# Лабораторная работа № 10

**Исключительные ситуации**

|  |
| --- |
| Настройка SQL. Необходимо включить режим ECHO и вывести протокол лаб. работы в файл *<Фамилия студента>.txt*. Этот файл является отчетом о проделанной лаб. работе.  SQL> set echo on  SQL> spool c:\spool. txt  ……………………………………..  Завершить протокол лаб. работы (команда spool off) и направить результаты преподавателю.  SQL> spool off |

В PL/SQL предусмотрены механизмы перехвата и обработки ошибок, возникающих при выполнении программы. При обнаружении ошибки генерируется исключительная ситуация, обработка которой производится в разделе EXCEPTION. Существуют два класса исключительных ситуаций: стандартные и определяемые пользователем.

Стандартные исключительные ситуации делятся на два типа:

имеющие и не имеющие предопределенное имя. Имеющие предопределенное имя исключительные ситуации помимо кода имеют еще и стандартное имя, которое используется для идентификации исключения. Ниже приведены примеры некоторых стандартных исключительных ситуаций, имеющих предопределенные имена.

1) ZERO\_DIVIDE – попытка деления на нуль;

2) NO\_DATA\_FOUND – предложение SELECT...INTO не возвращает

ни одной строки;

3) TOO\_MANY\_ROWS – предложение SELECT...INTO возвращает

более одной строки;

4) INVALID\_CURSOR – попытка выполнения запрещенной операции

с курсором (например, закрытие неоткрытого курсора);

5) CURSOR\_ALREADY\_OPEN – попытка открытия уже открытого

курсора;

6) VALUE\_ERROR – арифметическая ошибка, ошибка преобразова-

ния, усечения или ограничения;

7) INVALID\_NUMBER – отказ в преобразовании строки символов в

число.

Пользовательские исключительные ситуации описываются в разделе DECLARE, устанавливаются в выполняемом разделе, а обрабатываются в разделе EXCEPTION. Описание пользовательской исключительной ситуации выполняется заданием имени исключения и фразы

EXCEPTION. Чтобы сгенерировать исключительную ситуацию и передать управление обработчику пользовательской исключительной ситуации в случае обнаружения ошибки, используется оператор RAISE имя\_пользовательского\_исключения

Для перехвата исключительной ситуации любого типа в раздел EXCEPTION должна быть включена фраза WHEN имя\_исключения THEN текст\_обработчика\_исключения;

Тогда при возникновении соответствующей ошибки, вместо прекращения исполнения программы и выдачи типового сообщения об ошибке, будет выполняться созданный пользователем вариант обработки исключения. Если необходимо, чтобы две или более исключительные ситуации обрабатывались одинаково, то они должны быть записаны в одном операторе WHEN, разделенные ключевым словом OR. Для перехвата всех неописанных исключительных ситуаций используется специальный обработчик OTHERS, который записывается последним в разделе EXCEPTION.

Генерация исключительной ситуации с выдачей соответствующего сообщения в рабочую среду в случае обнаружения ошибки может быть выполнена с помощью следующего оператора:

RAISE\_APPLICATION\_ERROR (errnum, errtext);

где errnum – код ошибки, выбираемый в диапазоне –20000 .. –20999;

errtext – поясняющая символьная строка длиной до 512 байт.

При возникновении исключительной ситуации и отсутствии соответ ствующего обработчика в данном блоке система пытается найти такой обработчик в блоках, охватывающих этот блок. При отсутствии обработчика система вернет ошибку «необработанное исключение».

**П р и м е р ы**

1. Необходимо создать таблицу: BOOKS, указав все необходимые ограничения целостности данных.

Перечень, названия и тип данных столбцов таблицы BOOKS:

--------------------------------------------------------------------------

Код книги CODE\_BOOK Number(5)

Название книги TITLE Varchar2(25)

ФИО автора AUTHOR Varchar2(20)

Цена книги PRICE Number(7)

Издательство PUBLISH\_HOUSE Varchar2(15)

Жанр GENRE Varchar2(15)

--------------------------------------------------------------------------

Информация таблицы BOOKS:

Код кн. Назв. книги ФИО автора Цена Изд-во Жанр

----------------------------------------------------------------------------------------------

1 Гибель Богов Перумов Н. 345 Аст Фантастика

2 Казаки Толстой Л. 5568 Нова Роман

3 Ярость Перумов Н. 1385 Аст Детектив

4 Дюна Герберт Ф. 2668 Нова Фантастика

5 Гибель Титана Кристи А. 2345 Аст Роман

6 Дети Дюны Герберт Ф. 2500 Аст Фантастика

----------------------------------------------------------------------------------------------

Создать программу, которая осуществляет в таблице BOOKS повышение цен на книги жанра «Фантастика». При этом при стоимости книги менее 2000 руб., цена увеличивается на 20 %, а при стоимости больше или равной 2000 руб. – на 10 %.

Данная задача реализуется с помощью явно объявленного пользователем курсора. При этом в первых двух вариантах показываются возможности использования обычного курсора и курсора с параметром.

Приведенное решение демонстрирует два варианта обработки явно объявленного курсора: явную и неявную формы. В явной форме обработки по завершении просмотра строк активного набора осуществляется выход из цикла обработки. При этом используется курсорный атрибут

%NOTFOUND. В неявной форме обработки курсора используется конструкция цикл FOR с курсором. Модифицированное значение цены записывается обратно в таблицу BOOKS с использованием конструкции WHERE CURRENT OF, при этом системе с помощью конструкции FOR UPDATE OF PRICE указывается, что будет осуществляться обновление значений столбца PRICE таблицы BOOKS. В программе неявным способом объявлены переменная типа запись ZAP и скалярная переменная

NEW\_ PRICE.

