###### vМИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

###### ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

###### НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

###### Факультет информационных технологий

ОТЧЕТ

О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

«Реализация приложения базы данных в архитектуре клиент-сервер»

студента 3 курса, группы 20206

**Бондаря Дмитрия Андреевича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:

Сергей Александрович Рылов

Новосибирск 2023

Оглавление

[Задание 3](#_Toc136862704)

[Этап 1: проектирование БД 5](#_Toc136862705)

[Описание БД и атрибутов 5](#_Toc136862706)

[Ограничения на стороне клиента и сервера 10](#_Toc136862707)

[Код создания БД 11](#_Toc136862708)

[Этап 2: написание SQL запросов к спроектированной базе данных. 14](#_Toc136862709)

[Этап 3: PL/SQL 23](#_Toc136862710)

[Этап 4: реализация приложения 76](#_Toc136862711)

# Задание

Необходимо реализовать приложение базы данных в архитектуре клиент-сервер по заданию предметной области.

Аптека продает медикаменты и изготавливает их по рецептам. Лекарства могут быть разных

типов:

1. Готовые лекарства: таблетки, мази, настойки:

2. Изготовляемые аптекой: микстуры, мази, растворы, настойки, порошки.

Различие в типах лекарств отражается в различном наборе атрибутов, их характеризующих.

Микстуры и порошки изготавливаются только для внутреннего применения, растворы для наружного, внутреннего применения и для смешивания с другими лекарствами и мази только для наружного применения. Лекарство различны также по способу приготовления и по времени приготовления. Порошки и мази изготавливаются смешиванием различных компонент. При изготовлении растворов и микстур ингредиенты не только смешивают, но и отстаивают с последующей фильтрацией лекарства, что увеличивает время изготовления.

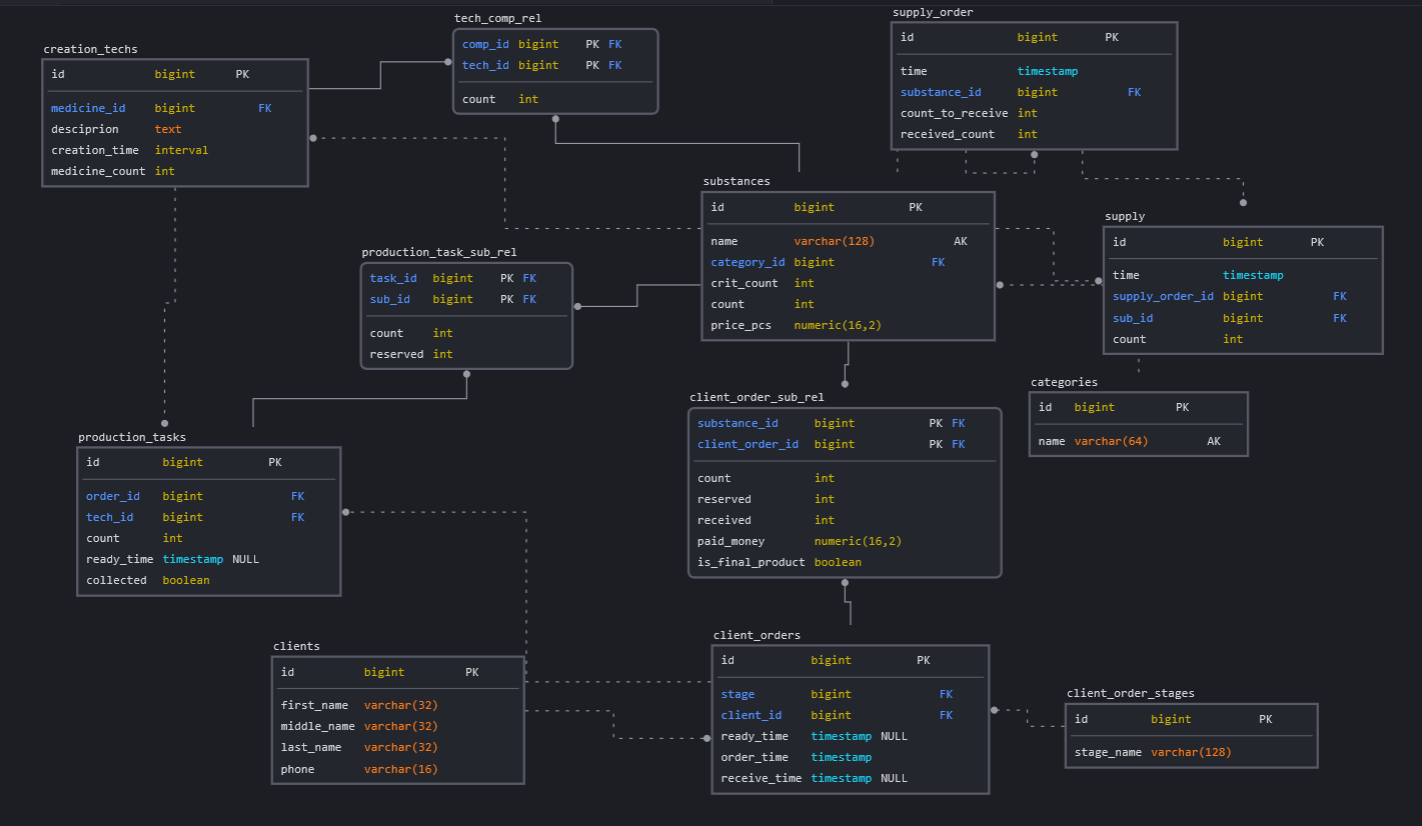
В аптеке существует справочник технологий приготовления различных лекарств. В нем указываются: идентификационный номер технологии, название лекарства и сам способ приготовления. На складе на все медикаменты устанавливается критическая норма, т.е. когда какого-либо вещества на складе меньше критической нормы, то составляются заявки на данные вещества и их в срочном порядке привозят с оптовых складов медикаментов.

Для изготовления аптекой лекарства, больной должен принести рецепт от лечащего врача. В рецепте должно быть указано: ФИО, подпись и печать врача, ФИО, возраст и диагноз пациента, также количество лекарства и способ применения. Больной отдает рецепт регистратору, он принимает заказ и смотрит, есть ли компоненты заказываемого лекарства. Если не все компоненты имеются в наличии, то делает заявки на оптовые склады лекарств и фиксирует ФИО, телефон и адрес необслуженного покупателя, чтобы сообщить ему, когда доставят нужные компоненты. Такой больной пополняет справочник заказов - это те заказы, которые находятся в процессе приготовления, с пометкой, что не все компоненты есть для заказа. Если все компоненты имеются, то они резервируются для лекарства больного. Покупатель выплачивает цену лекарства, ему возвращается рецепт с пометкой о времени изготовления. Больной также пополняет справочник заказов в производстве. В назначенное время больной приходит и по тому же рецепту получает готовое лекарство. Такой больной пополняет список отданных заказов.

Ведется статистика по объемам используемых медикаментов. Через определенный промежуток времени производится инвентаризация склада. Это делается для того, чтобы определить, есть ли лекарства с критической нормой, или вышел срок хранения или недостача.

# Этап 1: проектирование БД

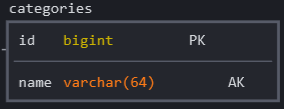
## Описание БД и атрибутов



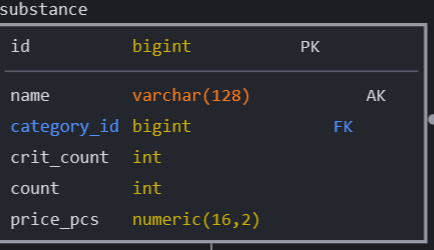
<https://drive.google.com/file/d/1W3EdZPh_xMTOmsMxHcvPnJ5GMSFOyfYC/view?usp=sharing>

Исходный код: https://github.com/Diif/NSU/tree/master/DB

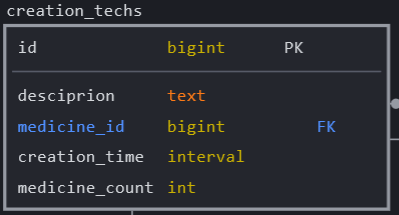
Сущности:



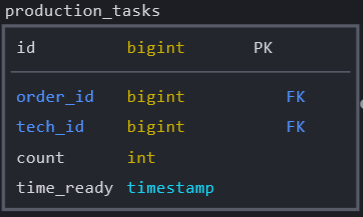
* Категория вещества. Атрибуты:
  + id (PK)
  + имя (уникальное, Not null)



* Вещество (substance) – медикамент или компонент, из которого делается медикамент. Атрибуты
  + Id (PK)
  + имя (уникальное, Not null)
  + id категории (FK, Not null)
  + критическое количество (Not null, >=0)
  + текущая цена (Not null, >=1)
  + количество на складе (Not null, >=0)



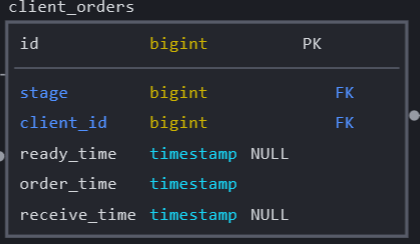
* Технология производства (creation\_techs) – технология производства. Имеет отношение Nк1(tech\_comp\_rel) с компонентами, в котором указывается необходимое для производства количество компонента. Атрибуты:
  + id (PK)
  + текст технологии (Not null)
  + id создаваемого медикамента
  + время, требуемое на создание (Not null)
  + создаваемое технологией количество медикамента



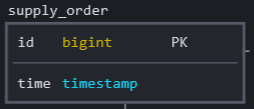
* Задачи производства. Показывает, какие медикаменты и в каком количестве находятся в процессе изготовления. Атрибуты:
  + id (PK)
  + id клиентского заказа (FK)
  + id технологии (FK, Not null)
  + изготавливаемое количество (> 0, Not null)
  + время окончания производства (Not null)



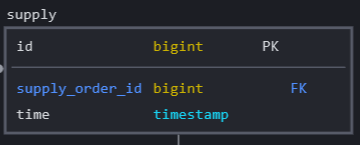
* Клиент. Атрибуты:
  + имя (Not null)
  + отчество (Not null)
  + фамилия (Not null)
  + телефон (Not null)



* Заказы клиентов. Заказы связаны с медикаментами отношением 1кN (client\_order\_sub\_rel), каждая связь имеет количество нужного вещества, количество зарезервированного вещества, количество выданного вещества, выплаченную стоимость, является ли данное вещество товаром для выдачи, или же оно только требуется для производства. Атрибуты:
  + id
  + id стадии заказа (PK)
  + id клиента (FK, Not null)
  + время готовности заказа
  + время создания заказа (Not null)
  + время получения заказа



* Заказ поставки. Отношение с веществами 1кN (supply\_order\_sub\_rel), отношение хранит нужное количество, полученное количество, суммарную стоимость. Атрибуты:
  + id (PK)
  + время создания заказа (Not null)



* Поставка. Отношение с веществами 1кN(supply\_sub\_rel), отношение хранит количества вещества в данной поставке. Атрибуты
  + id
  + id заказа поставки, к которой относится данная поставка (на один заказ может быть несколько поставок)
  + время данной поставки

## Ограничения на стороне клиента и сервера

* client\_orders:
  + timestamp получения заказа должна быть больше или равен даты готовности, даты создания заказа.
  + При наличии receive\_time стадия может быть только «выдано»
  + Порядок изменений статуса
* clients:
  + проверка номера телефона на отсутствие каких-либо символов помимо «+» и цифр
  + проверка ФИО на отсутствие каких-либо символов, кроме букв.

