDLIST
SUBSTRING_INDEX属性

- サブストリング索引を作成するか否かを指定
- サブストリング索引を作成すると、「%ing」や「%benz%」のような、左側切捨ておよび左右切捨てのワイルドカード問合せを高速に実行できる
- 日本語と組み合わせて使用することも可能
  - ただし、ワイルド・カード問合せが有効となるのは英数字部分のみ
- 指定可能な属性値:
  - TRUE ... サブストリング索引が作成される
  - FALSE(デフォルト) ... サブストリング索引が作成されない

BASIC_WORDLIST

SUBSTRING_INDEX属性(使用例)

```
-- 表作成
CREATE TABLE testtab (text VARCHAR2(4000));

-- 1件のデータをINSERT
INSERT INTO testtab VALUES ('オラクルOracle');
COMMIT;

-- プリファレンス作成（レクサーとワードリスト）
BEGIN
  CTX_DDL.CREATE_PREFERENCE('my_lexer', 'JAPANESE_VGRAM_LEXER');
  CTX_DDL.CREATE_PREFERENCE('my_wordlist', 'BASIC_WORDLIST');
  CTX_DDL.SET_ATTRIBUTE('my_wordlist', 'SUBSTRING_INDEX', 'TRUE');
END;
/

-- 索引作成
CREATE INDEX testidx ON testtab(text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('LEXER my_lexer WORDLIST my_wordlist');
```

BASIC_WORDLIST

SUBSTRING_INDEX属性（使用例）

```
SQL> -- 検索実行
SQL> SELECT text FROM testtab
  2    WHERE CONTAINS (text, '%racle') > 0;

TEXT
--------------------------------------------------------------------------------
オラクルOracle
```

「%racle」という検索条件を高速に評価

※ サブストリング索引を作成しなくても、上記の検索自体は可能（ただし遅い）
※ ワイルドカード演算子（「%」および「_」）の使用方法については前述のとおり。
※ 日本語文字については、ワイルドカード演算子を付けなくてもサブストリング検索となる（たとえば、「ラクル」という検索条件で、「オラクル」がヒットする）ため、SUBSTRING_INDEX 属性の設定状況に関係なく、高速に検索結果が返る。

※ Oracle8i Database Release 8.1.7以降で利用可能

BASIC_WORDLIST
PREFIX_INDEX属性

- プリフィックス索引を作成するか否かを指定
- プリフィックス索引を作成すると、「T0%」のような右側切捨てのワイルドカード問合せを高速に実行できる
- 日本語と組み合わせて使用することも可能
  - ただし、ワイルド・カード文字を付けた問合せが有効となるのは英数字部分のみ。日本語のトークンについては、内部的に一文字検索が発生する場合（ワイルドカード文字を付加せず）に、プリフィックス索引が利用される（日本語のトークンについてもプリフィックス索引が作成される）
- 指定可能な属性値：
  - TRUE ... プリフィックス索引が作成される
  - FALSE（デフォルト） ... プリフィックス索引が作成されない

※ Oracle8i Database Release 8.1.7以降で利用可能

BASIC_WORDLIST

PREFIX_INDEX属性(使用例)

```
-- 表作成
CREATE TABLE testtab (text VARCHAR2(4000));

-- 1件のデータをINSERT
INSERT INTO testtab VALUES ('オラクルOracle');
COMMIT;

-- プリファレンス作成（レクサーとワードリスト）
BEGIN
  CTX_DDL.CREATE_PREFERENCE('my_lexer', 'JAPANESE_VGRAM_LEXER');
  CTX_DDL.CREATE_PREFERENCE('my_wordlist', 'BASIC_WORDLIST');
  CTX_DDL.SET_ATTRIBUTE('my_wordlist', 'PREFIX_INDEX', 'TRUE');
END;
/

-- 索引作成
CREATE INDEX testidx ON testtab(text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('LEXER my_lexer WORDLIST my_wordlist');
```

※ Oracle8i Database Release 8.1.7以降で利用可能

BASIC_WORDLIST

PREFIX_INDEX属性（使用例）

```
SQL> -- 検索実行
SQL> SELECT text FROM testtab
  2    WHERE CONTAINS (text, 'ora%') > 0;

TEXT
--------------------------------------------------------------------------------
オラクルOracle
```

「ora%」という検索条件を高速に評価

※ プリフィックス索引を作成しなくても、上記の検索自体は可能（プリフィックス索引を作成しなくても、多くの場合、ある程度は高速）
※ ワイルドカード演算子（「%」および「_」）の使用方法については前述のとおり。
※ 日本語文字については、プリフィックス索引を作成することで、例えば「子」、「山」、「オ」などの一文字検索が、高速になる場合がある。（特に、CTX_QUERY.COUNT_HITS 関数によるヒット件数の全件カウント処理で効果が期待できる）

※ Oracle9i Database Release 1（9.0.1）以降で利用可能

BASIC_WORDLIST

WILDCARD_MAXTERMS属性

- ワイルドカード演算子による拡張語句の最大数を指定する（1以上50,000以下の整数値）
  - ワイルドカード検索は、既存のトークン情報を利用して拡張される
  - たとえば既存のトークン情報が「AAA」「AAB」「AAC」の場合、ワイルドカード検索「AA%」は、内部的には、「(AAA) EQUIV (AAB) EQUIV (AAC)」のように拡張される
- 拡張語句数がこの属性で指定された値を超えるとエラーが戻される
- デフォルトは20,000

※ Oracle9i Database Release 1(9.0.1)
以降で利用可能

BASIC_WORDLIST

WILDCARD_MAXTERMS属性(使用例)

```
-- 表作成
CREATE TABLE testtab (text VARCHAR2(4000));

-- 20001件のダミーデータをINSERT
BEGIN
  FOR i IN 1..20001 LOOP
    INSERT INTO testtab VALUES ('aa'||TRIM(TO_CHAR(i,'09999')));
  END LOOP;
END;
/
COMMIT;

-- プリファレンス作成 (レクサー)
BEGIN
  CTX_DDL.CREATE_PREFERENCE('my_lexer', 'JAPANESE_VGRAM_LEXER');
END;
/
```

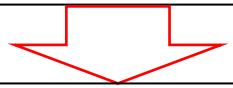
※ Oracle9i Database Release 1(9.0.1)
以降で利用可能

BASIC_WORDLIST

WILDCARD_MAXTERMS属性(使用例)

- WILDCARD_MAXTERMS属性を指定せずに索引を作成し、拡張語数が2万を超えるワイルドカード問合せを実行すると・・・

```
CREATE INDEX testidx ON testtab(text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('LEXER my_lexer');
```



```
SQL> SELECT text FROM testtab WHERE CONTAINS (text,'aa%') > 0;
SELECT text FROM testtab WHERE CONTAINS (text,'aa%') > 0
*
行1でエラーが発生しました。:
ORA-29902: ODCIIndexStart()ルーチンの実行中にエラーが発生しました。
ORA-20000: Oracle Text error:
DRG-51030: wildcard query expansion resulted in too many terms
```

※ Oracle9i Database Release 1(9.0.1)
以降で利用可能

BASIC_WORDLIST

WILDCARD_MAXTERMS属性(使用例)

- 次に、WILDCARD_MAXTERMS属性を50000に設定し、索引のメタ情報を変更した上で、拡張語数が2万を超えるワイルドカード問合せを実行すると・・・

```
-- プリファレンス作成（ワードリスト）
BEGIN
  CTX_DDL.CREATE_PREFERENCE('my_wordlist', 'BASIC_WORDLIST');
  CTX_DDL.SET_ATTRIBUTE('my_wordlist', 'WILDCARD_MAXTERMS', '50000');
END;
/

-- 索引のメタ情報を変更　※この場合、索引の作り直しは必要ない。
ALTER INDEX testidx REBUILD
  PARAMETERS ('REPLACE METADATA WORDLIST my_wordlist');
```

※ Oracle9i Database Release 1(9.0.1)
以降で利用可能

BASIC_WORDLIST

WILDCARD_MAXTERMS属性(使用例)

- 今度は正常に実行される(ただし、実行にはそれなりの時間がかかる)

```
SQL> SELECT text FROM testtab WHERE CONTAINS (text,'aa%') > 0;

TEXT
--------------------------------------------------------------------------------
aa00001
aa00002
aa00003
... (省略) ...
aa19998
aa19999
aa20000
aa20001

20001行が選択されました。
```

記憶域型

- 索引の格納先となる表領域、およびその格納方法を指定する
- 記憶域型は、BASIC_STORAGEの1種類のみ
  - Oracle Text の内部表（$I、$K、$R、$N、$P、$S）、内部索引（$X）のそれぞれの格納先表領域、およびそれぞれの格納方式を指定できる

BASIC_STORAGE の属性一覧

- I_INDEX_CLAUSE（$X索引）
- I_ROWID_INDEX_CLAUSE（$R索引）※CTXCAT索引のみ
- I_TABLE_CLAUSE（$I表）※「I」は「INDEX DATA TABLE」の頭文字
- K_TABLE_CLAUSE（$K表）※「K」は「KEYMAP TABLE」の頭文字
- R_TABLE_CLAUSE（$R表）※「R」は「ROWID TABLE」の頭文字
- N_TABLE_CLAUSE（$N表）※「N」は「NEGATIVE LIST TABLE」の頭文字
- P_TABLE_CLAUSE（$P表）
- S_TABLE_CLAUSE（$S表）

BASIC_STORAGE 使用例

```
BEGIN
  CTX_DDL.CREATE_PREFERENCE('mystore', 'BASIC_STORAGE');
  CTX_DDL.SET_ATTRIBUTE('mystore', 'I_TABLE_CLAUSE', 'TABLESPACE foo STORAGE
(INITIAL 1K)');  ←I_TABLE_CLAUSEのLOBセグメントに「DISABLE STORAGE IN ROW」を指定しないことを推奨
  CTX_DDL.SET_ATTRIBUTE('mystore', 'K_TABLE_CLAUSE', 'TABLESPACE foo STORAGE
(INITIAL 1K)');
  CTX_DDL.SET_ATTRIBUTE('mystore', 'R_TABLE_CLAUSE', 'TABLESPACE users STORAGE
(INITIAL 1K) LOB (data) STORE AS (DISABLE STORAGE IN ROW CACHE)');←「CACHE」の指定を
  CTX_DDL.SET_ATTRIBUTE('mystore', 'N_TABLE_CLAUSE', 'TABLESPACE foo STORAGE  推奨
(INITIAL 1K)');
  CTX_DDL.SET_ATTRIBUTE('mystore', 'I_INDEX_CLAUSE', 'TABLESPACE foo STORAGE
(INITIAL 1K) COMPRESS 2');←「COMPRESS 2」の指定を推奨
  CTX_DDL.SET_ATTRIBUTE('mystore', 'P_TABLE_CLAUSE', 'TABLESPACE foo STORAGE
(INITIAL 1K)');
  CTX_DDL.SET_ATTRIBUTE('mystore', 'S_TABLE_CLAUSE', 'TABLESPACE foo STORAGE
(INITIAL 1K)');
END;
/
```

