

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ  
Московский авиационный институт  
(национальный исследовательский университет)

Институт № 8  
Компьютерные науки и прикладная математика  
Кафедра 806 «Компьютерная математика»

КУРСОВАЯ РАБОТА  
по дисциплине «Базы данных»  
на тему: Проектирование базы данных  
«сервис по ремонту техники»

**Выполнил:** студент группы М8О-311Б-20

---

Иваненков Лев Михайлович

---

(Фамилия, имя, отчество)

---

(подпись)

---

**Принял:** доцент кафедры 806

---

Чумакова Екатерина Витальевна

---

(Фамилия, имя, отчество)

---

(подпись)

---

Оценка:

Дата:

Москва, 2022

# Содержание

<b>ВВЕДЕНИЕ</b>	<b>3</b>
<b>1 Анализ предметной области и постановка задачи</b>	<b>3</b>
1.1 Описание предметной области и назначение системы . . . . .	3
1.2 Цели разработки и способы достижения . . . . .	3
1.3 Ожидаемые результаты . . . . .	4
1.4 Информационный образ системы . . . . .	4
1.5 Функциональный образ системы . . . . .	5
1.6 Требования и ограничения на использование системы . . . . .	5
<b>2 Концептуальное и логическое проектирование базы данных</b>	<b>6</b>
2.1 Выделение информационных объектов и определение атрибутов объектов	6
2.2 Разработка логической структуры базы данных . . . . .	9
2.3 Построение диаграммы сущность-связь . . . . .	10
2.3.1 Раскрытие связей многие-ко-многим . . . . .	10
2.3.2 Поддержка ограничений целостности, списки доменов атрибутов	11
2.3.3 Нормализация объектной модели . . . . .	12
2.4 Выбор используемой СУБД . . . . .	12
<b>3 Физическое проектирование базы данных</b>	<b>13</b>
3.1 Генерация физической модели БД . . . . .	13
3.2 Генерация физической схемы БД . . . . .	13
3.3 Описание представлений, процедур, функций и триггеров к БД . . . . .	14
3.4 Права доступа к данным . . . . .	21
<b>4 Разработка прикладной программы</b>	<b>22</b>
4.1 Интерфейс программы . . . . .	22
4.2 Возможности программы . . . . .	23
<b>ЗАКЛЮЧЕНИЕ</b>	<b>25</b>
<b>СПИСОК ЛИТЕРАТУРЫ</b>	<b>26</b>
<b>ПРИЛОЖЕНИЕ</b>	<b>27</b>

# ВВЕДЕНИЕ

Современное общество всё чаще называют «информационным». Это связано с тем, что сегодня в любой среде человеческой деятельности одной из главных задач является организация хранения и обработки большого количества информации. В этом существенную помощь могут оказать компьютерные системы обработки данных. Основная цель подобных систем – повышение эффективности работы отдельной фирмы, предприятия или организации.

Многие существующие экономические, информационно-справочные, банковские программные комплексы реализованы с использованием инструментальных средств систем управления базами данных. Системы управления базами данных предназначены для автоматизации процедур создания, хранения, извлечения, обработки и анализа электронных данных.

## 1 Анализ предметной области и постановка задачи

### 1.1 Описание предметной области и назначение системы

Разрабатываемая подсистема носит название “Орбита-сервис” и предназначена для:

1. автоматизации обработки поступающих заказов в сервис по ремонту техники
2. обеспечения возможности оперативного поиска заказов по заданным параметрам
3. обеспечения возможности оперативного получения информации по

### 1.2 Цели разработки и способы достижения

Цели разработки:

1. обеспечить минимальные затраты времени для получения информации;
2. упростить поиск нужной информации о заказах;
3. обеспечить оперативность изменения информации.

Указанные цели разработки могут быть достигнуты следующими способами:

1. проанализировать документы, которые имеют отношение к решаемой задаче для выявления значимых объектов (сущностей) и атрибутов сущностей – данных, которые должны храниться в БД;
2. описать модель данных в удобном для разработчиков способе (в виде ER–диаграммы);
3. представить информацию в связанных таблицах;
4. разработать запросы и процедуры для автоматического учета заказов.

### 1.3 Ожидаемые результаты

В результате применения разрабатываемой системы ожидается:

1. упрощение работы менеджера по работе с клиентами;
2. формирование нового заказа не более чем за 2 минуты, вместо 5 минут при рукописном составлении заказов;
3. выбор наиболее оптимальной информации для конкретного пользователя;
4. увеличение количества обслуженных клиентов.

### 1.4 Информационный образ системы

Для создания ER-модели необходимо выделить сущности предметной области:

1. **client** Информация о клиенте
2. **component** Информация о деталях для ремонта
3. **issue** Информация о поломке техники
4. **master** Информация о мастере
5. **orders** Информация о заказе
6. **rent** Информация о аренде сервиса

- 7. **service** Информация о сервисе
- 8. **technic** Информация о технике

## 1.5 Функциональный образ системы

Система предназначена для решения следующих задач:

- 1. Выводить главный список заказов с основными данными заказа;
- 2. Выводить список заказов с разными состояниями ремонта;
- 3. Добавлять новые заказы;
- 4. Изменять/удалять заказы.
- 5. Считать зарплату мастеров за определенный промежуток времени.
- 6. Считать прибыль/расходы за определенный промежуток времени.

