Лабораторная работа №3

ОСНОВЫ ЯЗЫКА PL/SQL

PL/SQL - это процедурный язык, являющийся расширением языка SQL. Этот язык применяется для создания приложений Oracle. Любая программа на PL/SQL состоит из трех блоков:

```
DECLARE
объявление переменных
BEGIN
выполняемый код
EXCEPTION
обработка исключений
END;
```

Блок DECLARE предназначен для описания переменных, констант. Команда BEGIN отмечает начало тела основной программы. В теле программы могут быть помещены другие блоки ограниченные командами BEGIN ...END. EXCEPTION предназначен для определения исключительных ситуаций. Оператор END указывает конец программы. Для запуска программы на компиляцию необходимо в конце поставить символ /.

Определение переменных

Любую переменную можно задать в разделе DECLARE следующим образом:

```
DECLARE
A INTEGER;
C INTEGER;
B VARCHAR2(2);
BEGIN
A:=5;
B:='34';
```

Объявление переменных задается обязательно между операторами DECLARE...BEGIN. Переменные можно задать и следующим образом:

```
DECLARE
A INTEGER :=5;
C INTEGER;
B VARCHAR2(2):='34';
BEGIN
```

При работе с таблицами приходится объявлять большое количество переменных, что может привести в дальнейшем к ошибкам. Объявим переменные для таблицы KURS_RABOTA:

```
DECLARE
p_nom_studenta NUMBER(IO);
p_name_kurs_rab VARCHAR2(40);
BEGIN
```

Где переменная p_nom_ studenta будет хранить данные о поле NOM_ STUDENTA, а переменная p_name_ kurs_rab содержимое поля NAME_ KURS_RAB. А теперь представим ситуацию, что мы хотим поменять в этой

таблице тип для поля NAME_ KURS_RAB. Для этого нужно будет изменить все объявленные переменные. Чтобы избежать данной проблемы, можно воспользоваться оператором %TYPE и задать переменные так:

```
DECLARE
p_nom_studenta
KURS_RABOTA.NOM_ STUDENTA%TYPE;
p _ name _kurs_rab
KURS_RABOTA.NAME_ KURS_RAB%TYPE;
BEGIN
```

В данном случае, если тип поля изменится в таблице, то тип переменных поменяется автоматически.

Также в PL/SQL используется еще один оператор %ROW-TYPE. Рассмотрим на примере как работает данный оператор.

```
DECLARE
p_kurs_rab        KURS_RABOTA%ROWTYPE;
BEGIN
```

Переменная p_kurs_rab будет иметь следующую структуру:

```
p_kurs_rab (
NOM_STUDENTA NUMBER(IO), NAME_KURS_RAB VARCHAR2(40)
```

В итоге мы получаем, что результатом переменной будет запись. Для обращения к этим полям в отдельности необходимо использовать такую запись:

```
DECLARE
a NUMBER(IO) :=p_kurs_rab.NOM_STUDENTA;
b VARCHAR2(40):=p_kurs_rab.NAME_KURS_RAB;
BEGIN
```

Оператор ветвления

Конструкция оператора ветвления IF — THEN — ELSE
имеет вид:

```
IF <условие> THEN BEGIN <оператор 1> <оператор 2>
END; ELSE
BEGIN
END; END IF;
```

В зависимости от результатов проверки условия выполняются либо операторы после команды THEN, если условие истинно, либо операторы после команды ELSE, если условие ложно.

