ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ЦИФРОВЫХ ОБРАЗОВАТЕЛЬНЫХ ТЕХНОЛОГИЙ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ (ФИЛИАЛ) ДГТУ В Г. ТАГАНРОГЕ

ЦМК «ПРИКЛАДНАЯ ИНФОРМАТИКА»

ПРАКТИКУМ

по дисциплине

«Основы проектирования баз данных»

Авторы
Андриян И.В.,
Андриян О.В.

Ростов-на-Дону, 2024

Аннотация

Методические указания предназначены для студентов очной формы обучения по специальности 09.02.07 Информационные системы и программирование.

Авторы

Преподаватель ЦМК «Прикладная информатика»
Андриян Иван Васильевич

Преподаватель ЦМК «Прикладная информатика»
Андриян Оксана Вячеславовна

Введение

Данное учебно-методическое пособие предназначено для сопровождения практических занятий по курсу «Основы проектирования баз данных», содержит информацию, необходимую для успешного освоения практических навыков и закрепления теоретических знаний. В пособии описан процесс работы с инструментами, которые применяются на практических занятиях. Представлен ряд типичных задач, встречающихся при проектировании баз данных и подходы к их решению.

Цель настоящего пособия ориентирована на развитие практических навыков работы с базами данных, необходимых для успешного решения реальных задач, формирование у обучающихся глубокого понимания принципов проектирования и реализации баз данных. В результате освоения материала учебного пособия (дисциплины) обучающийся будет знать: основные понятия и принципы проектирования баз данных; различные модели данных и их характеристики; основные этапы жизненного цикла разработки баз данных; способы реализации безопасности данных в базах данных. Уметь: моделировать данные с использованием различных моделей; проектировать базы данных; интегрировать базы данных в единую систему; обеспечивать безопасность данных в базе данных.

Общие положения

Практические занятия выполняются каждым обучающимся самостоятельно в полном объеме и согласно содержанию методических указаний.

Перед выполнением обучающийся должен отчитаться преподавателю за выполнение предыдущего занятия (сдать отчет). Обучающийся должен на уровне понимания и воспроизведения предварительно усвоить необходимую для выполнения практических занятий теоретическую и информацию. Обучающийся, получивший положительную оценку и сдавший отчет по предыдущему практическому занятию, допускается к выполнению следующему занятию. Обучающийся, пропустивший практическое занятие по уважительной либо неуважительной причине, закрывает задолженность в процессе выполнения последующих практических занятий.

Форма отчета**:** титульный лист; введение (цель и задачи); результаты выполнения; заключение. Время работы: 2 часа.

Требования к оформлению отчетов

– расстояние от рамки формы до границ текста в начале и в конце строк – не менее 3 мм;
– расстояние от верхней и нижней строки текста до верхней и нижней рамки должно быть не менее 10 мм;
– гарнитура шрифта – Times New Roman;
– размер шрифта для основного текста – 14;

– междустрочный интервал – 1,5
– размер шрифта для примечаний, ссылок – 12;
– абзацный отступ –1,25 мм;
– выравнивание основного текста – по ширине страницы.
Перенос в словах допускается использовать, кроме заголовков.
Для заполнения ячеек основной надписи:
– гарнитура шрифта Arial;
– курсив;
– для обозначения работы размер – 20.

Практическая работа № 1. Создание базы данных

Цель работы: Ознакомиться с созданием баз данных, с типом баз данных, с понятием баз данных, внешними ключами. Сформировать пониманием и закрепить практическим заданием

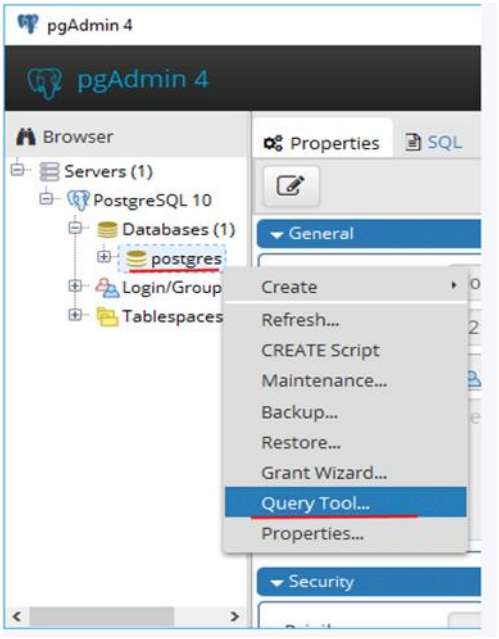
Теоретическая часть

Создание базы данных

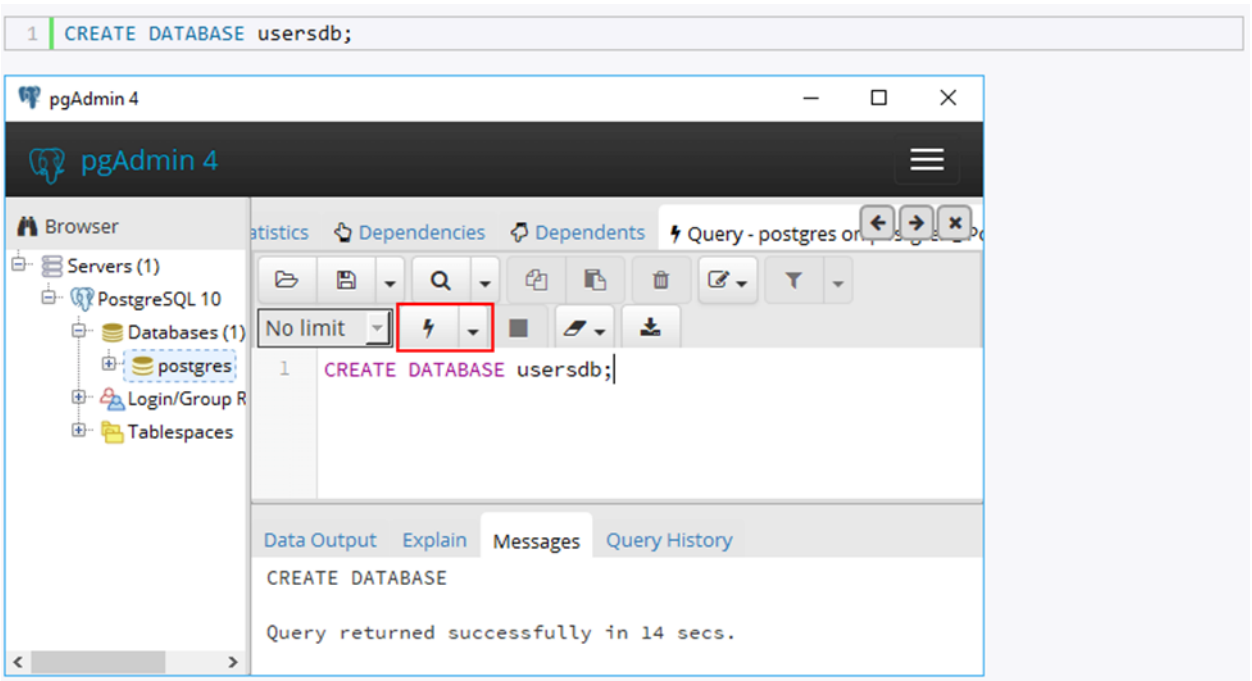
Для создания базы данных используется команда CREATE DATABASE, после которой указывается название базы данных.

Для выполнения запросов будем использовать графический клиент pgAdmin, хотя также можно использовать консольный клиент psql.

Чтобы создать новую базу, данных откроем pgAdmin. В левой части программы выберем какую-нибудь базу данных, например, стандартную бд postgres, и нажмем на нее правой кнопкой мыши.

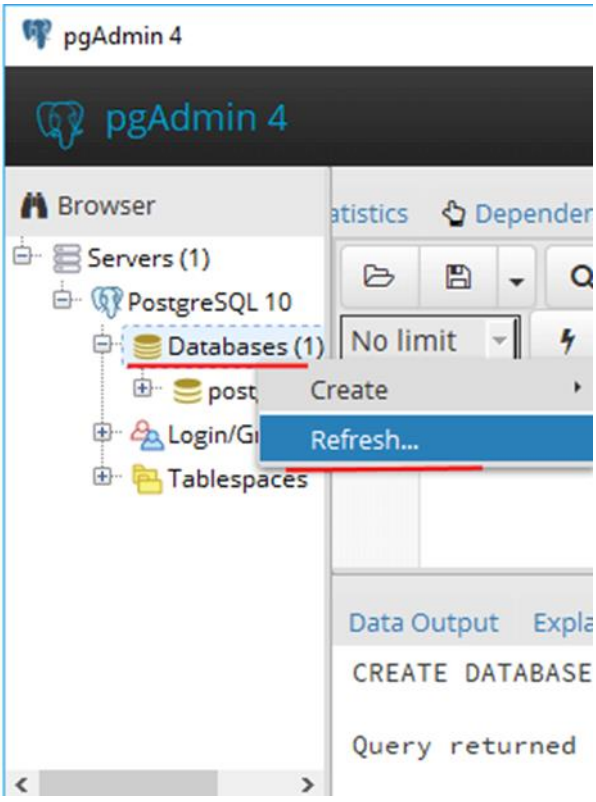


В появившемся меню выберем пункт Query Tool..., и в центральной части программы откроется поле для ввода кода SQL. В это поле введем следующий код:

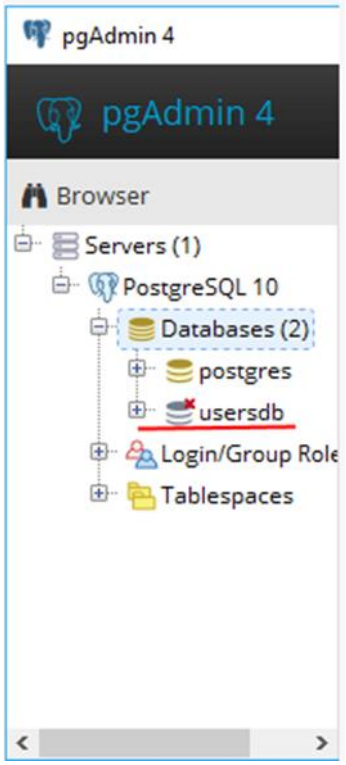


Для выполнения кода нажмем на значок молнии, и после этого будет создана база данных usersdb.

Чтобы увидеть нашу базу данных, нажмем в левой части на узел Databases правой кнопкой мыши и в контекстном меню выберем Refresh...:



Произойдет обновление, и мы увидим созданную базу данных.



По умолчанию база является неактивной, поэтому ее значок имеет серый цвет. Но чтобы к ней подключиться, достаточно нажать на нее и раскрыть ее узел.

Удаление базы данных

Для удаления базы данных применяется команда DROP DATABASE, после которой указывается название базы данных.

Удаляемая база данных должна быть неактивной, то есть подключение к ней должно быть закрыта.

Например, удаление базы данных usersdb:

```
DROP DATABASE usersdb;
```

Создание и удаление таблиц

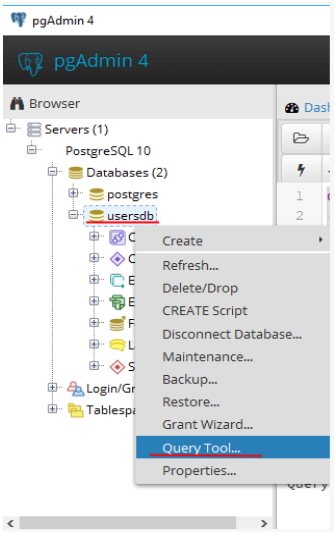
Для создания таблиц применяется команда CREATE TABLE, после которой указывается название таблицы. Также с этой командой можно использовать ряд операторов, которые определяют столбцы таблицы и их атрибуты. Общий синтаксис создания таблицы выглядит следующим образом:

```
CREATE TABLE название_таблицы
(название_столбца1 тип_данных атрибуты_столбца1,
 название_столбца2 тип_данных атрибуты_столбца2,
 .....................................................
 название_столбцаN тип_данных атрибуты_столбцаN,
 атрибуты_таблицы
);
```

После названия таблицы в скобках перечисляется спецификация для всех столбцов. Причем для каждого столбца надо указывается название и тип данных,

который он будет представлять. Тип данных определяет, какие данные (числа, строки и т.д.) может содержать столбец.

Например, создадим таблицу в базе данных через pgAdmin. Для этого вначале выберем в pgAdmin целевую базу данных, нажмем на нее правой кнопкой мыши и в контекстном меню выберем пункт Query Tool...:



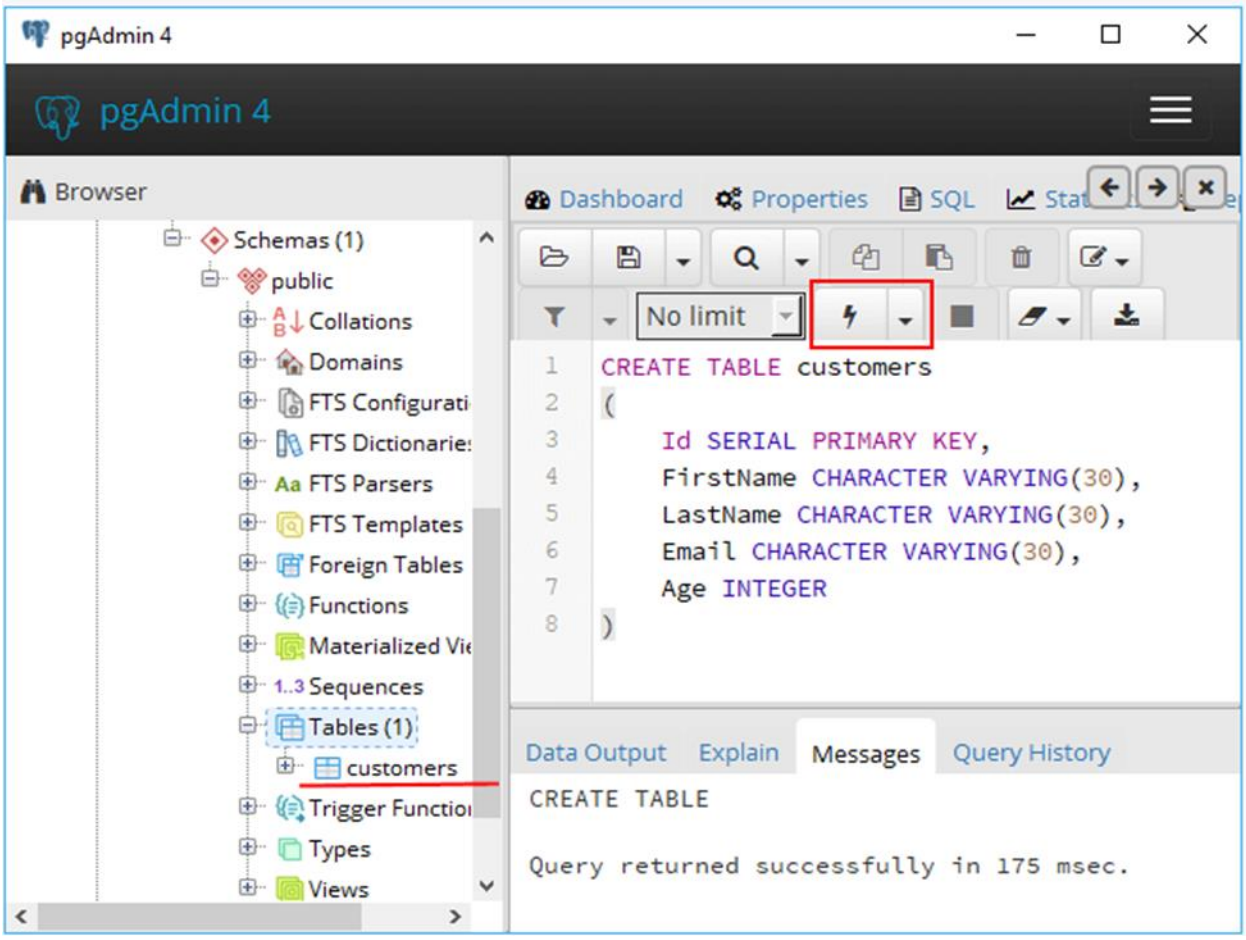
После этого откроется поле для ввода кода на SQL. Причем таблица будет создаваться именно для той базы данных, для которой мы открываем это поле для ввода SQL. Далее в открывшееся в центральной части программы поле введем следующий набор выражений:

```
CREATE TABLE customers
(
    Id SERIAL PRIMARY KEY,
    FirstName CHARACTER VARYING(30),
    LastName CHARACTER VARYING(30),
    Email CHARACTER VARYING(30),
    Age INTEGER
);
```

В данном случае в таблице Customers определяются пять столбцов: Id, FirstName, LastName, Age, Email. Первый столбец - Id представляет идентификатор клиента, он служит первичным ключом и поэтому имеет тип SERIAL. Фактически данный столбец будет хранить числовое значение 1, 2, 3 и т.д., которое для каждой новой строки будет автоматически увеличиваться на единицу.

Следующие три столбца представляют имя, фамилию клиента и его электронный адрес и имеют тип CHARACTER VARYING (30), то есть представляют строку длиной не более 30 символов.

Последний столбец - Age представляет возраст пользователя и имеет тип INTEGER, то есть хранит числа. И после выполнения этой команды в выбранную базу данных будет добавлена таблица customers.



Удаление таблиц

Для удаления таблиц используется команда DROP TABLE, которая имеет следующий синтаксис:

```
DROP TABLE table1 [, table2, ...];
```

Например, удаление таблицы customers:

```
DROP TABLE customers;
```

Типы данных в PostgreSQL

При определении таблицы для всех ее столбцов необходимо указать тип данных. Тип данных определяет диапазон значений, которые могут храниться в столбце, сколько они будут занимать места в памяти. PostgreSQL поддерживает богатую палитру различных типов данных, среди которые условно можно разделить на подгруппы: числовые, символьные, логические, дата и время, бинарные и ряд других.