## 1.6 Требования и ограничения на использование системы

Пользователи системы:

- 1. Администратор — имеет полный доступ к системе, в том числе может считать зарплату мастеров, считать прибыль/расходы.
- 2. Менеджер по работе с клиентами — может добавлять, удалять, изменять заказы.
- 3. Мастер — может изменять заказы.

## 2 Концептуальное и логическое проектирование базы данных

### 2.1 Выделение информационных объектов и определение атрибутов объектов

Для каждой сущности определим первичный ключ (служащий для идентификации записей) отношения и перечень атрибутов, имеющих значение в предметной области задачи. Для каждого отношения указаны атрибуты с их внутренним названием, типом и длиной:

#### **client**

Поле	Тип данных	Назначение	Примечание
idclient	integer	Идентификатор клиента	первичный ключ
fio_client	character varying	ФИО клиента	обязательное поле
tel_num	character varying	Номер телефона	12312
dop_tel_num	character varying	Дополнительный номер телефона	12312

#### **component**

Поле	Тип данных	Назначение	Примечание
idcomponent	integer	Идентификатор детали	первичный ключ
description_component	character varying	Описание детали	обязательное поле
address	character varying	Адресс для покупки детали	

## issue

Поле	Тип данных	Назначение	Примечание
idissue	integer	Идентификатор поломки	первичный ключ
description_issue	character varying	Описание поломки	обязательное поле

## master

Поле	Тип данных	Назначение	Примечание
idmaster	integer	Идентификатор мастера	первичный ключ
fio_master	character varying	ФИО мастера	обязательное поле

## orders

Поле	Тип данных	Назначение	Примечание
idorders	integer	Идентификатор заказа	первичный ключ
date_get	date	Дата получения заказа	обязательное поле
condition_order	character varying	Состояние заказа	
total_price	integer	Цена заказа	
clientid	integer	Идентификатор клиента	обязательное поле, внешний ключ
technicid	integer	Идентификатор техники	обязательное поле, внешний ключ
comment_orders	character varying	Комментарий к заказу	

## rent

Поле	Тип данных	Назначение	Примечание
idrent	integer	Идентификатор аренды	первичный ключ
rent_premises	integer	Стоимость аренды	обязательное поле
utilites	integer	Коммунальные услуги	
advertisement	integer	Реклама	

## service

Поле	Тип данных	Назначение	Примечание
idservice	integer	Идентификатор сервиса	первичный ключ
address	character varying	Адрес сервиса	обязательное поле
rentid	integer	Идентификатор аренды	обязательное поле, внешний ключ

## technic

Поле	Тип данных	Назначение	Примечание
idtechnic	integer	Идентификатор техники	первичный ключ
type_technic	character varying	Тип техники	обязательное поле
model	character varying	Модель	обязательное поле
brand	character varying	Бренд техники	
imei	character varying	IMEI	
comment_technic	character varying	Комментарий к технике	



## 2.2 Разработка логической структуры базы данных

Для построения ER-модели на основе выделенных сущностей, необходимо определить типы связей между ними.

Связь типа один-ко-многим — межтабличное отношение, при котором любая запись в первой таблице может быть связана несколькими записями во второй, но в то же время любая запись второй таблицы связана только с одной записью в первой.

Связь типа многие-ко-многим — это связь, при которой множественным записям из одной таблицы могут соответствовать множественные записи из другой.

