ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ
УНИВЕРСИТЕТ»

ОСНОВЫ ЯЗЫКА PL/SQL

Учебно-методическое пособие для вузов

Составители:
В.В. Гаршина,
С.В. Сапегин

Издательско-полиграфический центр
Воронежского государственного университета
2008

Утверждено ученым советом ФКН от 23 апреля 2008 г., протокол № 6

Рецензент доктор технических наук, проф. А.А. Рындин

Пособие подготовлено на кафедре программирования и информационных технологий факультета компьютерных наук Воронежского государственного университета.

Рекомендуется для студентов факультета компьютерных наук при изучении курсов «Проектирование БД» и «Администрирование БД».

Для специальности: 230201 – Информационные системы и технологии

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Реляционные системы управления базами данных (RDBMS) в настоящее время являются «сердцем» любой корпоративной системы. RDBMS являются наиболее удобными средствами хранения данных в информационных системах различного масштаба: от больших приложений обработки транзакций в банковских системах до персональных систем на PC. По сравнению с файловыми системами системы RDBMS обеспечивают возможность легкой интеграции и обработки значительных объемов операционных данных в реальных информационных системах. Развитие мощных процессоров баз данных сделало возможным применение архитектуры клиент-сервер, оперативной аналитической обработки, организации корпоративных хранилищ данных и других технологий, которые определяют лицо современных информационных систем.

Разработчики современных промышленных баз данных практически всегда сталкиваются с необходимостью обеспечить контроль качества данных, которыми оперируют КИС. Одним из наиболее перспективных подходов к контролю данных БД является разработка серверной логики, выполняющейся непосредственно на сервере БД и позволяющей существенно расширить возможности разработчика. Одной из наиболее зрелых технологий данного рода является расширение PL/SQL (Programming Language) компании Oracle.

1. СРЕДСТВА РАЗРАБОТКИ СЕРВЕРНЫХ ПРИЛОЖЕНИЙ ORACLE

1.1. Утилита SQL*Plus

SQL*Plus – инструмент от Oracle, который позволяет работать с сервером РБД. SQL*Plus распознает и передает команды SQL и операторы PL/SQL на выполнение серверу, а также имеет свой собственный язык команд. Обычно SQL*Plus используется для незапланированного ввода команд, отладочных действий и выборки данных с использованием SQL.

Способ запуска SQL*Plus зависит от используемой операционной системы. Из Windows эту программу можно вызвать, дважды щелкнув на пиктограмме SQL*Plus, после чего в диалоговом окне ввести имя пользователя, пароль и псевдоним базы данных.

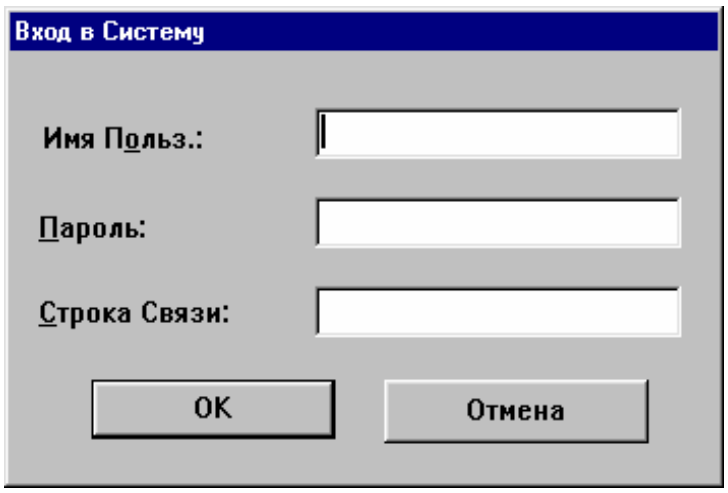

Рис. 1.1. Диалоговое окно регистрации пользователя SQL*Plus

Вызов SQL*Plus из командной строки осуществляется с помощью команды

```
sqlplusw [user[/password[@database]]]
```

где user – имя пользователя, password – пароль, @database – строка подключения к базе данных. Для сохранения секретности пароля рекоменду-

ется вводить его не в командной строке, а в ответ на приглашение SQL*Plus.

Информацию о структуре объекта в SQL*Plus можно получить с помощью команды DESCRIBE. Формат команды:

DESC[RIBE] table_name

где table_name – имя существующей таблицы, представления или синонима, доступных пользователю.

При вводе команды SQL она записывается в область памяти, называемую буфером SQL, и остается там до ввода новой команды. Команды SQL*Plus вводятся по одной строке и не хранятся в буфере SQL. Ввод в буфер SQL прекращается вводом одного из символов окончания (точки с запятой или дробоной черты).

Для редактирования буфера SQL в PL*SQL существуют следующие команды:

A[PPEND] текст – добавляет текст в конец текущей строки;

C[HANGE] /старый/новый/ – заменяет в текущей строке старый текст на новый. Если новый текст не указан – старый удаляется;

CL[EAR] BUFF[ER] – удаляет все строки из буфера SQL;

DEL – удаляет текущую строку;

DEL n – удаляет строку, заданную параметром n;

DEL m n – удаляет строки от m до n;

I[NPUT] – вставляет неопределенное количество строк;

I[NPUT] текст – вставить строку, состоящую из текста;

L[IST] – вывести список всех строк в буфере SQL;

L[IST] n – вывести строку с номером n;

L[IST] m n – вывести диапазон строк от n до m;

R[UN] – Вывести и выполнить команду из буфера SQL;

n – указать строку, которая должна стать текущей (n - число);

n текст – заменить строку n текстом;

0 текст – вставить текст перед первой строкой.

Для работы с файлами используются следующие команды SQL*Plus:

SAV[E] `filename[.ext] [REP[LACE]|APP[END]]` – сохраняет текущее содержимое буфера SQL в файле. Параметр APPEND используется для добавления информации в существующий файл, REPLACE перезаписывает существующий файл. По умолчанию файл имеет расширение `.sql`;

GET `filename` – вызывает содержимое ранее сохраненного файла в буфер SQL;

START `filename` – запускает на выполнение ранее сохраненный файл команд, синоним этой команды – @;

EDIT – вызывает внешний редактор для редактирования буфера SQL;

EDIT `filename[.ext]` – редактирует указываемый файл;

SPO[OL] `[filename[.ext]|OFF|OUT]` – перенаправляет результаты запроса в файл. OFF закрывает буферный файл. OUT закрывает буферный файл и посылает результаты из файла на системный принтер;

EXIT – выход из SQL*Plus;

CONNECT `[user[/password[@database]]]` – подсоединение к базе данных под именем user.

Пример использования команд SQL*Plus:

```
CONNECT scott/tiger@orcl
DESC EMP
SPOOL screen.lst
START sctipt.sql
SPOOL OFF
EXIT
```

С помощью приведенной последовательности команд производится подключение к схеме базы данных scott, выводится описание таблицы emp, запускается на выполнение последовательность команд из файла script.sql и результат выполнения этих команд отражается в файле screen.lst. После выполнения этих действий осуществляется выход из утилиты SQL*Plus.

Управление выводом результатов запроса на экран осуществляется с помощью команды SQL*Plus COLUMN, которая изменяет параметры столбцов отчетов. Синтаксис команды следующий:

COL[UMN] [{column|alias} [options]]

где column – специфицируемый столбец, alias – псевдоним столбца, option – задаваемые установки, которые могут быть следующими:

CLE[AR] – отменяет любые форматы столбцов;

FOR[MAT] format – меняет отображение данных столбца;

HEA[DING] text – задает заголовок столбца, если не используется выравнивание, то знаком "|" можно задать переход на новую строку;

JUS[TIFY] {align} – выравнивает заголовок столбца;

NOPRI[NT] – прячет столбец;

NUL[L] text – задает текст, который должен отображаться в случае неопределенных значений;

PRI[NT] – показывает столбец;

TRU[NCATED] – усекает строку в конце первой строки дисплея;

WRA[PPED] – переходит на следующую строку в конце строки;

WOR[D_WRAPPED] – WRAPPED без разбивки строк.

Если options не заданы, команда выводит значения текущих установок. Элементы, которые можно использовать при определении формата столбца, следующие:

An – задает ширину столбца n для вывода символьных данных и дат;

9 – представляет одну цифру;

0 – вставляет ведущий ноль;

$ – обозначает плавающий символ доллара;

L – обозначает местную валюту;

. – определяет позицию десятичной точки;

, – обозначает разделитель тысяч.

Примеры определения форматов вывода столбцов:

```
COLUMN T_NAME FORMAT A22 HEADING Alias WORD_WRAP
COLUMN MONEY FORMAT 999,999.99L HEADING New_money
```

1.2. Средство разработки TOAD

Инструментальное средство TOAD (Tool for Oracle Application Development) является одним из наиболее популярных средств разработки серверных приложений Oracle. Внешний вид TOAD показан на рисунке 1.2.

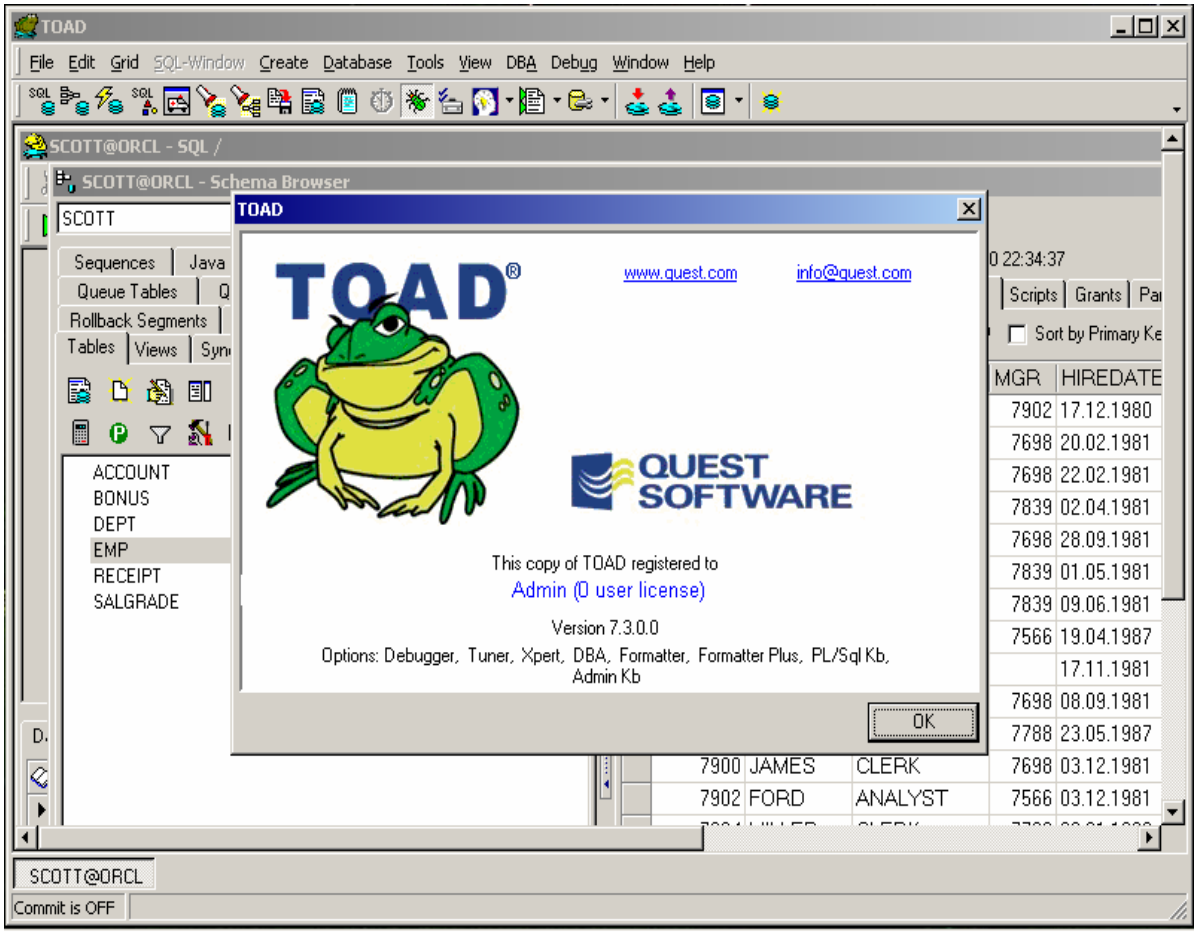

Рис. 1.2. Внешний вид средства TOAD

Запуск TOAD осуществляется путем двойного щелчка мышкой на соответствующей пиктограмме. После запуска средства на экране появляется окно, в котором пользователь может выбрать одно из ранее устанавливаемых соединений с сервером Oracle, либо ввести параметры нового соединения в поля Database, User / Schema и Password (см. рис. 1.3).

