Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Фирсов Михаил Александрович

Рекурсивные запросы в PosDB

Учебная практика

Научный руководитель:
ассистент кафедры ИАС Чернышев Г. А.

Санкт-Петербург
2021

Оглавление

Введение

Реляционные базы данных хранят информацию в виде таблиц. Поэтому для хранения в них каких-нибудь древовидных структур приходиться использовать специальные атрибуты-ссылки на другие строки этой таблицы.

Иногда в базах данных хранятся иерархические данные. Для их обработки International Organization for Standardization (ISO) в стандарт SQL:1999 добавела специальные рекурсивные запросы. Они позволяют обрабатывать данные SQL таблиц рекурсивно. Задавая некоторое начальное условие, иначе говоря начальные узлы рекурсии, можно получить данные находящиеся в некотором иерархическом подчинении от этих начальных узлов.

На начало учебной практики исполнитель колоночной РСУБД PosDB не имеет реализации рекурсивных запросов. Это является существенным недостатком, т.к. пользователю такой системы, желающему получить некоторые данные иерархической структуры, хранящейся в БД (базе данных), придётся вместо одного лаконичного запроса писать множество запросов к небольшому числу одних и тех же таблиц.

1. Постановка задачи

Целью работы является реализация поддержки рекурсивных запросов в РСУБД PosDB. Для её выполнения были поставлены следующие задачи:

1. сделать обзор семантики рекурсивных запросов и её реализаций в других СУБД
2. разработать и реализовать эффективный алгоритм обработки рекурсивных запросов с учётом архитектуры PosDB
3. написать тесты, проверяющие корректность и эффективность этого алгоритма

2. Обзор

2.1. Синтаксис рекурсивных запросов

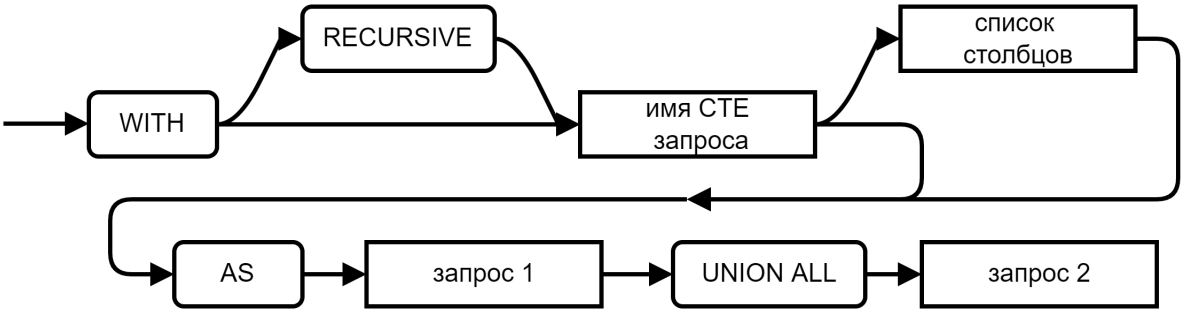

Рис. 1

Общий синтаксис рекурсивных запросов можно представить в виде диаграммы на Рис.1 или описать следующим шаблоном:

```
WITH [RECURSIVE] <имя CTE запроса> [(<список столбцов 0>)]
AS (<запрос 1>)
UNION ALL
<основной запрос 2>
```

2.2. Семантика рекурсивных запросов

- **Common Table Expression (CTE)** — обобщенное табличное выражение, которое можно использовать множество раз в запросе. CTE не сохраняет данные, а создает нечто вроде их временного представления [9].

- **<запрос 1>** является обычным, **<основной запрос 2>** — рекурсивным [8].

- **(<список столбцов 0>)** может быть выведен из списка столбцов запроса 1.

Количество столбцов CTE и рекурсивных элементов должно совпадать. Тип данных столбца в рекурсивном элементе должен совпадать с типом

данных соответствующего столбца в CTE. Рекурсивными могут быть только **монотонные запросы**. Если для любых входных наборов V_1, V_2, где $V_1 \subset V_2$, результат запроса на V_1 является подмножеством результата запроса на V_2, то такой запрос называется монотонным. Монотонность позволяет не пересчитывать результат рекурсивной части запроса на строках, полученных ранее, а просто добавлять в результат запроса данные, полученные подстановкой в CTE только строк, полученных на последнем шаге рекурсии. По этой причине, следующие конструкции не могут использоваться в рекурсивных запросах [8]:

- Агрегация
- Подзапрос, использующий рекурсивные запросы
- Разница множеств

2.3. Примеры

Worker_ID	Master_ID
1	NULL
2	5
3	1
4	3
5	4
6	5
7	5

Таблица 1

Worker_ID	Master_ID
1	NULL
3	1
4	3
5	4
2	5

Таблица 2

Пусть дана Таблица 1. Требуется найти всех косвенных подчиненных начальника (работника с Master_ID = NULL) вплоть до работников с неявным подчинения через 3 промежуточных начальников. Следующий запрос это сделает.

```
WITH RECURSIVE person(Worker_ID, Master_ID)
AS SELECT Table1.Worker_ID, Table1.Master_ID FROM
Table1 WHERE Table1.Master_ID = NULL
```

```
UNION ALL
SELECT Table1.Worker_ID, Table1.Master_ID FROM person JOIN Table1
ON Table1.Master_ID = person.Worker_ID
OPTION (MAXRECURSION 4);
```

Здесь person — это временное представление, которое неявно включает все пары (Table1.Worker_ID, Table1.Master_ID), найденные на данном шаге рекурсии. OPTION (MAXRECURSION 4) говорит о том, что нужно провести максимум 4 итерации рекурсивной части запроса. Результатом исполнения запроса станет Таблица 2.