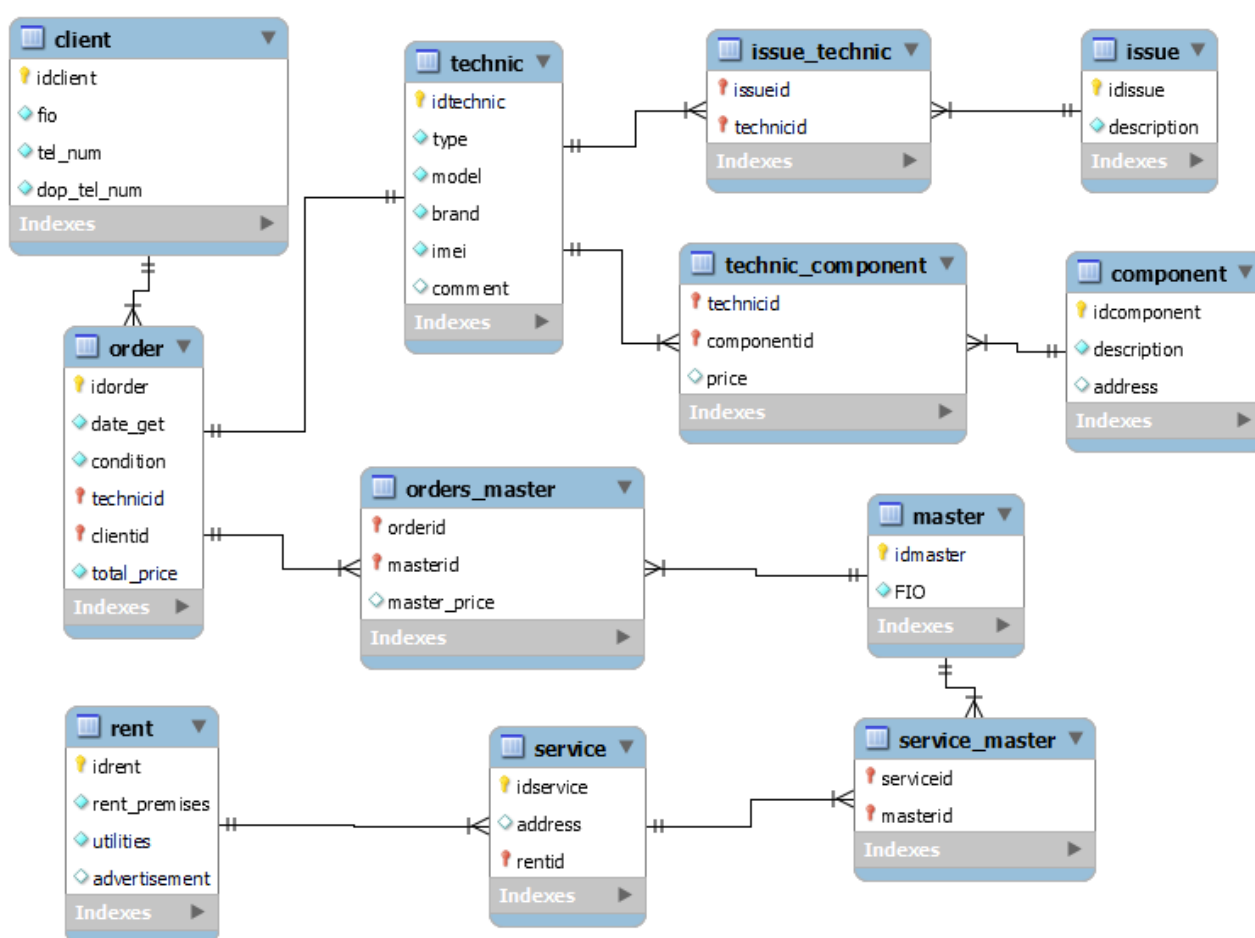


Рис. 1: Физическая модель базы данных

В данной базе данных используется четыре связи многие-ко-многим, между: service и master, technic и component, technic и issue, master и orders.

## 2.3 Построение диаграммы сущность-связь

### 2.3.1 Раскрытие связей многие-ко-многим

В полученной ER-модели имеется четыре связи многие-ко-многим, которые не поддерживаются ни одной из реляционных СУБД. Подобные связи преобразуются (раскрываются) через промежуточную таблицу. Таким образом, в модель будут добавлены две сущности, каждая из которых является подчинённой по отношению к таблицам, объединённым связью многие-ко-многим (т.е. связывается с этими таблицами связью многие-к-одному). При этом каждое из отношений будет иметь составной первичный ключ из двух внешних ключей главных таблиц.

Для связи `technic` и `component` в виде таблицы:

#### **`technic_component`**

Поле	Тип данных	Назначение	Примечание
<code>technic_idtechnic</code>	<code>integer</code>	Идентификатор техники	составной первичный ключ
<code>component_idcomponent</code>	<code>integer</code>	Идентификатор детали	составной первичный ключ
<code>component_price</code>	<code>integer</code>	цена компонента	

Отношение, описывающие технику и детали для нее.

Для связи `technic` и `issue` в виде таблицы:

#### **`technic_issue`**

Поле	Тип данных	Назначение	Примечание
<code>technic_idtechnic</code>	<code>integer</code>	Идентификатор техники	составной первичный ключ
<code>issue_idissue</code>	<code>integer</code>	Идентификатор поломки	составной первичный ключ

Отношение, описывающие технику и ее поломки.

Для связи master и orders в виде таблицы:

#### **master\_orders**

Поле	Тип данных	Назначение	Примечание
master_idmaster	integer	Идентификатор мастера	составной первичный ключ
orders_idorders	integer	Идентификатор заказа	составной первичный ключ
master_price	integer	цена работы мастера	

Отношение, описывающие мастера и его заказы.

Для связи master и service в виде таблицы:

#### **master\_service**

Поле	Тип данных	Назначение	Примечание
master_idmaster	integer	Идентификатор мастера	составной первичный ключ
service_idservice	integer	Идентификатор сервиса	составной первичный ключ

Отношение, описывающие сервис и его мастеров.

