**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение**

**Высшего профессионального образования**

**Московский технический университет связи и информатики**

Факультет повышения квалификации

**Лабораторная работа №1**

**по дисциплине**

**«Интеллектуальные системы»**

Выполнил: магистрант гр. 3mpp1901

Иванова Е.В.

Проверил: к.т.н., доцент каф. МКиИТ

Махров С.С.

**Москва 2020**

Цель работы:

Целью работы является изучение реляционных баз данных и типовых запросов CRUD.

Отчет по лабораторной работе № 1

* 1. Описание выбранной предметной области

В качестве предметной области рассматривается и описывается база данных «Страхового агентства». В таблице Dogovor хранится информация о договоре страхования. Эта таблица ссылается на таблицы clienti\_v\_dogovore, Vid\_dogovora, manager и Strahovaya. Таблица Vid\_dogovora определяет вид договора и ссылается на таблицу Vid\_yslygi, в ней хранится информация о видах услуг. Таблица clienti\_v\_dogovore содержит возраст клиента на момент составления договора и ссылается на таблицу client. Таким образом организована связь многие ко многим, значит один клиент может иметь несколько договоров, либо один договор может быть составлен на несколько клиентов. Таблица client содержит информацию о клиенте, и ссылается на таблицы Adres и Documenti, хранящие адреса и информацию о документах. Ссылка от клиента к документам осуществлена через связь многие ко многим, через таблицу documenti\_clienta. В таблице Strahovaya, ссылающуюся на таблицу Adres, хранится информация о страховых компаниях, сотрудничающих с этим агентством. В таблице manager - личная информация о менеджере, составляющем договор, при том один договор может обслуживать один менеджер. Информация о филиале агентства, в котором работает менеджер определена в таблице filial. Эта таблица связана с таблицей Adres.

Во всех таблицах уникальными ключами являются только id, остальные данные не являются уникальными.

* 1. ER — диаграмма insurance\_agency

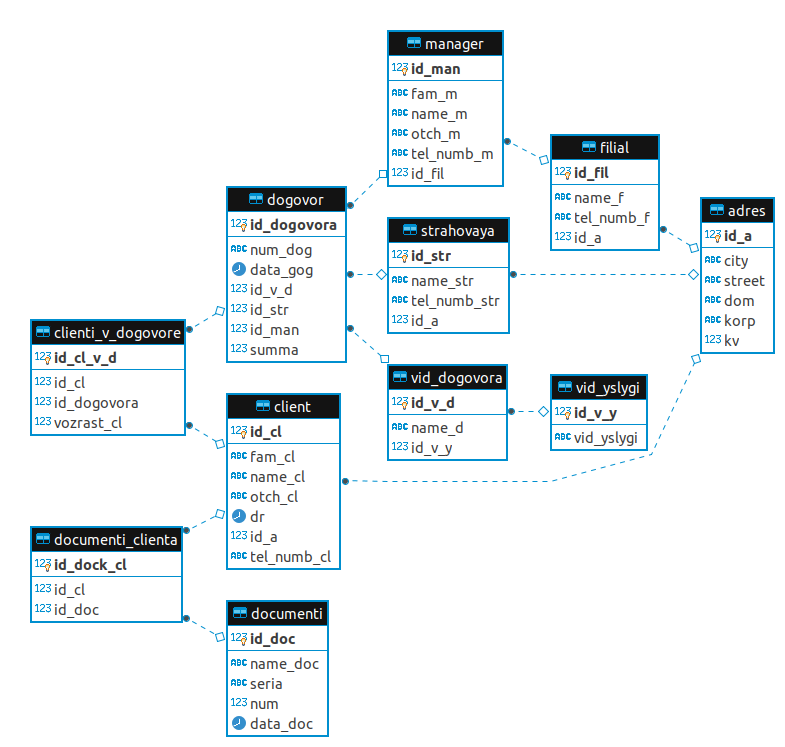


Рис. 1 - ER диаграмма insurance\_agency

Типы данных и описание столбцов указаны в Приложении 1.