2.4. Системно-зависимые особенности рекурсивных запросов

2.4.1. Microsoft SQL Server

Помимо основного синтаксиса, описанного выше, Microsoft SQL Server включает следующие особенности реализации рекурсивных запросов [1]:

- Задание в одном обобщенном табличном выражении нескольких предложений WITH недопустимо. Например, если запрос 1 содержит вложенный запрос, этот вложенный запрос не может содержать вложенное предложение WITH, определяющее другое обобщенное табличное выражение.

- Все столбцы, возвращаемые рекурсивным обобщенным табличным выражением, могут содержать значения NULL, независимо от того, могут ли иметь значения NULL столбцы, возвращаемые участвующими инструкциями SELECT.

- Неправильно составленное рекурсивное CTE может привести к бесконечному циклу. Для предотвращения бесконечного цикла можно ограничить количество уровней рекурсии, допустимых для определенной инструкции, при помощи указания MAXRECURSION и значения в диапазоне от 0 до 32 767.

- Представление, содержащее рекурсивное обобщенное табличное выражение, не может использоваться для обновления данных.

- В обобщенном табличном выражении могут быть ссылки на таблицы, находящиеся на удаленных серверах. Если на удаленный сервер имеются ссылки в рекурсивном элементе обобщенного табличного выражения, создается буфер для каждой удаленной таблицы, так что к таблицам может многократно осуществляться локальный доступ.

2.4.2. Oracle

Помимо основного синтаксиса рекурсивных запросов с использованием CTE [4] в Oracle есть альтернативный синтаксис иерархических запросов с использованием ключевого слова CONNECT BY. Его синтаксис можно описать следующей диаграммой [3] на Рис.2.

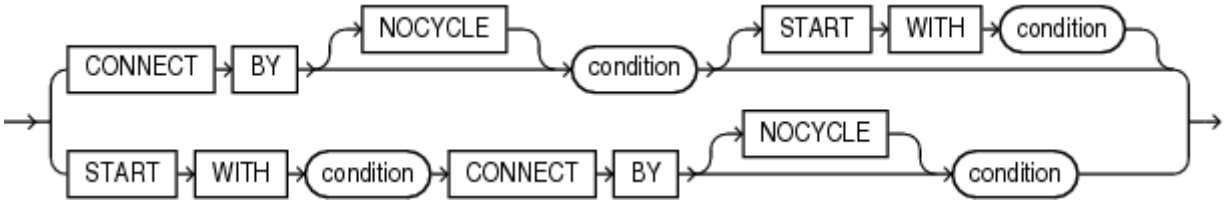

Рис. 2: Рекурсивные запросы в Oracle

condition — некоторое условие.

BEGIN WITH определяет корневую строку (строки) иерархии.

CONNECT BY определяет отношения между родительскими и дочерними строками иерархии.

Параметр **NOCYCLE** указывает Oracle возвращать строки из запроса, даже если в данных существует цикл.

В иерархическом запросе хотя бы одно выражение в условии должно быть дополнено оператором **PRIOR**, чтобы ссылаться на родительскую строку. Например, условие "PRIOR id = manager_id" будет использовано для поиска строк, у которых атрибут manager_id совпадает с атрибутом id некоторой строки, полученной на предыдущих шагах рекурсии.

Пример иерархического запроса в Oracle [3]:

```
SELECT last_name, employee_id, manager_id
```

```
FROM employees
START WITH employee_id = 100
CONNECT BY PRIOR employee_id = manager_id;
```

Здесь будут найдены все строки строки таблицы employees, находящиеся в иерархическом подчинении от начальных строк, у которых значение employee_id равно 100.

В условиях, описываемых после ключевого слова CONNECT BY, а также в разделе SELECT можно использовать специальный атрибут LEVEL, который хранит номер текущей итерации рекурсии.

Пример использования LEVEL в иерархических запросах ORACLE:

```
SELECT last_name, employee_id, manager_id, LEVEL
FROM employees
START WITH employee_id = 100
CONNECT BY PRIOR employee_id = manager_id AND LEVEL <= 4;
```

Здесь мы будет выполнено максимум 3 итерации рекурсии, т.к. для строк полученных с помощью START WITH неявный атрибут LEVEL выставляется в единицу, строк первой итерации рекурсии неявно выставляется в двойку, и т.д.. Кроме того, в итоговой выборке помимо данных самой таблицы мы получим глубину подчинения каждой строки относительно начальных строк.

Возможной причиной введения нового альтернативного синтаксиса рекурсивных запросов в Oracle может являться его большая корректность. Действительно, при таком определении отсутствует проблема несоответствия количества столбцов и их типов в рекурсивной и нерекурсивной частях запроса, которая есть в стандартном синтаксисе.