BASIC_STORAGE 使用上の注意事項

- I_TABLE_CLAUSE では、$I表のLOBセグメントに「DISABLE STORAGE IN ROW」の指定は非推奨(検索パフォーマンスが劣化してしまうため)
- I_INDEX_CLAUSE では、デフォルトの動作を上書きする場合も、「COMPRESS 2」の指定を含めることを推奨(必要なディスク容量の削減と検索パフォーマンスの向上のため)
- R_TABLE_CLAUSE のデフォルトは「LOB(DATA) STORE AS (CACHE)」。デフォルトの動作を上書きする場合も、CACHE句を含めることを推奨(検索パフォーマンス向上のため)

セクション・グループ型

- 次の場合に、ここでセクショナの指定を行う
  - 検索対象がXML、HTMLなどのタグ付きドキュメントの場合で、検索対象を特定のタグに限定した検索を実行する場合
  - 検索対象がプレーンテキストの場合で、文または段落に基づくセクション検索を実行する場合
  - 上記2つの組み合わせを実行する場合
- 上記のいずれにも該当しない場合は、この部分はスキップ

セクション・グループ型一覧

- **NULL_SECTION_GROUP（デフォルト）**
- **BASIC_SECTION_GROUP**
- **AUTO_SECTION_GROUP**
- **PATH_SECTION_GROUP**
- HTML_SECTION_GROUP
- XML_SECTION_GROUP
- NEWS_SECTION_GROUP

※ 各セクション・グループ型の詳細はp.214以降で解説

セクションの割り当て

- セクション・グループでのセクション割り当てには次のような方法がある
  1. フィールド・セクション（CTX_DDL.ADD_FILED_SECTION）
     - VISIBLE フィールド・セクション
     - INVISIBLE フィールド・セクション
  2. ゾーン・セクション（CTX_DDL.ADD_ZONE_SECTION）
  3. 文セクション、段落セクション（CTX_DDL.ADD_SPECIAL_SECTION）
  4. MDATAセクション（CTX_DDL.ADD_MDATA_SECTION）
     → セクション検索に**MDATA**演算子を利用
  5. SDATAセクション（CTX_DDL.ADD_SDATA_SECTION）
     → セクション検索に**SDATA**演算子を利用

セクション検索に**WITHIN**演算子を利用

※ PATH_SECTION_GROUPでは、さらに、**INPATH**演算子、**HASPATH**演算子によるXPath検索が可能。

セクション・グループ型とセクション割り当て方法の組み合わせ可否

セクション・グループ ＼ セクション割り当て方法	ZONE（ゾーン）	FIELD（フィールド）	STOP	MDATA	SDATA（タグ）	SDATA（列）	ATTRI-BUTE（属性）	SPECI-AL
NULL	×	×	×	×	×	○	×	○
BASIC	○	○	×	○	○	○	×	○
HTML	○	○	×	○	○	○	×	○
XML	○	○	×	○	○	○	○	○
NEWS	○	○	×	○	○	○	×	○
AUTO	×	×	○	×	×	○	×	×
PATH	×	×	×	×	×	○	×	×

フィールド・セクション

- タグに基づくセクションを作成する
- CTX_DDL.ADD_FIELD_SECTION プロシージャを利用して、既存のセクション・グループにフィールド・セクションを追加する
- フィールドセクションでは、タグをネストさせたり、オーバーラップさせることができない
  - ネストさせた場合には、最も内側のタグに所属する
  - オーバーラップさせた場合（＝同じタグ名で繰り返した場合）には、同じタグ内にあるものとして認識される

構文

```
CTX_DDL.ADD_FIELD_SECTION
(
  group_name    IN  VARCHAR2,
  section_name  IN  VARCHAR2,
  tag           IN  VARCHAR2,
  visible       IN  BOOLEAN DEFAULT FALSE
);
```

group_name	:	セクション・グループの名前を指定
section_name	:	問合せで使用するセクション名を指定（1グループに64まで指定可能）
tag	:	セクションの開始・終了を表すタグを指定
visible	:	このセクション以外からこのセクションを参照可能にするときはTRUE

フィールド・セクション
～ VISIBLE パラメータの影響

外部参照を false にした場合、語句「ガイド」を
検索してもそれは外から見えないためヒットしない

```
SQL> SELECT * FROM doc WHERE CONTAINS(text, 'ガイド') > 0;

レコードが選択されませんでした。

SQL> SELECT * FROM doc WHERE CONTAINS(text, 'ガイド WITHIN 題') > 0;

 NUM TEXT
---- --------------------------------------------------------------------------
   1 <題>Oracle Textガイド</題><概要>Oracleの全文検索製品ガイド</概要>
```

語句“ガイド”はセクション“題”を指定することでヒットする

ゾーン・セクション

- タグに基づくセクションを作成する
- CTX_DDL.ADD_ZONE_SECTIONプロシージャを利用して、既存のセクション・グループにゾーン・セクションを追加する
- ゾーン・セクションはタグのネストやオーバーラップが可能

構文

```
CTX_DDL.ADD_ZONE_SECTION
(
  group_name    IN  VARCHAR2,
  section_name  IN  VARCHAR2,
  tag           IN  VARCHAR2
);
```

group_name	:	セクション・グループの名前を指定
section_name	:	問合せで使用するセクション名を指定する
tag	:	セクションの開始・終了を表すタグを指定する

フィールド・セクションと ゾーン・セクションの違い

サンプル文書

```
<A>rat</A><A>ox</A><B>dragon<C>snake</C></B>
```

結果

条件	フィールド・セクション	ゾーン・セクション
(ox AND rat) WITHIN a	○	×
snake WITHIN b	×	○
snake WITHIN c	○	○

○ ヒットする
× ヒットしない

※この結果は ADD_FIELD_SECTION の VISIBLE パラメータには依存しない

フィールド・セクション使用例

- 表作成

```
CREATE TABLE testtab (id NUMBER PRIMARY KEY, text VARCHAR2(4000));
INSERT INTO testtab VALUES
  (1,'<A>rat</A><A>ox</A><B>dragon<C>snake</C></B>');
COMMIT;
```

- セクション・グループ作成

```
BEGIN
  CTX_DDL.CREATE_SECTION_GROUP('bsg','BASIC_SECTION_GROUP');
  CTX_DDL.ADD_FIELD_SECTION('bsg','A','A',TRUE);
  CTX_DDL.ADD_FIELD_SECTION('bsg','B','B',TRUE);
  CTX_DDL.ADD_FIELD_SECTION('bsg','C','C',TRUE);
END;
/
```

※ ここでは VISIBLE=TRUE を設定

フィールド・セクション使用例

- 索引作成

```
CREATE INDEX testidx ON testtab(text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('SECTION GROUP bsg');
```

- 検索実行

```
SQL> SELECT id FROM testtab WHERE CONTAINS (text, '(ox AND rat) WITHIN a') > 0;
        ID
----------
         1

SQL> SELECT id FROM testtab WHERE CONTAINS (text, 'snake WITHIN b') > 0;
no rows selected

SQL> SELECT id FROM testtab WHERE CONTAINS (text, 'snake WITHIN c') > 0;
        ID
----------
         1
```

<A>rat</A><A>ox</A><B>dragon<C>snake</C></B>

ゾーン・セクション使用例

- 表作成

```
CREATE TABLE testtab (id NUMBER PRIMARY KEY, text VARCHAR2(4000));
INSERT INTO testtab VALUES
  (1,'<A>rat</A><A>ox</A><B>dragon<C>snake</C></B>');
COMMIT;
```

- セクション・グループ作成

```
BEGIN
  CTX_DDL.CREATE_SECTION_GROUP('bsg','BASIC_SECTION_GROUP');
  CTX_DDL.ADD_ZONE_SECTION('bsg','A','A');
  CTX_DDL.ADD_ZONE_SECTION('bsg','B','B');
  CTX_DDL.ADD_ZONE_SECTION('bsg','C','C');
END;
/
```

ゾーン・セクション使用例

- 索引作成

```
CREATE INDEX testidx ON testtab(text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('SECTION GROUP bsg');
```

- 検索実行

```
SQL> SELECT id FROM testtab WHERE CONTAINS (text, '(ox AND rat) WITHIN a') > 0;
no rows selected
SQL> SELECT id FROM testtab WHERE CONTAINS (text, 'snake WITHIN b') > 0;
        ID
----------
         1
SQL> SELECT id FROM testtab WHERE CONTAINS (text, 'snake WITHIN c') > 0;
        ID
----------
         1
```

<A>rat</A><A>ox</A><B>dragon<C>snake</C></B>

文セクション、段落セクション

- 文や段落に基づくセクションを作成する
- CTX_DDL.ADD_SPECIAL_SECTIONプロシージャを利用して、既存のセクション・グループに文セクションや段落セクションを追加する
- 文および段落の認識は、レクサーにより行われる
- BASIC_LEXER では、次のルールで文と段落を認識
  - 文（以下のいずれかに一致する）
    - 単語＋文区切り記号（ピリオド、!、?）＋半角スペース
    - 単語＋文区切り記号（ピリオド、!、?）＋改行コード
  - 段落（以下のいずれかに一致する）
    - 単語＋文区切り記号（ピリオド、!、?）＋改行コード＋半角スペース
    - 単語＋文区切り記号（ピリオド、!、?）＋改行コード＋改行コード

文セクション、段落セクション使用例

- 表作成

```
CREATE TABLE testtab (id NUMBER PRIMARY KEY, text VARCHAR2(4000));

INSERT INTO testtab VALUES (1,'aaa bbb ccc. dddd eee ffffff.');

INSERT INTO testtab VALUES (2,'aaa bbb ccc.'||
CHR(13)||CHR(10)||CHR(13)||CHR(10)||'dddd eee ffffff.');

COMMIT;
```

※ CHR(13)、CHR(10) は、
いずれも改行コード。ここで
はCR+LFを2回挿入している。

CR…0x0D=CHR(13)
LF…0x0A=CHR(10)

ID	TEXT
1	aaa bbb ccc. dddd eee ffffff.
2	aaa bbb ccc.<改行><改行>dddd eee ffffff.

