Hewlett Packard
Enterprise

JPOUG
Japan Oracle User Group

Oracle Database 23c 新機能検証

Noriyoshi Shinoda

September 13, 2023

SPEAKER

Featured in
The Redgate 100

- 篠田 典良（しのだ のりよし）
- 所属
  - 日本ヒューレット・パッカード合同会社 HPE Services
  - Oracle ACE Pro (2009年4月～)
- 現在の業務など
  - Oracle Database をはじめ PostgreSQL, Microsoft SQL Server, Vertica 等 RDBMS 全般に関するシステムの設計、移行、チューニング、コンサルティング
  - PostgreSQL 開発 （PostgreSQL 10～17 dev）
- 関連する URL
  - Oracle ACE ってどんな人？
    http://www.oracle.com/technetwork/jp/database/articles/vivadeveloper/index-1838335-ja.html
  - Redgate 100 in 2022 (Most influential in the database community 2022)
    https://www.red-gate.com/hub/redgate-100/
  - 「PostgreSQL 虎の巻」シリーズ
    http://h30507.www3.hp.com/t5/user/viewprofilepage/user-id/838802

SQL Syntax

Good bye FROM DUAL

- FROM 句は省略可能

PostgreSQL　MySQL　SQL Server

- DUAL 不要

```
SQL> SELECT SYSDATE ;
SYSDATE
-----------------
12-SEP-23
```

Good bye FROM DUAL

- 実行計画上は FAST DUAL が残る

```
SQL> SELECT SYSDATE ;
Execution Plan
----------------------------------------------------------

Plan hash value: 1388734953
-------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Cost (%CPU)| Time     |
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     1 |     2   (0)| 00:00:01 |
|   1 |  FAST DUAL        |      |     1 |     2   (0)| 00:00:01 |
-------------------------------------------------------------------------
```

- トレースを取得すると FROM DUAL 句が追加されている
  - ALTER SESSION SET EVENTS '10053 trace name context forever' で取得

```
Final query after transformations:******* UNPARSED QUERY IS *******
SELECT SYSDATE@! "SYSDATE" FROM "SYS"."DUAL" "DUAL"
```

Good bye FROM DUAL

- DUAL 表にどうしてもアクセスしたい場合は
  - 初期化パラメーター _fast_dual_enabled = FALSE を指定（Oracle Database 10*g* 以降）

GROUP BY Using Column Aliases

- GROUP BY に列名エイリアス、列番号を指定可能（≒ORDER BY）

PostgreSQL　MySQL　~~SQL Server~~

- 出力列のエイリアスで GROUP BY/HAVING 指定

```
SQL> SELECT job_id jid, COUNT(*) cnt, MAX(salary) FROM employees WHERE job_id LIKE 'AD%' GROUP BY jid
        HAVING cnt > 3 ;
```

- 出力列リストの番号で GROUP BY 指定

```
SQL> SELECT job_id, COUNT(*) cnt, MAX(salary) FROM employees WHERE job_id LIKE 'AD%' GROUP BY 1
        HAVING cnt > 3 ;
```

- 最終的には従来と同じ構文に書き換え

```
Final query after transformations:******* UNPARSED QUERY IS *******
SELECT "EMPLOYEES"."JOB_ID" "JID",COUNT(*) "CNT",MAX("EMPLOYEES"."SALARY") "MAX(SALARY)" FROM
"SYSTEM"."EMPLOYEES" "EMPLOYEES" WHERE "EMPLOYEES"."JOB_ID" LIKE 'AD%' GROUP BY "EMPLOYEES"."JOB_ID"
HAVING COUNT(*)>3
```

GROUP BY Using Column Aliases

- 列番号の指定はデフォルトではエラーが発生
  - 初期化パラメーター group_by_position_enabled = TRUE に設定する必要がある

```
SQL> SELECT job_id, MAX(salary) FROM employees WHERE job_id LIKE 'AD%' GROUP BY 1 ;
SELECT job_id, MAX(salary) FROM employees WHERE job_id LIKE 'AD%' GROUP BY 1
       *
ERROR at line 1:
ORA-00979: "JOB_ID": must appear in the GROUP BY clause or be used in an aggregate function

SQL> ALTER SESSION SET group_by_position_enabled = TRUE ;
Session altered.

SQL> SELECT job_id, MAX(salary) FROM employees WHERE job_id LIKE 'AD%' GROUP BY 1 ;
...
```

