

1 Введение

Настоящее пособие содержит методические указания к выполнению лабораторных работ по дисциплине «Базы данных». Пособие адресовано студентам кафедры АСУ ТУСУР.

Цель лабораторного практикума – освоить основные приёмы реализации реляционных баз данных и компонентов приложений.

Практикум состоит из четырёх лабораторных работ (ЛР). На весь цикл студент получает одно индивидуальное задание (проект), содержащее описание структуры базы данных и функций приложения пользователя. В ходе выполнения ЛР этот проект должен быть реализован. Работы выполняются с использованием реляционной СУБД PostgreSQL. Индивидуальные задания выдаются преподавателем.

В конце каждого занятия следует скопировать текущее состояние проекта (.dump-файл) в директорию на сетевом диске. Подробнее о том как получить .dump-файл сказано в описании 1 лабораторной работы.

Отчёт о лабораторной работе необходимо сдать преподавателю до начала следующего занятия. Отчёт должен быть оформлен в соответствии с требованиями стандарта «Стандарт ОС ТУСУР 01-2021 по направлениям подготовки и специальностям технического профиля». На титульном листе отчёта обязательно должна быть личная подпись автора и дата представления. Ниже описаны отдельные лабораторные работы цикла и сформулированы требования к содержанию отчётов.

2 Общие сведения о СУБД PostgreSQL

Принцип работы

В книге «Postgres: Первое знакомство» [1] для PostgreSQL дано следующее определение:

«PostgreSQL — наиболее полнофункциональная, свободно распространяемая СУБД с открытым кодом. Разработанная в академической среде, за долгую историю сплотившая вокруг себя широкое сообщество разработчиков, эта СУБД обладает всеми возможностями, необходимыми большинству заказчиков.»

С этим определением сложно не согласиться, действительно, на сегодняшний день, PostgreSQL имеет очень большой набор встроенных возможностей, а так же некоторое количество дополнительно подключаемых расширений, которые позволяют использовать СУБД в проектах разной направленности и сложности.

Заметка

Кстати, произносить название СУБД следует как «постгрес-ку-эль» или просто «постгрес», но только не «постгре».

Работа с PostgreSQL подразумевает подключение к *серверу PostgreSQL*. Сам сервер при этом является особой программой, которая запущена в фоновом (т.е. неинтерактивном) режиме. Во время своей работы сервер всё время находится в режиме ожидания. Он ожидает запросов на подключение от *клиентов*, которые тоже являются программами. Программа *клиент* может управляться человеком, например, через графическую оболочку, а может выполнять действия согласно некоторому алгоритму, который реализован в её исходном коде. В любом случае, получив запрос на подключение, сервер сначала проводит процедуру аутентификации подключаемого клиента и только после этого начинает принимать и выполнять управляющие команды.

Команды, которые можно отправить на сервер PostgreSQL представляют собой запросы на языке SQL. Получив такой запрос, сервер его анализирует, проверяет имеет ли клиент права на выполнение подобного запроса, проводит оптимизацию и, наконец, выполняет, возвращая результат запроса обратно клиенту. Так как язык SQL является универсальным для всех реляционных СУБД, команды-запросы, сформулированные для сервера PostgreSQL будут мало отличаться от подобных же команд, построенных для другой СУБД, например MySQL.

Однако кроме SQL команд, сервер PostgreSQL может принимать некоторый набор инструкций, действительных только для СУБД PostgreSQL. В большинстве своём это команды каким-то образом управляющие поведением сервера. Подробнее о таких командах Вы узнаете в тексте описания 1 лабораторной работы.

Наглядно схема взаимодействия клиентской программы и сервера СУБД представлена на рисунке 2.1 на следующей странице.