Числовые типы данных

serial: представляет автоинкрементирующееся числовое значение, которое занимает 4 байта и может хранить числа от 1 до 2147483647. Значение дан-

ного типа образуется путем автоинкремента значения предыдущей строки. Поэтому, как правило, данный тип используется для определения идентификаторов строки.

smallserial: представляет автоинкрементирующееся числовое значение, которое занимает 2 байта и может хранить числа от 1 до 32767. Аналог типа serial для небольших чисел.

bigserial: представляет автоинкрементирующееся числовое значение, которое занимает 8 байт и может хранить числа от 1 до 9223372036854775807. Аналог типа serial для больших чисел.

smallint: хранит числа от -32768 до +32767. Занимает 2 байта. Имеет псевдоним int2.

integer: хранит числа от -2147483648 до +2147483647. Занимает 4 байта. Имеет псевдонимы int и int4.

bigint: хранит числа от -9223372036854775808 до +9223372036854775807. Занимает 8 байт. Имеет псевдоним int8.

numeric: хранит числа с фиксированной точностью, которые могут иметь до 131072 знаков в целой части и до 16383 знаков после запятой.

Данный тип может принимать два параметра precision и scale: numeric(precision, scale).

Параметр precision указывает на максимальное количество цифр, которые может хранить число.

Параметр scale представляет максимальное количество цифр, которые может содержать число после запятой. Это значение должно находиться в диапазоне от 0 до значения параметра precision. По умолчанию оно равно 0. Например, для числа 23.5141 precision равно 6, а scale - 4.

decimal: хранит числа с фиксированной точностью, которые могут иметь до 131072 знаков в целой части и до 16383 знаков в дробной части. То же самое, что и numeric.

real: хранит числа с плавающей точкой из диапазона от 1E-37 до 1E+37. Занимает 4 байта. Имеет псевдоним float4.

double precision: хранит числа с плавающей точкой из диапазона от 1E-307 до 1E+308. Занимает 8 байт. Имеет псевдоним float8.

Примеры использования:

```
1
2       Id SERIAL,
        TotalWeight NUMERIC(9,2),
3       Age INTEGER,
        Surplus REAL
4
```

Типы для работы с валютой (денежными единицами)

Для работы с денежными единицами определен тип money, который может принимать значения в диапазоне от -92233720368547758.08 до +92233720368547758.07 и занимает 8 байт.

Символьные типы

character(n): представляет строку из фиксированного количества символов. С помощью параметра задается задается количество символов в строке. Имеет псевдоним char(n).

character varying(n): представляет строку из переменной длины. С помощью параметра задается задается максимальное количество символов в строке. Имеет псевдоним varchar(n).

text: представляет текст произвольной длины.

Бинарные данные

Для хранения бинарных данных определен тип bytea. Он хранит данные в виде бинарных строк, которые представляют последовательность октетов или байт.

Типы для работы с датами и временем

timestamp: хранит дату и время. Занимает 8 байт. Для дат самое нижнее значение - 4713 г до н.э., самое верхнее значение - 294276 г н.э.

timestamp with time zone: то же самое, что и timestamp, только добавляет данные о часовом поясе.

date: представляет дату от 4713 г. до н.э. до 5874897 г н.э. Занимает 4 байта.

time: хранит время с точностью до 1 микросекунды без указания часового пояса. Принимает значения от 00:00:00 до 24:00:00. Занимает 8 байт.

time with time zone: хранит время с точностью до 1 микросекунды с указанием часового пояса. Принимает значения от 00:00:00+1459 до 24:00:00-1459. Занимает 12 байт.

interval: представляет временной интервал. Занимает 16 байт.

Распространенные форматы дат:

yyyy-mm-dd - 1999-01-08;

Month dd, yyyy - January 8, 1999;

mm/dd/yyyy - 1/8/1999.

Распространенные форматы времени:

hh:mi - 13:21;

hh:mi am/pm - 1:21 pm;

hh:mi:ss - 1:21:34.

Логический тип

Тип boolean может хранить одно из двух значений: true или false. Вместо true можно указывать следующие значения: TRUE, 't', 'true', 'y', 'yes', 'on', '1'.

Вместо false можно указывать следующие значения: FALSE, 'f', 'false', 'n', 'no', 'off', '0'.

Типы для представления интернет-адресов

cidr: интернет-адрес в формате IPv4 и IPv6. Например, 192.168.0.1. Занимает от 7 до 19 байт.

inet: интернет-адрес в формате cidr/y, где cidr это адрес в формате IPv4 или IPv6, а /y - количество бит в адресе (если этот параметр не указан, то используется 34 для IPv4, 128 для IPv6). Например, 192.168.0.1/24 или 2001:4f8:3:ba:2e0:81ff:fe22:d1f1/128. Занимает от 7 до 19 байт.

macaddr: хранит MAC-адрес. Занимает 6 байт.

macaddr8: хранит MAC-адрес в формате EUI-64. Занимает 8 байт.

Геометрические типы

point: представляет точку на плоскости в формате (x,y). Занимает 16 байт.

line: представляет линию неопределенной длины в формате {A,B,C}. Занимает 32 байта.

lseg: представляет отрезок в формате ((x1,y1),(x2,y2)). Занимает 32 байта.

box: представляет прямоугольник в формате ((x1,y1),(x2,y2)). Занимает 32 байта.

path: представляет набор содиненных точек. В формате ((x1,y1),...) путь является закрытым (первая и последняя точка соединяются линией) и фактически представляет многоугольник. В формате [(x1,y1),...] путь является открытым Занимает 16+16n байт.

polygon: представляет многоугольник в формате ((x1,y1),...). Занимает 40+16n байт.

circle: представляет окружность в формате <(x,y),r>. Занимает
24 байта.

Остальные типы данных

json: хранит данные json в текстовом виде.

jsonb: хранит данные json в бинарном формате.

uuid: хранит универсальный уникальный идентификатор (UUID), например, a0eebc99-9c0b-4ef8-bb6d-6bb9bd380a11. Занимает 32 байта.

xml: хранит даные в формате XML.

Ограничения столбцов и таблиц

При определении таблиц и их столбцов в SQL мы можем использовать ряд атрибутов, которые накладывают определенные ограничения. Рассмотрим эти атрибуты.

PRIMARY KEY

С помощью выражения PRIMARY KEY столбец можно сделать первичным ключом.

```
CREATE TABLE Customers
(
   Id SERIAL PRIMARY KEY,
   FirstName CHARACTER VARYING(30),
   LastName CHARACTER VARYING(30),
   Email CHARACTER VARYING(30),
   Age INTEGER
)
```

Первичный ключ уникально идентифицирует строку в таблице. В качестве первичного ключа необязательно должны выступать столбцы с типом SERIAL, они могут представлять любой другой тип. Установка первичного ключа на уровне таблицы:

```
CREATE TABLE Customers
(
   Id SERIAL,
   FirstName CHARACTER VARYING(30),
   LastName CHARACTER VARYING(30),
   Email CHARACTER VARYING(30),
   Age INTEGER,
   PRIMARY KEY(Id)
);
```

Первичный ключ может быть составным (compound key). Такой ключ может потребоваться, если у нас сразу два столбца должны уникально идентифицировать строку в таблице. Например:

```
CREATE TABLE OrderLines
(
   OrderId INTEGER,
   ProductId INTEGER,
   Quantity INTEGER,
   Price MONEY,
   PRIMARY KEY(OrderId, ProductId)
);
```

Здесь поля OrderId и ProductId вместе выступают как составной первичный ключ. То есть в таблице OrderLines не может быть двух строк, где для обоих из этих полей одновременно были бы одни и те же значения.

UNIQUE

Если мы хотим, чтобы столбец имел только уникальные значения, то для него можно определить атрибут UNIQUE.

```
CREATE TABLE Customers
(
   Id SERIAL PRIMARY KEY,
   FirstName CHARACTER VARYING(20),
   LastName CHARACTER VARYING(20),
   Email CHARACTER VARYING(30) UNIQUE,
   Phone CHARACTER VARYING(30) UNIQUE,
   Age INTEGER
);
```

В данном случае столбцы, которые представляют электронный адрес и телефон, будут иметь уникальные значения. И мы не сможем добавить в таблицу две строки, у которых значения для этих столбцов будет совпадать.

Также мы можем определить этот атрибут на уровне таблицы:

```
CREATE TABLE Customers
   Id SERIAL PRIMARY KEY,
   FirstName CHARACTER VARYING(20),
   LastName CHARACTER VARYING(20),
   Email CHARACTER VARYING(30),
   Phone CHARACTER VARYING(30),
   Age INTEGER,
   UNIQUE(Email, Phone)
```

Или так:

```
CREATE TABLE Customers
   Id SERIAL PRIMARY KEY,
   FirstName CHARACTER VARYING(20),
   LastName CHARACTER VARYING(20),
   Email CHARACTER VARYING(30),
   Phone CHARACTER VARYING(30),
   Age INTEGER,
   UNIQUE(Email),
   UNIQUE(Phone)
```

NULL и NOT NULL

Чтобы указать, может ли столбец принимать значение NULL, при определении столбца ему можно задать атрибут NULL или NOT NULL. Если этот атрибут явным образом не будет использован, то по умолчанию столбец будет допускать значение NULL. Исключением является тот случай, когда столбец выступает

в роли первичного ключа - в этом случае по умолчанию столбец имеет значение NOT NULL.

```
CREATE TABLE Customers
   Id SERIAL PRIMARY KEY,
   FirstName CHARACTER VARYING(20) NOT NULL,
   LastName CHARACTER VARYING(20) NOT NULL,
   Age INTEGER
```

DEFAULT

Атрибут DEFAULT определяет значение по умолчанию для столбца. Если при добавлении данных для столбца не будет предусмотрено значение, то для него будет использоваться значение по умолчанию.

```
CREATE TABLE Customers
Id SERIAL PRIMARY KEY,
   FirstName CHARACTER VARYING(20),
   LastName CHARACTER VARYING(20),
   Age INTEGER DEFAULT 18
```

Здесь для столбца Age предусмотрено значение по умолчанию 18.

CHECK

Ключевое слово CHECK задает ограничение для диапазона значений, которые могут храниться в столбце. Для этого после слова CHECK указывается в скобках условие, которому должен соответствовать столбец или несколько столбцов. Например, возраст клиентов не может быть меньше 0 или больше 100:

```
CREATE TABLE Customers
   Id SERIAL PRIMARY KEY,
   FirstName CHARACTER VARYING(20),
   LastName CHARACTER VARYING(20),
   Age INTEGER DEFAULT 18 CHECK(Age >0 AND Age < 100),
   Email CHARACTER VARYING(30) UNIQUE CHECK(Email !=''),
   Phone CHARACTER VARYING(20) UNIQUE CHECK(Phone !='')
```

Здесь также указывается, что столбцы Email и Phone не могут иметь пустую строку в качестве значения (пустая строка не эквивалентна значению NULL).

Для соединения условий используется ключевое слово AND. Условия можно задать в виде операций сравнения больше (>), меньше (<), не равно (!=).

Также с помощью CHECK можно создать ограничение в целом для таблицы:

```
CREATE TABLE Customers
   Id SERIAL PRIMARY KEY,
   Age INTEGER DEFAULT 18,
   FirstName CHARACTER VARYING(20),
   LastName CHARACTER VARYING(20),
   Email CHARACTER VARYING(30) UNIQUE,
```

```
    Phone CHARACTER VARYING(20) UNIQUE,
    CHECK((Age >0 AND Age<100) AND (Email !='') AND (Phone !='')
```

Оператор CONSTRAINT. Установка имени ограничений.

С помощью ключевого слова CONSTRAINT можно задать имя для ограничений. В качестве ограничений могут использоваться PRIMARY KEY, UNIQUE, CHECK.

Имена ограничений можно задать на уровне столбцов. Они указываются после CONSTRAINT перед атрибутами:

```
CREATE TABLE Customers
(
    Id SERIAL CONSTRAINT customer_Id PRIMARY KEY,
    Age INTEGER CONSTRAINT customers_age_check CHECK(Age >0 AND Age < 100),
    FirstName CHARACTER VARYING(20) NOT NULL,
    LastName CHARACTER VARYING(20) NOT NULL,
    Email CHARACTER VARYING(30) CONSTRAINT customers_email_key UNIQUE,
    Phone CHARACTER VARYING(20) CONSTRAINT customers_phone_key UNIQUE
```

В принципе необязательно задавать имена ограничений, при установке соответствующих атрибутов SQL Server автоматически определяет их имена. Но, зная имя ограничения, мы можем к нему обращаться, например, для его удаления. И также можно задать все имена ограничений через атрибуты таблицы

```
CREATE TABLE Customers
(
    Id SERIAL,
    Age INTEGER,
    FirstName CHARACTER VARYING(20) NOT NULL,
    LastName CHARACTER VARYING(20) NOT NULL,
    Email CHARACTER VARYING(30),
    Phone CHARACTER VARYING(20),
    CONSTRAINT customer_Id PRIMARY KEY(Id),
    CONSTRAINT customers_age_check CHECK(Age >0 AND Age < 100),
    CONSTRAINT customers_email_key UNIQUE(Email),
    CONSTRAINT customers_phone_key UNIQUE(Phone);
```

Вне зависимости от того, используется оператор CONSTRAINT для создания ограничений или нет (в этом случае при установке ограничений PostgreSQL сам дает им имена), мы можем просмотреть все ограничения в pgAdmin в узле базы данных в подузле:

Внешние ключи

Для связи между таблицами применяются внешние ключи. Внешний ключ устанавливается для столбца из зависимой, подчиненной таблицы (referencing table), и указывает на один из столбцов из главной таблицы (referenced table). Как правило, внешний ключ указывает на первичный ключ из связанной главной таблицы.

Общий синтаксис установки внешнего ключа на уровне столбца:

```
REFERENCES главная_таблица (столбец_главной_таблицы)
  [ON DELETE {CASCADE|RESTRICT}]
  [ON UPDATE {CASCADE|RESTRICT}]
```

Чтобы установить связь между таблицами, после ключевого слова REFERENCES указывается имя связанной таблицы и далее в скобках имя столбца из этой таблицы, на который будет указывать внешний ключ. После выражения REFERENCES может идти выражение ON DELETE и ON UPDATE, которые уточняют поведение при удалении или обновлении данных.

Общий синтаксис установки внешнего ключа на уровне таблицы:

```
FOREIGN KEY (стобец1, столбец2, ... столбецN)
  REFERENCES главная_таблица (столбец_главной_таблицы1, столбец_главной_таблицы2, ... столбец_главной_таблицыN)
  [ON DELETE {CASCADE|RESTRICT}]
  [ON UPDATE {CASCADE|RESTRICT}]
```

Например, определим две таблицы и свяжем их посредством внешнего ключа:

```
CREATE TABLE CUSTOMERS
(
ID SERIAL PRIMARY KEY,
AGE INTEGER,
FIRSTNAME VARCHAR(20) NOT NULL
);

CREATE TABLE ORDERS
(
ID SERIAL PRIMARY KEY,
CUSTOMERID INTEGER REFERENCES CUSTOMERS (ID),
QUANTITY INTEGER
);
```

Здесь определены таблицы Customers и Orders. Customers является главной и представляет клиента. Orders является зависимой и представляет заказ, сделанный клиентом. Эта таблица через столбец CustomerId связана с таблицей Customers и ее столбцом Id. То есть столбец CustomerId является внешним ключом, который указывает на столбец Id из таблицы Customers.

Определение внешнего ключа на уровне таблицы выглядело бы следующим образом:

```
CREATE TABLE CUSTOMERS
(
ID SERIAL PRIMARY KEY,
AGE INTEGER,
FIRSTNAME VARCHAR(20) NOT NULL
);

CREATE TABLE ORDERS
(
ID SERIAL PRIMARY KEY,
CUSTOMERID INTEGER,
QUANTITY INTEGER,
FOREIGN KEY (CUSTOMERID) REFERENCES CUSTOMERS (ID)
);
```

ON DELETE и ON UPDATE

С помощью выражений ON DELETE и ON UPDATE можно установить действия, которые выполняются соответственно при удалении и изменении связанной строки из главной таблицы. Для установки подобного действия можно использовать следующие опции:

- CASCADE: автоматически удаляет или изменяет строки из зависимой таблицы при удалении или изменении связанных строк в главной таблице.

- RESTRICT: предотвращает какие-либо действия в зависимой таблице при удалении или изменении связанных строк в главной таблице. То есть фактически какие-либо действия отсутствуют.
- NO ACTION: действие по умолчанию, предотвращает какие-либо действия в зависимой таблице при удалении или изменении связанных строк в главной таблице. И генерирует ошибку. В отличие от RESTRICT выполняет отложенную проверку на связанность между таблицами.
- SET NULL: при удалении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение NULL.
- SET DEFAULT: при удалении связанной строки из главной таблицы устанавливает для столбца внешнего ключа значение по умолчанию, которое задается с помощью атрибуты DEFAULT. Если для столбца не задано значение по умолчанию, то в качестве него применяется значение NULL.

