|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.01**

**Отчет**

**по лабораторной работе №** **2**

**Название:** Создание БД для приложения

**Дисциплина:** Базы данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-32Б |  |  | Н. И. Белоусов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Д. А. Миков |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2022

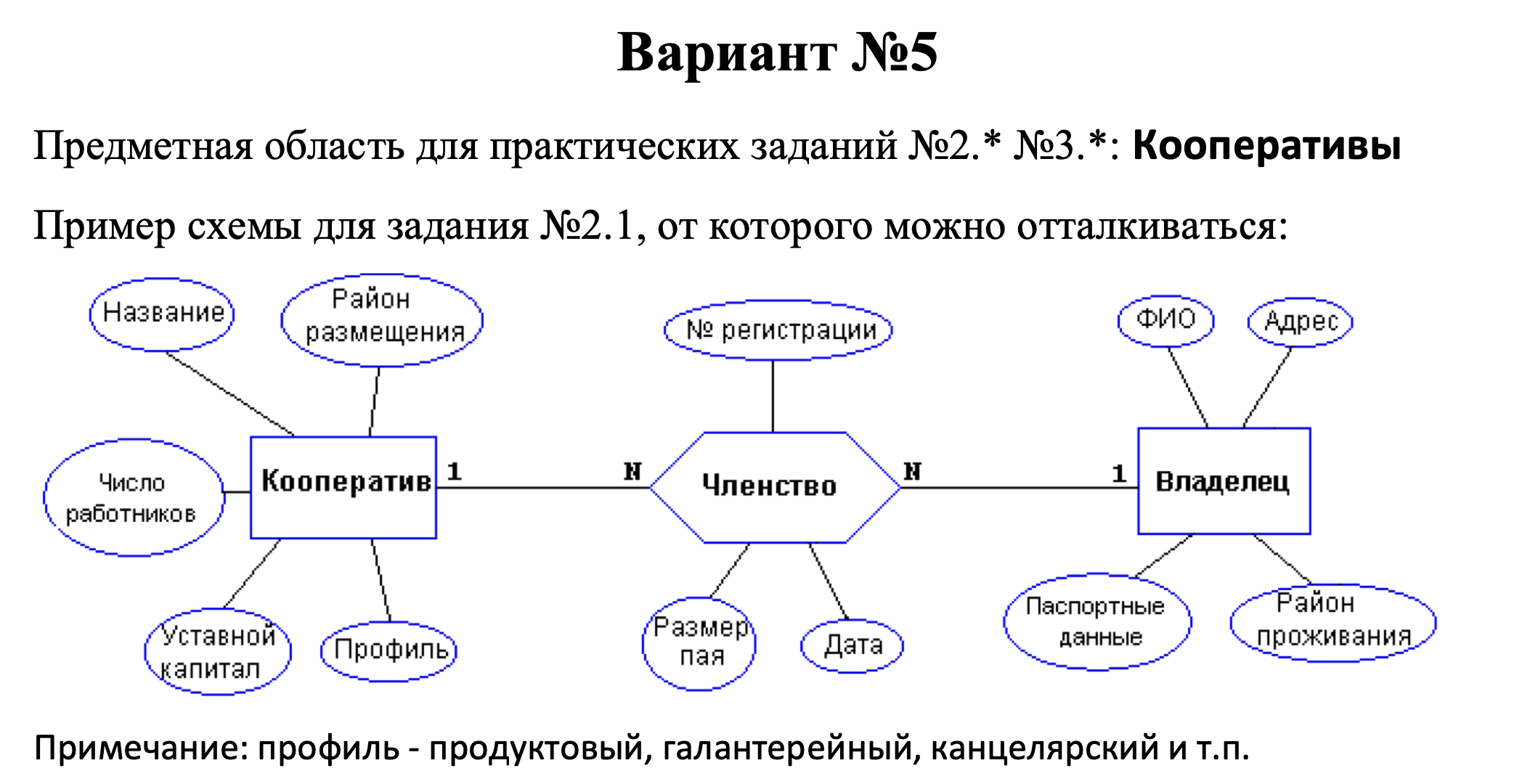
**Вариант 5.**

**Часть 1.**

**Задание:**

Второе практическое задание связано с проектированием схемы базы данных для работы приложения (WEB/Mobile/Desktop). Каждый индивидуальный вариант содержит предметную область, из которой должна быть проектируемая база данных. К данной предметной области необходимо добавить не менее 2х дополнительных таблиц, необходимых для детального решения поставленной задачи. Задачей студента является решить, для чего будет использоваться создаваемая база данных, и, исходя из этого, построить её концептуальную схему. Результатом данной части лабораторной работы является схема базы данных (в виде ER-диаграммы, содержащей таблицы и связи между ними, с уточнением типов столбцов, с описание внешних и первичных ключей). При сдаче задания студент должен обосновать соответствие созданной схемы поставленной задаче.

