Использование объектно-ориентированной технологии доступа к данным интервального типа

А. И. Мирошников[1], П. В. Сараев[2], А. В. Галкин[3]
Липецкий государственный технический университет
[1]a.i.miroshnikov@yandex.ru, [2]psaraev@yandex.ru, [3]avgalkin82@mail.ru

***Аннотация*. В статье рассматриваются подходы к организации хранения и доступа к данным интервального типа в реляционных базах данных. Анализируется время выполнения запросов различных видов при реализации пользовательских типов данных в СУБД и при объектно-ориентированной технологии доступа к данным.**

Ключевые слова: информационные системы; базы данных; интервальный тип данных; интервальные операции; объектно-ориентированные технологии доступа

I. Введение

Широкое внедрение информационных технологий, цифровая трансформация предприятий в условиях перехода к Индустрии 4.0 приводят к новым задачам по разработке методов генерирования, хранения, анализа и передачи данных. Объемы данных, собираемых при функционировании процессов, постоянно увеличиваются, что приводит к необходимости разрабатывать специализированные рациональные подходы их обработки. При принятии оптимальных управленческих решений требуется извлечение как можно большего числа знаний из имеющихся данных, выявление закономерностей. Даже несмотря на текущий уровень аппаратного развития вычислительной техники для использования интеллектуальных методов выявления зависимостей, например построение нейросетевых моделей, требуются значительные вычислительные затраты, и адекватные результаты не всегда могут быть получены за приемлемое время. Кроме того в больших объемах собираемых данных могут содержаться погрешности измерений, которые в случае совместной обработки накапливаются и значительно влияют на итоговый результат. Одним из способов решения данной проблемы является предобработка исходных данных с целью сокращения их объемов и учета погрешностей, в них содержащихся. Для этих целей подходит интервальное представление данных, позволяющее учесть погрешности и агрегировать определенные наборы данных путем задания нижней и верхней границы возможных значений. Методы работы с интервальными значениями изучаются в интервальном анализе и находят достаточно широкое применение на практике. Причем эти методы могут значительно отличаться от методов обработки скалярных данных [1, 2].

Для широкого практического использования подходов к интервальному представлению данных необходима реализация их хранения и прозрачного использования в промышленных СУБД и в средах разработки прикладных приложений [3]. Данная работа содержит описание реализованного подхода создания интервального типа данных в СУБД SQL Server, а также возможности его реализации при объектно-ориентированной технологии доступа к данным. Выявлены достоинства и недостатки такого подхода и сложность его реализации.

Исследование выполнено при финансовой поддержке РФФИ и Администрации Липецкой области в рамках научного проекта N 16-47-480929-р_а.

II. Теоретический анализ

A. *Подход, основанный на применении пользовательских типов данных*

В результате теоретических, методологических и экспериментальных исследований в области организации интервальных баз данных и выполнения интервальных запросов с целью аналитической обработки информации, в системе Microsoft SQL Server был разработан и реализован подход к представлению интервального типа данных при помощи пользовательского типа данных UDT (User-Defined Type), названного Interval [4–6].

Данный тип описывается в виде библиотеки классов на языке C# и подключается к MS SQL Server в виде скомпилированной сборки. Основными полями класса являются два вещественных числа down и up, хранящих нижнюю и верхнюю границы интервала, соответственно. В единственном конструкторе класса, принимающем эти значения, производится проверка down < up и выполняется округление нижней границы вниз, а верхней – вверх с определённой точностью, задаваемой в виде константного поля.

Для хранения данных на диске был принят формат записи границ интервала в строке через разделитель в виде точки с запятой при использовании интерфейса Serializable. Поскольку значения границ интервала в базе данных могут быть не заданы, для класса Interval реализуется интерфейс INullable.

Расчет специальных характеристик интервальных значений, например, радиуса и ширины интервала, реализуется в специальных методах класса Interval.

Поскольку в MS SQL Server отсутствует возможность переопределять для пользовательских типов данных такие агрегатные функции, как COUNT, SUM, AVG, в классе реализованы их аналоги, для каждого из которых требовалось переопределение четырёх обязательных методов: инициализации начальным значением, обработка следующего элемента, получаемого из столбца таблицы, объединение множеств промежуточной выборки (в том числе при использовании нескольких потоков обработки) и завершения операции с возвращением результата [7].

Для интервальных типов данных однозначно нельзя определить операции сравнения. Это связано с тем, что интервалы могут иметь различную длину, их границы могут быть вложены друг в друга или пересекаться в различных вариациях. В классе Interval реализованы методы сравнения интервалов, использующие параметр вероятности и представляющие собой перегрузку стандартных операторов сравнения встроенных типов [8].

Благодаря реализации операторов сравнения интервалов, становится возможным применение индексирования столбцов таблиц данных такого типа. Использование кластеризованных, некластеризованных и покрывающих индексов позволяет ускорить выборку интервальных данных. Несмотря на то, что при индексировании требуется вызов дополнительных функций, таких как функции сравнения и разбора формата хранения интервалов, задержка времени выполнения запросов составляет не более 1,5-3% при выборке из таблиц с 100000 строк интервальных данных.

