ПОДДЕРЖКА ПОЛЬЗОВАТЕЛЬСКИХ ТИПОВ ДАННЫХ В ПАРАЛЛЕЛЬНОЙ СУБД ОМЕГА ДЛЯ МВС-100 *

М.Л. Цымблер

В статье описывается подход к поддержке пользовательских типов данных в реляционной системе управления базами данных (СУБД). Тип данных понимается как совокупность внешнего представления экземпляров данного типа и набора операций над экземплярами данного типа. Фиксируется предопределенное множество стандартных типов данных. Внешним представлением пользовательского типа данных является последовательность стандартных и ранее определенных пользовательских типов данных. Описание пользовательской операции задается с помощью специальной нотации. СУБД интерпретирует описание и сохраняет результат интерпретации в базе данных. Рассматриваемый подход применен в разработке менеджера типов данных параллельной СУБД Омега для отечественного многопроцессорного вычислительного комплекса МВС-100.

Ключевые слова: *реляционная система управления базами данных, пользовательские типы данных.*

1. Введение

В настоящее время реляционные системы управления базами данных (СУБД) играют доминирующую роль на рынке программного обеспечения систем баз данных [1]. В рамках реляционной модели данных [2] база данных рассматривается как набор взаимосвязанных отношений (таблиц). Каждое значение в ячейке таблицы должно иметь скалярный тип.

Современные реляционные СУБД предоставляют пользователю довольно большой набор встроенных типов данных (например, целые

* Работа выполнена при финансовой поддержке Российского фонда фундаментальных исследований (гранты № 00-07-90077, 02-07-06027).

и вещественные числа, символьные строки, даты) и соответствующих операций над экземплярами этих типов (например, арифметические операции над числами, конкатенация строк, сравнение и другие операции с датами). Однако предоставляемый набор типов данных может быть неадекватным для ряда нетрадиционных приложений баз данных [1]. Необходимый пользователю тип данных может либо отсутствовать в наборе встроенных типов, либо иметь семантику, неприемлемую для приложения. Например, в приложениях географических баз данных, как правило, требуется тип данных “Прямоугольник” и операции “Пересечение”, “Расстояние”, “Вложение”. Другим примером может быть приложение, в котором тип "Дата"отличается от стандартного тем, что каждый месяц года состоит из 30 дней.

Указанная проблема определяет задачу разработки средств поддержки пользовательских типов данных и функций эффективной реализации операций этих типов как одну из важных задач для СУБД нового поколения [3].

Согласно [4], СУБД, поддерживающая расширение набора встроенных типов данных, должна предоставлять пользователю средства определения новых типов данных и операций над ними и обеспечивать оптимизацию запросов к данным новых типов.

В настоящей работе мы рассмотрим подход к определению новых типов данных и операций над этими типами, примененный при разработке менеджера типов данных для параллельной СУБД Омега для отечественной мультипроцессорной вычислительной системы МВС-100 [5].

2. Менеджер типов данных в СУБД Омега

В иерархии подсистем СУБД Омега менеджер типов данных занимает следующий, более высокий уровень, чем система управления файлами (СУФ) [6]. СУФ поддерживает понятие файла как набора записей фиксированной длины. Запись состоит из системного заголовка, имеющего структуру, и определяемого пользователем информационного поля, которое представляет собой массив байтов, не имеющий структуры. Основная функция менеджера типов — поддержка понятия типа данных для определения структуры информационного поля записи файла.

В этом разделе мы сначала перечислим концепции и принципы, которые были использованы при проектировании менеджера типов данных СУБД Омега, а затем опишем интерфейс и схему реализации данной подсистемы.

2.1. Базовые концепции

Классический подход к поддержке типов данных основывается на теории структурной организации данных Хоара [7], где тип определяет класс значений, которые могут принимать переменная или выражение, а набор рассматриваемых типов ограничивается типами, используемыми в математике (множества, последовательности и др.).

