# Лабораторная работа № 13

 ОБЪЕКТЫ

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

В СУБД Oracle обеспечивается возможность создания, хранения объектных данных и работы с ними. Управление объектными данными аналогично управлению реляционными данными. Для манипулирования как реляционными, так и объектными данными в объектно-реляционной базе данных Oracle используются языки SQL и PL/SQL.

Объекты (objects) представляют собой совокупность данных, объединен-ных с методами обработки этих данных, и отражают свойства реальных сущ-ностей и операции, выполняемые над этими сущностями. Данные, входящие в объект, носят название атрибутов или свойств. В качестве методов обычно используются процедуры и функции. Управление атрибутами объекта осуще-ствляется только через методы.

Для работы с объектами вначале необходимо определить объектный тип (object type), который описывает атрибуты и методы конкретного ви-да объектов, после чего можно объявлять переменные, или экземпляры, данного типа. Каждый экземпляр имеет собственную область памяти и копию атрибутов объекта.

В объявляемых объектных типах можно использовать в качестве па-раметра ранее созданные объектные типы. Созданные объектные типы могут быть типом для переменных, использоваться в качестве атрибутов для других типов, являться строкой в таблице, являться столбцом в таб-лице, могут быть частью объектного представления.

Создание объектного типа. Объектный тип состоит:

1) из описания (заголовка, спецификации) объектного типа;

2) тела объектного типа.

В описании типа содержатся атрибуты и предварительные объявления методов, а тело состоит из определений методов. Каждая из этих частей компилируется отдельно, но пока не будет откомпилирован заголовок объектного типа, не будет компилироваться и его тело.

Для создания заголовка объектного типа используется следующий синтаксис:

CREATE [OR REPLACE] TYPE [схема.] имя\_типа AS OBJECT

(имя\_атрибута тип\_данных [, имя\_атрибута тип\_данных] …]

[{MAP | ORDER} MEMBER описание\_функции]

[MEMBER {описание\_процедуры | описание\_функции}

[, MEMBER {описание\_процедуры | описание\_функции}]...]

[PRAGMA RESTRICT\_REFERENCES (имя\_метода, ограничения)

[, PRAGMA RESTRICT\_REFERENCES (имя\_метода, ограничения)]...]

…);

где имя\_типа – имя нового объектного типа, а схема – его владелец. Сна-чала через запятую перечисляются атрибуты типа:

имя\_атрибута тип\_данных

где имя\_атрибута – это имя атрибута, а тип\_данных – встроенный тип PL/SQL, или ранее определенный пользователем тип данных, или ссылка на объектный тип. После атрибутов указываются методы объектного ти-па. При описании методов каждый из них, кроме последнего, отделяется запятой и выглядит как обычная хранимая подпрограмма PL/SQL с до-бавлением ключевого слова MEMBER.

Прагма RESTRICT\_REFERENCES используется для определения по-рядка вызова метода из SQL-оператора.

Для создания тела объектного типа используется следующий синтаксис:

CREATE [OR REPLACE] TYPE BODY имя\_типа AS

[{MAP | ORDER} MEMBER определение\_функции]

[MEMBER {определение\_процедуры | определение\_функции}

[MEMBER {определение\_процедуры | определение\_функции}]...]

END;

причем в теле объектного типа должны быть реализованы все объявлен-ные в заголовке процедуры и функции.

З а м е ч а н и я

1. При создании спецификации или тела пакета с помощью SQL\*PLUS, необходимо указывать в строке, следующей за последним оператором, косую черту (/), чтобы оператор CREATE … TYPE выпол-нился.

2. Описание атрибутов производится так же, как и описание полей типа записи PL/SQL или столбцов таблицы в операторе CREATE TABLE, но в отличие от полей записи атрибуты объектного типа нельзя ограничивать как NOT NULL и инициализировать значениями по умол-чанию.

3. Так как объектные типы хранятся в словаре данных, то типы дан-ных для атрибутов должны быть только те, которые есть в языке SQL, за исключением типов данных: LONG, LONG RAW, NCHAR, NVAR-CHAR2, NCLOB и ROWID.

4. Не допускается использование типов данных, доступных в языке PL/SQL: BINARY\_INTEGER, BOOLEAN, PLS\_INTEGER, RECORD, REF CURSOR.