### **2.3.2 Поддержка ограничений целостности, списки доменов атрибутов**

Для сохранения целостности данных определим следующие ограничения целостности:

1. обязательные данные;
2. ограничения для доменов атрибутов;
3. целостность сущностей;
4. ссылочная целостность.

При определении атрибутов сущностей были указаны как обязательные, так и необязательные для заполнения данные, т.е. эти атрибуты могут иметь пустые (NULL) и непустые (NOT NULL) значения соответственно.

Первичный ключ любой сущности не может содержать пустого значения, для этого в свойствах атрибута в sql-скрипте ([приложение а](#)) зададим обязательность (NOT NULL). Внешний ключ связывает каждую строку дочернего отношения с той строкой родительского отношения, которая содержит это же значение соответствующего первичного ключа. Понятие ссылочной целостности означает, что если внешний ключ содержит некоторое значение, то оно обязательно должно присутствовать в первичном ключе одной из строк родительского отношения. Для обеспечения целостности данных в свойствах связей установим значение cascade на изменения delete для каскадного обновления связанных полей.

### 2.3.3 Нормализация объектной модели

Хранение информации в реляционной БД, предполагает такую структуру, в которой данные хранятся в оптимальном объеме. Для этого предусмотрен механизм нормализации. Проверим на соответствие как минимум 3-й нормальной форме полученных отношений диаграммы. Все таблицы в данной системе обладают следующими свойствами:

1. Каждый элемент таблицы представляет 1 элемент данных (атомарность).
2. Все столбцы таблиц однородные.
3. Отсутствуют повторяющиеся атрибуты (соответствие 1НФ).
4. Отсутствуют атрибуты, зависящие только от части уникального идентификатора (2НФ).
5. Отсутствуют атрибуты, зависящие от атрибутов, не входящих в уникальный идентификатор (3НФ), т.е. все отношения находится в 3-й нормальной форме, т.к. находятся во 2НФ и каждый не ключевой атрибут нетранзитивно зависит от первичного ключа.

## 2.4 Выбор используемой СУБД

Для выполнения курсового проекта используется СУБД PostgreSQL. PostgreSQL — это реляционная СУБД с открытым кодом и свободным распространением. База данных Postgre базируется на стандартном языке запросов SQL, а точнее,

на его расширении — процедурном языке PL/pgSQL. Также для взаимодействия с базой данных написана программа на C++ и QT.

## 3 Физическое проектирование базы данных

### 3.1 Генерация физической модели БД

В результате проведенной разработки получили ER-диаграмму физической модели БД, которая приведена на рисунке 1.

Для создания базы данных использовался sql-скрипт ([приложение а](#)), написанный на plpgsql ().

### 3.2 Генерация физической схемы БД

Результатом выполнения sql-скрипта ([приложение а](#)) является физическая база данных, схема данных которой представлена на рисунке 2.