* 1. Таблицы заполнены следующими тестовыми данными:

**Vid\_yslygi** (Приложение 2 п.1) – 2 вида услуг, которые оказывает страховое агентство;

**Vid\_dogovora** (Приложение 2 п.2) – 4 вида договора, 2 из которых относятся к 1-му виду услуг, 2 – ко второму;

**Documenti** (Приложение 2 п.3) – 4 документа - 2 паспорта и 2 водительских удостоверения (документы могут быть любыми, справки, разрешения и т.д.);

**Adres** (Приложение 2 п.4) – 6 адресов;

**Strahovaya** (Приложение 2 п.5) – 2 страховые компании, первая имеет первый адрес из таблицы Adres, вторая – второй;

**filial** (Приложение 2 п.6) – 2 филиала страхового агентства, первый филиал расположен по 5-ому адресу из таблицы Adres, второй – по 6-ому;

**manager** (Приложение 2 п.7) – 3 записи с данными о менеджерах, работающих в определенных фиалиалах таблицы filial.

**client** (Приложение 2 п.8) – 3 записи с данными о клиентах, где адрес клиента соответствует определенному адресу из таблицы Adres;

**Dogovor** (Приложение 2 п.9) – 3 записи с данными договоров;

**clienti\_v\_dogovore** (Приложение 2 п.10) – таблица, связывающая таблицы client и Dogovor, связь многие ко многим, т.е. в таблице указывается какому клиенту принадлежит договор и наоборот(на одного клиента можно составить несколько договоров или один договор может быть составлен на нескольких клиентов);

**documenti\_clienta** (Приложение 2 п.11) – таблица, связывающая таблицы client и Documenti, связь многие ко многим, т.е. в таблице указывается какой документы принадлежат клиенту и наоборот (одному человеку может принадлежать несколько документов или у разных людей может быть один и тот же документ).

1. Описание создания Dockerfile.

В контейнере postgres есть каталог docker-entrypoint-initdb.d, на этапе сборки в него помещается скрипт sql (sa.sql), состоящий из создания базы данных, схемы, таблиц и их заполнения. В этом скрипте после создания базы данных необходимо к ней подключить командой: \connect insurance\_agency. В Dockerfile (Приложение 3) в нужную директорию помещается sql скрипт, который после запуска автоматически выполнится. Таким образом в результате запуска Dockerfile строится и заполняется база.

Dockerfile состоит из двух строчек: первая – с каким образом будет осуществлена работа, вторая – путь к скрипту и путь, куда будет помещен скрипт автоматически выполнившись на старте (директория внутри образа).

1. Описание выгрузки Docker-образа.

Docker – образ извлечен и сохранен локально с помощью следующей команды:

sudo docker save -o lr1.tar lr1

Для открытия и использования созданного архива, были изменены права владельца и группы для архива:

sudo chown ekaterina:root lr1.tar

Для загрузки образа и состояния контейнера из архива необходимо выполнить следующую команду:

docker load < lr1.tar

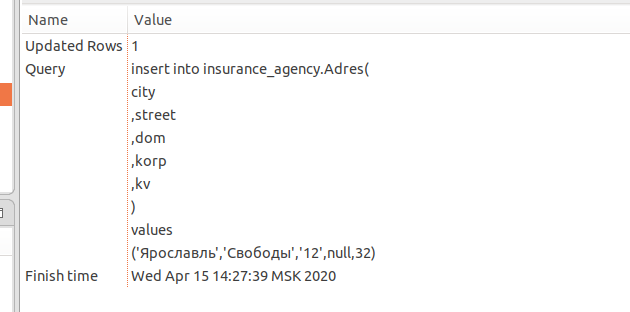
1. CRUD запросы и результаты выполнения
   1. INSERT (создание записей)

insert into insurance\_agency.Adres(city ,street ,dom ,korp ,kv)

values

('Ярославль','Свободы','12',null,32);

Результат выполнения:

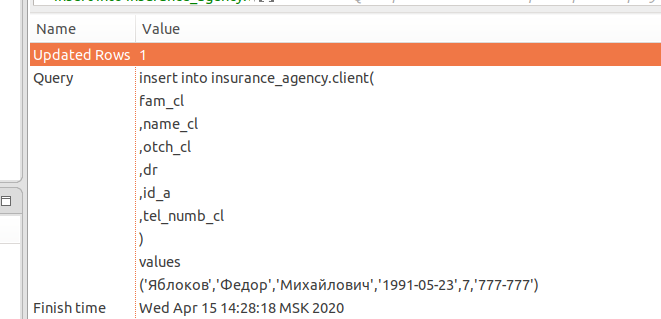


insert into insurance\_agency.client(fam\_cl ,name\_cl ,otch\_cl ,dr ,id\_a ,tel\_numb\_cl)

values

('Яблоков','Федор','Михайлович','1991-05-23',7,'777-777');