При проектировании менеджера типов СУБД Омега мы используем концепцию *абстрактного типа данных* [8]. Абстрактный тип данных (АТД) определяет не только класс допустимых значений экземпляров данного типа, но также набор допустимых операций над экземплярами данного типа. В определение АТД, в частности, входят внешнее представление (имя типа, имена допустимых операций, их аргументов и т.п.) и абстрактное описание операций на некотором языке спецификаций. Концепция АТД нашла конструктивное воплощение в языках программирования CLU [9] и Alphard [10].

В отличие от языков программирования, система поддержки типов данных в СУБД должна обеспечивать *персистентность* создаваемых типов, то есть возможность долговременного хранения информации о них в базе данных. Другими словами, если во время работы приложения базы данных был создан некоторый тип данных, то он продолжает существовать в базе данных и может использоваться после завершения данного приложения.

При проектировании менеджера типов СУБД Омега мы также используем концепцию *инкапсулированного типа данных* [8]. Инкапсулированный тип данных предусматривает защиту описания его операций от внешнего воздействия, то есть пользователь данного типа имеет доступ только к тем объектам, которые указаны во внешнем представлении данного типа.

Менеджер типов данных СУБД Омега рассматривает любой тип данных как инкапсулированный абстрактный тип данных. Менеджер типов предоставляет пользователю функции для определения внеш-

него представления и операций типа данных.

Система типов, поддерживаемых СУБД Омега, строится следующим образом. Менеджер типов предоставляет конечное множество *стандартных типов* данных. Стандартный тип — один из типов данных, определенных в языке баз данных SQL92 [11] . Внешнее представление и операции стандартного типа данных предопределены. В настоящее время менеджер типов данных СУБД Омега поддерживает в качестве стандартных типы CHAR и INTEGER. В дальнейшем множество стандартных типов может быть расширено. Прикладной программист с помощью функций менеджера типов определяет внешнее представление и операции *пользовательских типов* на базе стандартных и ранее определенных пользовательских типов.

Далее мы покажем, как формально определяются внешнее представление и операции пользовательского типа данных.

2.2. Внешнее представление типа данных

Пусть имеется непустое множество $B = \{T_1, T_2, ..., T_m\}$, элементы которого — стандартные и ранее определенные пользовательские типы. Множество B назовем *базисом* для построения нового пользовательского типа.

Пусть имеется также непустое множество $N = \{P_1, P_2, ..., P_n\}$, элементы которого — символьные строки. Натуральное n определяется пользователем. Множество N содержит имена свойств нового пользовательского типа (см. ниже).

Тогда *пользовательский тип* — это множество троек $U = \{< i : P_i : T_i >\}_{i \in 1...n}$, где $P_i \in N$, $T_i \in B$, и в данном множестве все P_i различны, а соответствующие им T_i не обязательно различны.

При этом тройка $< i : P_i : T_i >$ называется i-м *свойством типа*, P_i называется *именем* i-го *свойства*, а T_i — *типом* i-го *свойства*.

Частным случаем пользовательского типа является *тип список*, определяемый как пользовательский тип, базис которого состоит в точности из одного элемента. *Экземпляр* пользовательского типа U – это множество троек $\{< i : P_i : v_i >\}_{i \in 1..n}$, где v_i есть значение i-го свойства типа U. Если E – экземпляр пользовательского типа, то запись $E.P_i$ означает значение i-го свойства экземпляра E.