5. Запрещается использование в атрибутах объектного типа атрибутов %TYPE и %ROWTYPE, но разрешается использование атрибута %TYPE с атрибутами экземпляра объектного типа.

6. Функции MAP и ORDER используются для задания порядка сор-тировки в данном объектном типе. Эти функции обсуждаются ниже.

7. Допускается создание объектных типов путем предварительного объявления типа, которое аналогично предварительному объявлению процедуры или метода:

CREATE TYPE имя\_типа;

8. Тело объектного типа не может содержать операторы тех методов, которые не указаны в спецификации типа.

Создадим объектный тип для сущности «книга». Атрибутами данного типа являются код книги, название книги, фамилия автора, цена книги. Объявим два метода: FORMATTED\_NAME и CHANGE\_PRICE, первый из которых возвращает фамилию автора и название книги, а второй об-новляет поле PRICE значением, указанным в поле NEW\_PRICE. В каче-стве имени объектного типа используем идентификатор BOOK\_T. Сле-дующая последовательность операторов создает спецификацию объект-ного типа BOOK\_T.

|  |
| --- |
| CREATE OR REPLACE TYPE BOOK\_T AS OBJECT  ( ID NUMBER(2), -- код книги  TITLE VARCHAR2(15), -- название книги  AUTHOR VARCHAR2(15), -- фамилия автора  PRICE NUMBER(7), -- цена книги  MEMBER FUNCTION FORMATTED\_NAME  RETURN VARCHAR2,  MEMBER PROCEDURE CHANGE\_PRICE  (NEW\_PRICE IN NUMBER)  ); |

|  |
| --- |
| Создадим тело объектного типа BOOK\_T:  CREATE OR REPLACE TYPE BODY BOOK\_T AS  MEMBER FUNCTION FORMATTED\_NAME  RETURN VARCHAR2 IS  BEGIN  RETURN SELF.AUTHOR||SELF.TITLE;  END FORMATTED\_NAME;  MEMBER PROCEDURE CHANGE\_PRICE  (NEW\_PRICE IN NUMBER) IS  BEGIN  PRICE := NEW\_PRICE;  END CHANGE\_PRICE;  END; |

|  |
| --- |
| Объявление и инициализация объектов. Как и другие переменные PL/SQL, объект описывается в разделе объявлений блока, после имени объекта указывается соответствующий тип. Например:  DECLARE  BOOK BOOK\_T;  BEGIN  NULL;  END;  В этом блоке BOOK описывается как экземпляр объектного типа BOOK\_T. По правилам PL/SQL экземпляр объекта, объявленный таким образом, инициализируется NULL-значением, т. е. создается NULL-объект, на атрибуты которого ссылаться нельзя. Подобная попытка при-ведет к ошибке. В следующем примере система выдаст ошибку «Ссылка на неинициализированный составной тип»:  DECLARE  BOOK BOOK\_T;  BEGIN  BOOK. TITLE:= 'Война и мир';  END;  Для инициализации объектов используется специальная функция-конструктор (constructor), возвращающая инициализированный объект. Система создает функцию-конструктор по умолчанию, имя функции совпадает с именем объектного типа, а список параметров – со списком атрибутов.  На атрибуты объекта, как и на поля записи, можно ссылаться, исполь-зуя уточняющее имя, записываемое перед именем атрибута через точку. Создадим инициализированный экземпляр типа BOOK\_T и изменим значение атрибута TITLE.  DECLARE  BOOK BOOK\_T := BOOK\_T (10, 'Казаки', 'Толстой Л.', 5000);  BEGIN  BOOK.TITLE := 'Война и мир';  END;  Чтобы проверить, является объект NULL-объектом или нет, используется условие IS NULL. Например:  DECLARE  BOOK BOOK\_T := BOOK\_T (10, 'Казаки', 'Толстой Л.', 5000);  BEGIN  IF BOOK IS NULL THEN  RAISE\_APPLICATION\_ERROR(-20000, 'BOOK IS NULL');  ELSE  BOOK.TITLE := 'Ярость';  END IF;  END; |

После создания каждый экземпляр объекта находится в определенном, свойственном только ему состоянии. Для модификации его состоя-ния используются методы объекта, которые должны ссылаться на кон-кретный экземпляр объекта. Следовательно, при вызове метода нужно использовать следующий синтаксис:

имя\_объекта.имя\_метода

где имя\_объекта – имя объектной переменной, а имя\_метода – имя вы-зываемого метода. Если метод не содержит аргументов, его можно вы-зывать без круглых скобок, как и обычную процедуру PL/SQL, либо со скобками и пустым списком аргументов. Это продемонстрировано на примере следующего блока PL/SQL. Для проверки правильности работы методов создадим специальную таблицу TEST\_T, содержащую два поля: фамилию автора и название книги, цену книги.