Результат выполнения:



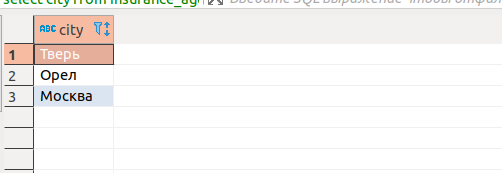
* 1. SELECT (чтение записей)

select city

from insurance\_agency.adres

group by city;

Результат выполнения:



select f.name\_f ,f.tel\_numb\_f ,a.city ,a.street ,a.dom ,a.korp ,m.fam\_m ,m.name\_m ,m.otch\_m

,m.tel\_numb\_m

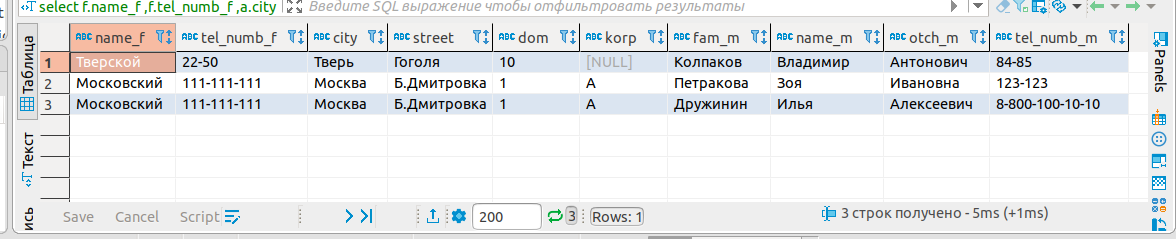
from insurance\_agency.filial f

inner join insurance\_agency.adres a on a.id\_a = f.id\_a

inner join insurance\_agency.manager m on m.id\_fil = f.id\_fil

order by f.name\_f desc;

Результат выполнения:



select cl.fam\_cl F ,cl.name\_cl I ,cl.otch\_cl O ,clvd.vozrast\_cl Vozrast ,d.num\_dog N\_dogovora ,d.summa Summa

from insurance\_agency.client cl

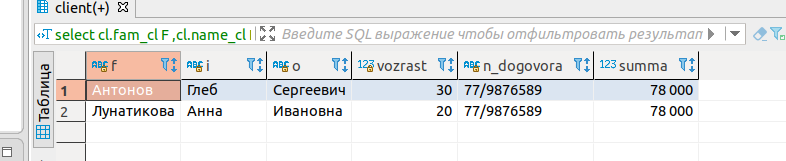
inner join insurance\_agency.clienti\_v\_dogovore clvd on clvd.id\_cl = cl.id\_cl

inner join insurance\_agency.dogovor d on d.id\_dogovora = clvd.id\_dogovora

where d.id\_dogovora =3

order by fam\_cl;

Результат выполнения:



select d.num\_dog ,d.data\_gog ,str.name\_str ,vy.vid\_yslygi ,vd.name\_d ,d.summa ,m.fam\_m

from insurance\_agency.dogovor d

inner join insurance\_agency.manager m on m.id\_man = d.id\_man

inner join insurance\_agency.strahovaya str on str.id\_str = d.id\_str

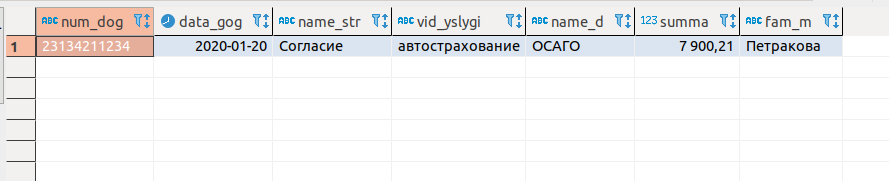
inner join insurance\_agency.vid\_dogovora vd on vd.id\_v\_d = d.id\_v\_d

inner join insurance\_agency.vid\_yslygi vy on vy.id\_v\_y = vd.id\_v\_y

where str.name\_str = 'Согласие' and vy.vid\_yslygi = 'автострахование'

limit 1;

Результат выполнения:



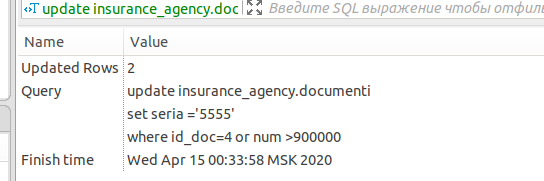
* 1. UPDATE (редактирование записей)

update insurance\_agency.documenti

set seria ='5555'

where id\_doc=4 or num >900000;

Результат выполнения:

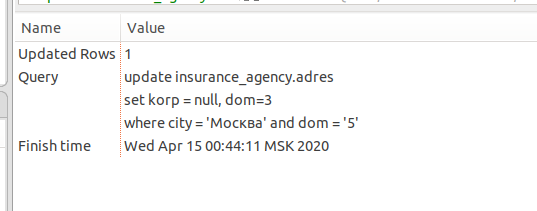


update insurance\_agency.adres

set korp = null, dom=3

where city = 'Москва' and dom = '5';

Результат выполнения:

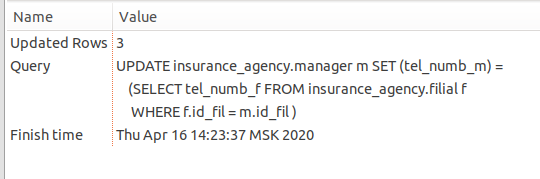


Update insurance\_agency.manager m SET (tel\_numb\_m) =

(Select tel\_numb\_f From insurance\_agency.filial f

Where f.id\_fil = m.id\_fil);

Результат выполнения:

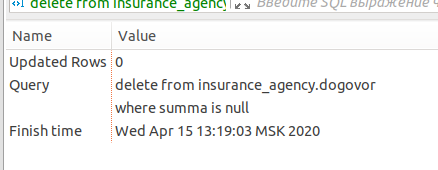


* 1. DELETE (удаление записей)

delete from insurance\_agency.dogovor

where summa is null;

Результат выполнения:

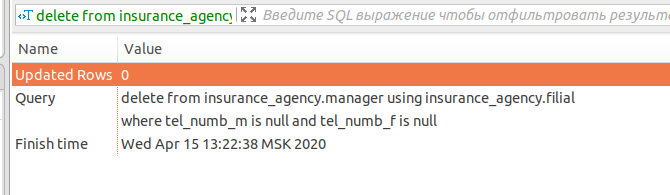


delete from insurance\_agency.manager

using insurance\_agency.filial

where (tel\_numb\_m is null) and (tel\_numb\_f is null);

Результат выполнения:

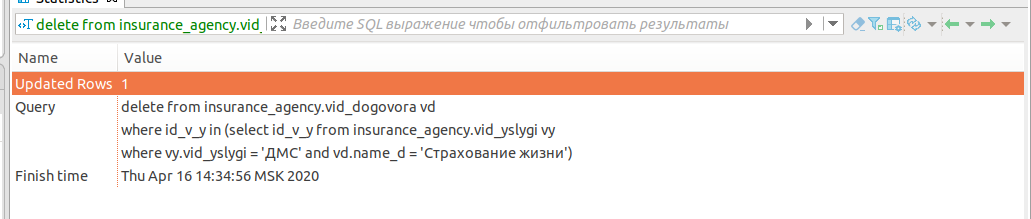


delete from insurance\_agency.vid\_dogovora vd

where id\_v\_y in (select id\_v\_y from insurance\_agency.vid\_yslygi vy

where vy.vid\_yslygi = 'ДМС' and vd.name\_d = 'Страхование жизни')

Результат выполнения:



1. Тестовые запросы SELECT FOR UPDATE и результаты выполнения запроса

select name\_doc ,seria, num

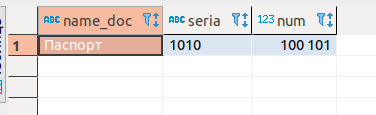
from insurance\_agency.documenti

where name\_doc = 'Паспорт'

LIMIT 1

for update;

Результат выполнения:



select d.num\_dog ,d.data\_gog ,str.name\_str ,vy.vid\_yslygi ,vd.name\_d ,d.summa

from insurance\_agency.dogovor d

inner join insurance\_agency.strahovaya str on str.id\_str = d.id\_str

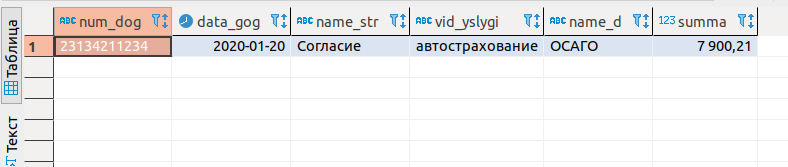
inner join insurance\_agency.vid\_dogovora vd on vd.id\_v\_d = d.id\_v\_d

inner join insurance\_agency.vid\_yslygi vy on vy.id\_v\_y = vd.id\_v\_y

where (d.summa between 1000 and 50000) and str.name\_str = 'Согласие'

for update;