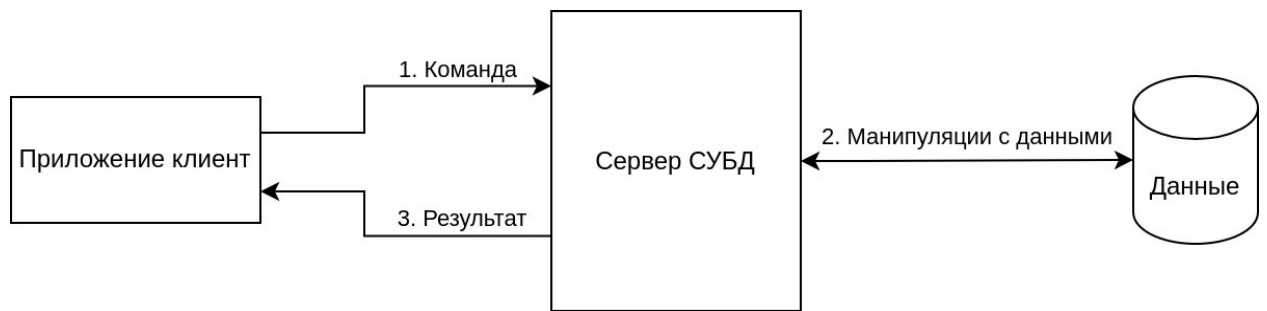


Рисунок 2.1 - Схема взаимодействия клиентского приложения с сервером СУБД

Структура данных

Любой сервер СУБД управляет в первую очередь данными. Организация этих данных в разных СУБД выполняется в целом одинаково, но есть и различия. Вот основные структурные единицы представления данных в PostgreSQL:

- Кластер баз данных – совокупность всех баз данных, которые имеются на сервере.
- База данных – совокупность именованных схем базы данных.
- Схема базы данных – совокупность логически объединенных отношений, типов данных, функций и операторов.
- Отношение – таблица базы данных, индекс, представление, материализованное представление.

В более наглядном виде структура данных сервера PostgreSQL представлена на рисунке 2.2.

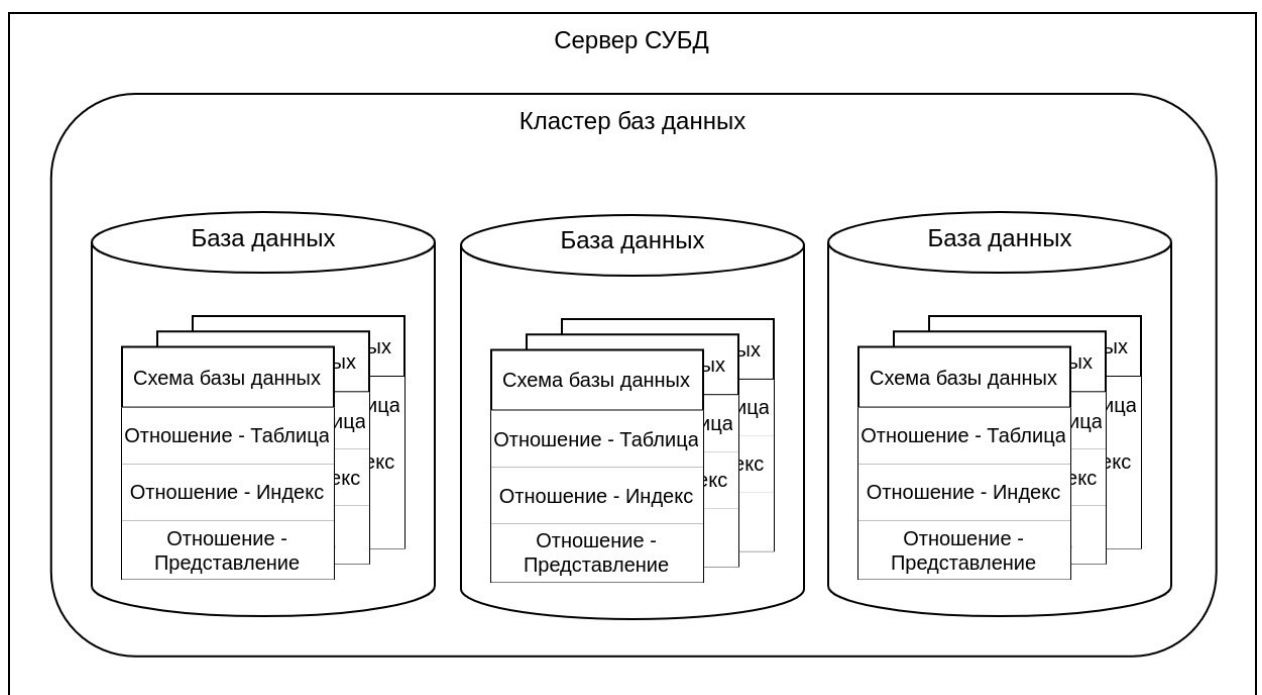


Рисунок 2.1 - Структура данных СУБД PostgreSQL

3 Предварительная подготовка

Студенты, выполняющие лабораторные работы с использованием компьютеров кафедры, должны убедиться, что на их рабочем месте установлен сервер PostgreSQL, а так же необходимый к нему инструментарий. Для этого достаточно убедиться в наличии приложения «SQL Shell (psql)» в меню «Пуск». Если такого приложения нет – следует обратиться к преподавателю.