Схема

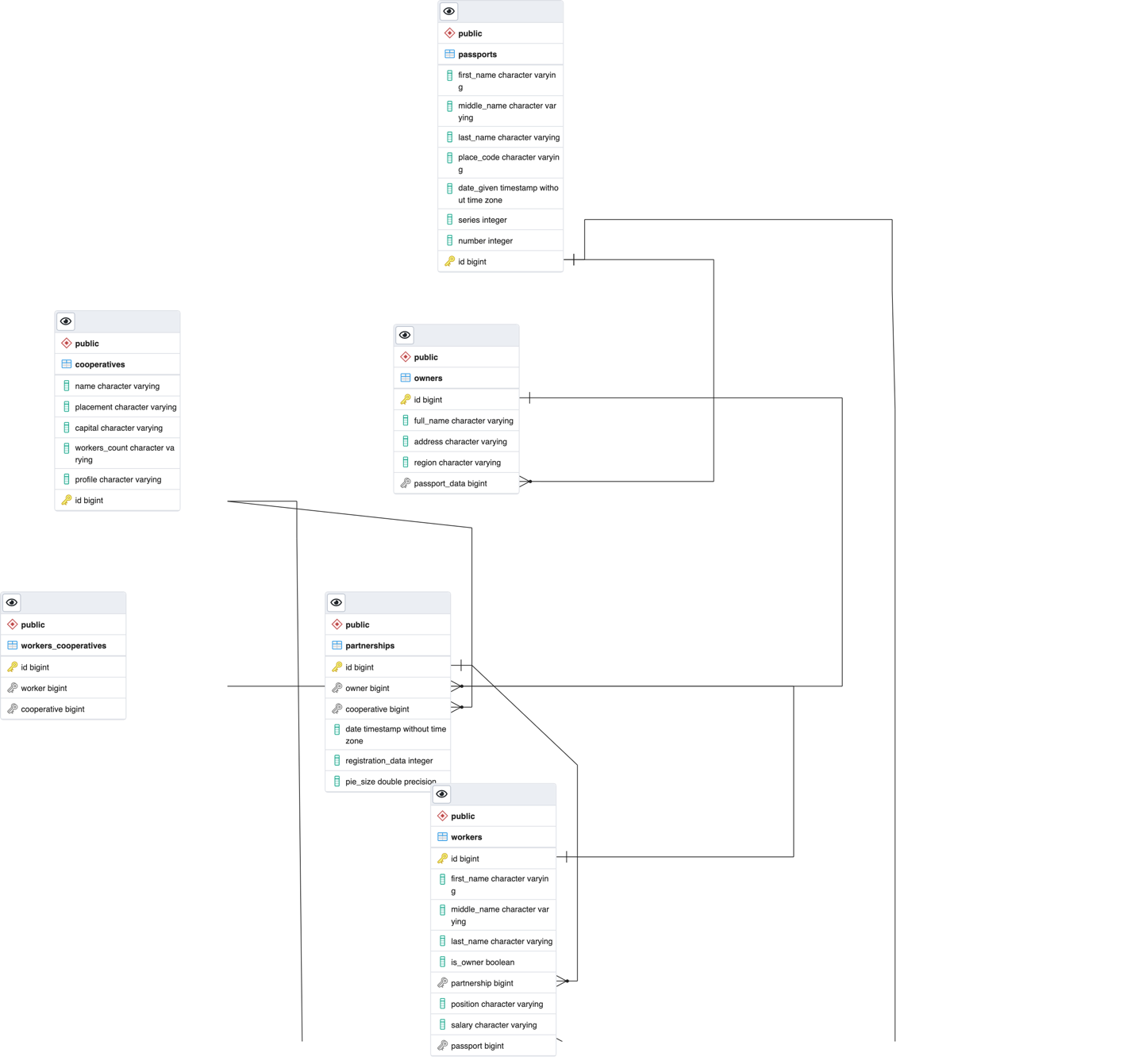
****

Добавим 2 таблицы к этой схеме:

**Рабочий**, содержит в себе имя, паспортные данные, зарплату, должность, является ли он владельцем какого-то кооператива. У одного кооператива может быть много рабочих, связываем с помощью внешней таблицы.

**Рабочий-кооператив** – содержит в себе внешние ключи рабочего и кооператива, создавая связь o2m между оными.

**Схема:**

****

**Часть 2:**

**Задание:**

Необходимо подготовить SQL-скрипт для создания таблиц согласно разработанной схеме, полученной в предыдущем задании (с уточнением типов столбцов). Необходимо определить первичные и внешние ключи, а также декларативные ограничения целостности (возможность принимать неопределенное значение, уникальные ключи, проверочные ограничения и т. д.). Таблицы следует создавать в отдельной базе данных. Кроме того, нужно подготовить данные для заполнения созданных таблиц. Объем подготовленных данных должен составлять не менее 10 экземпляров для каждой из стержневых сущностей и 1000 экземпляров для целевой сущности. На основе этих данных необходимо создать SQL-скрипт для вставки соответствующих строк в таблицы БД.

*-- Table: public.cooperatives*

*-- DROP TABLE IF EXISTS public.cooperatives;*

*CREATE* *TABLE* *IF* *NOT* *EXISTS* public.cooperatives

(

*name* *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

placement *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

capital *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

workers\_count *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

*profile* *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

id *bigint* *NOT* *NULL*,

*CONSTRAINT* cooperatives\_pkey *PRIMARY* *KEY* (id)

)

TABLESPACE pg\_default;

*ALTER* *TABLE* *IF* *EXISTS* public.cooperatives

*OWNER* *to* postgres;

*-- Table: public.owners*

*-- DROP TABLE IF EXISTS public.owners;*

*CREATE* *TABLE* *IF* *NOT* *EXISTS* public.owners

(

id *bigint* *NOT* *NULL*,

full\_name *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

*address* *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

region *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

passport\_data *bigint*,

*CONSTRAINT* owners\_pkey *PRIMARY* *KEY* (id),

*CONSTRAINT* passport *FOREIGN KEY* (passport\_data)

*REFERENCES* public.passports (id) MATCH *SIMPLE*

*ON* *UPDATE* *NO* *ACTION*

*ON DELETE* *NO* *ACTION*

NOT VALID

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public.owners

OWNER to postgres;

*-- Table: public.partnerships*

*-- DROP TABLE IF EXISTS public.partnerships;*

*CREATE* *TABLE* *IF* *NOT* *EXISTS* public.partnerships

(

id *bigint* *NOT* *NULL*,

*owner* *bigint* *NOT* *NULL*,

cooperative *bigint* *NOT* *NULL*,

*date* *timestamp* *without* *time* *zone* *NOT* *NULL*,

registration\_data *integer* *NOT* *NULL*,

pie\_size *double precision* *NOT* *NULL*,

*CONSTRAINT* partnerships\_pkey *PRIMARY* *KEY* (id),

*CONSTRAINT* cooperative *FOREIGN KEY* (cooperative)

