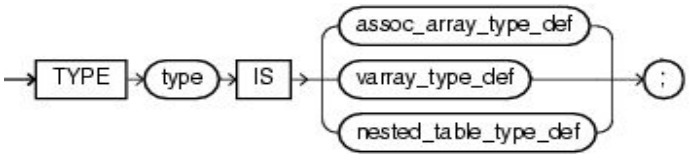
Оглавление

PL/SQL Collections

Общие сведения о коллекциях в pl/sql

- Создание коллекции происходит в два этапа
  1. Определить тип(type) коллекции (конструкции assoc_array_type_def, varray_type_def и nested_table_type_def будут приведены далее)



  2. Создать переменную этого типа

- Обращение к элементу коллекции: variable_name(index)
- Могут принимать значение NULL
- Возможны многомерные коллекции (коллекции коллекций)

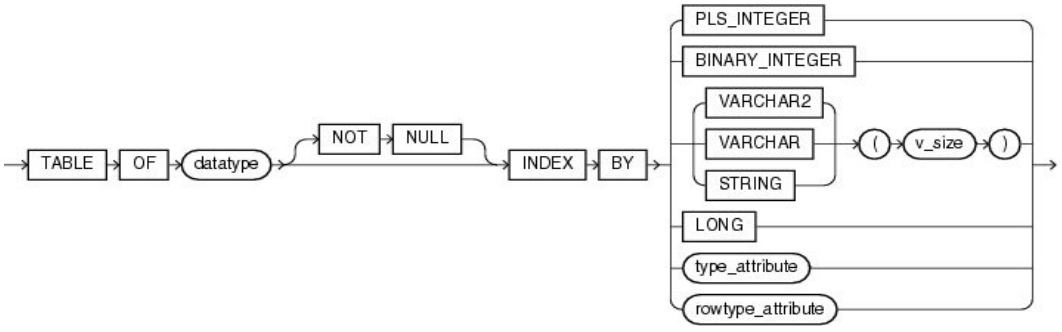
Типы коллекций

Тип коллекции	Количество элементов	Тип индекса	Плотная или разреженная	Без инициализации	Где объявляется	Использование в SQL
Ассоциативный массив (index by table)	Не задано	String Pls_integer	Плотная и разреженная	Empty	PL/SQL block Package	Нет
Varray (variable-size array)	Задано	Integer	Только плотная	Null	PL/SQL block Package Schema level	Только определенные на уровне схемы
Nested table	Не задано	Integer	При создании плотная, может стать разреженной	Null	PL/SQL block Package Schema level	Только определенные на уровне схемы

Ассоциативный массив

Также его называют index by table или pl/sql table.

Тип описывается следующим образом (assoc_array_type_def):



- ✓ Набор пар ключ-значение
- ✓ Данные хранятся в отсортированном по ключу порядке

- ✓ Не поддерживает DML-операции
- ✓ При объявлении как константа должен быть сразу инициализирован функцией
- ✓ Порядок элементов в ассоциативном массиве с строковым индексом зависит от параметров NLS_SORT и NLS_COMP
- ✓ Нельзя объявить тип на уровне схемы, но можно в пакете
- ✓ Не имеет конструктора
- ✓ Индекс не может принимать значение null
- ✓ Datatype – это любой тип данных, кроме ref cursor

Используются для:

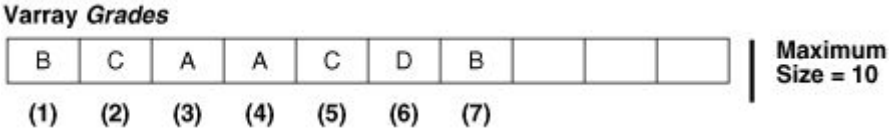
- Для помещения в память небольших таблиц-справочников
- Для передачи в качестве параметра коллекции

Restrictions:

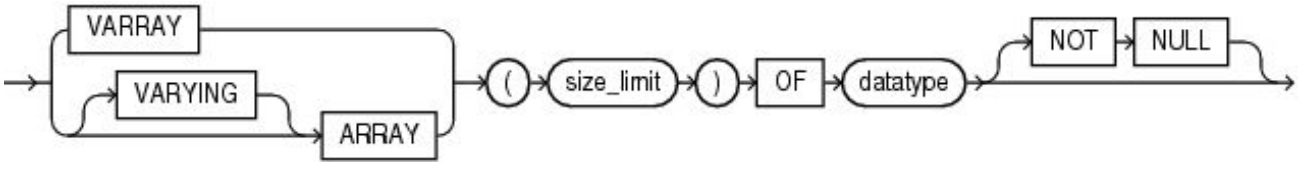
При изменении параметров NLS_SORT и NLS_COMP во время сессии после заполнения ассоциативного массива, можем получать неожиданные результаты вызовов методов first, last, next, previous. Также могут возникнуть проблемы при передаче ассоциативного массива в качестве параметра на другую БД с иными настройками NLS_SORT и NLS_COMP

Varray

Представляет собой массив последовательно хранящихся элементов



Тип описывается следующим образом (varay_type_def):



- ✓ Размер задается при создании

- ✓ Индексируется с 1
- ✓ Инициализируется конструктором

collection_type ([*value* [, *value*]...])

- ✓ Если параметры в конструктор не передаются, возвращается пустая коллекция
- ✓ Datatype – это любой тип данных, кроме ref cursor

Используется, если:

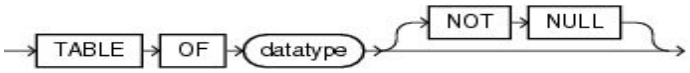
1. Знаем максимально возможное количество элементов
2. Доступ к элементам последовательный

Restrictions:

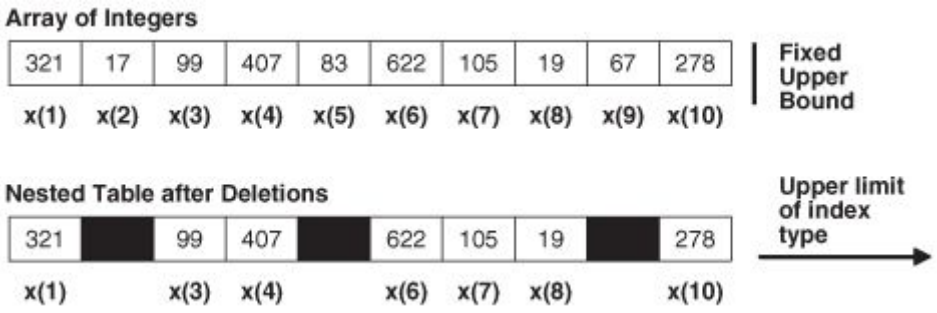
Максимальный размер – 2 147 483 647 элементов

Nested table

Тип описывается следующим образом (nested_table_type_def):



- ✓ Размер коллекции изменяется динамически
- ✓ Может быть в разряженном состоянии, как показано на картинке



- ✓ Инициализируется конструктором

collection_type ([*value* [, *value*]...])

- ✓ Если параметры в конструктор не передаются, возвращается пустая коллекция
- ✓ Datatype – это любой тип данных, кроме ref cursor
- ✓ Если содержит только одно скалярное значение, то имя колонки – Column_Value

✓ Если параметры в конструктор не передаются, возвращается пустая коллекция

```
SELECT column_value
  FROM   TABLE(nested_table)
```

Set operations с nested tables

Операции возможны только с коллекциями **nested table**. Обе коллекции, участвующие в операции, должны быть одного типа.

Результатом операции также является коллекция **nested table**.