Результат выполнения:



Вывод:

В результате выполнения лабораторной работы, были изучены основы работы с ПО для автоматизации развёртывания и управления приложениями Docker. Появились практические навыки работы с СУБД Postgres, а также ознакомились с процессом развертывания Postgres в контейнере Docker.

Приложение 1

* 1. Таблица Vid\_yslygi

create table insurance\_agency.Vid\_yslygi(

id\_v\_y serial PRIMARY KEY --id\_вида\_услуги

,vid\_yslygi varchar(512) NOT NULL --Вид\_услуги

);

* 1. Таблица Vid\_dogovora

create table insurance\_agency.Vid\_dogovora(

id\_v\_d serial primary key --id\_вида\_договора

,name\_d varchar(512) NOT NULL -- наименование\_вида\_договора

,id\_v\_y int references insurance\_agency.Vid\_yslygi (id\_v\_y) NOT NULL --id\_вида\_услуги\_ссылка\_на\_Vid\_yslygi

);

* 1. Таблица Documenti

create table insurance\_agency.Documenti(

id\_doc serial primary key --id\_документа

,name\_doc varchar(512) NOT NULL -- наименование\_документа

,seria varchar(64)-- серия\_документа

,num int-- номер\_документа

--,Кем\_выдан varchar(512) -- кем\_выдан\_документ

,data\_doc DATE-- дата\_выдачи\_документа

);

* 1. Таблица Adres

create table insurance\_agency.Adres(

id\_a serial primary key --id\_адреса

,city varchar(128) --адрес\_город

,street varchar(512) --адрес\_улица

,dom varchar(32) --адрес\_дом

,korp varchar(32) --адрес\_корпус

,kv int --номер\_квартиры

);

* 1. Таблица Strahovaya

create table insurance\_agency.Strahovaya(

id\_str serial primary key --id\_страховой

,name\_str varchar(512) NOT NULL --наименование\_страховой\_компании

,tel\_numb\_str varchar(128) --телефон\_страховой\_компании

,id\_a int REFERENCES insurance\_agency.Adres (id\_a) NOT NULL

--id\_ссылка\_на\_Адрес

);

* 1. Таблица filial

create table insurance\_agency.filial(

id\_fil serial primary key --id\_филиала

,name\_f varchar(512) NOT NULL --наименование\_филиала

,tel\_numb\_f varchar(128) --номер\_телефона\_филиала

,id\_a int REFERENCES insurance\_agency.Adres (id\_a) NOT NULL

--id\_ссылка\_на\_Адрес

);

* 1. Таблица manager

create table insurance\_agency.manager(

id\_man serial primary key --id\_менеджера

,fam\_m varchar(128) NOT NULL --фамилия\_менеджера

,name\_m varchar(128) NOT NULL--имя\_менеджера

,otch\_m varchar(128)--отчетсво\_менеджера

,tel\_numb\_m varchar(128) --номер\_телефона\_менеджера

,id\_fil int REFERENCES insurance\_agency.filial (id\_fil) NOT NULL

--id\_филиала\_ссылка\_на\_таблицу\_filial

);

* 1. Таблица client

create table insurance\_agency.client(

id\_cl serial primary key --id\_клиента

,fam\_cl varchar(128) NOT NULL --фамилия\_клиента

,name\_cl varchar(128) NOT NULL--имя\_клиента

,otch\_cl varchar(128)--отчетсво\_клиента

,dr date NOT NULL --дата\_рождения\_клиента

,id\_a int REFERENCES insurance\_agency.Adres (id\_a) NOT NULL

--id\_ссылка\_на\_Адрес

,tel\_numb\_cl varchar(128) --номер\_телефона\_клиента

);

