# Типы

**create or replace type TOperationAction force as object** Under TOperation

*/\* если есть Under – то наследуется от типа TOperation. Тогда в типе будут переопределенные функции - Overriding Member Function \*/*

(

*/\* переменные - могут начинаться с буквы f \*/*

fActionId varchar2(32),

…

*/\*конструкторы\*/*

constructor function TOperationAction

(

*/\* входные параметры конструктора - могут начинаться с буквы p \*/*

pOperationId in varchar2,

…

) return self as result,

constructor function TOperationAction(… ) return self as result, *--конструктор*

Member Function StepDecodeBarcode(Self In Out Nocopy TOperationPay) Return TOperationMeta, *–-простая функция*

Overriding Member Function doStartOperation(Self In Out Nocopy TOperationPay,

pSessionId In Number, pParameter In Varchar2)

Return TOperationMeta *–-переопределенная функция*

)

*/\*тело типа\*/*

**create or replace type body TOperationAction is**

constructor function TOperationAction(

*/\*тут тоже входные параметры конструктора\*/*

) return self as result

is

begin

*/\*логика. Например можно определить переменные в зависимости от входящих параметров\*/*

return;

end;

constructor function TOperationAction(…) return self as result

is

begin

*/\*логика\*/*

return;

end;

end;

Наследование

**create or replace type TOperation force as object**

(

*/\* force – пересоздает тип даже когда у него есть дочерние или он есть в таблицах\*/*

) not final not instantiable

*/\* not final – используется как родитель*

*not instantiable – нет конструктора и нельзя создать объект \*/*

ПРИМЕР:

*create or replace type TCURRENCY force as object*

*(*

*fId VARCHAR2(3),*

*fName VARCHAR2(40),*

*fDisplayName VARCHAR2(20),*

*constructor function TCurrency( pCode in varchar2 ) return self as result*

*)*

*create or replace type body TCurrency is*

*constructor function TCurrency( pCode in varchar2 ) return self as result*

*is*

*begin*

*fId := case when pCode is null then 'RUB' else replace(pCode, 'RUR', 'RUB') end;*

*select t.title, nvl(t.label,t.symbolic\_code)*

*into fName, fDisplayName*

*from CURRENCY t*

*where*

*t.symbolic\_code = fId;*

*if pCode = 'A98' then fId := 'GLD';*

*end if;*

*if pCode = 'A99' then fId := 'SLV';*

*end if;*

*return;*

*end;*

*end;*

# Пакеты

**create or replace package MB\_PAYMENTS is**

type rc is ref cursor;

CHANNEL varchar2(2) := 'MB';

function getTpS(pSessionId in number, pSearchString in varchar2) return TServices;

end MB\_PAYMENTS;

**create or replace package body MB\_PAYMENTS is**

function getTpS (pSessionId in number, pSearchString in varchar2) return TServices

is

type vRcType is ref cursor;

vCityList vRcType;

begin

end;

procedure LoginToken(…)is .. begin…end;

Begin

*/\*код пакета (Code section) – его может и не быть\*/*

End MB\_PAYMENTS;

# Функции

*/\* функция в спецификации типа.*

*not instantiable – значит что функция не имеет реализации в теле типа \*/*

not instantiable member function doStartOperation(

self in out nocopy TOperation,

pParameter in varchar2

) return TOperationMeta

# Триггеры

**ВСЕ о Триггерах** <https://habr.com/post/256655/>

create or replace trigger MB\_SESSION\_TRG\_IU

before insert or update of last\_used\_date on mb\_session

REFERENCING NEW AS Employee OLD AS Department --опционально. Переопределяем :new и :old

for each row

declare

vSessionIdleTimeout number;

begin

vSessionIdleTimeout := to\_number( nvl(STDLIB.GetConfigParam('SESSION\_IDLE\_TIMEOUT'), 30) );

:new.expires\_on := sysdate + NumToDSInterval(vSessionIdleTimeout, 'MINUTE');

end MB\_SESSION;

#### Instead of dml triggers

* Создаются для представлений (view) и служат для замещения DML операций своим функционалом.
* Позволяют производить операции вставки/обновления или удаления для не обновляемых представлений.

