МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Систем обработки информации и управления»

ОТЧЕТ

Лабораторная работа №__2__
по дисциплине «Постреляционные базы данных»

Тема: «Программирование объектно-реляционной базы данных на примере СУБД PostgreSQL»

ИСПОЛНИТЕЛЬ: __Журавлев Н.В.___
ФИО
группа ИУ5-24М _____
подпись
"14"_февраля_2024 г.

ПРЕПОДАВАТЕЛЬ: _Виноградова М.В_
ФИО
_____
подпись
"__"_____202_ г.

Москва - 2024

Цель работы

- Изучить постреляционные возможности языка SQL [1].
- Освоить языки и технологии SQL\PSM на примере PostgreSQL [2].
- Получить навыки программирования на стороне сервера.

Задание

1. Через PgAdmin [3] соединиться с PostgreSQL [2] и создать базу данных. В БД создать две-три связанные таблицы по теме, выданной преподавателем. Открыть таблицы на редактирование и заполнить тестовыми данными.
2. Создать скалярную функцию. Вызвать функцию из окна запроса.
3. Создать табличную функцию (inline). Вызвать функцию из окна запроса.
4. Создать табличную функцию (multi-statement). Продемонстрировать наполнение результирующего множества записей. Вызвать функцию из окна запроса.
5. Создать хранимую процедуру, содержащую запросы, вызов и перехват исключений. Вызвать процедуру из окна запроса. Проверить перехват и создание исключений.
6. Продемонстрировать в функциях и процедурах работу условных операторов и выполнение динамического запроса.

Ход работы

Определяемые пользователем функции создаются посредством инструкции:

```
CREATE OR REPLACE FUNCTION public.scal_fun(min_weight integer)
RETURNS integer AS
'SELECT max(age) FROM public."account" WHERE weight > min_weight'
LANGUAGE SQL
```

Определенную пользователем функцию можно вызывать с помощью инструкций SELECT, INSERT, UPDATE или DELETE. Вызов функции осуществляется, указывая ее имя с парой круглых скобок в конце, в которых

можно задать один или несколько аргументов.

Функция является скалярной, если предложение RETURNS определяет один из скалярных типов данных и возвращает в качестве ответа единственное значение при каждом вызове функции.

Табличная функция возвращает набор строк. За возвращаемое значение отвечает ключевое слово RETURNS. Если тело функции является одним SQL-запросом, то такая функция называется встроенной («inline»). Запрос встроенной функции может рассматриваться как обычный подзапрос с параметром. Инструкция SELECT встроенной функции возвращает результирующий набор в виде переменной с типом данных TABLE

```
CREATE FUNCTION height_div_weight (user_id integer)
RETURNS TABLE(age integer, div real) AS $$
SELECT age, height/weight FROM account WHERE id=user_id
$$ lANGUAGE SQL
```

Для функции типа «multi-statement» переменная, указанная как возвращаемый тип, содержит набор записей, которые возвращаются в качестве результата функции. Код функции должен обеспечить заполнение этой переменной, например, командой «insert».

Для возврата множество записей также используют конструкции вида:

RETURN NEXT выражение; -- добавить в результат строку

RETURN QUERY запрос; -- выполнить запрос и вернуть его результат

RETURN QUERY EXECUTE строка-команды; -- выполнить динамический запрос и вернуть его результат

```
CREATE OR REPLACE FUNCTION public.multi_acc_diets(user_id integer)
RETURNS TABLE (diet_id integer, name_diet varchar) AS $$
BEGIN
IF user_id = 1 THEN
UPDATE account SET comb_.a = 1 WHERE id = user_id;
END IF;
RETURN QUERY
SELECT d.id, d.name
FROM history_diet AS h
```

```
        JOIN diet AS d ON h.id_diet = d.id
    WHERE h.id_account = user_id;
    END
    $$
    LANGUAGE 'plpgsql';
```

Хранимые процедуры — это предварительно откомпилированные процедуры, программы, написанные на SQL/PSM и находящиеся в базе.

