5

Работа с составными типами данных

Составные типы данных

В PL/SQL существует 2 составных типа данных:

- Record (запись) - Запись как целое не имеет собственного значения; однако значение имеет каждый ее компонент или поле, а объединение их в единую запись позволяет хранить и обрабатывать все значения как одно целое. Записи сильно упрощают работу программиста, а переход от объявлений уровня полей к уровню записей повышает эффективность написания кода.
- Collection (коллекция) 3 типа
  - Ассоциативный массив
  - Вложенная таблица
  - VARRAY

Создание PL/SQL Record

```
TYPE type_name IS RECORD
(field_declaration[, field_declaration]…);

identifier type_name;
```

где field_declaration описывается как:

```
field_name {field_type |
        variable%TYPE |
    table.column%TYPE |
        table%ROWTYPE}
        [[NOT NULL] {:= | DEFAULT} expr]
```

```
TYPE имя_типа IS RECORD
   (имя_поля1 тип_данных1 [[NOT NULL]:=|DEFAULTзначение_по_умолчанию],
    имя_поля2 тип_данных2 [[NOT NULL]:=|DEFAULT значение_по_умолчанию],
     ...
    имя_поляN тип_данныхN [[NOT NULL]:=|DEFAULT значение_по_умолчанию]
);
```

RECORD

- Объявление записей:
  - Запись, определяемая программистом

```
DECLARE
  TYPE book_info IS RECORD (
                        author books.author%TYPE,
          category VARCHAR2(100),
          total_page_count POSITIVE);
    v_book book_info;
```

  - Запись на основе таблицы

```
DECLARE
    V_CNTY COUNTRIES%ROWTYPE;
```

- Record это НЕ запись таблицы БД

Пример

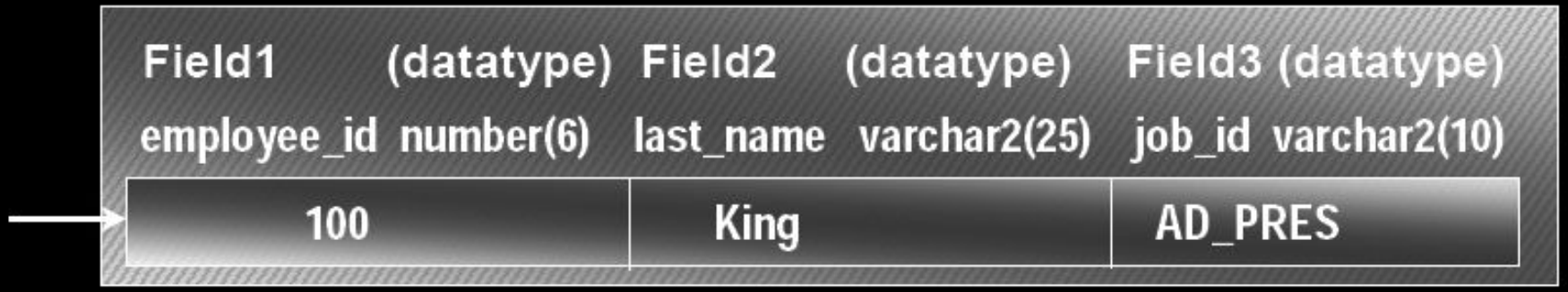
```
...
  TYPE emp_record_type IS RECORD
     (ename   VARCHAR2(25),
      job     VARCHAR2(10),
      sal     NUMBER(8,2));
  emp_record emp_record_type;
...
```

Создание PL/SQL Record

```
...
TYPE emp_record_type IS RECORD
  (empno  NUMBER(4) NOT NULL:=100,
   ename  emp.ename%TYPE,
   job    emp.job%TYPE  );
emp_record emp_record_type;
...
```

Структура PL/SQL Record

Example



Атрибут %ROWTYPE

- Переменная составного типа коллекции столбцов в таблице или представлении.
- Префикс %ROWTYPE используется для обращения к коллекции базирующейся на столбцах таблицы
- Поля в Record будут именованы также и того же типа что и поля таблицы БД