CREATE OR REPLACE VIEW order\_info AS

SELECT c.customer\_id, c.cust\_last\_name, c.cust\_first\_name,

o.order\_id, o.order\_date, o.order\_status

FROM customers c, orders o

WHERE c.customer\_id = o.customer\_id;

CREATE OR REPLACE TRIGGER order\_info\_insert

INSTEAD OF INSERT ON order\_info

DECLARE

duplicate\_info EXCEPTION;

PRAGMA EXCEPTION\_INIT (duplicate\_info, -00001);

BEGIN

INSERT INTO customers

(customer\_id, cust\_last\_name, cust\_first\_name)

VALUES (

:new.customer\_id,

:new.cust\_last\_name,

:new.cust\_first\_name);

INSERT INTO orders (order\_id, order\_date, customer\_id)

VALUES (

:new.order\_id,

:new.order\_date,

:new.customer\_id);

EXCEPTION

WHEN duplicate\_info THEN

RAISE\_APPLICATION\_ERROR (

num=> -20107,

msg=> 'Duplicate customer or order ID');

END order\_info\_insert;

#### Instead of triggers on Nested Table Columns of Views

Можно создать триггер для вложенной в представлении таблицы. В таком триггере также присутствует дополнительная псевдозапись – parent, которая ссылается на всю запись представления (стандартные псевдозаписи old и new ссылаются только на записи вложенной таблицы)

Источник <https://habr.com/post/256655/>

#### Составные DML триггера (compound DML triggers)

create or replace trigger tr\_table\_test\_compound

for update or delete or insert on table\_test

compound trigger

v\_count pls\_integer := 0;

before statement is

begin

dbms\_output.put\_line ( 'before statement' );

end before statement;

before each row is

begin

dbms\_output.put\_line ( 'before insert' );

end before each row;

after each row is

begin

dbms\_output.put\_line ( 'after insert' );

v\_count := v\_count + 1;

end after each row;

after statement is

begin

dbms\_output.put\_line ( 'after statement' );

end after statement;

end tr\_table\_test\_compound;

#### Триггеры уровня схемы и БД

# Исключения

[Источник](https://oracle-patches.com/%D0%B1%D0%BB%D0%BE%D0%B3%D0%B8/91-sql-pl/3178-%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0-%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9-%D0%B8-%D0%BF%D0%BE%D0%B8%D1%81%D0%BA-%D0%BE%D1%88%D0%B8%D0%B1%D0%BE%D0%BA-%D0%BA%D0%BE%D0%B4%D0%B5-pl-sql) (https://oracle-patches.com/%D0%B1%D0%BB%D0%BE%D0%B3%D0%B8/91-sql-pl/3178-%D0%BE%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0-%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9-%D0%B8-%D0%BF%D0%BE%D0%B8%D1%81%D0%BA-%D0%BE%D1%88%D0%B8%D0%B1%D0%BE%D0%BA-%D0%BA%D0%BE%D0%B4%D0%B5-pl-sql)

#### Встроенные функции ошибок

Функция SQLCODE возвращает код ошибки последнего исключения, инициированного в блоке. При отсутствии ошибок SQLCODE возвращает 0. Кроме того, SQLCODE возвращает 0 при вызове за пределами обработчика исключений.

Функция SQLERRM возвращает сообщение об ошибке для заданного кода ошибки. Если вызвать SQLERRM без указания кода ошибки, функция вернет сообщение, связанное со значением, возвращаемым SQLCODE.

#### Raise

Передает исключение во внешнюю функцию. Используется для формирования стека:

CREATE OR REPLACE PROCEDURE proc1 IS

BEGIN

DBMS\_OUTPUT.put\_line ('выполнение proc1');

RAISE NO\_DATA\_FOUND;

END;

/

CREATE OR REPLACE PROCEDURE proc2 IS

l\_str VARCHAR2 (30) := 'вызов proc1';

BEGIN

DBMS\_OUTPUT.put\_line (l\_str);

proc1;

END;

/

CREATE OR REPLACE PROCEDURE proc3 IS

BEGIN

DBMS\_OUTPUT.put\_line ('вызов proc2');

proc2;

EXCEPTION

WHEN OTHERS

THEN

DBMS\_OUTPUT.put\_line ('Стек ошибок верхнего уровня:');

DBMS\_OUTPUT.put\_line (**DBMS\_UTILITY.format\_error\_backtrace**);

END;

Функция DBMS\_UTILITY.FORMAT\_ERROR\_BACKTRACE выдает трассировку исполнения вплоть до последней команды RAISE в сеансе пользователя.