*REFERENCES* public.cooperatives (id) MATCH *SIMPLE*

*ON* *UPDATE* *NO* *ACTION*

*ON DELETE CASCADE*

*NOT* VALID,

*CONSTRAINT* *owner* *FOREIGN KEY* (*owner*)

*REFERENCES* public.owners (id) MATCH *SIMPLE*

*ON* *UPDATE* *NO* *ACTION*

*ON DELETE CASCADE*

*NOT* VALID

)

TABLESPACE pg\_default;

*ALTER* *TABLE* *IF* *EXISTS* public.partnerships

*OWNER* *to* postgres;

*-- Table: public.passports*

*-- DROP TABLE IF EXISTS public.passports;*

*CREATE* *TABLE* *IF* *NOT* *EXISTS* public.passports

(

first\_name *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

middle\_name *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

last\_name *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

place\_code *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

date\_given *timestamp* *without* *time* *zone* *NOT* *NULL*,

series *integer* *NOT* *NULL*,

"number" *integer* *NOT* *NULL*,

id *bigint* *NOT* *NULL*,

*CONSTRAINT* passports\_pkey *PRIMARY* *KEY* (id)

)

TABLESPACE pg\_default;

ALTER TABLE IF EXISTS public.passports

OWNER to postgres;

*-- Table: public.workers\_cooperatives*

*-- DROP TABLE IF EXISTS public.workers\_cooperatives;*

*CREATE* *TABLE* *IF* *NOT* *EXISTS* public.workers\_cooperatives

(

id *bigint* *NOT* *NULL*,

worker *bigint* *NOT* *NULL*,

cooperative *bigint* *NOT* *NULL*,

*CONSTRAINT* workers\_cooperatives\_pkey *PRIMARY* *KEY* (id),

*CONSTRAINT* cooperative *FOREIGN KEY* (cooperative)

*REFERENCES* public.cooperatives (id) MATCH *SIMPLE*

*ON* *UPDATE* *NO* *ACTION*

*ON DELETE* *NO* *ACTION*

*NOT* VALID,

*CONSTRAINT* worker *FOREIGN KEY* (worker)

*REFERENCES* public.workers (id) MATCH *SIMPLE*

*ON* *UPDATE* *NO* *ACTION*

*ON DELETE* *NO* *ACTION*

*NOT* VALID

)

TABLESPACE pg\_default;

*ALTER* *TABLE* *IF* *EXISTS* public.workers\_cooperatives

*OWNER* *to* postgres;

*-- Table: public.workers*

*-- DROP TABLE IF EXISTS public.workers;*

*CREATE* *TABLE* *IF* *NOT* *EXISTS* public.workers

(

id *bigint* *NOT* *NULL*,

first\_name *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

middle\_name *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

last\_name *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

is\_owner *boolean* *NOT* *NULL*,

partnership *bigint*,

"position" *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

salary *character* varying COLLATE pg\_catalog."default" *NOT* *NULL*,

passport *bigint* *NOT* *NULL*,

*CONSTRAINT* workers\_pkey *PRIMARY* *KEY* (id),

*CONSTRAINT* partnership *FOREIGN KEY* (partnership)

*REFERENCES* public.partnerships (id) MATCH *SIMPLE*

*ON* *UPDATE* *NO* *ACTION*

*ON DELETE* *NO* *ACTION*,

*CONSTRAINT* passport *FOREIGN KEY* (passport)

*REFERENCES* public.passports (id) MATCH *SIMPLE*

*ON* *UPDATE* *NO* *ACTION*

*ON DELETE* *NO* *ACTION*

*NOT* VALID

)

TABLESPACE pg\_default;

*ALTER* *TABLE* *IF* *EXISTS* public.workers

*OWNER* *to* postgres;

*from* faker *import* Faker

*from* rich.console *import* Console

*from* rich.table *import* Table

*from* rich.style *import* Style

*import* random

*from* datetime *import* datetime

*from* rich.progress *import* track

*import* json

*import* psycopg2

*from* os.path *import* isfile, join

*from* os *import* listdir

currencies *=* ("рубль(ей)", "доллар(ов)", "евро", "юань(и)")

profiles *=* ("продуктовый", "галантерейный", "канцелярский",

"сапожный", "машиностроительный", "фермерский")

positions *=* ("директор", "уборщик", "менеджер")

OWNER\_COUNT *=* 1000

COOPERATIVE\_COUNT *=* 1000

PARTNERSHIP\_COUNT *=* 10000

WORKER\_COUNT *=* 10000

PASSPORT\_COUNT *=* OWNER\_COUNT *+* WORKER\_COUNT

*class* FakerWrapper:

*def* \_\_init\_\_(*self*):

*self*.faker *=* Faker(*locale=*'ru-RU')

*def* generate\_money(*self*):

currency *=* random.choice(currencies)

*return* str(random.randint(1, 200000000)) *+* *f*" {currency}"

*def* generate\_location(*self*):

*return* *self*.faker.address()

*def* generate\_region(*self*): *return* *self*.faker.administrative\_unit()

*def* generate\_company\_name(*self*): *return* *self*.faker.company()

*def* generate\_name(*self*): *return* *self*.faker.name()

*def* generate\_date\_time(*self*): *return* *self*.faker.date\_object()

*def* generate\_job(*self*): *return* *self*.faker.job()

*class* DBWorker:

*def* \_\_init\_\_(*self*, *host*, *port*, *database\_name*, *user*) -> None:

*self*.connection *=* psycopg2.connect(

*dbname=database\_name*, *user=user*, *port=port*, *host=host*)

*self*.cursor *=* *self*.connection.cursor()

*self*.connection.autocommit *=* True

*def* drop\_all(*self*):

*try*:

*self*.cursor.execute("""drop table public.cooperatives cascade;""")