Рекурсию в SQL на самом деле можно заменить итеративным подходом, т.к. на каждом новом шаге рекурсии могут появиться новые данные, только если они связаны с данными, полученными на предыдущем шаге рекурсии. Действительно, если мы нашли новые данные, которые связаны со строками полученными на шаге до предыдущего или раньше, то эти данные уже были рассмотрены на предыдущем шаге или ещё раньше соответственно. Значит такие строки уже были добавлены в итоговый результат

рекурсивного запроса. Т.е. этот новый синтаксис, который на каждом шаге использует строки только с прошлого шага рекурсии, полностью заменяет стандартный синтаксис рекурсивных запросов, делая код более простым и корректным.

2.4.3. PostgreSQL

Синтаксис рекурсивных запросов в PostgreSQL не отличается от стандартного синтаксиса, описанного выше. Однако в документации PostgreSQL описан непосредственно сам алгоритм вычисления рекурсивных запросов [6]:

1. Вычисляется не рекурсивная часть. Для UNION (но не UNION ALL) удаляются повторяющиеся строки. Все оставшиеся строки включаются в результат рекурсивного запроса и помещаются во временную рабочую таблицу.

2. Пока рабочая таблица не пуста, повторяются следующие действия:

   (a) Вычисляется рекурсивная часть так, что рекурсивная ссылка на сам запрос обращается к текущему содержимому рабочей таблицы. Для UNION (но не UNION ALL) удаляются повторяющиеся строки и ранее полученные строки. Все оставшиеся строки включаются в результат рекурсивного запроса и также помещаются во временную промежуточную таблицу.

   (b) Содержимое рабочей таблицы заменяется содержимым промежуточной таблицы, а затем промежуточная таблица очищается.

Избавиться от циклов в некоторых ситуациях можно за счет использования UNION вместо UNION ALL.

2.4.4. MariaDB

MariaDB следует стандартному синтаксису рекурсивных запросов. Данные, полученные в рекурсивной части запроса, могут быть усечены, если они неправильно преобразованы: размер атрибута будет преобразован до

правильной ширины, если рекурсивная часть CTE дает более широкие значения для атрибута, чем нерекурсивная часть [2].

2.4.5. SQLite

Семантика рекурсивных запросов в SQLite может быть описана следующей диаграммой на Рис.3.

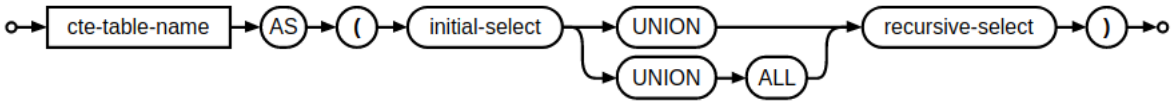

Рис. 3: Рекурсивные запросы в SQLite

Можно выделить следующие особенности реализации рекурсивных запросов в этой СУБД [7]:

- Может быть несколько SELECT в рекурсивной и нерекурсивной частях запроса.
- CTE таблица может появляться только один раз в предложении FROM каждого оператора SELECT, следовательно она не может появляться в подзапросах.
- initial-select может быть составным, но он не может включать ORDER BY, LIMIT или OFFSET. recursive-select также может быть составным с ограничением, что все элементы этого соединения должны быть разделены одним и тем же оператором UNION или UNION ALL. Пример корректного запроса: (нерекурсивная часть запроса) UNION (рекурсивная часть запроса 1) UNION (рекурсивная часть запроса 2)... UNION (рекурсивная часть запроса n).
- recursive-select может включать ORDER BY, LIMIT и(или) OFFSET, но не может использовать агрегатные функции или оконные функции.
- Возможность для recursive-select быть составным была добавлена в версии 3.34.0 (2020-12-01). В более ранних версиях SQLite recursive-select мог быть только одним простым оператором SELECT.

Базовый алгоритм вычисления содержимого рекурсивной таблицы следующий [7]:

1. Результаты(сроки) initial-select добавляются в очередь.
2. Пока очередь не пуста:
   (a) Извлекается одна строка из очереди.
   (b) Эта строка вставляется в CTE таблицу.
   (c) для таблицы из одной этой строки запускается рекурсивная часть запроса.
   (d) Результат работы рекурсивной части добавляется в очередь.

Вышеуказанная основная процедура может быть изменена следующими дополнительными правилами [9]:

- Если соединяющим рекурсивную и нерекурсивную части, является оператор UNION, то в очередь добавляются только строки, которые прежде не обрабатывались.
- Предложение LIMIT, если оно присутствует, определяет максимальное количество строк, которое когда-либо будет добавлено в CTE таблицу. Нулевое ограничение означает, что в CTE таблицу никогда не добавляются строки, а отрицательное ограничение означает, что в CTE таблицу может быть добавлено неограниченное количество строк.