Также не должны существовать заказы, которые не привязаны с веществами для выдачи. Перемещение веществ должно быть согласованным.

## Код создания БД

CREATE SCHEMA IF NOT EXISTS public;

CREATE TABLE categories(

    id bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    name varchar(64) UNIQUE NOT NULL

);

CREATE TABLE substances(

    id bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    name varchar(128) UNIQUE NOT NULL,

    category\_id bigint REFERENCES categories NOT NULL ,

    crit\_count int NOT NULL check (crit\_count >= 0 ),

    count int NOT NULL check ( count >= 0 ),

    price\_pcs numeric(16,2) NOT NULL check (price\_pcs >= 0 )

);

CREATE TABLE creation\_techs(

    id bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    medicine\_id bigint REFERENCES substances NOT NULL ,

    description text NOT NULL,

    creation\_time interval NOT NULL,

    medicine\_count int NOT NULL check ( medicine\_count > 0 )

);

CREATE TABLE tech\_comp\_rel(

    comp\_id bigint REFERENCES substances NOT NULL,

    tech\_id bigint REFERENCES creation\_techs NOT NULL,

    count int NOT NULL check ( count > 0 ),

    PRIMARY KEY (comp\_id, tech\_id)

);

CREATE TABLE clients(

    id bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    first\_name varchar(32) NOT NULL,

    middle\_name varchar(32) NOT NULL,

    last\_name varchar(32) NOT NULL,

    phone varchar(16) NOT NULL UNIQUE

);

CREATE TABLE client\_order\_stages(

    id bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    stage\_name varchar(128) UNIQUE NOT NULL

);

CREATE TABLE client\_orders(

    id bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    stage bigint REFERENCES client\_order\_stages NOT NULL ,

    client\_id bigint REFERENCES clients NOT NULL,

    ready\_time timestamp,

    receive\_time timestamp,

    order\_time timestamp NOT NULL

);

CREATE TABLE production\_tasks(

                       id bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

                       order\_id bigint REFERENCES client\_orders,

                       tech\_id bigint REFERENCES creation\_techs NOT NULL ,

                       count int NOT NULL check ( count > 0 ),

                       ready\_time timestamp,

                       collected boolean NOT NULL DEFAULT False

);

CREATE TABLE production\_task\_sub\_rel(

                       task\_id bigint REFERENCES production\_tasks NOT NULL,

                       sub\_id bigint REFERENCES substances NOT NULL,

                       count int NOT NULL check (count > 0),

                       reserved int NOT NULL check (reserved >= 0),

                       PRIMARY KEY(sub\_id, task\_id)

);

UPDATE production\_tasks

SET order\_id=(select id from client\_orders where client\_id=(select id from clients where phone='+79530684959'))

WHERE ready\_time='28-02-2023 10:23:40';

UPDATE production\_tasks

SET order\_id=(select id from client\_orders where client\_id=(select id from clients where phone='+74862968401'))

WHERE ready\_time='26-02-2023 23:23:40';

CREATE TABLE client\_order\_sub\_rel(

    substance\_id bigint REFERENCES substances NOT NULL ,

    client\_order\_id bigint REFERENCES client\_orders NOT NULL ,

    count int NOT NULL check ( count > 0 ),

    reserved int NOT NULL check ( count >= 0 ),

    received int NOT NULL check ( count >= 0 ),

    paid\_price numeric(16,2) check ( paid\_price >= 0),

    is\_final\_product boolean NOT NULL,

    PRIMARY KEY (substance\_id, client\_order\_id),

    CHECK (

        (received <= reserved)

        AND

        (

            (is\_final\_product=false AND received=0)

            OR

            (is\_final\_product=true)

        )

    )

);

CREATE TABLE supply\_order(

    id bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    time timestamp NOT NULL,

    substance\_id bigint REFERENCES substances NOT NULL ,

    count\_to\_receive int NOT NULL check ( count\_to\_receive > 0 ),

    received\_count int NOT NULL check ( received\_count >= 0 ),

    CHECK (received\_count <= count\_to\_receive)

);

CREATE TABLE supply(

    id bigint GENERATED ALWAYS AS IDENTITY PRIMARY KEY,

    time timestamp NOT NULL,

    supply\_order\_id bigint REFERENCES supply\_order NOT NULL ,

    sub\_id bigint REFERENCES substances NOT NULL,

    count int NOT NULL check ( count > 0 )

);

CREATE TABLE supply\_sub\_rel(

    supply\_id bigint REFERENCES supply NOT NULL ,

    substance\_id bigint REFERENCES substances NOT NULL ,

    count int NOT NULL check ( count > 0 ),

    PRIMARY KEY (substance\_id, supply\_id)

);

# Этап 2: написание SQL запросов к спроектированной базе данных.

1)Получить сведения о покупателях, которые не пришли забрать свой заказ в назначенное им время и общее их число.

SELECT DISTINCT first\_name, last\_name,middle\_name, phone , ROW\_NUMBER() OVER() AS num FROM

(

SELECT DISTINCT client\_id FROM client\_orders

WHERE stage IN

(

SELECT id FROM client\_order\_stages

WHERE stage\_name='готово' OR stage\_name='выдано'

)

AND (((receive\_time IS NOT NULL) AND (CAST(receive\_time AS DATE) > CAST(ready\_time AS DATE))) OR (receive\_time IS NULL AND CAST(ready\_time AS DATE) < CAST(CURRENT\_DATE AS DATE)))

)

AS SQ INNER JOIN clients ON (clients.id = SQ.client\_id)

ORDER BY num DESC

2.1)Получить перечень и общее число покупателей, которые ждут прибытия на склад нужных им медикаментов в целом

SELECT first\_name, last\_name, middle\_name, phone, ROW\_NUMBER() OVER() AS num FROM

(

SELECT first\_name, last\_name, middle\_name, phone FROM clients

WHERE id IN

(

SELECT client\_id FROM client\_orders

WHERE stage=(SELECT id FROM client\_order\_stages WHERE stage\_name='ожидание')

)

)SQ1

ORDER BY num DESC

2.2) и по указанной категории медикаментов

SELECT first\_name, last\_name, middle\_name, phone, ROW\_NUMBER() OVER() AS num FROM

(

SELECT DISTINCT client\_id FROM

(

SELECT client\_order\_id FROM

(

SELECT substance\_id, client\_order\_id FROM client\_order\_sub\_rel

WHERE count!=reserved AND is\_final\_product=true

) AS SQ1

INNER JOIN substances ON (substances.id = substance\_id)

WHERE substances.category\_id=(SELECT id FROM categories WHERE name='таблетки')

GROUP BY client\_order\_id

) AS SQ2

INNER JOIN client\_orders ON (client\_order\_id = client\_orders.id)

WHERE stage=(SELECT id FROM client\_order\_stages WHERE stage\_name='ожидание')

) AS SQ3

INNER JOIN clients ON (clients.id = SQ3.client\_id)

ORDER BY num DESC

3.1)Получить перечень десяти наиболее часто используемых медикаментов в целом

SELECT name, total\_count FROM

(

SELECT \*, RANK() OVER(ORDER BY total\_count DESC)FROM

(

SELECT substances.id, substances.name, SUM(client\_order\_sub\_rel.count) total\_count FROM substances

INNER JOIN client\_order\_sub\_rel

ON (substances.id = client\_order\_sub\_rel.substance\_id)

GROUP BY substances.id, substances.name

) SQ

) SQ2

WHERE SQ2.rank <= 10

3.1) и указанной категории медикаментов.

SELECT name, total\_count FROM

(

SELECT \*, RANK() OVER(ORDER BY total\_count DESC)FROM

(

SELECT substances.id, substances.name, SUM(client\_order\_sub\_rel.count) total\_count FROM substances

INNER JOIN client\_order\_sub\_rel

ON (substances.id = client\_order\_sub\_rel.substance\_id)

WHERE category\_id=(SELECT id FROM categories WHERE name='таблетки')

GROUP BY substances.id, substances.name

) SQ

) SQ2

WHERE SQ2.rank <= 10

4)Получить какой объем указанных веществ использован за указанный период.

SELECT substances.name, COUNT(client\_order\_sub\_rel.count) AS total\_count FROM

(

SELECT id FROM client\_orders

WHERE order\_time BETWEEN '2002-01-26 10:23:40' AND '2024-02-28 10:23:40'

) AS SQ1

INNER JOIN client\_order\_sub\_rel ON (SQ1.id = client\_order\_sub\_rel.client\_order\_id)

INNER JOIN substances ON (client\_order\_sub\_rel.substance\_id = substances.id)

WHERE name IN ('аспирин')

GROUP BY substances.id, substances.name

ORDER BY total\_count DESC

LIMIT 10

5.1)Получить перечень и общее число покупателей, заказывавших определенное лекарство за данный период.

SELECT first\_name, last\_name, middle\_name, phone, ROW\_NUMBER() OVER() num FROM clients

WHERE id IN

(

SELECT client\_id FROM client\_orders

WHERE id IN

(

SELECT client\_order\_id FROM client\_order\_sub\_rel

WHERE substance\_id=

(

SELECT id FROM substances

WHERE name='пихкофф'

)

)

AND order\_time BETWEEN '2023-01-26 10:23:40' AND '2023-02-28'

)

ORDER BY num DESC

5.2) или определенные типы лекарств за данный период.

SELECT first\_name, last\_name, middle\_name, phone, ROW\_NUMBER() OVER() num FROM clients

WHERE id IN

(

SELECT client\_id FROM client\_orders

WHERE id IN

(

SELECT client\_order\_id FROM client\_order\_sub\_rel

WHERE substance\_id IN

(

SELECT id FROM substances

WHERE category\_id=

(

SELECT id FROM categories

WHERE name='таблетки'

)

)

)

AND order\_time BETWEEN '2023-01-26 10:23:40' AND '2023-02-28'

)

ORDER BY num DESC

6)Получить перечень и типы лекарств, достигших своей критической нормы или закончившихся

SELECT name, crit\_count, count FROM substances

WHERE count<=crit\_count

7.1)Получить перечень лекарств с минимальным запасом на складе в целом

SELECT \* FROM substances

WHERE count <=

(

SELECT MIN(count) FROM substances

)

7.2) и по указанной категории медикаментов.

SELECT \* FROM substances

WHERE count <=

(

SELECT MIN(count) FROM substances

WHERE category\_id=

(

SELECT id FROM categories

WHERE name='таблетки'

)

)

8) Получить полный перечень и общее число заказов находящихся в производстве.

SELECT client\_id, ready\_time, receive\_time, order\_time, ROW\_NUMBER() OVER() num FROM client\_orders

WHERE stage=

(

SELECT id FROM client\_order\_stages

WHERE stage\_name='изготовление'

)

ORDER BY num DESC

9) Получить полный перечень и общее число препаратов требующихся для заказов, находящихся в производстве.