*self*.cursor.execute("""drop table public.passports cascade;""")

*self*.cursor.execute("""drop table public.workers cascade;""")

*self*.cursor.execute("""drop table public.owners cascade;""")

*self*.cursor.execute(

"""drop table public.workers\_cooperatives cascade;""")

*self*.cursor.execute("""drop table public.partnerships cascade;""")

*except*:

Console().log("[orange] all tables are already deleted [/orange]")

*def* create\_all(*self*):

*self*.drop\_all()

create\_files *=* [f *for* f *in* listdir(

"./scripts") *if* isfile(join("./scripts", f))]

*with* open("./scripts/create\_passports.sql", 'r') *as* f:

*self*.cursor.execute(f.read())

f.close()

*with* open("./scripts/create\_cooperatives.sql", 'r') *as* f:

*self*.cursor.execute(f.read())

f.close()

*with* open("./scripts/create\_owners.sql", 'r') *as* f:

*self*.cursor.execute(f.read())

f.close()

*with* open("./scripts/create\_partnerships.sql", 'r') *as* f:

*self*.cursor.execute(f.read())

f.close()

*with* open("./scripts/create\_workers.sql", 'r') *as* f:

*self*.cursor.execute(f.read())

f.close()

*with* open("./scripts/create\_worker\_cooperatives.sql", 'r') *as* f:

*self*.cursor.execute(f.read())

f.close()

*# for i in track(range(len(create\_files))):*

*# with open(f'./scripts/{create\_files[i]}', 'r') as f:*

*# if (f.name.find("insert") == -1):*

*# self.cursor.execute(f.read())*

*def* fill\_passports(*self*):

t *=* datetime.now()

*for* i *in* track(range(PASSPORT\_COUNT), *description=*"Generating passports"):

tmp *=* Passport()

*self*.cursor.execute(

"""INSERT INTO public.passports(

first\_name, middle\_name, last\_name, place\_code, date\_given, series, "number", id)

VALUES (%s, %s, %s, %s, %s, %s, %s, %s);""", (tmp.name, tmp.middle\_name, tmp.last\_name, tmp.place\_code, tmp.data\_given, tmp.series, tmp.number, i))

Console().log(*f*"[green] {*self*.fill\_passports} done [/green] in [yellow] {datetime.now() *-* t} [/yellow] ")

*def* fill\_cooperatives(*self*):

t *=* datetime.now()

*for* i *in* track(range(COOPERATIVE\_COUNT), *description=*"Generating cooperatives"):

tmp *=* Cooperative()

*self*.cursor.execute(

"""INSERT INTO public.cooperatives(

name, placement, capital, workers\_count, profile, id)

VALUES (%s, %s, %s, %s, %s, %s);""", (tmp.name, tmp.placement, tmp.capital, tmp.workers\_count, tmp.profile, i)

)

Console().log(*f*"done in [yellow] {datetime.now() *-* t}")

*def* fill\_owners(*self*):

t *=* datetime.now()

*for* i *in* track(range(OWNER\_COUNT), *description=*"Generating owners"):

tmp *=* Owner()

*self*.cursor.execute("""INSERT INTO public.owners(

id, full\_name, address, region, passport\_data)

VALUES (%s, %s, %s, %s, %s);""", (i, tmp.name, tmp.adress, tmp.region, tmp.passport\_data))

Console().log(*f*"done in [yellow] {datetime.now() *-* t}")

*def* fill\_workers(*self*):

t *=* datetime.now()

*for* i *in* track(range(WORKER\_COUNT), *description=*"Generating workers"):

tmp *=* Worker()

*self*.cursor.execute("""INSERT INTO public.workers(

id, first\_name, middle\_name, last\_name, is\_owner, partnership, "position", salary, passport)

VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s);

""", (i, tmp.first\_name, tmp.middle\_name, tmp.last\_name, tmp.is\_owner, tmp.cooperative, tmp.position, tmp.salary, tmp.passport))

Console().log(*f*"done in [yellow] {datetime.now() *-* t}")

*def* fill\_partnerships(*self*):

t *=* datetime.now()

*for* i *in* track(range(PARTNERSHIP\_COUNT), *description=*"Generating partnerships"):

tmp *=* Partnership()

*self*.cursor.execute("""INSERT INTO public.partnerships(

id, owner, cooperative, date, registration\_data, pie\_size)

VALUES (%s, %s, %s, %s, %s, %s);""", (i, tmp.owner, tmp.cooperative, tmp.date, tmp.registration\_data, tmp.pie\_size))

Console().log(*f*"done in [yellow] {datetime.now() *-* t}")

*def* fill\_worker\_cooperatives(*self*):

t *=* datetime.now()

*for* i *in* track(range(WORKER\_COUNT)):

tmp *=* WorkerCooperative()

*self*.cursor.execute("""INSERT INTO public.workers\_cooperatives(

id, worker, cooperative)

VALUES (%s, %s, %s);""", (i, tmp.worker, tmp.cooperative))

Console().log(*f*"done in [yellow] {datetime.now() *-* t}")

*def* generate\_all(*self*):

t *=* datetime.now()

*self*.fill\_passports()

*self*.fill\_cooperatives()

*self*.fill\_owners()

*self*.fill\_partnerships()

*self*.fill\_workers()

*self*.fill\_worker\_cooperatives()

Console().log(*f*"done generating in [yellow] {datetime.now() *-* t}")

*class* Worker:

*def* \_\_init\_\_(*self*) -> None:

*self*.fakerWrapper *=* FakerWrapper()

*self*.generate\_worker()

*def* generate\_worker(*self*):

fullname: str *=* *self*.fakerWrapper.generate\_name()

*self*.first\_name *=* fullname.split(" ")[0]

*self*.middle\_name *=* fullname.split(" ")[1]

*self*.last\_name *=* fullname.split(" ")[2]