Каскадное удаление

По умолчанию, если на строку из главной таблицы по внешнему ключу ссылается какая-либо строка из зависимой таблицы, то мы не сможем удалить эту строку из главной таблицы. Вначале нам необходимо будет удалить все связанные строки из зависимой таблицы. И если при удалении строки из главной таблицы необходимо, чтобы были удалены все связанные строки из зависимой таблицы, то применяется каскадное удаление, то есть опция CASCADE:

```
CREATE TABLE ORDERS
(
ID SERIAL PRIMARY KEY,
CUSTOMERID INTEGER,
QUANTITY INTEGER,
FOREIGN KEY (CUSTOMERID) REFERENCES CUSTOMERS (ID) ON DELETE
CASCADE
);
```

Аналогично работает выражение ON UPDATE CASCADE. При изменении значения первичного ключа автоматически изменится значение связанного с ним внешнего ключа. Но так как первичные ключи, как правило, изменяются очень редко, да и в принципе не рекомендуется использовать в качестве первичных ключей столбцы с изменяемыми значениями, то на практике выражение ON UPDATE используется редко.

Установка NULL

При установке для внешнего ключа опции SET NULL необходимо, чтобы столбец внешнего ключа допускал значение NULL:

```
CREATE TABLE ORDERS
(
```

```
    ID SERIAL PRIMARY KEY,
    CUSTOMERID INTEGER,
    QUANTITY INTEGER,
FOREIGN KEY (CUSTOMERID) REFERENCES CUSTOMERS (ID) ON DELETE SET
NULL
);
```

Установка значения по умолчанию

```
CREATE TABLE ORDERS
(
ID SERIAL PRIMARY KEY,
CUSTOMERID INTEGER DEFAULT 1,
QUANTITY INTEGER,
FOREIGN KEY (CUSTOMERID) REFERENCES CUSTOMERS (ID) ON DELETE SET
DEFAULT
);
```

Если для столбца значение по умолчанию не задано через параметр DEFAULT, то в качестве такового используется значение NULL (если столбец допускает NULL).

Практическая часть

Рабочее задание: Создание и удаление баз данных и таблиц в PostgreSQL

– Создание базы данных:

1. Откройте pgAdmin: В левой части выберите базу данных (например, "postgres").

2. Правой кнопкой мыши по базе данных: Выберите "Query Tool...".

3. Введите код: `CREATE DATABASE usersdb;`

4. Выполните код: Нажмите на значок молнии.

5. Обновите список баз данных: В левой части, правой кнопкой мыши по "Databases", выберите "Refresh..."

– Удаление базы данных:

1. Убедитесь, что база данных неактивна: Закройте все подключения к ней.

2. В pgAdmin: Выберите "Query Tool..." для целевой базы данных (например, "postgres").

3. Введите код: `DROP DATABASE usersdb;`

4. Выполните код: Нажмите на значок молнии.

– Создание таблицы:

1. В pgAdmin: Выберите "Query Tool..." для целевой базы данных (например, "usersdb").

2. Введите код:

```sql
CREATE TABLE Customers (
    Id SERIAL PRIMARY KEY,
    FirstName VARCHAR(30),
    LastName VARCHAR(30),
    Age INTEGER,
    Email VARCHAR(50)
);
```

3. Выполните код: Нажмите на значок молнии.

– Удаление таблицы:

1. В pgAdmin: Выберите "Query Tool..." для целевой базы данных (например, "usersdb").

2. Введите код: `DROP TABLE Customers;`

3. Выполните код: Нажмите на значок молнии.

– Объяснение синтаксиса:

`CREATE DATABASE имя_базы_данных;` - создает новую базу данных.

`DROP DATABASE имя_базы_данных; - удаляет существующую базу данных.

`CREATE TABLE имя_таблицы ( столбец1 тип1, столбец2 тип2, ... );` - создает таблицу с указанными столбцами и их типами данных.

`DROP TABLE имя_таблицы;` - удаляет существующую таблицу.

Дополнительная информация:

`SERIAL PRIMARY KEY` - автоматически генерирует уникальный идентификатор (ключ) для каждой строки.

`VARCHAR(n)` - тип данных для хранения строк с максимальной длиной `n` символов.

`INTEGER` - тип данных для хранения целых чисел.

Важно:

Перед удалением базы данных убедитесь, что она неактивна и не используется. Используйте правильные имена баз данных и таблиц. При работе с базами данных всегда делайте резервные копии, чтобы избежать потери данных.

Практическая работа № 2. Ознакомление с основами работы в среде PostgreSQL

Цель работы: Изучить базовые операции по работе с базой данных. Изучить синтаксис команд. Приобрести навыки создания баз данных, создания, заполнения и модификации таблиц в PostgreSQL

Теоретическая часть

Коды команд

Создание базы данных происходит с помощью запроса CREATE DATABASE, подключение к базе данных происходит с помощью команды \c.

```
postgres=# create database var4;
CREATE DATABASE
postgres=# \c var4;
Вы подключены к базе данных "var4" как пользователь "postgres".
```

Создадим таблицы genres, books, publishers, cities, согласно варианту, через запрос CREATE TABLE.

```
var4=# CREATE TABLE genres(
var4(# id integer,
var4(# genre_name char(20)
var4(# );
CREATE TABLE
var4=# CREATE TABLE books(
var4(# id integer,
var4(# book_name text,
var4(# book_author char(20),
var4(# book_amount integer,
var4(# genre_id integer,
var4(# publisher_id integer,
var4(# city_id integer
var4(# );
CREATE TABLE
var4=# CREATE TABLE publishers(
var4(# id integer,
var4(# publisher_name char(20),
var4(# publisher_address text
var4(# );
CREATE TABLE
var4=# CREATE TABLE cities(
var4(# city_name char(30)
var4(# );
CREATE TABLE
```

Выведем созданные таблицы запросом SELECT * FROM

```
var4=# SELECT * FROM genres;
 id | genre_name
----+------------
(0 строк)


var4=# SELECT * FROM books;
 id | book_name | book_author | book_amount | genre_id | publisher_id | city_id
----+-----------+-------------+-------------+----------+--------------+---------
(0 строк)


var4=# SELECT * FROM publishers;
 id | publisher_name | publisher_address
----+----------------+-------------------
(0 строк)


var4=# SELECT * FROM cities;
 city_name
-----------
(0 строк)
```

Заполним таблицу cities данными, с помощью запроса INSERT INTO.

```
var4=# insert into cities values ('Москва');
INSERT 0 1
var4=# insert into cities values ('Санкт-Петербург');
INSERT 0 1
var4=# insert into cities values ('Новосибирск');
INSERT 0 1
var4=# insert into cities values ('Екатеринбург');
INSERT 0 1
var4=# insert into cities values ('Казань');
INSERT 0 1
var4=# insert into cities values ('Нижний Новгород');
INSERT 0 1
var4=# insert into cities values ('Челябинск');
INSERT 0 1
var4=# insert into cities values ('Красноярск');
INSERT 0 1
var4=# insert into cities values ('Самара');
INSERT 0 1
var4=# insert into cities values ('Уфа');
INSERT 0 1
var4=# SELECT * FROM cities;
          city_name
-------------------------------
 Москва
 Санкт-Петербург
 Новосибирск
 Екатеринбург
 Казань
 Нижний Новгород
 Челябинск
 Красноярск
 Самара
 Уфа
(10 строк)
```

Далее добавим новый столбец (id) в таблицу cities через запрос ALTER TABLE.

```
var4=# ALTER TABLE cities ADD id integer;
ALTER TABLE
var4=# SELECT * FROM cities;
          city_name            | id
-------------------------------+----
 Москва                        |
 Санкт-Петербург               |
 Новосибирск                   |
 Екатеринбург                  |
 Казань                        |
 Нижний Новгород               |
 Челябинск                     |
 Красноярск                    |
 Самара                        |
 Уфа                           |
(10 строк)
```

Запросом UPDATE обновим текущие строки, добавив при этом значение идентификатора. Также заполним остальные таблицы данными.

```
var4=# UPDATE cities SET id=1 WHERE city_name='Москва';
UPDATE 1
var4=# UPDATE cities SET id=2 WHERE city_name='Санкт-Петербург';
UPDATE 1
var4=# UPDATE cities SET id=3 WHERE city_name='Новосибирск';
UPDATE 1
var4=# UPDATE cities SET id=4 WHERE city_name='Екатеринбург';
UPDATE 1
var4=# UPDATE cities SET id=5 WHERE city_name='Казань';
UPDATE 1
var4=# UPDATE cities SET id=6 WHERE city_name='Нижний Новгород';
UPDATE 1
var4=# UPDATE cities SET id=7 WHERE city_name='Челябинск';
UPDATE 1
var4=# UPDATE cities SET id=8 WHERE city_name='Красноярск';
UPDATE 1
var4=# UPDATE cities SET id=9 WHERE city_name='Самара';
UPDATE 1
var4=# UPDATE cities SET id=10 WHERE city_name='Уфа';
UPDATE 1
var4=# SELECT * FROM cities;
          city_name            | id
-------------------------------+----
 Москва                        |  1
 Санкт-Петербург               |  2
 Новосибирск                   |  3
 Екатеринбург                  |  4
 Казань                        |  5
 Нижний Новгород               |  6
 Челябинск                     |  7
 Красноярск                    |  8
 Самара                        |  9
 Уфа                           | 10
(10 строк)
```

```
var4=# insert into genres values (1, 'Басня');
INSERT 0 1
var4=# insert into genres values (2, 'Повесть');
INSERT 0 1
var4=# insert into genres values (3, 'Сказка');
INSERT 0 1
var4=# insert into genres values (4, 'Роман');
INSERT 0 1
var4=# insert into genres values (5, 'Стихотворение');
INSERT 0 1
var4=# insert into genres values (6, 'Ода');
INSERT 0 1
var4=# insert into genres values (7, 'Комедия');
INSERT 0 1
var4=# insert into genres values (8, 'Новелла');
INSERT 0 1
var4=# insert into genres values (9, 'Поэма');
INSERT 0 1
var4=# insert into genres values (10, 'Трагедия');
INSERT 0 1
var4=# SELECT * FROM genres;
 id |      genre_name
----+----------------------
  1 | Басня
  2 | Повесть
  3 | Сказка
  4 | Роман
  5 | Стихотворение
  6 | Ода
  7 | Комедия
  8 | Новелла
  9 | Поэма
 10 | Трагедия
(10 строк)
```

```
var4=# INSERT INTO publishers VALUES (1, 'Эксмо', 'г. Москва, ул. Зорге');
INSERT 0 1
var4=# INSERT INTO publishers VALUES (2, 'АСТ', 'г. Москва, Пресненская наб.');
INSERT 0 1
var4=# INSERT INTO publishers VALUES (3, 'Триумф', 'г. Москва, ул. Михалковская');
INSERT 0 1
var4=# INSERT INTO publishers VALUES (4, 'РИПОЛ классик', 'г. Москва, Нижегородская ул.');
INSERT 0 1
var4=# INSERT INTO publishers VALUES (5, 'Росмэн', 'г. Москва, Ленинградский проспект');
INSERT 0 1
var4=# INSERT INTO publishers VALUES (6, 'Феникс', 'г. Ростов-на-Дону, Ростовская обл., ул. Варфоломеева');
INSERT 0 1
var4=# INSERT INTO publishers VALUES (7, 'Центрполиграф', 'г. Москва, 1-я ул.Энтузиастов');
INSERT 0 1
var4=# INSERT INTO publishers VALUES (8, 'Проспект', 'г. Москва, ул.Мосфильмовская');
INSERT 0 1
var4=# INSERT INTO publishers VALUES (9, 'Текст', 'г. Москва, ул. Усиевича');
INSERT 0 1
var4=# INSERT INTO publishers VALUES (10, 'Питер', 'г. Санкт-Петербург, Б. Сампсониевский пр.');
INSERT 0 1
```

```
var4=# SELECT * FROM publishers;
 id |     publisher_name     |                  publisher_address
----+------------------------+-----------------------------------------------------
  1 | Эксмо                  | г. Москва, ул. Зорге
  2 | АСТ                    | г. Москва, Пресненская наб.
  3 | Триумф                 | г. Москва, ул. Михалковская
  4 | РИПОЛ классик          | г. Москва, Нижегородская ул.
  5 | Росмэн                 | г. Москва, Ленинградский проспект
  6 | Феникс                 | г. Ростов-на-Дону, Ростовская обл., ул. Варфоломеева
  7 | Центрполиграф          | г. Москва, 1-я ул.Энтузиастов
  8 | Проспект               | г. Москва, ул.Мосфильмовская
  9 | Текст                  | г. Москва, ул. Усиевича
 10 | Питер                  | г. Санкт-Петербург, Б. Сампсониевский пр.
(10 строк)
```

```
var4=# INSERT INTO books VALUES (1, 'Война и мир', 'Лев Толстой', 27000, 10, 10, 1);
INSERT 0 1
var4=# INSERT INTO books VALUES (2, '1984', 'Джордж Оруэлл', 14000, 10, 7, 4);
INSERT 0 1
var4=# INSERT INTO books VALUES (3, 'Улисс', 'Джеймс Джойс', 6000, 2, 2, 4);
INSERT 0 1
var4=# INSERT INTO books VALUES (4, 'Звук и ярость', 'Уильям Фолкнер', 13000, 1, 7, 1);
INSERT 0 1
var4=# INSERT INTO books VALUES (5, 'Человек-невидимка', 'Ральф Эллисон', 23000, 1, 10, 8);
INSERT 0 1
var4=# INSERT INTO books VALUES (6, 'Гордость и предубеждение', 'Джейн Остен', 18000, 5, 9, 9);
INSERT 0 1
var4=# INSERT INTO books VALUES (7, 'Божественная комедия', 'Данте Алигьери', 15000, 7, 5, 3);
INSERT 0 1
var4=# INSERT INTO books VALUES (8, 'Кентерберийские рассказы', 'Джеффри Чосер', 6000, 8, 6, 10);
INSERT 0 1
var4=# INSERT INTO books VALUES (9, 'Путешествия Гулливера', 'Джонатан Свифт', 16000, 1, 7, 5);
INSERT 0 1
var4=# INSERT INTO books VALUES (10, 'Распад', 'Чинуа Ачебе', 48000, 9, 1, 9);
INSERT 0 1
var4=# SELECT * FROM books;
 id |        book_name         |     book_author      | book_amount | genre_id | publisher_id | city_id
----+--------------------------+----------------------+-------------+----------+--------------+---------
  1 | Война и мир              | Лев Толстой          |       27000 |       10 |           10 |       1
  2 | 1984                     | Джордж Оруэлл        |       14000 |       10 |            7 |       4
  3 | Улисс                    | Джеймс Джойс         |        6000 |        2 |            2 |       4
  4 | Звук и ярость            | Уильям Фолкнер       |       13000 |        1 |            7 |       1
  5 | Человек-невидимка        | Ральф Эллисон        |       23000 |        1 |           10 |       8
  6 | Гордость и предубеждение | Джейн Остен          |       18000 |        5 |            9 |       9
  7 | Божественная комедия     | Данте Алигьери       |       15000 |        7 |            5 |       3
  8 | Кентерберийские рассказы | Джеффри Чосер        |        6000 |        8 |            6 |      10
  9 | Путешествия Гулливера    | Джонатан Свифт       |       16000 |        1 |            7 |       5
 10 | Распад                   | Чинуа Ачебе          |       48000 |        9 |            1 |       9
(10 строк)
```

Создадим новую таблицу, в которой будут соединены данные из разных таблиц. Сделать это можно с помощью ключевого слова INNER JOIN. Соединим столбец book_name из таблицы books со столбцом genre_name из таблицы genres. Соединение будет происходить по идентификатору.

```
var4=# CREATE TABLE books_and_genres AS
var4-# SELECT books.book_name, genres.genre_name
var4-# FROM books
var4-# INNER JOIN genres
var4-# ON books.genre_id=genres.id;
SELECT 10
var4=# SELECT * FROM books_and_genres;
        book_name          |      genre_name
---------------------------+---------------------
 Путешествия Гулливера     | Басня
 Человек-невидимка         | Басня
 Звук и ярость             | Басня
 Улисс                     | Повесть
 Гордость и предубеждение  | Стихотворение
 Божественная комедия      | Комедия
 Кентерберийские рассказы  | Новелла
 Распад                    | Поэма
 1984                      | Трагедия
 Война и мир               | Трагедия
(10 строк)
```

Реляционная база данных (РБД): Организация данных в виде таблиц, где каждая строка представляет собой запись, а каждый столбец - атрибут.

PostgreSQL: Системы управления базами данных (СУБД) с открытым исходным кодом, известная своей надежностью, безопасностью и функциональностью.

Таблица: Структурированный набор данных, состоящий из строк и столбцов.

Справочники (lookup tables): Таблицы, которые содержат фиксированный набор значений для определенного атрибута (например, список жанров).

Связи между таблицами:

- Один к одному (1:1): Каждая запись в одной таблице соответствует одной записи в другой таблице;
- Один ко многим (1:N): Одна запись в одной таблице может соответствовать многим записям в другой таблице;
- Многие ко многим (M:N): Множество записей в одной таблице может соответствовать множеству записей в другой таблице.

Структура базы данных.

Наша база данных будет состоять из следующих таблиц:

Жанры (genres):

– genre_id` (INTEGER, PRIMARY KEY): Уникальный идентификатор жанра.

– genre_name` (VARCHAR): Название жанра.

– (Дополнительные поля при необходимости).

– **Книги (books):**

– `book_id` (INTEGER, PRIMARY KEY): Уникальный идентификатор книги.

– book_title` (VARCHAR): Название книги.

– `book_author` (VARCHAR): Автор книги.

– book_edition` (INTEGER): Тираж.
– genre_id` (INTEGER, FOREIGN KEY REFERENCES genres(genre_id)): Ссылка на таблицу "Жанры".
– publisher_id` (INTEGER, FOREIGN KEY REFERENCES publishers(publisher_id)): Ссылка на таблицу "Издательства".
– author_city_id` (INTEGER, FOREIGN KEY REFERENCES cities(city_id)): Ссылка на таблицу "Города".
(Дополнительные поля при необходимости).
Издательства (publishers):

- `publisher_id` (INTEGER, PRIMARY KEY): Уникальный идентификатор издательства.
- `publisher_name` (VARCHAR): Название издательства.
- `publisher_address` (VARCHAR): Адрес издательства.
- (Дополнительные поля при необходимости).