* 1. Таблица Dogovor

create table insurance\_agency.Dogovor(

id\_dogovora serial primary key --id\_договора

,num\_dog varchar(128) NOT null--номер\_договора

,data\_gog date NOT NULL--дата\_заключения\_договора

,id\_v\_d int REFERENCES insurance\_agency.vid\_dogovora (id\_v\_d) NOT NULL

--id\_ссылка\_на\_вид\_договора

,id\_str int REFERENCES insurance\_agency.strahovaya (id\_str) NOT NULL

--id\_ссылка\_на\_страховую

,id\_man int REFERENCES insurance\_agency.manager (id\_man) NOT NULL

--id\_ссылка\_на\_менеджера

,summa numeric(10,2) NOT NULL--сумма\_договора

);

* 1. Таблица clienti\_v\_dogovore

create table insurance\_agency.clienti\_v\_dogovore(

id\_cl\_v\_d serial primary key --id\_клиента

,id\_cl int REFERENCES insurance\_agency.client(id\_cl) NOT NULL

--id\_ссылка\_на\_клиента

,id\_dogovora int REFERENCES insurance\_agency.Dogovor (id\_dogovora) NOT NULL --id\_ссылка\_на\_договор

,vozrast\_cl int --возраст\_клиента\_на\_момент\_составления\_договора

);

* 1. Таблица documenti\_clienta

create table insurance\_agency.documenti\_clienta(

id\_dock\_cl serial primary key --id\_документа\_клиента

,id\_cl int REFERENCES insurance\_agency.client(id\_cl) NOT NULL

--id\_ссылка\_на\_клиента

,id\_doc int REFERENCES insurance\_agency.Documenti (id\_doc) NOT NULL

--id\_ссылка\_документ

);

Приложение 2

* + 1. Заполнение таблицы Vid\_yslygi

insert into insurance\_agency.Vid\_yslygi(

vid\_yslygi )

values

('ДМС')

,('автострахование');

1. Заполнение таблицы Vid\_dogovora

insert into insurance\_agency.Vid\_dogovora(

name\_d

,id\_v\_y )

values

('Страхование жизни',1)

,('Страхование от несчастных случаев и болезни',1)

,('ОСАГО',2)

,('КАСКО',2);

1. Заполнение таблицы Documenti

insert into insurance\_agency.Documenti(

name\_doc

,seria

,num

,data\_doc )

values

('Паспорт','1010','100101','2010-08-30')

,('Паспорт','M-1010','150401',null)

,('ВУ',null,'923043', '2019-11-08')

,('ВУ','2111','456789', '2018-01-31');

1. Заполнение таблицы Adres

insert into insurance\_agency.Adres(

city

,street

,dom

,korp

,kv )

values

('Москва','Садовая','5','А',null)

,('Орел','Победы','25',null,null)

,('Москва','Академика Чаломея','19',null,'8')

,('Тверь','Советская','1','1','16')

,('Москва','Б.Дмитровска','1','А',1)

,('Тверь','Гоголя','10',null,'2');

1. Заполнение таблицы Strahovaya

insert into insurance\_agency.Strahovaya(

name\_str

,tel\_numb\_str

,id\_a )

values

('АльфаСтрахование','999-99-99',1)

,('Согласие','8-222-222-22-22',2);

1. Заполнение таблицы filial

insert into insurance\_agency.filial(

name\_f

,tel\_numb\_f

,id\_a )

values

('Московский','111-111-111',5)

,('Тверской','22-50',6);

1. Заполнение таблицы manager

insert into insurance\_agency.manager(

fam\_m

,name\_m

,otch\_m

,tel\_numb\_m

,id\_fil )

values

('Петракова','Зоя','Ивановна','123-123',1)

,('Колпаков','Владимир','Антонович','84-85',2)

,('Дружинин','Илья','Алексеевич','8-800-100-10-10',1);

1. Заполнение таблицы client

insert into insurance\_agency.client(

fam\_cl

,name\_cl

,otch\_cl

,dr

,id\_a

,tel\_numb\_cl )

values

('Антонов','Глеб','Сергеевич','1980-01-01',3,'900-99')

,('Лунатикова','Анна','Ивановна','2000-05-31',3,null)

,('Лимонова','Габриэлла',null,'1960-04-21',4,'878-987-22-11');

1. Заполнение таблицы Dogovor

insert into insurance\_agency.Dogovor(

num\_dog

,data\_gog

,id\_v\_d

,id\_str

,id\_man

,summa )

values

('23134211234','2020-01-20',3,2,1,'7900.21')

,('ТН-1234567','2019-12-03',2,1,2,'19900.21')

,('77/9876589','2019-03-04',4,2,3,'78000');