*self*.is\_owner *=* True *if* random.randint(0, 10) *%* 2 *==* 0 *else* False

*if* (*self*.is\_owner):

*self*.cooperative *=* random.randint(1, PARTNERSHIP\_COUNT *-* 1)

*else*:

*self*.cooperative *=* None

*self*.job *=* *self*.fakerWrapper.generate\_job()

*self*.position *=* random.choice(positions)

*self*.salary *=* *self*.fakerWrapper.generate\_money()

*self*.passport *=* random.randint(1, PASSPORT\_COUNT *-* 1)

*class* Cooperative:

*def* \_\_init\_\_(*self*):

*self*.fakerWrapper *=* FakerWrapper()

*self*.generate\_cooperative()

"""генерирует кооператив

"""

*def* generate\_cooperative(*self*):

*self*.name *=* *self*.fakerWrapper.generate\_company\_name()

*self*.placement *=* *self*.fakerWrapper.generate\_region()

*self*.capital *=* *self*.fakerWrapper.generate\_money()

*self*.workers\_count *=* random.randint(1, 255)

*self*.profile *=* random.choice(profiles)

*def* generate\_places(*self*):

*pass*

*def* to\_string(*self*) -> str:

*return* *f*"{*self*.name}\n {*self*.placement}"

*class* WorkerCooperative:

*def* \_\_init\_\_(*self*) -> None:

*self*.generate\_connection()

*self*.generate\_connection()

*def* generate\_connection(*self*):

*self*.worker *=* random.randint(1, WORKER\_COUNT *-* 1)

*self*.cooperative *=* random.randint(1, COOPERATIVE\_COUNT *-* 1)

*class* Passport:

*def* \_\_init\_\_(*self*) -> None:

*self*.faker *=* FakerWrapper()

*self*.generate\_passport()

*def* generate\_passport(*self*):

fullname *=* *self*.faker.generate\_name()

*self*.name *=* fullname.split(" ")[0]

*self*.middle\_name *=* fullname.split(" ")[1]

*self*.last\_name *=* fullname.split(" ")[2]

*self*.place\_code *=* str(random.randint(100, 999)) *+* \

"-" *+* str(random.randint(100, 999))

*self*.data\_given *=* *self*.faker.generate\_date\_time()

*self*.series *=* str(random.randint(10, 99)) *+* str(random.randint(10, 99))

*self*.number *=* random.randint(100000, 999999)

*def* to\_map(*self*):

*return* *self*.\_\_dict\_\_

*class* Owner:

*def* \_\_init\_\_(*self*) -> None:

*self*.faker *=* FakerWrapper()

*self*.generate\_owner()

*def* generate\_owner(*self*):

*self*.name *=* *self*.faker.generate\_name()

*self*.adress *=* *self*.faker.generate\_location()

*self*.region *=* *self*.faker.generate\_region()

*self*.passport\_data *=* random.randint(1, PASSPORT\_COUNT *-* 1)

*def* to\_map(*self*):

d *=* *self*.\_\_dict\_\_

d['passport\_data'] *=* *self*.passport\_data.to\_map()

*return* d

*class* Partnership:

*def* \_\_init\_\_(*self*) -> None:

*self*.cooperative *=* random.randint(1, COOPERATIVE\_COUNT *-* 1)

*self*.owner *=* random.randint(1, OWNER\_COUNT *-* 1)

*self*.registration\_data *=* random.randint(1, 1000000)

*self*.date *=* FakerWrapper().generate\_date\_time()

*self*.pie\_size *=* random.randint(1, 1000) */* 1000

*class* Interface:

*def* \_\_init\_\_(*self*):

*self*.console *=* Console()

*self*.db\_worker *=* DBWorker(

*host=*"127.0.0.1", *port=*"8014", *user=*"postgres", *database\_name=*"labs")

*def* start(*self*):

*self*.select\_variant()

"""

(a, b, c) = interface.ask\_for\_input("a","b","c")

"""

*def* ask\_for\_input(*self*, *\*vars*) -> list:

target *=* []

*for* i *in* *vars*:

*self*.console.log(*f*'provide {i*=*}')

target.append(input())

*return* target

"""prints menu for generator"""

*def* print\_menu(*self*):

table\_of\_choice *=* Table(*title=*"Generator menu")

table\_of\_choice.add\_column("hotkey", *style=*Style(*color=*"green"))

table\_of\_choice.add\_column("description")

*with* open("./data\_generator/configs/menu\_config.json", 'r')*as* f:

*try*:

*self*.hotkeys *=* json.load(*fp=*f)

print(*self*.hotkeys)

*for* i *in* list(*self*.hotkeys.keys()):

table\_of\_choice.add\_row(*f*"{i}", str(*self*.hotkeys[i]))

*self*.console.print(table\_of\_choice)

*except*:

*self*.console.log("print\_menu method [red] error [/red]")

*def* select\_variant(*self*):

*self*.print\_menu()

variant *=* *self*.ask\_for\_input("variant")

variant *=* variant[0]

*while* variant *not* *in* list(*self*.hotkeys.keys()):

*self*.console.clear()

*self*.console.print("[orange] wrong input [/orange]")

*self*.print\_menu()

variant *=* *self*.ask\_for\_input("variant")

*match* variant:

*case* "0":

*self*.shutdown()

*case* "1":

*self*.db\_worker.generate\_all()

input()

*self*.console.clear()

*self*.select\_variant()

*case* "2":

*self*.db\_worker.create\_all()

input()

*self*.console.clear()

*self*.select\_variant()

*case* "3":

*self*.db\_worker.fill\_owners()

input()

*self*.console.clear()

*self*.select\_variant()

*case* "4":

*self*.db\_worker.fill\_partnerships()

input()

*self*.console.clear()

*self*.select\_variant()

*case* "5":

*self*.db\_worker.fill\_workers()

input()

*self*.console.clear()

*self*.select\_variant()

*case* "6":