После соединения с сервером на экране появляется главное окно программы, в котором пользователь применяет необходимые инструментальные средства.

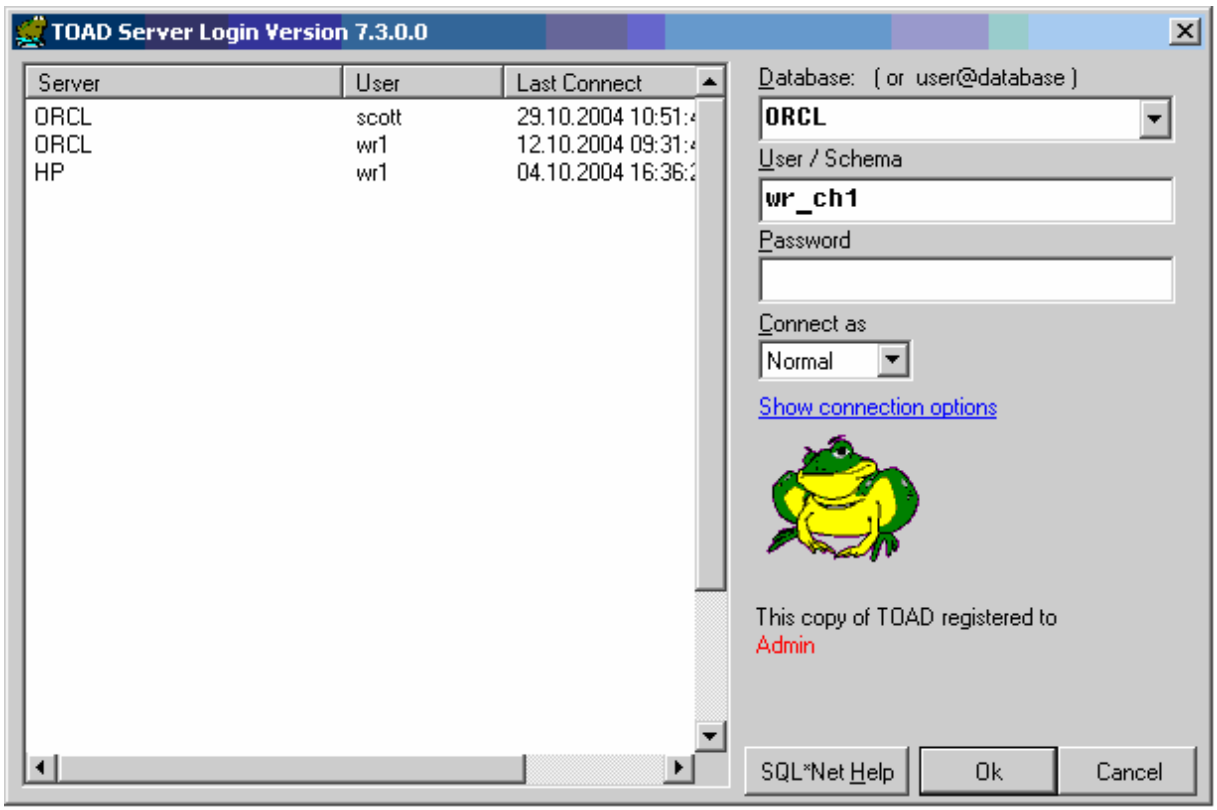

Рис. 1.3. Окно запуска TOAD

TOAD в процессе работы позволяет устанавливать несколько соединений с БД. Для управления соединениями используются следующие пункты меню:

File → New Connection – позволяет установить новое соединение с сервером Oracle;

File → End Connection – закрывает текущее соединение с Oracle.

File → End All Connection – закрывает все соединения с Oracle.

Список открытых соединений отображается на панели, расположенной сверху от строки состояния (см. рис. 1.4). С помощью данной панели пользователь может переключаться между активными соединениями.

Доступ к набору инструментальных средств, используемых при разработке и отладке серверных приложений и схем баз данных осуществляется, преимущественно, с помощью пункта главного меню *Database*. В процессе работы доступны следующие пункты:

Database → Schema Browser – открывает окно браузера, с помощью которого осуществляется доступ к информации об объектах схем, видимых в текущем соединении;

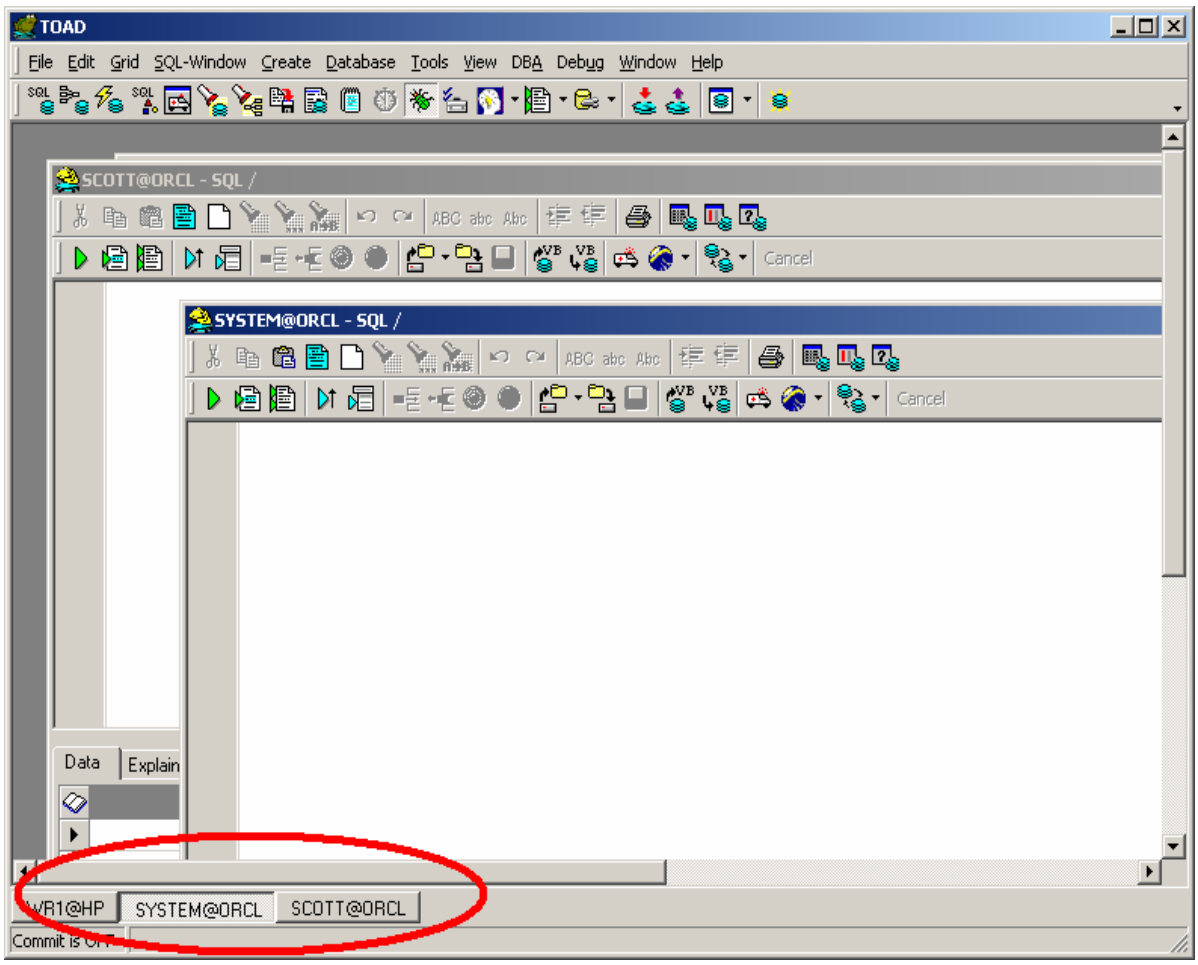

Рис. 1.4. Строка активных соединений

Database → SQL Editor – используется для написания и выполнения команд и скриптов SQL;

Database → Procedure Editor – предоставляет пользователю инструментальные средства для разработки блоков PL/SQL;

Database → SQL Modeller – поддерживает процесс визуального составления SQL-запросов;

Database → Export → ... – предоставляет пользователю возможность выгрузки данных из БД в виде скриптов SQL;

Database → Import → ... – импортирует данные в БД;

Database → Commit – осуществляет фиксацию транзакции в текущем соединении;

Database → Rollback – осуществляет откат транзакции в текущем соединении.

Инструмент *Schema Browser* предназначен для визуализации объектов схемы БД и работы с ними посредством визуального интерфейса. Внешний вид *Schema Browser* показан на рисунке 1.5. С помощью *Schema Browser* пользователь может изменять как данные в таблицах БД, так и структуру БД.

