ОБЪЕКТЫ СО СЛОЖНОЙ СТРУКТУРОЙ В СОВРЕМЕННЫХ БАЗАХ ДАННЫХ

Пржиялковский В. В.

(Россия, Москва)

Во второй половине 90-х годов в основных универсальных СУБД появились возможности работать не со скалярами, как того предписывает реляционная модель данных и стандарт SQL, а с объектами, имеющими структуру. Реализация в разных типах СУБД выполнена по-разному, но везде следует идеям объектно-ориентированного подхода к БД. Точнее, для работы со структурированными объектами в БД стандарт SQL и ряд современных типов СУБД предлагают три взаимосвязанных механизма: больших неструктурированных объектов, явно структурированных объектов и механизм группировки объектов. В докладе показано, как эти механизмы могут использоваться и сочетаться для моделирования данных со сложной структурой, например, для моделирования метрики и упорядоченности на множестве объектов.

Введение

Реляционная модель данных, предложенная более 30 лет назад, вскоре после своего появления дала начало двум важным для системы управления базами данных (СУБД) явлениям: языку запросов SQL и диалектам SQL в реализациях основных современных универсальных СУБД. От реляционной модели SQL, а от SQL — диалекты этого языка унаследовали правило скалярности атрибутов таблиц. Согласно этому правилу значения в полях таблицах SQL могут быть только скалярами, то есть атомарными единицами, о структуре которых SQL ничего не известно, и которые можно занести в БД, обновить или удалить только целиком (а не частями).

Некоторые отклонения от этого правила составили типы «дата» и «строка», скалярность которых в SQL спорна, однако с некоторой натяжкой, «по духу», их можно полагать скалярными. Основная же критика скалярности атрибутов выражается утверждением, что «реляционная модель» (а следом и SQL) «не позволяет хранить в БД сложно устроенные данные». Примерами сложно устроенных данных служат фотографии, изображения, временные ряды и другие виды данных, имеющих собственную структуру. Возражения защитников реляционной модели (К. Дейт, Х. Дарвен, Ф. Паскаль) сводятся к тому, что реляционная модель *не* запрещает хранить в БД сложно устроенные данные, но делегирует работу с их внутренней структурой приложению, таким образом отвечая только за собственно хранение и извлечение.

Тем не менее в 90-х годах были предприняты попытки включить возможности работы со сложно устроенными структурами в SQL. В результате они появились в стандарте SQL:1999, а во второй половине 90-х и в диалектах SQL основных поставщиков универсальных СУБД. Ниже эти возможности будут рассмотрены на примере СУБД Oracle, диалект SQL которой наиболее функционален на фоне прочих СУБД.

Для хранения сложно устроенных данных Oracle предлагает несколько разных средств. Они рассматриваются ниже.

Большие неструктурированные объекты

Большие неструктурированные объекты (large objects, LOBs) ближе всего напоминают скаляры реляционной модели, так как являются расширением понятий «строка символов» и «строка байтов», позволяющим работать со строками особо больших размеров. В стандарте SQL:1999 предусмотрено два типа для таких данных: CLOB (character large object) и BLOB (binary large object); Oracle их предлагает четыре:

- CLOB: для хранения строк символов в однобайтовой кодировке
- BLOB: для хранения последовательности байтов
- NCLOB: для хранения строк символов в многобайтовой кодировке, например, Unicode

— BFILE: для ссылки на последовательность байтов, хранимую вне БД (в файловой системе; фактически – ссылка на файл, например документ)

Все типы категории LOB способны хранить в поле таблицы до 4 Гб данных. Типы CLOB и NCLOB позволяют хранить тексты, а BLOB и BFILE — последовательность байтов, то есть файл: в первом случае в БД, а во втором вне (и слово «хранить» здесь требует кавычек). Сама СУБД ничего не знает и не предполагает о прикладном предназначении файла, не накладывает на содержимое никаких ограничений, и, соответственно, не несет за него никакой ответственности. «Неструктурированны» эти типы именно с точки зрения SQL, а с точки зрения приложения у объектов этих типов может иметься структура, какая требуется. Хранить в таком поле можно все, что угодно: документ в формате Word или PDF, изображение, массив данных и так далее. Пример:

```
CREATE TABLE experiment (
  identification VARCHAR2 ( 100 ) PRIMARY KEY
  , datetime TIMESTAMP
  , description BFILE /*файл в формате Word*/
  , sample BLOB     /*массив показаний*/
);
```