3. Реализация рекурсивных запросов в PosDB

3.1. Устройство PosDB

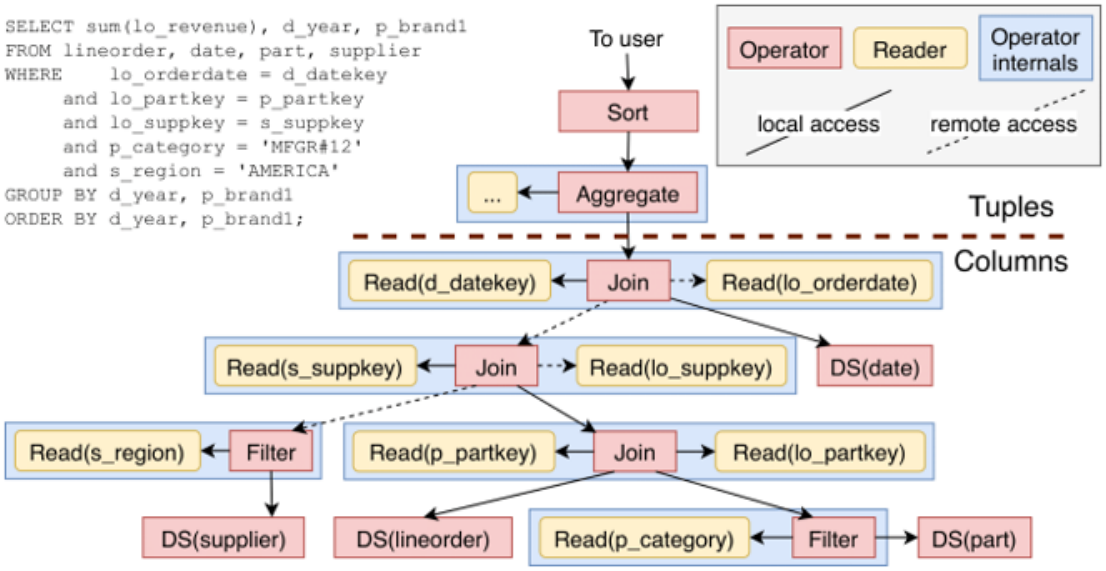

Рис. 4: Пример плана запроса в PosDB

PosDB — распределённая колоночная СУБД [5]. Главные особенности PosDB:

- PosDB хранит каждый атрибут таблицы в отдельном файле. Одна таблица может быть разделена на несколько частей — **партиций**, которые хранятся в разных файлах, возможно, на разных машинах.
- Все запросы на языке SQL, проходя через парсер, переводятся в специальный план запроса. **План запроса**, изображенный на Рис. 4, это диаграмма перехода табличных данных через специальные операторы.
- Исполнение запросов в PosDB основано на модели Volcano с поблочной обработкой: каждый оператор в плане пропускает через себя поток данных и реализует функции open(), next() и close(). next() возвращает блок кортежей или блок позиций. Поэтому большинство операторов делится на позиционные и кортежные.
- Позиционные операторы имеют специальные Reader-ы для чтения отдельных атрибутов таблицы.

- План запроса делится на позиционную и кортежную части. Момент преобразования позиций в кортежи называется **материализацией**.

- Каждый оператор в PosDB имеет два представления: в виде узла логического дерева плана запроса (пространство имён operators) и в виде узла физического дерева плана запроса (пространство имён phys_tree).

3.2. Идеи реализации

Для того, чтобы в дальнейшем реализовать рекурсивные запросы в парсере PosDB было предложено ввести два новых оператора плана запроса. Их иерархию можно представить в виде диаграммы на Рис. 5.

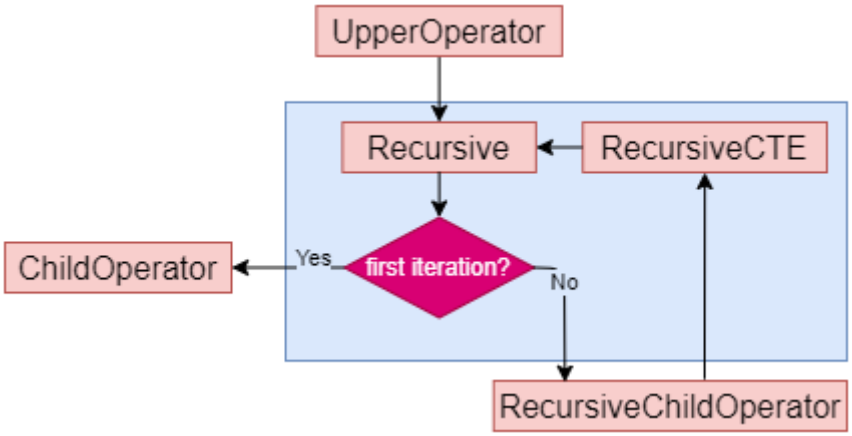

Рис. 5: Примерное изображение плана запроса с использованием Recursive и RecursiveCTE

- Recursive — хранит в себе указатели на RecursiveCTE, ChildOperator и RecursiveChildOperator. ChildOperator используется для нерекурсивной части запроса, с его помощью получим стартовые строки или позиции. Recursive-ChildOperator — обычный оператор, но внутри себя он либо явно, либо косвенно (через несколько промежуточных операторов) получает данные от RecursiveCTE.

- RecursiveCTE — хранит указатель на Recursive, от которого просит новые строки для их передачи методом Next в RecursiveChildOperator.

В PosDB есть два типа блоков: **TupleBlock**, который хранит кортежи, и **PMapBlock**, который хранит позиции. С учетом этого, необходимо ввести две следующих пары операторов:

- **TRecursive** и **TRecursiveCTE** — будут работать только с блоками кортежей.
- **PRecursive и PRecursiveCTE** — будут работать только с блоками позиций.