**а) Использование обычного курсора:**

DECLARE

CURSOR KUR IS --явное объявление курсора KUR

SELECT CODE\_BOOK, PRICE FROM BOOKS

WHERE GENRE = 'Фантастика' FOR UPDATE OF PRICE;

ZAP KUR%ROWTYPE; --объявление переменной-записи

NEW\_PRICE BOOKS.PRICE%TYPE;--объявление переменной

BEGIN

OPEN KUR; --явное открытие курсора

LOOP

FETCH KUR INTO ZAP; --выборка текущей записи

EXIT WHEN KUR%NOTFOUND;--выход из цикла

IF ZAP.PRICE < 2000 THEN --изменение цены

NEW\_PRICE := ZAP.PRICE\*1.2;

ELSE

NEW\_PRICE := ZAP.PRICE\*1.1;

END IF;

UPDATE BOOKS

SET PRICE = NEW\_PRICE --обновление цены

WHERE CURRENT OF KUR;

END LOOP;

CLOSE KUR; --явное закрытие курсора

COMMIT; --завершение транзакции

END;

**б) Использование курсора с параметром:**

DECLARE

CURSOR KUR (GANR BOOKS.GENRE%TYPE) IS --курсор

имеет параметр

SELECT CODE\_BOOK, PRICE FROM BOOKS

WHERE GENRE=GANR FOR UPDATE OF PRICE;

ZAP KUR%ROWTYPE;

NEW\_PRICE BOOKS.PRICE%TYPE;

BEGIN

OPEN KUR ('Фантастика'); -- значение параметра

LOOP

FETCH KUR INTO ZAP;

EXIT WHEN KUR%NOTFOUND;

IF ZAP.PRICE < 2000 THEN

NEW\_PRICE := ZAP.PRICE\*1.2;

ELSE

NEW\_PRICE := ZAP.PRICE\*1.1;

END IF;

UPDATE BOOKS SET PRICE = NEW\_PRICE

WHERE CURRENT OF KUR;

END LOOP;

CLOSE KUR;

COMMIT;

END;

**в) Использование цикла FOR с курсором:**

DECLARE

NEW\_PRICE BOOKS.PRICE%TYPE;

CURSOR KUR IS

SELECT CODE\_BOOK, PRICE FROM BOOKS

WHERE GENRE = 'Фантастика' FOR UPDATE OF PRICE;

BEGIN

FOR ZAP IN KUR LOOP --неявная обработка курсора

IF ZAP.PRICE < 2000 THEN --переменная ZAP неявно объяв-

ляется системой

NEW\_PRICE := ZAP.PRICE\*1.2;

ELSE

NEW\_PRICE := ZAP.PRICE\*1.1;

END IF;

UPDATE BOOKS SET PRICE = NEW\_PRICE

WHERE CURRENT OF KUR;

END LOOP;

COMMIT;

END;

**Oбpaбoткa oшибoк в PL/SQL**

Нельзя создать приложение, которое будет безошибочно работать в любых ситуациях: возможны аппаратные сбои, невыявленные ошибки приложения и ошибки из-за некорректных действий пользователей приложения (клиентов). Если при этом программная ошибка произошла в блоке PL/SQL, вложенном в другой блок, а тот, в свою очередь, вложен в третий блок и т.д., то она может дойти до клиентского приложения. Чтобы устранить возможную отмену большого объема ранее выполненных операций и трафик из-за возвращаемых клиенту ошибок, чтобы посылать клиенту точные сообщения о причине ошибки и способе ее устранения (если она все же дошла до клиента), разработчики приложения должны предусматривать возможные программные ошибки и создавать процедуры, адекватно реагирующие на них.

В PL/SQL предусмотрен механизмы перехвата и обработки ошибок, возникающих при выполнении программы. Эти механизмы называются исключительными ситуациями.  
  
Когда программа обнаруживает заданное условие ошибки, то вызывается соответствующая исключительная ситуация. Обработки исключительных ситуаций в программе производится в разделе EXCEPTION.  
При обнаружении исключительной ситуации, обработка основного тела программы останавливается и управление передается соответствующему обработчику исключительной ситуации, который определяет дальнейшие действия.

В PL/SQL используются следующие типы исключительных ситуаций:

* встроенные исключительные ситуации;
* исключительные ситуации, определяемые пользователем;
* обработчик OTHERS.

**Встроенные исключительные ситуации**

Oracle включает четырнадцать встроенных исключительных ситуаций, соответствующих типовым ошибкам, приведенным в следующей таблице:

Исключительная ситуация ORACLE Описание

---------------------- --------- ---------------------------------------------

CURSOR\_ALREADY\_OPEN ORA-06511 Попытка открытия уже открытого курсора

DUP\_VAL\_ON\_INDEX ORA-00001 Попытка вставить дубликат значения для

уникального индекса

INVALID\_CURSOR ORA-01001 Попытка выполнения запрещенной операции с кур-

сором (например, закрытие неоткрытого курсора)

INVALID\_NUMBER ORA-01722 Отказ преобразования строки символов в число

LOGIN\_DENIED ORA-01017 Неправильное имя пользователь/пароль

NO\_DATA\_FOUND ORA-01403 Предложение SELECT...INTO возвращает ноль строк

NOT\_LOGGED\_ON ORA-01012 Нет подключения к Oracle7

PROGRAM\_ERROR ORA-06501 Внутренняя ошибка PL/SQL

STORAGE\_ERROR ORA-06500 Пакет PL/SQL вышел из пределов памяти или если

память разрушена

TIMEOUT\_ON\_RESOURCE ORA-00051 Истекло время ожидания ресурса Oracle7

TOO\_MANY\_ROWS ORA-01422 Предложение SELECT...INTO возвращает более

одной строки

TRANSACTION\_BACKED\_OUT ORA-00061 Удаленный сервер отменил транзакцию

VALUE\_ERROR ORA-06502 Арифметическая ошибка, ошибка преобразования,

усечения или ограничения

ZERO\_DIVIDE ORA-01476 Попытка деления на ноль

Если в раздел EXCEPTION программы (блока) включена фраза

WHEN имя\_исключения THEN  
текст\_обработчика\_исключения;

с именем какого-либо встроенного исключения и возникла соответствующая ошибка, то вместо прекращения исполнения программы и выдачи типового сообщения об ошибке, будет исполняться созданный пользователем текст обработчика исключения.

Такой обработчик может, например, выяснить ситуацию, при которой произошло **деление на ноль**, и выдать правдоподобный результат операции деления или прервать исполнение программы и дать сообщение об изменении каких-либо данных.  
В последнем случае это может быть не типовое сообщение «Вы пытаетесь делить на ноль», а любое подготовленное пользователем сообщение, например, инструкцию длиной до 2048 символов.