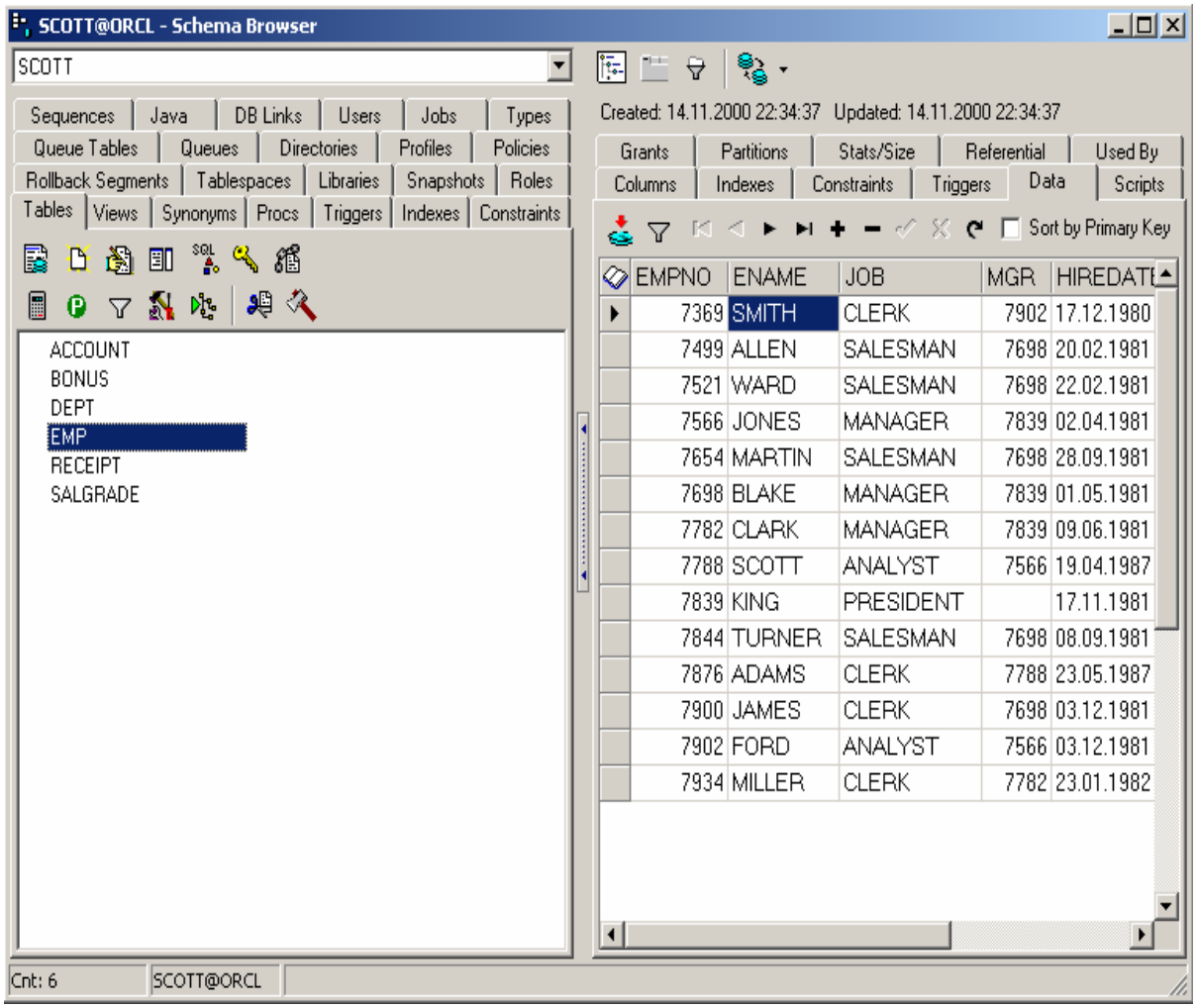

Рис. 1.5. Schema Browser

Инструмент *SQL Editor* предназначен для редактирования и запуска на выполнение скриптов SQL и PL/SQL. Внешний вид окна SQL Editor показан на рисунке 1.6.

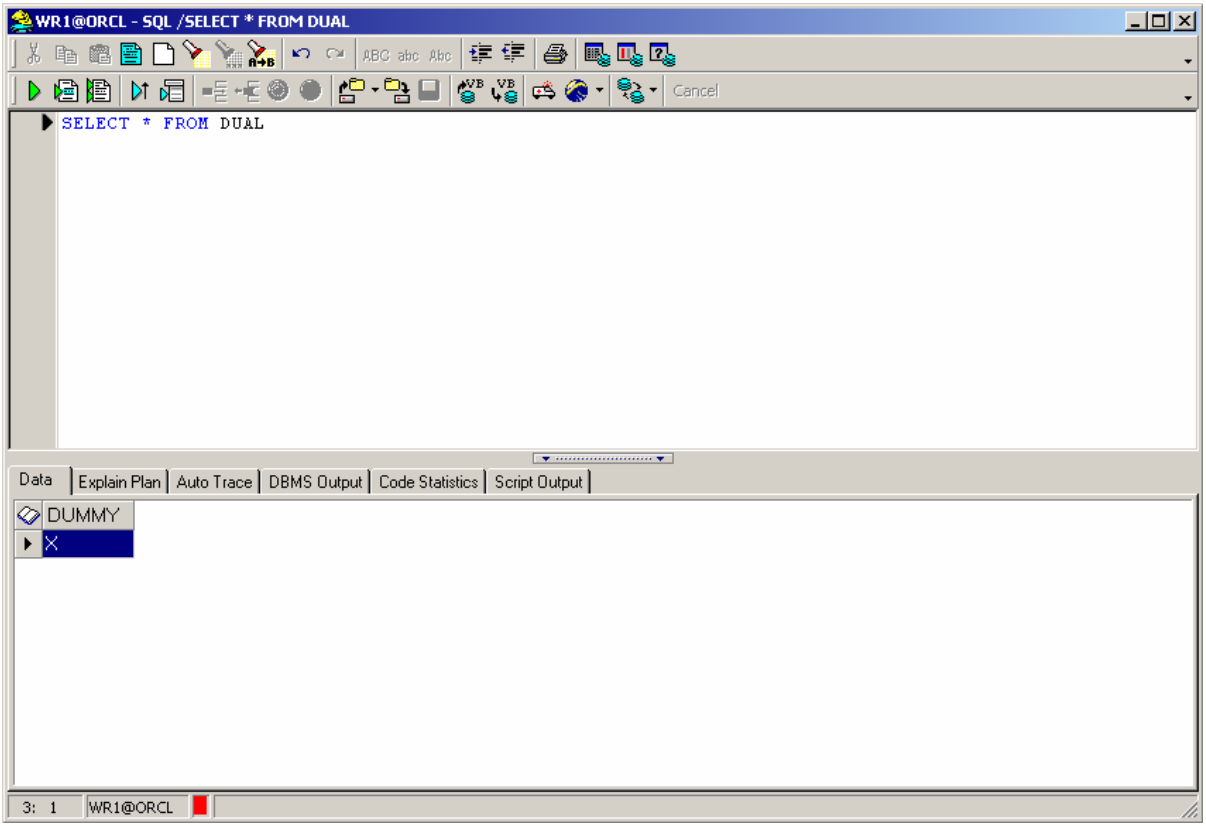

Рис. 1.6. Внешний вид SQL Editor

Пользователь имеет возможность запускать на выполнение как отдельные команды SQL, так и скрипты. В нижней части SQL Editor расположена панель результатов выполнения команд, где пользователь может просмотреть как данные, выбираемые с помощью запросов SQL, так и выбираемые из потоков DBMS Output и Script Output. Также пользователь имеет возможность просмотреть план разбора SQL-выражения, Trace-информацию и статистику выполнения скрипта.

2. КОНСТРУКЦИИ PL/SQL

2.1. Основы синтаксиса. Блоки PL/SQL

Язык PL/SQL является процедурно-ориентированным языком поколения 4GL. Основной программно-лексической единицей языка является логический блок, который может представлять собой процедуру, функцию или анонимный блок.

Схема обработки блоков PL/SQL показана на рисунке 2.1. Ядро PL/SQL является одной из частей сервера Oracle и осуществляет передачу кода PL/SQL компоненту Procedure Statement Executor (исполнитель процедурных команд). Команды SQL, содержащиеся в блоке, передаются для обработки компоненту SQL Statement Executor (исполнитель команд SQL).

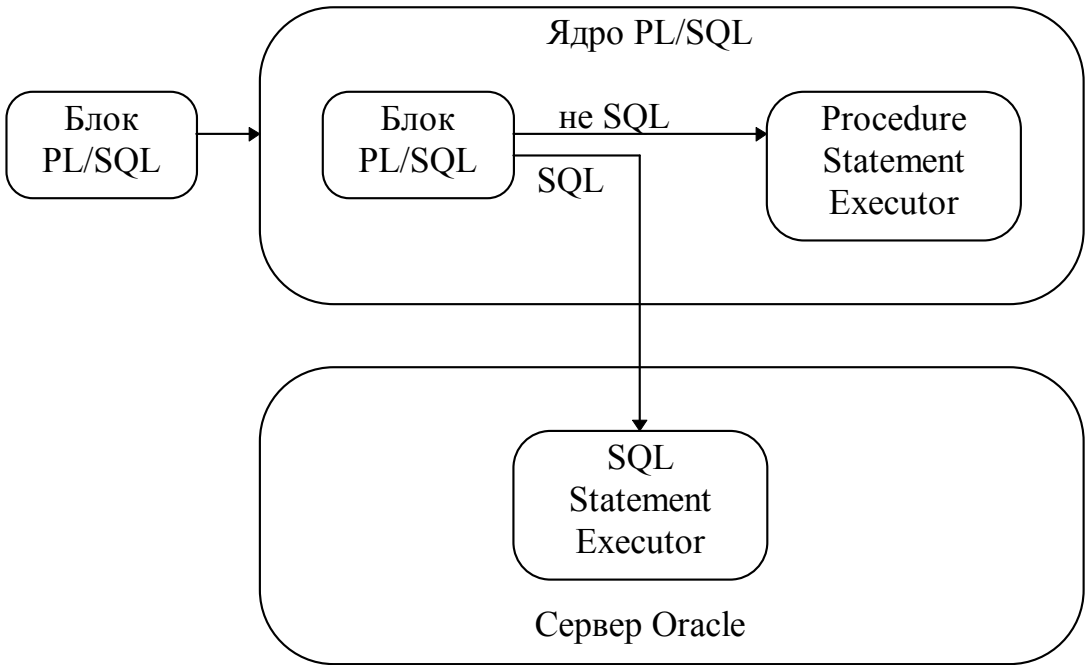

Рис. 2.1. Схема обработки блоков PL/SQL

Программные блоки могут создаваться в различных средах и подразделяются на хранимые подпрограммы и прикладные подпрограммы. Хранимые подпрограммы находятся в базе данных и доступны из любого инструментального средства или приложения базы данных в соответствии

с мерами защиты базы от несанкционированного доступа. Прикладные программы доступны только из приложения, где они созданы.

Каждый блок PL/SQL может включать до трех секций: описательную или декларативную (эта секция необязательна), секцию исполняемого кода (обязательна) и секцию обработки исключений (необязательна). Структура блока PL/SQL:

```
DECLARE - необязательно
  - переменные, константы, курсоры
BEGIN - обязательно
  - команды SQL,
    управляющие команды PL/SQL
EXCEPTION - необязательно
  - действия в случае возникновения ошибок
END; - обязательно
```

Блоки PL/SQL состоят из строк, каждая из которых может содержать следующие лексические элементы:

– разделители (простые и составные);

– идентификаторы (включая зарезервированные слова);

– литералы;

– комментарии.

Разделителем является простой или составной символ, имеющий специальное значение в рамках PL/SQL. Разделителями могут быть символы пробела или перевода строки, символы, обеспечивающие выполнение арифметических или логических операций, и т.д. В таблице 2.1 приведен список простых разделителей PL/SQL.