CREATE TABLE TEST\_T(FORM VARCHAR2(40), PRICE NUMBER (7));

DECLARE

BOOK1 BOOK\_T := BOOK\_T (10, 'Казаки', 'Толстой Л.', 5000);

BOOK2 BOOK\_T:= BOOK\_T (20, 'Война и мир', 'Толстой Л.', 15000);

FORM1 VARCHAR(40);

FORM2 VARCHAR(40);

BEGIN

-- Изменяет значение атрибута PRICE.

BOOK1. CHANGE\_PRICE(7000);

BOOK2. CHANGE\_PRICE(17000);

-- Выбираем фамилию автора и название книги.

FORM1 := BOOK1.FORMATTED\_NAME;

FORM2 := BOOK2.FORMATTED\_NAME();

-- Полученные данные помещаем в таблицу TEST\_T.

INSERT INTO TEST\_T VALUES(FORM1, BOOK1.PRICE);

INSERT INTO TEST\_T VALUES(FORM2, BOOK2.PRICE);

END;

Осуществить проверку правильности работы методов можно, выбрав из таблицы TEST\_T информацию оператором

SELECT \* FROM TEST\_T;

Все, что касается списка параметров, подчинено тем же правилам, что и для хранимых процедур и функций. Методы можно вызывать с исполь-зованием как именного, так и позиционного представления, а их пара-метры могут иметь значения по умолчанию. Методы могут быть пере-гружаемыми. Можно переопределять тип и число аргументов метода. Если в момент выполнения метода возникает исключение, то процедура завершает свою работу, причем выходным параметрам типа OUT и IN OUT ничего не присваивается; если же были выполнены какие-то изме-нения в SELF, то они также ликвидируются.

Ключевое слово SELF. Рассмотрим метод CHANGE\_PRICE:

MEMBER PROCEDURE CHANGE\_PRICE

(NEW\_PRICE IN NUMBER) IS

BEGIN

PRICE := NEW\_PRICE;

END CHANGE\_PRICE;

Этот метод вызывается для модификации атрибута PRICE типа BOOK\_T. Для автоматической привязки идентификатора PRICE к экзем пляру объекта внутри метода применяется ключевое слово SELF, т. е. в данном случае вместо оператора

PRICE := NEW\_PRICE;

можно было написать оператор

SELF.PRICE := NEW\_PRICE;

При передаче же текущего экземпляра объекта другой процедуре или функции в качестве аргумента указывать SELF необходимо. Приведем пример.

MEMBER PROCEDURE CHANGE\_PRICE\_ANOTHER

(BOOK BOOK\_T) IS

BEGIN

SELF.PRICE := BOOK.PRICE;

END CHANGE\_PRICE\_ANOTHER;

Размещение объектов в базе данных. В PL/SQL имеется возможность для создания объектов двух типов: устойчивых и неустойчивых, что определяет их свойства, а также операции, которые разрешено вы-полнять над ними. Устойчивым (persistent) объектом называется объект, хранящийся в базе данных, а неустойчивым (transient) – объект, объяв-ленный локально в блоке PL/SQL. С завершением работы блока PL/SQL неустойчивый объект разрушается. Устойчивый объект остается доступ-ным до тех пор, пока не будет удален явным образом. Приведем пример неустойчивого объекта BOOK:

DECLARE

FORM1 VARCHAR2(40);

-- Создание объекта.

BOOK BOOK\_T := BOOK\_T (10, 'Война и мир', 'Толстой Л.', 5000);

BEGIN

BOOK.CHANGE\_PRICE(7000);

FORM1 := BOOK.FORMATTED\_NAME;

-- Занесение значений атрибутов TITLE, AYTHOR, PRICE

-- в таблицу TEST\_T;

INSERT INTO TEST\_T VALUES(FORM1, BOOK.PRICE);

END;

По завершении работы блока объект BOOK будет разрушен. Сохра-нится лишь информация, занесенная в таблицу TEST\_T.