Операция	Описание
MULTISET UNION	Возвращает объединение двух коллекций
MULTISET UNION DISTINCT	Возвращает объединение двух коллекций с дистинктом(убирает дубли)
MULTISET INTERSECT	Возвращает пересечение двух коллекций
MULTISET INTERSECT DISTINCT	Возвращает пересечение двух коллекций с дистинктом(убирает дубли)
SET	Возвращает коллекцию с дистинктом(т.е. коллекцию без дублей)
MULTISET EXCEPT	Возвращает разницу двух коллекций
MULTISET EXCEPT DISTINCT	Возвращает разницу двух коллекций с дистинктом (убирает дубли)

Небольшой пример(обратите внимание на результат операции **MULTISET EXCEPT DISTINCT**)

```
DECLARE
    TYPE nested_typ IS TABLE OF NUMBER;
    nt1 nested_typ := nested_typ(1,2,3);
    nt2 nested_typ := nested_typ(3,2,1);
    nt3 nested_typ := nested_typ(2,3,1,3);
    nt4 nested_typ := nested_typ(1,2,4);
    answer nested_typ;
BEGIN
    answer := nt1 MULTISET UNION nt4;
    answer := nt1 MULTISET UNION nt3;
    answer := nt1 MULTISET UNION DISTINCT nt3;
    answer := nt2 MULTISET INTERSECT nt3;
    answer := nt2 MULTISET INTERSECT DISTINCT nt3;
    answer := SET(nt3);
    answer := nt3 MULTISET EXCEPT nt2;
    answer := nt3 MULTISET EXCEPT DISTINCT nt2;
END;
```

Результат:

```
nt1 MULTISET UNION nt4: 1 2 3 1 2 4
nt1 MULTISET UNION nt3: 1 2 3 2 3 1 3
nt1 MULTISET UNION DISTINCT nt3: 1 2 3
nt2 MULTISET INTERSECT nt3: 3 2 1
nt2 MULTISET INTERSECT DISTINCT nt3: 3 2 1
SET(nt3): 2 3 1
nt3 MULTISET EXCEPT nt2: 3
nt3 MULTISET EXCEPT DISTINCT nt2: empty set
```

Логические операции с коллекциями

Операция	Описание
IS NULL (IS NOT NULL)	Сравнивает коллекцию со значением NULL
Сравнение =	Две коллекции nested table можно сравнить, если они одного типа и не содержат записей типа record. Они равны, если имеют одинаковые наборы элементов (не зависимо от порядка хранения элементов внутри коллекции
IN	Сравнивает коллекцию с перечисленными в скобках
SUBMULTISET OF	Проверяет, является ли коллекция подмножеством другой коллекции
MEMBER OF	Проверяет, является ли конкретный элемент (объект) частью коллекции
IS A SET	Проверяет, содержит ли коллекция дубли
IS EMPTY	Проверяет, пуста ли коллекция

Небольшой пример использования логический операций с коллекциями:

```
DECLARE
    TYPE nested_typ IS TABLE OF NUMBER;
    nt1 nested_typ := nested_typ(1, 2, 3);
    nt2 nested_typ := nested_typ(3, 2, 1);
    nt3 nested_typ := nested_typ(2, 3, 1, 3);
    nt4 nested_typ := nested_typ();
BEGIN
    IF nt1 = nt2 THEN
        DBMS_OUTPUT.PUT_LINE('nt1 = nt2');
    END IF;

    IF (nt1 IN (nt2, nt3, nt4)) THEN
        DBMS_OUTPUT.PUT_LINE('nt1 IN (nt2,nt3,nt4)');
    END IF;

    IF (nt1 SUBMULTISET OF nt3) THEN
        DBMS_OUTPUT.PUT_LINE('nt1 SUBMULTISET OF nt3');
    END IF;

    IF (3 MEMBER OF nt3) THEN
        DBMS_OUTPUT.PUT_LINE('3 MEMBER OF nt3');
    END IF;
```

```
    IF (nt3 IS NOT A SET) THEN
        DBMS_OUTPUT.PUT_LINE('nt3 IS NOT A SET');
    END IF;

    IF (nt4 IS EMPTY) THEN
        DBMS_OUTPUT.PUT_LINE('nt4 IS EMPTY');
    END IF;
END;
```

Результат:

nt1 = nt2
nt1 IN (nt2,nt3,nt4)
nt1 SUBMULTISET OF nt3
3 MEMBER OF nt3
nt3 IS NOT A SET
nt4 IS EMPTY