文セクション、段落セクション使用例

- プリファレンスおよびセクション・グループ作成

```
BEGIN
  CTX_DDL.CREATE_PREFERENCE('bl','BASIC_LEXER');
  CTX_DDL.CREATE_SECTION_GROUP('nsg','NULL_SECTION_GROUP');
  CTX_DDL.ADD_SPECIAL_SECTION('nsg','SENTENCE');
  CTX_DDL.ADD_SPECIAL_SECTION('nsg','PARAGRAPH');
END;
/
```

- 索引作成

```
CREATE INDEX testidx ON testtab(text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('LEXER bl SECTION GROUP nsg');
```

文セクション、段落セクション使用例

- 検索実行

```
SELECT id FROM testtab
  WHERE CONTAINS (text, '(aaa AND ccc) WITHIN SENTENCE') > 0;
```

↳ 実行結果:ID＝1、2 が共にヒット

```
SELECT id FROM testtab
  WHERE CONTAINS (text, '(aaa AND eee) WITHIN SENTENCE') > 0;
```

↳ 実行結果:ヒットせず

```
SELECT id FROM testtab
  WHERE CONTAINS (text, '(aaa AND eee) WITHIN PARAGRAPH') > 0;
```

↳ 実行結果:
ID＝1 のみがヒット

ID	TEXT
1	aaa bbb ccc. dddd eee ffffff.
2	aaa bbb ccc.<改行><改行>dddd eee ffffff.

MDATAセクション
（Oracle Database 10g Release 1（10.1）以降）

- タグまたは列に基づくセクションを作成する
  - CTX_DDL.ADD_MDATA_SECTIONプロシージャを利用して、既存のセクション・グループに、タグに基づくMDATAセクションを追加する
  - CTX_DDL.ADD_MDATA_COLUMNプロシージャを利用して、既存のセクション・グループに、列に基づくMDATAセクションを追加する
- MDATAでは、**文字列型の完全一致のみ**を評価できる ※
- レクサーによるトークン分割は行われない
  - 値は64バイトに丸められる
  - セクション名は、大文字・小文字が同一視される
  - セクション値は、大文字・小文字が区別される
  - 値が複数語から成る場合も、単一の値として索引に格納される
- スコア
  - SDATAの条件に一致する場合、スコアとして100が返る
  - SDATAの条件に一致しない場合、スコアとして0が返る

※ ワイルドカード文字による部分一致検索は不可能

SDATAセクション（Oracle Database 11g Release 1（11.1）以降）

※ デフォルトでは、CREATE INDEX文のFILTER BY句で指定されたすべての列がSDATAセクションに設定される。

- タグまたは列に基づくセクションを作成する
  - CTX_DDL.ADD_SDATA_SECTIONプロシージャを利用して、既存のセクション・グループに、タグに基づくSDATAセクションを追加する
  - CTX_DDL.ADD_SDATA_COLUMNプロシージャを利用して、既存のセクション・グループに、列に基づくSDATAセクションを追加する
- SDATAでは、**文字列型の完全一致と部分一致**、**数値型の完全一致と範囲一致**、**日付型の完全一致と範囲一致**を評価できる

※ 文字列型の部分一致ではワイルドカード文字を使用

- レクサーによるトークン分割は行われない
  - 文字列型の値は249バイト以下である必要がある（249バイトを超えるデータがある場合、CREATE INDEX時にエラーとなる）
  - セクション名は、大文字・小文字が同一視される
  - セクション値は、大文字・小文字が区別される
  - 値が複数語から成る場合も、単一の値として索引に格納される

SDATAセクション
（Oracle Database 11g Release 1（11.1）以降）

- スコア
  - SDATAの条件に一致する場合、スコアとして100が返る
  - SDATAの条件に一致しない場合、スコアとして0が返る
  - SDATAの対象がNULL値の場合、スコアとして0が返る

SDATAセクション
（Oracle Database 11g Release 1（11.1）以降）

- SDATA問合せは、文字列、数値、日付に対して実行する
  - 文字列および日付文字列は、一重引用符または二重引用符で囲む
    - 日付型の書式は、YYYY-MM-DD、YYYY-MM-DD HH24:MI:SS のいずれかである必要がある
  - 数値は引用符で囲まず、SQL書式のNUMBERに準拠する必要がある
- SDATA演算子の使用例
  - CONTAINS(text, 'dog and SDATA(category = ''news'')')>0 ...
  - SDATA(rating between 1.2 and 3.4) ...
  - SDATA(author LIKE "FFORDE%") ...
  - SDATA(date >= "2005-09-18") ...

SDATAセクション
（Oracle Database 11g Release 1（11.1）以降）

- SDATA演算子の構文

```
SData           :=  "SDATA" "(" SDataPredicate ")"
SDataPredicate  :=  sectionname SDataTest
SDataTest       :=  <SDataSingleOp SDataLiteral> | SDataBetweenOp | <"is" ("not")? "null">
SDataSingleOp   :=  ("<" | "<=" | "=" | ">=" | ">" | "!=" | "<>" | "like") SDataLiteral
SDataBetweenOp  :=  "between" SDataLiteral "and" SDataLiteral
SDataLiteral    :=  numeric_literal | "'" string_literal "'" | "'" date_string "'"
```

  - sectionname ... SDATAセクションの名前
  - SDataLiteral ... SDATAセクションの値

SDATAセクション
（Oracle Database 11g Release 1（11.1）以降）

- SDATAは、「SDATA以外の子もあるAND演算子」の子として使用することを推奨（検索パフォーマンスの観点から）
  - 推奨される使用例
    - `dog & SDATA(foo = 5)`
      このSDATAは、SDATA以外の子もあるAND演算子の子になっている
    - `dog & (SDATA(foo = 5) | SDATA(x = 1))`
      このSDATA演算子はORの子だが、SDATA以外の子を持つAND演算子の子でもある
  - 推奨されない使用例
    - `SDATA(foo = 5)`
      SDATAが単体で使用されている
    - `dog | SDATA(bar = 9)`
      SDATAがAND演算子ではなく、OR演算子の子になっている
    - `SDATA(foo = 5) & SDATA(bar = 7)`
      両方のSDATAがAND演算子の子だが、このAND演算子にはSDATA以外の子がない

フィールド・セクション、MDATA、SDATAの使い分け

- トークンの分割方法から見た特徴
  - フィールド・セクション → セクション値がレクサーによってトークン分割される
  - MDATA、SDATA → セクション値がトークン分割されずに、そのまま索引に格納される
- 文字列型の部分一致であれば、フィールド・セクションを利用する
- 文字列型の完全一致であれば、MDATAを使用する
- 文字列型の完全一致、部分一致以外の要件（数値や日付による範囲検索）もある場合には、SDATAを利用する

NDATAセクション
（Oracle Database 11g Release 2（11.2）以降）

- タグに基づくセクションを作成する
  - CTX_DDL.ADD_NDATA_SECTIONプロシージャを利用して、既存のセクション・グループに、タグに基づくNDATAセクションを追加する
    ※ MULTI_COLUMN_DATASTORE を利用すれば、列に基づくセクションを作成することも可能
- セクション内の文字列の中から、スペルの近いもの、トークン順序の入れ替わったものを検索する
- NDATA演算子の構文
  ndata（*sectionname*, *phrase* [, *order*] [, *proximity*]）

NDATAセクション
（Oracle Database 11g Release 2（11.2）以降）

- NDATA演算子の構文
  ndata（*sectionname*, *phrase* [, *order*] [, *proximity*]）

変数名	デフォルト値	説明
sectionname		NDATAセクションの名前を指定
phrase		検索文字列を指定
order	NOORDER	phrase変数で複数トークンが指定されている場合、その出現順序を保つか否かを指定 ORDER または O … 順序が一致するもののみがヒット NOORDER または N … 順序が一致しないものもヒット
proximity	NOPROXIMITY	phrase変数に指定されたトークンの類似度をスコアに反映させる PROXIMITY または P … 類似度をスコアに反映させる NOPROXIMITY または N … 類似度をスコアに反映させない

NDATAセクション 使用例
（Oracle Database 11g Release 2（11.2）以降）

- 表作成

```
CREATE TABLE testtab (
  entryid     NUMBER  PRIMARY KEY,
  text        VARCHAR2(80),
  idx_column  VARCHAR2(5)
);
```

- データINSERT

```
INSERT INTO testtab VALUES(1, '00000 11111'       , 'dummy');
INSERT INTO testtab VALUES(2, '11111 00000'       , 'dummy');
INSERT INTO testtab VALUES(3, '00000 1111x'       , 'dummy');
INSERT INTO testtab VALUES(4, '1111x 00000'       , 'dummy');
INSERT INTO testtab VALUES(5, '00000 22222 1111x', 'dummy');
INSERT INTO testtab VALUES(6, '1111x 22222 00000', 'dummy');
COMMIT;
```

NDATAセクション 使用例
(Oracle Database 11g Release 2(11.2)以降)

- プリファレンス作成

```
BEGIN
  CTX_DDL.CREATE_PREFERENCE('mcd', 'MULTI_COLUMN_DATASTORE');
  CTX_DDL.SET_ATTRIBUTE('mcd', 'COLUMNS', 'text');
  CTX_DDL.CREATE_SECTION_GROUP('bsg', 'BASIC_SECTION_GROUP');
  CTX_DDL.ADD_NDATA_SECTION('bsg', 'TEXT', 'TEXT');
END;
/
```

- 索引作成

```
CREATE INDEX testidx ON testtab(idx_column)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS('SECTION GROUP bsg DATASTORE mcd');
```

NDATAセクション 使用例
(Oracle Database 11g Release 2(11.2)以降)

- NDATA演算子を利用した検索を実行

```
SQL> SELECT entryid, SCORE(1) FROM testtab WHERE CONTAINS
(idx_column, 'NDATA(text, 00000 11111)',1)>0;

   ENTRYID   SCORE(1)
---------- ----------
         1        100
         2        100
         3         94
         4         94
         5         94
         6         94
```

EntryID	Text
1	00000 11111
2	11111 00000
3	00000 1111x
4	1111x 00000
5	00000 22222 1111x
6	1111x 22222 00000

NDATAセクション 使用例
(Oracle Database 11g Release 2(11.2)以降)

- NDATA演算子を利用した検索を実行

```
SQL> SELECT entryid, SCORE(1) FROM testtab WHERE CONTAINS
(idx_column, 'NDATA(text, 00000 11111,,P)',1)>0;

   ENTRYID   SCORE(1)
---------- ----------
         1        100
         2        100
         3         94
         4         94
         5         65
         6         65
```

EntryID	Text
1	00000 11111
2	11111 00000
3	00000 1111x
4	1111x 00000
5	00000 22222 1111x
6	1111x 22222 00000

NDATAセクション 使用例
(Oracle Database 11g Release 2(11.2)以降)

- NDATA演算子を利用した検索を実行

```
SQL> SELECT entryid, SCORE(1) FROM testtab WHERE CONTAINS
(idx_column, 'NDATA(text, 00000 11111,0)',1)>0;

   ENTRYID   SCORE(1)
---------- ----------
         1        100
         3         94
         5         94
```