Direct Joins for UPDATE/DELETE

- シンプルな UPDATE/DELETE 文の実行

~~PostgreSQL~~ ~~MySQL~~ SQL Server

- 従来の記述方法

```
SQL> UPDATE (
       SELECT dst1.col2 d2, src1.col2 s2 FROM dst1
       INNER JOIN src1 ON src1.col1 = dst1.col1
     )
     SET d2 = s2 ;
2 rows updated.
```

- 新しい記述方法

```
SQL> UPDATE dst1 SET dst1.col2 = src1.col2 FROM src1 WHERE src1.col1 = dst1.col1 ;
2 rows updated.
```

RETURNING enhancement

- DML の RETURNING 句で、変更前のデータと変更後のデータを参照可能に
- 例

```
SQL> DECLARE
  2     old_c2 data1.c2%TYPE ;
  3     new_c2 data1.c2%TYPE ;
  4  BEGIN
  5     UPDATE data1 SET c2 = 'updated' WHERE c1=100
  6     RETURNING OLD c2, NEW c2 INTO old_c2, new_c2 ;
  7
  8     DBMS_OUTPUT.PUT_LINE('OLD C2: ' || old_c2) ;
  9     DBMS_OUTPUT.PUT_LINE('NEW C2: ' || new_c2) ;
 10  END;
 11  /
OLD C2: original
NEW C2: updated
```

Multi record INSERT

- １回の INSERT 文で複数レコード指定可能

PostgreSQL　MySQL　SQL Server

```
SQL> INSERT INTO data1 VALUES (100, 'data1'),(200, 'data2') ;
2 rows created.
SQL> COMMIT ;
Commit complete.
SQL> INSERT INTO data1 VALUES (300, 'data3'),(100, 'conflict') ;
INSERT INTO data1 VALUES (300, 'data3'),(100, 'conflict')
*
ERROR at line 1:
ORA-00001: unique constraint (SYS.SYS_C008320) violated
SQL> SELECT COUNT(*) FROM data1 ;
  COUNT(*)
----------
         2
```

Multi record INSERT

- 従来は INSERT ALL 文が必要だった

```
INSERT ALL
INTO data1 VALUES (100, 'data1')
INTO data1 VALUES (200, 'data2')
SELECT * FROM DUAL ;
```

- SELECT FROM DUAL UNION ALL 文に変換されている模様

```
SVS: Query after subsumption:******* UNPARSED QUERY IS *******
(SELECT 100 "100",'data1' "'data1'" FROM "SYS"."DUAL" "DUAL") UNION ALL (SELECT 200,'data2' FROM
"SYS"."DUAL" "DUAL")
SU: Considering subquery unnesting in query block INS$1 (#0)
```

Row constructor

- VALUES 句によるレコード生成

PostgreSQL | ~~MySQL~~ | SQL Server

```
SQL> SELECT * FROM ( VALUES (100, 'data1'),(200, 'data2') ) table1 (col1, col2) ;
COL1       COL2
---------- -----
       100 data1
       200 data2

Execution Plan
----------------------------------------------------------------------------
| Id  | Operation         | Name | Rows  | Bytes | Cost (%CPU)| Time     |
----------------------------------------------------------------------------
|   0 | SELECT STATEMENT  |      |     2 |    20 |     4   (0)| 00:00:01 |
|   1 |  VIEW             |      |     2 |    20 |     4   (0)| 00:00:01 |
|   2 |   VALUES SCAN     |      |     2 |       |     4   (0)| 00:00:01 |
----------------------------------------------------------------------------
```

Functions

- New Functions

```
SQL> SELECT CONCAT('Abc', 'Def', 'Ghi') CONCAT ;
CONCAT
---------
AbcDefGhi

SQL> SELECT FUZZY_MATCH(LEVENSHTEIN, 'Clark Kent', 'Claire Kent') ;
FUZZY_MATCH(LEVENSHTEIN,'CLARKKENT','CLAIREKENT')
-------------------------------------------------
                                               82

SQL> SELECT PHONIC_ENCODE(DOUBLE_METAPHONE, 'Knight') ;
PHONIC_ENCODE(DOUBLE_METAPHONE,'KNIGHT')
----------------------------------------
NT
```