SELECT \*, ROW\_NUMBER() OVER() num FROM substances

WHERE id IN

(

SELECT DISTINCT medicine\_id FROM creation\_techs

WHERE id IN

(

SELECT DISTINCT tech\_id FROM production\_tasks

WHERE ready\_time <= clock\_timestamp()

)

)

ORDER BY num DESC

10.1)Получить все технологии приготовления лекарств указанных типов

SELECT \* FROM creation\_techs

WHERE medicine\_id IN

(

SELECT id FROM substances

WHERE category\_id IN (SELECT id FROM categories WHERE name IN ('таблетки', 'настойки'))

)

10.2)конкретных лекарств

SELECT \* FROM creation\_techs

WHERE medicine\_id IN

(

SELECT id FROM substances

WHERE name IN ('пихкофф')

)

10.3)лекарств, находящихся в справочнике заказов в производстве.

SELECT \* FROM creation\_techs

WHERE id IN

(

SELECT DISTINCT tech\_id FROM production\_tasks

WHERE ready\_time <= clock\_timestamp()

)

11)Получить сведения о ценах на указанное лекарство в готовом виде, об объеме и ценах на

все компоненты, требующиеся для этого лекарства.

SELECT med\_name, med\_price, med\_id, tech\_id, comp\_id, name, price\_pcs as comp\_price FROM

(

SELECT med\_name, med\_price, med\_id, SQ2.tech\_id, comp\_id FROM

(

SELECT med\_name, med\_price, med\_id,creation\_techs.id as tech\_id FROM

(

SELECT name as med\_name, price\_pcs as med\_price, id as med\_id FROM substances

WHERE name='пихкофф'

) SQ1

INNER JOIN creation\_techs

ON (SQ1.med\_id = creation\_techs.medicine\_id)

) SQ2

INNER JOIN tech\_comp\_rel

ON (SQ2.tech\_id = tech\_comp\_rel.tech\_id)

) SQ3

INNER JOIN substances

ON (substances.id = SQ3.comp\_id)

12.1)Получить сведения о наиболее часто делающих заказы клиентах на медикаменты

определенного типа.

SELECT client\_id, COUNT(client\_order\_id) as count FROM

(

SELECT client\_order\_id FROM client\_order\_sub\_rel

WHERE substance\_id IN

(

SELECT id FROM substances

WHERE category\_id=

(

SELECT id FROM categories

WHERE name='таблетки'

)

)

) SQ1

INNER JOIN client\_orders

ON (client\_orders.id = SQ1.client\_order\_id)

GROUP BY client\_id

ORDER BY count DESC

12.2)на конкретные медикаменты.

SELECT client\_id, COUNT(client\_order\_id) as count FROM

(

SELECT client\_order\_id FROM client\_order\_sub\_rel

WHERE substance\_id=

(

SELECT id FROM substances

WHERE name='пихкофф'

)

) SQ1

INNER JOIN client\_orders

ON (client\_orders.id = SQ1.client\_order\_id)

GROUP BY client\_id

ORDER BY count DESC

13.1)Получить сведения о конкретном лекарстве: его тип, цена, его количество на складе.

SELECT med\_name, count, med\_price, categories.name AS category\_name FROM

(

SELECT name AS med\_name, count, price\_pcs AS med\_price, category\_id FROM substances

WHERE name='пихкофф'

) SQ1

INNER JOIN categories

ON (SQ1.category\_id = categories.id)

13.2)способ приготовления, названия всех компонент, их цены

SELECT tech\_desc, name, price\_pcs as comp\_price FROM

(

SELECT description as tech\_desc, comp\_id FROM

(

SELECT creation\_techs.id as tech\_id, creation\_techs.description FROM

(

SELECT id as med\_id FROM substances

WHERE name='пихкофф'

) SQ1

INNER JOIN creation\_techs

ON (SQ1.med\_id = creation\_techs.medicine\_id)

) SQ2

INNER JOIN tech\_comp\_rel

ON (SQ2.tech\_id = tech\_comp\_rel.tech\_id)

) SQ3

INNER JOIN substances

ON (substances.id = SQ3.comp\_id)

# Этап 3: PL/SQL

1. Проверка на имя/фамилию/отчество/номер юзера

CREATE OR REPLACE FUNCTION valid\_phone (num IN varchar) RETURNS BOOLEAN

LANGUAGE SQL

IMMUTABLE

RETURN num SIMILAR TO '\+7[0-9]{10}';

CREATE OR REPLACE FUNCTION valid\_alphabet (str IN varchar) RETURNS BOOLEAN

LANGUAGE SQL

IMMUTABLE

RETURN str NOT SIMILAR TO '%[^А-Яа-я]+%';

CREATE OR REPLACE FUNCTION trigger\_for\_client\_insert\_update()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS

$$

DECLARE

ok bool;

BEGIN

ok = valid\_phone(NEW.phone);

IF NOT ok THEN

RAISE EXCEPTION 'Bad phone number format.'

using errcode = 'BDNUM';

END IF;

ok = valid\_alphabet(NEW.first\_name);

IF NOT ok THEN

RAISE EXCEPTION 'Bad first name.'

using errcode = 'BDSTR';

END IF;

ok = valid\_alphabet(NEW.middle\_name);

IF NOT ok THEN

RAISE EXCEPTION 'Bad middle name.'

using errcode = 'BDSTR';

END IF;

ok = valid\_alphabet(NEW.last\_name);

IF NOT ok THEN

RAISE EXCEPTION 'Bad last name.'

using errcode = 'BDSTR';

END IF;

RETURN NEW;

END;

$$;

CREATE OR REPLACE TRIGGER check\_update\_insert\_clients

BEFORE INSERT OR UPDATE

ON

clients

FOR EACH ROW

EXECUTE PROCEDURE trigger\_for\_client\_insert\_update();

1. Проверка на новые заказы и их обновления: при добавлении проставляется только время заказа, ставится 1 стадия. При обновлении различные проверки.

CREATE OR REPLACE FUNCTION trigger\_for\_client\_orders\_insert()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS

$$

DECLARE

fst\_stage constant int = 1;

BEGIN

NEW.stage = 1;

NEW.order\_time = now();

NEW.ready\_time = NULL;

NEW.receive\_time = NULL;

RETURN NEW;

END;

$$;

CREATE OR REPLACE TRIGGER check\_client\_orders\_insert

BEFORE INSERT

ON

client\_orders

FOR EACH ROW

EXECUTE PROCEDURE trigger\_for\_client\_orders\_insert();

1. Тригер на обновление заказа

CREATE OR REPLACE FUNCTION trigger\_for\_client\_orders\_update()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS

$$

DECLARE

last\_stage constant int = 5;

BEGIN

IF NEW.stage < OLD.stage THEN

RAISE EXCEPTION 'Invalid state change.'

USING errcode = 'BDSTG';

END IF;

IF NEW.order\_time != OLD.order\_time THEN

RAISE EXCEPTION 'Can''t change order time.'

USING errcode = 'BDTME';

END IF;

IF NEW.ready\_time IS NOT NULL AND NEW.ready\_time < OLD.order\_time THEN

RAISE EXCEPTION 'Invalid ready time.'

USING errcode = 'BDTME';

END IF;

IF NEW.receive\_time IS NOT NULL AND NEW.receive\_time < NEW.ready\_time THEN

RAISE EXCEPTION 'Invalid receive time.'

USING errcode = 'BDTME';

END IF;

IF NEW.receive\_time IS NOT NULL AND NEW.stage != last\_stage THEN

RAISE EXCEPTION 'Received time only possible with the last stage.'

USING errcode = 'BDSTG';

END IF;

RETURN NEW;

END;

$$;

CREATE OR REPLACE TRIGGER check\_client\_orders\_update

BEFORE UPDATE

ON

client\_orders

FOR EACH ROW

EXECUTE PROCEDURE trigger\_for\_client\_orders\_update();

1. Функция для первичного создания заказа с добавлением к нему заказываемых ингрединтов/компонентов. Предполагается, что оплата уже была произведена, т.к. в столбец "paid price" записывается: цена\_товара\*количество\_товара. Процедура требует уникальности в substances\_to\_add (и проверяет соблюдение этого требования). Возвращает id нового заказа

CREATE OR REPLACE FUNCTION create\_order(client\_id bigint, substances\_to\_add bigint ARRAY, sub\_count int ARRAY)

RETURNS bigint

LANGUAGE plpgsql

AS

$$

DECLARE

sub\_len constant int = array\_length(substances\_to\_add, 1);

count\_len constant int = array\_length(sub\_count, 1);

counter int = sub\_len;

cur\_sub\_id bigint;

cur\_count int;

order\_id bigint;

total\_price numeric(16,2);

BEGIN

IF sub\_len IS NULL OR count\_len IS NULL THEN

RAISE EXCEPTION 'Invalid array(s) sizes: % and %', sub\_len, count\_len

USING errcode = 'NULSZ';

END IF;

IF sub\_len < 1 OR count\_len < 1 THEN

RAISE EXCEPTION 'Invalid array(s) sizes: % and %', sub\_len, count\_len

USING errcode = 'BADSZ';

END IF;

IF sub\_len != count\_len THEN

RAISE EXCEPTION '''count'' and ''substances\_to\_add'' arrays are not equal: % and %',sub\_len, count\_len

USING errcode = 'DIFSZ';

END IF;

IF sub\_len != array\_length(ARRAY(SELECT DISTINCT \* FROM unnest(substances\_to\_add)), 1) THEN

RAISE EXCEPTION 'Substances id must be unique.'

USING errcode = 'NTUNQ';

END IF;

SELECT id

INTO client\_id

FROM clients

WHERE id = client\_id;

IF client\_id IS NULL THEN

RAISE EXCEPTION 'Bad client id.'

USING errcode = 'BADCL';

END IF;

INSERT INTO client\_orders VALUES(default, 1, client\_id, NULL, NULL, NULL) RETURNING id INTO order\_id;

LOOP

EXIT WHEN counter = 0;

cur\_count := sub\_count[counter];

IF cur\_count IS NULL OR cur\_count < 1 THEN

RAISE EXCEPTION 'Invalid substance count %.', cur\_count

USING errcode = 'BADSB';

END IF;

SELECT id, price\_pcs

INTO cur\_sub\_id, total\_price

FROM substances

WHERE id = substances\_to\_add[counter];

IF cur\_sub\_id IS NULL THEN

RAISE EXCEPTION 'Invalid substance id: %', substances\_to\_add[counter]

USING errcode = 'BADSB';

END IF;

INSERT INTO client\_order\_sub\_rel VALUES(cur\_sub\_id, order\_id, cur\_count, 0,0,total\_price \* cur\_count, True);

counter := counter - 1;

END LOOP;

RETURN order\_id;

END;

$$;

1. Функция минимума двух чисел:

CREATE OR REPLACE FUNCTION min(num1 int, num2 int)

RETURNS int

LANGUAGE plpgsql

AS

$$

BEGIN

IF num1 > num2 THEN

RETURN num2;

ELSE

RETURN num1;

END IF;

END;

$$;

1. Процедура для автоматического резервирования лекарств. Пробегает по всем запрашиваемым лекарствам, смотрит их на складе и меняет кол-во на складе, кол-во резервированного.