EntryID	Text
1	00000 11111
2	11111 00000
3	00000 1111x
4	1111x 00000
5	00000 22222 1111x
6	1111x 22222 00000

NDATAセクション 使用例
(Oracle Database 11g Release 2(11.2)以降)

- NDATA演算子を利用した検索を実行

```
SQL> SELECT entryid, SCORE(1) FROM testtab WHERE CONTAINS
(idx_column, 'NDATA(text, 00000 11111,0,P)',1)>0;

   ENTRYID   SCORE(1)
---------- ----------
         1        100
         3         94
         5         65
```

EntryID	Text
1	00000 11111
2	11111 00000
3	00000 1111x
4	1111x 00000
5	00000 22222 1111x
6	1111x 22222 00000

STOPセクション

- タグに基づくセクションのうち、索引から除外するセクションを指定する
- CTX_DDL.ADD_STOP_SECTIONプロシージャを利用して、既存のセクション・グループにSTOPセクションを定義する

属性（ATTRIBUTE）セクション

- XMLタグの属性に基づくセクションを定義する
  - XML_SECTION_GROUPでのみ利用可能
  - AUTO_SECTION_GROUP、PATH_SECTION_GROUPについては、全てのXMLタグの属性に対して自動的にセクションが定義される
- CTX_DDL.ADD_ATTR_SECTIONプロシージャを利用して、既存のセクション・グループに属性（ATTRIBUTE）セクションを追加する

NULL_SECTION_GROUP（デフォルト）

- 次のいずれかの場合に指定する
  - セクション定義を行わない場合
  - SENTENCEセクション（文セクション）またはPARAGRAPHセクション（段落セクション）のみを定義する場合
    - 文セクション、段落セクションの検索にはWITHIN演算子を利用する

BASIC_SECTION_GROUP

- 開始および終了タグが<A>および</A>という形式のセクションを定義する
- 対になっていないタグの使用は不可
  → 対になっていないタグがある場合には、HTML_SECTION_GROUP を使用する

AUTO_SECTION_GROUP

- XML文書の開始タグ・終了タグに対して自動的にゾーン・セクションを作成する
- タグ名は大文字と小文字が区別される
- 属性セクションは、属性を持つXMLタグに対して自動的に作成される
  - 属性セクションは、<*タグ名*>@<*属性名*> という形式でネーミングされる
- セクション検索にはWITHIN演算子を利用する

※ AUTO_SECTION_GROUPおよびPATH_SECTION_GROUPでは、自動的にゾーン・セクションが設定される。このため、CTX_DDL.ADD_ZONE_SECTIONでゾーン・セクションを設定する必要はない。

PATH_SECTION_GROUP
（Oracle9i Database Release 1（9.0.1）以降）

- AUTO_SECTION_GROUPの動作を包含する
- AUTO_SECTION_GROUPとの違いは、INPATH、HASPATH演算子が使用可能になる点
  - これにより、XPathを利用したセクション検索が可能となる

※ AUTO_SECTION_GROUPおよびPATH_SECTION_GROUPでは、自動的にゾーン・セクションが設定される。このため、CTX_DDL.ADD_ZONE_SECTIONでゾーン・セクションを設定する必要はない。

PATH_SECTION_GROUP
（Oracle9i Database Release 1（9.0.1）以降）

AUTO_SECTION_GROUP

- ゾーン検索やフィールド検索を行えるセクショナ
- XML文書のようにタグが何層にもネスト化され、属性を持つような文書の検索には適していない。

PATH_SECTION_GROUP

- **INPATH演算子**が利用可能･･･ワイルド･カード（子孫ノード）検索などの複雑な検索が行える。
- **HASPATH演算子**が利用可能･･･タグ値のテスト、等価性のチェックなどが行える。

INPATH演算子によるXMLタグ検索①
（Oracle9i Database Release 1（9.0.1）以降）

XMLドキュメントでパス検索をする時、この演算子を用いる。

トップレベルのタグ検索

dog INPATH(A)	<A>dog</A>	○
	<B><A>dog</A></B>	×

任意レベルのタグ検索

dog INPATH(//A)	<A>dog</A>	○
	<B><A>dog</A></B>	○

直接の親子関係の検索

dog INPATH(A/B)	<A><B>My dog is friendly</B></A>	○
	<C><B>My dog is friendly</B></C>	×

INPATH演算子によるXMLタグ検索②

（Oracle9i Database Release 1（9.0.1）以降）

単一レベルのワイルド・カード検索

dog INPATH (A/*/B)	<A><D><B>dog</B></D></A>	○

マルチレベルのワイルド・カード検索

dog INPATH (A/*/*/B)	<A><C><D><B>dog</B></D></C></A>	○

任意レベルの子検索

dog INPATH (A//B)	<A><C><D><B>dog</B></D></C></A>	○

属性の検索

dog INPATH(//A/@B)	<C><A B= “snoop dog” ></A></C>	○

タグ値のテスト

dog INPATH(A[B= “dog”])	<A><B>dog</B></A>	○
	<A><B>My dog is friendly</B></A>	✕

HASPATH演算子による検索
（Oracle9i Database Release 1（9.0.1）以降）

- 指定したセクション・パスを含むすべてのXMLドキュメントを検索する
- セクションの等価性をテスト

<A><B><C>という構造のタグが存在するかどうかを確かめたい時・・・

HASPATH(A/B/C)	<A><B><C>dog</C></B></A>	○

<A>dog</A>というタグが存在するかどうかを確かめたい時・・・

HASPATH(A="dog")	<A>dog</A>	○
	<A>dog park</A>	×

ストップリスト

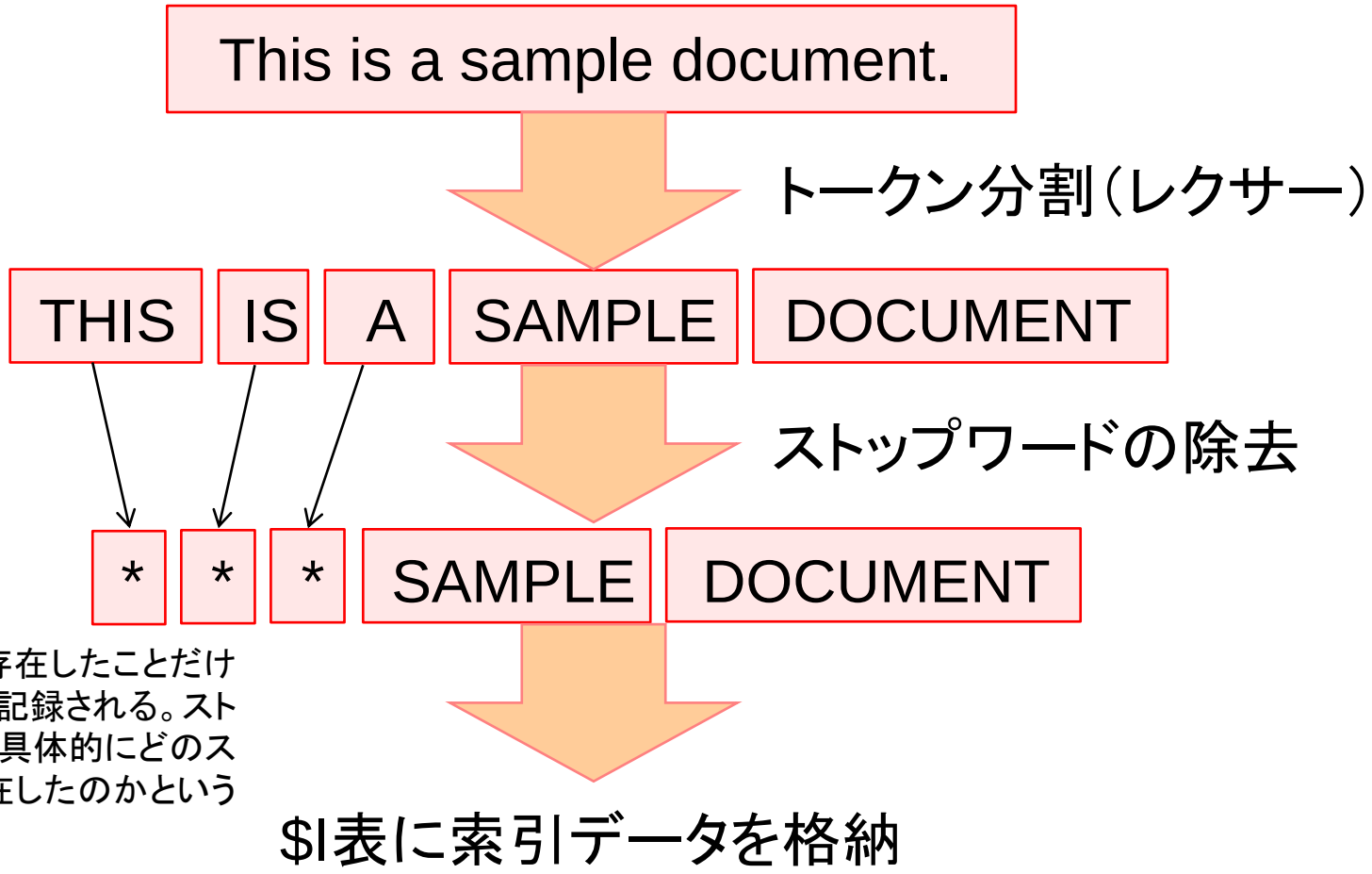
- 索引付けの対象外とするキーワードを指定する
  - 英語の冠詞a、theなど、ドキュメントを特定するのに役立たないキーワードを索引情報から排除する
- 以下の言語についてはデフォルトのストップリストが提供される:
  - 英語、中国語(繁体字・簡体字)、デンマーク語、オランダ語、フィンランド語、フランス語、ドイツ語、イタリア語、ポルトガル語、スペイン語、スウェーデン語
- 多言語ストップリストを作成することもできる
- ストップリストの機能を無効にするには、CREATE INDEX時に明示的に CTXSYS.EMPTY_STOPLIST を指定する

```
CREATE INDEX testidx ON testtab (text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('STOPLIST CTXSYS.EMPTY_STOPLIST');
```

英語のデフォルト・ストップリスト

a	can	him	Ms	should	this	which
all	could	his	my	since	those	while
almost	d	how	no	so	though	who
also	did	however	non	some	through	whose
although	do	i	nor	still	thus	why
an	does	if	not	such	to	will
and	either	in	of	t	too	with
any	for	into	on	than	until	would
are	from	is	one	that	ve	yet
as	had	it	only	the	very	you
at	has	its	onto	their	was	your
be	have	just	or	them	we	yours
because	having	ll	our	then	were	
been	he	me	ours	there	what	
both	her	might	s	therefore	when	
but	here	Mr	shall	these	where	
by	hers	Mrs	she	they	whether	