Вызов процедуры происходит через команду:

```
call create_table('diet')
```

Команда RAISE предназначена для вывода сообщений и вызова ошибок.

По умолчанию любая возникающая ошибка прерывает выполнение функции на PL/pgSQL и транзакцию, в которой она выполняется. Использование в блоке секции EXCEPTION позволяет перехватывать и обрабатывать ошибки.

Формально синтаксис описания условного оператора можно представить следующим образом:

IF <условие> THEN <операторы> [ELSIF <условие> THEN <операторы>] [ELSE <операторы>] END IF

Динамические запросы — это запросы, текст которых формируется во время выполнения приложения и заранее не компилируется. Под динамическими запросами понимаются запросы SQL, текст которых формируется и затем выполняется внутри PL/pgSQL-блока, например, в хранимых функциях или в анонимных блоках на этом процедурном языке

```
    CREATE OR REPLACE PROCEDURE public.create_table(tablename varchar(100))
    AS $$
    BEGIN
    IF length(tablename) > 3 THEN
    EXECUTE format('CREATE TABLE %I (id integer PRIMARY KEY, title
varchar(100))', tablename);
    ELSE RAISE EXCEPTION USING errcode='E0001',
    message='Длина меньше 3';
    END IF;
```

```
EXCEPTION WHEN SQLSTATE '42P07' THEN
RAISE EXCEPTION USING errcode='E0002',
message='Таблица уже существует';
END
$$
LANGUAGE 'plpgsql';
```

Внутри рекурсивный запрос (то, что записано в скобках после ключевого слова AS) можно разделить на две части, которые объединены ключевым словом UNION. Первая часть (базис) — это запрос для поиска элемента, с которого следует начать рекурсивный запрос. Вторая часть (индукция) — то, что выполняется в каждой итерации.

Сначала выполняется база рекурсии, и ее результат помещается в промежуточное отношение t. Затем многократно выполняется индукционная часть, каждая итерация пополняет содержимое отношения t. Итерации выполняются до тех пор, пока меняется содержимое отношения t. После завершения всех шагов индукции будет выполнен запрос к построенному отношению: SELECT * FROM t. База индукции, то есть нерекурсивная часть, отделяется от индукционного шага с помощью UNION (количество и типы столбцов в обоих запросах должны совпадать). Также необходимо обеспечить возможность остановки рекурсивного запроса.

Пример рекурсивного запроса, который находит кратчайший путь между городами:

```
WITH RECURSIVE p(last_arrival, destination, point_arrival, found) AS (
  SELECT a_from.name,
        a_to.name,
        ARRAY[a_from.name],
        a_from.name = a_to.name
  FROM   point a_from, point a_to
  WHERE  a_from.name = 'A'
  AND    a_to.name = 'D'
  UNION ALL
  SELECT r.out,
```

```
        p.destination,
        (p.point_arrival || r.out),
        bool_or(r.out = p.destination) OVER ()
    FROM   way r, p
    WHERE  r.in = p.last_arrival
    AND    NOT r.out = ANY(p.point_arrival)
    AND    NOT p.found
)
SELECT point_arrival
FROM   p
WHERE  p.last_arrival = p.destination;
```

Конструкция LIMIT позволяет получить только часть строк от результата запроса:

```
CREATE OR REPLACE FUNCTION public.scal_fun(min_weight integer)
RETURNS integer AS
'SELECT max(age) FROM public."account" WHERE weight > min_weight LIMIT 3'
LANGUAGE SQL volatile
```

Выполнение функций row_number(), Rank(), dense_rank(), ntile(4) на таблице Account:

```
SELECT age, special, rank() OVER (order by age), row_number() OVER (order by age), dense_rank() OVER (order by age), ntile(4) OVER (order by age) FROM account;
```