Таблица 2.1

Список простых разделителей PL/SQL

Символ	Значение
+	Оператор сложения
%	Индикатор атрибута
‘	Ограничитель строки символов
.	Селектор компонентов
/	Оператор деления

Окончание таблицы 2.1

Символ	Значение
(	Выражение или разделительный символ списка
)	Выражение или разделительный символ списка
:	Индикатор хост-переменной
,	Разделитель предметов списка
*	Оператор умножения
“	Разделитель закавыченного идентификатора
=	Оператор сравнения
<	Оператор сравнения
>	Оператор сравнения
@	Оператор удаленного доступа
;	Завершитель выражения
-	Оператор вычитания / отрицания

Идентификаторы используются для именования программных компонентов и модулей, констант, переменных, исключений, курсоров, подпрограмм и пакетов. Идентификатор может состоять из букв алфавита, цифр (не на первой символьной позиции идентификатора), знаков $ и #, подчеркиваний. Использование других символов в обозначении идентификаторов запрещено. Регистр алфавитных символов не учитывается.

Некоторые из идентификаторов, называемые *зарезервированными словами*, имеют специальный смысл в рамках PL/SQL и не могут быть переопределены. Список зарезервированных слов приведен в таблице 2.2.

Таблица 2.2

Список зарезервированных слов

ALL	DESC	ISOLATION	OUT	SQLERRM
ALTER	DISTINCT	JAVA	PACKAGE	START
AND	DO	LEVEL	PARTITION	STDDEV
ANY	DROP	LIKE	PCTFREE	SUBTYPE
ARRAY	ELSE	LIMITED	PLS_INTEGER	SUCCESSFUL
AS	ELSIF	LOCK	POSITIVE	SUM
ASC	END	LONG	POSITIVEN	SYNONYM
AUTHID	EXCEPTION	LOOP	PRAGMA	SYSDATE
AVG	EXCLUSIVE	MAX	PRIOR	TABLE

Окончание таблицы 2.2

BEGIN	EXECUTE	MIN	PRIVATE	THEN
BETWEEN	EXISTS	MINUS	PROCEDURE	TIME
BINARY_ INTEGER	EXIT	MINUTE	PIBLIC	TIMESTAMP
BODY	EXTENDS	MLSLABEL	RAISE	TO
BOOLEAN	FALSE	MOD	RANGE	TRIGGER
BULK	FETCH	MODE	RAW	TRUE
BY	FLOAT	MONTH	REAL	TYPE
CHAR	FOR	NATURAL	RECORD	UID
CHAR_BASE	FORALL	NATURALN	REF	UNION
CHECK	FROM	NEW	RELEASE	UNIQUE
CLOSE	FUNCTION	NEXTVAL	RETURN	UPDATE

2.2. Типы данных PL/SQL

Все константы, переменные и параметры функций в PL/SQL характеризуются *типом*, который описывает формат хранения данных, ограничения и диапазон допустимых значений. PL/SQL предоставляет возможность использовать широкий набор предопределенных типов. В дополнение к этому, есть возможность определения пользовательских типов данных.

Предопределенные типы данных можно условно разбить на четыре разновидности (см. рис. 2.2):

1) скалярные типы (не имеющие внутренних компонентов);

2) составные типы (имеющие внутренние компоненты, с каждым из которых можно работать индивидуально);

3) ссылочные типы;

4) LOB-типы, предназначенные для хранения объектов большого размера (Large Objects types).

Скалярные типы в свою очередь делятся на четыре разновидности:

1) числовые типы;

2) символьные типы;

3) логический тип (BOOLEAN);

4) тип для хранения времени / даты (DATE).

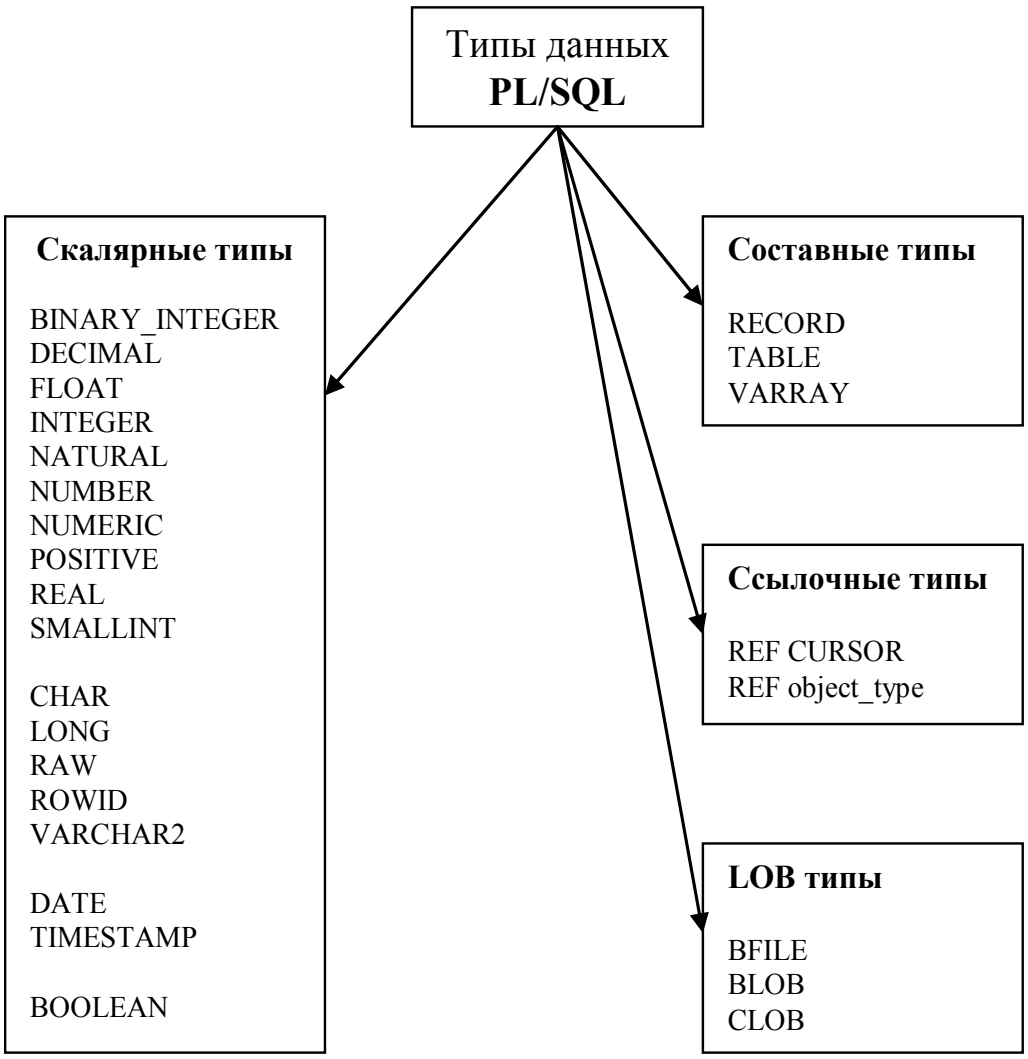

Рис. 2.2. Типы данных PL/SQL

Наиболее часто используемыми из числовых скалярных типов являются следующие типы данных:

BINARY_INTEGER – целое число в диапазоне $-2^{31}...2^{31}$. Его подтипы – NATURAL и POSITIVE представляют соответственно множество натуральных и множество положительных целых чисел.

NUMBER – используется как для хранения целых, так и дробных чисел с требуемой точностью. С помощью него можно описывать числа от 10^{-130} до 10^{125}. Тип NUMBER позволяет явным образом описывать диапазон допустимых числовых значений и их точность. Синтаксис определения следующий:

```
NUMBER(precision, scale)
```

где precision – общее количество разрядов числа, scale – количество разрядов справа от десятичного разделителя. Для целочисленных значений используется следующий синтаксис:

```
NUMBER(precision)
```

Тип NUMBER имеет подтипы DEC, DECIMAL, NUMERIC (для описания числа с фиксированной точкой точностью 38 знаков), DOUBLE PRECISION, FLOAT (фиксированная точность, 126 знаков), INTEGER, INT, SMALLINT (для описания целочисленных значений размером не более 38 знаков).

CHAR – тип, предназначенный для манипуляции символьными данными. Тип CHAR позволяет хранить строку символов фиксированного размера от 1 до 32 767 символов длины. Синтаксис определения типа CHAR:

```
CHAR(length)
```

Если длина явно не указывается, она принимается равной 1.

LONG и **LONG RAW** – типы для хранения символьных строк и объемов бинарных данных размером до 32 760 байт. Для работы с типами столбцов таблиц БД LONG и LONG RAW, позволяющих хранить объемы данных до 2 Гб рекомендуется использовать LOB-типы данных PL/SQL.

RAW – тип для хранения бинарных данных переменного размера с фиксированным максимальным размером не более 32 767 байт. Синтаксис определения типа:

```
RAW(max_length)
```

ROWID – тип данных для хранения физических идентификаторов строк таблиц БД. Каждая строка таблицы в БД имеет такой уникальный идентификатор вне зависимости от наличия первичного ключа и каких-либо уникальных ключей.

VARCHAR2 – тип данных для хранения строки символов переменной длины. Максимальный размер строки определяется пользователем и не

должен превышать 32 767 байт. Для увеличения производительности выделение памяти под данные типа VARCHAR2 производится блоками по 2000 байт, соответственно максимальный размер больших строк рекомендуется устанавливать кратно этому значению. Для типа VARCHAR2 используются псевдонимы STRING и VARCHAR.

LOB-типы предназначены для хранения больших объемов неструктурированных данных (тексты, графика, аудио и видеоданные) размером до 4 Гб.

BFILE – тип, предназначенный для спецификации бинарных файлов, хранящихся вне БД. В качестве дескриптора бинарного объекта используется полный физический путь к файлу (логические пути не поддерживаются). Данные типа BFILE нельзя модифицировать, они открыты только для чтения. С данными такого типа не работают транзакции, они не подлежат репликации и резервированию средствами сервера.

BLOB – тип данных, предназначенный для хранения больших бинарных объектов внутри таблиц БД.

CLOB – используется для хранения больших текстовых массивов, причем поддерживаются кодировки как фиксированного, так и переменного размеров. Общий размер текстового массива не должен превышать 4 Гб.

Для хранения значений даты, времени, а также манипуляций с этими значениями используются следующие типы данных:

DATE – основной тип, с помощью которого хранится информация о дате и времени с точностью до секунды.

TIMESTAMP – тип, расширяющий тип DATE до долей секунды (число знаков в дробной части секунды определяется параметром precision):

```
TIMESTAMP(precision)
```

BOOLEAN – тип, который используется для описания логических переменных. Следует заметить, что переменные типа BOOLEAN могут принимать одно из следующих значений:

– TRUE
– FALSE
– NULL.

Синтаксис PL/SQL позволяет определять тип переменной на основе какой-либо уже заданной переменной, либо на основе типа столбца базы данных. Для этих целей используется атрибут **%TYPE**. Перед атрибутом %TYPE указывается либо имя переменной, либо таблицы и столбец базы данных. На такие переменные не распространяется NOT NULL. Пример использования атрибута %TYPE:

```
v_maximum  NUMBER (7,2);
v_optimal v_maximum%TYPE;
```

Составные типы данных имеют внутренние компоненты и могут использоваться повторно. В PL/SQL имеются следующие составные типы данных:

– TABLE
– RECORD
– VARRAY.