英語のデフォルト・ストップリストを使用した場合の動作



ストップワードが存在したことだけが索引情報として記録される。ストップワードのうち、具体的にどのストップワードが存在したのかという情報は失われる。

英語のデフォルト・ストップリストを使用した場合の動作

- 表・索引作成

```
CREATE TABLE testtab (
  id    NUMBER  PRIMARY KEY,
  text  VARCHAR2(4000)
);
INSERT INTO testtab VALUES (1, 'This is a sample document.');
COMMIT;
CREATE INDEX testidx ON testtab (text) INDEXTYPE IS CTXSYS.CONTEXT;
```

- $I表のデータを確認

```
SQL> SELECT token_text FROM dr$testidx$i;

TOKEN_TEXT
----------------------------------------------------------------
DOCUMENT
SAMPLE
```

英語のデフォルト・ストップリストを使用した場合の動作

- 検索条件「That was the sample document.」で、文字列「This is a sample document.」を含む文書がヒットする

```
SQL> SELECT id, text FROM testtab WHERE CONTAINS (text, 'That was the
sample document.') > 0;

        ID TEXT
---------- --------------------------------------------------
         1 This is a sample document.
```

→ 全てのストップワードは同一視される

- 検索条件「That was the very sample document.」の場合は・・・

```
SQL> SELECT id, text FROM testtab WHERE CONTAINS (text, 'That was the
very sample document.') > 0;

no rows selected
```

→ ストップワードの個数が一致しないため、ヒットしない。

ストップリストを設定した場合の動作

- ストップワードが、どの位置に、何個存在したのかという情報は保持される
- ストップワードのうち、具体的にどのストップワードがそこにあったのかという情報は失われる
- 検索文字列に含まれるストップワードは、検索対象の任意のストップワードと一致する
  - 検索条件「That was the sample document.」で、文字列「This is a sample document.」を含む文書がヒットする
    ※ 青文字（下線）部分がストップワード

※ データベース言語が英語に設定されたデータベースで、日本語文書に対して索引を作成すると、日本語文書に含まれる英単語が、意図せずストップワードとして処理されてしまう場合がある。例えば「IT産業」に含まれる「IT」など。日本語文書に含まれる英単語をストップワードとして扱いたくない場合には、索引作成時に CTXSYS.EMPTY_STOPLIST を明示的に指定し、ストップワードの機能を無効にする。

デフォルト・ストップリストの言語

- デフォルト・ストップリストは、データベース作成時のデータベース言語に基づいて作成される
- このため、1つのデータベースで利用可能なデフォルト・ストップリストは1種類（1言語）のみ
- 例えば、データベース言語を英語で作成した場合、フランス語のデフォルト・ストップリストは利用できない
- 日本語ではデフォルト・ストップリストが提供されないため、データベース言語が日本語の場合には、ストップリストは空になる

システム定義プリファレンス

- システム定義プリファレンスは、ユーザーが明示的に作成せずに、即時に利用できる
- システム定義プリファレンス（利用頻度が高いと思われるものを抜粋）
  - フィルタ
    CTXSYS.AUTO_FILTER
  - セクション・グループ
    CTXSYS.NULL_SECTION_GROUP
    CTXSYS.AUTO_SECTION_GROUP
    CTXSYS.PATH_SECTION_GROUP
  - ストップリスト
    CTXSYS.EMPTY_STOPLIST（ストップリストの機能を利用しない場合に指定）

システム定義プリファレンス

- 例えば、システム定義プリファレンスのうち、
  - CTXSYS.AUTO_FILTER
  - CTXSYS.PATH_SECTION_GROUP
  - CTXSYS.EMPTY_STOPLIST

を指定して索引を作成するには・・・

```
CREATE INDEX testidx ON testtab (text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('
    FILTER          CTXSYS.AUTO_FILTER
    SECTION GROUP   CTXSYS.PATH_SECTION_GROUP
    STOPLIST        CTXSYS.EMPTY_STOPLIST
  ');
```

システム・パラメータ（抜粋）

システム・パラメータ	説明
MAX_INDEX_MEMORY	CREATE INDEXおよびALTER INDEXのPARAMETERS句に指定できる最大の索引付けメモリー。このパラメータの最大値は2GB-1。
DEFAULT_INDEX_MEMORY	CREATE INDEXおよびALTER INDEXとともに使用されるデフォルトの索引付けメモリー。
LOG_DIRECTORY	CTX_OUTPUTログ・ファイル用のディレクトリ。
CTX_DOC_KEY_TYPE	CTX_DOCプロシージャに対するデフォルト入力キー・タイプ（ROWIDまたはPRIMARY_KEYのいずれか）。インストール時にROWIDに設定される。

システム・パラメータの全リスト、およびその設定値は、CTX_PARAMETERSビューで確認できる

索引作成(2) ～ 様々なオプション機能

※ この章では、プリファレンス以外によるテキスト索引のカスタマイズ方法について解説する

キャラクタ・セット列

ID	FILENAME	CSET
1	sjis.txt	JA16SJIS
2	utf8.txt	AL32UTF8
3	euc.txt	JA16EUC
…	…	…

索引作成対象の列

キャラクタ・セット列

※ キャラクタ・セット列には、JA16SJISなどのグローバリゼーション・サポート・キャラクタ・セット名を指定する。

※ データベースキャラクタセットと異なるキャラクタセットの文書を検索対象とする場合に指定する。

```
CREATE INDEX testidx ON testtab(filename)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('DATASTORE my_filestore
    CHARSET COLUMN cset');
```

フォーマット列（AUTO_FILTER使用時のみ）

ID	FILENAME	CSET	FMT
1	index.html	AL32UTF8	TEXT
2	otext.pptx	JA16SJIS	BINARY
3	map.jpg		IGNORE
…	…	…	…

索引作成対象の列

キャラクタ・セット列

フォーマット列

※ フィルタ処理をスキップして索引を作成する場合
→TEXT
フィルタ処理を実行して索引を作成する場合
→BINARY
フィルタ処理をスキップして索引作成対象からも除外する場合
→IGNORE

```
CREATE INDEX testidx ON testtab(filename)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('DATASTORE my_filestore FILTER auto_filter
    CHARSET COLUMN cset FORMAT COLUMN fmt');
```

言語列(MULTI_LEXER使用時のみ)

ID	BODY	LANG
1	日本語文書の本文です・・・	ja
2	An English document ...	us
3	Un Français documente ...	f
...	...	...

索引作成対象の列

言語列

※ 言語列に、言語名またはその略称を指定する。

※ MULTI_LEXER使用時は、言語列の指定が必須。

※ 指定可能な全ての言語名とその略称は、巻末付録「Oracleデータベースがサポートしている言語」を参照。

```
CREATE INDEX testidx ON testtab(body)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('LEXER my_multi LANGUAGE COLUMN lang');
```

Index Memory

- 索引作成に使用するランタイム・メモリ量を指定する
- 設定方法

```
CREATE INDEX testidx ON testtab(body)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('MEMORY 512M');
```

※ 使用可能な単位は、K(キロバイト)、M(メガバイト)、G(ギガバイト)の3種類。

※ デフォルト値は、Oracle Textのシステム変数DEFAULT_INDEX_MEMORYに設定された値。

※ DEFAULT_INDEX_MEMORYのデフォルト値は12 MB。通常、この値は小さすぎる。

Index Memory

- **Index Memory は、物理メモリが許す限り大きく取る**
  ※ Index Memory は **PGA 内に確保される**ため、PGAをIndex Memoryよりも大きく確保しておく必要がある
  ※ パラレル索引作成では、Index Memory は、各スレーブ・プロセス毎に取られる。たとえば、「MEMORY 256M」と「PARALLEL 4」を同時に指定した場合、Index Memory は合計で 256 MB × 4 = 1,024 MB となる
- Index Memory を大きく設定すると・・・
  - 索引作成時のディスクI/Oが削減される→高速な索引作成処理
  - 索引の断片化が削減される→高速な検索処理
- Index Memory は、Oracle Textのシステム変数 MAX_INDEX_MEMORYよりも小さい値を設定する必要がある

※ MAX_INDEX_MEMORYのデフォルト値は1 GB。

※ DEFAULT_INDEX_MEMORYおよびMAX_INDEX_MEMORYの値は、CTXSYS.CTX_PARAMETERS ビューで確認できる。これらのパラメータ値は、CTX_ADM.SET_PARAMETERプロシージャで変更可能。

同期化オプション

※ EVERY句およびON COMMITでの索引同期化は、Oracle Database 10g Release 1（10.1）以降で利用可能。

- 検索対象の表に対するDML処理を、索引情報に反映させるタイミングを指定する
  - **MANUAL（デフォルト）**
    CTX_DDL.SYNC_INDEXを使用して、索引を手動で同期化する
  - **EVERY "<*interval-string*>"**
    *interval-string*の値で指定された一定の間隔で、索引を自動的に同期化する。*interval-string*は、スケジューラ・ジョブと同じ構文をとる（索引作成ユーザーにCREATE JOB権限が必要）
  - **ON COMMIT**
    コミットが発行されたら直ちに索引が同期化される。コミットは、同期化が完了するまで戻らない。

同期化オプション

※ EVERY句およびON COMMITでの索引同期化は、Oracle Database 10g Release 1（10.1）以降で利用可能。

- 使用例（ON COMMIT）

```
CREATE INDEX testidx ON testtab (text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('SYNC (ON COMMIT)');
```

- 使用例（EVERY句）

```
CREATE INDEX testidx ON testtab (text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('SYNC (EVERY
"NEXT_DAY(ADD_MONTHS(TRUNC(SYSDATE,''Q''),3),''THURSDAY'')")'
);
```

※ TRUNC(SYSDATE, 'Q') で四半期（3ヶ月）の切り捨てを行い、次に ADD_MONTHS(*, 3) で3ヶ月の繰り上げを行い、最後に NEXT_DAY(*, 'THURSDAY') で * 以降の最初の木曜日まで繰り上げを行っている。即ち、次に同期化が実行されるのは、次の四半期の最初の木曜日。

※ EVERY句およびON COMMITでの索引同期化は、Oracle Database 10g Release 1（10.1）以降で利用可能。

同期化オプション

すでに作成済みの索引の同期化オプションを変更する方法

- 作成済みの索引の同期化オプションをON COMMITに変更するには・・・

```
ALTER INDEX <索引名> REBUILD PARAMETERS
('REPLACE METADATA SYNC (ON COMMIT)');
```

- 作成済みの索引の同期化オプションをMANUALに変更するには・・・

```
ALTER INDEX <索引名> REBUILD PARAMETERS
('REPLACE METADATA SYNC (MANUAL)');
```