  Может возникнуть желание использовать гибридный оператор, у которого ChildOperator и RecursiveCTE возвращают блок типа PMapBlock, а RecursiveChildOperator — TupleBlock. При такой реализации, нам придётся кортежи, полученные от RecursiveChildOperator переводить обратно в позиции (это в некоторых случаях невозможно), чтобы использовать для второго и последующих шагов рекурсии. Поэтому использование гибридных операторов для реализации рекурсивных запросов невозможно.

Пример: Пусть задан следующий рекурсивный запрос:

```
WITH RECURSIVE person(Worker_ID, Master_ID)
AS SELECT Table1.Worker_ID, Table1.Master_ID FROM
Table1 WHERE Table1.Master_ID = NULL
UNION ALL
SELECT Table1.Worker_ID, Table1.Master_ID FROM person JOIN Table1
ON Table1.Master_ID = person.Worker_ID
OPTION (MAXRECURSION 4);
```

Дерево этого запроса с использованием введенных структур можно представить с помощью диаграммы на Рис. 6. Здесь:

- Левый оператор Materialize — это ChildOperator, он будет выполнен 1 раз для инициализации стартового набора кортежей.
- Join — это RecursiveChildOperator.

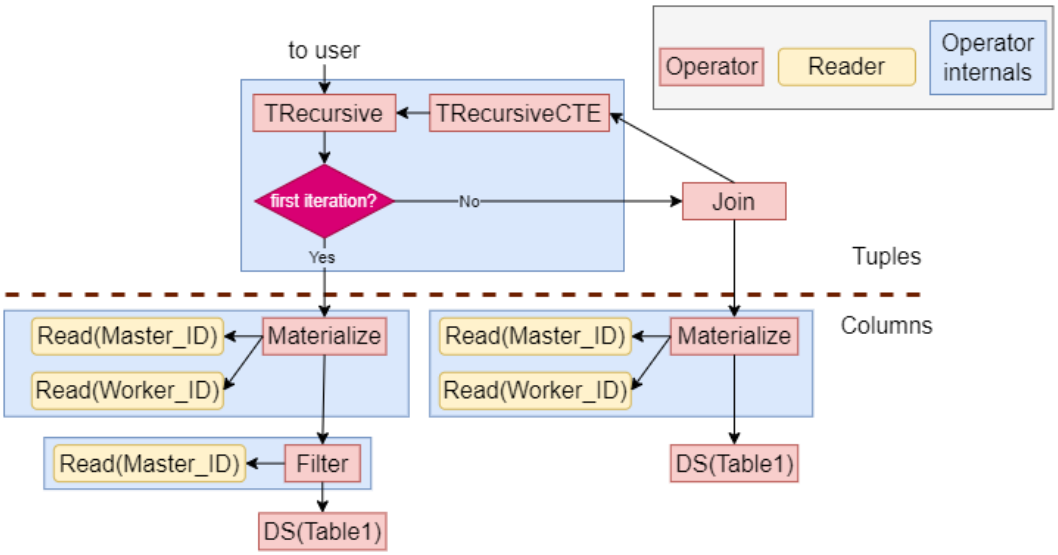

Рис. 6: Дерево запроса с использованием TRecursive и TRecursiveCTE

- Набор блоков кортежей текущего шага рекурсии хранится внутри TRecursive, назовём его curLevel, кроме того TRecursive будет хранить позицию блока в curLevel, который в следующий раз нужно передать TRecursive.

Ход вычислений будет следующим:

1. TRecursive запрашивает блок от левого Materialize, пока они не пусты, и сохраняет их в curLevel.

2. TRecursive передаёт выше все блоки из curLevel, пока не дойдёт до конца curLevel.

3. Чтобы получить блок нового шага рекурсии, TRecursive запросит блок у Join.

4. Join будет запрашивать блоки у TRecursiveCTE и у правого Materialize.

5. TRecursiveCTE будет просить блоки у TRecursive.

6. TRecursive будет, увеличивая внутренний счетчик, передавать TRecursiveCTE в ответ на его запросы блоки из curLevel.

7. Набрав новый блок определенного размера, Join отдаст его TRecursive. Если это непустой блок, то TRecursive сохранит его во временное хранилище nextLevel и перейдёт к шагу 3. Если это пустой блок, то curLevel

заменяется на nextLevel. Если теперь curLevel — это пустой набор блоков, то считаем, что TRecursive закончил свою работу, иначе TRecursive переходит к шагу два.