Данный подход предполагает реализацию интервального типа данных в пределах SQL Server, то есть все операции выполняются на сервере данных, а прикладному приложению передаются результаты запроса. При этом разработчик клиентских программ имеет возможность писать прозрачные SQL-запросы с применением простого синтаксиса, получая автоматические проверки целостности интервальных данных, возможность использования нового типа данных, как единого объекта, в отличие от необходимости использования двух или более полей для описания этого типа.

B. Объектно-ориентированные технологии доступа

Ещё одним подходом к реализации интервального типа данных является использование объектно-ориентированной технологии доступа к данным (Object-relational mapping – ORM) при разработке прикладных приложений, использующих реляционные базы данных. Примером такой технологии является ADO.Net Entity Framework. Использование библиотек Entity Framework позволяет отображать реляционную модель базы данных на множество классов, и наоборот, предоставляя разработчикам прикладных программ получить ряд преимуществ, связанных с отсутствием необходимости писать SQL-код [9, 10]. DDL инструкции выполняются средствами Entity Framework, отображая классы на сущности, а свойства классов – на атрибуты. Если в ADO.Net требовалось вставлять SQL-запросы в код программы, следя за чётким соответствием имён таблиц и полей в базе данных, и в объектно-ориентированном коде (поскольку имена передавались в виде строковых констант), то теперь программистам можно работать только со свойствами классов, а остальную работу берёт на себя ORM.

Entity Framework предоставляет три основных подхода доступа к данным:

- model-first – позволяет создавать базу данных при помощи графического интерфейса, связанного с EDMX (Entity Data Model XML) файлом. В этом случае не требуется написания ни Transact-SQL кода, ни кода на объектно-ориентированном языке, что привносит существенные ограничения при создании пользовательских типов данных;
- database-first – предполагает создание сначала базы данных, например, в приложении SQL Server Management Studio, а затем её отображение на классы пользовательского приложения. Таким образом, возможно производить создание пользовательского типа данных с помощью разработанной библиотеки типа, настраивать индексы и вызывать хранимые процедуры из объектно-ориентированного кода;
- code-first – в этом случае, в первую очередь, создаётся модель классов, на её основе Entity Framework формирует и исполняет код создания базы данных, а запросы к СУБД выполняются посредством LINQ (Language Integrated Query) to Entities запросов.

Поскольку подход code-first является самым гибким из представленных, позволяющий, как использовать возможности языка Transact-SQL на уровне СУБД, так и объектно-ориентированных языков, например C#, на уровне прикладных приложений, рассмотрим варианты реализации интервального типа данных с использованием возможностей, предоставляемых ORM.

III. Методика

A. Наивная реализация интервального типа данных с использованием ORM

Создадим в рамках консольного приложения класс Interval, содержащий свойства Id (обязательное свойство для отображения на идентифицирующее поле таблицы), Down и Up (нижнее и верхнее поля интервала, соответственно) и класс IntervalContext, реализующий контекст данных.

```
public class Interval
{
    public int Id { get; set; }
    public double Down { get; set; }
    public double Up { get; set; }
}
```

Рис. 1. Базовый класс Interval

```
class IntervalContext : DbContext
{
    public IntervalContext()
        : base("DatabaseConnection") { }

    public DbSet<Interval> Intervals
        { get; set; }
}
```

Рис. 2. Контекст данных

В данном случае DatabaseConnection представляет собой строку подключения к пустой базе данных. При выполнении этого кода будет сгенерирован SQL-запрос к СУБД на создание новой таблицы на основе описания класса Interval.

Для добавления новых интервалов в базу данных необходимо воспользоваться соответствующими методами контекста данных.

```
using (IntervalContext db =
    new IntervalContext())
{
    db.Intervals.Add(new Interval
        { Down = 1.0, Up = 2.0 });
    db.Intervals.Add(new Interval
        { Down = 2.0, Up = 4.0 });
    db.Intervals.Add(new Interval
        { Down = 1.0, Up = 3.0 });
    db.SaveChanges();
}
```

Рис. 3. Добавление интервалов в базу данных

Для выборки значений из базы данных воспользуемся LINQ-запросом.

```
using (IntervalContext db =
    new IntervalContext())
{
    var res =
        db.Intervals.Where(p => p.Down <= 5.0);
    foreach (Interval i in res)
        Console.WriteLine("{0}. [{1}; {2}]",
            i.Id, i.Down, i.Up);
}
```

Рис. 4. Запрос на выборку данных с условием

Для реализации проверки правильности вводимых данных и округления их в нужную сторону до записи в базу данных можно реализовать соответствующий конструктор класса Interval.

```
int precision = 10;
public Interval(double down, double up) {
    if (down < up){
        this.Down = Math.Round
            (down, precision,
            MidpointRounding.ToEven);
        this.Up = Math.Round
            (up, precision,
            MidpointRounding.AwayFromZero);
    }
    else
        throw new IntervalException();
}
```

Рис. 5. Конструктор класса Interval

Одними из преимуществ данного подхода является то, что для каждого поля таблицы Interval можно создать индексы, используя средства СУБД, а также использовать хранимые процедуры.

