**Объектные типы данных.**

Объекты могут быть постоянными и временными. Хранимый (persistent) объект может быть как значением столбца обычной таблицы, так и строкой таблицы (в этом случае таблицу называют объектной).

Временный (transient) объект создается в памяти и уничтожается после окончания работы программы. Отправив такой объект в базу данных, вы делаете его постоянным. А постоянный объект можно извлечь из БД и поместить его во временный объект такого же типа.

Тип объекта (то есть абстрактный тип данных и методы) всегда хранится в словаре. Экземпляр хранимого объекта хранится в таблицах (одной или более) как значение столбца или строка объектной таблицы.

Выборка и манипулирование хранимыми объектами в базе данных осуществляется на языке SQL. Для работы с временными объектами на сервере предназначен язык PL/SQL. Язык C++ позволяет работать с объектами на стороне клиента. Используются вызовы программ на C++ из PL/SQL. В Java можно работать на всех трёх слоях архитектуры MVC.

Можно выделить следующие разновидности объектных типов:

* простой объектный тип, который строится на скалярных предопределённых типах данных;
* составной объектный тип, использующий другие объектные типы;
* ссылочный объектный тип;
* типы коллекций двух разновидностей VARRAY и NESTED TABLES.

Ссылочный объектный тип REF это логический указатель, определяющий отношения между экземплярами классов. Он основывается на одноэлементных или коллекционных типах данных.

Указатели REF задают ассоциации UML и заменяют внешние ключи, предоставляя прямую навигацию между объектами разных типов.

VARRAY – это упорядоченная коллекция фиксированной длины. Хранится в сегменте таблицы, использующей такой тип.

Вложенная таблица NESTED TABLE это неограниченная и неупорядоченная коллекция. Хранится в своём сегменте, не совпадающем с сегментом основной таблицы.

**Создание пользовательского типа данных**

Создадим тип данных dept\_type, описывающий таблицу dept.

CREATE TYPE dept\_type AS OBJECT (

Deptno NUMBER(2),

Dname VARCHAR2(14),

Loc VARCHAR2(23)

);

Затем создадим таблицу emp\_dept (то есть emp, включающую в себя dept).

CREATE TABLE emp\_dept (

empno NUMBER(4),

ename VARCHAR2(10),

job VARCHAR2(9),

mgr NUMBER(4),

hiredate DATE,

sal NUMBER(7,2),

comm NUMBER(7,2),

dept1 DEPT\_TYPE

);.

**Спецификация типа**

В предыдущем примере была создана только спецификация типа. Её синтаксис:

CREATE [OR REPLACE] TYPE [schema .]type\_name

{ { IS | AS } OBJECT }

[ { attribute datatype [sqlj\_object\_type\_attr] } ] |

{ [ {[[[NOT] OVERRIDING]

[[NOT] FINAL] [[NOT] INSTANTIABLE]]

{ { MEMBER | STATIC }

{ procedure\_spec | function\_spec } |

{{ MAP | ORDER } MEMBER function\_spec}}}]

.,:}

[[NOT] FINAL] [[NOT] INSTANTIABLE];

Мы не будем изучать её в полном объёме. Отметим только, что если спецификации методов (MEMBER FUNCTION) в описании типа появляются, то в теле типа эти методы должны быть раскрыты.

Методы могут быть функциями и процедурами.

Отметим, что методы MAP и ORDER задают отношения порядка.

**Тело типа**

CREATE TYPE BODY type\_name {IS | AS}

{ {MAP | ORDER}

MEMBER function\_body; | MEMBER {procedure\_body | function\_body};}

[MEMBER {procedure\_body | function\_body};]... END;

Пример: спецификация и тело типа

*Спецификация типа person:*

CREATE TYPE person AS OBJECT (

name VARCHAR2(40),

birthday DATE, -- дата рождения

address address\_type,

MEMBER FUNCTION Age – вернёт возраст

RETURN NUMBER – возвр. значение

);

*Тело типа person:*

CREATE OR REPLACE TYPE BODY person IS – задание методов

MEMBER FUNCTION Age RETURN NUMBER IS

BEGIN -- вычисление возраста

RETURN ROUND(MONTHS\_BETWEEN(sysdate, birthday)/12);

END;

END;.