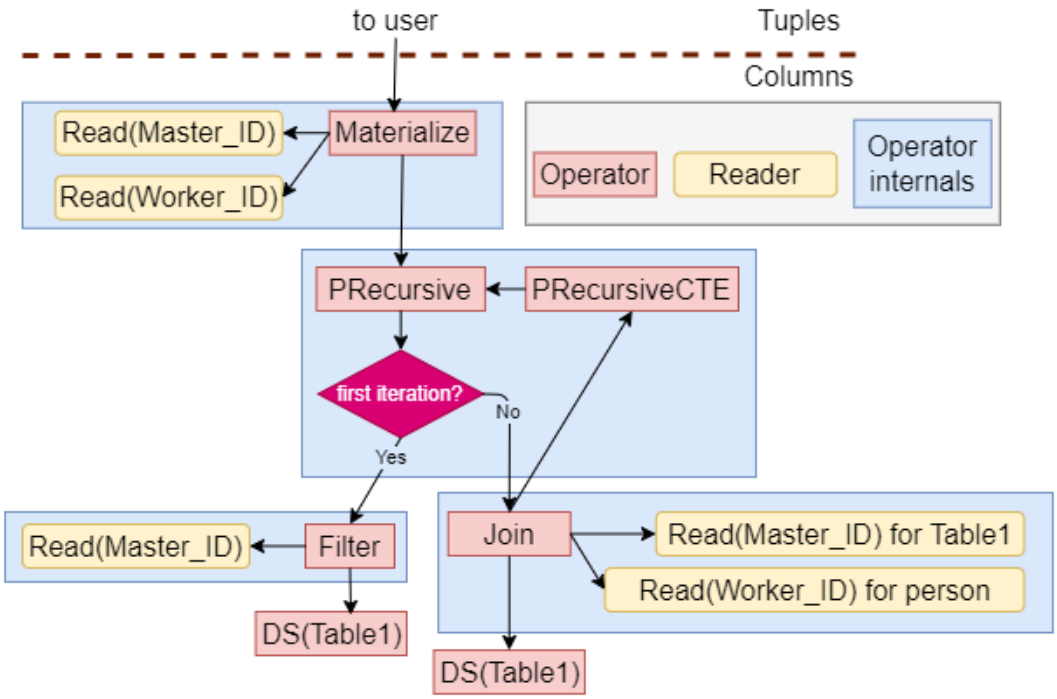

Рис. 7: Дерево запроса с использованием PRecursive и PRecursiveCTE

Вычисления для дерева запроса на рис. 7 с оператором PRecursive будут происходить похожим образом. Важным ограничением будет являться то, что мы можем работать только с позициями одной и той же таблицы, а значит оператор Join обязан возвращать позиции той же таблицы, что и левый оператор Filter. curLevel будет хранить не блоки кортежей, а блоки позиций. В остальном логика работы связки операторов PRecursive и PRecursiveCTE полностью совпадает с логикой TRecursive и TRecursiveCTE.

3.3. Кортежная обработка

Для обработки блоков кортежей были введены физические операторы, изображенные на Рис.8

Опишем поля и методы operators::TRecursive:

- **child** хранит указатель на нерекурсивный оператор, который возвращает кортежи нулевого шага рекурсии.

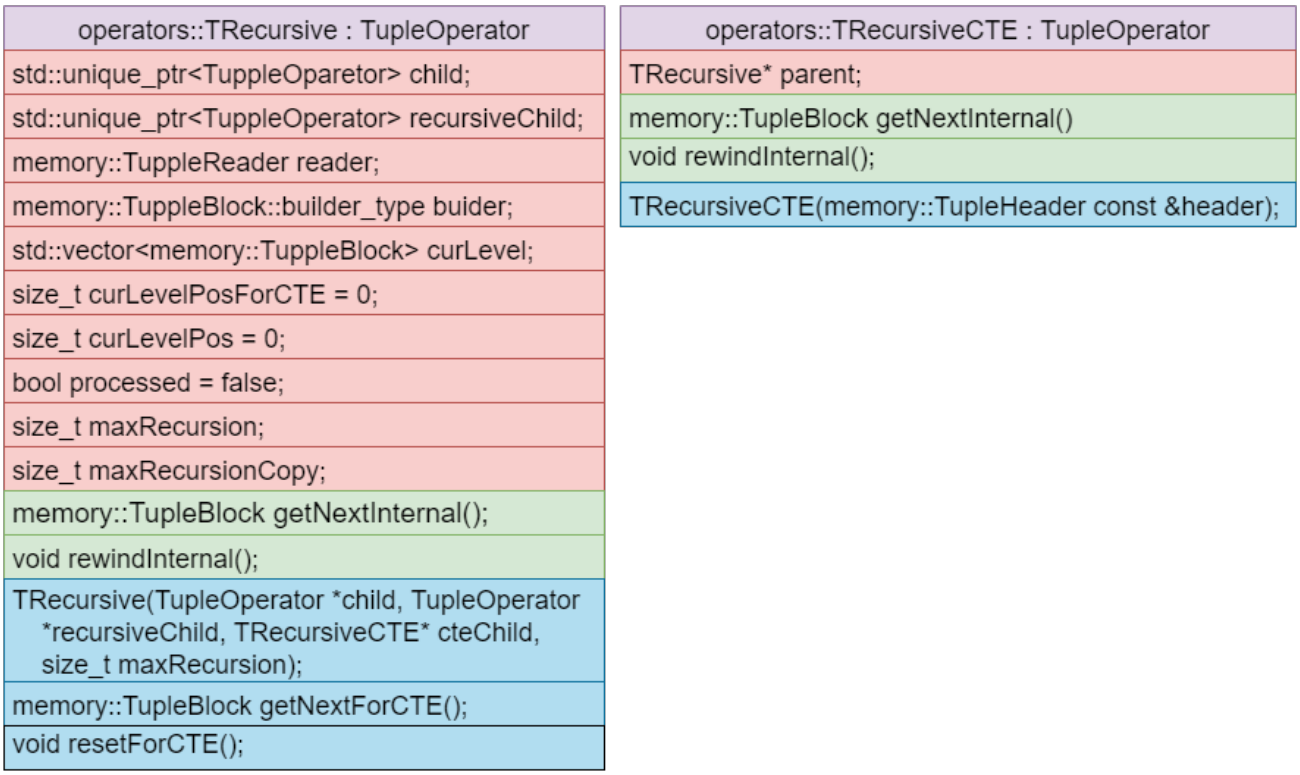

Рис. 8: Структура классов operators::TRecursive, operators::TRecursiveCTE

.