Таблица 1

Примеры построения пользовательских типов данных

№ шага	Имя типа	Базис	Свойства типа
1	Дата	{INTEGER}	{<1 :“Год” : INTEGER>, <2 :“Месяц” : INTEGER>, <3 : “День”: INTEGER>}
2	ФИО	{CHAR}	{<1 : “Фамилия” : CHAR>, <2 : “Имя” : CHAR>, <3 : “Отчество” : CHAR>}
3	Персона	{INTEGER, CHAR, Дата, ФИО}	{<1 : “Ф.И.О.” : ФИО>, <2 : “Дата рождения” : Дата>, <3 : “Пол” : CHAR>, <4 : “Число детей” : INTEGER>}

Следует отметить, что пользовательский тип {<1 : “День” : INTEGER>, <2 : “Месяц” : INTEGER>, <3 : “Год”: INTEGER>}, согласно данному нами определению, не является идентичным типу Дата.

2.3. Операции типа данных

Для работы с экземплярами пользовательского типа данных мы предусматриваем стандартные и пользовательские операции. К *стандартным* мы относим следующие минимально необходимые при работе с записями файла операции: перевод экземпляра указанного типа в символьную строку и сравнение двух экземпляров одного и того же типа. К *пользовательским* мы относим все остальные операции типа данных, которые задаются пользователем с помощью специальной нотации. Описание пользовательской операции интерпретируется менеджером типов и сохраняется в базе данных, обеспечивая персистентность типа. Ниже мы дадим определения стандартных операций и опишем нотацию для задания пользовательских операций типа.

2.3.1. Стандартные операции типа данных

Введем отношение порядка между экземплярами пользовательского типа данных следующим образом. Отношение порядка для экземпляров стандартных типов существует по определению (числа, символьные строки и значения прочих стандартных типов сравниваются обычным образом).

Пусть $U = \{< i : P_i : T_i >\}_{i \in 1..n}$ – пользовательский тип с базисом B, $E1$ и $E2$ — экземпляры типа U. Тогда $E1 > E2$, если либо $E1.P_1 > E2.P_1$, либо $\exists k \in 1..n\ \forall i < k\ E1.P_i = E2.P_i$ и $E1.P_k > E2.P_k$; $E1 = E2$, если для всех $i \in 1..n\ E1.P_i = E2.P_i$; иначе $E1 < E2$. При этом для каждого пользовательского типа $T_j \in B$ отношение порядка вводится аналогичным образом.

Операция сравнения $Compare(T, E1, E2)$ экземпляров $E1$ и $E2$ типа T определяется нами как целочисленная функция, которая возвращает 1, если $E1 > E2$, 0, если $E1 = E2$, и -1 в случае, если $E1 < E2$.

Примеры выполнения операции сравнения экземпляров пользовательского типа данных приведены в табл. 2.

Определим теперь операцию перевода экземпляра типа в символьную строку. Пусть функция $ToChar(T, E, d)$ возвращает результат преобразования экземпляра E типа T в символьную строку, где значения свойств разделяются символьной строкой d. Пусть функция $Concat(S_1, S_2, ..., S_n)$ возвращает символьную строку, которая представляет собой сцепление символьных строк $S_1, S_2, ..., S_n$ в порядке их перечисления.

Для экземпляров стандартных типов результат функции $ToChar$ известен по определению (числа, символьные строки и значения прочих стандартных типов преобразуются в символьную строку обычным образом).

Пусть $U = \{< i : P_i : T_i >\}_{i \in 1..n}$ – пользовательский тип с базовым множеством B, E – экземпляр типа U. Тогда операция преобразования экземпляра E типа T в символьную строку определяется следующим образом:
$ToChar(U, E, d) = Concat(ToChar(T_1, E.P_1, d), d,$
$ToChar(T_2, E.P_2, d), ..., d, ToChar(T_n, E.P_n, d)).$

При этом для каждого значения свойства $E.P_i$, имеющего поль-

зовательский тип $T_i \in B$, результат функции $ToChar$ определяется аналогичным образом.

Примеры выполнения операции преобразования экземпляра пользовательского типа данных в символьную строку приведены в табл. 3.