Города (cities):

- `city_id` (INTEGER, PRIMARY KEY): Уникальный идентификатор города.
- `city_name` (VARCHAR): Название города.
- (Дополнительные поля при необходимости).

Дополнительная таблица:

Книги_с_автором_и_жанром (books_with_author_genre):

- `book_title` (VARCHAR): Название книги.
- `book_author` (VARCHAR): Автор книги.
- `author_city` (VARCHAR): Город проживания автора.
- `genre` (VARCHAR): Жанр книги.
- `publisher` (VARCHAR): Издательство книги.

Практическая часть

Целью данной практической работы является**:**

- Построение базы данных для книжной лавки с использованием PostgreSQL;
- Определение связей между таблицами (1:N, M:N);
- Использование справочников для хранения стандартных значений;
- Создание таблицы, содержащей агрегированную информацию из других таблиц;
- Ознакомление с использованием команд интерактивного терминала psql для: просмотра структуры базы данных и таблиц; просмотра данных в таблицах; изменения структуры таблиц (добавление столбцов); добавления данных в таблицы; создание столбцов с пользовательскими типами данных.

Данная практическая работа позволит вам:

- Получить практический опыт создания и управления базами данных с использованием PostgreSQL;
- Углубить понимание концепций реляционных баз данных; связей между таблицами и справочников;
- Развить навыки работы с интерактивным терминалом psql;
- Улучшить способность структурировать и представлять данные в виде таблиц.

Рабочее задание: Создать и заполнить базу данных своего варианта в PostgreSQL. Таблицы (минимум по 10 записей в каждой) связать между собой полями идентификаторов. Предусмотреть наличие связей типа: «один ко многим». С помощью команд интерактивного терминала psql просмотреть структуру базы данных, структуру таблиц, просмотреть данные в них, изменить структуру таблиц, добавить столбцы, добавить данные, создать столбцы с пользовательскими типами данных. Предусмотреть наличие таблиц-справочников и таблиц, использующих справочники.

Создать и заполнить базу данных для обработки данных по работе книжной лавки, состоящую из четырех таблиц. Первая таблица должна содержать поля: идентификатор жанра, наименование жанра и другие поля при необходимости. Вторая: идентификатор книги, название книги, автор книги, тираж, идентификатор жанра книги, идентификатор издательства, идентификатор города проживания автора и другие поля при необходимости. Третья: идентификатор издательства, наименование издателя, адрес издательства и другие поля при необходимости. Четвертая: справочник городов. На основании созданных таблиц создать таблицу, содержащую, например, поля: название книги, автор книги, город проживания автора, жанр, издательство.

Практическая работа № 3. Транзакции и ограничения в среде PostgreSQL

Цель: Приобрести навыки создания пользователей баз данных, работы с транзакциями в среде PostgreSQL. Изучить основные команды по работе с базой данных. Получение навыков работы с ограничениями.

Теоретическая часть

Новый пользователь создается с помощью запроса CREATE USER (рис. 1).

```
postgres=# CREATE USER user1 PASSWORD 'asdf';
CREATE ROLE
postgres=#
```

Рисунок 1 – Создание пользователя user1 с паролем.

В нашем случае был создан простой пользователь с паролем и без каких-либо прав. На данный момент в СУБД существуют два пользователя: postgres и user1. Чтобы выдать новому пользователю права на редактирование таблицы используется запрос GRANT.

```
var4=# GRANT ALL ON books TO user1;
GRANT
var4=# GRANT ALL ON books_and_genres TO user1;
GRANT
var4=# GRANT ALL ON cities TO user1;
GRANT
var4=# GRANT ALL ON genres TO user1;
GRANT
var4=# GRANT ALL ON publishers TO user1;
GRANT
```

Рисунок 2 – Выдача прав пользователю user1 на редактирование таблиц в нашей БД.

При входе в СУБД под новым пользователем в приглашении psql будет отображаться символы “=>”, поскольку созданный пользователь не наделен правами суперпользователя (рис. 3).

```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]: user1
Пароль пользователя user1:
psql (15.4)
ПРЕДУПРЕЖДЕНИЕ: Кодовая страница консоли (866) отличается от основной
                страницы Windows (1251).
                8-битовые (русские) символы могут отображаться некорректно.
                Подробнее об этом смотрите документацию psql, раздел
                "Notes for Windows users".
Введите "help", чтобы получить справку.

postgres=>
```

Рисунок 3 – Вход в СУБД под новым пользователем.

Добавим ограничения в таблицу books, следуя нашему варианту. Для существующих столбцов запрос с ограничением будет выглядеть так:

ALTER TABLE <имя_таблицы> ADD CONSTRAINT <имя_ограничения> CHECK (условие);

На рис. 4-5 добавляются ограничения существующим столбцам согласно запросу выше.

```
var4=# ALTER TABLE books ADD CONSTRAINT amount_100max CHECK (book_amount < 100);
ALTER TABLE
var4=#
```

Рисунок 4 – Добавление ограничения столбцу book_amount.

```
var4=# ALTER TABLE books ADD CONSTRAINT genre_constraint CHECK (genre_id=1 OR genre_id=3 OR genre_id=5 OR genre_id=6 OR genre_id=7);
ALTER TABLE
var4=#
```

Рисунок 5 – Добавление ограничения столбцу genre_id.

Добавление столбцов с ограничениями происходит по следующему запросу (рис. 6):

ALTER TABLE <имя_таблицы> ADD <имя_нового_столбца> <тип_значений>

CONSTRAINT <имя_ограничения> CHECK (условие);

```
var4=# ALTER TABLE books ADD price integer CONSTRAINT price_50min CHECK (price >= 50);
ALTER TABLE
var4=# ALTER TABLE books ADD year integer CONSTRAINT year_2023max CHECK (year <= 2023);
ALTER TABLE
var4=#
```

Рисунок 6 – Добавление столбцов price и year с ограничениями.

Структура обновленной таблицы books с новыми столбцами и ограничениями приведена на рис. 7.

```
var4=# \d books
                          Таблица "public.books"
   Столбец    |       Тип       | Правило сортировки | Допустимость NULL | По умолчанию
--------------+-----------------+--------------------+-------------------+--------------
 id           | integer         |                    |                   |
 book_name    | text            |                    |                   |
 book_author  | character(20)   |                    |                   |
 book_amount  | integer         |                    |                   |
 genre_id     | integer         |                    |                   |
 publisher_id | integer         |                    |                   |
 city_id      | integer         |                    |                   |
 price        | integer         |                    |                   |
 year         | integer         |                    |                   |
Ограничения-проверки:
    "amount_100max" CHECK (book_amount < 100)
    "genre_constraint" CHECK (genre_id = 1 OR genre_id = 3 OR genre_id = 5 OR genre_id = 6 OR genre_id = 7)
    "price_50min" CHECK (price >= 50)
    "year_2023max" CHECK (year <= 2023)
```

Рисунок 7 – Просмотр структуры таблицы и ее ограничений.

При попытке добавить новые значения в таблицу, которые не будут удовлетворять одним или нескольким условиям, psql выведет ошибку (рис. 8).

```
var4=# INSERT INTO books VALUES (11, 'Безымянная книга', 'Неизвестный автор', 120, 8, 3, 6, 35, 2025);
ОШИБКА:  новая строка в отношении "books" нарушает ограничение-проверку "amount_100max"
ПОДРОБНОСТИ:  Ошибочная строка содержит (11, Безымянная книга, Неизвестный автор   , 120, 8, 3, 6, 35, 2025).
var4=# INSERT INTO books VALUES (11, 'Безымянная книга', 'Неизвестный автор', 88, 8, 3, 6, 35, 2025);
ОШИБКА:  новая строка в отношении "books" нарушает ограничение-проверку "genre_constraint"
ПОДРОБНОСТИ:  Ошибочная строка содержит (11, Безымянная книга, Неизвестный автор   , 88, 8, 3, 6, 35, 2025).
var4=# INSERT INTO books VALUES (11, 'Безымянная книга', 'Неизвестный автор', 88, 7, 3, 6, 35, 2025);
ОШИБКА:  новая строка в отношении "books" нарушает ограничение-проверку "price_50min"
ПОДРОБНОСТИ:  Ошибочная строка содержит (11, Безымянная книга, Неизвестный автор   , 88, 7, 3, 6, 35, 2025).
var4=# INSERT INTO books VALUES (11, 'Безымянная книга', 'Неизвестный автор', 88, 7, 3, 6, 350, 2025);
ОШИБКА:  новая строка в отношении "books" нарушает ограничение-проверку "year_2023max"
ПОДРОБНОСТИ:  Ошибочная строка содержит (11, Безымянная книга, Неизвестный автор   , 88, 7, 3, 6, 350, 2025).
var4=# INSERT INTO books VALUES (11, 'Безымянная книга', 'Неизвестный автор', 88, 7, 3, 6, 350, 1998);
INSERT 0 1
```

Рисунок 8 – Попытки добавить новые значения, не удовлетворяющим условиям.

Следующие запросы использовались для заполнения столбцов price и year случайными значениями (рис. 9):

UPDATE books SET price = random()*150+50;

UPDATE books SET year = random()*73+1950;

id	book_name	book_author	book_amount	genre_id	publisher_id	city_id	price	year
5	Человек-невидимка	Ральф Эллисон	23	1	10	8	191	2013
6	Гордость и предубеждение	Джейн Остен	18	5	9	9	132	1981
9	Путешествия Гулливера	Джонатан Свифт	16	1	7	5	115	2023
4	Звук и ярость	Уильям Фолкнер	95	1	7	1	61	1964
7	Божественная комедия	Данте Алигьери	95	7	5	3	81	1994
1	Война и мир	Лев Толстой	27	3	10	1	139	2020
2	1984	Джордж Оруэлл	95	3	7	4	99	2006
3	Улисс	Джеймс Джойс	60	5	2	4	78	1956
8	Кентерберийские рассказы	Джеффри Чосер	60	6	6	10	167	1960
10	Распад	Чинуа Ачебе	48	6	1	9	121	2019
11	Безымянная книга	Неизвестный автор	88	7	3	6	73	2004

(11 строк)

Рисунок 9 – Таблица books.

Выполнение транзакции начинается с запроса BEGIN, завершение транзакции выполняется с помощью COMMIT, а отмена транзации – ROLLBACK. На рис. 10 показано начало выполнение транзакции, в которой было добавлено новое значение и изменены цены на книги.

```
var4=# begin;
BEGIN
var4=*# insert into books values (99, 'book1', 'author1', 10, 1, 8, 7, 500, 2023);
INSERT 0 1
var4=*# update books set price = 2.5 + price * 0.95;
UPDATE 12
var4=*# select * from books;
```

id	book_name	book_author	book_amount	genre_id	publisher_id	city_id	price	year
5	Человек-невидимка	Ральф Эллисон	23	1	10	8	184	2013
6	Гордость и предубеждение	Джейн Остен	18	5	9	9	128	1981
9	Путешествия Гулливера	Джонатан Свифт	16	1	7	5	112	2023
4	Звук и ярость	Уильям Фолкнер	95	1	7	1	60	1964
7	Божественная комедия	Данте Алигьери	95	7	5	3	79	1994
1	Война и мир	Лев Толстой	27	3	10	1	135	2020
2	1984	Джордж Оруэлл	95	3	7	4	97	2006
3	Улисс	Джеймс Джойс	60	5	2	4	77	1956
8	Кентерберийские рассказы	Джеффри Чосер	60	6	6	10	161	1960
10	Распад	Чинуа Ачебе	48	6	1	9	117	2019
11	Безымянная книга	Неизвестный автор	88	7	3	6	72	2004
99	book1	author1	10	1	8	7	478	2023

(12 строк)

Рисунок 10 – Начало выполнения транзакции.

Теперь, не завершая транзакцию, подключимся к нашей БД под новым созданным пользователем. На рис. 11 показан вывод таблицы, над которой выполняется транзакция. Из нее видно, что выполненные запросы внутри транзакционного блока не повлияли на текущее состояние таблицы.

```
var4=> \! chcp 1251
Текущая кодовая страница: 1251
var4=> select * from books;
```

id	book_name	book_author	book_amount	genre_id	publisher_id	city_id	price	year
5	Человек-невидимка	Ральф Эллисон	23	1	10	8	191	2013
6	Гордость и предубеждение	Джейн Остен	18	5	9	9	132	1981
9	Путешествия Гулливера	Джонатан Свифт	16	1	7	5	115	2023
4	Звук и ярость	Уильям Фолкнер	95	1	7	1	61	1964
7	Божественная комедия	Данте Алигьери	95	7	5	3	81	1994
1	Война и мир	Лев Толстой	27	3	10	1	139	2020
2	1984	Джордж Оруэлл	95	3	7	4	99	2006
3	Улисс	Джеймс Джойс	60	5	2	4	78	1956
8	Кентерберийские рассказы	Джеффри Чосер	60	6	6	10	167	1960
10	Распад	Чинуа Ачебе	48	6	1	9	121	2019
11	Безымянная книга	Неизвестный автор	88	7	3	6	73	2004

(11 строк)

Рисунок 11 – Просмотр таблицы под другим пользователем.

Под новым пользователем выполним запросы добавления и удаления значения (рис. 12-13). Добавление значения в таблицу выполнилось без проблем. В

то же время при попытке удаления значения из таблицы, psql не возвращает никакого результата, что говорит о том, что psql ожидает, пока завершится транзакция или произойдет ее откат.

```
var4=*# select * from books;
 id |        book_name         |     book_author      | book_amount | genre_id | publisher_id | city_id | price | year
----+--------------------------+----------------------+-------------+----------+--------------+---------+-------+------
  5 | Человек-невидимка        | Ральф Эллисон        |          23 |        1 |           10 |       8 |   184 | 2013
  6 | Гордость и предубеждение | Джейн Остен          |          18 |        5 |            9 |       9 |   128 | 1981
  9 | Путешествия Гулливера    | Джонатан Свифт       |          16 |        1 |            7 |       5 |   112 | 2023
  4 | Звук и ярость            | Уильям Фолкнер       |          95 |        1 |            7 |       1 |    60 | 1964
  7 | Божественная комедия     | Данте Алигьери       |          95 |        7 |            5 |       3 |    79 | 1994
  1 | Война и мир              | Лев Толстой          |          27 |        3 |           10 |       1 |   135 | 2020
  2 | 1984                     | Джордж Оруэлл        |          95 |        3 |            7 |       4 |    97 | 2006
  3 | Улисс                    | Джеймс Джойс         |          60 |        5 |            2 |       4 |    77 | 1956
  8 | Кентерберийские рассказы | Джеффри Чосер        |          60 |        6 |            6 |      10 |   161 | 1960
 10 | Распад                   | Чинуа Ачебе          |          48 |        6 |            1 |       9 |   117 | 2019
 11 | Безымянная книга         | Неизвестный автор    |          88 |        7 |            3 |       6 |    72 | 2004
 99 | book1                    | author1              |          10 |        1 |            8 |       7 |   478 | 2023
 98 | book2                    | author2              |          10 |        1 |            8 |       7 |   500 | 2023
(13 строк)
```

Рисунок 12 – Успешное добавление значения в таблицу с незавершенной транзакцией.

```
var4=> delete from books where id = 11;
```

Рисунок 13 - Попытка удаления значения из таблицы с незавершенной транзакцией

PostgreSQL предоставляет механизм создания и управления пользователями для обеспечения безопасности и контроля доступа к данным. Для создания пользователя используются команды `CREATE USER` и `ALTER USER`.

–`CREATE USER` - создает нового пользователя с заданным именем, атрибутами (пароль, роли) и ограничениями.

–`ALTER USER` - позволяет изменять атрибуты существующего пользователя.

Транзакция - это логическая единица работы, которая рассматривается как единая неделимая операция. В PostgreSQL транзакции обеспечивают ACID-свойства:

– Атомарность (Atomicity): Транзакция либо выполняется полностью, либо не выполняется вообще;

– Согласованность (Consistency): Транзакция должна переводить базу данных из одного корректного состояния в другое;

– Изоляция (Isolation): Транзакции выполняются независимо друг от друга, как если бы они выполнялись поочередно;

– Долговечность (Durability): Изменения, внесенные в базу данных в ходе успешно завершенной транзакции, сохраняются.

Транзакции управляются командами `BEGIN`, `COMMIT` и `ROLLBACK`.

– `BEGIN` - инициирует транзакцию.

– `COMMIT` - завершает транзакцию, сохраняя изменения.

– `ROLLBACK` - отменяет транзакцию, откатывая все изменения.

Ограничения в PostgreSQL**

Ограничения - это правила, которые накладываются на таблицы и столбцы для обеспечения целостности данных и соответствия бизнес-правилам. В PostgreSQL существуют следующие типы ограничений:

– NOT NULL: Запрещает вставку пустых значений в столбец;

– UNIQUE: Гарантирует уникальность значений в столбце или комбинации столбцов;

– PRIMARY KEY: Определяет уникальный ключ для таблицы, автоматически устанавливая ограничение NOT NULL и UNIQUE;

– FOREIGN KEY: Создает связь между двумя таблицами, обеспечивая целостность данных при удалении или обновлении записей;

– CHECK: Проверяет значения столбца или комбинации столбцов, ограничивая их допустимый диапазон;

– DEFAULT: Задает значение по умолчанию для столбца, если при вставке не указано другое значение.

PostgreSQL позволяет создавать производные таблицы, которые содержат данные, полученные из других таблиц.