*SQL> SET SERVEROUTPUT ON*

*SQL> BEGIN*

*2 DBMS\_OUTPUT.put\_line ('Proc3 -> Proc2 -> Proc1 backtrace');*

*3 proc3;*

*4 END;*

*5 /*

*Proc3 -> Proc2 -> Proc1 backtrace*

*вызов proc2*

*вызов proc1*

*выполнение proc1*

*Error stack at top level:*

*ORA-06512: at "SCOTT.PROC1", line 4*

*ORA-06512: at "SCOTT.PROC2", line 5*

*ORA-06512: at "SCOTT.PROC3", line 4*

#### Инициализация пользователем

PROCEDURE delete\_company (company\_id\_in IN NUMBER)

IS

/\* Объявление исключения. \*/

still\_have\_employees EXCEPTION;

/\* Имя исключения связывается с номером ошибки. \*/

PRAGMA EXCEPTION\_INIT (still\_have\_employees, 2292);

BEGIN

/\* Попытка удаления информации о компании. \*/

DELETE FROM company

WHERE company\_id = company\_id\_in;

EXCEPTION

/\* При обнаружении дочерних записей инициируется это исключение! \*/

WHEN still\_have\_employees

THEN

DBMS\_OUTPUT.PUT\_LINE

('Пожалуйста, сначала удалите данные о служащих компании.');

END;

# Последовательности (sequens)

Синтаксис

CREATE SEQUENCE sequence\_name  
MINVALUE value  
MAXVALUE value  
START WITH value  
INCREMENT BY value  
CACHE value;

Псевдостолбец currval дает текущее значение последовательности, а nextval — новое, или следующее, значение последовательности.

Обычный способ применения последовательности для генерации уникального первичного ключа для столбца.

CREATE SEQUENCE employee\_seq

START WITH 10000

INCREMENT BY 1

*--MINVALUE 1*

NOMAXVALUE

*--MAXVALUE 999999999999999999999999999*

*--CACHE 20*

NOCYCLE;

Просмотреть текущий номер последовательности:

SELECT employee\_seq.currval FROM dual;

Вставить новую строку в таблицу employee, используя nextval из последовательности employee\_seq:

INSERT INTO employees(employee\_id, first\_name, last\_name, email,

phone\_number, hire\_date)

VALUES

(employee\_seq.nextval,'sam','alapati','salapati.tnt.org'

,345-555-5555,to\_char('21-JUN-2005');

COMMIT;

# Ограничения constraints

https://oracle-patches.com/%D0%B1%D0%BB%D0%BE%D0%B3%D0%B8/70-%D0%B1%D0%B0%D0%B7%D1%8B-%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85-oracle/3014-%D0%BE%D0%B3%D1%80%D0%B0%D0%BD%D0%B8%D1%87%D0%B5%D0%BD%D0%B8%D1%8F-oracle

#### Ограничения первичного ключа

Таблица может иметь только один первичный ключ, который можно создать при создании таблицы:

CREATE TABLE dept

(dept\_id number(9) PRIMARY KEY);

Или добавить ограничение и к существующей таблице:

ALTER TABLE dept ADD PRIMARY KEY(dept\_id);

Поскольку в предыдущем примере ограничению не присвоено имя, Oracle даст ему имя, сгенерированное системой. Если требуется присвоить ограничению собственное имя, воспользуйтесь следующей командой, которая назовет ограничение dept\_pk:

ALTER TABLE dept ADD CONSTRAINT dept\_pk PRIMARY KEY(dept\_id);

Если первичный ключ будет построен по более чем одному столбцу (т.е. будет составным), писать назначение первичного ключа напротив имени столбца при создании таблицы нельзя. Столбцы первичного ключа должны быть указаны как отдельный элемент оператора CREATE TABLE после списка всех столбцов.

#### Ограничения NOT NULL

CREATE TABLE employee

(last\_name VARCHAR(30) NOT NULL);

Если таблица уже создана:

ALTER TABLE employee MODIFY last\_name NOT NULL;

#### Проверочные ограничения

Для обеспечения соответствия данных столбца определенным параметрам. Например, годовая зарплата сотрудника фирмы не может быть равна или превышать $100 000 при определенных обстоятельствах. Это ограничение можно навязать с помощью следующего оператора, который устанавливает ограничение CHECK на столбце SALARY:

CREATE TABLE employee

(employee\_id NUMBER,

last\_name VARCHAR2(30),

first\_name VARCHAR2(30),

department\_id NUMBER,

salary NUMBER CHECK(salary < 100000));

#### Ограничение уникальности

В таблице могут существовать сразу несколько таких ограничений. В следующем примере первый оператор специфицирует ограничение уникальности на комбинации столбцов dept\_name и location:

CREATE TABLE dept(

dept\_no NUMBER(3),

dept\_name VARCHAR2(15),

location VARCHAR2(25),

CONSTRAINT dept\_name\_ukey UNIQUE(dept\_Name,location);

или:

ALTER TABLE dept ADD CONSTRAINT dept\_idx UNIQUE(dept\_no);

#### Ограничения ссылочной целостности

гарантируют, что значения определенных важных столбцов будут иметь смысл. Предположим, что есть родительская таблица, которая ссылается на значения из другой таблицы, как в случае таблиц dept и employee.Сотрудник в таблице employee не может быть назначен в департамент, если такой департамент в таблице dept не существует.

Гарантировать существование действительного департамента можно с помощью ограничения ссылочной целостности. В этом случае столбец departament\_id служит первичным ключом таблицы dept, а столбец dept\_id в таблице employee, ссылающийся на соответствующий столбец таблицы department, называется внешним ключом. Таблица,содержащая внешний ключ, обычно называется дочерней таблицей, а таблица, содержащая ключ, на который ссылается внешний ключ — родительской таблицей. Как и со всеми прочими типами ограничений, ссылочные ограничения целостности можно создавать во время создания таблицы или позднее, с помощью оператора ALTER TABLE:

CREATE TABLE employee

(employee\_id NUMBER(7),

last\_name VARCHAR2(30),

first name VARCHAR2(30),

job VARCHAR2(15),

dept\_id NUMBER(3) NOT NULL

CONSTRAINT dept\_fkey REFERENCES dept(dept\_id));

База данных назначает столбец dept\_id таблицы employee внешним ключом, потому что он ссылается на столбец dept\_id таблицы dept. Обратите внимание, что для того, чтобы столбец служил ссылочным (на который ссылаются), он должен быть уникальным или же быть первичным ключом в таблице, на которую установлена ссылка.

#### Включение отключение удаление ограничений

Синтаксис

ALTER TABLE table  
DROP PRIMARY KEY | UNIQUE (column) |  
CONSTRAINT constraint [CASCADE];

ALTER TABLE supplier DISABLE CONSTRAINT имя\_ограничения;

ENABLE

DROP

ALTER TABLE dept2 DROP PRIMARY KEY CASCADE;

#### Состояния ограничений целостности

**DISABLE NOVALIDATE** (отключенное непроверенное состояние)  
При отключении без проверки, Oracle не предпринимает проверку полей. Это по существу то же самое, что командаDISABLE CONSTRAINT.

Данные, находящиеся в таблице, а также вводимые заново, могут не подчиняться правилам, определяемым в ограничении. Такое состояние часто используется, когда данные были получены из проверенного источника и таблица доступна только на чтение, поэтому в нее не вводятся новые данные.  
  
**DISABLE VALIDATE** (отключенное проверенное состояние)  
В случае использования команды DISABLE VALIDATE выполняются следующие два действия сразу. 1) все имеющиеся данные в таблице удовлетворяют условию ограничения. 2) конструкция DISABLE избавляет от необходимости поддерживать ограничение. Oracle сбрасывает индекс ограничения,но сохраняет его действительным. Вот пример:

ALTER TABLE sales\_data

ADD CONSTRAINT quantity\_unique

UNIQUE (prod\_id,customer\_id) DISABLE VALIDATE;

После выдачи приведенного выше оператора наличие только уникальных комбинаций уникальных ключей prod\_id и customer\_id в таблице гарантируется, однако уникальный индекс не поддерживается.

Если ограничение находится в этом состоянии, тогда запрещены все изменения столбцов этого ограничения, поскольку это может привести к несогласованности существующих данных или же внесению непроверенных значений. Такое состояние часто используется, когда существующие данные должны быть проверенными, но не должны изменяться в дальнейшем и индекс не нужен для обеспечения производительности.  
  
**ENABLE NOVALIDATE**(включенное непроверенное состояние)  
база данных проверяет все новые вставки и обновления на соответствие условию ограничения. Но существующие данные на такое соответствие не проверяются, и могут не удовлетворять условию ограничения. Обычно эта опция применяется при загрузке крупных таблиц, когда есть основания для уверенности, что эти таблицы уже удовлетворяют условию ограничения.

**ENABLE VALIDATE** (включенное проверенное состояние)  
включает ограничение, гарантируя проверку всех данных на соответствие условию ограничения. Это состояние в точности то же, что обычное состояние “включено”. Это состояние по умолчанию.