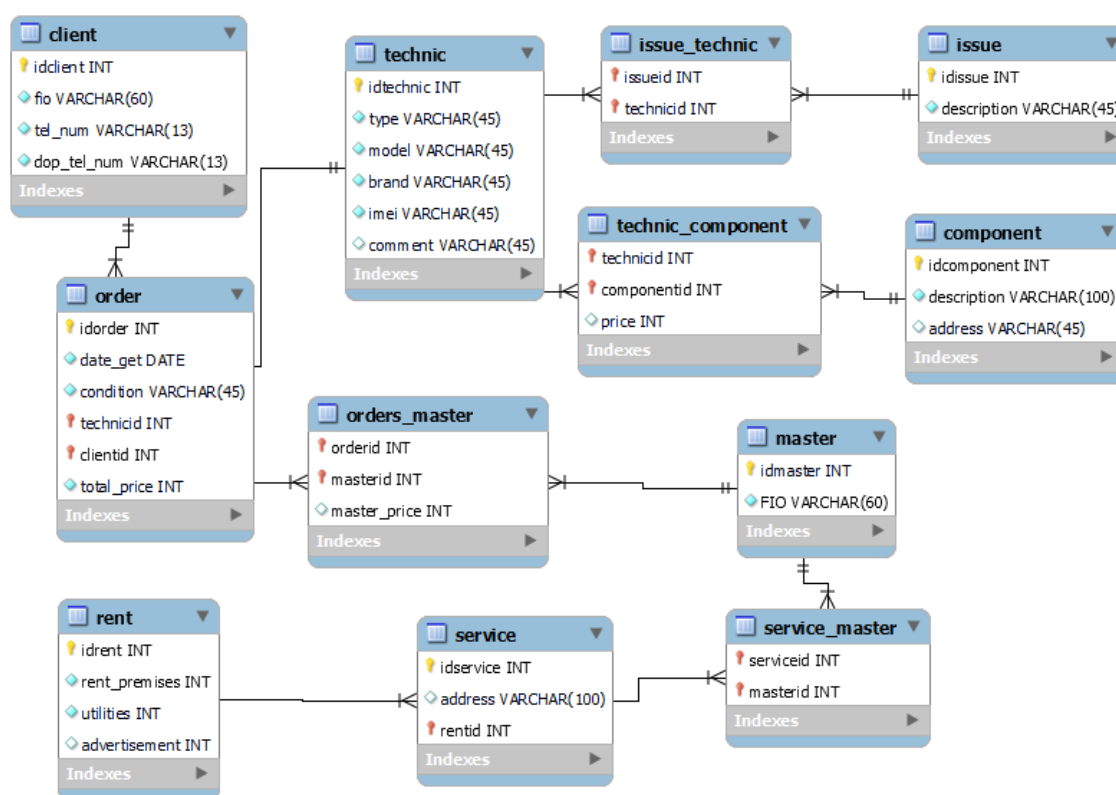


Рис. 2: Физическая схема базы данных

### 3.3 Описание представлений, процедур, функций и триггеров к БД

#### 1. Function: make\_order\_client(text, text, text)

```
1 CREATE OR REPLACE FUNCTION make_order_client(  
2   IN in_fio text,  
3   IN in_tel text,  
4   IN in_dop_tel text DEFAULT '0'  
5 )  
6 RETURNS INTEGER AS $$  
7 BEGIN  
8     IF EXISTS (select * from client where fio_client = in_fio and tel_num = in_tel) THEN  
9         RETURN (select idclient from client where fio_client = in_fio and tel_num = in_tel);  
10    ELSE  
11        INSERT INTO client(fio_client, tel_num, dop_tel_num)  
12            VALUES (in_fio, in_tel, in_dop_tel);  
13        RETURN (select idclient from client where fio_client = in_fio and tel_num = in_tel);  
14    END IF;  
15 END;  
16 $$  
17 LANGUAGE 'plpgsql';
```

Функция принимает на вход данные клиента, если клиент с такими параметрами существует то возвращает его идентификатор, иначе создает нового пользователя и возвращает его идентификатор.

#### 2. Function: make\_order\_techinc(text, text, text)

```
1 CREATE OR REPLACE FUNCTION make_order_techinc(  
2   IN in_type text,  
3   IN in_model text,  
4   IN in_brand text,  
5   IN in_imei text DEFAULT ''  
6 )  
7 RETURNS INTEGER AS $$  
8 BEGIN  
9     INSERT INTO technic(type_techinc, model, brand, imei)  
10        VALUES(in_type, in_model, in_brand, in_imei);  
11    RETURN (select idtechnic from technic where type_techinc = in_type and model = in_model and  
12        ⇐ imei = in_imei and brand = in_brand);  
13 END;  
14 $$  
15 LANGUAGE 'plpgsql';
```

Функция принимает на вход данные техники, создает новую запись.

### 3. Function: make\_order\_issue(text, integer)

```
1 CREATE OR REPLACE FUNCTION make_order_issue(  
2   IN in_description_issue text,  
3   IN in_technicid integer  
4 )  
5 RETURNS void AS $$  
6 BEGIN  
7     IF EXISTS (select * from issue where description_issue = in_description_issue) THEN  
8         INSERT INTO technic_issue(technic_idtechnic, issue_idissue)  
9             VALUES(in_technicid, (select idissue from issue where description_issue =  
↪ in_description_issue));  
10    ELSE  
11        INSERT INTO issue(description_issue)  
12            VALUES (in_description_issue);  
13        INSERT INTO technic_issue(technic_idtechnic, issue_idissue)  
14            VALUES(in_technicid, (select idissue from issue where description_issue =  
↪ in_description_issue));  
15    END IF;  
16 END;  
17 $$  
18 LANGUAGE 'plpgsql';
```

Функция принимает на вход данные поломки, если поломка с такими параметрами существует, то создает запись в таблице соединяющей технику и поломку, иначе создает новую запись в таблице с поломками и создает запись в таблице соединяющей технику и поломку.

### 4. Function: make\_order\_component(text, integer, integer)

```
1 CREATE OR REPLACE FUNCTION make_order_component(  
2   IN in_description_component text,  
3   IN in_technicid integer,  
4   IN in_price_component integer DEFAULT 0  
5 )  
6 RETURNS void AS $$  
7 BEGIN  
8     IF EXISTS (select * from component where description_component = in_description_component)  
↪ THEN  
9         INSERT INTO technic_component(price_component, technic_idtechnic, component_idcomponent)  
10            VALUES(in_price_component, in_technicid, (select idcomponent from component where  
↪ description_component = in_description_component));  
11    ELSE  
12        INSERT INTO component(description_component)
```

```

13      VALUES (in_description_component);
14      INSERT INTO technic_component(price_component, technic_idtechnic, component_idcomponent)
15      VALUES(in_price_component, in_technicid, (select idcomponent from component where
↪ description_component = in_description_component));
16      END IF;
17  END;
18  $$
19  LANGUAGE 'plpgsql';

```

Функция принимает на вход данные компонента, если компонент с такими параметрами существует, то создает запись в таблице соединяющей технику и компоненты, иначе создает новую запись в таблице с компонентами и создает запись в таблице соединяющей технику и компоненты.