– VIEW: Виртуальная таблица, которая не хранит данные, а представляет собой запрос к одной или нескольким базовым таблицам.

– MATERIALIZED VIEW: Таблица, которая хранит данные, полученные из базовых таблиц.

Для просмотра структуры таблицы используется команда `\d <имя_таблицы>`. Она отображает информацию о столбцах, ограничениях, индексах и других характеристиках таблицы.

Работа с базой данных книжной лавки.

В задании требуется заполнить базу данных, относящуюся к книжной лавке. В качестве примера будут использованы следующие поля:

- `название`: название книги
- `жанр`: жанр книги
- `стоимость`: цена книги
- `год_выпуска`: год выпуска книги
- `тираж`: тираж книги

На эти поля будут наложены ограничения, чтобы обеспечить целостность данных.

Практическая часть

Рабочее задание: Ознакомиться с теоретическими сведениями о возможностях создания пользователей баз данных, использования транзакций в

PostgreSQL. Создать нового пользователя и зайти под его именем. Ознакомиться с теоретическими сведениями о возможностях создания ограничений в PostgreSQL. Наложить ограничения согласно своему варианту. Проверить работоспособность ограничений путем добавления в таблицы данных, удовлетворяющих и не удовлетворяющих условиям ограничений. Создать транзакционный блок, в котором производится добавление в таблицы произвольных полей, создать несколько производных таблиц, просмотреть структуру измененных таблиц. Не завершая транзакции параллельно запустить еще одно окно терминала, подключиться к базе и попробовать добавить и удалить записи в таблицы. Сделать откат транзакций, просмотреть структуру таблиц. Просмотреть и проанализировать полученную в результате выполнения операций информацию.

Заполнить базу данных для обработки данных по работе книжной лавки (при необходимости). Ограничить заполнение поля «жанр» следующими наименованиями: детектив, фантастика, комедия, мелодрама, боевик. На поле «стоимость» наложить ограничение таким образом, чтобы нельзя было ввести цену ниже 50 р. Год выпуска книги не должен быть в будущем (должен соответствовать действительности). Для поля «тираж» ограничить ввод значений менее 100 штук. Продемонстрировать результаты работы.

Практическая работа № 4. Использование агрегатных функций в среде PostgreSQL. Массивы.

Цель работы: Изучить базовые операции по работе с массивами. Изучить синтаксис команд. Приобрести навыки работы с агрегатными функциями в PostgreSQL.

Теоретическая часть

Добавим в существующие таблицы столбцы, содержащие одномерный и многомерный массивы (рис. 1-2).

```
var4=# ALTER TABLE authors ADD book_list text[];
ALTER TABLE
var4=# \d authors
                    Таблица "public.authors"
  Столбец  |   Тип   | Правило сортировки | Допустимость NULL | По умолчанию
-----------+---------+--------------------+-------------------+--------------
 id        | integer |                    |                   |
 author    | text    |                    |                   |
 city_id   | integer |                    |                   |
 book_list | text[]  |                    |                   |
```

Рисунок 1 – Таблица с одномерным массивом и его структура.

```
var4=*# alter table publishers add book_info integer[][];
ALTER TABLE
var4=*# \d publishers
                              Таблица "public.publishers"
      Столбец      |      Тип       | Правило сортировки | Допустимость NULL | По умолчанию
-------------------+----------------+--------------------+-------------------+--------------
 id                | integer        |                    |                   |
 publisher_name    | character(20)  |                    |                   |
 publisher_address | text           |                    |                   |
 book_info         | integer[]      |                    |                   |
```

Рисунок 2 – Таблица с многомерным массивом и его структура.

В таблице authors будут храниться книги авторов, в таблице publishers будут храниться массивы из двух элементов, содержащие идентификаторы книг и авторов.

Вставим данные в новые столбцы (рис. 3-4). Для этого использовалась функция array_agg(), которая объединяет данные в массив.

```
var4=# update authors set book_list = (
var4(# select array_agg(name) from books
var4(# where books.author_id = authors.id);
UPDATE 6
var4=# select * from authors;
 id |     author     | city_id |                                book_list
----+----------------+---------+-------------------------------------------------------------------------
  3 | Иван Иванов    |       5 | {Человек-невидимка,"Божественная комедия","Тёртый калач",Превращение}
  1 | Ральф Эллисон  |       8 | {1984,Улисс,"Кентерберийские рассказы"}
  2 | Джейн Остен    |       9 | {"Тридцатилетняя женщина","Вокруг света в восемьдесят дней"}
  4 | Уильям Фолкнер |       1 | {"Путешествия Гулливера"}
  5 | Данте Алигьери |       3 | {"Гордость и предубеждение","Звук и ярость",Распад}
  6 | Лев Толстой    |       1 | {"Война и мир","Роб Рой"}
(6 строк)
```

Рисунок 3 – Вставка данных в таблицу с одномерным массивом.

```
var4=*# update publishers set book_info = (
var4(*# select array_agg(x) from (
var4(*#  select publisher_id, (array[books.id]  || author_id) x
var4(*#  from books, authors
var4(*#  where books.author_id = authors.id and publisher_id = publishers.id
var4(*# ) tmp1 group by publisher_id
var4(*# );
UPDATE 10
var4=*# select * from publishers;
 id |    publisher_name    |                publisher_address                |             book_info
----+----------------------+-------------------------------------------------+-----------------------------------
  9 | Сибирь               | г. Москва, ул. Усиевича                         | {{15,3},{8,1},{14,2},{13,2},{12,6}}
  2 | АСТ                  | г. Москва, Пресненская наб.                     |
  3 | Триумф               | г. Москва, ул. Михалковская                     | {{11,3}}
  4 | РИПОЛ классик        | г. Москва, Нижегородская ул.                    | {{10,5},{4,5}}
  5 | Росмэн               | г. Москва, Ленинградский проспект               | {{7,3}}
  6 | Феникс               | г. Ростов-на-Дону, Ростовская обл., ул. Варфоломеева | {{3,1}}
  7 | Центрполиграф        | г. Москва, 1-я ул.Энтузиастов                   | {{2,1},{6,5}}
  8 | Проспект             | г. Москва, ул.Мосфильмовская                    | {{5,3},{9,4}}
 10 | Питер                | г. Санкт-Петербург, Б. Сампсониевский пр.       | {{1,6}}
  1 | Эксмо                | г. Москва, ул. Зорге                            |
```

Рисунок 4 – Вставка данных в таблицу с многомерным массивом.

Данные в массив можно также добавить вручную (рис. 5).

```
var4=*# update authors set book_list[3] = 'Книга1' where id = 6;
UPDATE 1
var4=*# select * from authors;
 id |     author     | city_id |                              book_list
----+----------------+---------+-----------------------------------------------------------------------
  3 | Иван Иванов    |       5 | {Человек-невидимка,"Божественная комедия","Тёртый калач",Превращение}
  1 | Ральф Эллисон  |       8 | {1984,Улисс,"Кентерберийские рассказы"}
  2 | Джейн Остен    |       9 | {"Тридцатилетняя женщина","Вокруг света в восемьдесят дней"}
  4 | Уильям Фолкнер |       1 | {"Путешествия Гулливера"}
  5 | Данте Алигьери |       3 | {"Гордость и предубеждение","Звук и ярость",Распад}
  6 | Лев Толстой    |       1 | {"Война и мир","Роб Рой",Книга1}
(6 строк)
```

Рисунок 5 – Пример ручного добавления значения в массив.

При указании несуществующего элемента массива возвращается значение NULL. Чтобы предотвратить получение значения NULL используется условие IS NOT NULL (рис. 6).

```
var4=# select book_list[3] from authors;
        book_list
--------------------------
 Тёртый калач
 Кентерберийские рассказы


 Распад

(6 строк)


var4=# select book_list[3] from authors where book_list[3] is not null;
        book_list
--------------------------
 Тёртый калач
 Кентерберийские рассказы
 Распад
(3 строки)
```

Рисунок 6 – Пример предотвращения выборки значения NULL.

Выполним выборку данных согласно варианту с помощью агрегатных функций и созданных столбцов с массивами (рис. 7-11). Для разбиения массива на отдельные данные использовалась функция unnest(). Однако недостаток этой функции в том, что данные многомерного массива разбиваются на множество данных, которые хранились в подмассивах. Для решения данной проблемы использовалась следующая конструкция:

array [unnest(double_array [:][1])] || unnest(double_array[:][2:2])

Работа этой конструкции следующая: из двумерного массива выбирается первый элемент подмассива, затем они сворачиваются в обычный массив и с помощью оператора || в созданный массив добавляется второй элемент подмассива.

```
var4=# select publisher_name, x, books.size from (
var4(#   select (array[unnest(book_info[:][1])] || unnest(book_info[:][2:2])) x
var4(#   from publishers
var4(# ) tmp1, publishers, books
var4-# where x[1] = any(book_info[:][1])
var4-#   and books.id = x[1]
var4-#   and books.size = (select max(size) from books);
    publisher_name    |   x   | size
----------------------+-------+------
 Проспект             | {5,3} |  553
(1 строка)
```

Рисунок 7 – Получение книги с максимальным количеством страниц с использованием двумерного массива.

```
var4=# select publisher_name, x, books.size from (
var4(#   select (array[unnest(book_info[:][1])] || unnest(book_info[:][2:2])) x
var4(#   from publishers
var4(# ) tmp1, publishers, books
var4-# where x[1] = any(book_info[:][1])
var4-#   and books.id = x[1]
var4-#   and books.size = (select min(size) from books where genre_id = 7);
    publisher_name    |   x    | size
----------------------+--------+------
 Сибирь               | {15,3} |   64
(1 строка)
```

Рисунок 8 – Получение книги жанра “комедия” с минимальным количеством страниц с использованием двумерного массива.

```
var4=# select avg(books.price) from (select unnest(book_list) x from authors) tmp1, books
var4-# where books.name = x and books.genre_id = 5;
        avg
--------------------
 114.6666666666666667
(1 строка)
```

Рисунок 9 - Получение средней цены книг жанра “фантастика” с использованием одномерного массива.

```
var4=# select array_dims(book_info) from publishers where publisher_name = 'Сибирь';
 array_dims
------------
 [1:5][1:2]
(1 строка)
```

рисунок 10 – Получение количества книг издательства “Сибирь” с использованием одномерного массива

```
var4=# select sum(price)
var4-# from (select unnest(book_list) x from authors where city_id = 1) tmp1, books
var4-# where books.name = x;
 sum
-----
 381
(1 строка)
```

Рисунок 11 – Получение общей цены книг авторов из Москвы с использованием одномерного массива.

В PostgreSQL массивы представляют собой упорядоченные коллекции элементов одного типа данных. Они предоставляют удобный способ хранения и обработки множественных значений в одной записи базы данных.

Для создания таблицы с полем, содержащим массив, необходимо использовать тип данных `ARRAY`. Пример:

```sql
CREATE TABLE books (
  id SERIAL PRIMARY KEY,
  title TEXT,
  authors TEXT [],
  genres TEXT [],
  price NUMERIC (10,2)
);
```

В данном примере поля `authors` и `genres` имеют тип `TEXT[]`, что означает, что они будут хранить массивы текстовых значений.

Многомерные массивы:

В PostgreSQL можно создавать многомерные массивы, где каждый элемент массива также является массивом. Пример:

```sql
CREATE TABLE book_sales (
  id SERIAL PRIMARY KEY,
  book_id INTEGER,
  sales_data INTEGER [][]
);
```

В данном примере поле `sales_data` является двумерным массивом, где каждый элемент является массивом целых чисел.

Основные операции с массивами: *Вставка данных*: для вставки значений в массив при добавлении новой записи необходимо использовать синтаксис `'{значение1, значение2, ...}'`; *Выбор данных*: Для выборки данных из массива используются следующие функции:

`array_to_string (массив, разделитель)` - преобразует массив в строку с указанным разделителем.

`array_position (массив, значение)` - возвращает позицию элемента в массиве.

`array_append (массив, значение)` - добавляет значение в конец массива.

`array_remove (массив, значение)` - удаляет все вхождения значения из массива.;

Срезы: Для выборки подмножеств элементов массива можно использовать срезы. Например, `массив [1:3]` вернет элементы с позиций 1, 2, 3.

- Функция `array_dims(массив)`: Возвращает размерность массива.
- Обновление данных: Для обновления элементов массива используются операторы `=` и `:=`.

•Модификация среза массива: Можно обновлять значения в срезе массива, используя `массив [1:3] := ARRAY['значение1', 'значение2', 'значение3']`.

•Предотвращение выборки NULL: При работе с массивами важно учитывать, что NULL не является элементом массива. Для предотвращения выборки NULL можно использовать функцию `coalesce (массив, ARRAY[]).`

В контексте книжной лавки массивы могут быть использованы для хранения следующих данных:

- Авторы книги
- Жанры книги
- Список продаж книг
- Информация о издательстве

Примеры запросов:

Найти книгу с максимальным количеством страниц:

```sql
SELECT title, pages FROM books ORDER BY pages DESC LIMIT 1;
```

Найти книгу жанра «комедия» с минимальным количеством страниц:

```sql
SELECT title, pages FROM books WHERE 'комедия' = ANY(genres) ORDER BY pages LIMIT 1;
```

Найти среднюю цену книг жанра «фантастика»:

```sql
SELECT AVG(price) FROM books WHERE 'фантастика' = ANY(genres);
```

Найти количество книг издательства «Сибирь»:

```sql
SELECT COUNT(*) FROM books WHERE 'Сибирь' = ANY(publisher);
```

Найти общую стоимость книг для авторов из Москвы:

```sql
SELECT SUM(price) FROM books WHERE 'Москва' = ANY(authors_city);
```

Практическая честь

Рабочее задание: Ознакомиться с теоретическими сведениями о создании массивов. Создать таблицу с полем-массивом, таблицу с полем, содержащим многомерный массив. Выполнить вставку значений в созданные таблицы (минимум по 6 записей в каждой). Выполнить выборку из созданных таблиц (в том числе

продемонстрировать предотвращение выборки NULL в массивах). Осуществить выборку с использованием среза. Продемонстрировать работу функции array_dims(). Выполнить обновление данных в созданных таблицах. Осуществить модификацию среза массива, отдельного элемента массива. Проанализировать полученную в результате выполнения операций информацию.

Заполнить базу данных для обработки данных по работе книжной лавки (при необходимости). Найти книгу с максимальным количеством страниц. Найти книгу жанра «комедия» с минимальным количеством страниц. Найти среднюю цену книг жанра «фантастика». Найти количество книг издательства «Сибирь». Найти общую стоимость книг для авторов из Москвы. Продемонстрировать результаты работы.

Практическая работа № 5. Многотабличные запросы и подзапросы в среде PostgreSQL.

Цель работы: Изучить порядок формирования многотабличных запросов и подзапросов в среде PostgreSQL. Приобрести навыки работы с многотабличными запросами и подзапросами в среде PostgreSQL.

Теоретическая часть

Добавим в существующие таблицы столбцы (рис. 1-2) и создадим новую таблицу, содержащую информацию об издании книг (рис. 3).

```
var4=# alter table cities add country_name text;
ALTER TABLE
var4=# select * from cities;
          city_name           | id | country_name
------------------------------+----+--------------
 Москва                       |  1 |
 Санкт-Петербург              |  2 |
 Новосибирск                  |  3 |
 Екатеринбург                 |  4 |
 Казань                       |  5 |
 Нижний Новгород              |  6 |
 Челябинск                    |  7 |
 Красноярск                   |  8 |
 Самара                       |  9 |
 Уфа                          | 10 |
(10 строк)
```

Рисунок 1 - Добавление столбца country_name в таблицу cities.

```
var4=# alter table books add book_type text;
ALTER TABLE
var4=# select * from books;
 id |              name               | amount | price | year | author_id | genre_id | publisher_id | size | book_type
----+---------------------------------+--------+-------+------+-----------+----------+--------------+------+-----------
  6 | Гордость и предубеждение        |     18 |   132 | 1981 |         5 |        5 |            7 |   89 |
  1 | Война и мир                     |     27 |   139 | 2020 |         6 |        3 |           10 |  296 |
  2 | 1984                            |     95 |    99 | 2006 |         1 |        3 |            7 |  146 |
  3 | Улисс                           |     60 |    78 | 1956 |         1 |        5 |            6 |  332 |
  4 | Звук и ярость                   |     95 |    61 | 1964 |         5 |        1 |            4 |  391 |
  5 | Человек-невидимка               |     23 |   191 | 2013 |         3 |        1 |            8 |  553 |
  7 | Божественная комедия            |     95 |    81 | 1994 |         3 |        7 |            5 |  369 |
  8 | Кентерберийские рассказы        |     60 |   167 | 1960 |         1 |        6 |            9 |   70 |
  9 | Путешествия Гулливера           |     16 |   115 | 2023 |         4 |        1 |            8 |  335 |
 10 | Распад                          |     48 |   121 | 2019 |         5 |        6 |            4 |  180 |
 11 | Тёртый калач                    |     80 |   131 | 2015 |         3 |        1 |            3 |  324 |
 12 | Роб Рой                         |     57 |   127 | 2019 |         6 |        1 |            9 |   69 |
 13 | Тридцатилетняя женщина          |     25 |   134 | 2019 |         2 |        5 |            9 |  377 |
 14 | Вокруг света в восемьдесят дней |     67 |   141 | 2019 |         2 |        6 |            9 |  246 |
 15 | Превращение                     |      4 |   141 | 2019 |         3 |        7 |            9 |   64 |
(15 строк)
```

Рисунок 2 - Добавление столбца book_type в таблицу books.

```
var4=# create table publish_info (
var4(# book_id integer,
var4(# publisher_id integer,
var4(# pub_date date,
var4(# city_id integer
var4(# );
CREATE TABLE
var4=# \d publish_info
                    Таблица "public.publish_info"
   Столбец    |   Тип   | Правило сортировки | Допустимость NULL | По умолчанию
--------------+---------+--------------------+-------------------+--------------
 book_id      | integer |                    |                   |
 publisher_id | integer |                    |                   |
 pub_date     | date    |                    |                   |
 city_id      | integer |                    |                   |
```

Рисунок 3 - Создание таблицы publish_info.