Преимущества использования %ROWTYPE

- Тип данных и количество полей таблицы базы данных может быть неизвестно.
- Тип данных и количество полей таблицы базы данных может быть в любое время изменено.
- Атрибут может быть использован после того как данные выбраны Select *

%ROWTYPE (пример)

Пример объявления переменной, департамент:

```
dept_record departments%ROWTYPE;
```

Пример объявления переменной, сотрудники:

```
emp_record employee%ROWTYPE;
```

Пример

```
DECLARE
   e_rec emp%ROWTYPE;
BEGIN
   SELECT * INTO e_rec FROM employees
      WHERE empno = &employee_number;

   INSERT INTO retired_emps
   (empno, ename, job, mgr, hiredate, sal, comm,deptno
   VALUES (e_rec.empno, e_rec.ename,e_rec.job,
   e_rec.mgr, e_rec.hiredate, e_rec.sal, e_rec.comm,
   e_rec.deptno);

   COMMIT;
END;
```

Пример

```
DECLARE
   my_book books%ROWTYPE;
BEGIN
   SELECT *
    INTO my_book
    FROM books
   WHERE title = 'Oracle PL/SQL';
   DBMS_OUTPUT.put_line ('Код ISBN: ' || my_book.isbn);
END;
```

Операции над RECORD (записями)

PL/SQL поддерживает следующие операции над записями:

- копирование содержимого одной записи в другую (если они имеют совместимую структуру, то есть одинаковое количество полей одного или взаимопреобразуемых типов);
- присваивание записи значения NULL (простым оператором присваивания);
- передача записи в качестве аргумента и возврат записи функцией (RETURN)

Операции на уровне записей не поддерживаются:

- проверить, содержат ли все поля записи значение NULL, использовать синтаксис IS NULL нельзя. Осуществить такую проверку можно лишь путем приме нения оператора IS NULL по отношению к каждому полю
- Невозможно сравнить две записи в одной операции. Нельзя узнать, равны или нет две записи (то есть значения всех их полей), или же узнать, какая из записей больше. чтобы ответить на эти вопросы, нужно сравнить каждую пару полей.

*** Записи в PL/SQL. Глава 11. Oracle_PL-SQL_dlya_professionalov_6-e_izdanie.pdf**

Составные типы данных: Collection

В PL/SQL существует 2 составных типа данных:

- Record (запись)
- Collection (коллекция) – аналог традиционных массивов. 3 типа коллекций : Ассоциативный массив, Вложенная таблица, VARRAY.

  Ситуации для применения :
  - Ведение списков данных.
  - Ускорение многострочных операций SQL. Использование коллекций в сочетании с конструкциями FORALL и BULK COLLECT повышает производительность многострочных операций SQL.
  - Кэширование информации базы данных. Коллекции хорошо подходят для кэширования статической информации, которая часто запрашивается в ходе одного сеанса.

Составные типы данных: Collection

Концепции и терминология

- **Элементы и индексы**. Коллекция состоит из множества элементов, каждый элемент обладает своим индексом. Иногда элементы называются «строками», а индексы — «номерами строк».
- **Целочисленное / строковое индексирование**. Индекс представляет собой целочисленное значение. С Oracle9, в качестве индекса ассоциативных массивов можно использовать строки (до 32 Кбайт длиной) – эта возможность не поддерживается для вложенных таблиц, и VARRAY.
- **Тип коллекции**. Каждая переменная, представляющая коллекцию, должна быть объявлена на основании заранее определенного типа.
- **Разреженные и плотные коллекции.** Коллекция (или массив, или список) называется *плотной*, если все ее элементы, от первого до последнего, определены и каждому из них присвоено некоторое значение (таковым может быть и NULL). Коллекция считается *разреженной*, если отдельные ее элементы отсутствуют.
- **Ограниченная и неограниченная коллекция.** Коллекция называется ограниченной, если заранее определены границы возможных значений индексов.
- **Двумерные структуры** в настоящее время напрямую не поддерживаются, можно создать многомерный массив, объявляя коллекцию коллекцией

Составные типы данных: Collection

Ассоциативные массивы

- Это одномерные неограниченные разреженные коллекции, состоящие из однородных элементов, доступные только в PL/SQL. Ранее в Oracle7) они назывались *таблицами PL/SQL*, а в Oracle8 *индексируемыми таблицами*. С Oracle9 названются *ассоциативными массивами*. INDEX BY используется для индексирования посредством значений типа VARCHAR2 или PLS_INTEGER.