※ 索引の作り直しをせずに、同期化オプションを変更可能。

パラレル索引作成

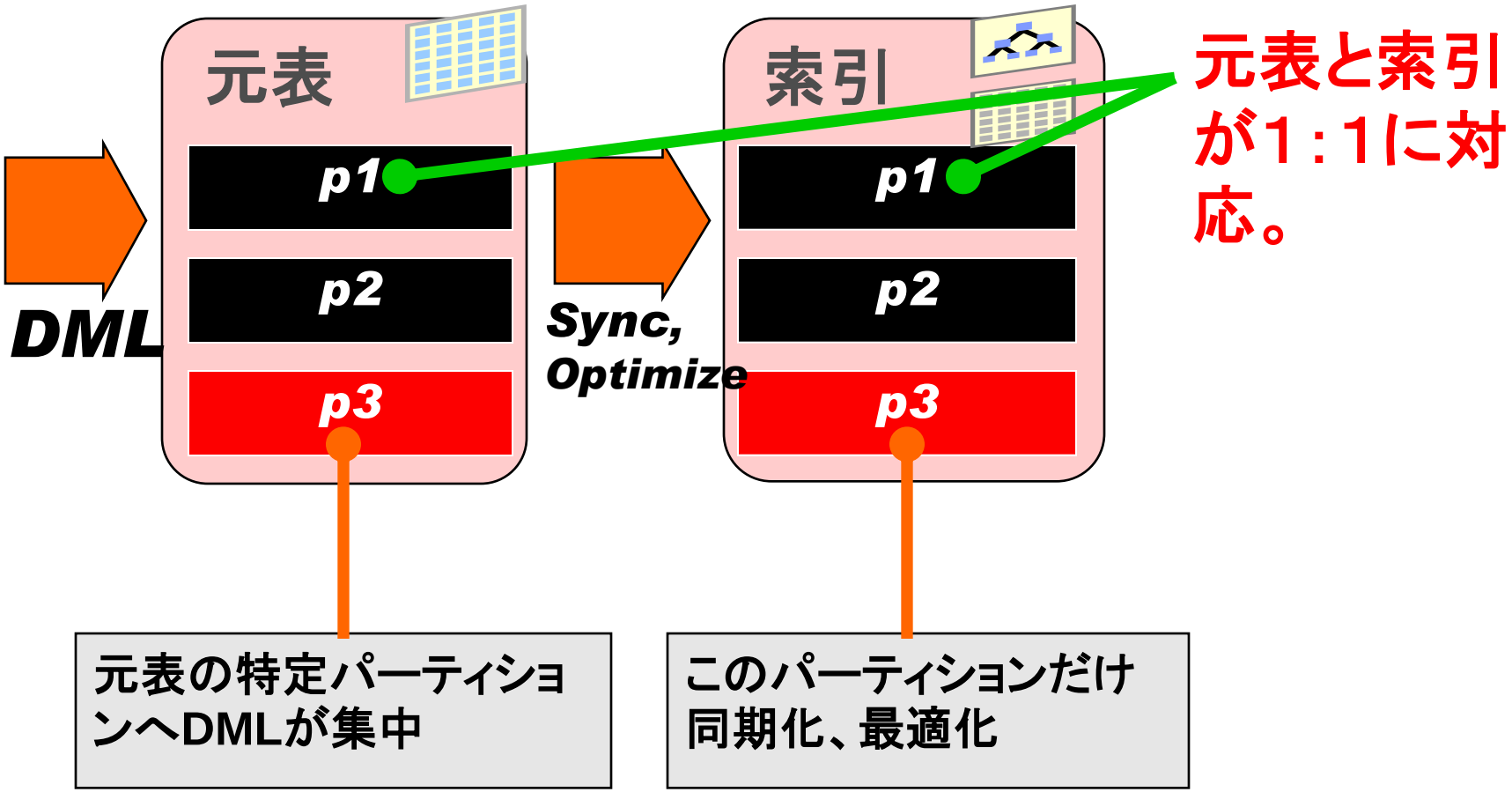
※ CREATE INDEX文で指定するパラレル度が実際のパラレル度数に与える影響は、Oracle Databaseのバージョンによって異なる。詳細は各バージョンのマニュアルを参照。

- 索引作成処理を、複数のプロセスで同時に実行可能
- 使用例

```
CREATE INDEX testidx ON testtab (text)
  INDEXTYPE IS CTXSYS.CONTEXT
  PARAMETERS ('...') PARALLEL 4;
```

- パーティション索引でなくてもパラレル索引作成が可能
  ※ Oracle9i Database Release 2(9.2)以降で可能
- Index Memory は、各スレーブ・プロセス毎に取られる
  - 例:「MEMORY 256M」と「PARALLEL 4」を同時に指定した場合、Index Memory は合計で 256 MB × 4 = 1,024 MB となる

ローカル・パーティション索引
（Oracle9i Database Release 1（9.0.1）以降）



ローカル・パーティション索引 使用例
(Oracle9i Database Release 1(9.0.1)以降)

- 準備(表作成)

```
CREATE TABLE part_tab3(
  id NUMBER PRIMARY KEY,
  text VARCHAR2(100))
 PARTITION BY RANGE(id)
 (PARTITION p1 VALUES LESS THAN (1000),
  PARTITION p2 VALUES LESS THAN (2000),
  PARTITION p3 VALUES LESS THAN (3000));
```

※ CONTEXT索引のローカル・パーティション索引は、
レンジ・パーティションのみに対応

ローカル・パーティション索引 使用例
(Oracle9i Database Release 1(9.0.1)以降)

- 索引作成

```
CREATE INDEX part_tab3x ON part_tab3(text)
  INDEXTYPE IS CTXSYS.CONTEXT LOCAL (
    PARTITION part_tabx1 PARAMETERS('SYNC(ON COMMIT)'),
    PARTITION part_tabx2,
    PARTITION part_tabx3
  )
  PARAMETERS('
    LEXER my_lexer
    MEMORY 256M
    SYNC (EVERY "NEXT_DAY(ADD_MONTHS(TRUNC(SYSDATE,
''Q''), 3),''THURSDAY'')")
  ')
  PARALLEL 2;
```

ローカル・パーティション索引
(Oracle9i Database Release 1(9.0.1)以降)

- パーティション毎にPARAMETERS句を設定可能
  - たとえば、前ページの例のように、パーティション毎に同期化オプションを設定できる
- パーティション毎にメンテナンス(同期化、最適化)を実行可能
  - たとえば、日付(年・月)によるレンジ・パーティションで、先月分以前のデータに更新がないため、今月分のパーティションのみを同期化・最適化する、などの運用が可能

索引メンテナンス

※ この章では、テキスト索引の同期化および最適化について解説する

索引メンテナンス　～ サンプルデータ

- 検索対象表(Master表)のTitle列に対し、全文検索用の索引を作成

検索対象表(表名:Master)

ID	Title	・・・・
1	ORACLEデータベース管理	・・・・
2	簡単ORACLEチューニング	・・・・

索引作成

トークン表

トークン	出現位置
ORACLE	(1,1), (2,2)
データ	(1,2)
タベ	(1,3)
ベース	(1,4)
管理	(1,5)
簡単	(2,1)
チューニ	(2,3)
ニング	(2,4)

※ (1,1)は、(ID, トークン位置)

索引メンテナンス　～ 新規文書情報の追加

- テキスト索引の同期化
  - ✓ リアルタイムでも可能だが、バッチ処理で行う方が索引が断片化せず、効率的

検索対象表（表名：Master）

ID	Title	・・
1	ORACLEデータベース管理	・・
2	簡単ORACLEチューニング	・・
追加 3	ORACLE バックアップ	・

新たに追加されたトークンが索引の末尾に追加

トークン表

トークン	出現位置
ORACLE	(1,1), (2,2)
データ	(1,2)
タベ	(1,3)
ベース	(1,4)
管理	(1,5)
簡単	(2,1)
チューニ	(2,3)
ニング	(2,4)
ORACLE	(3,1)
バック	(3,2)
クア	(3,3)
アップ	(3,4)

```
EXEC CTX_DDL.SYNC_INDEX( <索引名> );
```

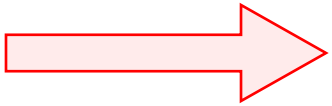
索引メンテナンス　～ 新規文書情報の追加

- テキスト索引の最適化

トークン表

トークン	出現位置
ORACLE	(1,1), (2,2)
データ	(1,2)
タベ	(1,3)
ベース	(1,4)
管理	(1,5)
簡単	(2,1)
チューニ	(2,3)
ニング	(2,4)
ORACLE	(3,1)
バック	(3,2)
クア	(3,3)
アップ	(3,4)

テキスト索引に
複数エントリ
されている
同一トークン
をまとめる



トークン表

トークン	出現位置
ORACLE	(1,1), (2,2), **(3,1)**
データ	(1,2)
タベ	(1,3)
ベース	(1,4)
管理	(1,5)
簡単	(2,1)
チューニ	(2,3)
ニング	(2,4)
バック	(3,2)
クア	(3,3)
アップ	(3,4)

```
EXEC CTX_DDL.OPTIMIZE_INDEX(<索引名>,' FAST' );
```

索引メンテナンス　～ 文書情報の削除

- テキスト索引の同期化

検索対象表（表名：Master）

ID	Title	・・
1	ORACLEデータベース管理	・・
2	簡単ORACLEチューニング	・・
3	ORACLEバックアップ	・・

追加

トークン表

トークン	出現位置
ORACLE	(1,1), (2,2), (3,1) 削除済
データ	(1,2)
タベ	(1,3)
ベース	(1,4)
管理	(1,5)
簡単	(2,1) 削除済
チューニ	(2,3) 削除済
ニング	(2,4) 削除済
バック	(3,2)
クア	(3,3)
アップ	(3,4)

検索対象表への削除処理を、Oracle Text は即座に検知して検索結果に反映させる

※ 文書情報の削除は、リアルタイムに検索結果に反映されるため、コマンドの実行は不要。

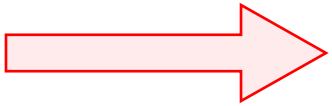
索引メンテナンス　～ 文書情報の削除

- テキスト索引の最適化

トークン表

トークン	出現位置
ORACLE	(1,1), ~~(2,2),~~ (3,1)
データ	(1,2)
タベ	(1,3)
ベース	(1,4)
管理	(1,5)
~~簡単~~	~~(2,1)~~
~~チューニ~~	~~(2,3)~~
~~ニング~~	~~(2,4)~~
バック	(3,2)
クア	(3,3)
アップ	(3,4)

削除されて既に検索対象表に存在していないトークンを削除



トークン表

トークン	出現位置
ORACLE	(1,1), (3,1)
データ	(1,2)
タベ	(1,3)
ベース	(1,4)
管理	(1,5)
バック	(3,2)
クア	(3,3)
アップ	(3,4)

```
EXEC CTX_DDL.OPTIMIZE_INDEX(<索引名>,'FULL');
```