*self*.db\_worker.fill\_worker\_cooperatives()

input()

*self*.console.clear()

*self*.select\_variant()

*def* shutdown(*self*):

*self*.console.print("[green] shutting down [/green]")

exit()

""" to display table

vars: list(n, m+1) - table with \*\*column names\*\*

example:

```python

interface.debug\_output\_table([["name", a, b], ["age", 12, 24]])

```

| name | age |

| a | 12 |

| b | 24 |

"""

*def* debug\_output\_table(*self*, *vars*: list):

*self*.console.log("print\_debud\_table method: [green] start [/green]")

target *=* Table()

*if* (len(*vars*)) *>* 10:

*for* i *in* range(5, len(*vars*) *-* 5):

*vars*.remove(i)

*for* i *in* range(len(*vars*)):

target.add\_column(*f*'{*vars*[i][0]}')

*# что я высрал еб твою мать*

*for* i *in* range(1, len(*vars*)):

target.add\_row(*\**[str(*vars*[\_][i]) *for* \_ *in* range(len(*vars*[i]))])

*self*.console.print(target)

*self*.console.log("print\_debud\_table method: [green] end [/green]\n")

*def* debug\_output\_obj(*self*, *value*: object):

*try*:

*self*.console.print(str(*value*))

*except*:

*self*.console.log(*self*.console.log(

*f*"debug\_output\_obj method: [red] error [/red]\nno convert to string for object of type {type(*value*)}"))

**Контрольные вопросы**

• В чем различие типов CHAR и VARCHAR? VARCHAR и TEXT?

Данные типа CHAR, которые имеют фиксированную длину, быстрее всего сохраняют и достают информацию, но могут потреблять излишнее дисковое пространство. VARCHAR, строка изменяющейся длины, работает медленнее чем CHAR, но не потребляют излишнее дисковое пространство.

• Что такое внешний ключ?

Внешний ключ SQL — это ключ, используемый для объединения двух таблиц.

• Какие существуют способы поддержания ссылочной целостности?

СУБД имеют механизм автоматического поддержания ссылочной целостности. Любая операция, изменяющая данные в таблице, вызывает автоматическую проверку ссылочной целостности. существуют три подхода, каждый из которых поддерживает целостность по ссылкам.

1) запрещается производить удаление кортежа, на который существуют ссылки

2) при удалении кортежа, на который имеются ссылки, во всех ссылающихся кортежах значение внешнего ключа автоматически становится неопределенным.

3) при удалении кортежа из отношения, на которое ведет ссылка, из ссылающегося отношения автоматически удаляются все ссылающиеся кортежи.

• Что такое уникальный ключ?

Уникальный ключ - столбец или группа столбцов, которые могут идентифицировать уникальность в строке, служат для однозначной идентификации записей в таблице.

• Что такое SERIAL?

SERIAL - автоинкрементирующееся числовое значение, которое занимает 4 байта и может хранить числа от 1 до 2147483647. Значение данного типа образуется путем автоинкрементазначения предыдущей строки.

• Рассказать о значениях по умолчанию и неопределенных значениях. При создании таблицы значение некоторых полей можно определить благодаря оператору DEFAULT. В случае, если при заполнении таблицы в поле не будет занесено значение, то будет использоваться значение, данное при создании таблицы. Оператор NOT NULL предотвращает возникновение неопределенных полей. Если упомянутые операторы не использованы при объявлении поля, значит поле может принимать значение NULL.

• Как можно хранить даты и время?

Для хранения даты и времени в SQL существует несколько типов данных:

-time в формате чч:мм:сс[.ннннннн]

-date в формате ГГГГ-ММ-ДД

-smalldatetime в формате ГГГГ-ММ-ДД чч:мм:сс

-datetime в формате ГГГГ-ММ-ДД чч:мм:сс[.ннн]

-datetime2 в формате ГГГГ-ММ-ДД чч:мм:сс[.ннннннн]

-datetimeOFFSET в формате ГГГГ-ММ-ДД чч:мм:сс[.ннннннн] [+|-]чч:мм

• Рассказать о числовых типах данных.

BIT: хранит значение 0 или 1. Занимает 1 байт.

TINYINT: хранит числа от 0 до 255. Занимает 1 байт.

SMALLINT: хранит числа от –32 768 до 32 767. Занимает 2 байта

INT: хранит числа от –2 147 483 648 до 2 147 483 647. Занимает 4 байта.

BIGINT: хранит числа от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807, которые занимают в памяти 8 байт.

DECIMAL: хранит числа c фиксированной точностью. Занимает от 5 до 17 байт в зависимости от количества чисел после запятой. Данный тип может принимать два параметра precision и scale: DECIMAL(precision, scale). Precision - максимальное количество цифр, которые может хранить число. Это значение должно находиться в диапазоне от 1 до 38. По умолчанию оно равно 18. Scale - 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.

NUMERIC: данный тип аналогичен типу DECIMAL.

SMALLMONEY: хранит дробные значения от -214 748.3648 до 214 748.3647. Предназначено для хранения денежных величин. Занимает 4 байта.

MONEY: хранит дробные значения от -922 337 203 685 477.5808 до 922 337 203 685 477.5807. FLOAT: хранит числа от –1.79E+308 до 1.79E+308. Занимает от 4 до 8 байт в зависимости от дробной части.

REAL: хранит числа от –340E+38 to 3.40E+38. Занимает 4 байта.

**Часть 3**

**Задание:**

Подготовить 3-4 выборки, которые имеют осмысленное значение для предметной области, и также составить для них SQL-скрипты. Сформулировать 3-4 запроса на изменение и удаление из базы данных. Запросы должны быть сформулированы в терминах предметной области. Среди запросов обязательно должны быть такие, которые будут вызывать срабатывание ограничений целостности. Составить SQL скрипты для выполнения этих запросов.

1. Получим серии и номера паспортов работников кооперативов.