#### 5. Function: make\_order\_orders(integer, integer, text, integer, integer, date)

```

1  CREATE OR REPLACE FUNCTION make_order_orders(
2  IN in_technicid integer,
3  IN in_clientid integer,
4  IN in_condition text,
5  IN in_masterid integer,
6  IN in_master_price integer,
7  IN in_total_price integer DEFAULT 0,
8  IN in_date_get date DEFAULT CURRENT_DATE
9  )
10 RETURNS INTEGER AS $$
11 BEGIN
12     INSERT INTO orders(date_get, condition_order, clientid, technicid, total_price)
13     VALUES(in_date_get, in_condition, in_clientid, in_technicid, in_total_price);
14     INSERT INTO master_orders(master_idmaster, orders_idorders, master_price)
15     VALUES(in_masterid, (select idorders from orders where clientid = in_clientid and
↪ technicid = in_technicid), in_master_price);
16
17     RETURN (select idorders from orders where clientid = in_clientid and technicid =
↪ in_technicid);
18 END;
19 $$
20 LANGUAGE 'plpgsql';

```

Функция принимает на вход данные заказа, создает запись в таблице заказа, создает запись в таблице, соединяющую заказы и мастеров, и возвращает его идентификатор.

#### 6. Function: edit\_order\_issue(text, integer)



```

1 CREATE OR REPLACE FUNCTION edit_order_issue(
2   IN in_description_issue text,
3   IN in_idorder integer
4 )
5 RETURNS void AS $$
6 BEGIN
7   IF EXISTS (select * from issue where description_issue = in_description_issue) THEN
8     UPDATE technic_issue
9       SET issue_idissue = (select idissue from issue where description_issue =
↵ in_description_issue)
10      WHERE technic_idtechnic in (select technicid from orders where idorders =
↵ in_idorder);
11  ELSE
12    INSERT INTO issue(description_issue)
13      VALUES (in_description_issue);
14    UPDATE technic_issue
15      SET issue_idissue = (select idissue from issue where description_issue =
↵ in_description_issue)
16      WHERE technic_idtechnic in (select technicid from orders where idorders =
↵ in_idorder);
17  END IF;
18 END;
19 $$
20 LANGUAGE 'plpgsql';

```

Функция принимает на вход название поломки и идентификатор заказа, если такая поломка уже существует то обновляет поломку техники из заказа с идентификатором принятым функцией, иначе создает новую запись в таблице с поломками и присваивает технике поломку с этим названием.

## 7. Function: edit\_order\_component(text, integer, integer)

```

1 CREATE OR REPLACE FUNCTION edit_order_component(
2   IN in_description_component text,
3   IN in_idorder integer,
4   IN in_price_component integer DEFAULT 0
5 )
6 RETURNS void AS $$
7 BEGIN
8   IF EXISTS (select * from component where description_component = in_description_component)
↵ THEN
9     UPDATE technic_component
10      SET component_idcomponent = (select idcomponent from component where
↵ description_component = in_description_component)
11      WHERE technic_idtechnic in (select technicid from orders where idorders =
↵ in_idorder);

```

```

12      UPDATE technic_component
13      SET price_component = in_price_component
14      WHERE technic_idtechnic in (select technicid from orders where idorders =
↵ in_idorder);
15  ELSE
16      INSERT INTO component(description_component)
17      VALUES (in_description_component);
18      UPDATE technic_component
19      SET component_idcomponent = (select idcomponent from component where
↵ description_component = in_description_component)
20      WHERE technic_idtechnic in (select technicid from orders where idorders =
↵ in_idorder);
21      UPDATE technic_component
22      SET price_component = in_price_component
23      WHERE technic_idtechnic in (select technicid from orders where idorders =
↵ in_idorder);
24  END IF;
25  END;
26  $$
27  LANGUAGE 'plpgsql';

```

Функция принимает на вход название компонента, цену компонента и идентификатор заказа, если такой компонент уже существует то обновляет компонент техники из заказа с идентификатором принятым функцией, иначе создает новую запись в таблице с компонентами и присваивает технике компонент с этим названием и указывает цену компонента.

## 8. Function: delete\_order(integer)

```

1  CREATE OR REPLACE FUNCTION delete_order(
2  IN in_idorders integer
3  )
4  RETURNS void AS $$
5  BEGIN
6      DELETE from master_orders
7      where orders_idorders = in_idorders;
8
9      DELETE from orders
10     where idorders = in_idorders;
11  END;
12  $$
13  LANGUAGE 'plpgsql';

```

Функция принимает на вход идентификатор заказа и удаляет связанные с заказом данные.