Если необходимо проверить несколько условий, то для этого существует еще и конструкция IF - THEN - ELSIF: IF <условие!> THEN BEGIN <оператор 1>

```
END;
ELSIF <условие2> THEN BEGIN
END;
ELSE
BEGIN
END; END IF;
```

Операторы цикла

В языке PL/SQL также как и в любом другом языке программирования существуют операторы цикла. Их всего три:

1. Простые циклы. Простой цикл имеет следующую структуру:

LOOP
<оператор 1>
EXIT [WHEN <условие выхода из цикла>];
END LOOP;

2. Интерактивные циклы (FOR). Используется, когда из вестно какое число раз буде выполнен данный цикл. Струк тура данного цикла:

FOR <переменная> IN <диапазон> LOOP <оператор 1>
END LOOP;

3. Условные циклы (WHILE). В условном цикле обязательно задается условие его выполнения с помощью операций сравнения. Ниже представлен синтаксис этого цикла:

WHILE <условие_выполнения_цикла> LOOP <оператор 1>
END LOOP;

Создание процедур и функций

CREATE [OR REPLACE]
PROCEDURE <имя_схемы.имя_процедуры>
IS <объявление_переменных>
BEGIN
< выполняемый _ код >
[EXCEPTION
< обработка_ исключений >]
END <имя_процедуры>;

Команда OR REPLACE замещает старый текст процедуры. Если процедура уже определена, и не указана команда OR REPLACE то замена старого значения на новое не произойдет и выведется сообщение об ошибке.

Пример.

```
SQL> CREATE PROCEDURE "SYSTEM"."PROG"
IS
BEGIN
DBMS_OUTPUT.enable;
DBMS_OUTPUT.put_line('ЭТО моя первая процедура');
END "PROG";
/
```

Пакет DBMS_OUTPUT позволяет осуществить вывод на экран данных. Команда SET SERVEROUTPUT ON заставляет сервер базы данных показать то, что в итоге выводит строка DBMS_OUTPUT.put_line.

Функция всегда возвращает значение параметра. Оператор определения функции Oracle имеет синтаксис:

CREATE [OR REPLACE]
FUNCTION <имя_схемы.имя_функции>

RETURN <возвращаемое_значение>
IS <объявление_переменных>
BEGIN
< выполняемый _ код >
[EXCEPTION
<обработка_ исключений >] END <имя_функции>;

Курсоры

Курсоры используются в базе данных Oracle для управлением операциями выборки из базы данных. Курсор - это указатель на набор строк. Все базовые операторы языка запросов SQL являются курсорами. Существуют курсоры явные и неявные. Неявный курсор создается для выполнения оператора SQL без специальных команд управления курсором. Неявный курсор должен всегда возвращать только одну строку! Создадим неявный курсор возвращающий номер студента по названию курсовой работы "Система мо...": SQL> SET SERVEROUTPUT ON DECLARE
a KURS_RABOTA.NOM_STUDENTA%TYPE;
BEGIN
SELECT "NOM_STUDENTA" INTO a FROM "SYSTEM"."KURS_RABOTA"
WHERE "NAME_KURS_RAB" LIKE 'Система мо%'; a:=a;
DBMS_OUTPUT.enable;
DBMS_OUTPUT.put_line('Код студента'||a); END;
/

При работе с явными курсорами выполняется следующая последовательность действий:

1. Создание курсора. Явный курсор в отличав от неявного имеет специальный синтаксис:

CURSOR <имя_схемы.имя_курсора>
IS
SELECT <имя_столбца>,...
FROM <имя_схемы.имя_таблицы>
WHERE <условие_выборки_в_курсор>;

2. Открытие курсора. Курсор открывается с помощью оператора OPEN:

OPEN <имя_схемы.имя_курсора>;

3. Выборка данных. Выборка данных из курсора производится с помощью оператора FETCH:

FETCH <имя_схемы.имя_курсора> INTO <список_ пе-ременных>;

4. Закрытие курсора. Оператор CLOSE закрывает курсор: CLOSE <имя_схемы.имя_курсора>.

Пример.