CREATE OR REPLACE PROCEDURE reserve\_for\_order(order\_id bigint)

LANGUAGE plpgsql

AS

$$

DECLARE

cur\_rel REFCURSOR;

cur\_sub\_id bigint;

stored\_count int;

max\_to\_reserve int;

cur\_reserved int;

to\_reserve int;

BEGIN

SELECT client\_order\_id

INTO order\_id

FROM client\_order\_sub\_rel

WHERE client\_order\_id = order\_id;

IF order\_id IS NULL THEN

RAISE EXCEPTION 'Invalid order id.'

USING errcode='BADRD';

END IF;

OPEN cur\_rel FOR

SELECT substance\_id, count, reserved FROM client\_order\_sub\_rel

WHERE client\_order\_id = order\_id

FOR NO KEY UPDATE;

FETCH cur\_rel INTO cur\_sub\_id, max\_to\_reserve, cur\_reserved;

LOOP

EXIT WHEN cur\_sub\_id IS NULL;

to\_reserve = max\_to\_reserve - cur\_reserved;

SELECT count FROM substances

WHERE id = cur\_sub\_id

FOR NO KEY UPDATE

INTO stored\_count;

to\_reserve := min(to\_reserve, stored\_count);

UPDATE substances

SET count = count - to\_reserve

WHERE id = cur\_sub\_id;

UPDATE client\_order\_sub\_rel

SET reserved = reserved + to\_reserve

WHERE CURRENT OF cur\_rel;

FETCH cur\_rel INTO cur\_sub\_id, max\_to\_reserve, cur\_reserved;

END LOOP;

CLOSE cur\_rel;

END;

$$;

1. Процедура для автоматического создания производственных заказов. Таски БЕЗ ингредиентов, т.е. эту штуку запускать в отдельной транзакции нельзя, только вместе со следующей.

CREATE OR REPLACE PROCEDURE create\_production\_tasks\_for\_order(order\_id bigint)

LANGUAGE plpgsql

AS

$$

DECLARE

cur\_rel REFCURSOR;

cur\_sub\_id bigint;

stored\_count int;

max\_to\_reserve int;

cur\_reserved int;

for\_production int;

tech\_id bigint;

count\_per\_tech int;

task\_id bigint;

BEGIN

SELECT client\_order\_id

INTO order\_id

FROM client\_order\_sub\_rel

WHERE client\_order\_id = order\_id;

IF order\_id IS NULL THEN

RAISE EXCEPTION 'Invalid order id.'

USING errcode='BADRD';

END IF;

OPEN cur\_rel FOR

SELECT substance\_id, count, reserved FROM client\_order\_sub\_rel

WHERE client\_order\_id = order\_id AND reserved!=count;

FETCH cur\_rel INTO cur\_sub\_id, max\_to\_reserve, cur\_reserved;

LOOP

EXIT WHEN cur\_sub\_id IS NULL;

for\_production = max\_to\_reserve - cur\_reserved;

SELECT id, medicine\_count FROM creation\_techs

WHERE medicine\_id = cur\_sub\_id

INTO tech\_id, count\_per\_tech;

IF tech\_id IS NULL THEN

FETCH cur\_rel INTO cur\_sub\_id, max\_to\_reserve, cur\_reserved;

CONTINUE;

END IF;

for\_production = CEIL(CAST(for\_production AS FLOAT) / count\_per\_tech);

INSERT INTO production\_tasks VALUES(default,order\_id, tech\_id, count\_per\_tech \* for\_production, NULL);

FETCH cur\_rel INTO cur\_sub\_id, max\_to\_reserve, cur\_reserved;

END LOOP;

CLOSE cur\_rel;

END;

$$;

1. Процедура создания необходимых для производства ингредиентов.

CREATE OR REPLACE PROCEDURE create\_sub\_rel\_for\_production\_tasks(\_order\_id bigint)

LANGUAGE plpgsql

AS

$$

DECLARE

cur\_rel REFCURSOR;

row\_to\_insert production\_task\_sub\_rel%ROWTYPE;

cur\_sub\_id bigint;

stored\_count int;

max\_to\_reserve int;

cur\_reserved int;

for\_production int;

count\_per\_tech int;

BEGIN

SELECT client\_order\_id

INTO \_order\_id

FROM client\_order\_sub\_rel

WHERE client\_order\_id = \_order\_id;

IF \_order\_id IS NULL THEN

RAISE EXCEPTION 'Invalid order id.'

USING errcode='BADRD';

END IF;

OPEN cur\_rel FOR

SELECT SQ1.task\_id, comp\_id, count \* SQ1.task\_count AS comp\_count, 0 AS reserved FROM

tech\_comp\_rel

INNER JOIN

(SELECT id AS task\_id, tech\_id, count AS task\_count FROM production\_tasks

WHERE production\_tasks.order\_id = \_order\_id) SQ1

ON (SQ1.tech\_id = tech\_comp\_rel.tech\_id);

FETCH cur\_rel INTO row\_to\_insert;

LOOP

EXIT WHEN row\_to\_insert IS NULL;

INSERT INTO production\_task\_sub\_rel (SELECT row\_to\_insert.\*);

FETCH cur\_rel INTO row\_to\_insert;

END LOOP;

CLOSE cur\_rel;

END;

$$;

1. Процедура для объединения двух предыдущих процедур:

CREATE OR REPLACE PROCEDURE first\_prepare\_for\_production(\_order\_id bigint)

LANGUAGE plpgsql

AS

$$

BEGIN

CALL create\_production\_tasks\_for\_order(\_order\_id);

CALL create\_sub\_rel\_for\_production\_tasks(\_order\_id);

END;

$$;

1. Процедура резервирования ингредиентов для производства:

CREATE OR REPLACE PROCEDURE reserve\_for\_tasks\_by\_order(\_order\_id bigint)

LANGUAGE plpgsql

AS

$$

DECLARE

cur\_rel REFCURSOR;

cur\_task REFCURSOR;

cur\_sub\_id bigint;

cur\_task\_id bigint;

stored\_count int;

max\_to\_reserve int;

cur\_reserved int;

to\_reserve int;

BEGIN

SELECT client\_order\_id

INTO \_order\_id

FROM client\_order\_sub\_rel

WHERE client\_order\_id = \_order\_id;

IF \_order\_id IS NULL THEN

RAISE EXCEPTION 'Invalid order id.'

USING errcode='BADRD';

END IF;

OPEN cur\_task FOR

SELECT id FROM production\_tasks

WHERE \_order\_id = order\_id;

FETCH cur\_task INTO cur\_task\_id;

LOOP

EXIT WHEN cur\_task\_id IS NULL;

OPEN cur\_rel FOR

SELECT sub\_id, count, reserved FROM production\_task\_sub\_rel

WHERE task\_id = cur\_task\_id

FOR NO KEY UPDATE;

FETCH cur\_rel INTO cur\_sub\_id, max\_to\_reserve, cur\_reserved;

LOOP

EXIT WHEN cur\_sub\_id IS NULL;

to\_reserve = max\_to\_reserve - cur\_reserved;

SELECT count FROM substances

WHERE id = cur\_sub\_id

FOR NO KEY UPDATE

INTO stored\_count;

to\_reserve := min(to\_reserve, stored\_count);

UPDATE substances

SET count = count - to\_reserve

WHERE id = cur\_sub\_id;

UPDATE production\_task\_sub\_rel

SET reserved = reserved + to\_reserve

WHERE CURRENT OF cur\_rel;

FETCH cur\_rel INTO cur\_sub\_id, max\_to\_reserve, cur\_reserved;

END LOOP;

CLOSE cur\_rel;

FETCH cur\_task INTO cur\_task\_id;

END LOOP;

CLOSE cur\_task;

END;

$$;

1. Процедура по переводу в стадию "ожидание"

CREATE OR REPLACE PROCEDURE move\_to\_waiting\_stage(\_order\_id bigint)

LANGUAGE plpgsql

AS

$$

BEGIN

UPDATE client\_orders

SET stage = (SELECT id FROM client\_order\_stages WHERE stage\_name='ожидание')

WHERE id = \_order\_id;

END;

$$;

1. Процедура по запуску производств при наличии необходимых компонентов.

CREATE OR REPLACE PROCEDURE start\_production\_tasks\_by\_order(\_order\_id bigint)

LANGUAGE plpgsql

AS

$$

DECLARE

cur\_task REFCURSOR;

cur\_task\_id bigint;

tech\_time timestamp;

cur\_tech\_id bigint;

BEGIN

SELECT client\_order\_id

INTO \_order\_id

FROM client\_order\_sub\_rel

WHERE client\_order\_id = \_order\_id;

IF \_order\_id IS NULL THEN

RAISE EXCEPTION 'Invalid order id.'

USING errcode='BADRD';

END IF;

OPEN cur\_task FOR

SELECT id, tech\_id FROM production\_tasks

WHERE \_order\_id = order\_id AND ready\_time IS NULL

FOR NO KEY UPDATE;

FETCH cur\_task INTO cur\_task\_id, cur\_tech\_id;

LOOP

EXIT WHEN cur\_task\_id IS NULL;

IF (SELECT task\_id FROM production\_task\_sub\_rel WHERE task\_id = cur\_task\_id AND count != reserved LIMIT 1) IS NULL THEN

UPDATE production\_tasks

SET ready\_time = (SELECT creation\_time FROM creation\_techs WHERE id = cur\_tech\_id) + now()

WHERE CURRENT OF cur\_task;

END IF;

FETCH cur\_task INTO cur\_task\_id, cur\_tech\_id;

END LOOP;

CLOSE cur\_task;

END;

$$;

1. Процедуры для перевода в стадию "изготовление", если всё производство стартануло и зарезервированы все лекарства, которые нельзя произвести.

CREATE OR REPLACE PROCEDURE move\_to\_in\_progress\_stage\_if\_ok(\_order\_id bigint)

LANGUAGE plpgsql

AS

$$

DECLARE

cur\_task REFCURSOR;

cur\_task\_id bigint;

tech\_time timestamp;

cur\_tech\_id bigint;

in\_progress\_stage constant bigint = (SELECT id FROM client\_order\_stages WHERE stage\_name = 'изготовление');

cur\_stage bigint;

BEGIN

SELECT id, stage

INTO \_order\_id, cur\_stage

FROM client\_orders

WHERE id = \_order\_id;

IF \_order\_id IS NULL THEN

RAISE EXCEPTION 'Invalid order id.'

USING errcode='BADRD';

END IF;

IF cur\_stage > in\_progress\_stage THEN

RAISE EXCEPTION 'Can not move stage backward.'

USING errcode='BDSTG';

END IF;

IF

(SELECT id FROM production\_tasks

WHERE \_order\_id = order\_id AND ready\_time IS NULL)

IS NOT NULL

THEN

RETURN;

END IF;

IF

(SELECT substance\_id FROM

(SELECT substance\_id FROM client\_order\_sub\_rel

WHERE client\_order\_id = \_order\_id AND count != reserved) SQ1

LEFT JOIN

creation\_techs

ON (medicine\_id = substance\_id)

WHERE medicine\_id IS NULL

LIMIT 1) IS NOT NULL

THEN

RETURN;

END IF;

UPDATE client\_orders

SET stage = in\_progress\_stage

WHERE id = \_order\_id;

END;

$$;

1. Процедура для автоматического расчета времени готовности.