## 9. Function: cash\_out\_master(integer, date, date)

```
1 CREATE OR REPLACE FUNCTION cash_out_master(  
2   IN in_idmaster int,  
3   IN in_date_1 date,  
4   IN in_date_2 date  
5 )  
6 RETURNS INTEGER AS $$  
7 #print_strict_params on  
8 DECLARE  
9   sum_master int;  
10 BEGIN  
11   select sum(master_price) into strict sum_master from master_orders, orders  
12     where master_orders.orders_idorders = orders.idorders and  
13       orders.date_get between in_date_1 and in_date_2 and  
14       master_orders.master_idmaster = in_idmaster;  
15   RETURN sum_master;  
16 END;  
17 $$  
18 language 'plpgsql';
```

Функция принимает на вход идентификатор мастера и две даты и возвращает сумму денег заработанную мастером за это время.

## 10. View: main\_view()

```
1 CREATE OR REPLACE VIEW main_view AS  
2   select orders.idorders, orders.date_get, client.fio_client, technic.type_technic,  
3     technic.brand, technic.model, technic.imei, issue.description_issue,  
4     orders.condition_order, orders.total_price, master_orders.master_price,  
5     technic_component.price_component, master.fio_master  
6   from component, master, issue, technic_issue, technic_component,  
7     orders join client on orders.clientid = client.idclient  
8       join technic on orders.technicid = technic.idtechnic  
9       join master_orders on orders.idorders = master_orders.orders_idorders  
10  where master.idmaster = master_orders.master_idmaster and  
11    technic.idtechnic = technic_issue.technic_idtechnic and  
12    issue.idissue = technic_issue.issue_idissue and  
13    technic.idtechnic = technic_component.technic_idtechnic and  
14    component.idcomponent = technic_component.component_idcomponent  
15  order by orders.idorders;
```

Представление хранит в себе основую таблицу с данными заказа, его клиента, техники, поломки, детали и мастера.

Аналогичным образом созданы три представления хранящие заказы с разным его состояниями(condition\_order): ready\_view, diagnostic\_view, component\_view.

#### 11. Trigger: technic\_component\_before\_insert

```
1 CREATE OR REPLACE TRIGGER technic_component_before_insert BEFORE INSERT ON
  ↳ public.technic_component FOR EACH ROW
2 EXECUTE FUNCTION change_total_price_tc();
3
4 CREATE OR REPLACE FUNCTION change_total_price_tc()
5 RETURNS trigger AS $$
6 BEGIN
7     UPDATE public.orders
8         SET total_price = (total_price + NEW.price_component)
9         WHERE idorders IN (select idorders from orders
10                             where orders.technicid = NEW.technic_idtechnic);
11 RETURN NEW;
12 END;
13 $$
14 LANGUAGE 'plpgsql';
```

Триггер вызывается до вставки в таблицу связывающую технику и компонент и обновляет итоговую сумму заказа(total\_price) в таблице заказов.

#### 12. Trigger: master\_orders\_before\_insert

```
1 CREATE OR REPLACE TRIGGER technic_component_before_updact BEFORE UPDATE ON
  ↳ public.technic_component FOR EACH ROW
2 EXECUTE FUNCTION change_total_price_tc();
3
4 CREATE OR REPLACE FUNCTION change_total_price_tc_upd()
5 RETURNS trigger AS $$
6 BEGIN
7     UPDATE public.orders
8         SET total_price = ((select master_price from master_orders
9                             where orders_idorders = NEW.orders_idorders)
10                            + NEW.price_component)
11     WHERE idorders IN (select idorders from orders
12                         where orders.technicid = NEW.technic_idtechnic);
13 RETURN NEW;
14 END;
15 $$
16 LANGUAGE 'plpgsql';;
```

Триггер вызывается до обновления таблицы связывающую технику и компонент и обновляет итоговую сумму заказа(`total_price`) в таблице заказов.

Аналогично созданы триггеры на добавление и обновление в таблице связывающую мастеров и заказы.

### 3.4 Права доступа к данным

Опишем для каждой группы пользователей права доступа к каждой таблице. Права доступа должны быть распределены так, чтобы для каждого объекта БД был хотя бы один пользователь, который имеет право добавлять и удалять данные из объекта.

Используются следующие сокращения:

1. s – чтение данных (select);
2. i – добавление данных (insert);
3. u – модификация данных (update);
4. d – удаление данных(delete).

Таблица 1. Права доступа к таблицам для групп пользователей

Таблицы	Администратор	Директор	Менеджер по работе с клиентами	Мастер
client	SIUD	SIUD	SIU	S
component	SIUD	SIU	SIU	SIU
issue	SIUD	SIU	SIU	SIU
master	SIUD	SIUD	S	S
orders	SIUD	SIUD	SIUD	S
rent	SIUD	SIUD		
service	SIUD	SIUD		
technic	SIUD	SIUD	SIUD	S

technic_issue	SIUD	SIUD	SIUD	SIU
technic_- component	SIUD	SIUD	SIUD	SIU
master_orders	SIUD	SIUD	SIUD	S
master_service	SIUD	SIUD		S

Права назначает администратор БД (или администратор безопасности, если система сложная администраторов несколько). Права доступа пользователей предоставляются с помощью команды GRANT.