Те же, кто хочет выполнять задания на своих устройствах должны выполнить установку PostgreSQL. Установщик можно найти на официальном сайте:

<https://www.postgresql.org/download/>

Подробная инструкция по установке представлена в книге «Postgres: Первое знакомство»[1], которая находится в открытом доступе на сайте:

<https://www.postgrespro.ru/education/books/introbook>

В случае возникновения проблем, можете обращаться к преподавателю.

4 Описания лабораторных работ

Лабораторная работа №1 «Работа с утилитой psql»

Цели:

- ознакомиться с принципами работы консольной утилиты psql;
- научиться создавать макеты таблиц с использованием команд SQL.

Общие сведения о таблицах и связях

Что такое таблица

Определение

Таблицей в большинстве реляционных СУБД называется объект базы данных, представляющий собой совокупность строк и столбцов, на пересечении которых размещаются значения простых типов данных.

При этом:

- Все значения столбца принадлежат одному домену, т.е. имеют один тип данных и удовлетворяют одному набору ограничений.

- Каждая таблица имеет уникальное имя и сохраняемое в системном каталоге определение (макет).
- Макет можно понимать как список имён столбцов таблицы (полей) с указанием их свойств.
- Данные не упорядочены (в частности, нельзя полагаться на то, что строки хранятся в порядке их добавления в таблицу).

Для чего предназначена таблица

В таблице сохраняются сведения, относящиеся к одной и только одной теме (объекту или факту). Например:

- таблица **СТУДЕНТ** содержит личные данные студентов;
- таблица **ДИСЦИПЛИНА** содержит сведения об учебных дисциплинах, изучаемых студентами;
- таблица **УСПЕВАЕМОСТЬ** содержит сведения об оценках, полученных студентами на экзаменах (фактах сдачи экзаменов) и т.п.

Замечание


Таблица **СТУДЕНТ** может содержать поля *Номер_студбилета*, *Фамилия*, *Имя*, *Отчество*, *Группа* и не может содержать поле *Оценка*. Это поле характеризует успеваемость студента и должно быть определено в таблице **УСПЕВАЕМОСТЬ**.

Что такое первичный ключ таблицы

Определение

Для каждой таблицы должен быть определён первичный ключ. Первичный ключ – это поле (группа полей), значения которого идентифицируют строки таблицы. Первичный ключ может быть простым и составным.

На рисунке 4.1 представлены примеры простого и составного первичных ключей.



Первичный ключ

- Идентифицирует строки таблицы
- Каждое значение ключа - **уникально** для данной таблицы
- **Не может** принимать значение NULL

Простой ключ

Номер_студбилета	ФИО	Адрес
434312	Федоров А.А.	Ленина, 21
437M2	Васильев Г.Ю.	Пушкина, 19
438M17	Иванов И.И.	Пушкина, 19

Любого студента можно однозначно определить по номеру его студенческого билета.

Составной ключ

Номер_студбилета	Аббревиатура дисциплины	Дата	Оценка
438M17	СиАОД	14.01.2020	отлично
438M17	ООП	17.01.2020	хорошо
434312	СиАОД	23.01.2020	отлично

Результаты сдачи любого экзамена можно однозначно определить по номеру студенческого билета и названию дисциплины.

Рисунок 4.1 - Первичный ключ

Первичный ключ обладает тремя важными свойствами:

1. Ни при каких условиях таблица не может содержать двух строк с одинаковыми значениями первичного ключа – свойство уникальности.
2. Составной первичный ключ не содержит подмножества полей, обладающего свойством уникальности – свойство избыточности.
3. Первичный ключ не может принимать неопределённые (NULL) значения.

Замечание

Первичным ключом **нельзя** объявить произвольное подмножество полей таблицы.

Заключение о наличии или отсутствии свойств первичного ключа у некоторого подмножества полей можно сделать только исходя из смысла данных, сохраняемых в таблице.

Примеры правильного и неправильного выбора первичного ключа можно увидеть на рисунках 4.2 и 4.3 (на следующей странице).

Примеры



Номер_студбилета	Номер_груп	ФИО	Адрес
434312	434-3	Федоров А.А.	Ленина, 21
434311	434-3	Петров Ю.А.	Пушкина, 19
434310	434-3	Скворцова Е.В.	Гагарина, 2

Поле “Номер_группы” не может быть первичным ключом, так его значения повторяются для студентов, зачисленных в одну группу.