Тип **TABLE** состоит из первичного ключа типа BINARY_INTEGER и столбец скалярного типа. Синтаксис объявления типа на основе типа TABLE:

```
TYPE type_name IS TABLE OF scalar_datatype
[NOT NULL] INDEX BY BINARY_INTEGER;

ident type_name;
```

где type_name – имя типа TABLE, scalar_datatype – тип данных для элементов таблицы PL/SQL, ident – имя идентификатора.

Тип **RECORD** (запись) позволяет рассматривать совокупность одной или нескольких компонент скалярного типа, как логическую единицу. Синтаксис:

```
TYPE type_name IS RECORD
 (field_name1 field_type
[NOT NULL {:=|DEFAULT} expr],
(field_name2 field_type)
[NOT NULL {:=|DEFAULT} expr],...);

ident type_name;
```

где type_name – имя типа RECORD, field_name – имя поля, field_type – тип данных поля, expr – любое выражение PL/SQL, ident – имя идентификатора.

Атрибут %ROWTYPE используется для объявления переменной на основе совокупности столбцов в таблице или представлении базы данных. Перед %ROWTYPE указывается имя таблицы. Пример использования этого атрибута:

```
dept_record s_dept%ROWTYPE;
```

2.3. Управляющие структуры

Для изменения логического потока операций в блоке PL/SQL используются следующие управляющие структуры:

– конструкции условного управления с оператором **IF**;

– простой цикл для безусловного повторного выполнения действий;

– цикл **FOR** для управления количеством повторов по счетчику;

– цикл **WHILE** для управления количеством повторов по результату проверки истинности выражения;

– оператор **EXIT** для выхода из цикла.

По структуре оператор IF в PL/SQL сходен с аналогичными операторами в других языках. Он позволяет выборочно выполнять действия в зависимости от некоторых условий. Синтаксис оператора:

```
IF condition THEN
operators;
[ELSIF condition THEN
 operators; ]
[ELSE]
 operators; ]
END IF;
```

где condition – условие перехода, operators – операторы PL/SQL. Пример использования оператора IF:

```
IF v_start>100 then
  RETURN (2*v_start);
```

```
ELSIF v_start>=50 THEN
  RETURN (.5*v_start);
ELSE
  RETURN (.1*v_start);
END IF;
```

Простое логическое выражение состоит из чисел, строк или дат с операторами сравнения. Как правило, неопределенные значения обрабатываются с помощью оператора IS NULL. Правила обработки NULL:

– любое выражение, содержащее неопределенное значение, дает результат NULL, за исключением выражений, образованных путем конкатенации, где неопределенное значение рассматривается как пустая строка;

– любое простое сравнение, содержащее неопределенное значение, дает результат NULL;

– сравнение IS NULL дает результат TRUE или FALSE.

Сложное логическое выражение состоит из простых логических условий и логических операторов AND, OR и NOT. В приведенных таблицах истинности для этих операторов FALSE имеет более высокий приоритет для оператора AND, а TRUE – для оператора OR.

AND	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

OR	*TRUE*	*FALSE*	*NULL*
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

NOT	*TRUE*	*FALSE*	*NULL*
	FALSE	TRUE	NULL

PL/SQL предоставляет несколько путей организации циклов для повторного выполнения предложений.

Простой цикл – это группа повторно выполняемых операторов, ограниченных операторами LOOP и END LOOP. Каждый раз, как поток управления достигает предложения END LOOP, управление передается LOOP. Чтобы цикл сделать конечным, добавляют оператор EXIT. Синтаксис такого цикла:

```
LOOP
  operator1;
  operator2;
   . . .
  EXIT [WHEN condition];
END LOOP;
```

где operator1, operator2 – операторы PL/SQL, condition – условие выхода. Пример использования такой конструкции:

```
LOOP
  INSERT INTO s_item (ord_id,item_id)
    VALUES (v_ord_id, v_counter);
  v_counter:=v_counter+1;
  EXIT WHEN v_counter>10;
END LOOP;
```

Цикл FOR имеет такую же структуру, как уже рассмотренные циклы. Но ключевому слову LOOP предшествует управляющий оператор, который задает количество повторов цикла.

```
FOR index in [REVERSE]
  low_val..high_val LOOP
operator1;
operator2;
. . .
END LOOP;
```

где index – переменная, являющаяся индексом цикла, low_val..high_val – интервал, в котором изменяется эта переменная, operator1, operator2 – операторы PL/SQL. Объявлять индекс цикла не нужно. Ссылки на индекс разрешены только внутри цикла. Нельзя ссылаться на индекс в качестве переменной в левой части оператора присваивания. Пример использования цикла FOR:

```
FOR i IN v_lower..v_upper LOOP
  v_counter:=v_counter+1;
  v_output:=i;
END LOOP;
```

Цикл WHILE используется для повторения последовательности операторов в течение всего времени, пока выполняется заданное условие. Если условие равно FALSE в начале цикла, последовательность операторов не выполняется. Синтаксис оператора WHILE:

```
WHILE condition LOOP
operator1;
operator2;
. . .
END LOOP;
```

где condition – условие выполнения цикла. Пример такого цикла:

```
WHILE v_counter<=10 LOOP
INSERT INTO s_item (ord_id, item_id)
   VALUES (v_ord_id, v_counter);
v_counter:=v_counter+1;
END LOOP;
```

Циклы могут быть вложены один в другой на несколько уровней. Можно вкладывать циклы FOR в циклы WHILE и наоборот. Обычно завершение вложенного цикла не вызывает завершения внешнего (за исключением случаев, когда возбуждается исключение). Но можно пометить циклы метками и выйти их внешнего цикла с помощью предложения EXIT.

Метки подчиняются тем же правилам, что и остальные идентификаторы. Метка находится перед оператором на той же или отдельной строке. Метки циклов указываются перед словом LOOP в двойных угловых скобках (<<метка>>). Пример использования меток:

```
BEGIN
  <<Outer_loop>>LOOP
      v_counter:=v_counter+1;
      EXIT WHEN v_counter>10;
      <<Inner_loop>>LOOP
          .  .  .
        EXIT Outer_loop WHEN
                  total_done='YES';
        EXIT WHEN inner_done='YES';
          .  .  .
      END LOOP Inner_loop;
          .  .  .
  END LOOP Outer_loop;
END;
```

Помимо традиционных операторов структурного программирования, PL/SQL поддерживает операторы GOTO и NULL. Оператор GOTO передает управление в произвольную часть программного блока, заранее обозначенную меткой.

```
BEGIN
...
GOTO insert_row;
...
<<insert_row>> -- метка
INSERT INTO emp VALUES ...
END;
```

Оператор NULL не выполняет никаких функций и используется в тех местах, где с точки зрения синтаксиса PL/SQL требуется наличие программного блока, но с точки зрения логики программы не должно ничего происходить.

2.4. Обработка исключений

Исключение – это переменная в PL/SQL, возбуждаемая во время выполнения блока и прекращающая выполнение действий в теле блока. Если PL/SQL возбуждает исключение, выполнение блока прерывается всегда, но вы можете указать обработчик исключения, который выполнит некоторые заключительные действия. Методы возбуждения исключения:

– Автоматически в случае возникновения ошибки Oracle.

– Явно с помощью предложения RAISE.

Если исключение возбуждается в выполняемой секции блока, управление передается соответствующему обработчику исключений в секции обработки исключений блока (секции EXCEPTION). Если PL/SQL успешно обработает исключение, оно не распространяется во внешний блок или среду. Если же исключение возбуждено в выполняемой секции блока, а соответствующего обработчика исключений нет, выполнение блока PL/SQL прекращается в аварийном порядке.

Различают три типа исключений:

– Предопределенное, возбуждаемое сервером Oracle. Одна из примерно 20 типичных ошибок, возникающих при выполнении программ PL/SQL. Описание не требуется.

– Непредопределенное, возбуждаемое сервером Oracle. Любая другая ошибка сервера Oracle. Требуется описание в декларативной секции.

– Пользовательское. Условие, которое разработчик считает ненормальным. Необходимо определять в декларативной секции и возбуждать явно.

Синтаксис описания исключений:

```
EXCEPTION
     WHEN exception1
          [OR exception2 ...]
     THEN
           operator1;
           operator2;
           .  .  .
     WHEN exception3
          [OR exception4 ...]
     THEN
```

```
        operator1;
        operator2;
        .  .  .
    WHEN OTHERS
    THEN
        operator1;
        operator2;
        .  .  .]
```

где exception1..N – описываемые исключения, operator1..N – операторы PL/SQL для обработки соответствующих исключений.

Примеры некоторых предопределенных исключений Oracle:

– NO_DATA_FOUND (ORA-01403). Команда SELECT на выборку одной строки не вернула данных.

– TOO_MANY_ROWS (ORA-01422). Команда SELECT на выборку одной строки вернула более одной строки.

– INVALID_CURSOR (ORA-01001). Недопустимая операция с курсором.

– ZERO_DIVIDE (ORA-01476). Попытка деления на ноль.

– DUP_VAL_ON_INDEX (ORA-00001). Попытка вставки дубликата значения в столбец, для которого имеется уникальный индекс.

Пример процедуры с блоком обработки предопределенных исключений:

```
PROCEDURE elim_inventory
   (v_product_id IN s_product.id%TYPE) IS
    v_id s_product.id%TYPE;
BEGIN
  SELECT id
    INTO v_id
    FROM s_product
    WHERE id=v_product_id;
  DELETE FROM s_inventory
    WHERE product_id=v_product_id;
  COMMIT;
EXCEPTION
   WHEN NO_DATA_FOUND THEN
```

```
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE(
                    TO_CHAR(v_product_id)
                    ||' is invalid.');
    WHEN TOO_MANY_ROWS THEN
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE(
                    'Data corruption!');
    WHEN OTHERS THEN
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Other error
                           occured.');
END;
```

Для перехвата непредопределенного исключения необходимо объявить его в декларативной секции блока. После этого объявленное исключение будет возбуждаться неявно. Синтаксис объявления исключения в декларативной секции:

```
except EXCEPTION;
```

где except – имя исключения. Связывание объявленного исключения со стандартным номером ошибки Oracle осуществляется следующим образом:

```
PRAGMA EXCEPTION_INIT (except, err_num);
```

где except – имя исключения, err_num – номер ошибки сервера Oracle.

Для перехвата пользовательского исключения необходимо сначала объявить его, а затем возбудить явно. Синтаксис объявления пользовательского исключения аналогичен синтаксису объявления непредопределенного исключения. Возбуждение исключения осуществляется с помощью предложения RAISE:

```
RAISE except;
```

где except – имя исключения.

Если возбуждается исключение, то его код и связанное с ним сообщение можно определить с помощью двух функций:

SQLCODE – возвращает числовое значение кода ошибки;

SQLERRM – возвращает сообщение, связанное с кодом ошибки.

Если блок или подблок PL/SQL обрабатывает исключение, он завершается нормально. Если же PL/SQL возбуждает исключение, а в текущем блоке нет соответствующего обработчика, исключение распространяется последовательно на внешние блоки, пока не будет найден обработчик. Если ни один из блоков не обрабатывает исключение, это вызывает необработанное исключение в хост-среде. Если исключение распространяется во внешний блок, оставшиеся исполняемые операторы в этом блоке пропускаются.