*select* w.first\_name *as* "имя", w.last\_name *as* "фамилия", p.series *as* "серия", p."number" *as* "number"

*from* public.workers w

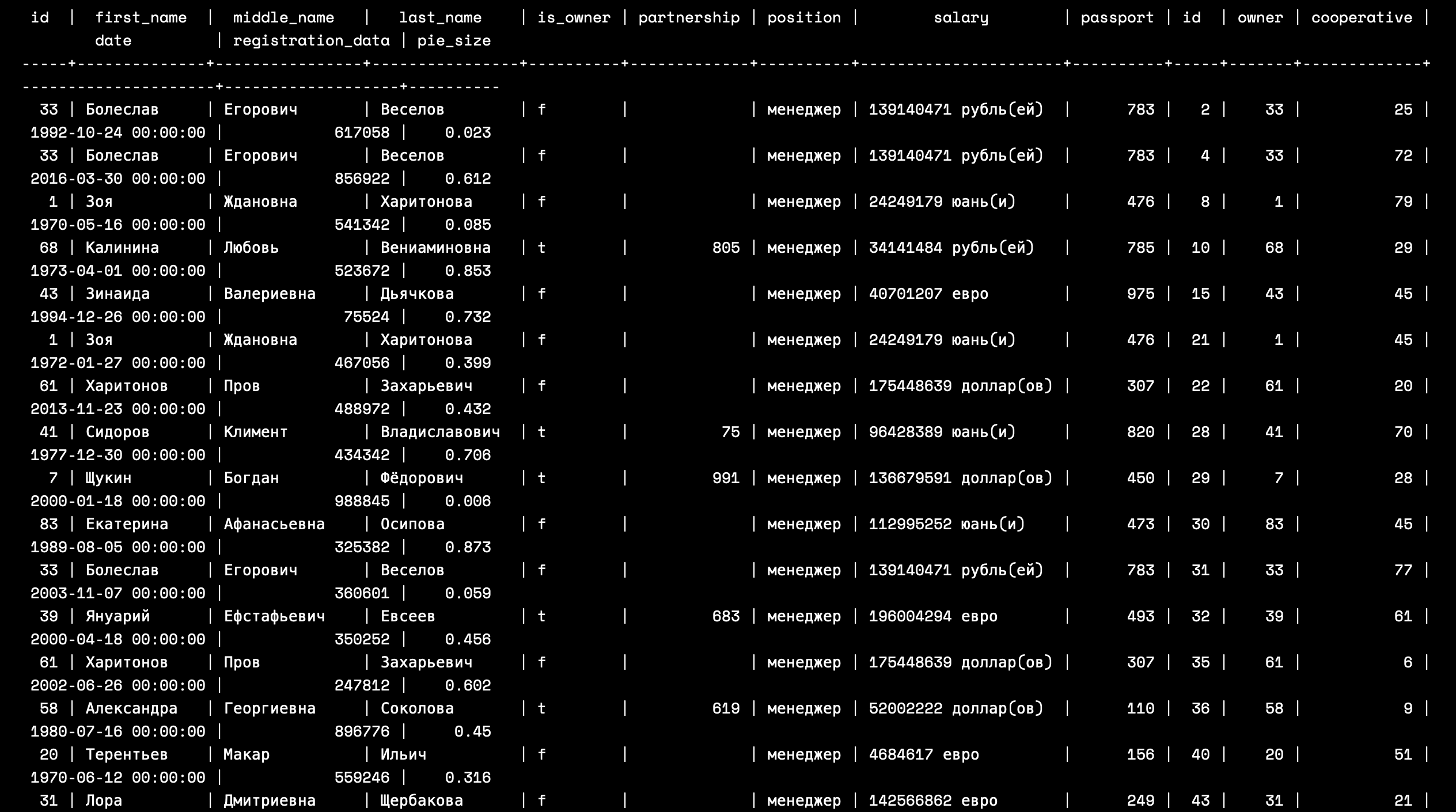
*left join* passport p *on* p.id *=* w.passport



1. Получаем всех владельцев кооперативов менеджеров

*select* *\** *from* workers w

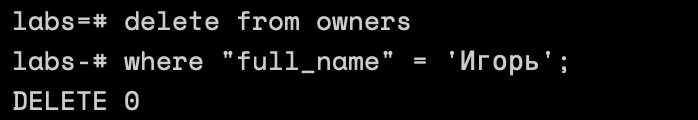
*left join* partnerships part *on* part.owner *=* w.id *and* w.position *=* 'менеджер'



1. Уберем всех Игорей из владельцев

*delete from owners*

*where "full\_name" = 'Игорь'*



**Контрольные вопросы**

• Рассказать об аномалиях доступа к БД.

-грязное чтение: транзакция читает данные, записанные параллельной незавершённой транзакцией.

-неповторяемое чтение: транзакция повторно читает те же данные, что и раньше, и обнаруживает, что они были изменены другой транзакцией (которая завершилась после первого чтения).

-фантомное чтение: транзакция повторно выполняет запрос, возвращающий набор строк для некоторого условия, и обнаруживает, что набор строк, удовлетворяющих условию, изменился из-за транзакции, завершившейся за это время.

-аномалия сериализации: результат успешной фиксации группы транзакций оказывается несогласованным при всевозможных вариантах исполнения этих транзакций по очереди.

• Перечислить аномалии, возникающие на каждом из уровней изолированности.

Грязное чтение возможно при уровне изоляции: Read uncommited, но не в PosgreSQL

Неповторяемое чтение возможно при уровне изоляции: Read Uncommited и Read Сommited

Фантомное чтение возможно при уровне изоляции: Read Uncommited, Read Сommited и Repeatable Read, но не в PosgreSQL

Аномалия сериализации возможно при уровне изоляции: Read Uncommited, Read Сommited и Repeatable Read

• Рассказать о свойствах транзакций.