Вложенные таблицы

- одномерные коллекции, состоящие из однородных элементов. Первоначально заполняются полностью, позднее из-за удаления элементов могут стать разреженными. Вложенные таблицы могут определяться и в PL/SQL, и в базах данных (например, в качестве столбцов таблиц). Вложенные таблицы представляют собой *мультимножества*, то есть элементы вложенной таблицы не упорядочены.

Массив типа VARRAY

- Размер всегда ограничен, не бывают разреженными. Как и вложенные таблицы, массивы VARRAY используются и в PL/SQL, и в базах данных. Однако порядок их элементов при сохранении и выборке, в отличие от вложенных таблиц, сохраняется.

Методы коллекций

Метод – встроенная функция, оперирующая с коллекциями.

Метод (функция или процедура)	Описание
COUNT (функция)	Возвращает текущее значение элементов в коллекции
DELETE (процедура)	Удаляет из коллекции один или несколько элементов. Уменьшает значение, возвращаемое функцией COUNT, если заданные элементы еще не удалены. Со структурами VARRAY может использоваться только для удаления всего содержимого
EXISTS (функция)	Возвращает значение TRUE или FALSE, определяющее, существует ли в коллекции заданный элемент
EXTEND (процедура)	Увеличивает количество элементов во вложенной таблице или VARRAY, а также значение, возвращаемое функцией COUNT
FIRST, LAST (функции)	Возвращают индексы первого (FIRST) и последнего (LAST) элемента в коллекции
LIMIT (функция)	Возвращает максимальное количество элементов в массиве VARRAY
PRIOR, NEXT (функции)	Возвращают индексы элементов, предшествующих заданному (PRIOR) и следующему за ним (NEXT). Всегда используйте PRIOR и NEXT для перебора коллекций, особенно при работе с разреженными (или потенциально разреженными) коллекциями
TRIM (функция)	Удаляет элементы, начиная с конца коллекции (элемент с наибольшим индексом)

Collection Ассоциативные массивы

```
DECLARE
  TYPE list_of_names_t IS TABLE OF
       EMPLOYEES.first_name%TYPE
       INDEX BY PLS_INTEGER;
   l_row PLS_INTEGER;
   happyfamily list_of_names_t;
BEGIN
 happyfamily (2020202020) := 'Eli';
 happyfamily (-15070) := 'Steven';
 happyfamily (-90900) := 'Chris';
 happyfamily (88) := 'Veva';

 l_row := happyfamily.FIRST;
 WHILE (l_row IS NOT NULL)
  LOOP
       DBMS_OUTPUT.put_line(happyfamily(l_row));
       l_row := happyfamily.NEXT (l_row);
  END LOOP;
END;
```

Объявление ассоциативного массива TYPE с характерной секцией INDEX BY. Коллекция, содержит список строк, тип - столбца first_name таблицы EMPLOYEES.
Объявление коллекции happyfamily на базе типа list_of_names_t.
Заполнение коллекции четырьмя именами. Можем использовать любые целочисленные значения.
Номера строк в ассоциативном массиве не обязаны быть последовательными, могут быть отрицательными. Не пишите код с непонятными значениями индексов. Это пример
всего лишь демонстрирует гибкость ассоциативных массивов
Вызов метода FIRST (функция, «прикрепленная» к коллекции) для получения первого (минимального) номера строки в коллекции.
Перебор коллекции в цикле WHILE, с выводом каждой строки. Вызывается метод NEXT, который переходит от текущего элемента к следующему без учета промежуточных пропусков