PostgreSQL | MySQL | SQL Server

PostgreSQL | ~~MySQL~~ | ~~SQL Server~~

PostgreSQL | ~~MySQL~~ | ~~SQL Server~~

- PostgreSQL は fuzzystrmatch Extension による実装

Object and DDL

IF EXISTS

- CREATE IF NOT EXISTS / DROP IF EXISTS句を使用可能

PostgreSQL　MySQL　~~SQL Server~~

```
SQL> CREATE TABLE IF NOT EXISTS data1(c1 NUMBER, c2 VARCHAR2(10)) ;
Table created.

SQL> DROP TABLE IF EXISTS data1 ;
Table dropped.
```

- 暗黙のコミットはEXISTS句の指定にかかわらず発生する
- 他のRDBMSとの違い
  - PostgreSQLはIF NOT EXISTSで作成されなかった場合に「NOTICE」ログが出力される
    - PostgreSQLはトランザクション実行中にエラーが発生すると強制ROLLBACKされる
  - MySQLはIF NOT EXISTSで作成されなかった場合に「WARNING」が出力される

IF EXISTS

- 従来の OR REPLACE 句も残っているが、同時には使用できない

```
SQL> CREATE OR REPLACE PROCEDURE IF NOT EXISTS remove_emp (employee_id NUMBER) AS
   BEGIN
     ...
   END;
/
CREATE OR REPLACE PROCEDURE IF NOT EXISTS remove_emp(employee_id NUMBER) AS
                                           *
行1でエラーが発生しました。:
ORA-11541: REPLACE and IF NOT EXISTS cannot coexist in the same DDL statement
```

BOOLEAN data type

- テーブル列のデータ型に BOOLEAN 型を使用可能（従来は PL/SQL 内のみ）

PostgreSQL　MySQL　~~SQL Server~~

  - PostgreSQL はネイティブなデータ型、MySQL は tinyint 型のエイリアス

```
SQL> CREATE TABLE bool1 (c1 BOOLEAN, c2 VARCHAR2(10)) ;
Table created.

SQL> INSERT INTO bool1 VALUES (TRUE, 'true') ;
1 row created.

SQL> INSERT INTO bool1 VALUES (FALSE, 'false') ;
1 row created.

SQL> SELECT * FROM bool1 WHERE c1 ;
...
```

BOOLEAN data type

- 暗黙の型変換（SQL Server は bit 型で確認）

入力値	Oracle 23c	PostgreSQL	MySQL	SQL Server	Note
TRUE	TRUE	TRUE	1	Error	
FALSE	FALSE	FALSE	0	Error	
'true'	TRUE	TRUE	Error	Error	't' でも可
'false'	FALSE	FALSE	Error	Error	'f'でも可
'YES'	TRUE	TRUE	Error	Error	'y' でも可
'NO'	FALSE	FALSE	Error	Error	'n' でも可
0	FALSE	Error	0	0	
1	TRUE	Error	1	1	
2	TRUE	Error	2	1	
'0'	FALSE	FALSE	0	0	
'1'	TRUE	TRUE	1	1	
'2'	Error	Error	2	1	

SQL DOMAIN

- SQL DOMAIN とは？ ⇒ カプセル化されたデータ型（日本語マニュアルの表記は「SQL ドメイン」）
  - データ型
  - デフォルト値
  - 制約
  - 表示
  - 順序

- 作成例

```
SQL> CREATE DOMAIN dom_email AS VARCHAR2(100)
         DEFAULT ON NULL 'default@hpe.com'
         CONSTRAINT chk_email CHECK (regexp_like (dom_email, '^(¥S+)¥@(¥S+)¥.(¥S+)$'))
         DISPLAY  LOWER(dom_email)
         ORDER LOWER(dom_email) ;
Domain created.
```

SQL DOMAIN

- テーブルのデータ型として使用可能
- 指定例

```
SQL> CREATE TABLE member (username VARCHAR2(50), email DOMAIN dom_email) ;
Table created.
```

- DOMAIN_DISPLAY 関数、DOMAIN_ORDER 関数を指定することで、ドメイン独自の出力を得られる
- 例

```
SQL> INSERT INTO member VALUES ('SHINODA', 'SHINODA@HPE.COM') ;
1 row created.

SQL> SELECT DOMAIN_DISPLAY(email) FROM member ORDER BY DOMAIN_ORDER(email) ;

DOMAIN_DISPLAY(EMAIL)
--------------------------------------------------------------------------------
shinoda@hpe.com
```