Для выдачи сообщения об ошибке, обеспечения возврата в среду, из которой вызывалась текущая программа (блок) и отмены всех действий, выполненных в текущей транзакции, целесообразно использовать процедуру **RAISE\_APPLICATION\_ERROR(errnum,errtext);** где **errnum** – отрицательное целое число в диапазоне -20000 .. -20999 и **errtext** – символьная строка длиной до 2048 символов.

В приведенном ниже триггере «shtins» использованы два типа встроенных исключительных ситуаций: NO\_DATA\_FOUND и TOO\_MANY\_ROWS.

**DROP** **TRIGGER** shtins;

**CREATE** **TRIGGER** shtins

BEFORE **INSERT** **ON** SHTAT

**FOR** EACH ROW

DECLARE

nach DATE;

kon DATE;

str NUMBER;

minraz NUMBER;

maxraz NUMBER;

nach\_kon EXCEPTION;

err\_str EXCEPTION;

nach\_nach EXCEPTION;

err\_razr EXCEPTION;

err\_razr\_pr EXCEPTION;

err\_stavka EXCEPTION;

BEGIN

**SELECT** min\_razr,max\_razr **INTO** minraz,maxraz **FROM** dolgnosti

**WHERE** dolgn = :new.dolgn;

**IF** :new.razr **NOT** **BETWEEN** minraz **AND** maxraz THEN RAISE err\_razr; END **IF**;

**IF** :new.razr\_proc **NOT** **BETWEEN** 50 **AND** 100 THEN RAISE err\_razr\_pr; END **IF**;

**IF** :new.stavka **NOT** **BETWEEN** 0.25 **AND** 100 THEN RAISE err\_stavka; END **IF**;

**IF** :new.nachalo > :new.konec THEN RAISE nach\_kon; END **IF**;

**SELECT** MAX(stroka) **INTO** str **FROM** shtat;

**IF** :new.stroka <> str+1 THEN RAISE err\_str; END **IF**;

<> *-- метка блока, в котором производится поиск строк с*

*-- параметрами, аналогичными вводимым значениям*

BEGIN

**SELECT** nachalo,konec **INTO** nach,kon **FROM** shtat

**WHERE** OTDEL = :new.otdel **AND** DOLGN = :new.dolgn **AND** RAZR = :new.razr

**AND** RAZR\_PROC = :new.razr\_proc **AND** KONEC =

(**SELECT** MAX(konec) **FROM** shtat

**WHERE** OTDEL = :new.otdel **AND** DOLGN = :new.dolgn

**AND** RAZR = :new.razr **AND** RAZR\_PROC = :new.razr\_proc);

**IF** :new.nachalo <= nach THEN RAISE nach\_nach; END **IF**;

**IF** :new.nachalo <= kon THEN

**UPDATE** SHTAT **SET** konec = (:new.nachalo - 1)

**WHERE** OTDEL =:new.otdel **AND** DOLGN = :new.dolgn **AND** RAZR =:new.razr

**AND** RAZR\_PROC = :new.RAZR\_PROC **AND** konec = kon;

END **IF**;

EXCEPTION *-- начало обработчика исключений блока find\_strings*

WHEN NO\_DATA\_FOUND THEN **NULL**; *-- вызывается, если SELECT блока find\_strings*

*-- не возвращает ни одной строки.*

END find\_strings;

EXCEPTION *-- начало обработчика исключений основной программы*

WHEN NO\_DATA\_FOUND THEN

RAISE\_APPLICATION\_ERROR(-20003,'Должности '||:new.dolgn||' не существует !');

WHEN err\_razr THEN

RAISE\_APPLICATION\_ERROR(-20004,'Значение разряда не попадает в "вилку" разрядов');

WHEN err\_razr\_pr THEN

RAISE\_APPLICATION\_ERROR(-20005,'Разрядный процент должен находиться в пределах 50-100');

WHEN err\_stavka THEN

RAISE\_APPLICATION\_ERROR(-20006,'Число ставок должно находиться в пределах 0.25-100');

WHEN nach\_nach THEN

RAISE\_APPLICATION\_ERROR(-20007,'Дата начала должна быть больше '||to\_char(nach));

WHEN TOO\_MANY\_ROWS THEN

RAISE\_APPLICATION\_ERROR(-20008,'Много строк; обратитесь к АБД.');

WHEN nach\_kon THEN

RAISE\_APPLICATION\_ERROR(-20009,'Дата начала не может быть больше даты конца');

WHEN err\_str THEN

RAISE\_APPLICATION\_ERROR(-20010,'Неправильный номер вводимой строки');

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20011,'Какая-то другая ошибка');

END shtins;

Так как в большом приложении могут часто повторяться встроенные или пользовательские исключительные ситуации, то целесообразно создать в базе данных таблицу (например, USERERR) с уникальными номерами (error\_number) и текстами (error\_text) исключений. Это позволит избежать определения лишних сообщений об ошибках и сделать их согласованными во всем приложении.

При использовании такой таблицы и процедуры RAISE\_APPLICATION\_ERROR надо описать в в разделе DECLARE блока две переменных (например, errnum типа NUMBER и errtext типа VARCHAR2) и использовать в обработчике исключений конструкцию:

WHEN TOO\_MANY\_ROWS THEN

SELECT error\_number,error\_text INTO errnum,errtext FROM usererr

WHERE error\_number = 20008;

RAISE\_APPLICATION\_ERROR(errnum,errtext);

или

WHEN TOO\_MANY\_ROWS THEN

SELECT error\_number,error\_text INTO errnum,errtext FROM usererr

WHERE errtext LIKE 'Много строк; обр%';

RAISE\_APPLICATION\_ERROR(errnum,errtext);

**Исключительные ситуации, определяемые пользователем**

Кроме встроенных могут быть использованы собственные исключительные ситуации, имена которых необходимо описать в разделе DECLARE блока PL/SQL (например, err\_stavka EXCEPTION). В разделе EXCEPTION блока должен быть описан соответствующий обработчик исключительной ситуации, например

WHEN err\_stavka THEN

SELECT error\_number,error\_text INTO errnum,errtext FROM usererr

WHERE errtext LIKE 'Число ставок должно находиться%';

RAISE\_APPLICATION\_ERROR(errnum,errtext);

В теле основной программы определяемые пользователем ошибки обычно проверяются с помощью операторов условия (IF…THEN). Для передачи управления обработчику пользовательской исключительной ситуации в случае обнаружения ошибки используется оператор **RAISE имя\_пользовательского\_исключения** Например

IF :new.stavka NOT BETWEEN 0.25 AND 100 THEN

RAISE err\_stavka;

END IF;

**Обработчик OTHERS**

Если исключительная ситуация не обрабатывается явным образом в блоке и для ее перехвата не используется обработчик OTHERS, то PL/SQL отменяет выполняемые блоком транзакции и возвращает необработанную исключительную ситуацию обратно в вызывающую среду.  
Обработчик особых ситуаций OTHERS описывается последним в программе (блоке) для перехвата всех исключительных ситуаций, которые не были описаны в этой программе (блоке). Он может иметь вид

WHEN OTHERS THEN

RAISE\_APPLICATION\_ERROR(-20011,'Какая-то другая ошибка');

**12. Лекция: Коллекции и записи**

В лекции обсуждаются вопросы создания и применения коллекций в языке PL/SQL.