Устойчивые объекты хранятся в таблицах базы данных, как и стан-дартные скалярные типы. Существует два различных способа хранения объектов в таблице: в качестве объекта-строки или объекта-столбца.

Объект-строка (row object) занимает целую строку таблицы базы данных, причем в строке не должно содержаться других полей. Таблица, состоящая из таких строк, называется объектной (object table) и создается при помощи следующего оператора:

CREATE TABLE имя\_таблицы OF объектный\_тип;

где имя\_таблицы – это имя создаваемой таблицы, а объектный\_тип – тип объекта-строки. В качестве примера создадим объектную таблицу BOOK\_S на базе объектного типа BOOK\_T:

CREATE TABLE BOOK\_S OF BOOK\_T;

В каждой строке таблицы BOOK\_S находится экземпляр типа BOOK\_T, являющийся строкой, атрибуты которой определены в объект-ном типе BOOK\_T. Следовательно, в эту таблицу можно вводить только объекты. Ниже приведен ряд примеров ввода объектов в таблицу BOOK\_S. Как указывалось выше вставка новых значений в таблицу осуществляется через функцию-конструктор.

INSERT INTO BOOK\_S

VALUES (BOOK\_T(10, 'Казаки', 'Толстой Л.',5000));

INSERT INTO BOOK\_S

VALUES (BOOK\_T(20, 'Война и мир', 'Толстой Л.',15000));

Объектные таблицы очень похожи на обычные реляционные таблицы. Поэтому при заполнении объектной таблицы в целях сопряжения с обычными реляционными таблицами, которые были до этого в БД, до-пускается и обычная вставка при условии, что в списке атрибутов нет других встроенных объектных типов. К примеру, можно ввести данные в таблицу BOOK\_S с помощью следующего оператора INSERT:

INSERT INTO BOOK\_S

VALUES(30, 'Дюна', 'Герберт Ф.', 8000);

Во время такой модернизации различные реляционные таблицы мож-но создать заново как объектные, причем существующие приложения изменять вовсе не обязательно.

Операции DML, выполняемые над таблицами, в которых содержатся объекты-строки, абсолютно идентичны реляционным операциям DML.

В операторах UPDATE и DELETE можно указывать объекты в каче-стве переменных привязки. В этом случае операции DELETE и UPDATE можно выполнять, как над обычными реляционными таблицами. Пока-жем это на примерах:

UPDATE BOOK\_S SET TITLE = ’Дети Дюны’ WHERE ID = 30;

DELETE FROM BOOK\_S WHERE ID = 30;

В запросе объект-строка может вести себя по-разному. Если просто написать оператор вывода информации, перечислив все нужные атрибу-ты, то система будет рассматривать их как обычный набор атрибутов, а не объект, поскольку объектная таблица описывается точно так же, как и реляционная. Поэтому для выбора объекта нужно использовать опера-цию VALUE.

Операция VALUE (значение) возвращает объект. В качестве аргумен-та VALUE используется переменная корреляции, которая в данном контексте представляет собой псевдоним таблицы. Использование VALUE и обычного варианта запроса иллюстрируется в следующем примере:

DECLARE

CURRENT\_ID BOOK\_S.ID%TYPE;

CURRENT\_TITLE BOOK\_S.TITLE%TYPE;

CURRENT\_AUTHOR BOOK\_S.AUTHOR%TYPE;

CURRENT\_PRICE BOOK\_S.PRICE%TYPE;

CURRENT\_BOOK BOOK\_T;

BEGIN

SELECT \* INTO CURRENT\_ID, CURRENT\_TITLE,

CURRENT\_AUTHOR, CURRENT\_PRICE FROM BOOK\_S

WHERE ID = 10;

SELECT VALUE(B) INTO CURRENT\_BOOK FROM

BOOK\_S B WHERE ID = 10;

END;

При формировании объектной таблицы, где объекты – строки, каждо-му объекту присваивается идентификатор. Идентификатор объекта — это уникальный указатель на устойчивый объект конкретного типа, ко-торый однозначно определяет объект. Идентификаторы объектов уни-кальны по всему пространству БД Oracle: два объекта не могут иметь один и тот же идентификатор. Если даже объект удаляется, этот номер никакому другому объекту никогда не присваивается. Идентификатор объекта – это внутрисистемная структура; общее число идентификаторов составляет 2128 различных значений.