Методы коллекций

Синтаксис вызова методов: ***collection_name.method***

Метод	Тип	Описание	Index by table	Varray	Nested table
DELETE	Процедура	Удаляет элементы из коллекции	Да	Нет	Да
TRIM	Процедура	Удаляет элементы с конца коллекции (работает с внутренним размером коллекции)	Нет	Да	Да
EXTEND	Процедура	Добавляет элементы в конец коллекции	Нет	Да	Да
EXISTS	Функция	Возвращает TRUE, если элемент присутствует в коллекции	Да	Да	Да
FIRST	Функция	Возвращает первый индекс коллекции	Да	Да	Да
LAST	Функция	Возвращает последний индекс коллекции	Да	Да	Да
COUNT	Функция	Возвращает количество элементов в коллекции	Да	Да	Да
LIMIT	Функция	Возвращает максимальное количество элементов, которые может хранить коллекция	Нет	Да	Нет
PRIOR	Функция	Возвращает индекс предыдущего элемента коллекции	Да	Да	Да
NEXT	Функция	Возвращает индекс следующего элемента коллекции	Да	Да	Да

Delete

- ✓ Delete удаляет все элементы. Сразу же очищает память, выделенную для хранения этих элементов.
- ✓ Delete(n) удаляет элемент с индексом n. Память не освобождает. Элемент можно восстановить (т.е. задать

новый) и он займет туже память, что занимал предыдущий.

- ✓ Delete(n, m) удаляет элементы с индексами в промежутке n..m
- ✓ Если удаляемого элемента в коллекции нет, ничего не делает.

Пример использования:

```
DECLARE
    TYPE nt_type IS TABLE OF NUMBER;
    nt nt_type := nt_type(11, 22, 33, 44, 55, 66);
BEGIN
    nt.DELETE(2); -- Удаляет второй элемент
    nt(2) := 2222; -- Восстанавливает 2-й элемент
    nt.DELETE(2, 4); -- Удаляет элементы со 2-го по 4-й
    nt(3) := 3333; -- Восстанавливает 3-й элемент
    nt.DELETE; -- Удаляет все элементы
END;
```

Результаты:

beginning: 11 22 33 44 55 66
after delete(2): 11 33 44 55 66
after nt(2) := 2222: 11 2222 33 44 55 66
after delete(2, 4): 11 55 66
after nt(3) := 3333: 11 3333 55 66
after delete: : empty set

Trim

- ✓ Trim() – удаляет один элемент в конце коллекции. Если элемента нет, генерирует исключение SUBSCRIPT_BEYOND_COUNT
- ✓ Trim(n) – удаляет n элементов в конце коллекции. Если элементов меньше, чем n, генерируется исключение SUBSCRIPT_BEYOND_COUNT
- ✓ Работает с внутренним размером коллекции. Т.е. если последний элемент был удален с помощью Delete, вызов Trim() удалит уже удаленный ранее элемент.
- ✓ Сразу очищает память, выделенную для хранения этих элементов
- ✓ Лучше не использовать в сочетании с Delete()

Пример использования:

```
DECLARE
    TYPE nt_type IS TABLE OF NUMBER;
    nt nt_type := nt_type(11, 22, 33, 44, 55, 66);
BEGIN
    nt.TRIM; -- Trim last element
    nt.DELETE(4); -- Delete fourth element
    nt.TRIM(2); -- Trim last two elements
END;
```

Результат:

beginning: 11 22 33 44 55 66
after TRIM: 11 22 33 44 55
after DELETE(4): 11 22 33 55
after TRIM(2): 11 22 33

Extend

- ✓ EXTEND добавляет один элемент со значением null в конец коллекции
- ✓ EXTEND(*n*) добавляет n элементов со значением null в конец коллекции
- ✓ EXTEND(*n*,*i*) добавляет n копий элемента с индексом i в конец коллекции. Если коллекция имеет NOT NULL констрейнт, только этой формой можно пользоваться.
- ✓ Если элементы были ранее удалены с помощью метода Delete, Extend не будет использовать сохранившиеся за коллекцией ячейки памяти, а добавит новый элемент (выделит новую память)

Пример использования:

```
DECLARE
    TYPE nt_type IS TABLE OF NUMBER;
    nt nt_type := nt_type(11, 22, 33);
BEGIN
    nt.EXTEND(2, 1); -- Append two copies of first element
    nt.DELETE(5); -- Delete fifth element
    nt.EXTEND; -- Append one null element
END;
```