**Коллекции**

Коллекцией называется упорядоченная группа элементов одного типа. Язык PL/SQL поддерживает три вида коллекций:

* вложенные таблицы (nested tables);
* индексированные таблицы;
* varray-массивы (variable-size arrays).

Доступ к любому элементу вложенной таблицы или varray-массива осуществляется по его индексу, который указывается в скобках после имени переменной типа коллекции. Коллекция может быть передана в качестве параметра. Коллекцию можно использовать:

* для обмена с таблицами баз данных и столбцами данных;
* для передачи столбца данных из приложения клиента в хранимую процедуру или обратно.

Для создания коллекции следует определить тип коллекции - TABLE или VARRAY - и объявить переменную этого типа. Определение типа выполняется в секции объявлений блока PL/SQL, подпрограммы или пакета.

**Вложенные таблицы**

Определение типа вложенной таблицы может иметь следующее формальное описание:

TYPE type\_name IS TABLE OF

element\_type [NOT NULL];

Параметр type\_name указывает имя определяемого типа, а element\_type - это любой допустимый тип данных PL/SQL, исключая некоторые типы, в том числе VARRAY, TABLE, BOOLEAN, LONG, REF CURSOR и т.п.

Вложенную таблицу можно рассматривать как одномерный массив, в котором индексами служат значения целочисленного типа в диапазоне от 1 до 2147483647. Вложенная таблица может иметь пустые элементы, которые появляются после их удаления встроенной процедурой DELETE. Вложенная таблица может динамически увеличиваться.

**Индексированные таблицы**

Индексированные таблицы позволяют работать со столбцами как с единой переменной - массивом.

Определение индексированной таблицы (index-by tables) может иметь следующее формальное описание:

TYPE type\_name IS TABLE

OF element\_type [NOT NULL]

INDEX BY BINARY\_INTEGER;

Индексированная таблица - это вариант вложенной таблицы, в которой элементы могут иметь произвольные целочисленные значения индексов. Такой тип данных очень удобен, если в качестве индекса использовать значение первичного ключа.

**VARRAY-массивы**

Определение типа Varray-массива может иметь следующее формальное описание:

TYPE type\_name IS

{VARRAY | VARYING ARRAY} (size\_limit)

OF element\_type [NOT NULL];

Параметр type\_name указывает имя определяемого типа, size\_limit - максимальное количество элементов, а element\_type - это любой допустимый тип данных PL/SQL, исключая некоторые типы, такие как VARRAY, TABLE, BOOLEAN, LONG, REF CURSOR и т.п.

Если типом элемента является тип "запись", то каждое поле записи должно быть скалярного или объектного типа.

Максимальное количество элементов в Varray-массиве указывается при определении типа и не может изменяться динамически. Доступ к каждому элементу Varray-массива осуществляется по индексу. Varray-массивы можно передавать в качестве параметров. Varray-массивы не могут иметь пустот, так как для них нет операции удаления произвольного элемента массива.

Например:

DECLARE

TYPE d1 IS VARRAY(365) OF DATE;

TYPE rec1 IS

RECORD (v1 VARCHAR2(10),

v2 VARCHAR2(10));

- Массив записей

TYPE arr\_rec IS VARRAY(250) OF rec1;

- Вложенная таблица

TYPE F1T1 IS TABLE OF tbl1.f1%TYPE;

CURSOR c1 IS SELECT \* FROM tbl1;

- Массив записей,

- основанный на курсоре

TYPE t1 IS VARRAY(50) OF c1%ROWTYPE;

TYPE t2 IS TABLE OF tbl1%ROWTYPE

- Индексированная таблица

INDEX BY BINARY\_INTEGER;

- Объявление переменной

rec\_t2 t2;

BEGIN

/\* Использование переменной

типа "индексированная таблица" \*/

SELECT \* INTO rec\_t2(120)

WHERE f1 = 120;

END;

**Инициализация коллекций**

Для инициализации коллекции используется конструктор - автоматически создаваемая функция, одноименная с типом коллекции.

Конструктор создает коллекцию из значений переданного ему списка параметров. Конструктор может быть вызван как в секции объявлений через знак присваивания после указания типа, так и в теле программы. Вызов конструктора без параметров означает инициализацию коллекции как пустой, но не устанавливает ее равной NULL.

Например:

DECLARE

CREATE TYPE rec\_var1

AS VARRAY(3) OF num;

CREATE TYPE rec\_var2

AS VARRAY(3) OF rec\_obj;

r1 rec\_var1; r2 rec\_var2;

BEGIN

/\* Инициализация коллекции

из трех элементов \*/

r1 := rec\_var1 (2.0, 2.1, 2.2);

/\*Инициализация varray-массива,

содержащего объекты типа rec\_obj \*/

r2 := rec\_var2 (rec\_obj(1, 100, 'fff'),

rec\_obj (2,110, 'ggg'),

rec\_obj (3,120, 'jjj'));

Оператор CREATE TYPE позволяет сохранить определяемый тип в базе данных.

Конструктор можно вызывать в любом месте, где допустим вызов функции. Для того чтобы добавить в таблицу базы данных строку, одно из полей которой имеет тип коллекции, следует использовать конструктор.

Например:

BEGIN

INSERT INTO tbl\_coll

VALUES (1, 'aaa', rec\_obj

(3,120, 'jjj'));

**Методы, применяемые при работе с коллекциями**

В PL/SQL реализован ряд встроенных методов для работы с коллекциями. Эти методы вызываются как

collection\_name.method\_name[(parameters)]

Эти методы нельзя вызывать из SQL-оператора.