Идентификаторы объекта имеются только у объектов-строк и строк объектных представлений. Если объект имеет идентификатор, на этот объект можно ссылаться. Ссылка на объект описывается в разделе объ-явлений или указывается в описании таблицы и выглядит следующим образом:

имя\_переменной REF объектный\_тип,

где имя\_переменной – это имя ссылки на объект, а объектный\_тип – оп-ределенный ранее объектный тип.

Ссылки на объекты можно указывать в блоках PL/SQL и в SQL-операторах, используя при этом операции VALUE и REF.

Операции REF и DEREF. Результатом выполнения операции REF(аргумент) является ссылка на запрашиваемый объект. Как и для VALUE, аргумент REF – это переменная корреляции. Приведем пример, в котором выбирается ссылка на строку таблицы с конкретным значени-ем атрибута ID:

DECLARE

CURRENT\_BOOK\_REF REF BOOK\_T;

BEGIN

SELECT REF(B) INTO CURRENT\_BOOK\_REF

FROM BOOK\_S B WHERE ID = 10;

END;

Операция DEREF(ссылка) возвращает исходный объект по заданной на него ссылке. Продолжим предыдущий пример, в котором обновим значение атрибута TITLE у объекта, выбранного по ссылке:

DECLARE

CURRENT\_BOOK\_REF REF BOOK\_T;

CURRENT\_BOOK BOOK\_T;

BEGIN

SELECT REF(B) INTO CURRENT\_BOOK\_REF

FROM BOOK\_S B WHERE ID = 10;

SELECT DEREF(CURRENT\_BOOK\_REF)

INTO CURRENT\_BOOK FROM DUAL;

CURRENT\_BOOK.TITLE := 'Анна Каренина';

END;

Если объект, на который указывает REF, удален, то ссылку называют висячей (dangling), поскольку теперь она указывает на несуществующий объект. Использовать операцию DEREF для висячей ссылки нельзя. Что-бы проверить, не является ли ссылка висячей, применяется предикат IS DANGLING. Все рассмотренные операции (VALUE, REF, DEREF и IS DANGLING) можно использовать только в SQL-операторах. В проце-дурных операторах это запрещено.

В операторах INSERT и UPDATE используется конструкция – RE-TURNING. Она применяется для считывания информации из вновь вве-денной или обновленной строки; при этом формировать дополнительный запрос не требуется. Синтаксис конструкции RETURNING таков:

RETURNING список\_выбора INTO список\_ввода;

где список\_выбора аналогичен списку выбора запроса, а список\_ввода – это то же самое, что и оборот INTO запроса. Например, если в объект-ную таблицу вводится некоторый объект, то можно вернуть ссылку на вновь вводимый объект следующим образом:

DECLARE

CURRENT\_BOOK\_REF REF BOOK\_T;

BEGIN

193

INSERT INTO BOOK\_S B VALUES

(BOOK\_T(40, 'Игрок', 'Достоевский Ф.', 3000))

RETURNING REF(B) INTO CURRENT\_BOOK\_REF ;

END;

Объект-столбец (column object) занимает ровно один столбец табли-цы. Для создания таблицы, содержащей объект-столбец, нужно просто указать объектный тип в качестве типа столбца в операторе CREATE TABLE. Допускается в такой таблице наличие и скалярных типов. Для примера создадим таблицу BOOK\_DELIVERY на базе описанного выше типа BOOK\_T:

CREATE TABLE BOOK\_DELIVERY

(SALESMAN VARCHAR2(15),

QUANTITY NUMBER(7),

BOOK BOOK\_T);

Таблица BOOK\_DELIVERY состоит из трех столбцов, один из кото-рых является объектным типом. В данном случае ввести информацию в таблицу без конструктора нельзя, т. е. для ввода новых значений необхо-димо использовать следующую конструкцию:

INSERT INTO BOOK\_DELIVERY

VALUES ('Иванов И. И.', 5, BOOK\_T(10, 'Казаки', 'Толстой Л.', 5000));

В противном случае будет выдана ошибка.

Информацию из таблицы можно выбрать обычным способом с помо-щью оператора SELECT:

SELECT \* FROM BOOK\_DELIVERY;

На экран будет выведено:

Иванов И. И. 5 BOOK\_T(10, 'Казаки', 'Толстой Л.', 5000));

То есть столбец BOOK\_T рассматривается как единое целое, обра-титься непосредственно к какому-то отдельному элементу нельзя.