Выполним выборку данных:

1. Найти все повести и романы, в которых от 250 до 500 страниц, и с ценой больше, чем средняя цена книг заданного издательства (рис. 4).
2. Найти всю литературу автора Иванова, выпущенную в Москве и Новосибирске, и с ценой выше, чем средняя цена произведений жанра «детектив», выпущенных за последние полгода, автором Ивановым (рис. 5).
3. Найти все поэтические произведения заданного издательства и с ценой больше, чем средняя цена книг, выпущенных в Чехии и Словакии (рис. 6).

```
var4=# select * from books
var4-# where (book_type = any(array['Повесть', 'Роман'])) and (size >= 250) and (size <= 500) and (price > (
var4(#   select avg(price) from books inner join publishers on
var4(#   publishers.id = books.publisher_id and publisher_name = 'РИПОЛ классик'
var4(# ));
 id |       name        | amount | price | year | author_id | genre_id | publisher_id | size | book_type
----+-------------------+--------+-------+------+-----------+----------+--------------+------+-----------
 41 | Убить пересмешника |     87 |  1537 | 2004 |         6 |        5 |            2 |  377 | Повесть
(1 строка)
```

Рисунок 4 - Вывод книг, согласно выборке.

```
var4=# select books.id, books.name, books.price, authors.author, cities.city_name from books
var4-# inner join authors on authors.id = books.author_id
var4-# inner join publish_info on books.id = publish_info.book_id
var4-# inner join cities on cities.id = publish_info.city_id
var4-# where (authors.author = 'Иван Иванов')
var4-#   and (city_name = any(array['Москва','Новосибирск']))
var4-#   and (price > (
var4(#   select avg(price) from books
var4(#   inner join genres on genres.id = books.genre_id
var4(#   inner join publish_info on books.id = publish_info.book_id
var4(#   inner join authors on authors.id = books.author_id
var4(#   where (genres.genre_name = 'Детектив')
var4(#     and (pub_date > (now() - interval '6 months')::date)
var4(#     and (authors.author = 'Иван Иванов')
var4(# ));
 id |         name         | price |   author    |           city_name
----+----------------------+-------+-------------+-------------------------------
  7 | Божественная комедия |   130 | Иван Иванов | Новосибирск
(1 строка)
```

Рисунок 5. Вывод книг, согласно выборке.

```
var4=# select books.* from books inner join publishers on publisher_id = publishers.id
var4-#   where (book_type = 'Поэма') and (publisher_name = 'Проспект') and (price > (
var4(#   select avg(price) from publish_info
var4(#   inner join cities on publish_info.city_id = cities.id
var4(#   inner join books on publish_info.book_id = books.id
var4(#   where country_name = any(array['Чехия','Словакия'])
var4(# ));
 id |   name   | amount | price | year | author_id | genre_id | publisher_id | size | book_type
----+----------+--------+-------+------+-----------+----------+--------------+------+-----------
 40 | В дороге |     31 |   602 | 1942 |        10 |        3 |            8 | 1164 | Поэма
(1 строка)
```

Рисунок 6 – Вывод книг, согласно выборке.

Многотабличные запросы в PostgreSQL позволяют объединять данные из нескольких таблиц в единый результат. Основные операции для объединения данных:

- JOIN: Объединяет данные из нескольких таблиц на основе общего столбца (ключей). Существуют различные типы JOIN:
- INNER JOIN: Возвращает только те строки, которые совпадают в обеих таблицах.
- LEFT JOIN: Возвращает все строки из левой таблицы, а из правой - только те, которые соответствуют условиям.
- RIGHT JOIN: Возвращает все строки из правой таблицы, а из левой - только те, которые соответствуют условиям.
- FULL JOIN: Возвращает все строки из обеих таблиц, независимо от того, есть ли совпадения.
- UNION: Объединяет результаты двух запросов. Строки в результирующем наборе должны иметь одинаковое количество столбцов.
- UNION ALL: Объединяет результаты двух запросов, сохраняя дубликаты строк.

Подзапросы (или вложенные запросы) - это запросы, вложенные внутри других запросов. Они позволяют:

- Выбрать данные из таблицы, используя результаты другого запроса. Например, найти всех сотрудников, чья зарплата выше средней по компании;
- Фильтровать данные по результатам другого запроса.** Например, выбрать всех клиентов, которые сделали больше трёх заказов;
- Изменить структуру данных. Например, вычислить среднее значение по каждой группе данных.

Синтаксис подзапросов:

```sql
SELECT * FROM table1
WHERE column1 IN (SELECT column2 FROM table2 WHERE condition);
```

Примеры использования:

Найти всех сотрудников, чья зарплата выше средней по компании:

```sql
SELECT * FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

Выбрать всех клиентов, которые сделали больше трёх заказов:

```sql
SELECT * FROM customers
WHERE id IN (SELECT customer_id FROM orders GROUP BY customer_id HAVING COUNT(*) > 3);
```

Преимущества использования многотабличных запросов из подзапросов:

- Увеличение гибкости и выразительности SQL-запросов;
- Возможность выполнения сложных операций с данными;
- Улучшение читаемости и понимания запросов.

Пример создания многотабличного запроса:

```sql
SELECT c.name AS customer_name, o.order_date, p.name AS product_name
FROM customers c
JOIN orders o ON c.id = o.customer_id
JOIN order_items oi ON o.id = oi.order_id
JOIN products p ON oi.product_id = p.id;
```

Пример создания подзапроса:

```sql
SELECT * FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

```

Используйте `EXPLAIN` для анализа плана выполнения запроса и оптимизации производительности.

Подбирайте оптимальные типы JOIN для выполнения запросов.

Оптимизируйте подзапросы для улучшения производительности, например, с помощью оператора `EXISTS`.

**Практическая часть**

**Рабочее задание:** Ознакомиться с теоретическими сведениями о возможностях создания многотабличных запросов и подзапросов в среде PostgreSQL. Выполнить выборку из созданных таблиц согласно варианту. Если в базе нет данных, удовлетворяющих условиям запроса, то добавить подходящие данные. Продемонстрировать результаты работы.

Найти все повести и романы, в которых от 250 до 500 страниц, и с ценой больше, чем средняя цена книг заданного издательства. Найти всю литературу автора Иванова, выпущенную в Москве и Новосибирске, и с ценой выше, чем средняя цена произведений жанра «детектив», выпущенных за последние полгода, автором Ивановым. Найти все поэтические произведения заданного издательства и с ценой больше, чем средняя цена книг, выпущенных в Чехии и Словакии.

**Практическая работа № 6. Изучение индексов в PostgreSQL**

**Цель:** Изучение индексов в PostgreSQL и их влияния на скорость поиска информации в базе данных. Изучение команды EXPLAIN. Получение навыков по работе с командой SELECT в среде PostgreSQL

**Теоретическая часть**

Создадим функцию add_n(int), которая будет добавлять псевдослучайно сгенерированные данные в таблицу books (рис. 1).
```

```
var4=# CREATE FUNCTION add_n(int) RETURNS char AS
var4-# $$
var4$# DECLARE
var4$#   t int;
var4$# BEGIN
var4$#   SELECT max(id) INTO t FROM books;
var4$#   FOR k IN (t+1)..($1+t) LOOP
var4$#   INSERT INTO books VALUES(
var4$#     k,
var4$#     'book_' || 2*k,
var4$#     10 + round(random() * 900),
var4$#     50 + round(random() * 1400),
var4$#     1990 + round(random() * 33),
var4$#     1 + round(random() * 5),
var4$#     (array[1, 3, 5, 6, 7])[1 + round(random() * 4)],
var4$#     1 + round(random() * 9),
var4$#     70 + round(random() * 1100),
var4$#     (array['Повесть', 'Роман', 'Поэма', 'Рассказ', 'Новелла'])[1 + round(random() * 4)]);
var4$#   END LOOP;
var4$#   RETURN 'Done!';
var4$# END;
var4$# $$
var4-# LANGUAGE 'plpgsql';
CREATE FUNCTION
var4=# \df
                          Список функций
 Схема  |  Имя   | Тип данных результата | Типы данных аргументов |  Тип
--------+--------+-----------------------+------------------------+-------
 public | add_n | character             | integer                | функ.
(1 строка)
```

Рисунок 1 – Функция add_n(int).

С помощью данной функции добавим 1000 записей (рис. 2).

```
var4=# select add_n(1000);
 add_n
-------
 Done!
(1 строка)


var4=# select count(*) from books;
 count
-------
  1031
(1 строка)
```

Рисунок 2 - Добавление 1000 записей.

Далее будем анализировать выполнение запроса SELECT * FROM books WHERE book_type = ‘Роман’ AND price > 1000. Чтобы получить данные о выполненном запросе, добавим EXPLAIN ANALYSE перед запросом. Также с помощью команды \timing включим отображение времени выполнения запроса.

Выполним выборку без индексов (рис. 3).

```
var4=# explain analyse
var4-# select * from books where book_type = 'Роман' and price > 1000;
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------------
 Seq Scan on books  (cost=0.00..27.46 rows=94 width=54) (actual time=0.015..0.177 rows=91 loops=1)
   Filter: ((price > 1000) AND (book_type = 'Роман'::text))
   Rows Removed by Filter: 940
 Planning Time: 0.887 ms
 Execution Time: 0.194 ms
(5 строк)


Время: 1,481 мс
```

Рисунок 3 - Результат выполненного запроса без индексов.

Теперь создадим индекс типа BTREE (рис. 4) и снова выполним запрос (рис. 5).

```
var4=# create index ib on books using btree (book_type);
CREATE INDEX
Время: 14,341 мс
```

Рисунок 4 - Создание индекса типа BTREE для столбца book_type.

```
var4=# explain analyse
var4-# select * from books where book_type = 'Роман' and price > 1000;
                                                  QUERY PLAN
---------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on books  (cost=6.29..22.52 rows=94 width=54) (actual time=0.097..0.224 rows=91 loops=1)
   Recheck Cond: (book_type = 'Роман'::text)
   Filter: (price > 1000)
   Rows Removed by Filter: 191
   Heap Blocks: exact=12
   ->  Bitmap Index Scan on ib  (cost=0.00..6.27 rows=282 width=0) (actual time=0.070..0.070 rows=282 loops=1)
         Index Cond: (book_type = 'Роман'::text)
 Planning Time: 0.279 ms
 Execution Time: 0.295 ms
(9 строк)


Время: 1,371 мс
```

Рисунок 5 - Результат выполненного запроса с индексом типа BTREE.

Как видно из рис. 5, время потраченное на запрос создание индекса, заметно больше, чем для обычного запроса. Это связано с тем, что запрос на создание индекса является весьма ресурсозатратным. Однако, из рис. 6 видно, что на выполнение select запроса с индексом потребовалось меньше времени, чем без индекса. также поменялся план выполнения запроса. Удалим текущий индекс и создадим такой же, только типа HASH (рис. 6) и снова выполним запрос (рис. 7).

```
var4=# create index ih on books using hash (book_type);
CREATE INDEX
Время: 10,171 мс
```

Рисунок 6 – Создание индекса типа HASH для столбца book_type.

```
var4=# explain analyse
var4-# select * from books where book_type = 'Роман' and price > 1000;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on books  (cost=10.14..26.37 rows=94 width=54) (actual time=0.045..0.209 rows=91 loops=1)
   Recheck Cond: (book_type = 'Роман'::text)
   Filter: (price > 1000)
   Rows Removed by Filter: 191
   Heap Blocks: exact=12
   ->  Bitmap Index Scan on ih  (cost=0.00..10.12 rows=282 width=0) (actual time=0.025..0.025 rows=282 loops=1)
         Index Cond: (book_type = 'Роман'::text)
 Planning Time: 0.142 ms
 Execution Time: 0.242 ms
(9 строк)


Время: 0,825 мс
```

Рисунок 7 – Результат выполненного запроса с индексом типа HASH.

Получаем аналогичную ситуацию, как с индексом типа BTREE: большое время на создание индекса, измененный план выполнения запроса. Также сократилось и время выполнения запроса, которое оказалось быстрее, чем время выполнения запроса с индексом типа HASH. Удалим текущий индекс и выполним тот же запрос, но с использованием функции UPPER () (рис. 8).

```
var4=# explain analyse
var4-# select * from books where upper(book_type) = 'POMAH' and price > 1000;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Seq Scan on books  (cost=0.00..30.04 rows=2 width=54) (actual time=0.054..0.554 rows=91 loops=1)
   Filter: ((price > 1000) AND (upper(book_type) = 'POMAH'::text))
   Rows Removed by Filter: 940
 Planning Time: 0.086 ms
 Execution Time: 0.571 ms
(5 строк)


Время: 1,087 мс
```

Рисунок 8 – Результат выполненного запроса с использованием функции UPPER ().

Создадим функциональный индекс (рис.9) и снова выполним запрос (рис.10).

```
var4=# create index ibf on books using btree (upper(book_type));
CREATE INDEX
Время: 9,074 мс
```

Рисунок 9 – Создание функционального индекса для столбца book_type.

```
var4=# explain analyse
var4-# select * from books where upper(book_type) = 'POMAH' and price > 1000;
                                                    QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on books  (cost=4.19..14.60 rows=2 width=54) (actual time=0.043..0.135 rows=91 loops=1)
   Recheck Cond: (upper(book_type) = 'POMAH'::text)
   Filter: (price > 1000)
   Rows Removed by Filter: 191
   Heap Blocks: exact=12
   ->  Bitmap Index Scan on ibf  (cost=0.00..4.19 rows=5 width=0) (actual time=0.029..0.029 rows=282 loops=1)
         Index Cond: (upper(book_type) = 'POMAH'::text)
 Planning Time: 0.105 ms
 Execution Time: 0.170 ms
(9 строк)


Время: 0,639 мс
```

Рисунок 10 – Результат выполненного запроса с функциональным индексом.

Результат с функциональным индексом был выполнен быстрее, чем без него. Такое ускорение было достигнуто благодаря тому, что индексация производилась по результатам функции UPPER () для каждого значения в столбце. Выполним по варианту следующие выборки:

- Вывести полную информацию о произведениях, в которых более 300 страниц (рис. 11).
- Найти все дороже 1000 р. заданного автора (рис. 12).
- Найти всю литературу заданного издательства, выпущенную за последние три месяца (рис. 13).
- Найти все книги, чей тираж более 500 экземпляров для жанра «мелодрама» (рис. 14).
- Найти рассказы и новеллы жанров «фантастика» и «комедия» издательств «Сибирь» и «Наука» (рис. 15).

```
var4=# select * from books where size > 300 limit 10;
 id |          name          | amount | price | year | author_id | genre_id | publisher_id | size | book_type
----+------------------------+--------+-------+------+-----------+----------+--------------+------+-----------
  7 | Божественная комедия   |     95 |   130 | 1994 |         3 |        7 |            5 |  369 | Повесть
 55 | book_110               |    335 |  1299 | 2007 |         6 |        6 |            8 |  971 | Рассказ
  4 | Звук и ярость          |     95 |    61 | 1964 |         6 |        1 |            4 |  391 | Повесть
 56 | book_112               |    220 |   293 | 2011 |         4 |        3 |            9 |  415 | Новелла
 13 | Тридцатилетняя женщина |     25 |   134 | 2019 |         5 |        5 |            9 |  377 | Роман
  3 | Улисс                  |     60 |    78 | 1956 |         5 |        5 |            6 |  332 | Поэма
  5 | Человек-невидимка      |     23 |   191 | 2013 |         2 |        1 |            8 |  553 | Поэма
 11 | Тёртый калач           |     80 |   131 | 2015 |         4 |        1 |            3 |  324 | Поэма
 58 | book_116               |    635 |   501 | 2002 |         6 |        1 |           10 |  758 | Новелла
 59 | book_118               |    457 |   352 | 1993 |         2 |        1 |            9 |  999 | Роман
(10 строк)


var4=# select count(*) from books where size > 300;
 count
-------
   811
(1 строка)
```

Рисунок 11 – Вывод книг с более 300 страниц.

```
var4=# select books.name, books.price, authors.author from books
var4-# inner join authors on books.author_id = authors.id
var4-# where (price > 1000) and (author = 'Иван Иванов')
var4-# limit 10;
    name     | price |    author
-------------+-------+--------------
 book_198    |  1398 | Иван Иванов
 book_208    |  1370 | Иван Иванов
 book_224    |  1177 | Иван Иванов
 Король Лир |  1130 | Иван Иванов
 book_284    |  1382 | Иван Иванов
 book_286    |  1447 | Иван Иванов
 book_386    |  1117 | Иван Иванов
 book_408    |  1149 | Иван Иванов
 book_446    |  1122 | Иван Иванов
 book_448    |  1185 | Иван Иванов
(10 строк)


var4=# select count(*) from books
var4-# inner join authors on books.author_id = authors.id
var4-# where (price > 1000) and (author = 'Иван Иванов');
 count
-------
    64
(1 строка)
```