На уровне SQL (то есть общения с БД) возможности такого поля ограничены практически только извлечением и присваиванием, например:

```
UPDATE experiment1 e1
SET e1.sample = (SELECT e.sample
                 FROM experiment e
                WHERE e.identification ='initial sample')
;
```

Формирование же и интерпретация таких полей почти исключительно выполняется приложением. В Oracle это можно сделать на языках PL/SQL, Java или C. Пример программирования на PL/SQL:

```
DECLARE sample_locator BLOB;
BEGIN
   SELECT sample INTO sample_locator
   FROM experiment
   WHERE identification = 'initial sample';
   -- далее обработка, например:
   put_picture ( sample );
   -- ...
END;
/
```

Стандарт SQL, а вслед и диалект SQL в Oracle, оба различают собственно неструктурированный объект (массив LOB) и указатель на него, *локатор*. Так, в программном коде выше переменная SAMPLE_LOCATOR фактически является локатором, который связывается с объектом (массивом) BLOB, хранимым в БД в поле SAMPLE.

В Java работа с объектами LOB реализована разработчиками Oracle через понятие потока (что разумно). Но поскольку в SQL есть локатор, а не поток, «подобраться» в программе к потоку можно только через локатор:

```
oracle.sql.BLOB blob_loc;

ResultSet rset = stmt.executeQuery(
      "SELECT sample
       FROM experiment
       WHERE identification = 'initial sample'");

rset.next();
oracle.sql.BLOB blob_loc = ((OracleResultSet)rset).getBLOB(1);

// получили локатор, можем читать поток:
InputStream byte_stream = blob_loc.getBinaryStream();

// далее читаем поток и обрабатываем данные ...
```

(С точки зрения Java переменная blob_loc – это «ссылка на локатор массива LOB», расположенного в базе данных !)

В приведенном фрагменте кода речь идет о потоке байтов, но приложение может интерпретировать его и как поток чисел, записей и т. д.

Объекты со структурой

Объекты категории LOB не являются объектами с точки зрения объектно-ориентированного подхода, и недаром носят определение «неструктурированных». Понятие структурированного объекта (довольно схематичное) появилось в стандарте SQL:1999, а в Oracle – в конце 90-х годов, но в собственной реализации. Сама реализация объектных возможностей в Oracle выполнена достаточно качественно и полнофункционально, так что претензии, которые к ней можно было бы предъявлять, вызваны скорее общими проблемами применения объектного подхода в базах данных.

Для работы со структурированными объектами Oracle требует сначала создать тип:

```
CREATE TYPE point_type AS OBJECT (x NUMBER, y NUMBER)
/
```

В соответствие с объектной терминологией X и Y будут называться *свойствами* объектов этого типа.

Созданный тип, во-первых, позволяет создать *таблицы объектов*, например:

```
[A] CREATE TABLE point OF point_type;
```

Хотя Oracle, вслед за стандартом SQL, использует термин «таблица», эта таблица специфическая: у нее всегда имется ровно один, нескалярный атрибут. Тем самым фактически в [A] описывается *множество* объектов (можно вспомнить, что в таблицах SQL, а вслед и в Oracle, строки образуют [неупорядоченные] множества).

Во-вторых, тип объекта можно использовать для организации нескалярного атрибута в обычной таблице:

```
[Б] CREATE TABLE position (datetime TIMESTAMP, point point_type);
```

С точки зрения моделирования разница между использованием типа объекта в [А] и [Б] в том, что первом случае в базе заводится самостоятельный объект (экземпляр типа), полноценный с позиции объектно-ориентированного подхода. В случае же [Б] речь идет об «объектном атрибуте» традиционной таблицы. В случае [А] объекты, создаваемые в таблице POINT, имеют собственный «идентификатор объекта» (термин объектного моделирования), вследствие чего на них можно ссылаться из других таблиц; в случае же [Б] ничего подобного нет, а адресоваться к свойствам объекта можно только найдя соответствующую строку обычными средствами SQL. Пример использования ссылки на объект:

```
CREATE TABLE remark (
        text VARCHAR2 ( 200 )
      , pos REF point_type SCOPE IS point
);
```

В этом примере в поле POS можно хранить ссылку на объект, существующий только в таблице объектов POINT, но можно и не ограничивать описание привязкой к конкретной таблице. Для этого конструкцию SCOPE IS point в предложении выше достаточно опустить.