Курсор – временный набор строк, которые можно перебирать последовательно, с первой до последней. При работе с курсорами используются следующие команды. Пример использование курсора, который считать средний возраст всех представленных вариантов особенностей:

```
CREATE OR REPLACE FUNCTION public.multi_acc_diets()
RETURNS TABLE (name_special varchar, avr real) AS $$
DECLARE cursor_enum CURSOR FOR SELECT age, special FROM account;
DECLARE cursor_enum_special CURSOR FOR SELECT DISTINCT special FROM account;
DECLARE count_value int;
DECLARE sum_value int;
BEGIN
count_value = 0;
```

```
sum_value = 0;
FOR enum_values_special IN cursor_enum_special LOOP
        FOR enum_values IN cursor_enum LOOP
                IF enum_values.special = enum_values_special.special THEN
                        sum_value := sum_value + enum_values.age;
                        count_value := count_value + 1;
                END IF;
        END LOOP;
        avr := sum_value / count_value;
        name_special := enum_values_special.special;
        count_value = 0;
        sum_value = 0;
        RETURN NEXT;
END LOOP;
END
$$
LANGUAGE 'plpgsql';
```

В функциях и процедурах можно использовать обращение к встроенным и системным функциям, например, current_query(), которая возвращает текст запроса:

```
CREATE OR REPLACE FUNCTION public.system_fun()
RETURNS text AS
'SELECT * FROM current_query()'
LANGUAGE SQL volatile;
```

Иногда бывает полезно получать данные из модифицируемых строк в процессе их обработки. Это возможно с использованием предложения RETURNING, которое можно задать для команд INSERT, UPDATE и DELETE:

```
UPDATE account SET age = 1 WHERE id = 100 RETURNING age;
```

Вывод

В результате выполнения работы были изучены постреляционные возможности языка SQL. Так же были освоить языки и технологии SQL\PSM на примере PostgreSQL. Приобретены навыки создания функций и процедур,

работы с исключениями и условным оператором.

Список используемой литературы

1. Виноградов В.И., Виноградова М.В. Постреляционные модели данных и языки запросов: Учебное пособие. — М.: Изд-во МГТУ им. Н.Э. Баумана, 2017. – 100с. - ISBN 978-5-7038-4283-6.
2. PostgreSQL 14.2 Documentation. – Текст. Изображение: электронные // PostgreSQL : [сайт]. – URL: https://www.postgresql.org/docs/14/index.html (дата обращения: 12.02.2024)
3. pgAdmin 4 6.5 documentation. – Текст. Изображение: электронные // pgAdmin - PostgreSQL Tools : [сайт]. – URL: https://www.pgadmin.org/docs/pgadmin4/6.5/index.html (дата обращения: 12.02.2024)
4. PostgreSQL : Документация: 14: 8.15. Массивы. – Текст. Изображение : электронные // Компания Postgres Professional : [сайт]. – URL: https://postgrespro.ru/docs/postgresql/14/arrays (дата обращения: 12.02.2024)
5. PostgreSQL : Документация: 14: 8.16. Составные типы. – Текст. Изображение : электронные // Компания Postgres Professional : [сайт]. – URL: https://postgrespro.ru/docs/postgresql/14/rowtypes (дата обращения: 12.02.2024)
6. PostgreSQL : Документация: 14: 8.7. Типы перечислений. – Текст. Изображение : электронные // Компания Postgres Professional : [сайт]. – URL: https://postgrespro.ru/docs/postgresql/14/datatype-enum (дата обращения: 12.02.2024)
7. PostgreSQL: Документация: 14: 5.10. Наследование. – Текст. Изображение : электронные // Компания Postgres Professional : [сайт]. – URL: https://postgrespro.ru/docs/postgresql/14/ddl-inherit (дата обращения: 12.02.2024)
8. PostgreSQL: Документация: 14: 38.13. Пользовательские типы. – Текст. Изображение : электронные // Компания Postgres Professional : [сайт]. –

URL: https://postgrespro.ru/docs/postgresql/14/xtypes (дата обращения: 12.02.2024)

9. PostgreSQL: Документация: 14: CREATE TYPE. – Текст. Изображение : электронные // Компания Postgres Professional : [сайт]. – URL: https://postgrespro.ru/docs/postgresql/14/sql-createtype (дата обращения: 12.02.2024)