2.5. Процедуры, функции, пакеты

Процедуры PL/SQL – это подпрограммы, выполняющие какие-либо действия. Процедуры могут быть как хранимые (расположенные на сервере БД), так и прикладные – доступные только из того приложения, где они созданы.

Существуют следующие команды SQL для создания хранимой процедуры:

CREATE [**OR REPLACE**] **PROCEDURE** – создает процедуру;

DROP PROCEDURE – удаляет процедуру;

Синтаксис описания процедуры имеет следующий вид:

```
PROCEDURE name
    [(par1,...,parN)]
IS
блок_PL/SQL;
```

где name – имя процедуры, par1,...,parN – параметры, которые используются для передачи значений из вызывающей среды и обратно. Существует два типа параметров. При объявлении процедуры указываются формальные параметры, которые служат для определения значений в исполняемой части блока PL/SQL. Фактические параметры или аргументы подставляются при вызове подпрограммы.

Синтаксис объявления параметров имеет следующий вид:

```
par_name [IN|OUT|IN OUT] type
         [{:=|DEFAULT} expr];
```

где par_name – имя параметра, type – его тип, expr – выражение PL/SQL.

Виды формальных параметров:

IN – входной. Передает значение из вызывающей среды в подпрограмму. Выступает в качестве константы, может быть выражением, константой, литералом или инициализированной переменной.

OUT – выходной. Возвращает значение из процедуры в вызывающую среду. Не может быть присвоен другой переменной или самому себе. Фактический параметр должен быть переменной и не может быть константой или выражением.

IN OUT – входной / выходной. Передает значение из вызывающей среды в процедуру и возвращает значение из процедуры в вызывающую среду. Выступает в качестве инициализированной переменной.

Пример процедуры PL/SQL:

```
PROCEDURE change_salary
  (v_emp_id IN NUMBER,
   v_new_salary IN NUMBER)
IS
BEGIN
   UPDATE s_emp
      SET salary=v_new_salary
    WHERE id=v_emp_id;
  COMMIT;
END change_salary;
```

Функции имеют два основных отличия от процедуры:

– функция вызывается как часть выражения;

– функция должна возвращать значение.

Существуют функции, определяемые пользователем, и функции SQL. Функции PL/SQL используются для возврата значения в вызывающую среду.

Существуют следующие команды SQL для создания хранимой функции:

CREATE [**OR REPLACE**] **FUNCTION** – создает процедуру;

DROP FUNCTION – удаляет процедуру;

Синтаксис объявления функции имеет следующий вид:

```
FUNCTION name
    [(par1,...,ParN)]
    RETURN data_type
IS
блок_pl/sql;
```

где name – имя функции, par1...parN – параметры вызова, data_type – тип возвращаемого функцией значения.

Особенностью функции является оператор RETURN. Тип возвращаемого значения должен обязательно совпадать с типом данных, указанным в предложении RETURN определения функции. Операторов RETURN может быть несколько, но при каждом вызове будет исполняться только один из них. Функции можно использовать в выражениях SQL, однако существуют ограничения:

1) функция работает с данными одной строки;

2) функция не может содержать команды DML;

3) все формальные параметры должны быть входными;

4) типы данных (в т. ч. и для RETURN) должны быть внутренними типами сервера Oracle;

Пример функции PL/SQL:

```
FUNCTION tax
   (value IN NUMBER)
   RETURN NUMBER
IS
BEGIN
   RETURN (value*.07);
END tax;
```

Пакеты позволяют объединять код логически связанных между собой процедур и функций PL/SQL. Также пакеты позволяют разрабатывать интерфейс приложения, отделенный от внутреннего кода процедуры. В

спецификации пакета описываются все публичные объявления и список общедоступных подпрограмм. Создать пакет можно с помощью команд

```
CREATE [OR REPLACE] PACKAGE name
AS
 <package specification>
END;

CREATE [OR REPLACE] PACKAGE BODY name
AS
  <package body>
END;
```

где name – имя пакета, package specification – спецификация пакета, package body – тело пакета. Синтаксис спецификации пакета:

```
PACKAGE name IS
   -- public declarations
   -- subprogram specifications
END [name];
```

где public declarations – раздел публичных объявлений пакета, subprogram specifications – спецификации входных и выходных параметров подпрограмм пакета. Синтаксис определения тела пакета:

```
PACKAGE BODY name IS
   -- private declarations
   -- subprogram definitions
[BEGIN
    -- initialization statements]
END [name];
```

где private declarations – локальные объявления, subprogram definitions – описания подпрограмм, initialization statements – действия, выполняемые при инициализации пакета. Пример определения пакета:

```
PACKAGE emp_actions IS
p_salary NUMBER(7,2);
```

```
p_jobtype VARCHAR2(15);
PROCEDURE fire_employee (emp_id NUMBER);
END;

PACKAGE BODY emp_actions IS
p_local BOOLEAN;
PROCEDURE fire_employee (emp_id NUMBER) IS
BEGIN
DELETE FROM s_emp WHERE empno=emp_id;
END fire_employee;
END emp_actions;
```

Пакеты являются достаточно мощным средством для применения объектно-ориентированного проектирования в разработке блоков PL/SQL. Пакеты обеспечивают инкапсуляцию, сокрытие информации и перегрузку функций. Пример перегрузки функций:

```
FUNCTION insert_emp
 (first_name in VARCHAR2,
  last_name in VARCHAR2)
    RETURN NUMBER;

FUNCTION insert_emp
 (first_name in VARCHAR2,
  last_name in VARCHAR2,
  title in VARCHAR2)
    RETURN NUMBER;
```

Перегрузка неприменима в автономных процедурах и функциях.

3. ВЗАИМОДЕЙСТВИЕ С ORACLE

3.1. Операторы DML

PL/SQL в полном объеме поддерживает команды DML и команды управления транзакциями SQL. Команды DDL и DCL не поодерживаются в PL/SQL. Эти команды могут выполняться с помощью пакета DBMS_SQL.

Для выборки данных из базы используется предложение SELECT. Формат команды:

```
SELECT sel_list
INTO var_name | rec_name
FROM table
WHERE condition;
```

где sel_list – список, содержащий по крайней мере один столбец, var_name – скалярная переменная для хранения возвращаемого значения, rec_name – запись PL/SQL для хранения возвращаемых значений, table – имя таблицы базы данных, condition – условие выборки. Команда SELECT должна возвращать только одну строку. Пример использования команды SELECT:

```
dept_record s_dept%ROWTYPE;

SELECT * INTO dept_record
FROM s_dept WHERE id=v_dept_id;
```

При выполнении команды SELECT могут возникнуть следующие исключения:

TOO_MANY_ROWS – выбрано более одной строки;

NO_DATA_FOUND – не выбрано ни одной строки.

Обработка данных в базе осуществляется с помощью команд DML, а именно – INSERT, UPDATE и DELETE.

Для каждой команды SQL сервер выделяет участок памяти, в котором эта команда интерпретируется и выполняется. Эта область называется курсором. Когда исполняемая часть блока выполняет команду SQL,

PL/SQL создает неявный курсор с идентификатором SQL. Существуют атрибуты, которые позволяют увидеть результат последнего неявного курсора:

SQL%ROWCOUNT – целое количество строк, выбранных последней командой SQL;

SQL%FOUND – логический атрибут, имеющий значение TRUE, если последней командой SQL была выбрана хотя бы одна строка;

SQL%NOTFOUND – имеет значение TRUE, если последней командой не было выбрано ни одной строки;

SQL%ISOPEN – всегда FALSE (так как PL/SQL всегда закрывает неявные курсоры сразу после их выполнения).

Пример процедуры, выполняющей удаление строк из таблицы и использующей атрибуты курсора:

```
PROCEDURE del_rows (v_ord_id NUMBER)
IS
  v_rows_deleted NUMBER;
BEGIN
   DELETE FROM s_item
        WHERE ord_id=v_ord_id;
 v_rows_deleted:=SQL%ROWCOUNT;
 DBMS_OUTPUT.PUT_LINE(
                TO_CHAR(v_rows_deleted)
                ||' rows deleted.');
END;
```

Управлять логикой транзакций можно с помощью команд COMMIT и ROLLBACK языка SQL. Эти действия могут происходить как в самом блоке PL/SQL, так и в результате событий, происходящих во внешней среде. Форматы команд:

```
COMMIT [WORK];

ROLLBACK [WORK] TO
         [SAVEPOINT] savepoint_name;

SAVEPOINT savepoint_name;
```

где savepoint_name – имя точки отката.

3.2. Обработка запросов с использованием курсоров

Для выполнения команд SQL и хранения информации об их обработке сервер Oracle использует рабочие области, называемые «личными областями SQL». Курсоры PL/SQL позволяют присвоить имя такой личной области и обращаться к информации, которая в ней хранится. Курсор управляет всеми фазами обработки команды. Существуют неявные и явные курсоры. Функции явных курсоров:

– Поочередная обработка строк, возвращаемых запросом.

– Отслеживание текущей обрабатываемой строки.

– Ручное управление курсорами в блоке PL/SQL программистом.

Процесс использования курсора можно разделить на следующие шаги (см. рис. 3.1):

1. Описание курсора. Курсор описывается путем присвоения ему имени и определения структуры запроса, который будет в нем выполняться.

2. Открытие курсора. Команда OPEN выполняет запрос и связывает все используемые переменные. Строки, выбранные запросом, называются активным множеством и доступны теперь для выборки.

3. Выборка данных курсора. Команда FETCH загружает текущую строку из курсора в переменные. Каждое выполнение команды FETCH перемещает указатель курсора на следующую строку активного множества. Следовательно, каждая команда FETCH получает доступ к разным строкам, выданным запросом.

4. Закрытие курсора. Команда CLOSE освобождает активное множество строк. После этого можно вновь открыть курсор для создания нового активного набора строк.

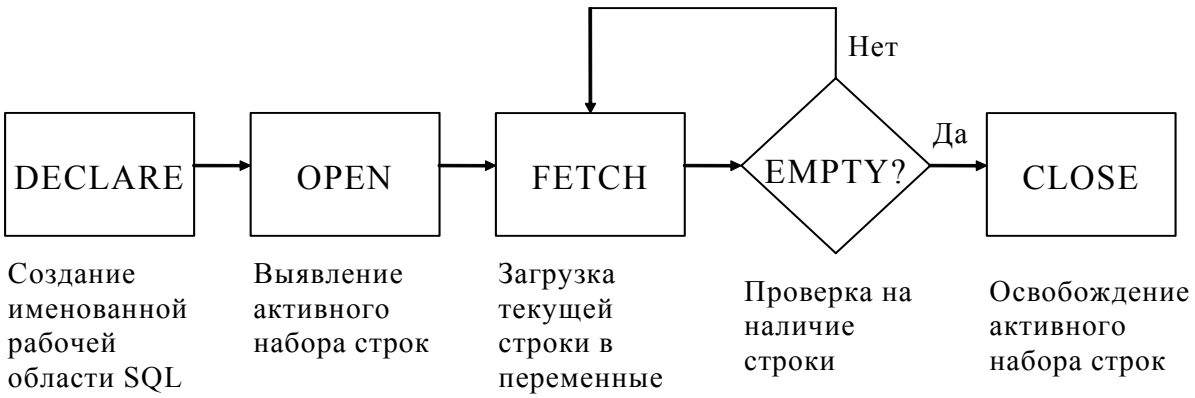

Рис. 3.1. Процесс использования курсора

Для описания явного курсора используется оператор CURSOR. Можно задать параметры для подстановки переменных в запрос при открытии курсора. Можно также ссылаться на переменные в запросе, но они должны быть объявлены до использования оператора CURSOR. Синтаксис оператора:

```
DECLARE
  CURSOR cursor_name IS
  select_statement;
```

где select_statement – SELECT без предложения INTO, cursor_name – имя курсора.