Примеры запросов, применимых в Oracle к трем вышеприведенным таблицам:

```
SELECT * FROM point;

SELECT p.x, p.y FROM point p;

UPDATE point p SET p.y = 6 WHERE p.x = 3;

SELECT * FROM position;

SELECT p.datetime, p.point.x, p.point.y FROM position p;

SELECT r.text, DEREF(r.pos), DEREF(r.pos).x FROM remark r;
```

(В последнем случае допускается сокращенная запись, например R.POS.X).

Свойствами объектов могут быть не обязательно «простые» скаляры, но и неструктурированные объекты LOB и даже структурированные объекты (представленные как «объектные атрибуты»):

```
CREATE TYPE sample_type AS OBJECT (
     identification VARCHAR2 (100) /* идентификация */
     , point point_type   /* место снятия показаний */
     , data BLOB )              /* массив показаний */
/
```

Методы

Пример указания метода в типе объекта Oracle:

```
CREATE TYPE line_type AS OBJECT (
        p1 point_type
      , p2 point_type
      , MEMBER FUNCTION length RETURN NUMBER
)
/
```

Процедурное описание метода оформляется отдельно, например:

```
CREATE TYPE BODY line_type IS
  MEMBER FUNCTION length RETURN NUMBER IS
  BEGIN
    RETURN
    SQRT (POWER (p1.x – p2.x, 2) + POWER (p1.y – p2.y, 2));
  END;
END;
/
```

Обращение к методу синтаксически требует указания круглых скобок, даже если параметры у метода отсутствуют:

```
SELECT VALUE(l), l.length() FROM line l;
```

Существенно, что языком программирования методов помимо собственного языка Oracle PL/SQL могут быть еще Java и C.

Две разновидности методов в Oracle представляют особый интерес. Во-первых, конструкторы (хотя они формально и не называются методами). Конструкторы можно использовать в выражениях SQL для обозначения «буквальных значений» объектов, чаще именуемых «литералами». Использовать же выражения можно в разных контекстах, например при добавлении новых значений в базу данных или при изменениях:

```
CREATE TABLE line OF line_type;

INSERT INTO line VALUES(point_type (1, 1), point_type (5, 5));

UPDATE point p SET VALUE(p) = point_type(4, 7) WHERE p.x = 3;
```

Другой пример — использование выражений для построения запросов к БД:

```
SELECT * FROM line l WHERE VALUE(l).p1 = point_type (1, 1);
```

Во-вторых, для каждого типа объекта в Oracle можно альтернативно задать либо специальный метод, задающий правило упорядочения, либо метод, задающий отображение объектов на числовую ось. Пример последнего мог бы выглядеть так:

```
CREATE TYPE line_type AS OBJECT (
        p1 point_type
      , p2 point_type
      , MAP MEMBER FUNCTION length RETURN NUMBER
)
NOT FINAL
/

CREATE TYPE BODY line_type IS
  MAP MEMBER FUNCTION length RETURN NUMBER IS
  BEGIN
    RETURN
    SQRT (POWER (p1.x - p2.x, 2) + POWER (p1.y - p2.y, 2));
  END;
END;
/
```

Существование у типа объектов метода упорядочения или метода отображения на числовую ось позволяет использовать в выражениях SQL не только сравнение объектов на равенство, но и на неравенство:

```
SELECT * FROM line l
WHERE VALUE(l) > line_type(point_type(0, 0),
point_type(5, 5));
```

Наследование типов

Oracle полностью поддерживает одиночное наследование типов, с возможностями наследования свойств и методов, переопределения методов, создания абстрактных (бесплодных) подтипов. Например:

```
CREATE TYPE annotated_line_type UNDER line_type (
        text VARCHAR2 (200)
)
/
```