Номер_студбилета	Номер_груп	ФИО	Адрес
434312	434-3	Федоров А.А.	Ленина, 21
434311	434-3	Петров Ю.А.	Пушкина, 19
434310	434-3	Скворцова Е.В.	Гагарина, 2

Значения поля “Номер_студбилета” уникальны для всей таблицы “Студент”. Это позволяет назначить данное поле первичным ключом таблицы.

Рисунок 4.2 - Пример назначения первичных ключей

Примеры



Номер_студбилета	Номер_груп	ФИО	Адрес
434312	434-3	Федоров А.А.	Ленина, 21
434311	434-3	Петров Ю.А.	Пушкина, 19
434310	434-3	Скворцова Е.В.	Гагарина, 2



Номер_студбилета	Номер_груп	ФИО	Адрес
434312	434-3	Федоров А.А.	Ленина, 21
434311	434-3	Петров Ю.А.	Пушкина, 19
434310	434-3	Скворцова Е.В.	Гагарина, 2

Пары полей “Номер_студбилета”, “Номер_группы” и “Номер_студбилета”, “ФИО” нельзя назначить первичными ключами, так как поле “Номер_студбилета” в каждой паре само обладает свойством уникальности.

Рисунок 4.3 - Пример назначения составных первичных ключей

Для чего нужен первичный ключ

Поскольку значения первичных ключей идентифицируют строки таблицы, их можно использовать:

- Для быстрого поиска отдельных строк.
- Для организации связей таблиц.

Заметка

Для ускорения поиска записей все СУБД позволяют создавать индексы – особые объекты схемы базы данных.

СУБД PostgreSQL может предложить следующие типы индексов: В-деревья, Hash, GiST, SP-GiST, GIN, RUM, BRIN, Bloom. Подробнее о каждом типе индекса смотрите в книге «PostgreSQL: первое знакомство» [1].

Связи таблиц и внешние ключи

Определение

Для организации связей в состав полей таблицы-потомка включается дубликат первичного ключа родителя. Он называется здесь *внешним ключом*.

Значения внешнего ключа – это ссылки на строки родительской таблицы, содержащие такие же значения первичного ключа. Ссылки реализуются как неуникальные индексы по внешнему ключу. Эти индексы хранят информацию о связи, т.е. соответствии строк различных таблиц.

На рисунке 4.4 показаны строки из двух таблиц – «Студент» и «Успеваемость». Видно, что поле «Номер_студбилета» таблицы «Успеваемость» ссылается на такое же поле родительской таблицы.

Таблица родитель - «Студент»

Номер_студбилета	Номер_группы	ФИО	Адрес
434312	434-3	Федоров А.А.	Ленина, 21

Таблица потомок - «Успеваемость»

Номер_студбилета	Аббревиатура дисциплины	Дата	Оценка
434312	ООП	15.01.2020	хорошо
434312	СиАОД	23.01.2020	отлично

Рисунок 4.4 - Строки из таблиц «Студент» и «Успеваемость»

Ссылочная целостность

Определение

Значения внешнего ключа в строках таблицы-потомка **не могут** быть произвольными. Они должны принадлежать множеству значений родительского ключа в существующих строках родительской таблицы. Это условие называется требованием *ссылочной целостности*.

Далее на рисунке 4.5 продемонстрирован пример нарушения ссылочной целостности. Строка таблицы «Успеваемость» ссылается на несуществующий номер студенческого билета.

Номер_студбилета
434310
434311
434312

Номер_студбилета	Аббревиатура дисциплины	Дата	Оценка
434310	СиАОД	23.01.2020	хорошо
434311	ЭВМиПУ	13.01.2020	удовлетворительно
434312	ООП	15.01.2020	хорошо
<u>434322</u>	СиАОД	23.01.2020	отлично

Последняя строка таблицы “Успеваемость” нарушает требование ссылочной целостности, так как в таблице “Студент” нет строки со значением ключа “434322”.

Рисунок 4.5 - Нарушение ссылочной целостности

Правила ссылочной целостности

Замечание

Одна из главных задач СУБД – поддерживать ссылочную целостность при выполнении операций обновления данных. Она будет это делать, если для каждой пары родитель-потомок определены правила совместного обновления множеств значений первичного и внешнего ключей.

Ссылочная целостность может быть нарушена в следующих случаях:

- При попытке добавления строки в таблицу-потомок.
- При попытке изменения значения внешнего ключа в существующей строке таблицы-потомка.

- При попытке удаления строки из родительской таблицы.
- При попытке изменения значения первичного ключа в существующей строке родительской таблицы.

В двух первых случаях СУБД должна проверить, принадлежит ли новое значение внешнего ключа множеству существующих значений родительского ключа. Если да, то обновление таблицы-потомка будет выполнено, иначе – отвергнуто.