#### Как посмотреть ограничения на таблице

SELECT constraint\_name, constraint\_type FROM DBA\_CONSTRAINTS WHHERE table\_name='TESTD';

# Ключи

|  |  |
| --- | --- |
| [Primary Keys](http://oracleplsql.ru/primary-keys.html) | Первичные ключи |
| [Foreign Keys](http://oracleplsql.ru/foreign-keys.html) | Внешние ключи |
| [Foreign Keys with cascade delete](http://oracleplsql.ru/foreign-keys-with-cascade-delete.html) | Внешние ключи с каскадным удалением |
| [Foreign Keys with set null on delete](http://oracleplsql.ru/foreign-keys-with-set-null-on-delete.html) | Внешние ключи с «set null on delete» |
| [Drop a foreign key](http://oracleplsql.ru/drop-a-foreign-key.html) | Удаление внешнего ключа |
| [Disable a foreign key](http://oracleplsql.ru/disable-a-foreign-key.html) | Выключение внешнего ключа |
| [Enable a foreign key](http://oracleplsql.ru/enable-a-foreign-key.html) | Включение внешнего ключа |

**primary key**

представляет собой одно поле или комбинацию полей, которые определяет уникальность записи.

CREATE TABLE supplier

(

  supplier\_id numeric(10) not null,

  supplier\_name varchar2(50) not null,

  contact\_name varchar2(50),

  CONSTRAINT supplier\_pk PRIMARY KEY (supplier\_id)

--CONSTRAINT supplier\_pk PRIMARY KEY (supplier\_id, supplier\_name)

);

**Foreign Key**

Ссылающаяся таблица называется **parent table** (родительской таблицей), а таблица с foreign key (внешним ключом) называется child table (дочерней таблицей). Foreign key в дочерней таблице, как правило, ссылаются на primary key (первичный ключ) в родительской таблице.

Родительская таблица

CREATE TABLE supplier

( supplier\_id numeric(10) not null,

  supplier\_name varchar2(50) not null,

  contact\_name varchar2(50),

  CONSTRAINT supplier\_pk PRIMARY KEY (supplier\_id, supplier\_name)

);

 Дочерняя

CREATE TABLE products

( product\_id numeric(10) not null,

  supplier\_id numeric(10) not null,

  supplier\_name varchar2(50) not null,

  CONSTRAINT fk\_supplier\_comp

    FOREIGN KEY (supplier\_id, supplier\_name)

    REFERENCES supplier(supplier\_id, supplier\_name) –-составной ключ

);

**Foreign keys with cascade delete**

означает, что если удаляется запись в родительской таблице, то соответствующие записи в дочерней таблице будут удалены автоматически. Это называется каскадное удаление в Oracle.

CREATE TABLE products

( product\_id numeric(10) not null,

  supplier\_id numeric(10) not null,

  CONSTRAINT fk\_supplier

    FOREIGN KEY (supplier\_id)

    REFERENCES supplier(supplier\_id)

**ON DELETE CASCADE**

);

Foreign **keys with «set null on delete»**

означает, что если запись в родительской таблице удаляется, и соответствующие записи в дочерней таблице, имеющие foreign key поле со значением NULL, то записи в дочерней таблице не могут быть удалены.

**Удаление отключение включение ключа**

ALTER TABLE table\_name  
DROP CONSTRAINT constraint\_name;

--DISABLE

--ENABLE

# Коллекции

#### Выбираем данные из вложенной таблицы

<http://apps-oracle.ru/nested_tables/>

DECLARE

  TYPE Roster IS TABLE OF VARCHAR2(15);  -- тип вложенная таблица

  -- переменная вложенная таблица, инициализированная конструктором:

  names Roster := Roster('D Caruso', 'J Hamil', 'D Piro', 'R Singh');

  PROCEDURE print\_names (heading VARCHAR2) IS

  BEGIN

    DBMS\_OUTPUT.PUT\_LINE(heading);

    FOR i IN names.FIRST .. names.LAST LOOP  -- Для первого элемента

      DBMS\_OUTPUT.PUT\_LINE(names(i));

    END LOOP;

    DBMS\_OUTPUT.PUT\_LINE('---');

  END;

BEGIN

  print\_names('Initial Values:');

  names(3) := 'P Perez';  -- Изменить значение одного элемента

  print\_names('Current Values:');

  names := Roster('A Jansen', 'B Gupta');  -- Изменить всю таблицу

  print\_names('Current Values:');

END;

В результате получим:

Initial Values:

D Caruso

J Hamil

D Piro

R Singh

---

Current Values:

D Caruso

J Hamil

P Perez

R Singh

---

Current Values:

A Jansen

B Gupta

Еще пример

CREATE OR REPLACE TYPE nt\_type IS TABLE OF NUMBER;

CREATE OR REPLACE PROCEDURE print\_nt (nt nt\_type) IS

  i  NUMBER;

BEGIN

  i := nt.FIRST;

  IF i IS NULL THEN

    DBMS\_OUTPUT.PUT\_LINE('nt is empty');

  ELSE

    WHILE i IS NOT NULL LOOP

      DBMS\_OUTPUT.PUT('nt.(' || i || ') = '); print(nt(i));

      i := nt.NEXT(i);

    END LOOP;

  END IF;

  DBMS\_OUTPUT.PUT\_LINE('---');

END print\_nt;

DECLARE

  nt nt\_type := nt\_type();  -- переменная вложенной таблицы, инициализированная пустой

BEGIN

  print\_nt(nt);

  nt := nt\_type(90, 9, 29, 58);

  print\_nt(nt);

END;

В результате:

nt is empty

---

nt.(1) = 90

nt.(2) = 9

nt.(3) = 29

nt.(4) = 58

#### Выбираем данные из вложенной таблицы во вложенной таблице

declare

r\_operation OPERATION%RowType; --переменная с типом "таблица" т.е. внутри нее поля совпадают с полями таблицы OPERATION

ID0 varchar2(100);

begin

r\_operation :=UIB$Data.getOperation(9921,12,439); --эта функция делает селект из таблицы. т.е. кусок таблицы помещаем в переменую

for i in (select \* from table(r\_operation.field\_values) where name='ID0')loop --выбираем значение из поля field\_values (которое само является таблицей) подполе name

dbms\_output.put\_line(i.value);

end loop;

end;

#### заполнение массивов

create or replace type tabl\_tree as object (

name varchar2(250)

, kod number(8)

)

/

create type tabl\_t is table of tabl\_tree

/

create or replace procedure k\_proba (kod\_error out tabl\_t) is

begin

kod\_error := tabl\_t();

kod\_error.extend();

kod\_error(1) := tabl\_tree(null, null);

kod\_error(1).name :=' 0';

kod\_error(1).kod := 0;

-- Можно так

kod\_error.extend();

kod\_error(2) := tabl\_tree(' 1', 1);

end;

/

declare tt tabl\_t;

begin

k\_proba(tt);

end;

Методы для вложенных таблиц  
  
Exists(n) - Возвращает TRUE, если элемент существует; FALSE – если элемент не существует  
Count - Возвращает текущее количество элементов. Если будет вызван для неинициализированной вложенной таблицы сгенерирует исключение COLLECTION\_IS\_NULL  
First/Last - Возвращает индекс первого и последнего элемента  
Prior(n) - Возвращает индекс предыдущего элемента  
Next(n) - Возвращает индекс следующего элемента  
Extend - Добавляет один пустой элемент в коллекцию  
Extend(n) - Добавляет n элементов в коллекцию  
Extend(n, j) - Добавляет n копий элемента с индексом j  
Trim - Удаляет один элемент с конца коллекции  
Trim(n) - Удаляет n элементов с конца коллекции  
Delete - Удаляет все элементы в коллекции, не вызывает исключение при несуществующем индексе  
Delete(n) - Удаляет элемент с индексом n в коллекции  
Delete(m, n) - Удаляет элементы с индексами из интервала от m до n

#### Cast multiset

select cast(

multiset ( --выбираем данные из таблицы сразу преобразуя их в тип TNamedItem()

-- multiset говорит что берем весь набор TNamedItem()

--а cast приводит этот набор к типу TNamedItems

select TNamedItem(mc.c008, mc.c002)

from mb\_collection\_v mc

where mc.collection\_name = MB\_INVOICES.AC\_CN\_AI\_ATTRIBUTE\_LIST

and mc.n001 = pServiceId

) as TNamedItems

)

into vNamedItems

from dual;

# REF CURSOR

<https://oracle-patches.com/%D0%B1%D0%BB%D0%BE%D0%B3%D0%B8/91-sql-pl/3574-%D0%BA%D1%83%D1%80%D1%81%D0%BE%D1%80%D0%BD%D1%8B%D0%B5-%D0%BF%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D1%8B%D0%B5-%D0%B8-ref-cursor-%D0%B2-pl-sql>