Лабораторная работы

Основы языка PL/SQL

**Цель работы:** Получить общее представление о PL/SQL и познакомиться с основными понятиями языка. Изучить реляционные свойства PL/SQL, включая синтаксис языка, типы данных, способы использования SQL, инструкции DML, а также особенности работы с транзакциями. Рассмотреть синтаксис создания подпрограмм (процедур и функций), которые могут храниться и выполнятся на стороне сервера БД.

**Теоретический материал:**

PL/SQL - это развитый язык программирования, используемый для доступа к базам данных Oracle из различных сред. PL/SQL интегрирован с сервером базы данных, поэтому программы PL/SQL обрабатываются быстро и эффективно. Этот язык доступен и в некоторых клиентских инструментальных средствах Oracle.

Язык программирования, используемый для доступа к реляционным базам данных, называется языком структурированных запросов (SQL, Structured Query Language). SQL — гибкий и эффективный язык, все средства которого служат для манипулирования реляционными данными и для их исследования. Например, с помощью SQL-оператора можно удалить из базы данных сведения обо всех студентах (students), профилирующей дисциплиной (major) которых является питание (nutrition):

|  |
| --- |
| *delete from students*  *where last\_name = ‘Иванов’* |

PL/SQL означает Procedural Language/SQL (процедурный язык/SQL). Как видно из названия, PL/SQL расширяет возможности SQL, добавляя в него такие конструкции процедурных языков, как:

      переменные и типы данных (как предварительно определенные, так и определяемые пользователями);

      управляющие структуры, такие как условные операторы и циклы;

      процедуры и функции;

      объектные типы и методы (OracleS и выше);

Процедурные конструкции объединяются с Oracle SQL, что дает в результате структурированный и эффективный язык программирования. Предположим, что требуется изменить профилирующую дисциплину некоего студента. Если записи о нем нет, нужно создать новую запись. Это можно сделать при помощи следующего кода PL/SQL:

|  |
| --- |
| *Declare*  */\*Объвление переменных \*/*  *v\_New VARCHAR2(10):= ‘History’;*  *v\_FirstName VARCHAR2(10):=’Scott’;*  *v\_LastName VARCHAR2(10):=’Urman’;*  *Begin*  */\* Обновление таблицы students \*/*  *UPDATE students*  *SET major= v\_New,*  *WHERE first\_name= v\_FirstName anf last\_name= v\_LastName;*  */\* Проверка, найдена ли запись. Если нет, нужно ее ввести \*/*  *If SQL%NOTFOUND THEN*  *INSERT INTO Students (ID, major, first\_name, last\_name)*  *VALUES (student\_sequence.NEXVAL, v\_New, v\_first\_name,*  *v\_last\_name);*  *End If;*  *End.* |

В этом примере содержатся два разных SQL-оператора (UPDATE (обновить) и INSERT (ввести)), а также SGL-конструкции (операторы объявления переменных и условный оператор IF).

**Блочная структура**

Базовой единицей PL/SQL является блок (block). Все программы PL/SQL строятся из блоков, которые могут быть вложены один в другой. Как правило, в программе каждый блок выполняет определенную логическую единицу работы, что помогает отделить друг от друга различные задачи. Блок имеет следующую структуру:

|  |
| --- |
| *[ DECLARE*  *-- описание переменных, констант и пользовательских типов данных ]*  *BEGIN*  *-- тело основной программы, в которую могут вкладываться другие  блоки (в них также можно вкладывать блоки и т.д.)*  *[ EXCEPTION*  *-- драйверы для обработки исключительных ситуаций в программе ]*  *END;* |

Программы PL/SQL могут быть неименовынными (анонимными блоками), но чаще всего используются именованные программы: процедуры, функции, пакеты и триггера. Обязательным в представленной схеме является только выполняемый раздел. Он должен содержать по меньшей мере один выполняемый оператор. Разные разделы блока в программе PL/SQL предназначены для выполнения различных функций.

В разделе объявлений размещаются объявления всех переменных, курсоров и типов, используемых данным блоком. В этом разделе могут быть объявлены также локальные процедуры и функции. Такие подпрограммы будут доступны только в пределах блока. В выполняемом разделе осуществляется работа блока. В этом разделе могут находиться как процедурные, так и SQL-операторы.

Раздел блока по обработке исключительных ситуаций используется для реагирования на ошибки, которые встречаются во время выполнения программы. При отделении программного кода, предназначенного для обработки ошибок, от основного тела программы структура программы становится гораздо понятнее.