**Изменение и удаление типов. Зависимости**

Объектные типы можно изменять оператором ALTER TYPE и удалять оператором DROP TYPE.

Форматы некоторых команд:

ALTER TYPE имя\_типа COMPILE [SPECIFICATION|BODY]

Компилирует спецификацию или тело типа, хранящиеся в словаре. Если не указано ни одно из слов SPECIFICATION, BODY, то компилируются оба.

ALTER TYPE имя\_типа REPLACE AS OBJECT(спецификация\_объектного\_типа)

Новое описание, заданное спецификация\_объектного\_типа должно со всем, кроме дополнительных методов совпадать с исходным.

После выполнения этого оператора, если тело типа существовало ранее, оно становится INVALID.

DROP TYPE [имя\_схемы.]имя\_типа [][FORCE]

Если указан параметр FORCE, то тип удалится, даже если существуют зависимые от него объекты. Естественно, все зависимые объекты становятся INVALID.

**Пример зависимости объектов**



**Конструкторы по умолчанию**

Метод конструктора возвращает новый экземпляр объектного типа. Для демонстрации применения конструктора по умолчанию создадим тип address\_type, затем на его основе объектный тип person и таблицу peoples со столбцом этого типа

CREATE TYPE address\_type as object (

zipcode number(5),

country varchar2(20),

city varchar2(30),

street varchar2(30),

numb number(4));

CREATE TYPE person AS OBJECT (

name VARCHAR2(40),

birthday DATE, -- дата рождения

address address\_type,

MEMBER FUNCTION Age -- спецификация метода, возвращающего возраст

return NUMBER -- возвращаемое значение

);

Ранее определена только спецификация типа, создаём тело типа:

CREATE OR REPLACE TYPE BODY person IS -- задание методов

MEMBER FUNCTION Age RETURN NUMBER IS -- вычисление возраста

BEGIN

RETURN ROUND(MONTHS\_BETWEEN(sysdate, birthday)/12);

END;

END;

Создадим таблицу типа person командой

CREATE TABLE peoples of person;

Посмотрим описание таблицы с созданным типом (команда desc person).

**Особенности объектных таблиц**

Из ограничений целостности в объектных таблицах работают только primary key и unique key. Например, создадим таблицу peoples1 c первичным ключом:

create table peoples1 of person (name primary key);

Проверим ее структуру для сравнения с peoples. В столбце Primary Key в строке NAME появится значение 1.

В объектных таблицах вставка строки “по-старому” не удастся. Например, вставка

INSERT INTO peoples

VALUES ('Сидоров И. П.', '22.11.80', 350000, 'Россия', 'Краснодар','Ставропольская', 149);

вызывает появление ошибки.

Теперь вставка должна производиться с указанием иерархии типов пользователя, например так:

INSERT INTO peoples

VALUES ('Сидоров И.П.', '22.11.80', address\_type (35000, 'Россия', 'Краснодар', 'Ставропольская', 149));

Читать как в обычных реляционных таблицах также не удастся. Так, команда

SELECT \* FROM peoples;

вызывает сообщение об ошибке.

Тот же результат дают запросы:

SELECT name, birthday, address FROM peoples;

SELECT name, birthday, address.country FROM peoples;

А вот следующая команда позволяет работает нормально:

SELECT name, birthday, p.address.country FROM peoples p;

**Индексы и ограничения целостности**

Индекс может быть создан на любой листовой столбец объектной таблицы, в том числе принадлежащий вложенной таблице или входящий в атрибут объектного типа.

Забегая вперёд, отметим, что индексировать можно и объектные ссылки, но только в том случае, когда при их определении не только указан тип, на который ссылаются, но и определена таблица этого типа.

Пример:

CREATE INDEX country\_idx ON peoples (address.country);

В этих же ситуациях можно определять и вводить ограничения целостности, например,

ALTER TABLE peoples ADD CONSTRAINT peoples\_cons1 CHECK (address.country IS NOT NULL);

При создании таблицы также следует использовать ограничения целостности уровня таблицы.