При открытии курсора выполняется запрос и формируется активное множество строк. Это происходит после указания значений входных переменных. Теперь курсор указывает на первую строку в активном наборе. Синтаксис команды открытия курсора:

```
OPEN cursor_name;
```

где cursor_name – имя курсора.

Команда FETCH используется для выборки значений из текущей строки в выходные переменные. После выборки можно работать с переменными с помощью других команд. Синтаксис команды FETCH:

```
FETCH cursor_name INTO variable1,
                       variable2, ...;
```

где cursor_name – имя курсора, variable1, variable2... – имена переменных, в которые осуществляется выборка данных.

В предложение INTO команды FETCH следует включать столько же переменных, сколько столбцов возвращает команда SELECT. Следует также проверять совместимость типов данных. Между именами переменных и столбцами устанавливается позиционное соответствие. Команда FETCH позволяет проверить, содержит ли курсор строки. Если команда FETCH не выбирает никаких значений, это значит, что в активном наборе не осталось необработанных строк, и состояние ошибки не возникает.

Закрытие курсора осуществляется командой CLOSE имя_курсора.

Атрибуты явного курсора:

%ISOPEN – значение равно TRUE, если курсор открыт;

%NOTFOUND – значение равно TRUE, если последняя команда FETCH не вернула строку;

%FOUND – значение равно TRUE, если FETCH возвращает строку. Обратен атрибуту NOTFOUND.

%ROWCOUNT – возвращает общее количество строк, выбранных на данный момент.

Обычно для обработки нескольких строк из явного курсора создается цикл, при каждом выполнении которого выбирается одна строка. В конце концов обрабатываются все строки активного набора, и в результате неудачной выборки атрибут %NOTFOUND принимает значение TRUE («истинно»). Прежде чем ссылаться на курсор, проверяйте успех каждой выборки с помощью атрибутов явного курсора.

Выборка строк возможна только при открытом курсоре (это можно проверить с помощью атрибута %ISOPEN). Для выборки точного количества строк можно либо создать цикл FOR с числовым параметром, либо использовать простой цикл и определять момент выхода из цикла с помощью атрибута %ROWCOUNT. Пример использования курсора:

```
PROCEDURE ord_process
  (v_ord_id IN s_item.ord%TYPE)
IS
  v_product_id s_item.product_id%TYPE;
  v_item_total  NUMBER (11,2);
  v_order_total NUMBER (11,2):=0;
  CURSOR item_cursor IS
    SELECT product_id, price * quantity
    FROM s_item
    WHERE ord_id=v_ord_id;
BEGIN
  OPEN item_cursor
  LOOP
    FETCH item_cursor
      INTO v_product_id, v_item_total;
```

```
    EXIT WHEN item_cursor%ROWCOUNT > 5
         OR item_cursor%NORFOUND;
    v_order_total:=v_order_total +
                    v_item_total;
DBMS_OUTPUT.PUT_LINE('Сумма по товару  '||
' равна '||TO_CHAR(v_order_total,
                  '$999,999,999.99'));
  END LOOP;
  CLOSE item_cursor
END ord_process;
```

На основе выборочного списка столбцов в явном курсоре можно определить запись. Этот способ удобен для обработки строк активного набора, поскольку позволяет производить выборку данных в запись. Пример использования записи:

```
CURSOR emp_cursor IS
     SELECT id, salary, start_date, rowid
     FROM s_emp
     WHERE dept_id=41;
emp_record emp_cursor%ROWTYPE;
BEGIN
     OPEN emp_cursor;
    .  .  .
  FETCH emp_cursor INTO emp_record;
```

Можно передавать значения курсору в момент открытия с помощью параметров. Синтаксис создания курсора с параметром:

```
CURSOR cursor_name
[(par_name data_type, ...)]
IS select_statement;
```

где cursor_name – имя курсора, par_name – имя параметра, data_type – тип параметра, select_statement – предложение SELECT. Пример использования курсора с параметром:

```
CURSOR emp_cursor
(v_dept NUMBER, v_job VARCHAR2) IS
SELECT last_name, salary, start_date
FROM s_emp
WHERE dept_id=v_dept
AND title=v_job;
```

Циклы FOR с курсорами обрабатывают строки в явном курсоре. Это сокращенная форма записи, поскольку происходит открытие курсора, выборка строк по одной за каждое прохождение цикла и автоматическое закрытие курсора после того, как все строки обработаны. Синтаксис открытия цикла FOR с курсором:

```
FOR record_name IN cursor_name LOOP
    operator1;
    operator2;
 . . .
END LOOP;
```

где record_name – имя записи, cursor_name – имя курсора, operator1, operator2... – операторы PL/SQL.

Пример использования цикла FOR с курсором:

```
FOR item_record IN item_cursor LOOP
  v_order_total:=v_order_total +
         (item_record.price*
          item_record.quantity);
  i:=i+1;
  product_id_table(i):=
     item_record.product_id;
  order_total_table (i):=v_order_total;
END LOOP;
```

Прежде чем удалять или обновлять строки, вы можете пожелать их заблокировать. Для этого в запрос, выполняемый в курсоре, включается предложение FOR UPDATE. Не фиксируйте изменения между выборками

из явного курсора, т. к. сервер Oracle снимает блокировку после завершения транзакции. Синтаксис предложения блокировки строк:

```
SELECT...FROM...FOR UPDATE
[OF column_reference] [NOWAIT];
```

Кроме того, можно использовать команду DELETE или UPDATE с предложением WHERE CURRENT OF имя_курсора для ссылки на последнюю строку, выбранную командой FETCH. Если используется это предложение, курсор, на который делается ссылка, должен существовать и содержать в запросе предложение FOR UPDATE. В противном случае возникнет ошибка. Это предложение позволяет производить обновления и удаления в текущей строке без явной ссылки на псевдостолбец ROWID.

3.3. Триггеры БД

Триггеры – это хранимые блоки PL/SQL, запуск которых осуществляется автоматически при наступлении какого-либо события.

Триггеры можно разделить на два типа:

– DML-триггеры, срабатывающие при выполнении команд DML к таблицам данных;

– INSTEAD OF-триггеры для работы с представлениями.

– Системные триггеры, принадлежащие непосредственно БД или конкретной схеме и реагирующие на системные события (команды DML, DDL и т. д.)

Создать DML-триггер на таблицу можно с помощью команды **CREATE TRIGGER**:

```
CREATE [OR REPLACE] TRIGGER trigger_name
(BEFORE|AFTER)
(INSERT|UPDATE|DELETE)
   [OR (INSERT|UPDATE|DELETE) …]
ON table
[FOR EACH ROW]
[WHEN (condition)]
Блок PL/SQL
```

где trigger_name – имя триггера, table – имя таблицы, condition – дополнительное условие срабатывания триггера. Предложение BEFORE/AFTER указывает на характер выполнения триггера (до / после произведенной операции); INSERT|UPDATE| DELETE – указывает DML-операцию, которая обеспечивает вызов триггера (есть возможность указания нескольких операций через ключевое слово OR); модификатор FOR EACH ROW указывает, что триггер должен выполняться для каждой измененной строки таблицы (в противном случае триггер выполняется один раз за операцию). В триггерах FOR EACH ROW имеется возможность доступа к данным строки через внешние переменные new и old (измененные, и данные до изменения соответственно).

Пример триггера:

```
CREATE OR REPLACE TRIGGER salary_changes
BEFORE DELETE OR INSERT OR UPDATE
  ON Emp_tab
FOR EACH ROW
WHEN (new.Empno > 0)
DECLARE
  sal_diff number;
BEGIN
  sal_diff := :new.sal - :old.sal;
  dbms_output.put('Old salary: '
                  || :old.sal);
  dbms_output.put(' New salary: '
                  || :new.sal);
  dbms_output.put_line(' Difference '
                  || sal_diff);
END;
```

Триггеры INSTEAD OF предназначены для реализации пользовательской логики во время операций INSERT, UPDATE или DELETE, примененных к *представлениям*. Триггеры INSTEAD OF обязательно отрабатывают для каждой изменяемой строки (режим FOR EACH ROW). Пример использования триггеров INSTEAD OF:

```
-- создание тестовых таблиц
CREATE TABLE Project_tab (
  Prj_level NUMBER,
  Projno NUMBER,
  Resp_dept NUMBER);

CREATE TABLE Emp_tab (
  Empno NUMBER NOT NULL,
  Ename VARCHAR2(10),
  Job VARCHAR2(9),
  Mgr NUMBER(4),
  Hiredate DATE,
  Sal NUMBER(7,2),
  Comm NUMBER(7,2),
  Deptno NUMBER(2) NOT NULL);

CREATE TABLE Dept_tab (
  Deptno NUMBER(2) NOT NULL,
  Dname VARCHAR2(14),
  Loc VARCHAR2(13),
  Mgr_no NUMBER,
  Dept_type NUMBER);

-- создание view на нескольких таблицах
CREATE OR REPLACE VIEW manager_info AS
SELECT e.ename, e.empno, d.dept_type,
       d.deptno, p.prj_level, p.projno
FROM Emp_tab e, Dept_tab d, Project_tab p
WHERE e.empno = d.mgr_no
AND d.deptno = p.resp_dept;


-- пример триггера
CREATE OR REPLACE TRIGGER m_info_insert
INSTEAD OF INSERT
ON manager_info
REFERENCING NEW AS n
```

```
FOR EACH ROW
DECLARE
  rowcnt NUMBER;
BEGIN
  SELECT COUNT(*) INTO rowcnt
    FROM Emp_tab
    WHERE empno = :n.empno;
  IF rowcnt = 0 THEN
    INSERT INTO Emp_tab (empno,ename)
      VALUES (:n.empno, :n.ename);
  ELSE
    UPDATE Emp_tab
      SET Emp_tab.ename = :n.ename
    WHERE Emp_tab.empno = :n.empno;
  END IF;

  SELECT COUNT(*) INTO rowcnt
    FROM Dept_tab
    WHERE deptno = :n.deptno;

  IF rowcnt = 0 THEN
    INSERT
      INTO Dept_tab (deptno,  dept_type)
      VALUES(:n.deptno, :n.dept_type);
  ELSE
    UPDATE Dept_tab
    SET Dept_tab.dept_type = :n.dept_type
    WHERE Dept_tab.deptno = :n.deptno;
  END IF;

END;
```

3.4. Динамический SQL

При создании приложений PL/SQL часто возникает ситуация, когда структура SQL-запросов заранее не определена и может изменяться в зависимости от каких-либо условий. Для решения ситуаций подобного типа используется механизм, называемый *динамический SQL*. Суть этого механизма заключается в том, что запросы внутри блоков PL/SQL формируются в виде текстовых строк непосредственно во время работы программы, что позволяет получить более широкие возможности. В частности, механизм динамического SQL позволяет:

1. Выполнять операторы DDL, либо другие операторы, выполнение которых запрещено в статическом коде PL/SQL.