2.3.2. Пользовательские операции типа данных

Для поддержки пользовательских операций типа данных в реляционной СУБД обычно используется следующий подход [4]. Описание

Таблица 2

Примеры выполнения стандартной операции сравнения экземпляров

Имя типа	Операнд 1	Операнд 2	*Compare*
Дата	{<1 :“Год” : 1970>, <2 :“Месяц” : 12>, <3 : “День”: 15>}	{<1 :“Год” : 1970>, <2 :“Месяц” : 12>, <3 : “День”: 15>}	0
ФИО	{<1 : “Фамилия” : Иванов>, <2 : “Имя": Иван>, <3 : “Отчество": Петрович>}	{<1 : “Фамилия” : Иванов>, <2 : “Имя” : Иван>, <3 : “Отчество” : Иванович>}	-1
Персона	{<1 : “Ф.И.О.” : {<1 : “Фамилия” : Иванов>, <2 : “Имя” : Иван>, <3 : “Отчество” : Иванович>}>, <2 : “Дата рождения” : {<1 :“Год” : 1970>, <2 :“Месяц” : 12>, <3 : “День”: 15>} >, <3 : “Пол” : МУЖ>, <4 : “Число детей” : 3>}	{<1 : “Ф.И.О.” : {<1 : “Фамилия” : Иванов>, <2 : “Имя” : Иван>, <3 : “Отчество” : Иванович>}>, <2 : “Дата рождения” : {<1 :“Год” : 1970>, <2 :“Месяц” : 12>, <3 : “День”: 15>} >, <3 : “Пол” : МУЖ>, <4 : “Число детей” : 0>}	1

Таблица 3

Примеры выполнения стандартной операции преобразования экземпляра в строку

Имя типа	Разделитель	Операнд	*ToChar*
Дата	"-"	{<1 :"Год" : 1970>, <2 :"Месяц" : 12>, <3 : "День": 15>}	1970-12-15
Дата	"."	{<1 :"Год" : 1970>, <2 :"Месяц" : 12>, <3 : "День": 15>}	1970.12.15
ФИО	BLANK (пробел)	{<1 : "Фамилия" : Иванов>, <2 : "Имя" : Иван>, <3 : "Отчество" : Иванович>}	Иванов Иван Иванович
Персона	","	{<1 : "Ф.И.О." : {<1 : "Фамилия" : Иванов>, <2 : "Имя" : Иван>, <3 : "Отчество" : Иванович>}>, <2 : "Дата рождения" : {<1 :"Год" : 1970>, <2 :"Месяц" : 12>, <3 : "День": 15>} >, <3 : "Пол" : МУЖ>, <4 : "Число детей" : 3>}	Иванов, Иван, Иванович, 15, 12, 1970, МУЖ, 3

операции выполняется с использованием специальной нотации. Пользователь указывает имя, параметры операции и их типы. Реализация операции выполняется на внешнем по отношению к СУБД определенном языке программирования (например, Си). В описании операции пользователь должен указать имя файла с исходным текстом реализации. Во время выполнения операции СУБД осуществляет вызов соответствующего внешнего компилятора и выполнение полученного кода. Данный подход, по нашему мнению, имеет определенные недостатки.

Выход за рамки СУБД в определении реализации пользовательской операции приводит к тому, что СУБД не может гарантировать целостность данных указанного пользовательского типа, поскольку пользователь сам отвечает за отсутствие синтаксических и логических ошибок в исходном тексте реализации. Кроме того, использование внешнего компилятора при выполнении пользовательской операции может привести к большим накладным расходам: компилятор является продуктом сторонней команды разработчиков и, как правило, не учитывает аппаратно-программные особенности СУБД.

В данной статье предлагается другой подход к поддержке пользовательских операций типа данных. Он предполагает замену внешнего по отношению к СУБД языка программирования и соответствующего компилятора на внутреннюю нотацию для записи операции и соответствующий интерпретатор. Описание операции предполагается хранить не во внешнем файле, а в словаре базы данных. *Словарь базы данных* представляет собой набор таблиц, хранящих системную информацию о базе данных (количество и имена таблиц, названия и типы полей в таблицах и др.). Пользователи базы данных имеют доступ по чтению к словарю, и могут производить выборку данных из его таблиц. Словарь базы данных дополняется таблицами, в которых будет храниться описание операции.