Результат:

```
beginning: 11 22 33
after EXTEND(2,1): 11 22 33 11 11
after DELETE(5): 11 22 33 11
after EXTEND: 11 22 33 11
```

Exists

- ✓ Для удаленных элементов возвращает false
- ✓ При выходе за границы коллекции возвращает false

Пример использования:

```
DECLARE
    TYPE NumList IS TABLE OF INTEGER;
    n NumList := NumList(1, 3, 5, 7);
BEGIN
    n.DELETE(2); -- Delete second element
    FOR i IN 1 .. 6
    LOOP
        IF n.EXISTS(i)
        THEN
            DBMS_OUTPUT.PUT_LINE('n(`||i||') = ' || n(i));
        ELSE
            DBMS_OUTPUT.PUT_LINE('n(`||i||') does not exist');
        END IF;
    END LOOP;
END;
```

First и Last

- ✓ Для varray First всегда возвращает единицу, Last всегда возвращает то же значение, что и Count

Пример использования:

```
DECLARE
    TYPE aa_type_str IS TABLE OF INTEGER INDEX BY VARCHAR2(10);
    aa_str aa_type_str;
BEGIN
    aa_str('Z') := 26;
    aa_str('A') := 1;
    aa_str('K') := 11;
    aa_str('R') := 18;

    DBMS_OUTPUT.PUT_LINE('Before deletions:');
    DBMS_OUTPUT.PUT_LINE('FIRST = ' || aa_str.FIRST);
    DBMS_OUTPUT.PUT_LINE('LAST = ' || aa_str.LAST);

    aa_str.DELETE('A');
    aa_str.DELETE('Z');

    DBMS_OUTPUT.PUT_LINE('After deletions:');
    DBMS_OUTPUT.PUT_LINE('FIRST = ' || aa_str.FIRST);
    DBMS_OUTPUT.PUT_LINE('LAST = ' || aa_str.LAST);
END;
```

Результат:

Before deletions:
FIRST = A
LAST = Z
After deletions:
FIRST = K
LAST = R

Count

Пример использования:

```
DECLARE
    TYPE NumList IS VARRAY(10) OF INTEGER;
    n NumList := NumList(1, 3, 5, 7);

BEGIN
    DBMS_OUTPUT.PUT('n.COUNT = ' || n.COUNT || ', ');
    DBMS_OUTPUT.PUT_LINE('n.LAST = ' || n.LAST);

    n.EXTEND(3);
    DBMS_OUTPUT.PUT('n.COUNT = ' || n.COUNT || ', ');
    DBMS_OUTPUT.PUT_LINE('n.LAST = ' || n.LAST);

    n.TRIM(5);
    DBMS_OUTPUT.PUT('n.COUNT = ' || n.COUNT || ', ');
    DBMS_OUTPUT.PUT_LINE('n.LAST = ' || n.LAST);
END;
```

Результат:

n.COUNT = 4, n.LAST = 4
n.COUNT = 7, n.LAST = 7

n.COUNT = 2, n.LAST = 2

Limit

- ✓ Для varray возвращает максимально допустимое количество элементов в коллекции, для остальных коллекций возвращает null

Пример использования:

```
DECLARE
    TYPE aa_type IS TABLE OF INTEGER INDEX BY PLS_INTEGER;
    aa aa_type; -- associative array

    TYPE va_type IS VARRAY(4) OF INTEGER;
    va va_type := va_type(2, 4); -- varray

    TYPE nt_type IS TABLE OF INTEGER;
    nt nt_type := nt_type(1, 3, 5); -- nested table

BEGIN
    aa(1) := 3;
    aa(2) := 6;
    aa(3) := 9;
    aa(4) := 12;
    DBMS_OUTPUT.PUT_LINE('aa.COUNT = ' || aa.count);
    DBMS_OUTPUT.PUT_LINE('aa.LIMIT = ' || aa.limit);

    DBMS_OUTPUT.PUT_LINE('va.COUNT = ' || va.count);
    DBMS_OUTPUT.PUT_LINE('va.LIMIT = ' || va.limit);

    DBMS_OUTPUT.PUT_LINE('nt.COUNT = ' || nt.count);
    DBMS_OUTPUT.PUT_LINE('nt.LIMIT = ' || nt.limit);
END;
```