Если раздел объявлений отсутствует, выполнение блока начинается с ключевого слова BEGIN. Если отсутствует раздел исключительных ситуаций, ключевое слово EXCEPTION пропускается и блок заканчивается ключевым словом END и точкой с запятой.

Язык программирования PL/SQL разработан на основе языка третьего поколения Ada. Многие конструкции, применяемые в Ada, можно найти в PL/SQL. Одним из общих свойств этих языков является их блочная структура. Из Ada заимствованы также обработка исключительных ситуаций, синтаксис объявления процедур и функций, модули.

**Типы данных, доступные в PL/SQL**

   Кроме типов данных Oracle7 (см. п. 5), PL/SQL поддерживает несколько дополнительных типов данных и позволяет использовать в своих конструкциях некоторые основные типы с большим диапазоном.

| **Тип данных** | **Описание** |
| --- | --- |
| BINARY\_INTEGER | Этот тип данных и его подтипы NATURAL и POSITIVE применяются для создания переменных и констант, которые хранят число со знаком. Двоичные целые числа могут принимать значения в диапазоне от -2 в 31 степени до 2 в 31 степени минус 1. |
| BOOLEAN | Принимается для создания переменных и констант, в которых хранятся логические значения TRUE и FALSE. |
| CHAR | Есть подтипы CHARACTER и STRING. Максимальный размер 32767. |
| NUMBER | Есть подтипы DEC, DECIMAL, DOUBLE, PRECISION, FLOAT, INT, INTEGER, NUMERIC, REAL иSMALLINT. |
| RECORD | Используется для создания пользовательских типов записей базы данных. |
| TABLE | Служит для создания табличных типов данных PL/SQL. |
| VARCHAR2 | Есть подтип VARCHAR. Максимальный размер 32767. |
| tab%ROWTYPE | Используется для определения типа данных записи по типу данных столбцов таблицы, к имени которой (tab) приписан суффикс %ROWTYPE. |
| col%TYPE | Используется для определения типа данных столбца или переменной по типу данных другого столбца или переменной, к имени которого или которой (col) приписан суффикс %TYPE. |

  Рассмотрим подробнее типы данных TABLE и RECORD, позволяющие создавать одномерные массивы и записи, широко используемые в программах PL/SQL.

***Переменные, константы и выражения***

В программах PL/SQL могут использоваться переменные и константы, описываемые в разделе DECLARE с помощью конструкции вида:

|  |
| --- |
| variable\_name [CONSTANT] type\_name [NOT NULL] [  { := | DEFAULT } expr ] |

   Например:

|  |
| --- |
| birthdate         DATE;      emp\_count     SMALLINT := 0;      emp\_count     SMALLINT DEFAULT 0;      acct\_id           VARCHAR2(5) NOT NULL := 'AP001';      pi                    CONSTANT REAL := 3.14159;      area                 REAL := pi \* radius\*\*2;      valid\_id          BOOLEAN;      valid\_id          VARCHAR2(5);  /\* недопустимое вторичное описание                                                        valid\_id\*/      i, j, k               SMALLINT;     /\* нельзя описывать список; надо:                                                           i SMALLINT; j  SMALLINT;                                                           k  SMALLINT;\*/      credit              REAL(7,2);      debit           credit%TYPE;  /\* тип данных аналогичный типу данных                                                             "credit" \*/ |

***Анонимный блок***

Анонимные блоки - это блоки, которые не имеют имени. Анонимные блоки не могут быть вызваны другими блоками, так как у них нет имени, на которое можно ссылаться.

|  |
| --- |
| *[локальные объявления]*  *BEGIN*  *исполняемые предложения*  *[EXCEPTION*  *обработчики исключений]*  *END [имя];* |

***Процедуры***

Процедура - это подпрограмма, которая выполняет специфическое действие. Вы пишете процедуры, используя следующий синтаксис:

|  |
| --- |
| *PROCEDURE имя [ (параметр [, параметр, ...]) ] IS*  *[локальные объявления]*  *BEGIN*  *исполняемые предложения*  *[EXCEPTION*  *обработчики исключений]*  *END [имя];* |

где каждый "параметр" имеет следующий синтаксис:

|  |
| --- |
| *имя\_перем [IN | OUT | IN OUT] тип\_данных [{:= | DEFAULT} знач]* |

В отличие от спецификатора типа данных в объявлении переменной, спецификатор типа данных для параметра не может иметь ограничений. Например, следующее объявление name незаконно:

|  |
| --- |
| *PROCEDURE ... (name CHAR(20)) IS  -- незаконно; должно быть CHAR*  *BEGIN ... END;* |

Процедура имеет две части: спецификацию и тело. Спецификация процедуры начинается с ключевого слова PROCEDURE и заканчивается именем процедуры или списком параметров. Объявления параметров необязательны. Если процедура не принимает параметров, скобки также не кодируются.