Кроме того, можно сослаться на объект-столбец в условии WHERE, указав перечень значений атрибутов этого объекта. Ниже приведен при-мер использования оператора SELECT.

SELECT SALESMAN FROM BOOK\_DELIVERY

WHERE BOOK = BOOK\_T(10, 'Казаки', 'Толстой Л.', 5000);

То же самое касается операторов DELETE, UPDATE.

Отметим, что в блоке PL/SQL для выполнения данного оператора для таблицы необходимо указать псевдоним.

DECLARE

CURRENT\_BOOK BOOK\_T;

BEGIN

SELECT BOOK INTO CURRENT\_BOOK

FROM BOOK\_DELIVERY D

WHERE D.BOOK.ID = 10;

END;

Иногда на практике такие таблицы, в которых имеются объекты-столбцы, учитывая, что существует таблица, где эти строки присутству-ют, строят следующим образом. В столбце указывается не объектный тип, а ссылка на этот тип.

CREATE TABLE BOOK\_DELIVERY\_M

(SALESMAN VARCHAR2(15),

QUANTITY NUMBER(7),

BOOK REF BOOK\_T);

Для вставки информации в такую таблицу используют следующую конструкцию:

INSERT INTO BOOK\_DELIVERY\_M

SELECT 'Иванов И. И.', 5, REF(B)

FROM BOOK\_S B

WHERE ID = 10;

Строка с ID = 10 имеет уникальный идентификатор, который извлека-ется с помощью операции REF(B) и размещается в столбце. В данном случае также необходимо использование синонима. Для просмотра ин-формации таблицы можно использовать следующий оператор:

SELECT \* FROM BOOK\_DELIVERY\_M;

Но в этом случае на экран выведется вместо третьего атрибута очень длинное число – идентификатор объекта. Поэтому для выдачи информа-ции лучше использовать следующий вариант:

SELECT SALESMAN, QUANTITY ,

DEREF(BOOK) FROM BOOK\_DELIVERY\_M;

В результате получим следующую строку:

Иванов И. И. 5 BOOK\_T(10, 'Казаки', 'Толстой Л.',5000));

В данном случае операция DEREF возвращает объект по ссылке. Если необходимо изменить информацию в таблице, то оператор UPDATE за-писывают в следующем виде:

UPDATE BOOK\_DELIVERY\_M SET SALESMAN = 'Петров П. П.'

WHERE DEREF(BOOK) = BOOK\_T(10, 'Казаки', 'Толстой Л.', 5000);

Методы MAP и ORDER. Методы MAP и ORDER – специальные ме-тоды, предназначенные для реализации упорядочения создаваемых на базе объектного типа реальных экземпляров объекта. При объявлении объектного типа можно использовать только один из этих методов. Ко-гда создается объект, который является структурой, то необходимо опре-делить порядок проверки условий, и именно MAP и ORDER устанавли-вают этот порядок. Если один из этих методов присутствует, то можно осуществить сравнение объектов не только в операторах SQL, но и в операторах PL/SQL. Если их нет, то объекты можно проверять только на эквивалентность и только в операторах SQL. Методы MAP и ORDER дают возможность упорядочивать объекты, а также сравнивать их в про-цедурных операторах. Вызов любого из этих методов происходит авто-матически.

Метод MAP всегда представляется функцией, которая не имеет па-раметров, и типом возвращаемого значения которой могут быть только DATE, NUMBER, VARCHAR2. На основании анализа данного объекта эта функция возвращает значение указанного типа, которое может быть использовано для упорядочивания объектов.

Ниже приведен пример использования метода MAP для объектного типа BOOK\_TM, аналогичного по структуре объектному типу BOOK\_T. Создадим заголовок объектного типа BOOK\_TM.

CREATE OR REPLACE TYPE BOOK\_TM AS OBJECT

( ID NUMBER(2),

TITLE VARCHAR2(15),

AUTHOR VARCHAR2(12),

PRICE NUMBER(7),

MAP MEMBER FUNCTION RETURN\_TITLE

RETURN VARCHAR2

);

Создадим тело объектного типа BOOK\_TM.