SQL DOMAIN

- 制約
  - COLLATE 設定は、初期化パラメーター max_string_size = 'EXTENDED' が必要
  - ALTER DOMAIN 文では DISPLAY, ORDER, ANNOTATION のみ変更可能
  - 使用している DOMAIN は DROP DOMAIN 文では削除できないが、FORCE オプションを指定することで削除可能
  - PL/SQL のデータ型として使用不可（{テーブル}%ROWTYPE で代替）

```
SQL> DECLARE
  2    email DOMAIN dom_email ;
  3  BEGIN NULL ; END ;
  4  /
  email DOMAIN dom_email ;
                *
ERROR at line 2:
ORA-06550: line 2, column 16:
PLS-00103: Encountered the symbol "DOM_EMAIL" when expecting one of the
following:
```

ANNOTATION

- ANNOTATION とは？
  - COMMENT の発展版（日本語マニュアルの表記は「注釈」）
  - テーブル、ビュー、マテリアライズド・ビュー、それぞれの列、インデックス、ドメインに指定可能
  - ANNOTATION 名と値のセット（名前のみでも可）
  - 複数指定可能
  - ANNOTATION のみ追加・削除・変更可能
- 例

```
CREATE TABLE employees (
  id NUMBER(5)  ANNOTATIONS ("NameOnly", Comment1 'Employee ID'),
  ...
  ) ANNOTATIONS ( Display 'Employee Table' )
```

← NameOnly=名前のみ

← Display=名前、 'Employee Table'=値

ANNOTATION

- ANNOTATION 名
  - テーブル等のオブジェクトと同じ（＝大文字で保存）
  - ダブル・クオーテーションで囲むことで大文字・小文字を区別

```
CREATE TABLE t1 (T NUMBER) ANNOTATIONS(Operations 'Sort', Operations 'Group', Hidden);
```

  - 予約語は使用できない

- 変更例

```
SQL> ALTER TABLE  employees  ANNOTATIONS ( ADD Identity 'Table identity1', DROP Display ) ;
```

  - テーブルの ANNOTATION Identity を追加、Display を削除する

ANNOTATION

- ANNOTATION の確認は USER|DBA|ALL_ANNOTATIONS_USAGE ビューを検索
- 例

```
SQL> SELECT annotation_name, annotation_value FROM user_annotations_usage
       WHERE object_name='EMPLOYEES' ;

ANNOTATION_NAME                  ANNOTATION_VALUE
-------------------------------- --------------------------------
IDENTITY                         Table identity 1
NameOnly
COMMENT1                         Employee ID
```

- USER|DBA|ALL_ANNOTATIONS ビューもあるが、同一名称の ANNOTATION はまとめられている（マニュアルに説明無し）

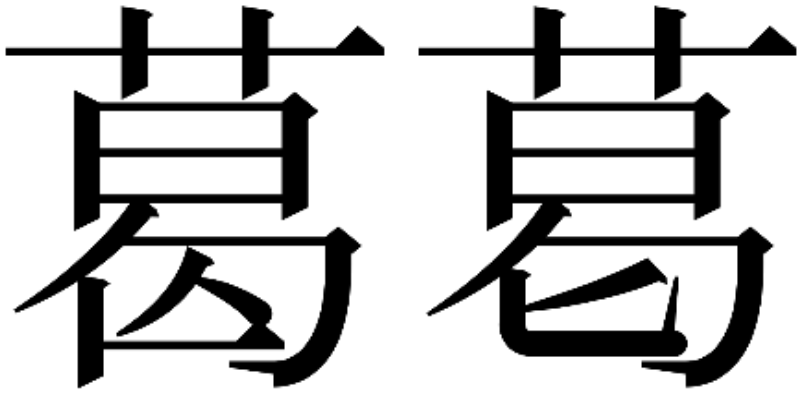
Architecture

UCA1210_JAPANESE_IVS Locale

- 異字体シーケンスのサポート

~~PostgreSQL~~ ~~MySQL~~ SQL Server

  - 例は「東京都葛飾区」と「奈良県葛城市」の字体
  - ほぼ同一字形の文字で同一とみなされることが多い
  - Oracle Database 23*c* では UCA1210_JAPANESE_IVS ロケールをサポート



文字	違い
葛 葛	文字コードは U+845B、異字体セレクタを付加して区別（異字体シーケンス = Ideographic Variation Sequence = IVS）
骨 骨	フォントが異なるだけ（文字コードは U+9AA8）
齋 齊	元々コードが異なる（文字コードは U+9F4B と U+9F4A）