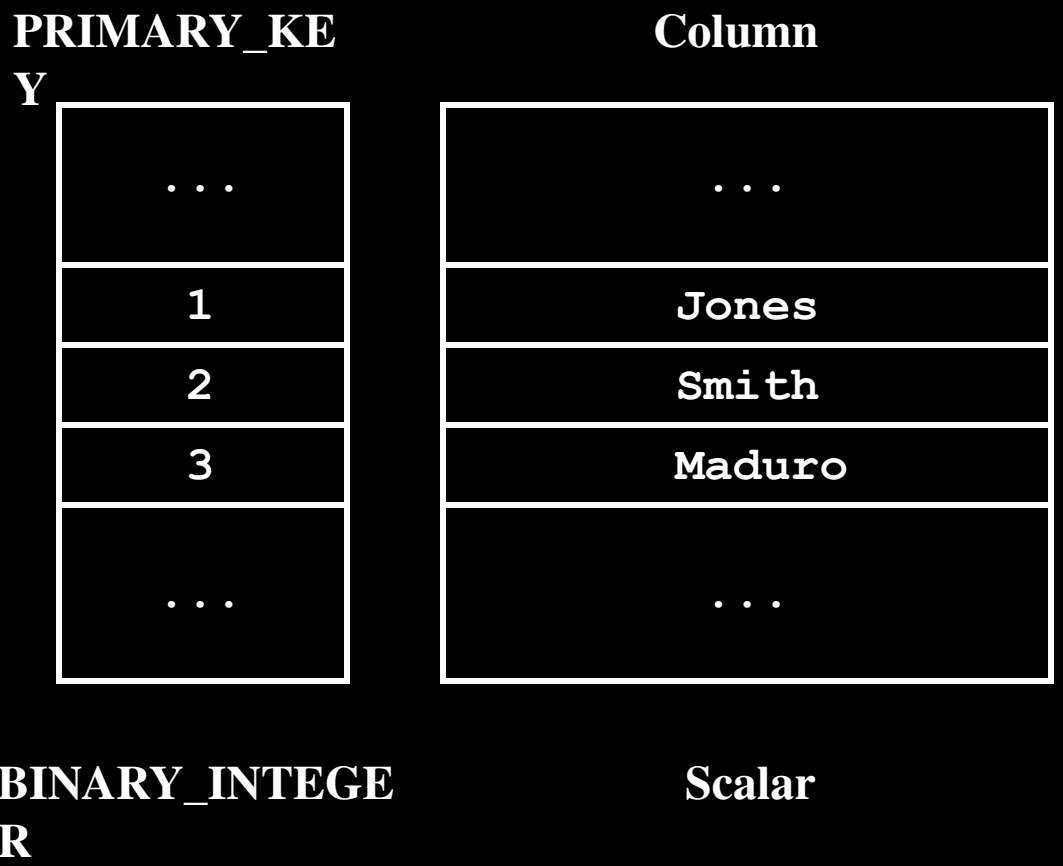
Создание Ассоциативных массивов

```
TYPE type_name IS TABLE OF
   {column_type | variable%TYPE
   | table.column%TYPE} [NOT NULL]
   | table.%ROWTYPE
   [INDEX BY BINARY_INTEGER];
Identifier_name type_name;
```

Описание:

```
...
TYPE ename_table_type IS TABLE OF
        emp.ename%TYPE
   INDEX BY BINARY_INTEGER;
e_table ename_table_type;
...
```

Структура Ассоциативных массивов



Пример

```
DECLARE
   TYPE ename_t_type IS TABLE OF employees.ename%TYPE
                INDEX BY BINARY_INTEGER;
   TYPE hiredate_t_type IS TABLE OF DATE
                INDEX BY BINARY_INTEGER;
   ename_table ename_t_type;
   hiredate_table hiredate_t_type;
BEGIN
   ename_table(1) := 'CAMERON';
   hiredate_table(8) := SYSDATE + 7;
   IF ename_table.EXISTS(1) THEN
   INSERT INTO ...
   ...
END;
```

Пример 1

```
DECLARE
   TYPE dept_table_type IS TABLE OF  dept%ROWTYPE
              INDEX BY BINARY_INTEGER;

   dept_table dept_table_type;
```

* dept%ROWTYPE – строка таблицы DEPT.

