|  |  |
| --- | --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  РОССИЙСКОЙ ФЕДЕРАЦИИ  Федеральное государственное автономное образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | |
| *Кафедра математического обеспечения  вычислительных систем* | |
|  | |
| ОТЧЕТ  по заданию № 9  по дисциплине Учебная практика по базам данных  «Нестандартный запрос» | |
|  | Работу выполнила студент группы *ПМИ-1* *3* курса механико-математического факультета  Жуков А.Е.  13 января 2022 г.  Принял:  ассистент каф. МОВС  Турова И.А.  “\_\_\_\_\_” 2021 г. |
| Пермь 2021 | |

# Постановка задачи

Описать основные алгоритмы, используемые в задании.

Написать отчет о проделанной работе.

# Описание функционала разработанной программы

1. Загрузка атрибутов, выбранной базы данных

Для корректной работы необходимы следующие данные: название таблицы, название атрибута, тип атрибута.

Для этого воспользуемся системными каталогами: pg\_class для получения имени таблиц; pg\_attribute для получения атрибутов, pg\_type для получения типов и pg\_namespace для отделения публичных таблиц.

Получился следующий запрос:

SELECT  
 pg\_class.relname as table\_name,  
 pg\_attribute.attname AS attribute\_name,  
 typcategory as type\_category  
FROM  
 pg\_catalog.pg\_attribute  
INNER JOIN  
 pg\_catalog.pg\_class ON pg\_class.oid = pg\_attribute.attrelid  
INNER JOIN  
 pg\_catalog.pg\_namespace ON pg\_namespace.oid = pg\_class.relnamespace  
INNER JOIN  
 pg\_catalog.pg\_type on pg\_attribute.atttypid = pg\_type.oid  
WHERE  
 pg\_attribute.attnum > 0  
 AND NOT pg\_attribute.attisdropped  
 AND pg\_namespace.nspname = 'public'  
 AND pg\_class.relkind='r'  
ORDER BY  
 table\_name ASC, attnum asc ;

Через первые два условия в блоке Where отделяются системные типы, далее указываем схему public и тип relation, чтобы выделить только таблицы

1. Загрузка внешних ключей

Здесь нам необходимы оригинальная таблиц и ее столбец, а также внешняя таблица и ее столбец.

Используем схему information\_schema и таблицы: table\_constraints, key\_column\_usage, constraint\_column\_usage. В конце указываем, что нам необходимы внешние ключи:

SELECT  
 tc.table\_name as original\_table,  
 kcu.column\_name as original\_attribute,  
 ccu.table\_name AS foreign\_table,  
 ccu.column\_name AS foreign\_attribute  
FROM  
 information\_schema.table\_constraints AS tc  
 JOIN information\_schema.key\_column\_usage AS kcu  
 ON tc.constraint\_name = kcu.constraint\_name  
 AND tc.table\_schema = kcu.table\_schema  
 JOIN information\_schema.constraint\_column\_usage AS ccu  
 ON ccu.constraint\_name = tc.constraint\_name  
 AND ccu.table\_schema = tc.table\_schema  
WHERE tc.constraint\_type = 'FOREIGN KEY';

1. Получение полей таблиц

Необходимо также выводить поля таблиц. Используем information\_schema и таблицу columns:

select table\_name, column\_name from information\_schema.columns  
join pg\_tables on table\_name=tablename  
where table\_schema='public'  
ORDER BY table\_name,column\_name, ordinal\_position;

1. Построение запроса
   1. Описание поля SELECT

При описании этого поля, на вход передаем данные, о выбранных полях на форме.

Если поля не выбраны, то выбираем все поля через \*

**private** **string** **BuildSelectClause**(List<Tuple<**string**,**string**>> fields)

{

**var** str = "";

**if** (fields.Count == **0**)

**return** "\* ";

**if** (fields.Count == **1**)

**return** fields[**0**].Item1 + "." + fields[**0**].Item2;

str += $"{fields[0].Item1}.{fields[0].Item2}";

**foreach**(**var** i **in** fields.Skip(**1**))

{

str += $", {i.Item1}.{i.Item2}";

}

**return** str;

}

* 1. Описание FROM

Здесь нам необходимы внешние ключи, чтобы построить связи.

Если выбрана лишь одна таблица, то возвращаем ее.

Далее, необходимо получить пути от начальной таблицы до следующих. Для этого воспользуемся поиском в глубину. Таблицы и связи между ними можно представить в виде графа, где вершины – таблицы, дуги – связи.

Если существует внешний ключ между двумя таблицами, то добавляем его в путь и возвращаем этот путь. Иначе запоминаем, что данная таблица использована, чтобы не возвращаться к ней в дальнейшем обходе.

Далее необходимо получить все соседние таблицы для текущей.

Для каждой вершины проверяем, использована ли она, если да, то переходим к следующей, иначе вызываем рекурсивно поиск пути от этой вершины до конечной.

Если путь найден, то добавляем в него связь от текущей вершины до вершины соседа и возвращаем этот путь.

В случае, если после прохода по всем соседним вершинам для данной путь не нашелся, то возвращаем null.

После того, как нашли все пути, пробегаем по ним и прописываем стандартное соединение в sql-запросе. Также проверяем, чтобы не было повторений и не соединялись уже использованные таблицы.

Функция поиска путей:

**private** List<ForeignKey> GetPathForeignKeys(**string** tableFrom, **string** tableTo, HashSet<**string**> usedTables = **null**)

{

List<ForeignKey> result = **new**();

**if** (usedTables **is** **null**)

usedTables = **new** HashSet<**string**>();

**var** fc = Model.GetForeignKey(tableFrom, tableTo);

**if** (fc != **null**)

{

result.Add(fc);

**return** result;

}

usedTables.Add(tableFrom);

**var** neighboors = Model.GetNeighboorTable(tableFrom);

**foreach**(**var** i **in** neighboors)

{

**if** (usedTables.Contains(i))

**continue**;

**var** path = GetPathForeignKeys(i, tableTo, usedTables);

**if** (path != **null**)

{

fc = Model.GetForeignKey(tableFrom, i);

path.Insert(**0**, fc);

**return** path;

}

}

**return** **null**;

}

* 1. Описание WHERE

Здесь просто собираем все условия и соединяем через AND:

**return** **string**.Join("\nAND ",conditions.Select(c => $"{c.TableName}.{c.AttributeName} {c.Operator} {c.Value}"));