2. Реализовать более гибкий подход в выполнении операторов DML за счет их формирования «на лету».

3. Использовать функции пакета DBMS_SQL для увеличения производительности запросов.

Для выполнения запросов, которые не возвращают данных, либо возвращают одиночную строку используется конструкция EXECUTE IMMEDIATE (начиная с Oracle 8i). Ее синтаксис:

```
EXECUTE IMMEDIATE dynamic_string
[INTO {define_variable[,
        define_variable]... | record}]
[USING [IN | OUT | IN OUT] bind_argument
[, [IN | OUT | IN OUT] bind_argument]...]
[{RETURNING | RETURN} INTO
       bind_argument[, bind_argument]...];
```

где dynamic_string – SQL-запрос либо анонимный PL/SQL блок; define_variable – переменные, через которые осуществляется связь с выполняемым запросом; bind_argument – список привязываемых аргументов. Выражение RETURNING INTO используется только для DML-выражений.

Примеры использования конструкции:

```
EXECUTE IMMEDIATE 'DROP TABLE :tab'
   USING table_name;
```

```
sql_stmt := 'INSERT INTO payroll
          VALUES (:x, :x, :y, :x)';
EXECUTE IMMEDIATE sql_stmt
   USING a, a, b, a;
```

Для выполнения SQL-запросов, возвращающих наборы данных, используется процесс, состоящий из следующих шагов:

1. Открытие явного курсора (OPEN FOR).
2. Построчная выборка данных (FETCH).
3. Закрытие (CLOSE).

Синтаксис открытия явного курсора следующий:

```
OPEN {cursor_variable |
                    :host_cursor_variable}
 FOR dynamic_string
[USING bind_argument[, bind_argument]...];
```

где cursor_variable – переменная курсора, bind_argument – аргументы связывания с запросом.

Синтаксис выборки данных:

```
FETCH {cursor_variable |
        :host_cursor_variable}
  INTO {define_variable[,
         define_variable]... | record};
```

где define_variable – переменные, куда осуществляется выборка данных.

Синтаксис закрытия курсора:

```
CLOSE {cursor_variable |
            :host_cursor_variable};
```

Пример работы механизмов динамического SQL с запросом, возвращающим набор строк:

```
DECLARE
  TYPE EmpCurTyp IS REF CURSOR;
```

```
    -- тип REF CURSOR
  emp_cv EmpCurTyp; -- определение курсора
  my_ename VARCHAR2(15);
  my_sal NUMBER := 1000;
BEGIN
  OPEN emp_cv FOR – открытие курсора
  'SELECT ename, sal FROM emp WHERE sal > :s'
    USING my_sal;
  LOOP
    FETCH emp_cv INTO my_ename, my_sal;
      -- выборка строки
    EXIT WHEN emp_cv%NOTFOUND;
      -- выход на последней строке

    -- здесь обрабатывается выбранная строка
  END LOOP;
CLOSE emp_cv; -- закрытие курсора
END;
```

3.5. Взаимодействие с Java

СУБД Oracle имеет достаточно широкие возможности интеграции со средой разработки Java. В частности, для формирования слоя серверной логики на основе связки Java+Oracle можно использовать следующие варианты архитектур:

– Хранимые процедуры Java.

– Сервлеты.

– JSP.

– CORBA-объекты.

– EJB.

Вариант, когда код Java представляется в виде хранимых процедур сервера Oracle, является единственным, в котором общее управление приложением возлагается на СУБД, а не на приложение Java.

Процесс загрузки Java-классов на сервер Oracle состоит из следующих этапов:

1. Создание классов Java, определение их методов, компиляция.

```
public class Hello
{
public static String world ()
{
return "Hello world";
}
}
```

2. Загрузка классов на сервер с использованием утилиты loadjava:

```
loadjava –user scott/tiger Hello.class
```

3. Публикация методов загруженных классов в соответствии со спецификациями вызовов:

```
SQL> connect scott/tiger
connected
SQL> create or replace function HELLOWORLD
2 return VARCHAR2 as language java name
3 'Hello.world () return java.lang.String';
4 /
Function created.
```

После выполнения этих действий методы Java-классов можно вызывать так же, как любые хранимые процедуры и функции PL/SQL:

```
SQL> variable myString varchar2[20];
SQL> call HELLOWORLD() into :myString;
Call completed.
SQL> print myString;
```

Для доступа к данным из Java-методов можно использовать технологии JDBC (эмулируемую серверной Java-машиной) либо SQLJ (технология размещения операторов SQL непосредственно внутри Java-кода).

JDBC – стандартная технология доступа к БД из Java. Для использования JDBC нужно выполнить следующие действия:

1. Импортировать необходимые пакеты

```
import java.sql.*; -- стандартные функции
import java.math.*; -- для типов
                    BigDecimal и BigInteger
```

2. Зарегистрировать драйвер Oracle (1 раз в приложении):

```
DriverManager.registerDriver
        (new oracle.jdbc.OracleDriver());
```

3. Открыть соединение с Oracle (при этом для драйвера OCI необходимо специфицировать базу данных как TNS-сервис):

```
Connection conn =
  DriverManager.getConnection
    ("jdbc:oracle:oci8:@MyHostString",
      "scott", "tiger");
Connection conn =
  DriverManager.getConnection
    ("jdbc:oracle:oci8:
      @(description=(address=
        (host= myhost)(protocol=tcp)
        (port=1521))
    (connect_data=(sid=orcl)))",
      "scott", "tiger");
```

Для использования JDBC Thin драйвера надо явно указывать имя хоста, порт и SID базы данных.

```
Connection conn =
  DriverManager.getConnection
    ("jdbc:oracle:thin:@myhost:1521:orcl",
      "scott", "tiger");

Connection conn =
  DriverManager.getConnection
  ("jdbc:oracle:thin:
```

```
        @(description=(address=(host=myhost)
          (protocol=tcp)(port=1521))
         (connect_data=(sid=orcl)))",
       "scott", "tiger");
```

Если процедура находится на сервере, соединение установлено по умолчанию. В этом случае надо использовать DriverManager.getConnection, либо OracleDriver.defaultConnection для определения соединения. Пример работы с DefaultConnection:

```
import java.sql.*;
import oracle.jdbc.*;
class JDBCConnection
{
public static Connection connect()
   throws SQLException
{
Connection conn = null;
try {
  OracleDriver ora = new OracleDriver();
  conn = ora.defaultConnection();
  }
  } catch (SQLException e) {...}
  return conn;
}
}
```

Пример работы с getConnection:

```
DriverManager.getConnection

("jdbc:oracle:kprb:");
```

или

```
DriverManager.getConnection

("jdbc:default:connection:");
```

Технология SQLJ – технология, упрощающая доступ к Oracle в Java путем использования выражений SQL непосредственно внутри Java-классов. Существуют две конструкции объявления SQLJ:

1. Итераторы. Это описания классов-итераторов, содержащие в себе результирующий набор данных:

```
#sql <modifiers> iterator
   iterator_classname (type declarations);

#sql public iterator EmpIter
                (String ename, double sal);
```

2. Контекст соединения – те классы, которые обычно применяются для доступа к БД, используя частичный набор SQL-сущностей:

```
#sql <modifiers> context
                                                    con-
text_classname;

#sql public context MyContext;
```

Синтаксис выражений SQLJ для операций без возвращаемых данных (таких, как INSERT и т. д.):

```
#sql { SQL operation };
```

Для SQL-выражений, возвращающих какой-либо результат, синтаксис использования следующий:

```
#sql result = { SQL operation };
```

Пример SQLJ выражения:

```
#sql { INSERT INTO emp (ename, sal)
            VALUES ('Joe', 43000) };
```

Пример простого метода, использующего SQLJ:

```
public static void writeSalesData
      (int[] itemNums, String[] itemNames)
throws SQLException
{
for (int i =0; i < itemNums.length; i++)
 #sql { INSERT INTO sales
        VALUES(:(itemNums[i]),
            :(itemNames[i]), SYSDATE) };
}
```

Пример реализации блока PL/SQL внутри Java-кода с помощью технологии SQLJ:

```
#sql {
DECLARE
  n NUMBER;
BEGIN
  n := 1;
  WHILE n <= 100 LOOP
  INSERT INTO emp (empno)
    VALUES (2000 + n);
  n := n + 1;
  END LOOP;
END
};
```

ЗАКЛЮЧЕНИЕ

Язык PL/SQL в настоящее время является мощным инструментальным средством, позволяющим разработку серверной логики практически любого уровня сложности. Использование PL/SQL позволяет с легкостью решать задачи контроля целостности данных средствами сервера Oracle, осуществлять интеграцию Oracle-приложений с сохранением их «прозрачности», а также осуществлять развертывание корпоративных многозвенных приложений. Помимо этого PL/SQL представляет собой наиболее простое и надежное средство расширения стандартной функциональности сервера БД Oracle.

Настоящее пособие представляет собой краткий справочник, позволяющий понять основы PL/SQL и изучить наиболее часто употребляемые конструкции этого языка. Среда PL/SQL представляет практически неограниченные возможности в области построения приложений любого размера: от небольших утилит до корпоративных систем, охватывающих все области деятельности предприятия.

ЛИТЕРАТУРА

1. Oracle 8. Энциклопедия пользователя : пер. с англ. / Компания Advanced Information Systems и др. – Киев : ДиаСофт, 1999.

2. Хансен Г. Базы данных: разработка и управление : пер. с англ. / Г. Хансен, Дж. Хансен. – М. : БИНОМ, 1999.

3. Саймон А. Р. Стратегические технологии баз данных: менеджмент на 2000 год : пер. с англ. / А.Р. Саймон, под ред. и с предисл. М. Р. Когаловского. – М. : Финансы и статистика, 1999.

4. Дейт К. Дж. Введение в системы баз данных : пер. с англ. / К. Дж. Дейт. – 6-е изд. – Киев : Диалектика, 1998.

5. Кузнецов С. Д. Основы современных баз данных / С.Д. Кузнецов – М. : 1998.

6. Астахова И. Ф. М. SQL в примерах и задачах : учеб. пособие / И.Ф. Астахова, А.П. Толстобров, В.М. Мельников. – 2-е изд., испр., доп. – Воронеж : Изд-во Воронеж. гос. ун-та, 2000.

Учебное издание

ОСНОВЫ ЯЗЫКА PL/SQL

Учебно-методическое пособие для вузов

Составители:
Гаршина Вероника Викторовна,
Сапегин Сергей Владимирович

Редактор Л.М. Носилова

Подписано в печать 26.06.08. Формат 60×84/16. Усл. печ. л. 3,4.
Тираж 50 экз. Заказ 1284.

Издательско-полиграфический центр
Воронежского государственного университета.
394000, г. Воронеж, пл. им. Ленина, 10. Тел. 208-298, 598-026 (факс)
http://www.ppc.vsu.ru; e-mail: pp_center@ppc.vsu.ru

Отпечатано в типографии Издательско-полиграфического центра
Воронежского государственного университета.
394000, г. Воронеж, ул. Пушкинская, 3. Тел. 204-133.

56