Результат:

aa.COUNT = 4
aa.LIMIT =
va.COUNT = 2
va.LIMIT = 4
nt.COUNT = 3
nt.LIMIT =

Prior и Next

- ✓ Позволяют перемещаться по коллекции
- ✓ Возвращают индекс предыдущего/следующего элемента (или null, если элемента нет)

Пример использования:

```
DECLARE
    TYPE nt_type IS TABLE OF NUMBER;
    nt nt_type := nt_type(18, NULL, 36, 45, 54, 63);

BEGIN
    nt.DELETE(4);
    DBMS_OUTPUT.PUT_LINE('nt(4) was deleted.');
```

```
    FOR i IN 1 .. 7
    LOOP
        DBMS_OUTPUT.PUT('nt.PRIOR(' || i || ') = ');
        print(nt.PRIOR(i));
        DBMS_OUTPUT.PUT('nt.NEXT(' || i || ')  = ');
        print(nt.NEXT(i));
    END LOOP;
END;
```

Результат:

```
nt(4) was deleted.
nt.PRIOR(1) =
nt.NEXT(1)  = 2
nt.PRIOR(2) = 1
nt.NEXT(2)  = 3
nt.PRIOR(3) = 2
nt.NEXT(3)  = 5
nt.PRIOR(4) = 3
nt.NEXT(4)  = 5
nt.PRIOR(5) = 3
nt.NEXT(5)  = 6
nt.PRIOR(6) = 5
nt.NEXT(6)  =
nt.PRIOR(7) = 6
nt.NEXT(7)  =
```

Bulk Collect

- ✓ Возвращает результаты sql-оператора в PL/SQL пачками, а не по одному
- ✓ SELECT BULK COLLECT INTO
- ✓ FETCH BULK COLLECT INTO [LIMIT]
- ✓ RETURNING BULK COLLECT INTO
- ✓ Не работает с ассоциативными массивами (кроме тех, что индексированы pls_integer)

Пример использования:

```
DECLARE
    TYPE NumTab IS TABLE OF employees.employee_id%TYPE;
    TYPE NameTab IS TABLE OF employees.last_name%TYPE;

    CURSOR c1 IS SELECT employee_id,last_name
        FROM   employees
        WHERE  salary > 10000
        ORDER  BY last_name;

    enums NumTab;
    names NameTab;
BEGIN
    SELECT employee_id, last_name
    BULK COLLECT INTO   enums, names
    FROM   employees
    ORDER  BY employee_id;
```

```
    OPEN c1;
    LOOP
        FETCH c1 BULK COLLECT INTO enums, names LIMIT 10;
        EXIT WHEN names.COUNT = 0;
        do_something();
    END LOOP;
    CLOSE c1;

    DELETE FROM emp_temp WHERE  department_id = 30
    RETURNING employee_id, last_name BULK COLLECT INTO enums, names;
END;
```

Цикл forall

- ✓ посылает DML операторы из PL/SQL в SQL пачками, а не по одному
- ✓ может содержать только один DML оператор
- ✓ для разряженных коллекций используется форма:

FORALL i IN INDICES OF cust_tab (конструкция не работает для ассоциативных массивов, индексированных строками)

- ✓ с разряженными коллекциями (или с частью коллекции) удобно работать с помощью индекс-коллекций (of pls_integer). Пример использования:

FORALL i IN VALUES OF rejected_order_tab

- **SQL%BULK_ROWCOUNT** – коллекция, содержит количество строк, на которые повлиял каждый dml оператор
- **SQL%ROWCOUNT** – общее количество строк, на которые повлияли dml-операторы в цикле forall