Рисунок 12 – Вывод книг автора Иванова стоимостью более 1000 рублей.

```
var4=# select books.* from books
var4-#  inner join publish_info on books.id = publish_info.book_id
var4-#  inner join publishers on books.publisher_id = publishers.id
var4-# where (publishers.publisher_name = 'Эксмо')
var4-#  and (pub_date > (now() - interval '3 months')::date)
var4-# limit 10;
 id  |   name    | amount | price | year | author_id | genre_id | publisher_id | size | book_type
-----+-----------+--------+-------+------+-----------+----------+--------------+------+-----------
 219 | book_438  |    856 |  1341 | 2013 |         4 |        3 |            1 |  483 | Поэма
 832 | book_1664 |    589 |  1049 | 2015 |         4 |        5 |            1 |  145 | Роман
 961 | book_1922 |    135 |   960 | 1999 |         2 |        1 |            1 |  238 | Рассказ
(3 строки)
```

Рисунок 13 – Вывод книг издательства «Эксмо», выпущенные за последние 3 месяца.

```
var4=# select books.* from books
var4-# inner join genres on books.genre_id = genres.id
var4-# where (amount > 500) and (genre_name = 'Мелодрама')
var4-# limit 10;
 id  |   name    | amount | price | year | author_id | genre_id | publisher_id | size | book_type
-----+-----------+--------+-------+------+-----------+----------+--------------+------+-----------
  69 | book_138  |    691 |   748 | 2013 |         2 |        3 |            8 | 1103 | Повесть
  96 | book_192  |    536 |   145 | 2011 |         2 |        3 |            5 |  896 | Новелла
  99 | book_198  |    532 |  1398 | 2001 |         3 |        3 |            7 | 1010 | Новелла
 130 | book_260  |    878 |  1036 | 1991 |         2 |        3 |            4 |  423 | Роман
 139 | book_278  |    801 |   852 | 1993 |         3 |        3 |            4 | 1127 | Новелла
 143 | book_286  |    816 |  1447 | 2021 |         3 |        3 |            4 |  349 | Роман
 158 | book_316  |    710 |   464 | 1990 |         3 |        3 |            3 |  274 | Рассказ
 180 | book_360  |    794 |  1320 | 1996 |         5 |        3 |            4 |  392 | Поэма
 204 | book_408  |    568 |  1149 | 1993 |         3 |        3 |            7 |  368 | Рассказ
 219 | book_438  |    856 |  1341 | 2013 |         4 |        3 |            1 |  483 | Поэма
(10 строк)


var4=# select count(*) from books
var4-# inner join genres on books.genre_id = genres.id
var4-# where (amount > 500) and (genre_name = 'Мелодрама');
 count
-------
    95
(1 строка)
```

Рисунок 14 – Вывод книг жанра «мелодрама» с тиражом более 500 единиц

```
var4=# select books.* from books
var4-# inner join genres on books.genre_id = genres.id
var4-# inner join publishers on books.publisher_id = publishers.id
var4-# where (book_type = any(array['Рассказ', 'Новелла']))
var4-#  and (genre_name = any(array['Фантастика', 'Комедия']))
var4-#  and (publisher_name = any(array['Сибирь', 'Наука']))
var4-# limit 10;
 id  |        name         | amount | price | year | author_id | genre_id | publisher_id | size | book_type
-----+---------------------+--------+-------+------+-----------+----------+--------------+------+-----------
  90 | book_180            |    833 |    83 | 2011 |         3 |        7 |            3 |  280 | Рассказ
  15 | Превращение         |      4 |   141 | 2019 |         2 |        7 |            9 |   64 | Новелла
  39 | Путешествие в Индию |      5 |   885 | 1981 |         4 |        7 |            3 |  589 | Новелла
  49 | Гамлет              |     19 |  1602 | 1834 |         5 |        7 |            3 |  528 | Рассказ
 155 | book_310            |    462 |   702 | 1992 |         5 |        5 |            9 |  442 | Новелла
 177 | book_354            |    510 |   891 | 2020 |         1 |        7 |            3 |  632 | Рассказ
 205 | book_410            |    188 |   942 | 2008 |         1 |        5 |            9 |  720 | Рассказ
 257 | book_514            |    244 |  1299 | 1993 |         3 |        5 |            3 | 1135 | Рассказ
 262 | book_524            |    185 |   927 | 2017 |         2 |        7 |            3 |  922 | Рассказ
 264 | book_528            |    473 |  1123 | 1996 |         6 |        5 |            3 |  938 | Рассказ
(10 строк)


var4=# select count(*) from books
var4-# inner join genres on books.genre_id = genres.id
var4-# inner join publishers on books.publisher_id = publishers.id
var4-# where (book_type = any(array['Рассказ', 'Новелла']))
var4-#  and (genre_name = any(array['Фантастика', 'Комедия']))
var4-#  and (publisher_name = any(array['Сибирь', 'Наука']));
 count
-------
    35
(1 строка)
```

Рисунок 15 – Вывод рассказов и новелл жанров «фантастика» и «комедия» издательств «Сибирь» и «Наука».

Сохраним текущую базу данных в файл (рис. 16). Для этого необходимо запустить командную строку и выполнить команду pg_dump -U <имя пользователя> <название БД> > <путь сохраняемого файла>.

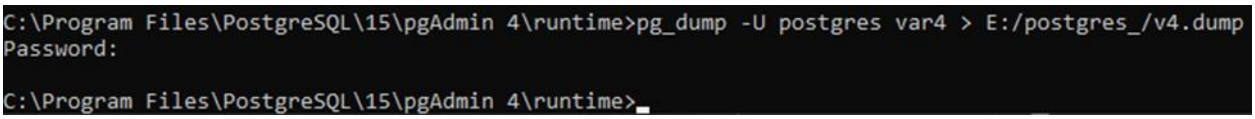


Рисунок 16 – Сохранение БД в файл.

При необходимости восстановить базу данных можно воспользоваться командой psql -U <имя пользователя> < <путь файла>.

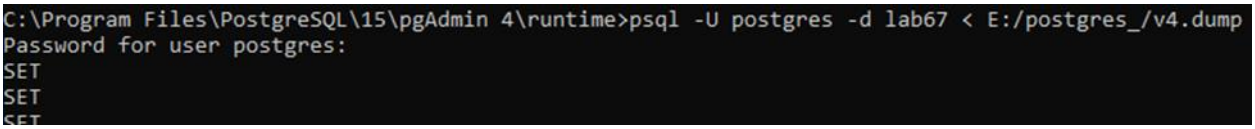


Рисунок 17 – Восстановление БД из файла.

Выводы. В ходе проделанной работы были изучены индексы, благодаря которым можно ускорить выполнение SELECT запроса. Индексы имеют различные типы, в данной работе были изучены типы BTREE и HASH и функциональные индексы. Выяснилось, что выполнение запросов с индексами действительно выполнялся быстрее, чем без них. Однако создание индексов требует времени и ресурсов, а также при изменении данных в таблице индексы необходимо пересоздавать. Поэтому использовать индексы следует в тех случаях, когда данные в таблицы изменяются редко или не изменяются вовсе, а также если столбец или функция часто используется внутри запросов.

Практическая часть

Рабочее задание: С помощью функции add_n(int) добавить в какую-либо из таблиц, созданных в лабораторной работе №1, 1000 записей. Выполнить команду *EXPLAIN ANALYSE select * from таблица where поле(условие)значение*. Проанализировать полученную после выполнения операции информацию.

Создать индекс типа BTREE для поля, по которому ранее осуществлялась выборка. Выполнить команду EXPLAIN ANALYSE с теми же условиями, что и без индекса, проанализировать полученную после выполнения операции информацию, сравнить результаты выполнения операции по каждому из условий выборки. Удалить индекс.

Создать индекс типа HASH для того же поля. Выполнить команду EXPLAIN ANALYSE с теми же условиями, что и ранее, проанализировать полученную после выполнения операции информацию, сравнить результаты выполнения операции по каждому из условий выборки. Удалить индекс.

Выполнить команду *EXPLAIN ANALYSE select * from таблица where upper(поле)(условие)значение*, условия те же, что и ранее, значение вводится большими буквами, поле должно быть типа char. По полю типа char той же таблицы создать функциональный индекс с функцией upper(поле). Выполнить команду

EXPLAIN ANALYSE с теми же условиями, что и ранее, проанализировать полученную после выполнения операции информацию, сравнить результаты выполнения операции по каждому из условий выборки с информацией, полученной при выборке по этому полю без индекса. Сохранить базу в файл для использования в следующих работах.

Вывести полную информацию о произведениях, в которых более 300 страниц. Найти все дороже 1000 р. заданного автора. Найти всю литературу заданного издательства, выпущенную за последние три месяца. Найти все книги, чей тираж более 500 экземпляров для жанра «мелодрама». Найти рассказы и новеллы жанров «фантастика» и «комедия» издательств «Сибирь» и «Наука».

Практическая работа № 7-8. Правила создания функций

Цель работы: Изучить правила создания функций. Приобрести практические навыки создания функций в среде PostgreSQL. Научиться проектировать БД в целом на основе поставленного задания с использованием всех полученных ранее базовых знаний и навыков

Теоретическая часть

1. Новые таблицы в БД

Создадим новые таблицы (рис. 1-5) и внесем правки в старые (рис. 6).

```
lab67=# create table storage (
lab67(# book_id int,
lab67(# amount int);
CREATE TABLE
```

Рисунок 1 – Добавление таблицы storage.

```
lab67=# create table orders (
lab67(# check_id int,
lab67(# customer_id int,
lab67(# order_status text,
lab67(# total_price int);
CREATE TABLE
lab67=# alter table orders add ord_date timestamp;
ALTER TABLE
```

Рисунок 2 - Добавление таблицы orders.

```
lab67=# create table customers (customer_id int);
CREATE TABLE
```

Рисунок 3– Добавление таблицы customers.

```
lab67=# create table checks (
lab67(# check_id int,
lab67(# book_id int,
lab67(# amount int);
CREATE TABLE
```

Рисунок 4 – Добавление таблицы checks.

```
lab67=# create table logs (
lab67(# n int,
lab67(# dat date,
lab67(# operation text);
CREATE TABLE
```

Рисунок 5 – Добавление таблицы logs.

```
lab67=# alter table books add age_group int;
ALTER TABLE
lab67=# \d books
                        Таблица "public.books"
   Столбец    |   Тип   | Правило сортировки | Допустимость NULL | По умолчанию
--------------+---------+--------------------+-------------------+--------------
 id           | integer |                    |                   |
 name         | text    |                    |                   |
 amount       | integer |                    |                   |
 price        | integer |                    |                   |
 year         | integer |                    |                   |
 author_id    | integer |                    |                   |
 genre_id     | integer |                    |                   |
 publisher_id | integer |                    |                   |
 size         | integer |                    |                   |
 book_type    | text    |                    |                   |
 age_group    | integer |                    |                   |
Ограничения-проверки:
    "genre_constraint" CHECK (genre_id = ANY (ARRAY[1, 3, 5, 6, 7]))
    "price_50min" CHECK (price >= 50)
    "year_2023max" CHECK (year <= 2023)
```

Рисунок 6 – Новый столбец в books.

В нашей БД в качестве таблиц-справочников можно отнести таблицу customers, checks, storage, а в качестве таблиц, использующие справочники – orders. Таблица logs используется в качестве таблицы-журнала.

2. Описание функций. Создадим функции (рис. 7-14):

- update_storage(book_id, amount) –добавление в таблицу storage *amount* книг с идентификатором *book_id*.
- fill_storage() – заполнение таблицы storage случайными книгами и со случайным количеством.
- new_customers(amount) – добавление в таблицу customers *amount* значений.
- make_order(customer_id) – создать заказ для покупателя с идентификатором *customer_id*.
- take_book(customer_id, book_id, amount) – добавление в заказ покупателя *customer_id* книг *book_id* в количестве *amount*.
- cancel_order(customer_id) – отменить заказ пользователю *customer_id*.
- show_order(customer_id) – показать заказ пользователя *customer_id*.
- remove(customer_id, book_id) – удалить из заказа пользователя *customer_id* книги с идентификатором *book_id*.

- confirm_order(customer_id) – подтвердить заказ пользователя *customer_id*.

```
lab67=# CREATE OR REPLACE FUNCTION update_storage(int, int) RETURNS char AS
lab67-# $$
lab67$# DECLARE
lab67$#   l int;
lab67$# BEGIN
lab67$#   IF NOT EXISTS (SELECT FROM books WHERE id = $1) THEN
lab67$#     RETURN 'Book with id = ' || $1 || ' not exists.';
lab67$#   END IF;
lab67$#   IF NOT EXISTS (SELECT FROM storage WHERE book_id = $1) THEN
lab67$#     INSERT INTO storage VALUES ($1, $2);
lab67$#   ELSE
lab67$#     UPDATE storage SET amount = amount + $2 WHERE book_id = $1;
lab67$#   END IF;
lab67$#   SELECT max(n) INTO l FROM logs;
lab67$#   INSERT INTO logs VALUES (l + 1, now(), 'update_storage(): added to book_id: ' || $1 || ', amount: ' || $2);

lab67$#   RETURN 'update_storage() done!';
lab67$# END;
lab67$# $$
lab67-# LANGUAGE 'plpgsql' SECURITY DEFINER;
CREATE FUNCTION
```

Рисунок 7 – Создание функции update_storage.

```
lab67=# CREATE OR REPLACE FUNCTION new_customers(int) RETURNS char AS
lab67-# $$
lab67$# DECLARE
lab67$#   t int; l int;
lab67$# BEGIN
lab67$#   SELECT max(customer_id) INTO t FROM customers;
lab67$#   IF ((SELECT count(*) from customers) = 0) THEN
lab67$#     t := 0;
lab67$#   END IF;
lab67$#   FOR k IN (t+1)..($1+t) LOOP
lab67$#     INSERT INTO customers VALUES (k);
lab67$#   END LOOP;
lab67$#   SELECT max(n) INTO l FROM logs;
lab67$#   INSERT INTO logs VALUES (l + 1, now(), 'new_customers(): added ' || $1 || ' elements');
lab67$#   RETURN 'new_customers done!';
lab67$# END;
lab67$# $$
lab67-# LANGUAGE 'plpgsql' SECURITY DEFINER;
CREATE FUNCTION
```

Рисунок 8 – Создание функции new_customers.

```
lab67=# CREATE OR REPLACE FUNCTION make_order(int) RETURNS char AS
lab67-# $$
lab67$# DECLARE
lab67$#   l int;
lab67$#   ch int;
lab67$#   ch2 int;
lab67$# BEGIN
lab67$#   IF EXISTS (SELECT check_id FROM orders WHERE (order_status = 'order') and (customer_id = $1)) THEN
lab67$#     RETURN 'Order for customer ' || $1 || ' already exists.';
lab67$#   END IF;
lab67$#   IF NOT EXISTS (SELECT * FROM customers WHERE customer_id = $1) THEN
lab67$#     RETURN 'Customer ' || $1 || ' not in database';
lab67$#   END IF;
lab67$#   SELECT max(check_id) INTO ch FROM checks;
lab67$#   SELECT max(check_id) INTO ch2 FROM orders;
lab67$#   IF ch IS NULL THEN
lab67$#     ch := 0;
lab67$#   END IF;
lab67$#   IF (ch2 > ch) THEN
lab67$#     ch := ch2;
lab67$#   END IF;
lab67$#   INSERT INTO orders VALUES (ch + 1, $1, 'order', 0);
lab67$#   SELECT max(n) INTO l FROM logs;
lab67$#   INSERT INTO logs VALUES (l + 1, now(), 'make_order(): order: ' || ch + 1 || ', customer: ' || $1);
lab67$#   RETURN 'make_order() done!';
lab67$# END;
lab67$# $$
lab67-# LANGUAGE 'plpgsql' SECURITY DEFINER;
CREATE FUNCTION
```

Рисунок 9 – Создание функции make_order.

```
lab67=# CREATE OR REPLACE FUNCTION take_book(int, int, int) RETURNS char AS
lab67-# $$
lab67$# DECLARE
lab67$#   l int;
lab67$#   ord int;
lab67$# BEGIN
lab67$#   SELECT check_id INTO ord FROM orders WHERE (order_status = 'order') and (customer_id = $1);
lab67$#   IF ord IS NULL THEN
lab67$#     RETURN ('Customer ' || $1 || E' didn\'t made order');
lab67$#   END IF;
lab67$#   IF NOT EXISTS (SELECT * FROM storage WHERE book_id = $2) THEN
lab67$#     RETURN ('Book ' || $2 || ' not exists in storage');
lab67$#   END IF;
lab67$#   IF EXISTS (SELECT check_id FROM checks WHERE (check_id = ord) AND (book_id = $2)) THEN
lab67$#     UPDATE checks SET amount = amount + $3 WHERE (check_id = ord) AND (book_id = $2);
lab67$#   ELSE
lab67$#     INSERT INTO checks VALUES (ord, $2, $3);
lab67$#   END IF;
lab67$#
lab67$#   SELECT max(n) INTO l FROM logs;
lab67$#   INSERT INTO logs VALUES (l + 1, now(), 'take_book(): order: ' || ord || ', book_id: ' || $2 || ', amount: '
 || $3);
lab67$#   RETURN 'take_book() done!';
lab67$# END;
lab67$# $$
lab67-# LANGUAGE 'plpgsql' SECURITY DEFINER;
CREATE FUNCTION
```

Рисунок 10 – Создание функции take_book.

```
lab67=# CREATE OR REPLACE FUNCTION cancel_order(int) RETURNS char AS
lab67-# $$
lab67$# DECLARE
lab67$#   l int;
lab67$#   ord int;
lab67$#   cnt int;
lab67$# BEGIN
lab67$#   SELECT check_id INTO ord FROM orders WHERE (order_status = 'order') and (customer_id = $1);
lab67$#   IF ord IS NULL THEN
lab67$#     RETURN ('Customer ' || $1 || E' don\'t have order.');
lab67$#   END IF;
lab67$#   SELECT count(*) INTO cnt FROM checks WHERE check_id = ord;
lab67$#   DELETE FROM checks WHERE check_id = ord;
lab67$#   DELETE FROM orders WHERE check_id = ord;
lab67$#
lab67$#   SELECT max(n) INTO l FROM logs;
lab67$#   INSERT INTO logs VALUES (l + 1, now(), 'cancel_order(): order: ' || ord || ', removed ' || cnt || ' strings
');
lab67$#   RETURN 'cancel_order() done!';
lab67$# END;
lab67$# $$
lab67-# LANGUAGE 'plpgsql' SECURITY DEFINER;
CREATE FUNCTION
```