CREATE OR REPLACE PROCEDURE calculate\_ready\_time(\_order\_id bigint)

LANGUAGE plpgsql

AS

$$

DECLARE

uncreatable\_sub bigint;

not\_reserved\_sub bigint;

\_ready\_time timestamp;

num\_tasks int = 0;

BEGIN

SELECT id

INTO \_order\_id

FROM client\_orders

WHERE id = \_order\_id;

IF \_order\_id IS NULL THEN

RAISE EXCEPTION 'Invalid order id.'

USING errcode='BADRD';

END IF;

SELECT SQ1.substance\_id FROM

(

SELECT substance\_id FROM client\_order\_sub\_rel

WHERE count != reserved AND client\_order\_id = \_order\_id

) SQ1

LEFT JOIN

creation\_techs

ON (medicine\_id = substance\_id)

WHERE id IS NULL

LIMIT 1

INTO uncreatable\_sub;

IF uncreatable\_sub IS NOT NULL THEN

RETURN;

END IF;

SELECT production\_task\_sub\_rel.sub\_id FROM

(

SELECT id FROM production\_tasks

WHERE order\_id = \_order\_id

) SQ1

INNER JOIN

production\_task\_sub\_rel

ON (SQ1.id = task\_id)

WHERE count != reserved

LIMIT 1

INTO not\_reserved\_sub;

IF not\_reserved\_sub IS NOT NULL THEN

RETURN;

END IF;

SELECT MAX(ready\_time), COUNT(id) FROM production\_tasks

WHERE order\_id = \_order\_id

INTO \_ready\_time, num\_tasks;

IF num\_tasks != 0 AND \_ready\_time IS NULL THEN

RAISE EXCEPTION 'Bad data in DB.'

USING errcode='XX001';

END IF;

IF num\_tasks = 0 THEN

UPDATE client\_orders

SET ready\_time = now(),

stage = stage+1

WHERE id = \_order\_id;

ELSE

UPDATE client\_orders

SET ready\_time = now()

WHERE id = \_order\_id;

END IF;

END;

$$;

1. Тригер заказа по достижению критического количества.

CREATE OR REPLACE PROCEDURE order\_substance\_if\_no\_order\_exist(\_sub\_id bigint)

LANGUAGE PLPGSQL

AS

$$

DECLARE

to\_order int;

BEGIN

SELECT id, crit\_count

INTO \_sub\_id, to\_order

FROM substances

WHERE id = \_sub\_id;

IF \_sub\_id IS NULL THEN

RAISE EXCEPTION 'Invalid sub id.'

USING errcode='BADSB';

END IF;

IF (SELECT id FROM supply\_order WHERE substance\_id = \_sub\_id AND received\_count != count\_to\_receive LIMIT 1) IS NOT NULL THEN

RETURN;

END IF;

INSERT INTO supply\_order VALUES(default, now(), \_sub\_id, to\_order \* 2, 0 );

END;

$$;

CREATE OR REPLACE FUNCTION trigger\_for\_substances\_update()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS

$$

BEGIN

IF NEW.crit\_count < NEW.count THEN

CALL order\_substance\_if\_no\_order\_exist(NEW.id);

END IF;

RETURN NEW;

END;

$$;

CREATE OR REPLACE TRIGGER check\_substances\_update

AFTER UPDATE

ON

substances

FOR EACH ROW

EXECUTE PROCEDURE trigger\_for\_substances\_update();

1. Тригер для получения поставки:

CREATE OR REPLACE FUNCTION trigger\_for\_supply\_insert()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS

$$

DECLARE

sub\_id bigint;

cur\_order REFCURSOR;

client\_order\_id bigint;

BEGIN

SELECT substance\_id

INTO sub\_id

FROM supply\_order

WHERE id = NEW.supply\_order\_id;

UPDATE supply\_order

SET received\_count = received\_count + NEW.count

WHERE id = NEW.supply\_order\_id;

UPDATE substances

SET count = count + NEW.count

WHERE id = sub\_id;

OPEN cur\_order FOR

SELECT id FROM client\_orders

WHERE ready\_time IS NULL;

FETCH cur\_order INTO client\_order\_id;

LOOP

EXIT WHEN client\_order\_id IS NULL;

CALL reserve\_for\_order(client\_order\_id) ;

CALL reserve\_for\_tasks\_by\_order(client\_order\_id);

CALL start\_production\_tasks\_by\_order(client\_order\_id);

CALL move\_to\_in\_progress\_stage\_if\_ok(client\_order\_id);

CALL calculate\_ready\_time(client\_order\_id);

FETCH cur\_order INTO client\_order\_id;

END LOOP;

RETURN NEW;

END;

$$;

CREATE OR REPLACE TRIGGER check\_supply\_insert

AFTER INSERT

ON

supply

FOR EACH ROW

EXECUTE PROCEDURE trigger\_for\_supply\_insert();

1. Функция поиска по номеру телефона:

CREATE OR REPLACE FUNCTION search\_clients(\_phone varchar)

RETURNS SETOF clients

LANGUAGE SQL

AS

$$

SELECT \* FROM clients WHERE phone LIKE ('%' || \_phone || '%')

ORDER BY last\_name, first\_name, middle\_name

LIMIT 10;

$$;

1. Поиск заказов:

CREATE OR REPLACE FUNCTION search\_orders(\_client\_id bigint)

RETURNS SETOF client\_orders

LANGUAGE SQL

AS

$$

SELECT \* FROM client\_orders

WHERE client\_id = \_client\_id

ORDER BY ready\_time, order\_time;

$$;

1. Поиск лекарств заказа:

CREATE OR REPLACE FUNCTION search\_client\_order\_sub\_rel(\_order\_id bigint)

RETURNS table(sub\_name varchar, count int)

LANGUAGE SQL

AS

$$

SELECT name, SQ1.count FROM

substances

INNER JOIN

(

SELECT substance\_id, count FROM client\_order\_sub\_rel

WHERE client\_order\_id =\_order\_id

) SQ1

ON (id = substance\_id)

$$;

1. Добавление клиента:

CREATE OR REPLACE PROCEDURE insert\_client(name varchar, middle varchar, last varchar, phone varchar)

LANGUAGE SQL

AS

$$

INSERT INTO clients VALUES(default, name, middle, last, phone);

$$;

1. Процедура обновления клиента.

CREATE OR REPLACE PROCEDURE update\_client(\_id bigint, \_fst\_name varchar,\_middle\_name varchar, \_last\_name varchar, \_phone varchar)

LANGUAGE SQL

AS

$$

UPDATE clients

SET

first\_name = \_fst\_name,

middle\_name = \_middle\_name,

last\_name = \_last\_name,

phone = \_phone

WHERE id = \_id;

$$;

1. Поиск лекарств по имени:

CREATE OR REPLACE FUNCTION search\_substances(\_name varchar)

RETURNS SETOF substances

LANGUAGE SQL

AS

$$

SELECT id, name, category\_id, crit\_count, count, price\_pcs FROM substances

WHERE LOWER(name) LIKE ('%' || LOWER(\_name) || '%')

ORDER BY name

LIMIT 10;

$$;

1. Процедура, которая пробегает по производствам и перносит готовые лекарства в заказы и склад.

CREATE OR REPLACE PROCEDURE check\_production()

LANGUAGE plpgsql

AS

$$

DECLARE

cur\_with\_info REFCURSOR;

cur\_time constant timestamp = now();

\_sub\_id bigint;

\_order\_id bigint;

\_task\_id bigint;

to\_storage int;

to\_order int;

BEGIN

OPEN cur\_with\_info FOR

SELECT task\_id,medicine\_id AS sub\_id, order\_id, ready\_count - count + reserved AS count\_to\_storage, count - reserved AS count\_to\_order FROM client\_order\_sub\_rel

INNER JOIN

(

SELECT order\_id, medicine\_id, medicine\_count \* PT.count AS ready\_count, task\_id FROM

creation\_techs

INNER JOIN

(

SELECT order\_id, tech\_id, count, id AS task\_id FROM production\_tasks

WHERE collected = false AND ready\_time < cur\_time

) PT

ON (tech\_id = id)

) SQ1

ON (substance\_id = medicine\_id) AND (client\_order\_id = order\_id)

FOR NO KEY UPDATE;

FETCH cur\_with\_info INTO \_task\_id, \_sub\_id, \_order\_id, to\_storage, to\_order;

LOOP

EXIT WHEN \_sub\_id IS NULL;

UPDATE substances

SET count = count + to\_storage

WHERE id = \_sub\_id;

UPDATE client\_order\_sub\_rel

SET reserved = reserved + to\_order

WHERE substance\_id = \_sub\_id AND client\_order\_id = \_order\_id;

UPDATE production\_tasks

SET collected = true

WHERE id = \_task\_id;

FETCH cur\_with\_info INTO \_task\_id, \_sub\_id, \_order\_id, to\_storage, to\_order;

END LOOP;

CLOSE cur\_with\_info;

END;

$$;

1. Процедура, для перемещения в стадию "готово". Просто пробегает по заказам с ненулевым временем готовности.

CREATE OR REPLACE PROCEDURE move\_to\_ready\_stage()

LANGUAGE SQL

AS

$$

UPDATE client\_orders

SET stage = stage + 1

WHERE stage = 3 AND ready\_time <= now();

$$;

1. Процедура по получению заказа:

CREATE OR REPLACE PROCEDURE move\_order\_to\_received\_stage(\_order\_id bigint)

LANGUAGE SQL

AS

$$

UPDATE client\_orders

SET stage = 5,

receive\_time = now()

WHERE id = \_order\_id AND stage = 4 AND ready\_time IS NOT NULL;

$$;

1. Процедура обновления лекарства.