CREATE OR REPLACE TYPE BODY BOOK\_TM AS

MAP MEMBER FUNCTION RETURN\_TITLE

RETURN VARCHAR2 IS

BEGIN

RETURN TITLE;

END RETURN\_TITLE;

END;

Функция RETURN\_TITLE возвращает название книги. Создадим объ-ектную таблицу BOOK\_K на базе объектного типа BOOK\_TM.

CREATE TABLE BOOK\_K OF BOOK\_TM;

Внесем в таблицу BOOK\_K следующие записи с помощью оператора INSERT:

INSERT INTO BOOK\_K VALUES

(BOOK\_TM(10, 'Казаки', 'Толстой Л.', 5000));

INSERT INTO BOOK\_K VALUES

(BOOK\_TM(20, 'Война и мир', 'Толстой Л.', 15000));

INSERT INTO BOOK\_K VALUES (30, 'Дюна', 'Герберт Ф.', 8000);

После создания этой таблицы для упорядочения выводимых записей по названию книг можно выполнить, например, такой оператор SELECT:

SELECT VALUE(B) FROM BOOK\_K B ORDER BY TITLE;

В результате чего будет выведена следующая информация:

VALUE(B) (ID, TITLE, AUTHOR, PRICE)

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

BOOK\_TM(20, 'Война и мир', 'Толстой Л.', 15000)

BOOK\_TM(30, 'Дюна', 'Герберт Ф.', 8000)

BOOK\_TM(10, 'Казаки', 'Толстой Л.', 5000)

Метод ORDER всегда имеет два параметра: первый является встро-енным (так называемый SELF), а второй параметр (объектного типа) пе-редается в эту функцию. Метод ORDER возвращает следующие значе-ния, по которым можно судить о соотношении этих двух переданных функции объектов:

• -1, если параметр больше SELF;

• 1, если параметр меньше SELF;

• 0, если параметр равен SELF.

Метод ORDER используется аналогично методу MAP. Создадим метод ORDER для объекта BOOK\_TM; этот метод упорядочивает све-дения о книгах по названиям книг. Пересоздадим заголовок объектного типа BOOK\_TM.

CREATE OR REPLACE TYPE BOOK\_TM AS OBJECT

( ID NUMBER(2),

TITLE VARCHAR2(15),

AUTHOR VARCHAR2(12),

PRICE NUMBER(7),

ORDER MEMBER FUNCTION COMPARE\_BOOKS

(CUR\_BOOK IN BOOK\_TM)

RETURN NUMBER

);

Пересоздадим тело объектного типа BOOK\_TM.

CREATE OR REPLACE TYPE BODY BOOK\_TM AS

ORDER MEMBER FUNCTION COMPARE\_BOOKS

(CUR\_BOOK IN BOOK\_TM)

RETURN NUMBER IS

BEGIN

IF CUR\_BOOK.TITLE=SELF.TITLE THEN

RETURN 0;

ELSIF CUR\_BOOK.TITLE>SELF.TITLE THEN

RETURN -1;

ELSE

RETURN 1;

END IF;

END COMPARE\_BOOKS;

END;

После создания метода ORDER выполним следующий оператор:

SELECT VALUE(B) FROM BOOK\_K B ORDER BY TITLE;

В результате чего будет выведена следующая информация:

VALUE(B)(ID, TITLE, AUTHOR, PRICE)

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

BOOK\_TM(20, 'Война и мир', 'Толстой Л.', 15000)

BOOK\_TM(30, 'Дюна', 'Герберт Ф.', 8000)

BOOK\_TM(10, 'Казаки', 'Толстой Л.', 5000)

Изменение и удаление типов. Для изменения объектного типа ис-пользуется оператор ALTER TYPE, который можно использовать для перекомпиляции описания или тела объектного типа, либо для добавле-ния в тип новых методов.

Команда ALTER TYPE имеет следующий синтаксис:

ALTER TYPE имя\_типа COMPILE [SPECIFICATION | BODY];

где имя\_типа – имя изменяемого типа. С помощью этой команды можно перекомпилировать спецификацию или тело типа, которые в скомпилированном виде хранятся в словаре данных. Если не указывает-ся ни SPECIFICATION, ни BODY, то перекомпилироваться будут как 198

описание, так и тело типа. Например, следующая команда вызывает пе-рекомпиляцию тела типа BOOK\_T:

ALTER TYPE BOOK\_T COMPILE BODY;

Перекомпиляция объекта может потребоваться в случае, когда объект в результате изменения связанных с данным объектом других объектов.