В двух последних случаях вариантов правил обновления гораздо больше. Они определяются требованиями пользователя и должны быть реализованы разработчиком БД в виде специальных программ – триггеров ссылочной целостности.

Начало работы

В первой лабораторной работе Вы будете взаимодействовать с сервером СУБД, используя стандартный терминальный клиент - `psql`.

Отсутствие графического интерфейса может показаться неудобным, однако `psql` является стандартным приложением, использовать которое можно в любой операционной системе (ОС) и в любой момент. Таким образом, имея навыки работы с `psql`, Вы сможете взаимодействовать с Вашей базой данных даже если работаете в незнакомой Вам ОС или когда все графические средства оказались недоступными.

Запуск `psql`

Замечание

Запуск `psql` и работа с сервером СУБД может немного отличаться в разных операционных системах.

Текст методического пособия предполагает, что читатель использует ОС Windows.

Для запуска терминального клиента достаточно найти в меню «Пуск» приложение с соответствующим названием:

SQL Shell (psql)

После запуска `psql` предложит Вам ввести пароль пользователя `postgres`. Здесь следует указать пароль, который Вы указывали при установке сервера PostgreSQL.

В случае успешной авторизации, Вы должны увидеть приглашающую строку следующего вида:

```
postgres=#
```

Это значит, что терминальный клиент успешно подключился к серверу PostgreSQL и ожидает ввода команды.

Замечание

На некоторых устройствах под управлением операционной системы Windows возникает проблема с отображением русского текста в окне `psql`.

Для устранения этой проблемы выполните следующие действия:

1. Запустите SQL Shell (`psql`).
2. Введите пароль, указанный при установке сервера PostgreSQL.
3. Введите команду:

```
\! chcp 1251
```

Команды `psql`

Как уже говорилось в вводной части, каждый сервер СУБД имеет некоторый набор команд, которые являются уникальными для каждого конкретного сервера. Такие команды не связаны с SQL. Найти список всех таких команд для PostgreSQL, можно в официальной документации, здесь же приведены только команды, важные для выполнения лабораторной работы.

Заметка

Все команды `psql` начинаются с символа обратного слэша (`\`). После него указывается один или несколько символов, которые и являются названием команды.

Если Вы хотите использовать команду, но забыли как она обозначается, всегда можно воспользоваться справкой, которая вызывается с помощью команды:

```
\?
```

Вот список полезных для нас команд:

```
\?
```

Выдаёт справку по командам `psql`

```
\h
```

Выдаёт справку по командам SQL

	(список доступных на данный момент команд или справка по конкретной команде)
\с название_базы_данных	Выполняет подключение к указанной базе данных
\l	Выводит список баз данных, доступных для подключения на текущем сервере
\dt	Выводит список таблиц подключенной базы данных
\d имя_таблицы	Выводит описание указанной таблицы
\s имя-файла-для-сохранения-истории-команд	Сохраняет историю вызова команд в указанный файл

SQL команды

Замечание

Данное методическое пособие не пытается быть учебником по SQL, а потому в этом разделе не стоит искать подробнейшее описание всех команд и принципов работы SQL. Подобные описания и теоретический материал представлены в учебнике «PostgreSQL. Основы языка SQL»[2], который представлен в открытом доступе по ссылке:

<https://postgrespro.ru/education/books/sqlprimer>

В рамках выполнения первой лабораторной работы Вам могут понадобиться следующие SQL команды:

CREATE DATABASE	Создание новой базы данных
CREATE TABLE	Создание новой таблицы
INSERT INTO	Вставка новых значений в таблицу
SELECT FROM	Выборка значений из таблицы
UPDATE	Обновление существующих значений в таблице
DELETE FROM	Удаление данных из таблицы
ALTER TABLE	Изменение структуры существующей таблицы
DROP TABLE	Удаление таблицы

Заметка

Все команды SQL должны завершаться точкой с запятой (;). Клиент psql не будет отправлять команду на сервер до тех пор пока не встретит этот символ.