チューニング

※ この章では、テキスト索引の作成時、メンテナンス時のパフォーマンスや、検索処理のパフォーマンスに関する考慮事項を解説する

索引作成が遅い場合

- パラレルでの索引作成を検討する
  - パラレル索引作成は、非パーティション索引、パーティション索引のいずれに対しても有効
- Index Memoryを、物理メモリが許す限り大きく設定する
  - Index MemoryはPGA内に取られるため、PGAについても十分に大きく設定しておく必要がある
  - Index Memory は、各スレーブ・プロセス毎に取られる
    - 例:「MEMORY 256M」と「PARALLEL 4」を同時に指定した場合、Index Memory は合計で 256 MB × 4 = 1,024 MB となる
- 特定のドキュメントのフィルタ処理に時間がかかっている場合には、フィルタのタイムアウト値を設定する

検索が遅い場合(全般)

- 1つのSELECT文に含まれるCONTAINS句を1つにする
  - 全文検索が複数列にまたがる場合には、MULTI_COLUMN_DATASTOREを利用する
- SELECT文に、CONTAINS句以外の条件式や、ORDER BY句が含まれている場合、コンポジット・ドメイン索引の利用を検討する
- 検索結果を全件フェッチするようなアプリケーションになっている場合には、FIRST_ROWS(*n*) ヒントを指定した上で、結果画面表示に必要な最低限の行数をフェッチする動作に変更する
- SELECT文で、CONTAINS句の条件と同時に、他表との結合処理が含まれる場合には、マテリアライズド・ビューの利用を検討する
  - 他表と結合した状態のマテリアライズド・ビューを作成し、このマテリアライズド・ビューに対してテキスト索引を作成する
  - これによって、SELECT文から結合処理を除外する

検索が遅い場合（索引断片化）

- 索引作成時、索引同期化時のIndex Memoryを、物理メモリが許す限り大きく設定する（これにより、索引の断片化を防ぐ）
- 索引同期化の頻度を出来る限り小さくする（これにより、索引の断片化を防ぐ）
  - 索引の同期化を繰り返すことによって、索引が断片化し、検索パフォーマンスが劣化する
    → 索引の最適化を実行することで、検索パフォーマンスを回復できる
- 索引が断片化している場合には、FULLモード、あるいはREBUILDモードでの索引最適化を実行する

検索が遅い場合（日本語）

- JAPANESE_VGRAM_LEXERで、特定のキーワードでの検索が遅い場合
  - レクサーを変更する
    - JAPANESE_LEXER＋レキシコンのカスタマイズ
      ※ Oracle Database 10g Release 1（10.1）以降で利用可能
    - AUTO_LEXER
      ※ Oracle Database 11g Release 1（11.1）以降で利用可能
- 内部的に一文字検索が発生している場合
  - BASIC_WORDLISTのPREFIX_INDEX属性をTRUEに設定する
    ※ ただし、この方法ではそれほど改善効果が見込めない場合が多い
- 日本語文書に含まれる英数字のワイルドカード検索が遅い場合
  - BASIC_WORDLISTのSUBSTRING_INDEX属性をTRUEに設定する

索引同期化が遅い場合

- パラレルでの索引同期化を検討する
  - パラレルでの索引同期化は、非パーティション索引、パーティション索引のいずれに対しても有効
- Index Memoryを、物理メモリが許す限り大きく設定する
  - Index MemoryはPGA内に取られるため、PGAについても十分に大きく設定しておく必要がある
  - Index Memory は、各スレーブ・プロセス毎に取られる
    - 例:「MEMORY 256M」と「PARALLEL 4」を同時に指定した場合、Index Memory は合計で 256 MB × 4 = 1,024 MB となる
- 特定のドキュメントのフィルタ処理に時間がかかっている場合には、フィルタのタイムアウト値を設定する

索引最適化が遅い場合

- パラレルでの索引最適化を検討する
- REBUILDモードでの索引最適化を検討する
  ※ REBUILDモードでの索引最適化はOracle Database 10g Release 1（10.1）以降で利用可能。
  - 索引が断片化が進むと、FULLモードでの最適化に、極端に長い時間がかかるようになる
  - この場合、REBUILDモードでの最適化を利用すると、新規に索引を作成するのと同程度の時間で最適化を完了できる
  - ただし、REBUILDモードでの最適化では、一時的に新旧2つの索引が同時に保持されるため、索引用のディスク領域が一時的に2倍必要となる

PGAとSGA

Oracle Textでは、

- 索引作成時、および索引メンテナンス時には、PGAを多く必要とする
- 検索時には、SGAを多く必要とする

参考資料

『Oracle Text パフォーマンス FAQ』
http://otndnld.oracle.co.jp/products/iserver/oracle9i/htdocs/o9i_920_otpf_10_1928.html

その他

※ この章では、テキスト索引に関連した話題を紹介する

索引作成・同期化・最適化のロギング

- CTX_OUTPUTパッケージ
  - ADD_EVENT
    - CTX_OUTPUT.EVENT_INDEX_PRINT_ROWID
      索引作成が完了したROWIDをログ・ファイルに出力する
      ※ Oracle9i Database Release 1（9.0.1）以降で利用可能。
    - CTX_OUTPUT.EVENT_OPT_PRINT_TOKEN
      最適化処理中のトークンをログ・ファイルに出力する
      ※ Oracle Database 10g Release 1（10.1）以降で利用可能。
    - CTX_OUTPUT.EVENT_INDEX_PRINT_TOKEN
      索引作成が完了したトークンをログ・ファイルに出力する
      ※ Oracle Database 10g Release 2（10.2）以降で利用可能。
    - CTX_OUTPUT.EVENT_DRG_DUMP_ERRORSTACK
      指定されたDRGエラー番号のスタック・トレースをログ・ファイルに出力する ※ Oracle Database 11g Release 2（11.2）以降で利用可能。
  - START_LOG、END_LOG

統計情報の取得

- 索引統計 ※ Oracle9i Database Release 2（9.2）以降で利用可能。
  - CTX_REPORT.INDEX_STATS プロシージャ
    - トークン情報
    - 断片化の状況
- 検索統計 ※ Oracle Database 10g Release 1（10.1）以降で利用可能。
  - CTX_OUTPUT.START_QUERY_LOG
  - CTX_OUTPUT.END_QUERY_LOG

  → 次ページのような検索ログが LOG_DIRECTORY に出力される

統計情報の取得

検索ログ・ファイル サンプル

```
<QuerySet>
  <TimeStamp>15:36:19 02/15/11</TimeStamp>
  <IndexName>IDX_QLOG1</IndexName>
  <Query>France</Query>
  <ReturnHit>Yes</ReturnHit>
  <QueryTag></QueryTag>
</QuerySet>
<QuerySet>
  <TimeStamp>15:36:19 02/15/11</TimeStamp>
  <IndexName>IDX_QLOG1</IndexName>
  <Query>cheese</Query>
  <ReturnHit>No</ReturnHit>
  <QueryTag></QueryTag>
</QuerySet>
...
```

フィルタ部分のみを単体で利用する（Oracle9i Database Release 2（9.2）以降）

- BLOB列に格納されたデータや、ファイルシステム上のファイルなどから、テキスト文字列を抽出できる
  - AUTO_FILTER（p.111）で利用されているのと同じモジュールによって、バイナリからテキスト文字列を抽出する
    ※ INSO_FILTERが動作する古いOracle Databaseバージョンでは、INSO_FILTERのバイナリが呼び出される。
- CTX_DOCパッケージ
  - FILTERプロシージャ（事前の索引作成が必要。索引メタデータのみが参照されるため、索引の中身は空でも構わない）
    ※ Oracle9i Database Release 2（9.2）以降で利用可能
  - POLICY_FILTERプロシージャ（事前の索引作成が不要）
    ※ Oracle Database 10g Release 1（10.1）以降で利用可能

PL/SQLパッケージによる索引メンテナンス
（ALTER INDEXコマンドとの違い）

- **索引の同期化、最適化には、PL/SQLパッケージの利用を推奨**
  - ALTER INDEX ... REBUILD による同期化、最適化は推奨されない

【理由】

- PL/SQLパッケージでも、ALTER INDEX コマンドでも、その内部動作は同じ
- ただし、ALTER INDEXコマンドは、SQLの制限として、「1つの索引に対して同時に1つまでしか実行できない」というものがある
- PL/SQLパッケージを利用すれば、特定のパーティションの同期化中に、別のパーティションの最適化するなどの運用が可能となる
  ※ 実際には、1つのパーティションに対して同期化と最適化を同時に実行することも可能

CREATE INDEX 文の取得
(Oracle9i Database Release 2(9.2)以降)

- 既存のテキスト索引に対して、その関連するプリファレンス作成のためのスクリプトと、索引作成のためのスクリプトを生成できる(CTX_REPORT.CREATE_INDEX_SCRIPT プロシージャ)

使用例

```
VARIABLE g_clob CLOB
BEGIN
    DBMS_LOB.CREATETEMPORARY(:g_clob, TRUE);
    CTX_REPORT.CREATE_INDEX_SCRIPT('<索引名>', :g_clob);
END;
/
PRINT g_clob
```

検索アプリケーション開発

- Oracle Text を利用することで、**全文検索アプリケーションの開発は、SQLのみで完結**する
  - 全文検索用の特別なAPIは特になし
  - 全文検索の条件は、SQL文中のCONTAINS関数の第2引数に、文字列型として渡すのみ
  - したがって、全文検索アプリケーションが、SQLを利用した従来のアプリケーションと同様に開発できる

CTXSYSユーザー

- Oracle Text 関連のデータベース・オブジェクトを管理するためのデータベース・ユーザー
  - CONTEXT索引タイプ、CTXAPP索引タイプ、CTXRULE索引タイプを所有
  - CONTAINS関数、SCORE関数、CATSEARCH関数、MATCHES関数、MATCH_SCORE関数を所有
  - CTX_* パッケージを所有
  - CTX_* ビューを所有

```
SQL> SELECT object_name, object_type
  2    FROM dba_objects
  3    WHERE OWNER = 'CTXSYS'
  4    AND OBJECT_TYPE IN ('INDEXTYPE','OPERATOR')
  5    ORDER BY object_type, object_name;

OBJECT_NAME                    OBJECT_TYPE
------------------------------ ----------------
CONTEXT                        INDEXTYPE
CTXCAT                         INDEXTYPE
CTXRULE                        INDEXTYPE
CTXXPATH                       INDEXTYPE
CATSEARCH                      OPERATOR
CONTAINS                       OPERATOR
MATCHES                        OPERATOR
MATCH_SCORE                    OPERATOR
SCORE                          OPERATOR
...
```