1. Заполнение таблицы clienti\_v\_dogovore

insert into insurance\_agency.clienti\_v\_dogovore(

id\_cl

,id\_dogovora

,vozrast\_cl )

values

(1,3,'30')

,(2,3,'20')

,(1,1,'30')

,(3,2,'50');

1. Заполнение таблицы documenti\_clienta

insert into insurance\_agency.documenti\_clienta(

id\_cl

,id\_doc )

values

(1,3)

,(2,1)

,(3,2)

,(2,4);

Приложение 3

Dockerfile:

FROM postgres

COPY /sa.sql /docker-entrypoint-initdb.d

Ответы на вопросы:

1. Что такое SQL, DDL, DML, DCL, TCL?

Structured Query Language (SQL) — язык структурированных запросов, с помощью него пишутся специальные запросы (SQL инструкции) к базе данных с целью получения этих данных из базы и для манипулирования этими данными.

Data Definition Language (DDL) – это группа операторов определения данных. Другими словами, с помощью операторов, входящих в эту группы, мы определяем структуру базы данных и работаем с объектами этой базы, т.е. создаем, изменяем и удаляем их.

В эту группу входят следующие операторы:

CREATE – используется для создания объектов базы данных;

ALTER – используется для изменения объектов базы данных;

DROP – используется для удаления объектов базы данных.

Data Manipulation Language (DML) – это группа операторов для манипуляции данными. С помощью этих операторов мы можем добавлять, изменять, удалять и выгружать данные из базы, т.е. манипулировать ими.

В эту группу входят самые распространённые операторы языка SQL:

* SELECT – осуществляет выборку данных;
* INSERT – добавляет новые данные;
* UPDATE – изменяет существующие данные;
* DELETE – удаляет данные.

Data Control Language (DCL) – группа операторов определения доступа к данным. Иными словами, это операторы для управления разрешениями, с помощью них мы можем разрешать или запрещать выполнение определенных операций над объектами базы данных.

Сюда входят:

GRANT – предоставляет пользователю или группе разрешения на определённые операции с объектом;

REVOKE – отзывает выданные разрешения;

DENY– задаёт запрет, имеющий приоритет над разрешением.

Transaction Control Language (TCL) – группа операторов для управления транзакциями. Транзакция – это команда или блок команд (инструкций), которые успешно завершаются как единое целое, при этом в базе данных все внесенные изменения фиксируются на постоянной основе или отменяются, т.е. все изменения, внесенные любой командой, входящей в транзакцию, будут отменены.

Сюда можно отнести:

BEGIN TRANSACTION – служит для определения начала транзакции;

COMMIT TRANSACTION – применяет транзакцию;

ROLLBACK TRANSACTION – откатывает все изменения, сделанные в контексте текущей транзакции;

SAVE TRANSACTION – устанавливает промежуточную точку сохранения внутри транзакции.

1. Что такое CRUD?

CRUD — аббревиатура, обозначающая четыре базовые функции, используемые при работе с [базами данных](https://ru.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85): создание ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) create), чтение (read), модификация (update), удаление (delete). В [SQL](https://ru.wikipedia.org/wiki/SQL) этим функциям операциям соответствуют операторы [Insert](https://ru.wikipedia.org/wiki/Insert_(SQL)) (создание записей), [Select](https://ru.wikipedia.org/wiki/Select_(SQL)) (чтение записей), [Update](https://ru.wikipedia.org/wiki/Update_(SQL)) (редактирование записей), [Delete](https://ru.wikipedia.org/wiki/Delete_(SQL)) (удаление записей).

1. Что такое ER диаграмма?

Диаграмма привязки сущности ER - это блок-схемы, которые иллюстрируют, как «сущности» (люди, объекты или концепции) относятся друг к другу в системе.

1. Зачем нужен оператор SELECT FOR UPDATE?

В режиме FOR UPDATE строки, выданные оператором SELECT, блокируются как для изменения. При этом они защищаются от блокировки, изменения и удаления другими транзакциями до завершения текущей. То есть другие транзакции, пытающиеся выполнить UPDATE, DELETE,

SELECT FOR UPDATE, SELECT FOR NO KEY UPDATE, SELECT FOR SHARE или SELECT FOR KEY SHARE с этими строками, будут заблокированы до завершения текущей транзакции; и наоборот, команда SELECT FOR UPDATE будет ожидать окончания параллельной транзакции, в которой выполнилась одна из этих команд с той же строкой, а затем установит блокировку и вернёт изменённую строку (или не вернёт, если она была удалена).