В следующей таблице приведено описание встроенных функций, используемых для работы с коллекциями.

|  |  |
| --- | --- |
| **Метод** | **Описание** |
| EXISTS(n) | Если n-ый элемент коллекции существует, то функция возвращает значение TRUE |
| COUNT | Функция возвращает реальное количество элементов, которые содержит коллекция |
| LIMIT | Функция возвращает размер varray-массива или NULL - для вложенных таблиц |
| DELETE(m,n) | Эта процедура удаляет элементы из вложенной или индексированной таблицы. Если параметров не задано, то удаляются все элементы коллекции. При задании параметра n удаляется n-ый элемент вложенной таблицы, а если задано оба параметра, то удаляются все элементы в диапазоне от n до m |
| FIRST, LAST | Функции возвращают наименьший и наибольший индекс элементов коллекции. Для пустой вложенной таблицыобе функции возвращают значение NULL. Для Varray-массивов вызов функции FIRST всегда возвращает значение 1 |
| PRIOR(n) | Эта функция используется для цикла или последовательного просмотра элементов вложенных таблиц и возвращает индекс элемента, предшествующего указанному параметром n. Если такого элемента нет, то возвращается значение NULL |
| NEXT(n) | Функция употребляется для цикла или последовательного просмотра элементов вложенных таблиц и возвращает индекс элемента, следующего за указанным параметром n. Если такого элемента нет, то возвращается значение NULL |
| EXTEND(n,i) | Функция увеличивает размер вложенной или индексированной таблицы, позволяя добавлять в конец коллекции как один элемент, так и несколько элементов. Если параметров не задано, то в коллекцию добавляется один null-элемент, а если указан только параметр n, то добавляются n null-элементов. Если задано оба параметра, то добавляются n элементов, являющихся копиями i-го элемента коллекции |
| TRIM(n) | Функция выполняет удаление одного или нескольких элементов вложенной или индексированной таблицы. Если параметры не указаны, то удаляется один последний элемент, а при задании параметра удаляются n последних элементов коллекции. Если значение параметра превышает реальное количество элементов, возвращаемое функцией COUNT, то инициируется исключение. Если элемент был ранее удален функцией DELETE, то он все равно будет входить в число удаляемых функцией TRIM элементов |

Применение функций TRIM и EXTEND реализует для вложенных таблиц механизм стека, позволяя удалять элементы и добавлять их в конец вложенной таблицы. Функция DELETE выполняет удаление элементов, оставляя пустые места, которые впоследствии учитываются функцией TRIM.

Значение, возвращаемое функцией COUNT, может использоваться как максимальное значение для счетчика цикла по элементам коллекции.

Например:

FOR i IN 1..tbl1.COUNT LOOP

END LOOP;

Функция COUNT также позволяет определить количество строк, которые были извлечены из столбца базы данных во вложенную таблицу.

Для Varray-массивов значение, возвращаемое функцией COUNT, эквивалентно значению, возвращаемому функцией LAST. Для вложенных таблиц эти значения могут быть различны в том случае, если выполнялась процедура DELETE, удаляющая элементы из коллекции.

Перед тем как коллекция будет проинициализирована, можно использовать только метод EXISTS. При вызове любого другого встроенного метода будет инициировано исключение.

Например:

DECLARE

- Вложенная таблица

TYPE cl IS TABLE OF VARCHAR2(10);

c1 cl;

BEGIN

- Инициализации

- коллекции конструктором

c1 := cl('с 1', 'с 2', 'с 3');

- Удаление последнего (3-го) элемента

c1.DELETE(c1.LAST);

- Удаление двух последних элементов:

- (2-го и 3-го)

c1.TRIM(c1.COUNT);

- Запись в поток

- вывода значения 'с 1'

DBMS\_OUTPUT.PUT\_LINE(c1(1));

END;

Если при работе с коллекцией происходит ошибка, то Oracle инициирует бросок исключения. В следующей таблице приведены основные причины возникновения ошибок для коллекций.

|  |  |
| --- | --- |
| **Исключение** | **Причина ошибки** |
| COLLECTION\_IS\_NULL | Коллекция не была инициализирована |
| NO\_DATA\_FOUND | Индекс ссылается на ранее удаленный элемент коллекции |
| SUBSCRIPT\_BEYOND\_COUNT | Индекс больше, чем количество элементов в коллекции |
| SUBSCRIPT\_OUTSIDE\_LIMIT | Индекс не принадлежит допустимому диапазону значений индекса |
| VALUE\_ERROR | Значение индекса равно NULL или не может быть преобразовано в целое |

Исключительная ситуация не инициируется, если для процедуры DELETE в качестве параметра передан индекс, равный NULL, а также при указании индекса ранее удаленного элемента в случае его замещения.

**Записи**

Записью называется набор элементов, хранимых в полях записи. Каждое поле имеет свое имя и тип.

Записи нельзя сравнивать на равенство или неравенство и на эквивалентность значению NULL.

Запись может быть объявлена на базе существующей структуры таблицы или как новый тип.

Для объявления записи, имеющей структуру, соответствующую строке таблицы базы данных, используется атрибут %ROWTYPE.

Объявление нового типа "запись" может иметь следующее формальное описание:

TYPE type\_name IS RECORD

(field\_declaration

[, field\_declaration]...);

Описание поля (field\_declaration) указывается как:

field\_name field\_type

[[NOT NULL] {:= | DEFAULT} expression]

Параметр type\_name задает имя определяемого типа; field\_type указывает тип поля как любой тип PL/SQL за исключением REF CURSOR; expression определяет значение инициализации.

Как и для коллекций, для создания записи следует сначала определить тип RECORD, а затем объявить запись данного типа.

Тип "запись" существует только на время выполнения программы и не может быть сохранен в базе данных в отличие от типов TABLE и VARRAY.

Для доступа к любому полю записи используется следующая нотация: record\_name.field\_name. Если field\_name также является записью, то для доступа к его вложенному полю используется та же нотация: record\_name.field\_name.field1\_nameи т.д.

В выражениях языка PL/SQL выполнять присваивание значения можно как отдельно полю, так и всей записи.