В СУБД встраивается *интерпретатор описания* операций. Интерпретатор — это языковый процессор, который последовательно выполняет программу на исходном языке, в отличие от компилятора, создающего исполняемый код данной программы [12]. В нашем случае под выполнением описания операции мы понимаем синтаксический разбор этого описания, создающий такую конечную цепочку атомов [12], которую мы можем сохранить в вышеуказанных таблицах словаря базы данных без изменения структуры этих таблиц. При выполнении пользовательской операции СУБД последовательно выполняет каждый атом цепочки синтаксического разбора этой операции, осуществляя поиск в соответствующих таблицах словаря базы данных. Использование интерпретатора позволит СУБД контролировать целостность данных пользовательского типа и снизить накладные расходы при выполнении пользовательской операции, однако требует адекватной нотации для описания этой операции. Для описания пользовательской операции типа данных предлагается *функциональ-*

ная нотация, синтаксис которой схож с одним из функциональных языков программирования. Функциональная парадигма программирования [13] рассматривает программу как процесс вычисления значения математических функций. Значение данных функций определяется лишь их аргументами, а не контекстом выполнения. Кроме того, порядок вычисления выражений, задающих математические функции, управляется рекурсивными и условными выражениями, а не итеративным повторением и последовательностью выполнения операторов, как в нефункциональных языках программирования, например в Си.

Для данной нотации является существенным требование сохранения результатов синтаксического разбора описания операции в таблицах словаря базы данных. В частности, максимальное количество фактических параметров в операторе вызова функции должно быть ограничено некоторой константой, поскольку таблицы словаря базы данных, сохраняющие атомы синтаксического разбора, должны иметь аналогичное число столбцов.

Таким образом, мы определяем описание *пользовательской операции типа данных* как последовательность операторов вызова подпрограммы. Подпрограмма в данном случае — это функция, возвращающая результат стандартного типа. Функция может быть зарезервированной или пользовательской (ранее определенной пользовательской операцией типа данных). На зарезервированные и пользовательские функции накладываются следующие ограничения:

1. *Количество формальных параметров функции* не превышает заранее фиксированного числа. Количество параметров функции является параметром компиляции менеджера типов данных СУБД Омега. В настоящей реализации менеджера типов максимальное количество параметров функции равно двум.

2. При вызове функции в качестве ее параметра может фигурировать переменная, значение или вложенный вызов другой функции. При этом *уровень вложенности вызовов функций* не превышает заранее фиксированного числа. Уровень вложенности вызовов функций является параметром компиляции менеджера типов. В настоящей реализации менеджера типов максимальный уровень вложенности вызовов функций равен двум.

3. Все *фактические параметры* передаются в функцию *по ссылке*, то есть все изменения формального параметра функции происходят непосредственно над фактическим параметром [13].

4. *Функция не может быть рекурсивной*, то есть не должна содержать вызовы самой себя.

5. Типы формальных и фактических параметров функции должны совпадать. Соответствие между формальными и фактическими параметрами функции позиционное.

Ниже приведен пример описания пользовательской операции типа данных. В данном случае тип данных — пользовательский тип "Дата", в котором каждый месяц года состоит из 30 дней, а операция — добавление дней к дате.

```
INCDAYS(DATE D, INTEGER Days) RETURN INTEGER
BEGIN
ADD(Days, MUL(D.YEAR, 360)); /* Дни:=Дни+Д.Год*360 */
ADD(Days, MUL(D.MONTH, 30));
ADD(Days, D.DAY);
MOV(D.YEAR, DIV(Days, 360)); /* Д.Год:=Дни/360 */
SUB(Days, MUL(D.YEAR, 360)); /* Дни:=Дни - Д.Год*360 */
MOV(D.MONTH, DIV(Days, 30));
SUB(Days, MUL(D.MONTH, 30));
MOV(D.DAY, Days);
RET(1);
END INCDAYS
```