Режим блокировки FOR UPDATE также запрашивается на уровне строки любой командой DELETE и командой UPDATE, изменяющей значения определённых колонок.

1. Зачем нужен оператор MERGE и есть ли он в PostgreSQL?

Merge — оператор языка SQL, который позволяет слить данные одной таблицы с данными другой таблицы. При слиянии таблиц проверяется условие, и если оно истинно, то выполняется Update, а если нет - Insert. Причём нельзя изменять поля таблицы в секции Update, по которым идет связывание двух таблиц. Данные изменяются или добавляются только для таблицы в предложении MERGE INTO, таблица в предложении USING остается без изменений.

Есть поддержка этого оператора в PostgreSQL 8.4devel Documentation. Присутствует подробная информация, а так же параметры которые используются в данном операторы, получаемые вывод при использовании данного оператора.

1. Какой уровень изоляции транзакций используется по умолчанию в PostgreSQL, какие его особенности и как его поменять?

По умолчанию в PostgreSQL используется уровень изоляции Read Committed. Такой уровень изоляции всегда позволяет видеть изменения внесённые успешно завершёнными транзакциями в оставшихся параллельно открытых транзакциях. В транзакции, работающей на этом уровне, запрос SELECT (без предложения FOR UPDATE/SHARE) видит только те данные, которые были зафиксированы до начала запроса; он никогда не увидит незафиксированных данных или изменений, внесённых в процессе выполнения запроса параллельными транзакциями. По сути запрос SELECT видит снимок базы данных в момент начала выполнения запроса. Однако SELECT видит результаты изменений, внесённых ранее в этой же транзакции, даже если они ещё не зафиксированы. Также два последовательных оператора SELECT могут видеть разные данные даже в рамках одной транзакции, если какие-то другие транзакции зафиксируют изменения после выполнения первого SELECT.

Частичная изоляция транзакций, обеспечиваемая в режиме Read Committed, приемлема для множества приложений. Этот режим быстр и прост в использовании, однако он подходит не для всех случаев. Приложениям, выполняющим сложные запросы и изменения, могут потребоваться более строго согласованное представление данных.

Для выбора нужного уровня изоляции транзакций используется команда SET TRANSACTION.

1. Чем Dockerfile отличается от Docker-образа? Когда нужно писать Dockerfile, а когда выгружать образ?

Dockerfile – это сценарий, который состоит из последовательности команд и аргументов, необходимых для создания образа. Такие сценарии упрощают развёртывание и процесс подготовки приложения к запуску.

Сначала Dockerfile определяет образ, на основе которого будет происходить сборка. Затем идёт ряд методов, команд и аргументов, которые создадут новый образ.

Содержимое Dockerfile передаётся демону Docker для сборки образа.

Образы Docker - это шаблоны для контейнеров. Они разработаны для того, чтобы быть эффективными и предлагать максимальное повторное использование, используя драйвер хранения файловой системы наложения.

Писать Dockerfile нужно в том случае, если нет возможности подключится к интернету (например сервер изолирован от общей сети), чтобы развернуть контейнер. А выгружать можно в том случае, когда необходимо сделать общедоступным контейнер для всех пользователей, которые могут скачать его через Docker Hub (при условии, что они в репозитории).

1. В чем отличие Dockerfile от docker-compose.yml?

Dockerfile – это подробный пошаговый сценарий для автоматизации сборки.

Docker-compose.yml — это как Dockerfile, но для распределенного приложения целиком.

При изучении основ Docker, в лабораторной работе столкнулись с созданием простейших приложений, работающих автономно, не зависящих, например, от внешних источников данных или от неких сервисов. На практике же подобные приложения — редкость. Реальные проекты обычно включают в себя целый набор совместно работающих приложений.

Например, когда создают веб-сайт, которому, для выполнения аутентификации пользователей, нужно подключиться к базе данных. Подобный проект может состоять из двух сервисов — того, что обеспечивает работу сайта, и того, который отвечает за поддержку базы данных.  
 Технология Docker Compose, если описывать её упрощённо, позволяет, с помощью одной команды, запускать множество сервисов.

И отличие заключает в том, что Dockerfile служит для создания простых приложений работающих автономно, не зависящих, например, от внешних источников данных или от неких сервисов. А docker-compose.yml –это совокупность различных контенеров виде сервисов без которых сложное приложение работать не будет.