CREATE DATABASE

Команда CREATE DATABASE позволяет создать новую базу данных. Синтаксис команды довольно прост:

```
CREATE DATABASE имя_создаваемой_базы_данных;
```

Если в системе уже есть база данных с указанным именем – команда выполнена не будет. Важно так же учесть, что для того чтобы начать работать с созданной базой данных, необходимо выполнить подключение к ней:

```
\с имя_базы_данных
```

CREATE TABLE

Общая схема этой команды выглядит следующим образом:

```
CREATE TABLE имя_таблицы (  
    имя_поля1 тип_поля [ограничения_целостности],  
    имя_поля2 тип_поля [ограничения_целостности],  
    ...  
    имя_поляN тип_поля [ограничения_целостности],  
    [ограничения_целостности_таблицы]  
);
```

В качестве типа данных поля можно использовать любой тип из доступных в СУБД. Список типов поддерживаемых PostgreSQL можно найти на странице официальной документации по ссылке: <https://postgrespro.ru/docs/postgresql/15/datatype>

Рассмотрим здесь лишь некоторые базовые типы:

int\integer	Целочисленное значение (размер 4 байта)
serial	Целочисленное значение с автоувеличением (размер 4 байта)

char(n)	Строка ограниченной длины n. Автоматически дополняется пробелами до указанной длины
varchar(n)	Строка ограниченной длины n. Не дополняется пробелами
text	Строка неограниченной длины
timestamp	Дата и время

Уже сейчас, зная только типы и общую структуру команды, можно создать простую таблицу:

```
CREATE TABLE simple_table (
    attr1 serial,
    attr2 varchar(15),
    attr3 char(100)
);
```

Созданная таблица будет иметь 3 столбца: attr1, attr2, attr3. Ничего кроме этих столбцов в таблице не будет. На практике подобные таблицы - без первичного ключа и дополнительных ограничений, не используются.

Ограничения целостности поля задаются в одну строку без запятых там же, где указывается его имя и тип. В качестве ключевых слов, задающих ограничения используются:

NULL \ NOT NULL	Допускает\Запрещает наличие неопределенных значений в поле
PRIMARY KEY	Устанавливает поле первичным ключом таблицы
REFERENCES имя_таблицы(имя_поля)	Связывает поле с первичным ключом другой таблицы
UNIQUE	Указывает на то, что все значения в поле должны быть уникальными (т.е. запрещает наличие дублирующихся значений в этом поле)
CHECK(логическое выражение)	Задаёт правило проверки значений поля. При внесении данных СУБД отбросит все строки, где значения в столбцах не проходят проверку CHECK

Попробуем использовать все рассмотренные ограничения:

```
CREATE TABLE constraint_table (  
    id serial PRIMARY KEY,  
    fullname varchar(120) NULL,  
    username varchar(60) NOT NULL UNIQUE,  
    age int NOT NULL CHECK(age > 17 AND age <= 110),  
    department_id int REFERENCES department(id)  
);
```

Разберем каждую строку отдельно:

1. *id serial PRIMARY KEY* – описывает первичный ключ таблицы с именем «id» и типом «serial». Указывать, что поле не может быть пустым\неопределенным (NOT NULL) не нужно, так как PRIMARY KEY создаёт такое ограничение по умолчанию.
2. *fullname varchar(120) NULL* – описывает строковое поле длиной 120 символов, которое может принимать неопределенные значения. На самом деле указывать в ограничениях ключевое слово «NULL» не нужно, так как оно присваивается полю по умолчанию. Поэтому разницы между объявлениями:

```
    fullname varchar(120) NULL,  
    fullname varchar(120),
```

нет.

3. *username varchar(60) NOT NULL UNIQUE* – описывает строковое поле длиной 60 символов, значения которого должны быть уникальными и не могут быть пустыми. Важно понимать, что в отличие от PRIMARY KEY, ключевое слово UNIQUE не задаёт дополнительных ограничений на запрет неопределенных значений в поле. Поэтому объявления:

```
    username varchar(60) NOT NULL UNIQUE,  
    username varchar(60) UNIQUE,
```

отличаются и приведут к созданию разных полей.

4. *age int NOT NULL CHECK(age > 17 AND age <= 110)* – описывает целочисленное поле, которое не может быть пустым. Дополнительно на значения в поле накладывается ограничение CHECK, которое можно описать так - значение не должно быть меньше 18 и больше 110.

5. *department_id int REFERENCES department(id)* – описывает целочисленное поле, значения которого ссылаются на поле «id» таблицы «department». Важно понимать, что типы связываемых полей должны совпадать, либо быть приводимыми друг к другу. Так, если поле в «родительской» таблице имеет тип serial, т.е. целочисленный счетчик, то в зависимой таблице ссылочное поле должно иметь тип integer.