CREATE OR REPLACE PROCEDURE update\_sub(\_sub\_id bigint, \_name varchar, \_category\_name varchar,\_crit int, \_price numeric(16,2))

LANGUAGE PLPGSQL

AS

$$

DECLARE

\_category\_id bigint;

BEGIN

SELECT id FROM categories

WHERE name = \_category\_name

INTO \_category\_id;

IF \_category\_id IS NULL THEN

INSERT INTO categories VALUES(default, \_category\_name) RETURNING id INTO \_category\_id;

END IF;

UPDATE substances

SET

name = \_name,

crit\_count = \_crit,

price\_pcs = \_price,

category\_id = \_category\_id

WHERE id = \_sub\_id;

END;

$$;

1. Процедура добавления лекарства. Должна добавлять категории, если такой не существует

CREATE OR REPLACE PROCEDURE create\_sub(\_name varchar, \_category\_name varchar, \_crit int, \_price numeric(16,2))

LANGUAGE PLPGSQL

AS

$$

DECLARE

category\_id bigint;

BEGIN

SELECT id FROM categories

WHERE name = \_category\_name

INTO category\_id;

IF category\_id IS NULL THEN

INSERT INTO categories VALUES(default, \_category\_name) RETURNING id INTO category\_id;

END IF;

INSERT INTO substances VALUES (default, LOWER(\_name), category\_id, \_crit, 0, \_price);

END;

$$;

1. Функция поиска технологии по названию лекарства:

CREATE OR REPLACE FUNCTION search\_tech(\_name varchar)

RETURNS TABLE(tech\_id bigint, sub\_id bigint, sub\_name varchar, description varchar, creation\_time interval, count int)

LANGUAGE SQL

AS

$$

SELECT creation\_techs.id, medicine\_id, name, description, creation\_time, medicine\_count

FROM creation\_techs

INNER JOIN

(

SELECT id, name FROM substances

WHERE name = \_name

) SQ1

ON (SQ1.id = medicine\_id);

$$;

1. Процедура обновления технологии:

CREATE OR REPLACE PROCEDURE update\_tech(\_tech\_id bigint, \_sub\_name varchar, \_desc varchar, \_time interval, \_count int)

LANGUAGE PLPGSQL

AS

$$

DECLARE

\_sub\_id bigint;

BEGIN

SELECT id FROM creation\_techs

WHERE id = \_tech\_id

INTO \_tech\_id;

IF \_tech\_id IS NULL THEN

RAISE EXCEPTION 'Bad tech id.'

using errcode = 'BDTCH';

END IF;

SELECT id FROM substances

WHERE name = \_sub\_name

INTO \_sub\_id;

IF \_sub\_id IS NULL THEN

RAISE EXCEPTION 'Bad substance id.'

using errcode = 'BDSUB';

END IF;

UPDATE creation\_techs

SET

description = \_desc,

medicine\_id = \_sub\_id,

creation\_time = \_time,

medicine\_count = \_count

WHERE id = \_tech\_id;

END;

$$;

1. Добавление технологии:

CREATE OR REPLACE PROCEDURE create\_tech(med\_name varchar,\_desc varchar, \_time interval, \_count int, substances\_to\_add bigint ARRAY, sub\_count int ARRAY)

LANGUAGE plpgsql

AS

$$

DECLARE

sub\_len constant int = array\_length(substances\_to\_add, 1);

count\_len constant int = array\_length(sub\_count, 1);

counter int = sub\_len;

cur\_sub\_id bigint;

cur\_count int;

tech\_id bigint;

med\_id bigint;

total\_price numeric(16,2);

BEGIN

IF sub\_len IS NULL OR count\_len IS NULL THEN

RAISE EXCEPTION 'Invalid array(s) sizes: % and %', sub\_len, count\_len

USING errcode = 'NULSZ';

END IF;

IF sub\_len < 1 OR count\_len < 1 THEN

RAISE EXCEPTION 'Invalid array(s) sizes: % and %', sub\_len, count\_len

USING errcode = 'BADSZ';

END IF;

IF sub\_len != count\_len THEN

RAISE EXCEPTION '''count'' and ''substances\_to\_add'' arrays are not equal: % and %',sub\_len, count\_len

USING errcode = 'DIFSZ';

END IF;

IF sub\_len != array\_length(ARRAY(SELECT DISTINCT \* FROM unnest(substances\_to\_add)), 1) THEN

RAISE EXCEPTION 'Substances id must be unique.'

USING errcode = 'NTUNQ';

END IF;

SELECT id FROM substances

WHERE name = med\_name

INTO med\_id;

INSERT INTO creation\_techs VALUES(default, med\_id, \_desc, \_time, \_count) RETURNING id INTO tech\_id;

LOOP

EXIT WHEN counter = 0;

cur\_count := sub\_count[counter];

IF cur\_count IS NULL OR cur\_count < 1 THEN

RAISE EXCEPTION 'Invalid substance count %.', cur\_count

USING errcode = 'BADSB';

END IF;

SELECT id

INTO cur\_sub\_id

FROM substances

WHERE id = substances\_to\_add[counter];

IF cur\_sub\_id IS NULL THEN

RAISE EXCEPTION 'Invalid substance id: %', substances\_to\_add[counter]

USING errcode = 'BADSB';

END IF;

INSERT INTO tech\_comp\_rel VALUES(cur\_sub\_id,tech\_id, cur\_count);

counter := counter - 1;

END LOOP;

END;

$$;

1. удаление технологии:

CREATE OR REPLACE PROCEDURE remove\_tech(\_id bigint)

LANGUAGE SQL

AS

$$

DELETE FROM creation\_techs

WHERE id = \_id;

$$;

1. Поиск ингредиентов для технологии

CREATE OR REPLACE FUNCTION search\_tech\_comp\_rel(\_tech\_id bigint)

RETURNS TABLE(id bigint, name varchar, count int)

LANGUAGE SQL

AS

$$

SELECT id, name, SQ1.count FROM substances

INNER JOIN

(

SELECT comp\_id, count FROM tech\_comp\_rel

WHERE tech\_id = \_tech\_id

) SQ1

ON (substances.id = SQ1.comp\_id)

$$;

1. Обновление ингредиентов для технологии:

CREATE OR REPLACE PROCEDURE update\_tech\_rels(\_tech\_id bigint, substances\_to\_add bigint ARRAY, sub\_count int ARRAY)

LANGUAGE plpgsql

AS

$$

DECLARE

sub\_len constant int = array\_length(substances\_to\_add, 1);

count\_len constant int = array\_length(sub\_count, 1);

counter int = sub\_len;

cur\_sub\_id bigint;

cur\_count int;

BEGIN

IF (sub\_len IS NOT NULL AND sub\_len < 1) OR (count\_len IS NOT NULL AND count\_len < 1) THEN

RAISE EXCEPTION 'Invalid array(s) sizes: % and %', sub\_len, count\_len

USING errcode = 'BADSZ';

END IF;

IF sub\_len != count\_len THEN

RAISE EXCEPTION '''count'' and ''substances\_to\_add'' arrays are not equal: % and %',sub\_len, count\_len

USING errcode = 'DIFSZ';

END IF;

IF sub\_len != array\_length(ARRAY(SELECT DISTINCT \* FROM unnest(substances\_to\_add)), 1) THEN

RAISE EXCEPTION 'Substances id must be unique.'

USING errcode = 'NTUNQ';

END IF;

DELETE FROM tech\_comp\_rel

WHERE tech\_id = \_tech\_id;

IF sub\_len IS NULL THEN

RETURN;

END IF;

LOOP

EXIT WHEN counter = 0;

cur\_count := sub\_count[counter];

IF cur\_count IS NULL OR cur\_count < 1 THEN

RAISE EXCEPTION 'Invalid substance count %.', cur\_count

USING errcode = 'BADSB';

END IF;

SELECT id

INTO cur\_sub\_id

FROM substances

WHERE id = substances\_to\_add[counter];

IF cur\_sub\_id IS NULL THEN

RAISE EXCEPTION 'Invalid substance id: %', substances\_to\_add[counter]

USING errcode = 'BADSB';

END IF;

INSERT INTO tech\_comp\_rel VALUES(cur\_sub\_id,tech\_id, cur\_count);

counter := counter - 1;

END LOOP;

END;

$$;

1. Поиск заказов на поставку:

CREATE OR REPLACE FUNCTION search\_supply\_orders()

RETURNS TABLE(id bigint, \_time timestamp, sub\_id bigint, count\_to\_receive bigint, received\_count bigint, sub\_name varchar)

LANGUAGE SQL

AS

$$

SELECT SQ1.id AS id, time, substance\_id, count\_to\_receive, received\_count, name AS sub\_name

FROM

(

SELECT id, time, substance\_id, count\_to\_receive, received\_count

FROM supply\_order

WHERE count\_to\_receive != received\_count

) SQ1

INNER JOIN substances

ON (substances.id = substance\_id)

ORDER by time

$$;

1. Процедура создания заказа на поставку

CREATE OR REPLACE PROCEDURE create\_supply\_order(\_sub\_name varchar, \_count int)

LANGUAGE PLPGSQL

AS

$$

DECLARE

\_sub\_id bigint;

BEGIN

SELECT id FROM substances

WHERE name = \_sub\_name

INTO \_sub\_id;

IF \_sub\_id IS NULL THEN

RAISE EXCEPTION 'Bad substance id.'

using errcode = 'BDSUB';

END IF;

INSERT INTO supply\_order VALUES (default, now(), \_sub\_id, \_count,0);

END;

$$;

1. Процедура добавления поставки

CREATE OR REPLACE PROCEDURE create\_supply(\_order\_id bigint, \_count int)

LANGUAGE PLPGSQL

AS

$$

DECLARE

\_sub\_id bigint;

BEGIN

SELECT id FROM supply\_order

WHERE id = \_order\_id

INTO \_order\_id;

IF \_order\_id IS NULL THEN

RAISE EXCEPTION 'Bad supply order id.'

using errcode = 'BDSRD';

END IF;

INSERT INTO supply VALUES (default, now(), \_order\_id, \_count);

END;

$$;

1. Итоговая процедура-обертка для добавления заказа:

CREATE OR REPLACE PROCEDURE create\_order\_wrapper(client\_id bigint, substances\_to\_add bigint ARRAY, sub\_count int ARRAY)

LANGUAGE plpgsql

AS

$$

DECLARE

order\_id bigint;

BEGIN

SELECT create\_order(client\_id, substances\_to\_add, sub\_count) INTO order\_id;

CALL reserve\_for\_order(order\_id);

CALL first\_prepare\_for\_production(order\_id);

CALL reserve\_for\_tasks\_by\_order(order\_id);

CALL move\_to\_waiting\_stage(order\_id);

CALL start\_production\_tasks\_by\_order(order\_id);

CALL move\_to\_in\_progress\_stage\_if\_ok(order\_id);

CALL calculate\_ready\_time(order\_id);

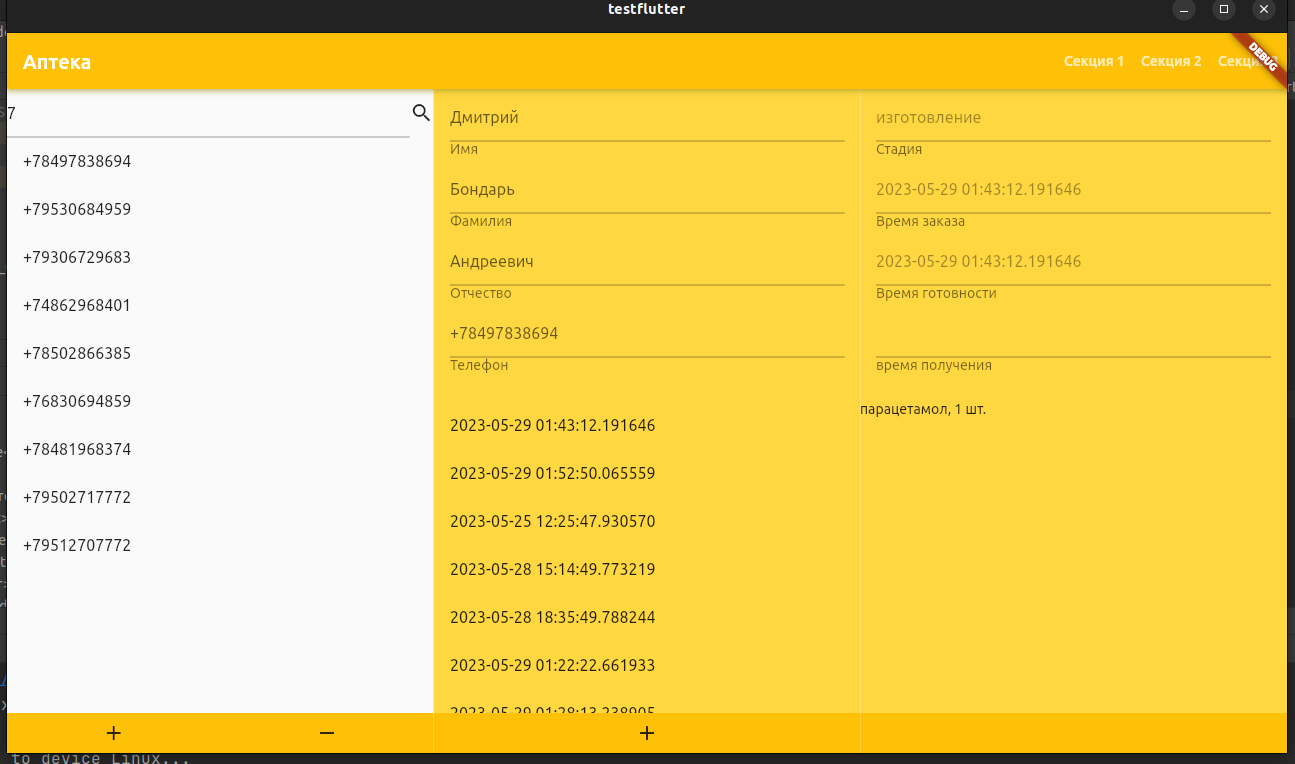
END;

$$;