Пример 2

```
DECLARE

   TYPE emp_table_type is table of emp.ename%TYPE
              INDEX BY BINARY_INTEGER;

   my_emp_table emp_table_type;

BEGIN
   my_emp_table(1) : = 'SMITH';
   UPDATE emp SET sal = 1.1 * sal
    WHERE ename = my_emp_table(1);
   COMMIT;
END;
```

Collection Вложенная таблица

```
--CREATE TYPE list_of_names_t IS TABLE OF VARCHAR2 (100);
DECLARE
  TYPE list_of_names_t IS TABLE OF VARCHAR2 (100);
  happyfamily list_of_names_t := list_of_names_t ();
  children list_of_names_t := list_of_names_t ();
  parents list_of_names_t := list_of_names_t ();
BEGIN
   happyfamily.EXTEND (4);   happyfamily (1) := 'Eli';
   happyfamily (2) := 'Steven';   happyfamily (3) := 'Chris';
   happyfamily (4) := 'Veva';
   children.EXTEND;        children (1) := 'Chris';
   children.EXTEND;        children (2) := 'Eli';
   parents := happyfamily MULTISET EXCEPT children;
   FOR I IN parents.FIRST .. parents.LAST
    LOOP
         DBMS_OUTPUT.put_line (parents (i));
    END LOOP;
END;
```

Команда CREATE TYPE создает тип вложенной таблицы в базе данных. Такое решение позволяет объявлять вложенные таблицы в любой программе и объявлять столбцы этого типа в таблицах.

Вызов метода EXTEND «выделяет место» во вложенной таблице для данных членов семьи. В отличие от ассоциативных массивов, во вложенных таблицах необходимо явно выполнить операцию создания элемента, прежде чем размещать в нем данные.

С версии Oracle10g, с появлением высокоуровневых команд для работы с множествами вроде MULTISET EXCEPT (аналог SQL MINUS). Перед заполнением parents вызывать метод EXTEND не нужно. База данных делает это автоматически.

Операция MULTISET EXCEPT плотно заполняет коллекцию, для перебора можно пользоваться FOR со счетчиком. При использовании конструкции с разреженной коллекцией произойдет исключение NO_DATA_FOUND

Collection VARRAY

```
CREATE TYPE first_names_t IS VARRAY (2) OF VARCHAR2 (100);
CREATE TYPE child_names_t IS VARRAY (1) OF VARCHAR2 (100);
CREATE TABLE family ( surname VARCHAR2(1000),
        parent_names first_names_t,
        children_names child_names_t);

 DECLARE
   parents first_names_t := first_names_t ();
   children child_names_t := child_names_t ();
 BEGIN
  parents.EXTEND (2);   parents (1) := 'Samuel';   parents (2) := 'Charina';
  children.EXTEND;   children (1) := 'Feather';
  INSERT INTO family
  ( surname, parent_names, children_names )
  VALUES ( 'Assurty', parents, children );
 END;

SELECT * FROM family
SURNAME PARENT_NAMES  CHILDREN_NAMES
----------------------------------------
Assurty FIRST_NAMES_T('Samuel', 'Charina') CHILD_NAMES_T('Feather')
```

CREATE TYPE используется для создания двух типов VARRAY. VARRAY необходимо задать максимальную длину коллекции. Создание таблицы из трех столбцов: VARCHAR2 для фамилии и двух столбцов VARRAY (для родителей и детей).

Как и для вложенной таблицы (и в отличие от ассоциативных массивов), должны вызвать функцию-конструктор, имя которой совпадает с именем TYPE, для инициализации структур. Расширение и заполнение коллекций; добавляются имена родителей и одного ребенка. При попытке добавления второго ребенка произойдет ошибка. Простая вставка коллекций в таблицу.