SET SERVEROUTPUT ON DECLARE
a STUDENT_DATE.SURNAME%TYPE;
b STUDENT_DATE.ADRESS%TYPE;
CURSOR GET_ADRESS
IS
SELECT "SURNAME", "ADRESS"

```
FROM "SYSTEM"."STUDENT_DATE"
WHERE "ADRESS" LIKE 'Бородина%';
BEGIN
OPEN GET_ADRESS;
FETCH GET_ADRESS INTO a, b;
DBMS_OUTPUT.enable;
OBM8_OиTРиТ.рш,_Нпе('Фамилии студентов проживающих по улице Бородина: '||
a||' '||Ь);
CLOSE GET_ADRESS; END;
/
```

Но если мы запустим курсор, то в результате выборки увидим лишь одну запись:

```
Фамилии студентов проживающих по улице Бородина: Петров К.О. Бородина 67/3-21

Процедура PL/SQL успешно завершена.
```

Для выбора всех записей воспользуемся циклом LOOP и изменим наш курсор вот так: SET SERVEROUTPUT ON DECLARE
CURSOR GET_ADRESS IS
SELECT * FROM "SYSTEM"."STUDENT_DATE"
WHERE "ADRESS" LIKE 'Бородина%';
a GET_ADRESS%ROWTYPE;
BEGIN
OPEN GET_ADRESS; LOOP
DBMS_OUTPUT.enable;
DBMS_OUTPUT.put_line('AaHHbie о студентах, проживающих по улице Бородина: '||a.NOM_ STUDENTA||' ' ||a.SURNAME||' ' ||a.ADRESS||' '|| a.DATE_BORN); FETCH GET_ADRESS INTO a; EXIT WHEN GET_ADRESS %NOTFOUND; END LOOP; CLOSE GET_ADRESS; END;
/ Получаем:

```
Данные о студентах, проживающих по улице Бородина:1112 Петров К.О. Бородина
67/3-21 23.11.86
Данные о студентах, проживающих по улице Бородина:1114 Колмогоров Т.Г. Бородина
23/2 -98 07.11.85

Процедура PL/SQL успешно завершена.
```

В предыдущем примере мы использовали запрос на выборку select по определенному условию: живущие по улице Бородина. Но очень часто необходимо менять для конкретной ситуации условие отбора. Чтобы не создавать новый курсор, можно для в этом случае использовать курсор с параметром. В этом случае курсор GET_ADRESS изменится следующим образом:

```
SET SERVEROUTPUT ON
DECLARE
CURSOR GET_ADRESS
```

```
(INAGE " STUDENT_DATE"." ADRESS" %TYPE)
IS
SELECT * FROM "SYSTEM"."STUDENT_DATE"
WHERE "ADRESS" = IN AGE;
a          GET_ADRESS%ROWTYPE;
BEGIN
OPEN CET_ABKE88('Бородина 67/3-21'); LOOP
DBMS_OUTPUT.enable;
DBMS_OUTPUT.put_line('AaHHbie о студентах, проживающих по улице Бородина:
'||a.NOM_STUDENTA||' ' ||a.SURNAME||' ' ||a.ADRESS||' '||a.DATE_BORN); FETCH
GET_ADRESS INTO a; EXIT WHEN GET_ADRESS%NOTFOUND; END LOOP;
CLOSE GET_ADRESS; END;
```

Триггеры

Триггер — это специальная программа, вызываемая СУБД когда пользователь выполняет запрос на модификацию данных.

В основном триггеры используются для:

1. создания сложных ограничений целостности данных, которые невозможно описать при создании таблиц;
2. реализации так называемых "бизнес правил";
3. организации аудита;
4. выполнения каскадных операций над таблицами.

Структура триггера имеет вид: CREATE [OR REPLACE] TRIGGER <имя_схемы.имя_триггера> BEFORE | AFTER <активизирующее_событие> ON <ссыл-ка_ на_ таблицу > FOR EACH ROW
[WHEN <условие_выполнения_триггера>] <тело_триггера>

• <активизирующее_событие> - указывает момент активации триггера BEFORE до срабатывания любого оператора на модификацию данных, AFTER после срабатывания оператора на модификацию данных;

• FOR EACH ROW - если указано активируется от воз действия на строку если нет, то после любого оператора на модификацию данных;

• <условие_выполнения_триггера> если верно, то триггер срабатывает, если ложно, то нет.

Пример.