При наличии подтипа ANNOTATED_LINE_TYPE поле таблицы, имеющее тип LINE_TYPE, по умолчанию сможет хранить объекты как супертипа, так и подтипа, то есть и просто линии, и аннотированные линии. Однако можно обязать его хранить объекты только супертипа (LINE_TYPE; «просто линии» в нашем случае), невзирая на наличие подтипов («аннотированных линий»), например:

```
CREATE TABLE figure (id NUMBER, aline line_type)
   COLUMN aline NOT SUBSTITUTABLE AT ALL LEVELS;
```

В то же время, если такого запрета нет, и в поле разрешено хранить объекты и супертипа и его подтипов, отбор данных из базы можно сузить только объектами какого-то определенного типа в дереве подтипов, например:

```
SELECT *
FROM figure1
WHERE aline IS OF (ONLY annotated_line_type);
```

Будут выбраны только аннотированные линии.

Oracle допускает переопределение существующего типа при наличии подтипов и супертипов и даже при наличии объектов этих типов в базе, но не всегда, или это не всегда просто. Эти ограничения почти целиком вызваны общими ограничениями объектно-ориентированного подхода к базам данных.

Массивы данных

Возможность формировать группы из объектов в той же степени противоречит реляционному подходу, как и работа со структурированными объектами. Тем не менее, она введена в SQL, отдаляя его еще больше от реляционного подхода к построению модели данных. Справедливости ради, идея расширить реляционную модель понятием «коллекция объектов» в свое время рассматривалась ее автором Э. Коддом, но не получила завершения. К тому же часть разработчиков стандарта SQL имеют свою точку зрения на массивы: они полагают, что массивы разрешают хранить информацию, которую можно *разложить* на элементы, то есть они являются аналогом строки (текста, байтов), которая традиционно полагается скаляром. С последним, однако, можно было бы согласиться, если бы речь шла только о массивах *из* скаляров; в Oracle же (см. примеры ниже) массивы могут состоять не только из структурированных объектов, но даже из ссылок на объекты в БД, то есть быть именно коллекциями в понимании Э. Кодда (Кодд предлагал в качестве поясняющего коллекцию примера понятие «конвой» из самостоятельно существующих «судов»).

Стандарт SQL:1999 вводит понятие массива данных, однако Oracle не следует ему буквально и предлагает в своем диалекте языка две разновидности коллекций: вложенные таблицы и массивы VARRAY.

Так, вместо определения SQL:1999

```
|sample NUMBER NUMERIC ARRAY[100]
```

в Oracle можно дать следующие (разные) определения:

```
CREATE TYPE sample_nt_type IS TABLE OF NUMBER (20, 2);
```

```
CREATE TYPE sample_var_type IS VARRAY (100) OF NUMBER
(20, 2);
```

Первое определяет тип вложенной таблицы из чисел указанного формата, а второе — тип массива VARRAY. Обратите внимание, что число элементов в реальных вложенных таблицах не подвержено ограничениям, а в массивах VARRAY ограничено максимумом, наподобие максимального числа символов в строке переменной длины.

Приведенные определения двух типов коллекций можно употребить для описания полей обычных таблиц по общим правилам:

```
sample sample_nt_type

sample sample_var_type
```

(Таким же образом можно задавать и типы переменных в программе на PL/SQL).

Основная потребительская разница между типами вложенной таблицы и массива VARRAY в том, что первый применяется для моделирования в базе данных неупорядоченного множества, а второй — для упорядоченного.

Oracle дает разнообразные возможности работать с массивами на уровне элементов программно, средствами PL/SQL, но целый ряд действий допускается и в SQL. Например, в SQL можно получить из массива список значений:

```
SELECT * FROM TABLE(SELECT varray_column FROM some_table);
```

Как следует из примера, для этого нужно извлечь из базы данных значение массива (из поля, представляющего собой массив) и преобразовать его в список строк специальной функцией TABLE. Полученный результат можно использовать в качестве источника данных для «охватывающего» оператора SELECT наряду с обычными таблицами. С помощью функции CAST, известной по стандартному SQL, Oracle выполняет преобразования данных их одно-

го типа коллекции в другой (например, из SAMPLE_NT в SAMPLE_VAR) или из списка значений в коллекцию.

Работа выполнена отчасти за счет средств гранта РФФИ 04-01-00401-а.

OBJECTS WITH COMPLEX STRUCTURE IN MODERN DATABASES

Przhiyalkovsky V. V.

(Russia, Moscow)