Выбор типа коллекции

Как правило, разработчики PL/SQL склонны к использованию ассоциативных массивов. Потому что ассоциативные массивы требуют минимального объема кода. Их не нужно инициализировать или расширять. Традиционно они считались самой эффективной разновидностью коллекций. Но если коллекция должна храниться в таблице баз данных, ассоциативный массив отпадает. Остается вопрос: вложенная таблица или VARRAY?

Таблица 12.2. Сравнение разновидностей коллекций в Oracle

Характеристика	Ассоциативный массив	Вложенная таблица	Массив VARRAY
Размерность	Одномерная коллекция	Одномерная коллекция	Одномерная коллекция
Может использоваться в SQL?	Нет	Да	Да
Может использоваться как тип данных столбца таблицы?	Нет	Да; данные хранятся в отдельной вспомогательной таблице	Да; данные хранятся в той же таблице
Неинициализированное состояние	Пустая коллекция (не может быть равна NULL); элементы не определены	Атомарное значение NULL; ссылки на элементы недействительны	Атомарное значение NULL; ссылки на элементы недействительны
Инициализация	Автоматическая при объявлении	При вызове конструктора, выборке или присваивании	При вызове конструктора, выборке или присваивании
Ссылка в элементах PL/SQL	BINARY_INTEGER и любые из его подтипов (-2 147 483 647 .. 2,147,483,647)	VARCHAR2 (Oracle9i Release2 и выше)	Положительное целое число от 1 до 2 147 483 647
Разреженная?	Да	Изначально нет, но может стать после удалений	Нет
Ограниченная?	Нет	Может расширяться	Да
Допускает присваивание любому элементу в любой момент времени?	Да	Нет; может потребоваться предварительный вызов EXTEND	Нет; может потребоваться предварительный вызов EXTEND, который не может выходить за верхнюю границу
Способ расширения	Присваивание значения элементу с новым индексом	Встроенная процедура EXTEND (или TRIM для сжатия) без заранее определенного максимума	EXTEND (или TRIM), но не более объявленного максимального размера
Поддерживает проверку равенства?	Нет	Да, Oracle10g и выше	Нет
Поддерживает использование операций над множествами?	Нет	Да, Oracle10g и выше	Нет
Сохраняет порядок следования элементов и индексы при сохранении в базе данных?	–	Нет	Да

Массовая обработка

Массовая обработка - команда FORALL и секция BULK COLLECT - *конструкции массовой обработки* (bulk processing).

PL/SQL выполняет процедурные команды самостоятельно, а команды SQL передает ядру SQL. Каждое переключение контекста приводит к дополнительным затратам ресурсов.