Пример использования:

```
DECLARE
    TYPE NumList IS TABLE OF NUMBER;
    depts NumList := NumList(10, 20, 30);

    TYPE enum_t IS TABLE OF employees.employee_id%TYPE;
    e_ids enum_t;

    TYPE dept_t IS TABLE OF employees.department_id%TYPE;
    d_ids dept_t;
BEGIN
    FORALL j IN depts.FIRST .. depts.LAST
        DELETE FROM emp_temp
        WHERE  department_id = depts(j)
        RETURNING employee_id, department_id BULK COLLECT INTO e_ids, d_ids;
END;
```

Exceptions in forall

- ✓ При возникновении исключения в любом из dml-операторов в цикле, транзакция полностью откатывается
- ✓ Если описать обработчик ошибок, в нем можно зафиксировать успешно выполнившиеся операторы dml (это те операторы, которые выполнились до возникновения исключения).
- ✓ FORALL j IN collection.FIRST.. collection.LAST **SAVE EXCEPTIONS**

  Генерит ORA-24381 в конце, если в цикле возникали исключения

- ✓ SQL%BULK_EXCEPTIONS – коллекция, содержащая информацию о всех сгенерированных во время выполнения цикла исключениях

  .Count

  .ERROR_INDEX – значение индекса j, при котором произошло исключение (sql%bulk_exception(i).error_index)

  .ERROR_CODE - код возникшей ошибки. Информацию об ошибке можно извлечь с помощью функции sqlerrm:
SQLERRM(-(SQL%BULK_EXCEPTIONS(i).ERROR_CODE))

Collection exceptions

COLLECTION_IS_NULL – попытка работать с неинициализированной коллекцией
NO_DATA_FOUND – попытка прочитать удаленный элемент
SUBSCRIPT_BEYOND_COUNT – выход за границы коллекции
SUBSCRIPT_OUTSIDE_LIMIT – индекс вне предела допустимого диапазона
VALUE_ERROR – индекс равен null или не конвертируется в integer

Пример ситуаций, генерирующих исключения:

```
DECLARE
    TYPE NumList IS TABLE OF NUMBER;
    nums NumList;
BEGIN
    nums(1) := 1; -- raises COLLECTION_IS_NULL
    nums := NumList(1, 2);
    nums(NULL) := 3; -- raises VALUE_ERROR
    nums(0) := 3; -- raises SUBSCRIPT_BEYOND_COUNT
    nums(3) := 3; --raises SUBSCRIPT_OUTSIDE_LIMIT
    nums.Delete(1);
    IF nums(1) = 1 THEN ... -- raises NO_DATA_FOUND
END;
```

DBMS_SESSION.FREE_UNUSED_USER_MEMORY

- ✓ Процедура DBMS_SESSION.FREE_UNUSED_USER_MEMORY возвращает неиспользуемую более память системе

- ✓ В документации Oracle процедуру советуют использовать «редко и благоразумно».
- ✓ В случае подключения в режиме **Dedicated Server** вызов этой процедуры возвращает неиспользуемую PGA память операционной системе
- ✓ В случае подключения в режиме **Shared Server** вызов этой процедуры возвращает неиспользуемую память в **Shared Pool**

В каких случаях нужно освобождать память:

- Большие сортировка, когда используется вся область sort_area_size
- Компиляция больших PL/SQL пакетов, процедур или функций
- Хранение больших объемов данных в индексных таблицах PL/SQL

Пример использования:

```
CREATE PACKAGE foobar
   type number_idx_tbl is table of number indexed by binary_integer;

   store1_table  number_idx_tbl;      --  PL/SQL indexed table
   store2_table  number_idx_tbl;      --  PL/SQL indexed table
   store3_table  number_idx_tbl;      --  PL/SQL indexed table
   ...
END;             --  end of foobar

DECLARE
   ...
   empty_table   number_idx_tbl;      --  uninitialized ("empty") version
BEGIN
   FOR i in 1..1000000 loop
     store1_table(i) := i;            --  load data
   END LOOP;
   ...
   store1_table := empty_table;       --  "truncate" the indexed table
   ...
   -
   dbms_session.free_unused_user_memory;  -- give memory back to system

   store1_table(1) := 100;            --  index tables still declared;
   store2_table(2) := 200;            --  but truncated.
   ...
END;
```