UCA1210_JAPANESE_IVS Locale

- データの準備

```
SQL> CREATE TABLE ivs1 (col1 VARCHAR2(10)) ;
Table created.

SQL> INSERT INTO ivs1 VALUES (UNISTR('¥845B¥DB40¥DD00')) ;
1 row created.

SQL> INSERT INTO ivs1 VALUES (UNISTR('¥845B¥DB40¥DD01')) ;
1 row created.

SQL> ALTER SESSION SET nls_comp=LINGUISTIC ;
Session altered.
```

← 「葛」城市

← 「葛」飾区

← nls_sort により比較を行う

UCA1210_JAPANESE_IVS Locale

- 検索

SQL> ALTER SESSION SET nls_sort=uca1210_japanese ; Session altered. SQL> SELECT COUNT(*) FROM ivs1 WHERE c1 = UNISTR('¥845B¥DB40¥DD00') ; COUNT(*) ---------- 2 SQL> ALTER SESSION SET nls_sort=uca1210_japanese_ivs ; Session altered. SQL> SELECT COUNT(*) FROM ivs1 WHERE c1 = UNISTR('¥845B¥DB40¥DD00') ; COUNT(*) ---------- 1	← Oracle Database 20*c* から ←同一文字とみなされて２件 ← Oracle Database 23*c* から ←文字の区別ができるので１件

UCA1210_JAPANESE_IVS Locale

- 文字列長

```
SQL> SELECT LENGTH(c1), LENGTHB(c1), LENGTH2(c1), LENGTH4(c1), LENGTHC(c1) FROM ivs1 ;

LENGTH(C1) LENGTHB(C1) LENGTH2(C1) LENGTH4(C1) LENGTHC(C1)
---------- ---------- ---------- ---------- ----------
         2          7          3          2          1
```

Max columns

- テーブル列の最大数

PostgreSQL=1,600	MySQL=4,096	SQL Server=1,024

- 初期化パラメーター max_columns を 'EXTENDED' に設定することで拡張可能
  - デフォルト値は 'STANDARD' = 従来通り最大値 1,000

- 条件
  - 初期化パラメーター compatible = '23.0.0.0' 以上
  - 動的変更不可
  - PDB で変更可能

Max columns

- 初期化パラメーター max_string_size と異なり元に戻すことができる
  - 1,000 列を超えるテーブルやビューが無いことが条件

```
SQL> ALTER SYSTEM SET max_columns='STANDARD' SCOPE=spfile ;
ALTER SYSTEM SET max_columns='STANDARD' SCOPE=spfile
*
ERROR at line 1:
ORA-32017: failure in updating SPFILE
ORA-60471: max_columns can not be set to STANDARD as there are one or more
objects with more than 1000 columns
```

- 日本語のエラーメッセージが変
  - 列数が 4,096 を超えた場合のエラー

```
ORA-01792: 表またはビューに指定できる最大列数は1000です。
```

  - 1,000 列を超えるテーブルがあるときに初期化パラメーターを元に戻そうとしてエラー

```
ORA-32017: SPFILEの更新中に障害が発生しました ORA-60471:
1000を超える列を持つ1つ以上のオブジェクトがあるため、_max_column_limitをSTANDARDに設定することはできません
```

SCHEMA LEVEL Privilege

- スキーマ（ユーザー）単位でオブジェクト権限を付与

~~PostgreSQL~~ MySQL ~~SQL Server~~

- テーブルが追加されると自動的に権限が付与される

```
GRANT ANY オブジェクト権限 ON SCHEMA {スキーマ名} TO {付与先}
```

- USER1 ユーザーが USER2 ユーザーに対して全テーブルの SELECT 権限を付与する

```
SQL> GRANT SELECT ANY TABLE ON SCHEMA user1 TO user2 ;
Grant succeeded.

USER2 ユーザーによる確認
SQL> SELECT * FROM USER_SCHEMA_PRIVS ;
```

- 個別のテーブルに対する GRANT / REVOKE よりも優先する

まとめ

まとめ

- 大規模システムへの対応や、管理機能だけでなくアプリケーション用 SQL も充実
- より詳しい Oracle Database 23c の新機能はこちらのセミナーで語られると思います
  2023年9月28日 18:45〜
  https://oracle-code-tokyo-dev.connpass.com/event/295513/

Thank you

Mail: noriyoshi.shinoda@hpe.com
Twitter: @nori_shinoda
Qiita: @plusultra