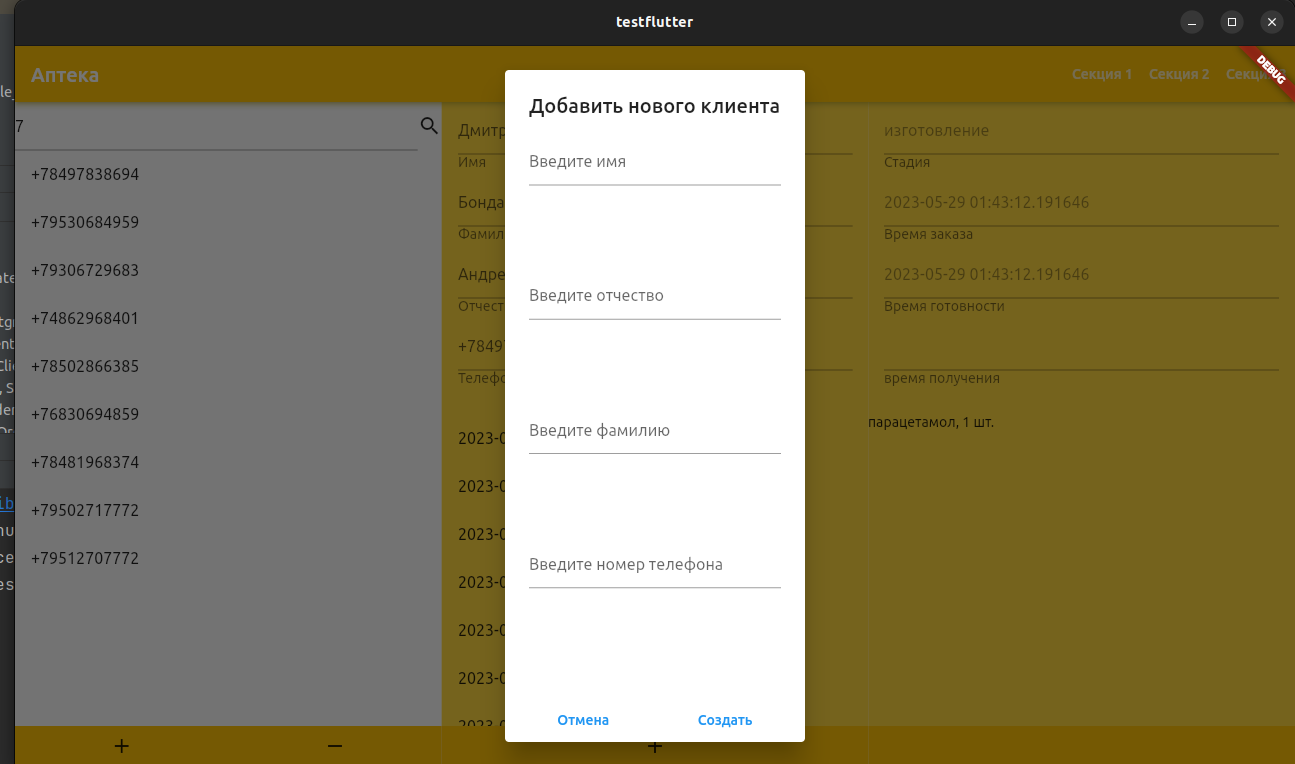
# Этап 4: реализация приложения

С помощью языка Dart и фреймворка Flutter был написан клиент, который способен вносить основные изменения в БД (ограничения функциональности связаны с сохранением целостности, в частности внешних ключей).

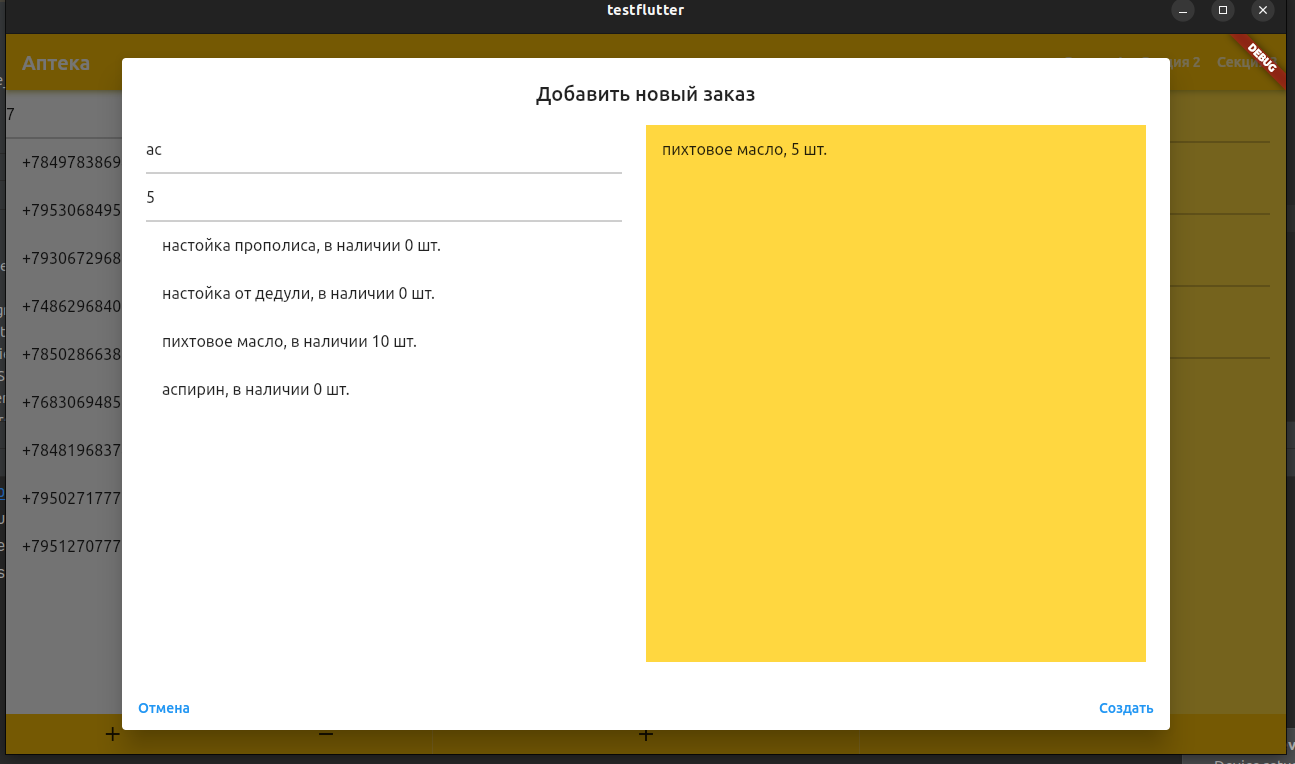
Главная страница, на которой можно найти, добавить, изменить клиента, найти/создать заказ, выдать заказ.



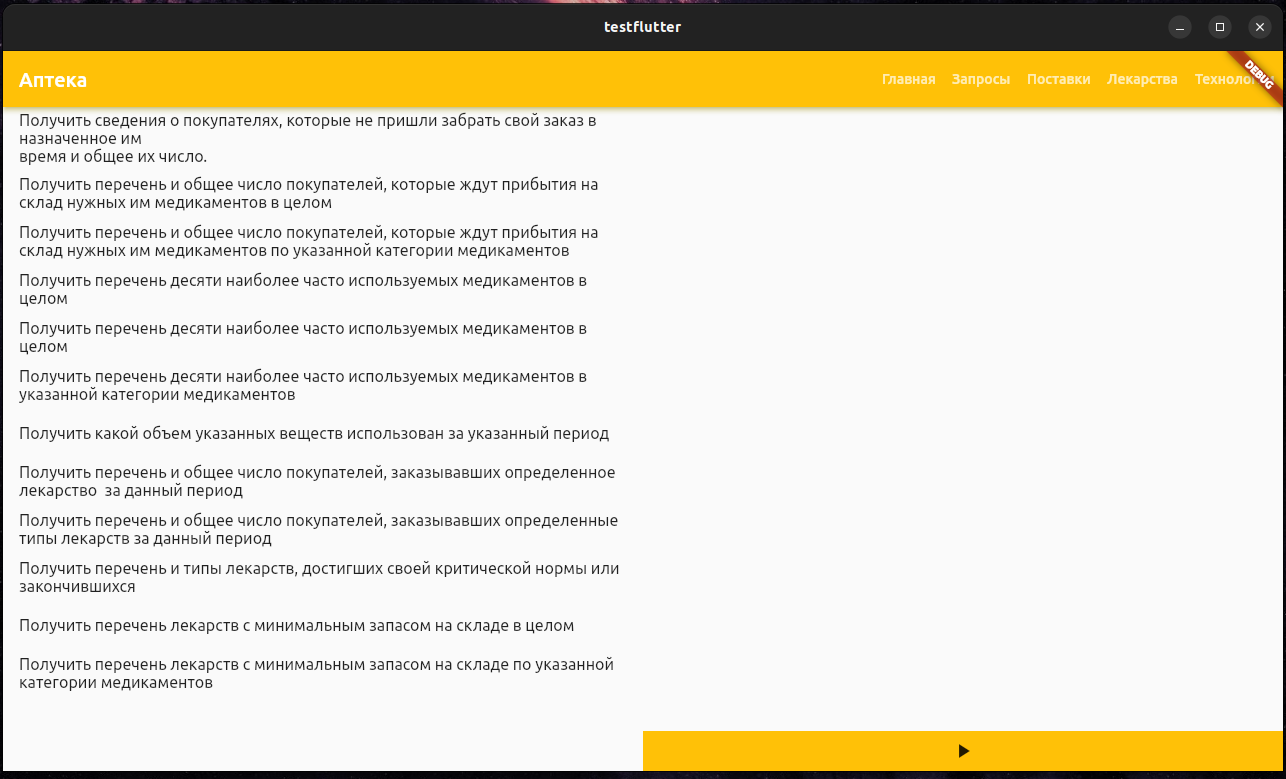
Добавление клиента:



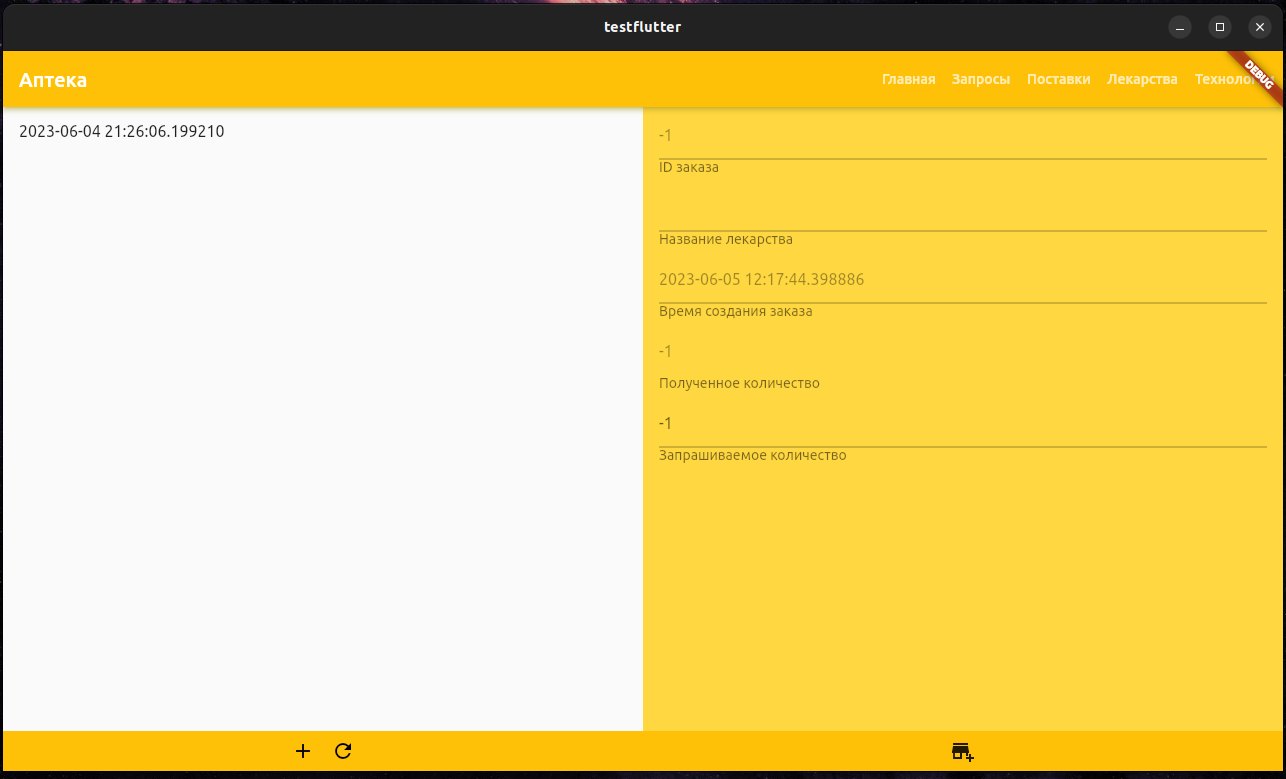
Добавление заказа:



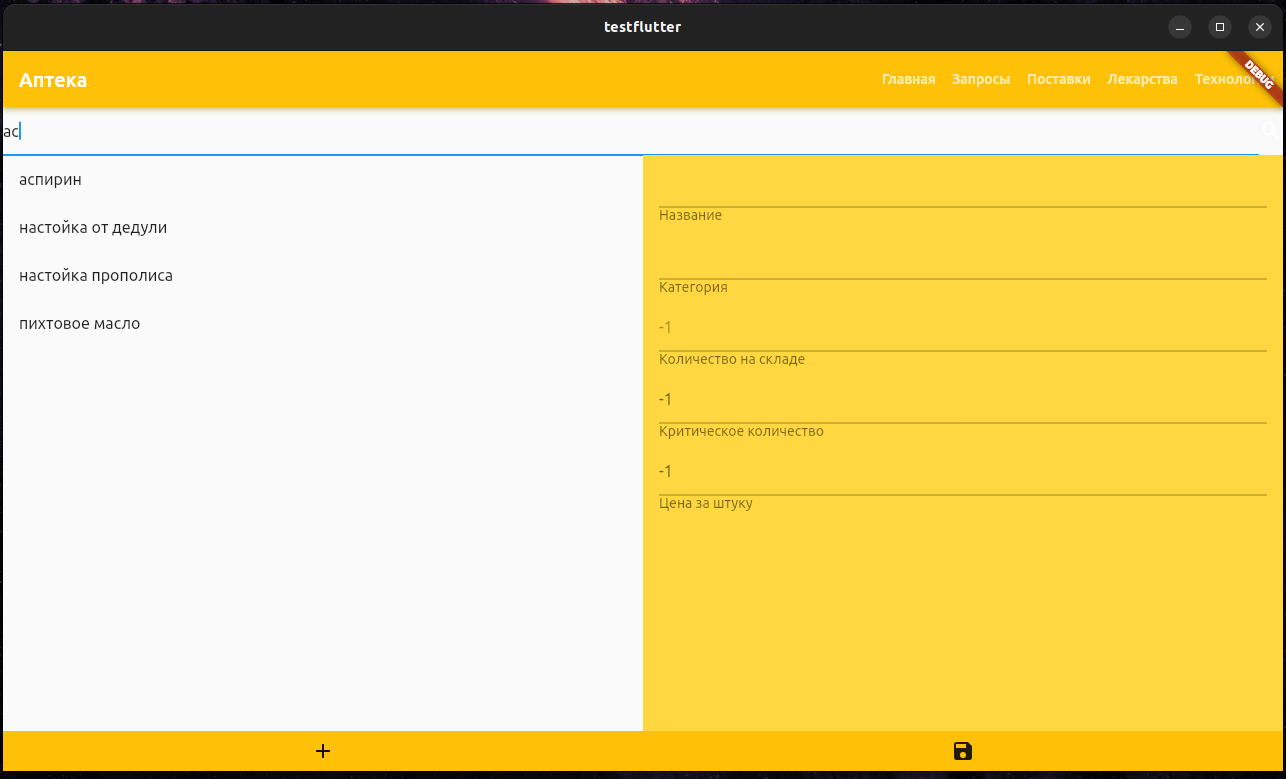
На вкладке «запросы» можно выполнить заготовленные запросы в БД:



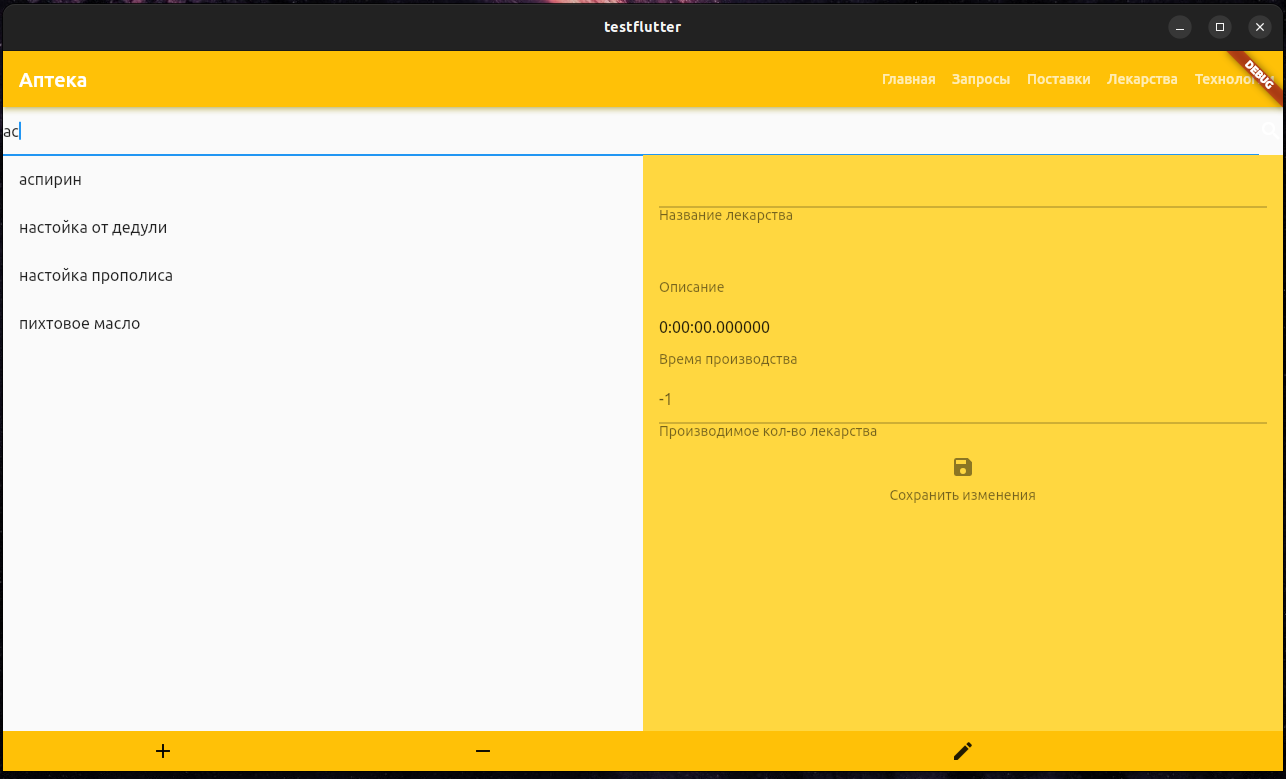
Раздел с поставками, где можно создать заказ на поставку, а также зарегистрировать поставку для конкретного заказа:



Раздел с лекарствами, где можно добавить, удалить и изменить лекарства:



Раздел с технологиями, где можно добавить, удалить, изменить технологию, в том числе необходимые компоненты:



# Выводы

В результате работы было написано клиент-серверное приложение для аптеки, где в качестве сервера была выбрана БД Postgres, а клиент написан для платформ Mac/Linux/Windows на языке Dart с использованием Flutter. В ходе работы были получены навыки работы с БД Postgres, SQL, PLPGSQL, Dart, Flutter.

В первом этапе была продумана и создана архитектура для БД, которая бы удовлетворяла условиям задания, оставалась гибкой для возможных изменений и с защитой от неправильных данных. Использован язык SQL.

Во втором этапе были написаны SQL запросы для извлечения данных в удобном формате.

В третьем этапе были написаны триггеры, функции и процедуры, которые бы предоставляли удобный интерфейс взаимодействия с БД и также поддерживали целостность данных.

В четвертном этапе был написан GUI для удобного взаимодействия между БД и человеком, не имеющего знаний внутреннего устройства.