Тело процедуры начинается с ключевого слова IS и заканчивается ключевым словом END, за которым может следовать имя процедуры. Тело процедуры состоит из трех частей: декларативной части, исполняемой части и необязательной части обработки исключений.

Декларативная часть содержит локальные объявления, которые помещаются между ключевыми словами IS и BEGIN. Ключевое слово DECLARE, которое начинает декларативную часть в анонимном блоке PL/SQL, здесь не используется. Исполняемая часть содержит предложения, которые помещаются между ключевыми словами BEGIN и EXCEPTION (или END). В исполняемой части процедуры должно быть хотя бы одно предложение. Часть обработки исключений содержит обработчики исключений, которые помещаются между ключевыми словами EXCEPTION и END.

Рассмотрим процедуру raise\_salary, которая увеличивает жалованье сотрудника:

|  |
| --- |
| *PROCEDURE raise\_salary (emp\_id NUMBER, increase REAL) IS*  *current\_salary  REAL;*  *salary\_missing  EXCEPTION;*  *BEGIN*  *SELECT sal INTO current\_salary FROM emp*  *WHERE empno = emp\_id;*  *IF current\_salary IS NULL THEN*  *RAISE salary\_missing;*  *ELSE*  *UPDATE emp SET sal = sal + increase*  *WHERE empno = emp\_id;*  *END IF;*  *EXCEPTION*  *WHEN NO\_DATA\_FOUND THEN*  *INSERT INTO emp\_audit VALUES (emp\_id, 'No such number');*  *WHEN salary\_missing THEN*  *INSERT INTO emp\_audit VALUES (emp\_id, 'Salary is null');*  *END raise\_salary;* |

При своем вызове эта процедура принимает номер сотрудника и сумму, на которую увеличивается его жалованье. Она использует номер, чтобы выбрать текущее жалованье из таблицы базы данных emp. Если номер сотрудника не найден или текущая зарплата не определена, возбуждается исключение. В противном случае жалованье обновляется.

Процедура вызывается как предложение PL/SQL. Например, процедура raise\_salary может быть вызвана так:

|  |
| --- |
| *raise\_salary(emp\_num, amount);* |

***Функции***

Функция - это подпрограмма, которая вычисляет значение. Функции структурируются так же, как и процедуры, с той разницей, что функции содержат фразу RETURN. Вы пишете функции, используя синтаксис

|  |
| --- |
| *FUNCTION имя [ (аргумент [, аргумент, ...]) ] RETURN тип\_данн IS*  *[локальные объявления]*  *BEGIN*  *исполняемые предложения*  *[EXCEPTION*  *обработчики исключений]*  *END [имя];* |

где каждый "аргумент" имеет следующий синтаксис:

|  |
| --- |
| *имя\_перем [IN | OUT | IN OUT] тип\_данных [{:= | DEFAULT} знач]* |

В отличие от спецификатора типа данных в объявлении переменной, спецификатор типа данных для параметра не может иметь ограничений.

Как и процедура, функция имеет две части: спецификацию и тело. Спецификация функции начинается с ключевого слова FUNCTION и заканчивается фразой RETURN, которая специфицирует тип результирующего значения. Объявления аргументов необязательны. Если функция не принимает аргументов, скобки также не кодируются.

Тело функции начинается с ключевого слова IS и заканчивается ключевым словом END, за которым может следовать имя функции. Тело функции состоит из трех частей: декларативной части, исполняемой части и необязательной части обработки исключений. Декларативная часть содержит локальные объявления, которые помещаются между ключевыми словами IS и BEGIN. Ключевое слово DECLARE, которое начинает декларативную часть в анонимном блоке PL/SQL, здесь не используется. Исполняемая часть содержит предложения, которые помещаются между ключевыми словами BEGIN и EXCEPTION (или END). В исполняемой части процедуры должно встретиться хотя бы одно предложение RETURN. Часть обработки исключений содержит обработчики исключений, которые помещаются между ключевыми словами EXCEPTION и END.