Другая форма команды ALTER TYPE используется для добавления к типу новых методов. Она имеет следующий синтаксис:

ALTER TYPE имя\_типа REPLACE AS OBLECT

(спецификация\_объектного\_типа);

где имя\_muna – это имя объектного типа, а специфика-ция\_объектного\_типа – полное описание типа, определенное командой CREATE TYPE. Новое описание должно во всем, кроме дополнительных методов, совпадать с исходным. Должны быть указаны первоначальные атрибуты и типы. Если тело типа уже существует, оно становится недос-товерным, поскольку в нем не описаны новые методы.

Использование команды ALTER TYPE ... REPLACE AS OBLECT ил-люстрируется на следующем примере. Запросим статус объекта BOOK\_T.

SELECT OBJECT\_NAME, OBJECT\_TYPE, STATUS

FROM USER\_OBJECTS

WHERE OBJECT\_NAME = 'BOOK\_T';

В результате получим сообщение о достоверности заголовка и тела объекта BOOK\_T.

OBLECT\_NAME OBJECT\_TYPE STATUS

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

BOOK\_T TYPE VALID BOOK\_T TYPE BODY VALID

Изменим тип, добавив к нему новый метод. При этом тело типа стано-вится недостоверным.

ALTER TYPE BOOK\_T REPLACE AS OBJECT

( ID NUMBER(2), -- код книги

TITLE VARCHAR2(15), -- название книги

AUTHOR VARCHAR2(15), -- фамилия автора

PRICE NUMBER(7), -- цена книги

--Возвращает фамилию автора и название книги.

MEMBER FUNCTION FORMATTED\_NAME

RETURN VARCHAR2,

--Обновляет поле PRICE значением, указанным в поле NEW\_PRICE.

MEMBER PROCEDURE CHANGE\_PRICE

(NEW\_PRICE IN NUMBER),

--Возвращает код книги

MEMBER FUNCTION BOOK\_ID RETURN NUMBER

);

Запросим еще раз статус объекта BOOK\_T.

SELECT OBJECT\_NAME, OBJECT\_TYPE, STATUS

FROM USER\_OBJECTS

WHERE OBJECT\_NAME = 'BOOK\_T';

В результате получим сообщение о недостоверности тела объекта BOOK\_T.

OBLECT\_NAME OBJECT\_TYPE STATUS

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

BOOK\_T TYPE VALID BOOK\_T TYPE BODY INVALID

Пересоздадим тело типа BOOK\_T с добавленным новым методом.

CREATE OR REPLACE TYPE BODY BOOK\_T AS

MEMBER FUNCTION FORMATTED\_NAME

RETURN VARCHAR2 IS

BEGIN

RETURN SELF.AUTHOR||SELF.TITLE;

END FORMATTED\_NAME;

MEMBER PROCEDURE CHANGE\_PRICE

(NEW\_PRICE IN NUMBER) IS

BEGIN

PRICE := NEW\_PRICE;

END CHANGE\_PRICE;

MEMBER FUNCTION BOOK\_ID RETURN NUMBER IS BEGIN RETURN ID;

END BOOK\_ID;

END;

Запросим статус объекта BOOK\_T.

SELECT OBJECT\_NAME, OBJECT\_TYPE, STATUS

FROM USER\_OBJECTS

WHERE OBJECT\_NAME = 'BOOK\_T';

В результате получим сообщение о достоверности тела объекта BOOK\_T.

OBLECT\_NAME OBJECT\_TYPE STATUS

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

BOOK\_T TYPE VALID BOOK\_T TYPE BODY VALID

Команда DROP TYPE используется для удаления объектного типа и имеет следующий синтаксис:

DROP TYPE имя\_типа [FORCE];

При этом если используется опция FORCE, то объектный тип удаля-ется принудительно, несмотря на имеющиеся связи, и делая при этом не-действительными все зависимое от него объекты. Если данный параметр отсутствует, то объектный тип будет удален только в том случае, если на него нет ссылок, т. е. при отсутствии в схеме других объектов, завися-щих от данного объектного типа.

Команда

DROP TYPE BODY имя\_типа

удаляет только тело объектного типа, не трогая спецификацию типа и все зависимые от него объекты.