CHECK

CHECK можно использовать не только для простых логических выражений вида: $x > y$ AND $x < z$. Но и для проверки вхождения некоторого x в множество элементов, например:

```
... CHECK(x in ('one', 'two', 'five'))
```

А так же для сопоставления строки некоторому шаблону:

```
... CHECK(имя_поля SIMILAR TO 'регулярное_выражение')
```

Как Вы могли заметить, конструкция SIMILAR TO ожидает в качестве шаблона строку в виде регулярного выражения. Подробнее о механизме регулярных выражений можно прочитать по ссылке:

https://ru.wikibooks.org/wiki/Регулярные_выражения

А потренироваться в создании таких выражений можно здесь:

<https://regex101.com/>

Рассмотрим пару примеров применения регулярных выражений для проверки строкового поля:

```
1. phone char(11) UNIQUE CHECK(phone SIMILAR TO '\d{11}')
```

Здесь создаётся строковое поле для хранения телефонного номера. Очевидно, что номер должен содержать только цифры и ничего кроме цифр. Чтобы гарантировать это, в поле создаётся ограничение CHECK в котором используется шаблон '\d{11}'. В этом шаблоне подстрока '\d' указывает на то, что в проверяемой строке ожидается символ цифры (т.е. символ от 0 до 9), а подстрока '{11}' говорит о том, что таких цифровых символов должно быть ровно 11.

Допустим, что в некоторой предметной области, нам необходимо оперировать строками вида «SSXXXXXX», где S – буква английского алфавита в верхнем регистре, а X – цифра от 1 до 5. Если поле хранящее такую строку имеет название c_code, то ограничение будет выглядеть так:

```
2. ... CHECK(c_code SIMILAR TO '[A-Z]{2}[1-5]{6}')
```

Здесь подстрока '[A-Z]' задаёт в качестве разрешенных только символы из диапазона от A до Z (включительно). '{2}' говорит, что таких разрешенных символов должно быть 2. Та же логика используется в подстроке '[1-5]{6}' – т.е. [1-5] говорит о том, какие символы мы ожидаем увидеть, а {6} указывает на количество таких символов.

Ограничения целостности таблицы

Кроме ограничений целостности, накладываемых на столбцы в SQL можно задавать ограничения таблицы в целом. Список ключевых слов позволяющих это сделать довольно мал:

PRIMARY KEY(поле1, поле2, ..., полеN)	Задаёт составной первичный ключ из указанных полей
FOREIGN KEY(имена полей) REFERENCES имя_таблицы(имена_полей)	Связывает несколько полей таблицы с первичным ключом другой таблицы
CHECK(логическое выражение)	Устанавливает правило проверки на поля таблицы

Применение ключевого слова CHECK в области ограничений целостности таблицы ничем не отличается от использования этого слова при описании ограничений отдельного столбца. Поэтому рассмотрим только вариант с PRIMARY и FOREIGN KEY:

```
CREATE TABLE constraint_table2 (  
    id serial,  
    table1_pk1 int NOT NULL,  
    table1_pk2 varchar(20) NOT NULL,  
    attr4 int,  
    PRIMARY KEY(id, table1_pk1, table1_pk2),  
    FOREIGN KEY(table1_pk1, table1_pk2)  
        REFERENCES table1(id1, id2)  
        ON DELETE CASCADE  
);
```

В этом примере создаётся составной первичный ключ с полями id, table1_pk1 и table1_pk2. Кроме того поля table1_pk1 и table1_pk2 связываются с составным первичным ключом таблицы table1.

Строка `ON DELETE CASCADE` задаёт стратегию поведения СУБД при удалении значений из родительской таблицы. То есть, если в таблице `test1` будет удалена строка, на значения которой ссылаются строки таблицы `constraint_table2`, СУБД каскадно удалит и их.

Внешний ключ можно объявить и без указания `ON DELETE CASCADE`. В таком случае СУБД будет придерживаться стратегии `NO ACTION`, которая запретит удаление в случае, если на удаляемые значения есть ссылки в других таблицах.

Кроме стратегии при удалении, можно настроить поведение СУБД и при изменении значений в родительской таблице. Синтаксис меняется не сильно:

`ON UPDATE CASCADE` – измененное значение будет автоматически распространено на все зависимые таблицы

`ON UPDATE NO ACTION` – СУБД не даст изменять значения столбца до тех пор, пока хотя бы одна зависимая таблица ссылается на них (применяется по умолчанию)

INSERT INTO

Команда `INSERT INTO` позволяет вставить значения в существующую таблицу. Синтаксис этой команды выглядит следующим образом:

```
INSERT INTO имя_таблицы(поле1, поле2, ..., полеN) VALUES  
(значение_поля1, значение_поля2, ..., значение_поляN),  
(значение_поля1, значение_поля2, ..., значение_поляN),  
...  
(значение_поля1, значение_поля2, ..., значение_поляN);
```

Допустим мы имеем таблицу `table1`, которая определена так:

```
CREATE TABLE table1 (  
    id serial PRIMARY KEY,  
    fullname varchar(120) NULL,  
    username varchar(60) NOT NULL UNIQUE,  
);
```

Тогда команда вставки может выглядеть так:

```
INSERT INTO table1(username) VALUES ('username1'),  
('my_username'), ('user123name');
```

Можно заметить, что значения будут вставляться не во все поля, а только в поле `username`. Это возможно благодаря следующим факторам:

- поле `id` указано как счетчик, который будет получать значения автоматически;
- поле `fullname` имеет пометку `NULL`, т.е. может содержать неопределенные значения.