Рисунок 11 – Создание функции cancel_order.

```
lab67=# CREATE OR REPLACE FUNCTION show_order(int) RETURNS table (book_id int, amount int) AS
lab67-# $$
lab67$# BEGIN
lab67$#   RETURN QUERY (
lab67$#   SELECT checks.book_id, checks.amount FROM checks WHERE checks.check_id = (
lab67$#     SELECT orders.check_id FROM orders WHERE
lab67$#         (orders.order_status = 'order') and (orders.customer_id = $1)
lab67$#     )
lab67$#   );
lab67$# END;
lab67$# $$
lab67-# LANGUAGE 'plpgsql' SECURITY DEFINER;
CREATE FUNCTION
```

Рисунок 12 – Создание функции show_order.

```
lab67=# CREATE OR REPLACE FUNCTION remove(int, int) RETURNS char AS
lab67-# $$
lab67$# DECLARE
lab67$#   l int;
lab67$#   ord int;
lab67$#   cnt int;
lab67$# BEGIN
lab67$#   SELECT check_id INTO ord FROM orders WHERE (order_status = 'order') and (customer_id = $1);
lab67$#   IF ord IS NULL THEN
lab67$#     RETURN ('Customer ' || $1 || E' don\'t have order.');
lab67$#   END IF;
lab67$#   DELETE FROM checks WHERE (check_id = ord) and (book_id = $2);
lab67$#
lab67$#   SELECT max(n) INTO l FROM logs;
lab67$#   INSERT INTO logs VALUES (l + 1, now(), 'remove(): order: ' || ord
lab67$#     || ', removed: ' || $2);
lab67$#   RETURN 'remove() done!';
lab67$# END;
lab67$# $$
lab67-# LANGUAGE 'plpgsql' SECURITY DEFINER;
CREATE FUNCTION
```

Рисунок 13 –. Создание функции remove.

```
lab67=# CREATE OR REPLACE FUNCTION confirm_order(int) RETURNS char AS
lab67-# $$
lab67$# DECLARE
lab67$#   l int;
lab67$#   ord int;
lab67$#   t_pr int;
lab67$#   cnt int;
lab67$# BEGIN
lab67$#   SELECT check_id INTO ord FROM orders WHERE (order_status = 'order') and (customer_id = $1);
lab67$#   IF ord IS NULL THEN
lab67$#     RETURN ('Customer ' || $1 || E' don\'t have order.');
lab67$#   END IF;
lab67$#
lab67$#   SELECT count(*) INTO cnt FROM storage
lab67$#     INNER JOIN checks ON storage.book_id = checks.book_id
lab67$#     WHERE (storage.amount < checks.amount) and (check_id = ord);
lab67$#   IF (cnt > 0) THEN
lab67$#     RETURN 'Order failed: ' || cnt || ' books have more than available amount';
lab67$#   END IF;
lab67$#
lab67$#   SELECT sum(amount * price) INTO t_pr FROM (
lab67$#     SELECT books.price, checks.amount FROM books
lab67$#     INNER JOIN checks ON checks.book_id = books.id
lab67$#         WHERE checks.check_id = ord) x;
lab67$#
lab67$#   UPDATE storage SET amount = storage.amount - checks.amount FROM checks
lab67$#     WHERE storage.book_id = checks.book_id;
lab67$#   UPDATE orders SET order_status = 'done' WHERE check_id = ord;
lab67$#   UPDATE orders SET total_price = t_pr WHERE check_id = ord;
lab67$#   UPDATE orders SET ord_date = now() WHERE check_id = ord;
lab67$#
lab67$#   SELECT max(n) INTO l FROM logs;
lab67$#   INSERT INTO logs VALUES (l + 1, now(), 'confirm_order(): order: ' || ord
lab67$#     || ', total price: ' || t_pr);
lab67$#   RETURN 'confirm_order() done! ' || t_pr;
lab67$# END;
lab67$# $$
lab67-# LANGUAGE 'plpgsql' SECURITY DEFINER;
CREATE FUNCTION
```

Рисунок 14 – Создание функции confirm_order.

Практическая часть

Рабочее задание: Ознакомиться с теоретическими сведениями о возможностях создания пользовательских функций в PostgreSQL. Разработать БД в соответствии с индивидуальным заданием. Создать функции, реализующие интерфейс для работы с базой данных. Проверить работоспособность функций путем выполнения этих функций с параметрами, обеспечивающими как успешное выполнение функции, так и невыполнение функции. Обязательные требования к БД:

1. Наличие таблиц-справочников и таблиц, использующих справочники. Предусмотреть сохранение ссылочной целостности для таблиц, использующих таблицы-справочники.
2. Предусмотреть следующие роли:

a. оператор БД (пополнение справочников);

b. пользователь БД (основная работа с БД, с ограничениями для некоторого вида операций);
c. аналитик (разрешено выполнение запросов и функций, не изменяющих данные в БД);
d. администратор БД (просмотр протокола операций, любые изменения БД);
3. Действия, изменяющие БД пользователем с любой ролью, протоколируются в таблице-журнале операций.
4. Для всех запросов необходимо создать индексы (для гарантированного использования индексов можно использовать отключение параметра enable_seqscan в текущей сессии).

База данных сети книжных магазинов. Должна содержать следующие данные: текущие складские запасы печатной продукции, информацию о заказах и продажах. Предусмотреть анализ следующих показателей: наиболее часто заказываемые книги, средний чек по разным группам товаров, средний чек в зависимости от сезона (например, лето, осень, зима, весна), рейтинг популярности книг для разных возрастных групп (например, дети, подростковый возраст, студенты, пенсионеры).

Практическая работа № 9. Перегружаемые функции и триггеры в PostgreSQL

Цель работы: Изучение перегружаемых функций и триггеров в PostgreSQL. Изучить синтаксис команд. Получение навыков работы с триггерами.

Теоретическая часть

Коды команд. Для следующей работы обновим таблицу books (рис. 1).

```
lab8=*# alter table books drop constraint genre_constraint;
ALTER TABLE
lab8=*# alter table books add constraint genre_constraint check (genre_id = ANY (ARRAY[0, 1, 3, 5, 6, 7]));
ALTER TABLE
lab8=*# \d books
```

Рисунок 1 – Обновления ограничения в таблице books.

Добавим в базу данных перегруженную функцию, которая добавляет книги в таблицу books. В зависимости от аргументов, перегруженная функция будет делать следующее:

- add_book(text) — вставка только названия книги (рис. 2);
- add_book(text, int) — вставка названия книги, идентификатора автора (рис. 3);
- add_book(int) — вставка нескольких книг, названия создаются автоматически (рис. 4).

```
lab8=# CREATE OR REPLACE FUNCTION add_book(fname text) RETURNS char AS
lab8-# $$
lab8$# DECLARE
lab8$#   m_id int;
lab8$# BEGIN
lab8$#   SELECT max(id) INTO m_id FROM books;
lab8$#   INSERT INTO books VALUES (m_id+1,
lab8$#   fname, 0, 50, 1900,
lab8$#   0, 0, 0,
lab8$#   0, '', 0);
lab8$#   RETURN 'Used function: add_book(text)';
lab8$# END;
lab8$# $$
lab8-# LANGUAGE 'plpgsql' SECURITY DEFINER;
CREATE FUNCTION
```

Рисунок 2 – Перегруженная функция (только название).

```
lab8=# CREATE FUNCTION add_book(fname text, a_id int) RETURNS char AS
lab8-# $$
lab8$# DECLARE
lab8$#   m_id int;
lab8$# BEGIN
lab8$#   SELECT max(id) INTO m_id FROM books;
lab8$#   INSERT INTO books VALUES (m_id+1,
lab8$#   fname, 0, 50, 1900,
lab8$#   a_id, 0, 0,
lab8$#   0, '', 0);
lab8$#   RETURN 'Used function: add_book(text, author_id)';
lab8$# END;
lab8$# $$
lab8-# LANGUAGE 'plpgsql' SECURITY DEFINER;
CREATE FUNCTION
```

Рисунок 3 – Перегруженная функция (название + автор).

```
lab8=# CREATE OR REPLACE FUNCTION add_book(famount int) RETURNS char AS
lab8-# $$
lab8$# DECLARE
lab8$#   m_id int;
lab8$# BEGIN
lab8$#   SELECT max(id) INTO m_id FROM books;
lab8$#   FOR k IN (m_id + 1)..(m_id + $1) LOOP
lab8$#     INSERT INTO books VALUES (k,
lab8$#     'default_book_' || k, 0, 50, 1900,
lab8$#     0, 0, 0,
lab8$#     0, '', 0);
lab8$#   END LOOP;
lab8$#   RETURN 'Added ' || famount || ' books!';
lab8$# END;
lab8$# $$
lab8-# LANGUAGE 'plpgsql' SECURITY DEFINER;
CREATE FUNCTION
```

Рисунок 4 – Перегруженная функция (несколько книг).

Добавим функцию для триггера, которая перед удалением данных из таблицы authors, будет обнулять идентификатор автора в таблице books (рис. 5).

```
lab8=# CREATE OR REPLACE FUNCTION rem_tr() RETURNS trigger AS
lab8-# $$
lab8$# BEGIN
lab8$#   UPDATE books SET author_id = 0 WHERE author_id = OLD.id;
lab8$#   RETURN OLD;
lab8$# END;
lab8$# $$
lab8-# LANGUAGE plpgsql;
CREATE FUNCTION
```

Рисунок 5 – Добавление функции для триггера.

Добавим триггер на удаление записей из таблицы authors (рис. 6).

```
lab8=# CREATE TRIGGER trig1
lab8-# BEFORE DELETE ON authors FOR EACH ROW
lab8-# EXECUTE PROCEDURE rem_tr();
CREATE TRIGGER
```

Рисунок 6 – Добавление триггера на удаление записей.

Проверим работу триггера. Для этого из нашей базы данных удалим автора с идентификатором 10 (рис. 7).

```
lab8=*# select id, name, amount, author_id from books where author_id = 10 limit 5;
 id |             name             | amount | author_id
----+------------------------------+--------+-----------
 17 | Американская трагедия. Том 1 |     23 |        10
 25 | Клуб самоубийц               |     94 |        10
 27 | Хорошие жены                 |     10 |        10
(3 строки)


lab8=*# savepoint s2;
SAVEPOINT
lab8=*# delete from authors where id = 10;
DELETE 1
lab8=*# select id, name, amount, author_id from books where author_id = 0 limit 5;
 id |             name             | amount | author_id
----+------------------------------+--------+-----------
 17 | Американская трагедия. Том 1 |     23 |         0
 25 | Клуб самоубийц               |     94 |         0
 27 | Хорошие жены                 |     10 |         0
(3 строки)
```

Рисунок 7 – Пример работы триггера на удаление записи.

Добавим функцию для триггера (рис. 8), которая перед добавлением данных в таблицу authors, будет проверять добавляемые значения и изменять, если не указано имя автора или город автора. Также перед добавлением автора, обновляются значения в таблице books, если идентификатор автора равен нулю.

```
lab8=# CREATE OR REPLACE FUNCTION ins_tr() RETURNS trigger AS
lab8-# $$
lab8$# DECLARE
lab8$#   samara int;
lab8$# BEGIN
lab8$#   SELECT id INTO samara FROM cities WHERE city_name = 'Самара';
lab8$#   IF NEW.author IS NULL THEN
lab8$#     NEW.author := 'Avtor' || NEW.id;
lab8$#   END IF;
lab8$#   IF NEW.city_id IS NULL THEN
lab8$#     NEW.city_id := samara;
lab8$#   END IF;
lab8$#   UPDATE books SET author_id = NEW.id WHERE author_id = 0;
lab8$#   IF NEW.city_id = samara THEN
lab8$#     UPDATE books SET amount = 50 WHERE amount < 50;
lab8$#   ELSE
lab8$#     UPDATE books SET amount = 0 WHERE amount < 50;
lab8$#   END IF;
lab8$#   RETURN NEW;
lab8$# END;
lab8$# $$
lab8-# LANGUAGE plpgsql;
CREATE FUNCTION
```

Рисунок 8 – Добавление функции для второго триггера.

Добавим триггер на добавление записей в таблицу authors (рис. 9).

```
lab8=# CREATE TRIGGER trig2
lab8-# BEFORE INSERT ON authors FOR EACH ROW
lab8-# EXECUTE PROCEDURE ins_tr();
CREATE TRIGGER
```

Рисунок 9 – Добавление триггера на добавление записей.

Проверим работу триггера. Для этого добавим в таблицу authors значение без названия автора и города и проверим значения в таблице books, идентификатор автора которых был приравнен к нулю (рис. 10).

```
lab8=*# insert into authors values(50);
INSERT 0 1
lab8=*# select * from authors where id = 50;
 id | author  | city_id
----+---------+---------
 50 | Avtor50 |       9
(1 строка)


lab8=*# select id, name, amount, author_id from books where author_id = 50 limit 5;
 id |             name             | amount | author_id
----+------------------------------+--------+-----------
 25 | Клуб самоубийц               |     94 |        50
 17 | Американская трагедия. Том 1 |     50 |        50
 27 | Хорошие жены                 |     50 |        50
(3 строки)
```

Рисунок 10 – Пример работы второго триггера.

Как видно из рис. 10, идентификатор города был приравнен к 9, который соответствовал городу «Самара». Также был изменен тираж книг, у которых ранее стояло значение, менее 50. Воспользуемся ранее созданными перегруженными функциями и добавим значения в таблицу books (рис. 11).

```
lab8=*# select add_book(4);
    add_book
---------------
 Added 4 books!
(1 строка)


lab8=*# select add_book('unnamed_book');
           add_book
-------------------------------
 Used function: add_book(text)
(1 строка)


lab8=*# select id, name, amount, author_id from books where author_id = 0 limit 5;
  id  |       name        | amount | author_id
------+-------------------+--------+-----------
 1055 | default_book_1055 |      0 |         0
 1056 | default_book_1056 |      0 |         0
 1057 | default_book_1057 |      0 |         0
 1058 | default_book_1058 |      0 |         0
 1059 | unnamed_book      |      0 |         0
(5 строк)
```

Рисунок 11 – Использование перегруженных функций дли добавления книг.

Приведем второй пример использования триггера на добавления значений, когда город автора отличный от Самары (рис. 12).

```
lab8=*# insert into authors values (44, 'cwjl', 3);
INSERT 0 1
lab8=*# select id, name, amount, author_id from books where author_id = 44 limit 5;
  id  |       name        | amount | author_id
------+-------------------+--------+-----------
 1055 | default_book_1055 |      0 |        44
 1056 | default_book_1056 |      0 |        44
 1057 | default_book_1057 |      0 |        44
 1058 | default_book_1058 |      0 |        44
 1059 | unnamed_book      |      0 |        44
(5 строк)
```

Рисунок 12 – Второй пример работы триггера на добавление записей.

Из рис. 12 видно, что тираж не был приравнен к 50, поскольку город автора не является «Самара».

На рис. 13-14 показаны удаление триггеров.

```
lab8=# DROP TRIGGER trig1 ON authors;
DROP TRIGGER
```

Рисунок 13 – Удаление триггера на удаление записей.

```
lab8=# DROP TRIGGER trig2 ON authors;
DROP TRIGGER
```

Рисунок 14 – Удаление триггера на добавление записей.

Выводы. В ходе проделанной работы были изучены перегруженные функции и работа триггеров. На основе таблиц с авторами и книгами были сделаны два триггера на добавление и удаление записей из таблицы с авторами. Оба триггера при выполнения соответствующего запроса изменяли обе таблицы.

Практическая часть

Рабочее задание: Ознакомиться с теоретическими сведениями о возможностях создания перегружаемых функций и триггеров в PostgreSQL. Создать перегружаемые функции и триггеры. Продемонстрировать работу триггеров на примерах вставки и удаления записей из таблицы. Если в базе нет подходящих данных, то добавить подходящие данные. Удалить триггер. По заданию преподавателя создать триггер и проверить его работоспособность. Просмотреть и проанализировать полученную в результате выполнения операций информацию.

Триггер выполняется перед удалением записи из таблицы автора. Триггер проверяет наличие в другой таблице записей, относящихся к удаляемому автору, и, если такие записи есть, удаляет их. Триггер выполняется перед вставкой новой записи в таблицу авторов. Триггер проверяет значения, которые должна содержать новая запись и может их изменить:

- если не указано имя автора – оно генерируется по схеме – Avtor + уникальный номер из последовательности;
- если не указан город автора – ставится значение по умолчанию – “Samara”;
- если не указан тираж издания или тираж <=50 – устанавливается тираж, равный 50 для автора из города “Samara” и 0 для всех остальных.

Перечень использованных информационных источников

1. Коннолли, Т. Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Коннолли. - М.: Вильямс И.Д., 2021. - 1440 с.

2. Лукин, В.Н. Введение в проектирование баз данных / В.Н. Лукин. - М.: Вузовская книга, 2022. - 144 с.

3. Пирогов, В. Информационные системы и базы данных: организация и проектирование: Учебное пособие / В. Пирогов. - СПб.: BHV, 2020. - 528 с.

4. Малыхина, М.П. Базы данных: основы, проектирование, использование / М.П. Малыхина. - СПб.: BHV, 2022. - 528 с.