CTXAPPロールに含まれる権限

- Oracle Textの索引作成に必要な権限が含まれる（Oracle Textが提供するPL/SQLパッケージの実行権限など）

※ 索引の作成対象となる表に対する読み取り権限は別途必要

```
SQL> SELECT * FROM role_sys_privs WHERE role='CTXAPP';

レコードが選択されませんでした。

SQL> SELECT * FROM role_tab_privs WHERE role='CTXAPP'
  2    ORDER BY table_name, privilege;

ROLE   OWNER  TABLE_NAME      COLUMN_NAME PRIVILEGE GRANTABLE
------ ------ --------------- ----------- --------- ---------
CTXAPP CTXSYS CTX_DDL                     EXECUTE   NO
CTXAPP CTXSYS CTX_ENTITY                  EXECUTE   NO
CTXAPP CTXSYS CTX_OUTPUT                  EXECUTE   NO
CTXAPP CTXSYS CTX_THES                    EXECUTE   NO
CTXAPP CTXSYS CTX_TREE                    EXECUTE   NO
CTXAPP CTXSYS CTX_ULEXER                  EXECUTE   NO
CTXAPP CTXSYS DR$THS                      INSERT    NO
CTXAPP CTXSYS DR$THS_BT                   INSERT    NO
CTXAPP CTXSYS DR$THS_FPHRASE              INSERT    NO
CTXAPP CTXSYS DR$THS_PHRASE               INSERT    NO
CTXAPP CTXSYS DR$THS_PHRASE               UPDATE    NO
CTXAPP CTXSYS DRIENTL                     EXECUTE   NO
CTXAPP CTXSYS DRITHSL                     EXECUTE   NO

13行が選択されました。
```

Enterprise Edition のライセンスが必要となる場合（例）

- ローカル・パーティション索引を利用する場合
  - Enterprise Edition（以下EE）＋Partitioning オプションが必要
- 検索対象がCLOB列またはBLOB列で、これらにSecureFilesを利用する場合
  - EE＋Advanced Compression オプションが必要
- パラレル索引作成を利用する場合
  - EEが必要

  ※ パーティション索引でなくてもパラレル作成は可能
- パラレル・クエリを利用する場合
  - EE＋Partitioning オプションが必要

  ※ Oracle Textのパラレル・クエリはパーティション索引に対してのみ有効

  ※ ただし、パラレル・クエリを利用して処理が高速になるのは、シリアルでの処理時間が10秒以上程度のDSS系の処理

ドメイン索引

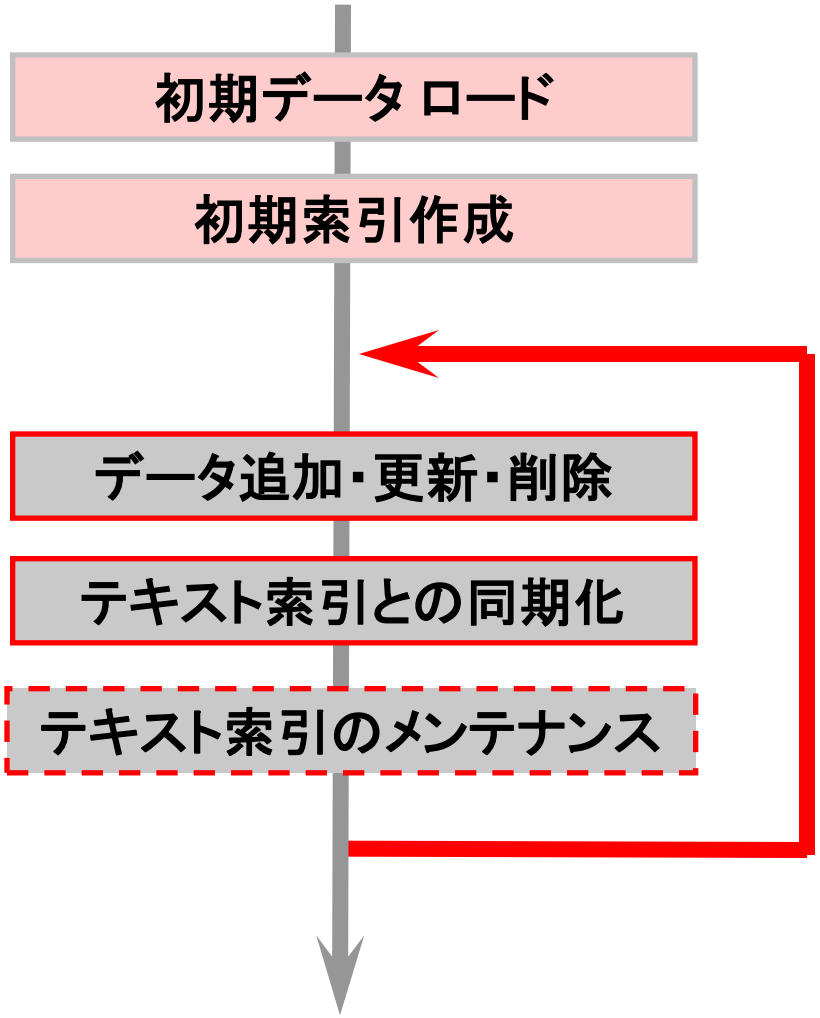
- ドメイン索引とは、空間処理やイメージ処理など、特定のアプリケーションのために作成される索引のこと
- Oracle Text の3種類の索引（CONTEXT索引、CTXAPP索引、CTXRULE索引）は、ドメイン索引として実装されている
- 他に、Oracle Database の構成要素でドメイン索引を利用しているものとしては、Oracle Multimedia、Oracle Spatial、Oracle XML DBなどがある
- ユーザー定義索引として、ドメイン索引を新規に作成することも可能

ドメイン索引固有のSQL関数（Oracle Text）

- CONTEXT索引 → CONTAINS関数
- CTXCAT索引 → CATSEARCH関数
- CTXRULE索引 → MATCHES関数

【付録】Oracle Text の簡単な使用例

Oracle Text 使用手順 ～ 一連の流れ



- 環境作成
  - ✓ 初期データロード
  - ✓ 初期索引作成
- 索引メンテナンス
  - ✓ データ追加・更新・削除
  - ✓ 索引の同期化（SYNC）
    - 検索表にデータが追加され、新しく抽出されたトークンが、トークン表に格納される
    - リアルタイム or バッチで行うことが可能
  - ✓ 索引のメンテナンス（OPTIMIZE）
    - トークン表の断片化を解消し、検索パフォーマンスを回復させる
    - FASTモード、FULLモード、REBUILDモード等の最適化が可能

Oracle Text 使用手順 ～ 初期索引作成 1

1. 利用するユーザに以下の権限を与える
   - CTXAPP ロール

```
SQL> CONNECT / AS SYSDBA
SQL> GRANT CTXAPP TO 〈ユーザ名〉;
```

2. プリファレンス（全文検索対象のデータがどんなものかを示す指定）を作成

```
SQL> CONNECT 〈ユーザ名〉/〈パスワード〉
SQL> BEGIN
  2    CTX_DDL.CREATE_PREFERENCE('〈プリファレンス名〉',
  3                              'JAPANESE_VGRAM_LEXER');
  4  END;
  5  /
```

これで「対象データは日本語」という指定になる

※ 今回の場合は、以下の項目を入力。
<プリファレンス名>：任意

Oracle Text 使用手順 ～ 初期索引作成 2

3. 全文検索用の索引を作成

```
SQL> CREATE INDEX 〈索引名〉 ON 〈表名〉(〈列名:〉)
  2    INDEXTYPE IS CTXSYS.CONTEXT
  3    PARAMETERS ('LEXER 〈プリファレンス名〉');

索引が作成されました。
```

※ 〈プリファレンス名〉: 2.と同じ名前

4. 全文検索を実行

```
SQL> SELECT 〈列名〉 FROM 〈表名〉
  2    WHERE CONTAINS (〈列名〉,'〈検索文字列〉')>0;
```

【付録】参考文献

- 製品マニュアル
  - 『Oracle Text リファレンス』
    http://otndnld.oracle.co.jp/document/products/oracle11g/111/doc_dvd/text.111/E05789-02/toc.htm
  - 『Oracle Text アプリケーション開発者ガイド』
    http://otndnld.oracle.co.jp/document/products/oracle11g/111/doc_dvd/text.111/E05788-02/toc.htm
- OTN（Oracle Technology Network）
  - Oracle Text 製品情報
    http://www.oracle.com/technetwork/jp/database/enterprise-edition/index-086432-ja.html
  - 『Oracle Text パフォーマンス FAQ』
    http://blogs.oracle.com/oracle4engineer/column/technical/026207.html

【付録】参考文献

- 書籍
  - ✓ 日下部明、他 著『これは使えるOracle新機能活用術』（翔泳社、2009年6月16日、ISBN: 9784798119915）
    - 第10章「Oracleカーネルの高速な全文検索機能」（p.180～199）

【付録】Oracleデータベースがサポートしている言語

※出典:『Oracle Databaseグローバリゼーション・サポート・ガイド』付録A

言語名	略称	言語名	略称	言語名	略称
ALBANIAN	sq	FRENCH	f	MALAYALAM	ml
AMERICAN	us	GERMAN	din	MARATHI	mr
ARABIC	ar	GERMAN DIN	d	MEXICAN SPANISH	esm
ASSAMESE	as	GREEK	el	NORWEGIAN	n
AZERBAIJANI	az	GUJARATI	gu	ORIYA	or
BANGLA	bn	HEBREW	iw	POLISH	pl
BELARUSIAN	be	HINDI	hi	PORTUGUESE	pt
BRAZILIAN PORTUGUESE	ptb	HUNGARIAN	hu	PUNJABI	pa
BULGARIAN	bg	ICELANDIC	is	ROMANIAN	ro
CANADIAN FRENCH	frc	INDONESIAN	in	RUSSIAN	ru
CATALAN	ca	IRISH	ga	SIMPLIFIED CHINESE	zhs
CROATIAN	hr	ITALIAN	i	SLOVAK	sk
CYRILLIC KAZAKH	ckk	JAPANESE	ja	SLOVENIAN	sl
CYRILLIC SERBIAN	csr	KANNADA	kn	SPANISH	e
CYRILLIC UZBEK	cuz	KOREAN	ko	SWEDISH	s
CZECH	cs	LATIN AMERICAN SPANISH	esa	TAMIL	ta
DANISH	dk	LATIN SERBIAN	lsr	TELUGU	te
DUTCH	nl	LATIN UZBEK	luz	THAI	th
EGYPTIAN	eg	LATVIAN	lv	TRADITIONAL CHINESE	zht
ENGLISH	gb	LITHUANIAN	lt	TURKISH	tr
ESTONIAN	et	MACEDONIAN	mk	UKRAINIAN	uk
FINNISH	sf	MALAY	ms	VIETNAMESE	vn

以上の事項は、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。以下の事項は、マテリアルやコード、機能を提供することをコミットメント（確約）するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。