В качестве недостатков можно выделить то, что поддержание целостности различных полей интервала должно реализовываться на уровне СУБД; при использовании стандартных агрегатных функций возникает необходимость написания сложных запросов на языке Transact-SQL так как, во-первых, необходима одновременная обработка левой и нижней границ интервала, представленных двумя независимыми полями, во-вторых, операции сравнения интервалов отличаются от аналогичных операций для стандартных типов данных.

B. Использование комплексных типов

Для того чтобы создать не скалярный тип данных в Entity Framework существуют комплексные типы. Их принцип аналогичен использованию UDT, однако они имеют ряд ограничений:

- для комплексных типов не может выбираться ключевое поле;
- комплексные типы могут быть только свойствами более сложных типов или других комплексных типов;
- комплексные типы не могут иметь null-значение;
- комплексные типы не могут наследоваться от других комплексных типов;
- комплексные типы должны реализовываться в виде отдельного класса.

Создадим комплексный тип iInterval на основе двух вещественных чисел.

```
[ComplexType]
public class iInterval
{
    public double Down { get; set; }
    public double Up { get; set; }

    public iInterval()
    {
    }

    public iInterval(double down, double up)...
}
```

Рис. 6. Комплексный тип iInterval

Тогда для его использования в классе Interval, на основе которого будет создана таблица в базе данных, в конструкторе по умолчанию вызовем создание экземпляра комплексного типа, содержащего границы интервала.

```
public class Interval
{
    public int Id { get; set; }
    public string Name { get; set; }

    public iInterval Values { get; set; }

    public Interval()
    {
        Values = new iInterval();
    }
}
```

Рис. 7. Класс Interval, использующий комплексный тип

Запись в базу данных осуществляется через контекст.

```
using (IntervalContext db =
    new IntervalContext())
{
    db.Interval.Add(new Interval
    {
        Name = "Interval1",
        iInterval = new iInterval (1.0, 3.0)
    });
    db.Interval.Add(new Interval
    {
        Name = "Interval2",
        iInterval = new iInterval(2.0, 5.0)
    });
    db.SaveChanges();
}
```

Рис. 8. Запись интервальных значений в базу данных

IV. Результаты

ORM применяется при создании пользовательских приложений на объектно-ориентированных языках. В отличие от похода использования пользовательских типов данных в СУБД, ORM имеет ряд недостатков, связанных с необходимостью перегрузки операторов сравнения для интервальных типов данных и невозможностью переопределения агрегатов стандартных типов, что приводит к замедлению времени выполнения запросов.

Список литературы

[1] Шарый С.П. Конечномерный интервальный анализ. Институт вычислительных технологий СО РАН. 2016. 617 с.

[2] Калмыков С.А., Шокин Ю.А., Юлдашев З.Х. Методы интервального анализа. Новосибирск: Наука. 1986. 222 с.

[3] Погодаев А.К., Сараев П.В., Татаринов Е.П. Универсальное информационное и программное обеспечение для аналитической обработки данных // Информационные технологии моделирования и управления. 2010. № 4 (63). С. 543-550.

[4] Галкин А.В., Мирошников А.И., Погодаев А.К. Разработка интервального типа данных и операций над ним в системе MS SQL Server // Системы управления и информационные технологии, №1(67), 2017. С. 48-51.

[5] Miroshnikov A.I., Nikolskaya A.A. The research of the interval databases design and interval queries execution methods // Modern informatization problems in the technological and telecommunication systems analysis and synthesis: Proceedings of the XXII-th International Open Science Conference (Yelm, WA, USA, January 2017) / Editor in Chief Dr. Sci., Prof. O.Ja. Kravets. - Yelm, WA, USA: Science Book Publishing House, 2017. 273-278 p.

[6] Погодаев А.К., Мирошников А.И., Никольская А.А., Сараев П.В. Применение интервального типа данных при выполнении алгоритмов в СУБД MS SQL Server // Современные сложные системы управления, Ч.2. Липецк: ЛГТУ. 2017. С. 60-64.

[7] Сараев П.В., Галкин А.В., Мирошников А.И., Никольская А.А. Обработка объектов интервального типа в системе управления базами данных SQL Server // Управление большими системами: материалы XIV Всероссийской школы-конференции молодых учёных / Пермь: Изд-во Перм.нац.исслед.политехн.ун-та, 2017. С. 485-492.

[8] Погодаев А.К., Галкин А.В., Сараев П.В., Мирошников А.И. Сравнение интервальных типов данных в системе MS SQL Server // Системы управления и информационные технологии. 2018. Т. 71. № 1. С. 68-72.

[9] Lerman Julia Programming Entity Framework. O'Reilly Media, 2010. 920 p.

[10] Hirani Z., Tenny L., Gupta N., Driscoll B., Vettor R. Entity Framework 6 Recipes. Apress, 2013. 536 p.