После выполнения команды, которая приведена выше, таблица `table1` будет выглядеть так:

Таблица 4.1 – Содержимое таблицы `table1`

<code>id</code>	<code>fullname</code>	<code>username</code>
1		<code>username1</code>
2		<code>my_username</code>
3		<code>user123name</code>

SELECT ... FROM

Выборка строк таблицы всегда осуществляется командой `SELECT`. Синтаксис этой команды выглядит так:

```
SELECT имя_поля1, имя_поля2, ..., имя_поляN
FROM источник
[WHERE условие]
[GROUP BY элемент_группирования]
[HAVING условие]
[ORDER BY выражение [ASC | DESC]];
```

Можно заметить, что кроме обязательных элементов (`SELECT` и `FROM`) команда может содержать большое количество дополнительных ключевых слов. В этом разделе мы рассмотрим только использование ключевых слов `SELECT`, `FROM` и `WHERE`.

...глава находится в доработке...

СОЗДАНИЕ ИНДЕКСОВ

Резервное сохранение результатов работы

Использование сервера СУБД предполагает, что данные будут надежно сохранены и без дополнительных действий со стороны пользователя. Тем не менее важно знать как можно сохранить текущее состояние базы данных в отдельный файл. Особенно важным будет это умение для тех студентов, которые выполняют

лабораторные на компьютерах кафедры, где нельзя гарантировать, что кто-то случайно не удалит все ваши труды (например, командой DROP DATABASE).

Текущее состояние базы данных можно сохранить в виде скрипта, то есть в виде набора инструкций SQL. В дальнейшем СУБД, используя эти инструкции, сможет воссоздать сохраненную базу данных вместе с информацией, присутствующей в ней на момент создания скрипта.

В PostgreSQL функцию сохранения БД выполняет утилита pg_dump. В операционных системах семейства UNIX эту утилиту можно вызвать прямо в терминале, однако в ОС Windows сделать этого напрямую нельзя. Поэтому вызов утилиты в командной строке Windows выглядит следующим образом:

«Полный путь до pg_dump.exe» имя_базы_данных -f имя_файла

После выполнения данной команды база данных с указанным именем будет сохранена в виде набора SQL инструкций в файл имя_файла. При этом пользователь Windows должен обладать правами на просмотр сохраняемой БД. В случае, если это не так, команду можно запустить от имени пользователя postgres, который по умолчанию является пользователем с администраторскими правами:

«Всё так же путь» -U postgres имя_базы_данных -f имя_файла

Если сервер PostgreSQL был установлен в директорию «Program Files» на диске «С», то путь до pg_dump будет выглядеть так:

«C:\Program Files\PostgreSQL\15\bin\pg_dump»

Заметка

Чтобы не прописывать полный путь до утилиты pg_dump при каждом вызове, можно добавить директорию установки PostgreSQL в системный путь (PATH) Windows. Это позволит, в том числе, вызывать в командной строке и утилиту psql.

Вот несколько ссылок на сторонние ресурсы, где описано как можно добавить новые значения в PATH:

<https://blog.sqlbackupandftp.com/setting-windows-path-for-postgres-tools>

[https://learn.microsoft.com/ru-ru/previous-versions/office/developer/sharepoint-2010/ee537574\(v=office.14\)](https://learn.microsoft.com/ru-ru/previous-versions/office/developer/sharepoint-2010/ee537574(v=office.14))

Требования к содержанию отчета

Отчёт о ЛР1 должен содержать следующие разделы.

- 1 Цель лабораторной работы.
- 2 Описание таблиц БД из индивидуального задания.

3 Описание процесса реализации таблиц.

4 Выводы.

В разделе 3 следует привести описание использованных Вами команд SQL и логики ограничений в выражениях CHECK. Здесь же должны быть приведены исходные тексты SQL всех созданных таблиц.

В разделе 4 перечислите то, чему Вы научились в ходе выполнения работы.