Рассмотрим функцию sal\_ok, которая определяет, не выходит л жалованье для данной должности за допустимые границы:

|  |
| --- |
| *FUNCTION sal\_ok (salary REAL, title REAL) RETURN BOOLEAN IS*  *min\_sal  REAL;*  *max\_sal  REAL;*  *BEGIN*  *SELECT losal, hisal INTO min\_sal, max\_sal FROM sals*  *WHERE job = title;*  *RETURN (salary >= min\_sal) AND (salary <= max\_sal);*  *END sal\_ok;* |

При своем вызове эта функция принимает жалованье сотрудника и его должность. Она использует должность, чтобы выбрать границы оклада из таблицы базы данных sals. Если жалованье выходит за эти границы, sal\_ok возвращает FALSE; в противном случае sal\_ok возвращает TRUE.

Функция вызывается как часть выражения. Например, функция sal\_ok может быть вызвана так:

|  |
| --- |
| *IF sal\_ok(new\_sal, new\_title) THEN*  *...*  *END IF;*  *...*  promotable := sal\_ok(new\_sal, new\_title) AND (rating > 3); |

В обоих случаях идентификатор функции является выражением, которое заменяется своим значением.

**Задания:**

# Лабораторная работа 1

1.Создайте таблицу MyTable(id number, val number)

2. Напишите анонимный блок, который записывает в таблицу MyTable 10 000 целых случайных записей.

3. Напишите собственнуюфункцию, которая выводит TRUE если четных значений val в таблице MyTable больше, FALSE если больше нечетных значений и EQUAL если количество четных и нечетных равно

4. Напишите функцию, которая по введенному значению ID, сгенерирует и выведет в консоль текстовое значение команды insert для вставки указанной строки

5. Написать процедуры, реализующие DML операции (INSERT, UPDATE, DELETE) для указанной таблицы

6. Создайте функцию, вычисляющую общее вознаграждение за год. На вход функции подаются значение месячной зарплаты и процент годовых премиальных. В общем случае общее вознаграждение= (1+ процент годовых премиальных)\*12\* значение месячной зарплаты. При этом предусмотреть что процент вводится как целое число, и требуется преобразовать его к дробному. Предусмотреть защиту от ввода некорректных данных.

# Лабораторная работа 2

1. Построить две таблицы STUDENTS и GROUPS реализующих соответственно справочник студентов и справочник групп

|  |  |  |
| --- | --- | --- |
| Поле | Тип | комментарий |
| **STUDENTS** | | |
| ID | Number | Код студента |
| NAME | VARCHAR2 | Имя студента |
| GROUP\_ID | Number | Код группы |
| **GROUPS** | | |
| ID | Number | Код группы |
| NAME | VARCHAR2 | Название группы |
| C\_VAL | Number | Количество студентов в группе |

2. Реализовать триггеры для таблиц задания 1 проверку целостности (проверка на уникальность полей ID), генерацию автоинкрементного ключа и проверку уникальности для поля GROUP.NAME

3. Реализовать триггер реализующий Foreign Key с каскадным удалением между таблицами STUDENTS и GROUPS

4. Реализовать триггер реализующий журналирование всех действий над данными таблицы STUDENTS

5. Исходя из данных предыдущей задачи, реализовать процедуру для восстановления информации на указанный временной момент и на временное смещение

6. Реализовать триггер, который в случае изменения данных в таблице STUDENTS будет соответственно обновлять информацию C\_VAL таблицы GROUPS

# Лабораторная работа 3

1. Написать процедуру/функцию на вход которой подаются два текстовых параметра (dev\_schema\_name, prod\_schema\_name), которые являются названиями схем баз данных (условно схема для разработки(Dev) и промышленная схема(Prod)), на выход процедура должна предоставить перечень таблиц, которые есть в схеме Dev, но нет в Prod, либо в которых различается структура таблиц. Наименования таблиц должны быть отсортированы в соответствии с очередностью их возможного создания в схеме prod (необходимо учитывать foreign key в схеме). В случае закольцованных связей выводить соответствующее сообщение

2. Доработать предыдущий скрипт с учетом возможности сравнения не только таблиц, но и процедур, функций, индексов пакетов

3. Доработать предыдущий скрипт с генерацией ddl-скрипта на обновление объектов, а также с учетом необходимости удаления в схеме prod объектов, отсутствующих в схеме dev.

**Вывод таблиц которые есть в Dev, но нет в Prod или различна структура**

**Скрипт на обновление объектов**

1. **Есть в Dev, нет в Prod -> добавить в Prod**
2. **Нет в Dev, есть в Prod -> удалить в Prod**

**Удаление в Prod того, чего нет в Dev**