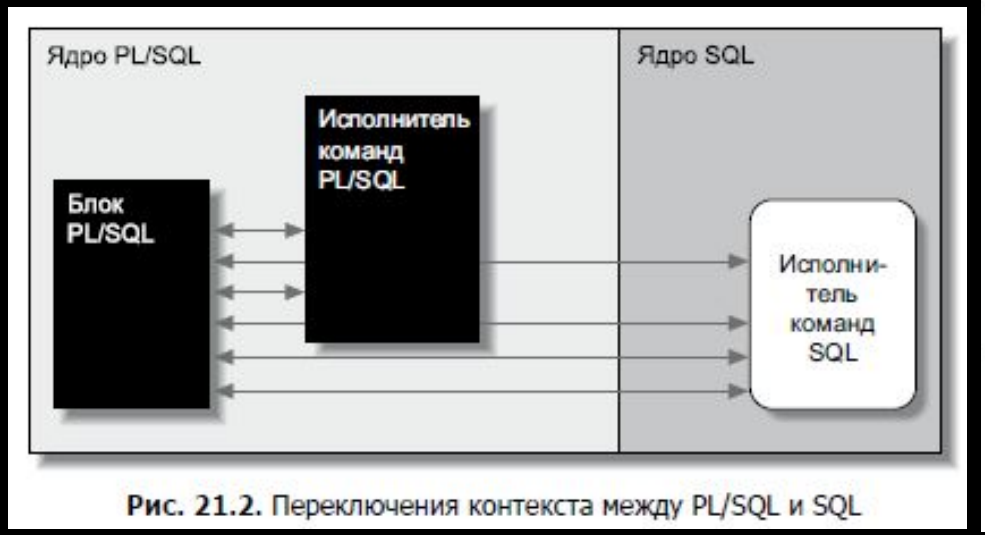


Рис. 21.2. Переключения контекста между PL/SQL и SQL

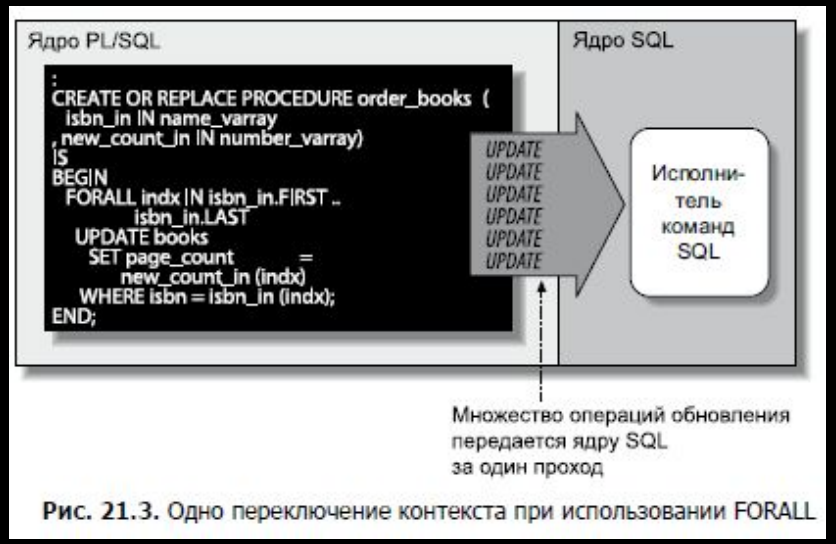


Рис. 21.3. Одно переключение контекста при использовании FORALL

FORALL

FORALL говорит ядру PL/SQL выполнить массовую привязку всех элементов одной или нескольких коллекций перед отправкой команд ядру SQL.
Тело FORALL должно представлять одну команду DML: INSERT, UPDATE, DELETE или MERGE (с Oracle11).
Команда DML должна содержать ссылки на элементы коллекции, индексируемые в команде FORALL. *нижняя_граница* и *верхняя_граница* не обязаны задавать все множество элементов коллекции.

```
for cnt in 1..100000 loop
      insert into temp_bulk values(numb(cnt),name(cnt));
end loop;

FORALL i in 1..100000
      insert into temp_bulk values(numb(i),name(i));
```

* Демонстрация 3х примеров.
* Скорость и ошибки.

```
Execution Time (secs)
---------------------
FOR loop: 4,9
FORALL:   ,14
```

BULK COLLECT

BULK COLLECT позволяет за одно обращение к базе данных извлечь из явного или неявного курсора несколько строк данных. Выборка данных с помощью запроса с секцией BULK COLLECT сокращает количество переключений контекста между PL/SQL и SQL, благодаря чему работа выполняется быстрее и с меньшими затратами.

```
DECLARE
  TYPE first_name_t IS TABLE OF EMPLOYEES.first_name%TYPE;
  TYPE salary_t IS TABLE OF EMPLOYEES.salary%TYPE;
  v_first_name first_name_t;
  v_salary salary_t;
BEGIN
   SELECT first_name, salary BULK COLLECT INTO v_first_name, v_salary
                FROM employees
               WHERE job_id = 'IT_PROG' ;
   FOR i IN 1 .. v_first_name.COUNT
         LOOP
          dbms_output.put_line(v_first_name(i)||' earns - ' ||v_salary(i) );
         END LOOP;
 END;
```

Итоги

- Изучены составные типы данных:
  - Записи
  - Коллекции 2 типа
  - FORALL и BULK COLLECT

Практика №5!

40 минут