Выполнять присвоение значения всей записи можно двумя способами:

* использовать в качестве присваиваемого значения запись того же типа;
* задать запись в качестве INTO-переменной в SQL-операторе SELECT или FETCH.

Например:

DECLARE

- Определение типа

TYPE T\_rec IS RECORD (

f1 tbl1.f1%TYPE,

f2 VARCHAR2(15),

f3 REAL(7,2));

- Тип на основе строки таблицы

rec3 tbl1%ROWTYPE;

- Объявление записи

rec1 T\_rec;

f\_sum1 REAL;

FUNCTION sum\_f3 (n INTEGER)

RETURN T\_rec IS

rec2 T\_rec;

BEGIN

- :

- Функция возвращает значение

- типа запись

RETURN rec2;

END;

BEGIN

- Вызов функции

rec1 := sum\_f3(4);

f\_sum1 := sum\_f3(4).f3;

SELECT \* INTO rec3 FROM tbl1

WHERE f1 = 1;

END;

* [Том Кайт о сервере Oracle Database 11g](http://www.cyberguru.ru/database/oracle/kyte-oracle11g.html?showall=&start=0)
* [Кешируйте](http://www.cyberguru.ru/database/oracle/kyte-oracle11g.html?showall=&start=1)
* [Постойте, это ещё не все...](http://www.cyberguru.ru/database/oracle/kyte-oracle11g.html?showall=&start=2)
* [Если WHEN OTHERS, то ничего не возвращается](http://www.cyberguru.ru/database/oracle/kyte-oracle11g.html?showall=&start=3)
* [Пустячок, а приятно. . .](http://www.cyberguru.ru/database/oracle/kyte-oracle11g.html?showall=&start=4)
* [Все страницы](http://www.cyberguru.ru/database/oracle/kyte-oracle11g.html?showall=1&start=0)

**КЕШИРУЙТЕ**

Сервер Oracle Database 11g поднимает планку кеширования данных. В прошлом сервер Oracle Database кешировал блоки базы данных. Он мог кешировать эти блоки в различных местах, таких как пул по умолчанию, удерживающий буферный пул или рециклирующий буферный пул. Но он всегда кешировал блоки данных - строительные блоки, используемые для построения результирующих наборов.  
  
Начиная с Oracle Database 11g, сервер может кешировать результирующие наборы! Если у вас есть запрос, который многократно обращается к медленно изменяющимся или никогда не изменяющимся данным, вы обнаружите, что новый серверный кеш результатов выполнения запросов (server results cache) вызовет чрезвычайный интерес. Практически любое приложение может и будет получать выгоду от использования этого кеша.  
  
Если кратко, идея такова: когда вы выполняете запрос, сервер Oracle Database 11g сначала выяснит, а не были ли результаты этого запроса уже вычислены и кешированы другим сеансом или пользователем, если это так, то он извлечет результат из серверного кеша результатов, вместо того, чтобы снова с нуля собирать все нужные блоки базы данных и вычислять результат. Для первого выполнения запроса потребуется столько же времени, сколько и обычно (поскольку вычисляется ответ), а последующие выполнения будут мгновенными, поскольку ответ не вычисляется, а сразу же возвращается.  
  
Я считаю, что это средство аналогично динамическим "материализованным представлениям" (just-in-time materialized view). В сервере Oracle8i Database Release 1 корпорация Oracle ввела понятие материализованных представлений. С помощью такого представления АБД мог создать таблицу итогов, почти так же, как он создавал индексы, и оптимизатор распознавал, что эта таблица итогов существует и, если возможно, использовал ее для ответа на запросы, вместо того, чтобы запрашивать и обрабатывать детальную информацию, которая хранится в базовых таблицах. Этот способ работы хорош, но он скорее статический, очень похожий на схему индексирования. Однако в сервере Oracle Database 11g с помощью серверного кеша результатов материализованные представления создаются и сопровождаются в сущности "на проходе". Этот кеш (динамическое материализованное представление) заполняется при необходимости, без какого-либо вмешательства АБД. В качестве примера я скопирую представление ALL\_OBJECTS в таблицу:

SQL> create table t  
  2  as  
  3  select \*  
  4    from all\_objects;  
Table created.

Теперь я создам небольшую функцию, которая формирует отчет - выполняет запрос к этой таблице. Для четкой демонстрации отличий в производительности этого запроса я не буду ничего делать внутри цикла (обычно в нем я печатаю строки и выполняю некоторые вычисления):

SQL> create or replace procedure   
  2  my\_function  
  3  as  
  4  begin  
  5    for x in  
  6    (select owner,   
  7              object\_type,   
  8              count(\*) cnt  
  9       from t  
 10          group by owner, object\_type  
 11          order by owner, object\_type )  
 12    loop  
 13          -- do\_something  
 14          null;  
 15    end loop;  
 16  end;  
 17  /  
Procedure created.

Теперь я три раза выполню эту процедуру, засекая время каждого выполнения:

SQL> set timing on  
  
SQL> exec my\_function  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:01.54  
  
SQL> exec my\_function  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.10  
  
SQL> exec my\_function  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.11  
  
SQL> set timing off

Обратите внимание, для первого выполнения потребовалось существенное время, около 1.5 секунд, из-за физического ввода-вывода, необходимого для чтения данных с диска в обычный буферный кеш. Для второго и третьего выполнения потребовалось намного меньше времени, около 0.1 секунды, поскольку блоки, необходимые для получения ответа запроса, были найдены в кеше, а не на диске. Сравним это с тем, что произойдет, если я к запросу добавлю подсказку "cache the results" (кешировать результаты):

SQL> create or replace procedure   
  2  my\_function  
  3  as  
  4  begin  
  5    for x in  
  6    (select /\*+ result\_cache \*/  
  7               owner,   
  8               object\_type,   
  9               count(\*) cnt  
 10       from t  
 11          group by owner, object\_type  
 12          order by owner, object\_type )  
 13    loop  
 14          -- do\_something  
 15          null;  
 16    end loop;  
 17  end;  
 18  /  
Procedure created.  
  
SQL> set timing on  
  
SQL> exec my\_function  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.10  
  
SQL> exec my\_function  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.00  
  
SQL> exec my\_function  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.01  
  
SQL> set timing off

Обратите внимание, для первого выполнения потребовалось около 0.1 секунды, поскольку для этого первого выполнения ответ нужно было компоновать, а последующие выполнения были чертовски быстрыми - иногда такими быстрыми, что казались мгновенными.  
  