- **recursiveChild** хранит указатель на нерекурсивный оператор, один из потомков которого это operators::TRecursiveCTE.
- **reader** и **builder** нужны для чтения кортежей из специальных кортежных блоков.
- **curLevel** хранит блоки (наборы кортежей) текущего шага рекурсии.
- **curLevelPosForCTE** — позиция блока в curLevel, который нужно будет передать в operators::TRecursiveCTE методом **getNextForCTE**. Если он равен размеру curLevel, то передавать нужно пустой блок.
- **curLevelPos** — позиция блока в curLevel, который следующим нужно будет передать оператору, стоящему над operators::TRecursive, посредством метода **getNextInternal** базового класса **TupleOpeartor**. Если он равен размеру curLevel, то будем запрашивать блоки из recursiveChild, пока они не пусты, после чего обновим curLevel этими новыми блоками. Таким образом, в момент получения этих новых блоков

оператору operators::TRecursiveCTE мы будем передавать ещё старые блоки.

- **processed** — флаг, означающий, что все рекурсивные вызовы закончились. Его значение установим в истинное, если из recursiveChild не пришло ни одного непустого блока.
- **maxRecursionCopy** — максимальное количество итераций рекурсивной части запроса. Если этот параметр равен 0, то количество итераций рекурсии неограниченно.
- **maxRecursion** — счетчик с начальным значением maxRecursionCopy.
- **rewindInternal** обновляет весь оператор TRecursive. Наполняет curLevel за счет блоков, получаемых от child. Он также вызывается в конструкторе operators::TRecursive, для начальной инициализации curLevel.
- **resetForCTE** обнуляет curLevelPosForCTE.

Недостатком такого подхода является отсутствие возможности отследить цикл. Для его отслеживания можно хранить хэш-таблицу всех прошлых кортежей из curLevel. Каждый кортеж каждого блока, полученного от recursiveChild, проверять на существование в этой хэш-таблице. Однако даже такой подход не гарантирует отсутствие циклов. Простой контрпример — это запрос, который в рекурсивной части просто прибавляет 1 к одному из аргументов кортежа. На данный момент было принято решение оставить предотвращение появления циклов на руки пользователя системы.

Оператор operators::TRecursiveCTE делает следующее. В методе getNextInternal возвращает блок, полученный от parent. В методе rewindInternal вызывает метод resetForCTE объекта parent.

Для представления этих операторов физического дерева плана запроса были разработаны операторы логического дерева плана запроса, изображенные на Рис. 9

Для перехода от логического дерева к физическому выполняется вызов метода internalTransform узла логического дерева. Сперва завершается