3. Заключение

В статье был представлен подход к поддержке пользовательских типов данных в реляционной СУБД. Данный подход базируется на концепциях абстрактного и инкапсулированного типов данных. В соответствии с этим тип данных рассматривается как совокупность внешнего представления экземпляров данного типа и набора операций над экземплярами данного типа. Система поддерживаемых типов

данных строится как объединение множества стандартных и пользовательских типов. Множество стандартных типов фиксировано и состоит из типов, определенных в языке баз данных SQL. Пользовательский тип строится на базе стандартных и ранее определенных пользовательских типов. Внешним представлением пользовательского типа является последовательность стандартных и ранее определенных пользовательских типов. Описание пользовательской операции задается с помощью специальной нотации. Пользовательская операция рассматривается как функция, которая возвращает результат стандартного типа данных и состоит из операторов вызова стандартных и ранее определенных пользовательских операций. СУБД интерпретирует описание и сохраняет результат интерпретации в словаре базы данных, обеспечивая персистентность нового пользовательского типа.

Данный подход был реализован при разработке менеджера типов данных параллельной СУБД Омега для отечественного многопроцессорного вычислительного комплекса МВС- 100.

Список литературы

1. **Когаловский М.Р.** Энциклопедия технологий баз данных. М.: Финансы и статистика, 2002. 800 с.
2. **Дейт К. Дж.** Введение в системы баз данных. 7-е изд. М.: Вильямс, 2001. 1072 с.
3. **Зильбершатц А., Стоунбрейкер М., Ульман Д.** Базы данных: достижения и перспективы на пороге 21-го столетия // СУБД. 1996. № 3. С. 103–117.
4. **Stonebraker M.** Inclusion of New Types in Relational Data Base Systems // ICDE 1986: Proc. of the Second Int. Conf. on Data Engineering, February 5-7, 1986, Los Angeles, CA, USA. IEEE Comput. Soc. Press, 1986. P. 262–269.
5. **Соколинский Л.Б., Цымблер М.Л.** Проект создания параллельной СУБД Омега на базе суперкомпьютера МВС-100/1000 // Телематика'98: Тез. докл. Всерос. науч.-метод. конф. (7-10 июня 1998 г., Санкт-Петербург). СПб.: Вузтелекомцентр, 1998. С. 154–155.
6. **Соколинский Л.Б., Цымблер М.Л.** Принципы реализации системы управления файлами в параллельной СУБД Омега для МВС-100 // Вестн. Челяб. ун–та. Сер. 3. Математика, механика, информатика. 1999. № 2(5). С. 176–199.

7. **Дал У., Дейкстра Э., Хоар К.** Структурное программирование. М.: Мир, 1975. С. 98–197.

8. **Агафонов В.Н.** Типы и абстракция данных в языках программирования (обзор) // Данные в языках программирования. М.: Мир, 1982. С. 265–327.

9. **Liskov B.** Introduction to CLU // New Directions in Programming Languages. IRIA, 1975. P. 139–156.

10. **Wulf W., London R., Shaw M.** An Introduction to the Construction and Verification of Alphard programs // IEEE Trans., SE-2, 1976. № 4. P. 253–265.

11. **Date C.J., Darwen H.** A Guide to the SQL Standard. Reading, Mass.: Addison-Wesley, 1993.

12. **Льюис Ф., Розенкранц Д., Стирнз Р.** Теоретические основы проектирования компиляторов. М.: Мир, 1979. 655 с.

13. **Себеста Р.** Основные концепции языков программирования. М.: Вильямс, 2001. 672 с.

Челябинский государственный университет,
mzym@csu.ru