Еще хорошая вещь об этом - этот кеш автоматически делается недействительным и обновляется сервером базы данных, этот процесс совершенно прозрачен для приложения. Приложение не должно заботиться об устаревших или недействительных результатах. Например, если я обновлю одну строку, изменив, таким образом, результаты:

SQL> update t  
  2  set owner = lower(owner)  
  3  where rownum = 1;  
1 row updated.  
  
SQL> commit;  
Commit complete.  
  
- то я вижу следующее:  
  
SQL> set timing on  
  
SQL> exec my\_function  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.10  
  
SQL> exec my\_function  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.00  
  
SQL> exec my\_function  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.01  
  
SQL> set timing off

Обратите внимание, что после выполнения оператора UPDATE для первого выполнения запроса снова потребовалось около 0.1 секунды, поскольку нужно было строить новый ответ. От этой работы получают выгоды последующие выполнения, которые кажутся мгновенными.  
  
Я уверен, что, если вы удобно усядетесь и поразмышляете о ваших собственных приложениях, вы сможете найти более одного места, где пригодится серверный кеш результатов. Этот кеш предоставляет собой многие из преимуществ некоторых материализованных представлений, но не требует настройки и административных накладных расходов.

**ПОСТОЙТЕ, ЭТО ЕЩЁ НЕ ВСЕ...**

Как говорят в ночных телепрограммах: "Следующий сюжет удивит вас ещё больше!" В сервере Oracle Database 11g есть также новый кеш результатов вычисления функций PL/SQL (PL/SQL function results cache). Если в вышерассмотренном серверном кеше результатов кешируются результирующие наборы SQL-операторов, то в этом новом кеше кешируются результаты вызовов функций и процедур PL/SQL.  
  
Раньше, если вы вызывали PL/SQL-функцию 1 000 раз и каждый вызов потреблял 1 секунду, то для 1 000 вызовов требовалось 1 000 секунд. В зависимости от входных данных и изменения используемых данных, которые находятся в базе данных, кеш результатов вычисления функций позволяет выполнить 1 000 вызовов функции всего приблизительно за 1 секунду. Будет полезен небольшой пример: я создам две функции, они идентичны, за исключением имен и опций параметров компилятора. Обе они обращаются к ранее созданной таблице T:

SQL> create or replace  
  2  function not\_cached  
  3  ( p\_owner in varchar2 )  
  4  return number  
  5  as  
  6          l\_cnt number;  
  7  begin  
  8          select count(\*)  
  9            into l\_cnt  
 10            from t  
 11           where owner = p\_owner;  
 12          dbms\_lock.sleep(1);  
 13          return l\_cnt;  
 14  end;  
 15  /  
Function created.  
  
SQL> create or replace  
  2  function cached  
  3  ( p\_owner in varchar2 )  
  4  return number  
  5  result\_cache  
  6  relies\_on(T)  
  7  as  
  8          l\_cnt number;  
  9  begin  
 10          select count(\*)  
 11            into l\_cnt  
 12            from t  
 13           where owner = p\_owner;  
 14          dbms\_lock.sleep(1);  
 15          return l\_cnt;  
 16  end;  
 17  /  
Function created.

Единственное различие в этих функциях (исключая их имена) - параметры компилятора: RESULT\_CACHE (кеш результатов) и RELIES\_ON (основан на). Директива RESULT\_CACHE указывает серверу Oracle Database, что вы хотите сохранить ответы этой функции, так что, если потом кто-нибудь вызовет ее с такими же входными параметрами, то код этой функции не будет выполнятся, а сразу же будет выдаваться уже известный ответ. Предложение RELIES\_ON указывает серверу базы данных, когда делать недействительным значение кеша результатов вычисления этой функции - в данном случае при модификации таблицы T (в этом случае изменяется ответ моей кешированной функции, поэтому его нужно вычислить снова). Обратите внимание, для большего эффекта я обе функции перевожу в состояние односекундного ожидания, оно позволяет сделать более заметными отличия реальных вызовов функции от повторного использования результатов.  
  
Я начну с трехкратного вызова обычной (некешированной) функции с включенным таймированием:

SQL> exec dbms\_output.put\_line( not\_cached( 'SCOTT' ) );  
6  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:01.93  
  
SQL> exec dbms\_output.put\_line( not\_cached( 'SCOTT' ) );  
6  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:01.29  
  
SQL> exec dbms\_output.put\_line( not\_cached( 'SCOTT' ) );  
6  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:01.07

Как видите, для каждого вызова требуется по меньшей мере одна секунда - работа самой функции плюс выполнение ею SQL-оператора. Теперь я испытаю кешированную версию этой функции:

SQL> exec dbms\_output.put\_line( cached( 'SCOTT' ) );  
6  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:01.09  
  
SQL> exec dbms\_output.put\_line( cached( 'SCOTT' ) );  
6  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.01  
  
SQL> exec dbms\_output.put\_line( cached( 'SCOTT' ) );  
6  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.01

Как видите, для первого выполнения потребовалось немного больше секунды, примерно также как и для некешированной версии, а последующие вызовы выполнялись со скоростью света - просто потому, что на самом деле кешированная функция не выполнялась. Если я модифицирую используемую в функции таблицу или изменю использованные мной входные данные, я могу увидеть, что сервер базы данных делает то, что нужно:

SQL> update t set owner = initcap(owner) where rownum = 1;  
1 row updated.  
  
SQL> commit;  
Commit complete.  
  