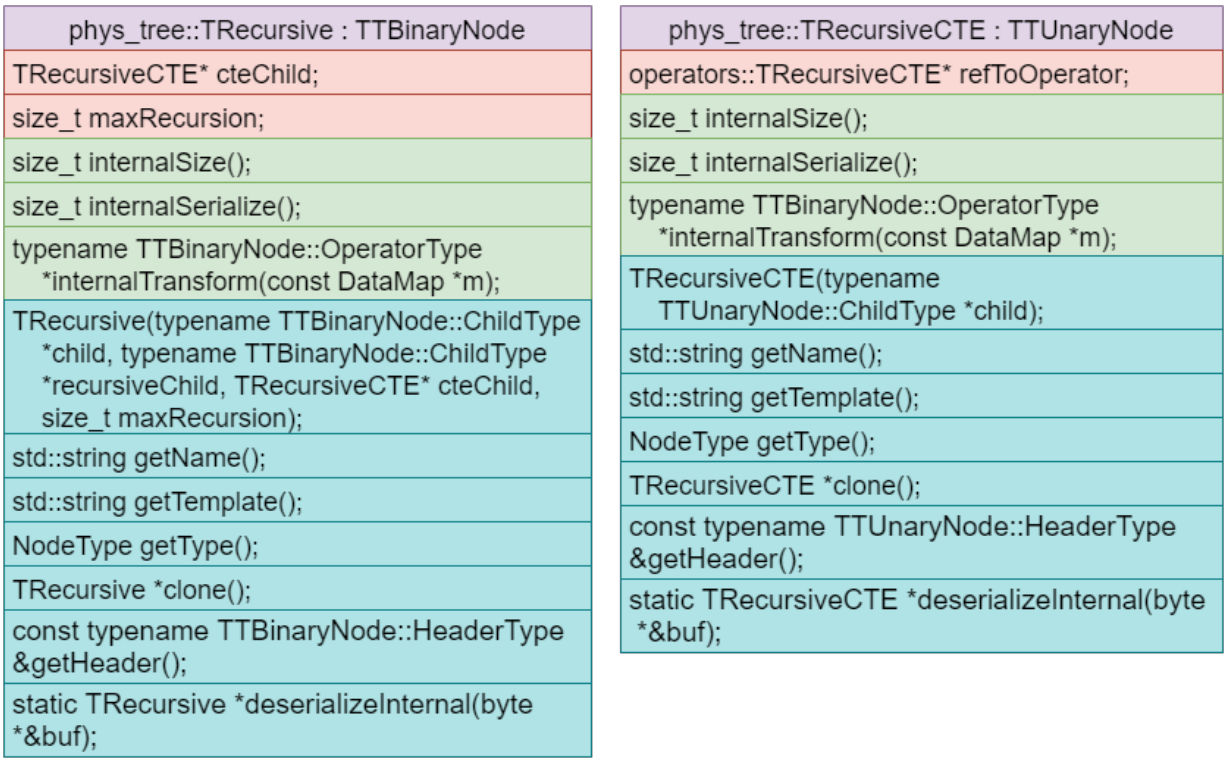

Рис. 9: Структура классов phys_tree::TRecursive, phys_tree::TRecursiveCTE

трансформация для листьев дерева, в затем на основе полученных листьев физического дерева строиться узлы на уровень выше и так далее до вершины дерева. Но нам для построения узла operators::TRecursiveCTE нужен узел operators::TRecursive, который будет построен позже.

Для решения этой проблемы было решено сделать следующее: phys_tree::TRecursive будет содержать указатель на phys_tree::TRecursiveCTE. У **cteChild** метод internalTransform закончит свою работу раньше и результат работы, т.е. указатель на узел физического дерева, сохранит в поле **refToOperator**. В методе internalTransform класса phys_tree::TRecursive мы у поля cteChild возьмем поле refToOperator и передадим в конструктор operators::TRecursive, внутри которого мы полю parent класса operators::TRecursiveCTE зададим значение this, т.е. указатель на нужный нам рекурсивный оператор физического дерева.

3.4. Обработка позиций

Для обработки блоков позиций были введены операторы PRecursive и PRecursiveCTE. Их реализация отличается от реализации кортежных опе-

раторов лишь типом используемых блоков.

3.5. Тесты

Для проверки корректности написанных операторов была создана таблица аналогичная таблице из примера 1. Были написаны запросы на получение следующих данных из этой таблицы:

- Получение всех неявных подчиненных корня, листа, и двух узлов (не листьев и не корней) дерева подчинения. Их аналогом на языке SQL будет запрос вида:

```
WITH RECURSIVE person AS
SELECT T1.id, T1.master_id FROM T1 WHERE T1.id = k
# Здесь k - номер стартового узла
UNION ALL
SELECT T9.id, T9.master_id FROM T9 JOIN person
ON person.id = T9.master_id;
```

- Получение трёх строк в нерекурсивной части запроса и просто тождественный запрос в рекурсивной части запроса с использованием параметра MAXRECURSION. Аналогом на языке SQL будет запрос вида:

```
WITH RECURSIVE person AS
SELECT T1.id, T1.master_id FROM T1 WHERE T1.id < 3
UNION ALL
SELECT T1.id, T1.master_id FROM person
MAXRECURSION 4;
```

Были написаны 10 тестов: 5 для TRecursive, TRecursiveCTE; 5 для PRecursive, PRecursiveCTE.

Заключение

В процессе работы были достигнуты следующие результаты:

- Проанализированы особенности реализаций рекурсивных запросов в основных СУБД.
- Разработаны операторы, реализующие рекурсивные запросы в PosDB.
- Написаны тесты, проверяющие корректность работы разработанных операторов.

Список литературы

[1] MS SQL Server Documentation. — Online; accessed 29 October 2021. Access mode: https://docs.microsoft.com/ru-ru/sql/t-sql/queries/with-common-table-expression-transact-sql?view=sql-server-ver15.

[2] MariaDB Documentation. — Online; accessed 29 October 2021. Access mode: https://mariadb.com/kb/en/recursive-common-table-expressions-overview/.

[3] Oracle SQL Documentation. — Online; accessed 29 October 2021. Access mode: https://docs.oracle.com/database/121/SQLRF/queries003.htm.

[4] Oracle keyword "WITH" Documentation. — Online; accessed 29 October 2021. Access mode: https://docs.oracle.com/cd/E17952_01/mysql-8.0-en/with.html.

[5] Chernishev G. A., Galaktionov V. A., Grigorev V. D., Klyuchikov E. S., and Smirnov K. K. PosDB: An Architecture Overview // Program. Comput. Softw. — 2018. — jan. — Vol. 44, no. 1. — P. 62–74. — Access mode: https://doi.org/10.1134/S0361768818010024.

[6] PostgreSQL Documentation. — Online; accessed 29 October 2021. Access mode: https://postgrespro.ru/docs/postgresql/14/queries-with.

[7] SQLite Documentation. — Online; accessed 29 October 2021. Access mode: https://www.sqlite.org/lang_with.html.

[8] Silberschatz Abraham, Korth Henry F., and Sudarshan S. Database Systems Concepts. — 7th ed. — McGraw-Hill Higher Education, 2020. — ISBN: 978-0-07-802215-9, 978-1-260-51504-6.

[9] Рекурсия в MS SQL. — Online; accessed 29 October 2021. Access mode: https://www.fastreport.ru/ru/blog/show/recursion-mssql/.