-Свойство атомарности (Atomicity) - транзакция должна быть выполнена в целом или не выполнена вовсе.

-Свойство согласованности (Consistency) - по мере выполнения транзакций данные переходят из одного согласованного состояния в другое — транзакция не разрушает взаимной согласованности данных.

-Свойство изолированности (Isolation) - конкурирующие за доступ к базе данных транзакции физически обрабатываются последовательно, изолированно друг от друга, но для пользователей это выглядит так, как будто они выполняются параллельно.

-Свойство долговечности (Durability) - если транзакция завершена успешно, то те изменения в данных, которые были ею произведены, не могут быть потеряны ни при каких обстоятельствах (даже в случае последующих ошибок).

• Рассказать об управлении транзакциями.

Команды COMMIT - cохраняет изменения

ROLLBACK - откатывает (отменяет) изменения

SAVEPOINT - создаёт точку к которой группа транзакций может откатиться

SET TRANSACTION - размещает имя транзакции.

• Что такое тупики? Как бороться с тупиками?

Тупик - состояние, когда два (или больше) процесса пытаются обратиться к ресурсу, который заблокирован другим процессом. Поскольку каждый процесс запрашивает ресурс другого процесса, ни один из них не может быть завершен.

Тупики могут быть устранены через использование триггеров.

• Как обеспечивается изолированность транзакций в СУБД?

-Read Uncommitted: ничего не блокируется и не создаются снэпшоты, транзакция просто читает всё.

-Read Committed: обеспечивается блокировкой на запись данных (строк), которые мы пытаемся прочитать. Эта блокировка гарантирует, что мы подождём завершения транзакций, которые уже меняют наши данные, или заставим их подождать, пока мы будем читать. В итоге, мы точно прочитаем только данные, которые были зафиксированы, избежав тем самым грязное чтение.

-Repeatable Read: обеспечивается почти как read commited, за тем исключением, что блокировка на запись работает до конца транзакции, а не отдельной операции. Единожды “коснувшись” блока данных, транзакция блокирует его изменение до конца работы.

-Serializable: обеспечивается блокировкой и на запись, и на чтение любого блока данных, с которым мы работаем. Блокируется даже вставка данных, которые могут попасть в блок, который мы прочитали.

• Как бороться с проблемой фантомов?

Установить уровень транзакций Serializable.

• Что такое журнал транзакций?

Журнал транзакций - основной компонент системы управления базами данных (СУБД). Все изменения в базе данных записываются в журнал транзакций. Используя эту информацию, СУБД может определить какая транзакция какие изменения внесла в данные SQL Server.

• Как обеспечивается постоянство хранения (durability) в СУБД?

СУБД обеспечивают надежность, записывая транзакции в журнал транзакций, который можно повторно обработать для воссоздания состояния системы прямо перед последующим отказом. Транзакция считается совершенной только после того, как она занесена в журнал

• Какие бывают типы триггеров?

-построчный триггер: триггерная функция вызывается один раз для каждой строки, затронутой оператором, запустившим триггер.

-операторный триггер вызывается только один раз при выполнении соответствующего оператора, независимо от количества строк, которые он затрагивает.

Триггеры также классифицируются в соответствии с тем, срабатывают ли они до, после или вместо операции. Они называются триггерами BEFORE, AFTER и INSTEAD OF, соответственно.

• Когда может срабатывать триггер?

Для обычных и сторонних таблиц можно определять триггеры, которые будут срабатывать до или после любой из команд INSERT, UPDATE или DELETE; либо один раз для каждой модифицируемой строки, либо один раз для оператора SQL. Триггеры на UPDATE можно установить так, чтобы они срабатывали, только когда в предложении SET оператора UPDATE упоминаются определённые столбцы. Также триггеры могут срабатывать для операторов TRUNCATE. Если происходит событие триггера, для обработки этого события в установленный момент времени вызывается функция триггера.

• В каком порядке срабатывают триггеры?

Триггеры BEFORE уровня строки срабатывают непосредственно перед обработкой конкретной строки, в то время как триггеры AFTER уровня строки срабатывают в конце работы всего оператора (но до любого из триггеров AFTER уровня оператора). Эти типы триггеров могут определяться только для таблиц и сторонних таблиц. Триггеры INSTEAD OF уровня строки могут определяться только для представлений и срабатывают для каждой строки, сразу после того, как строка представления идентифицирована как нуждающаяся в обработке. Если есть несколько триггеров на одно и то же событие для одной и той же таблицы, то они будут вызываться в алфавитном порядке по имени триггера.

• Можно ли менять порядок срабатывания триггеров?

Нет, нельзя. Так как при создании триггера указывается, когда он будет выполнятся (before,after,instead of)

• Сработает ли триггер, если оператор, выполненный пользователем, не затрагивает ни одну строку таблицы? Если в триггере нет ни before statement секции, ни after statement секции, и оператор не затрагивает ни одну запись, такой триггер не срабатывает

• Продемонстрировать возникновение тупика.

Для того чтобы смоделировать тупик нужно запустить 2 транзакции параллельно и по очереди выполнять взаимоисключающие команды. Прежде чем будет возможно зафиксировать значения, одна из транзакций будет откатана с исключением взаимоблокировки. Например, будем менять зп работникам.

Transaction 1

BEGIN;

BEGIN;

SELECT salary1 FROM dead\_dem WHERE worker\_id = 1 FOR UPDATE;

UPDATE dead\_dem SET salary1 = 100 WHERE worker\_id = 2;

Transaction 2

BEGIN;

SELECT salary1 FROM dead\_dem WHERE worker\_id FOR UPDATE;

UPDATE dead\_dem SET salary1 = 100 WHERE worker\_id = 1;... deadlock

**Вывод**: в ходе выполнения лабораторной работы получил опыт в проектировании и составлении схем баз данных, научился заполнять таблицы и ближе познакомился с механизмом внешних ключей.