В данном примере с помощью триггера организуем несложный аудит доступа STUDENT_DATE. Для этого сначала создадим таблицу:

```
SQL> CREATE TABLE "SYSTEM"."AUDIT"
("USER_NAME" VARCHAR2(50), "TIME" DATE);
```

Далее создадим триггер:

```
SQL> CREATE OR REPLACE
TRIGGER "SYSTEM"."ADT"
AFTER INSERT OR DELETE OR UPDATE
ON "STUDENT_DATE"
DECLARE
```

```
BEGIN
INSERT INTO "SYSTEM"."AUDIT"("USER_NAME",
"TIME") VALUES (USER, SYSDATE);
END "SYSTEM"."ADT";
/
```

Проверим работу данного триггера. Вставим в таблицу STU-DENT_DATE запись:

```
SQL> INSERT INTO "SYSTEM"."STUDENT_DATE"
("SURNAME", "GROUP", "NOM_STUDENTA")
VALUES ('Савинов', 'СВ-105', 1117);
```

А теперь посмотрим в таблицу "SYSTEM"."AUDIT" и увидим следующее:

USER_NAME	TIME
SYSTEM	07.04.05

В таблице AUDIT существует запись о том кто и когда осуществлял манипулирование данными в таблице STUDENT_DATE. В данном примере мы создали операторный триггер. Для создания строкового триггера необходимо в нашем примере добавить строку FOR EACH ROW:

```
SQL> CREATE OR REPLACE
TRIGGER "SYSTEM"."ADT"
AFTER INSERT OR DELETE OR UPDATE
ON "STUDENT_DATE"
FOR EACH ROW
DECLARE
BEGIN
INSERT INTO "SYSTEM"."AUDIT"("USER_NAME",
"TIME") VALUES (USER, SYSDATE);
END "SYSTEM"."ADT";
/
```

Посмотрим как выполняется данный триггер. Обновим данные в таблице STUDENT_DATE:

```
SQL> UPDATE "SYSTEM"."STUDENT_DATE"
SET "GROUP" = 'св-404'
WHERE "GROUP" = 'св-304';
```

В данном случае обновится 2 записи (см. лабораторную работу №2). Теперь посмотрим, как изменилось содержимое таблицы AUDIT (предварительно удалим из нее все записи):

```
USER_NAME                                                  TIME
---------------------------------------------------------- --------
SYSTEM                                                     07.04.05
SYSTEM                                                     07.04.05
```

Как видим в таблице AUDIT теперь две записи для каждого обновления.

Очень часто при создании триггеров используются псевдозаписи :old и :new. Обращение к псевдозаписям производится через имена полей. Псевдозапись необходима, когда нужно обратиться к обрабатываемой строке в данный момент времени. Рассмотрим еще один пример создания триггера, позволяющий автоматически заносить данные в поля являющиеся первичным ключом.

Создадим последовательность:

```
SQL> CREATE SEQUENCE AVTO START WITH 1120 INCREMENT BY 1
/
```

Далее создаем триггер с применением псевдозаписи:

```
SQL> CREATE OR REPLACE TRIGGER IDENT
BEFORE INSERT ON "STUDENT_DATE"
FOR EACH ROW
DECLARE
BEGIN
SELECT AVTO.NEXTVAL
INTO :NEW.NOM_STUDENTA
FROM DUAL;
END IDENT;
/
```

Теперь можно добавлять записи в таблицу STUDENT_DATE, не указывая данных для поля NOM_STUDENTA. Это поле будет автоматически заполняться уникальными данными.

Задание к лабораторной работе №3

1. Создайте явный курсор, выбирающий некоторые данные из вашей базы данных (для создания воспользуйтесь любым запросом на выборку из задания к лабораторной работе №2).
2. Создайте курсор такой же, как в предыдущем пункте, но с параметром.
3. Создайте триггер позволяющий учитывать, кто и когда вносил данные в одну из ваших таблиц.
4. Создайте триггеры ко всем своим таблицам, позволяющим автоматически вводить данные для полей являющихся первичным ключом и имеющих числовой тип.