SQL> exec dbms\_output.put\_line( cached( 'SCOTT' ) );  
6  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:01.25

Для первого выполнения кешированной функции потребовалось больше секунды, поскольку она должна обновить кеш результатов, а последующие выполнения получают выгоды, используя этот кешированный результат:

SQL> exec dbms\_output.put\_line( cached( 'SCOTT' ) );  
6  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.01  
  
SQL> exec dbms\_output.put\_line( cached( 'SCOTT' ) );  
6  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.01

Чтобы показать, что кеш результатов вычисления функций PL/SQL понимает, что изменение входных данных приводит к изменению выходных данных, я могу вызвать кешированную функцию с другим именем пользователя:

SQL> exec dbms\_output.put\_line( cached( 'SYS' ) );  
29339  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:01.21  
  
SQL> exec dbms\_output.put\_line( cached( 'SYS' ) );  
29339  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.01

Обратите внимание, для первого выполнения потребовалось относительно много времени (для построения кеша результатов), а последующее выполнение было быстрым. Вызов этой функции с другим именем пользователя не делает недействительными другие кешированные результаты:

SQL> exec dbms\_output.put\_line( cached( 'SCOTT' ) );  
6  
PL/SQL procedure successfully completed.  
Elapsed: 00:00:00.00

Вызовы функции с входным параметром SCOTT будут выполняться быстро до тех пор, пока кеш не должен будет сделан недействительным или же не закончиться выделенная для него память, которая понадобилась для других вызовов.  
  
Кроме того, эту возможность можно реализовать без основательного изменения архитектуры приложений - фактически, совсем без изменения архитектуры. Эта возможность может быть активирована параметром компилятора, а выгоды будет получать любой клиент, который вызывает эту функцию. Например, неофициальные тесты в среде Oracle Application Express показали примерно 15-процентное уменьшение времени выполнения - конечно, у вас могут получиться другие результаты тестировани!

**ЕСЛИ WHEN OTHERS, ТО НИЧЕГО НЕ ВОЗВРАЩАЕТСЯ**

В выпуске этой колонки за июль-август 2007 (русский перевод: "Том Кайт: об игнорировании, блокировании и разборах") я писал о программной конструкции языка PL/SQL ¬- обработчике исключительных ситуаций WHEN OTHERS. Я писал: "В языке PL/SQL есть крайне нежелательная для меня программная конструкция WHEN OTHERS."  
  
Я знаю, что корпорация Oracle никогда не будет устранять эту особенность языка, но сейчас она сделана более безопасной. Только чтобы напомнить, почему я хочу, чтобы эта программная конструкция не существовала, я снова процитирую себя:  
Проблема заключается в том, что многие используют предложение WHEN OTHERS без последующего инициирования исключительных ситуаций (вызовы RAISE или RAISE\_APPLICATION\_ERROR). Это фактически скрывает ошибку. На самом деле ошибка происходит, но она не обрабатывается каким-то осмысленным образом, а просто игнорируется - молча. Вызывающий вашего кода не имеет никакого понятия, что случилось нечто чрезвычайное и ваш код сбился, обычно он думает, что на самом деле все работает успешно.  
  
Много раз, очень много раз, нежели я могу вспомнить или сосчитать, причиной "странного" поведения программных модулей PL/SQL оказывалось ненадлежащее использование предложения WHEN OTHERS - за ним не следовал вызов RAISE. Ошибка скрывается, обработка ошибок фактически отключается, а в результате данные обычно логически разрушаются (приложение не делает свою работу) или получается неверный ответ.  
  
Теперь в сервере Oracle Database 11g я легко могу найти кусок проблемного кода. Когда я полагаю, что была инициирована и скрыта исключительная ситуация, я быстро могу проверить мое подозрение. Рассмотрим следующую безобидную процедуру:

SQL> create table t( x varchar2(4000) );  
Table created.  
  
SQL> create or replace  
  2  procedure maintain\_t  
  3  ( p\_str in varchar2 )  
  4  as  
  5  begin  
  6    insert into t  
  7    ( x ) values  
  8    ( p\_str );  
  9  exception  
 10    when others  
 11    then  
 12      -- call some log\_error() routine  
         -- вызов некоторой подпрограммы протоколирования ошибок  
 13      null;  
 14  end;  
 15  /  
Procedure created.

Она настолько проста, насколько это возможно. Ничего не должно работать неправильно, но для случаев, когда происходит ошибка, я протоколирую ее, используя написанную мною вспомогательную процедуру. Где-то она пишет хорошее сообщение об ошибке (будем надеяться!), а, случается, что кто бы не вызвал эту подпрограмму, он не будет иметь понятия, что произошла непредвиденная исключительная ситуация, поэтому она не может быть локализована. (Я постоянно встречаю подобный код.) Теперь, когда кто-то вызовет эту подпрограмму:

SQL> exec maintain\_t( rpad( 'x', 4001,   
'x' ) );  
PL/SQL procedure successfully completed.

Ее выполнение кажется успешным, но это не так:

SQL> select \* from t;  
no rows selected

Здесь и начинается неразбериха: пользователи "обрывают" телефон, пишут электронные письма, они говорят: "Сломался сервер Oracle Database, транзакция завершилась успешно, однако данные - некорректные". На самом деле, проблема заключается в скрытой ошибке. Теперь в среде сервера Oracle Database 11g для обнаружения таких ошибок я просто буду просить людей сначала выполнить над их кодом следующие действия:

SQL> alter procedure maintain\_t compile  
  2  PLSQL\_Warnings = 'enable:all'  
  3  reuse settings  
  4  /  
  
SP2-0805: Procedure altered with   
compilation warnings  
  
SQL> show errors procedure maintain\_t  
Errors for PROCEDURE MAINTAIN\_T:  
  
LINE/COL     ERROR  
----------   ---------------------------------------  
9/8          PLW-06009: procedure   
             "MAINTAIN\_T" OTHERS handler   
             does not end in RAISE or  
             RAISE\_APPLICATION\_ERROR

И тотчас же вы извлечете список приложений (и строки исходного кода), чтобы незамедлительно добавить простые вызовы RAISE, которые позволят увидеть, откуда поступает эта скрытая ошибка. Работа сделана.

**ПУСТЯЧОК, А ПРИЯТНО. . .**

Недаром говорят - пустячок, а приятно. Есть у меня напоследок одна мелочь, которая немного упрощает вашу программистскую жизнь, а язык PL/SQL делает немного более совершенным:

SQL> create sequence this\_is\_nice;  
Sequence created.  
  
SQL> declare  
  2    n number;  
  3  begin  
  4    n := this\_is\_nice.nextval;  
  5    dbms\_output.put\_line( n );  
  6  end;  
  7  /  
1  
PL/SQL procedure successfully completed.

Сравните это с тем, как такое нужно было делать в сервере Oracle Database 10g и предыдущих версиях. В сервере Oracle Database 11g не нужно больше выбирать последовательность из таблицы DUAL. Так что, язык PL/SQL стал намного более совершенным.