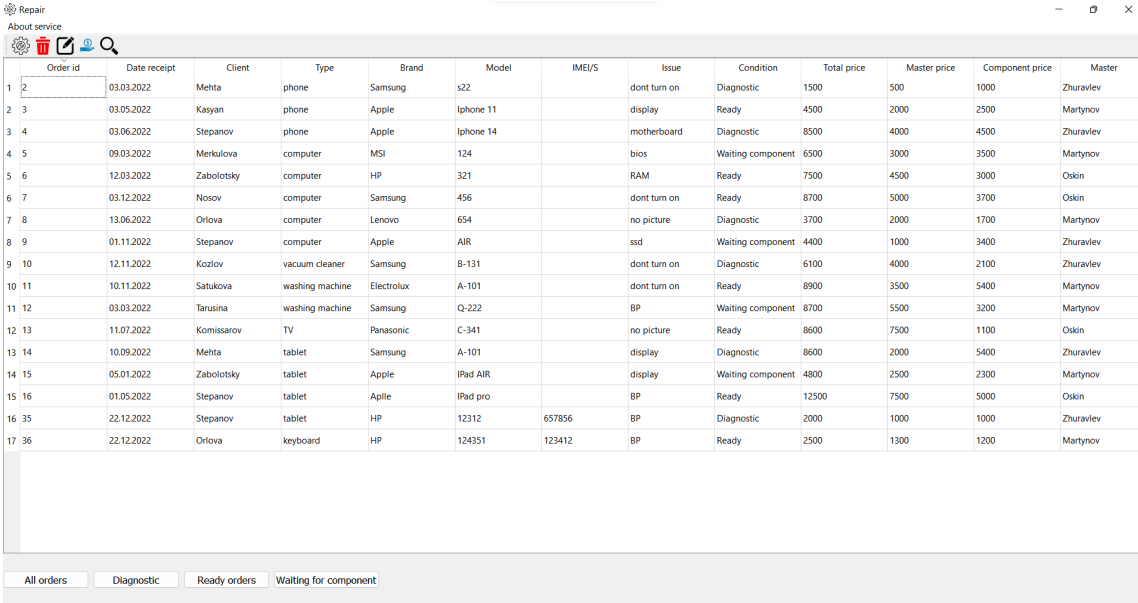
Права доступа пользователей предоставляются с помощью команды GRANT. Рассмотрим для примера права мастера (master\_user). Права доступа к отношениям component и orders могут быть описаны следующим образом:

grant select, insert, update on component to master\_user;

grant select on orders to master\_user;

## 4 Разработка прикладной программы

### 4.1 Интерфейс программы



	Order id	Date receipt	Client	Type	Brand	Model	IMEI/S	Issue	Condition	Total price	Master price	Component price	Master
1	2	03.03.2022	Mehta	phone	Samsung	s22		dont turn on	Diagnostic	1500	500	1000	Zhuravlev
2	3	03.05.2022	Kasyan	phone	Apple	Iphone 11		display	Ready	4500	2000	2500	Martynov
3	4	03.06.2022	Stepanov	phone	Apple	Iphone 14		motherboard	Diagnostic	8500	4000	4500	Zhuravlev
4	5	09.03.2022	Merkulova	computer	MSI	124		bios	Waiting component	6500	3000	3500	Martynov
5	6	12.03.2022	Zabolotsky	computer	HP	321		RAM	Ready	7500	4500	3000	Oskin
6	7	03.12.2022	Nosov	computer	Samsung	456		dont turn on	Ready	8700	5000	3700	Oskin
7	8	13.06.2022	Orlova	computer	Lenovo	654		no picture	Diagnostic	3700	2000	1700	Martynov
8	9	01.11.2022	Stepanov	computer	Apple	AIR		ssd	Waiting component	4400	1000	3400	Zhuravlev
9	10	12.11.2022	Kozlov	vacuum cleaner	Samsung	B-131		dont turn on	Diagnostic	6100	4000	2100	Zhuravlev
10	11	10.11.2022	Satukova	washing machine	Electrolux	A-101		dont turn on	Ready	8900	3500	5400	Martynov
11	12	03.03.2022	Tarusina	washing machine	Samsung	Q-222		BP	Waiting component	8700	5500	3200	Martynov
12	13	11.07.2022	Komissarov	TV	Panasonic	C-341		no picture	Ready	8600	7500	1100	Oskin
13	14	10.09.2022	Mehta	tablet	Samsung	A-101		display	Diagnostic	8600	2000	5400	Zhuravlev
14	15	05.01.2022	Zabolotsky	tablet	Apple	IPad AIR		display	Waiting component	4800	2500	2300	Martynov
15	16	01.05.2022	Stepanov	tablet	Apfle	IPad pro		BP	Ready	12500	7500	5000	Oskin
16	35	22.12.2022	Stepanov	tablet	HP		12312	BP	Diagnostic	2000	1000	1000	Zhuravlev
17	36	22.12.2022	Orlova	keyboard	HP	124351	123412	BP	Ready	2500	1300	1200	Martynov

Рис. 3: Общий вид программы

## 4.2 Возможности программы

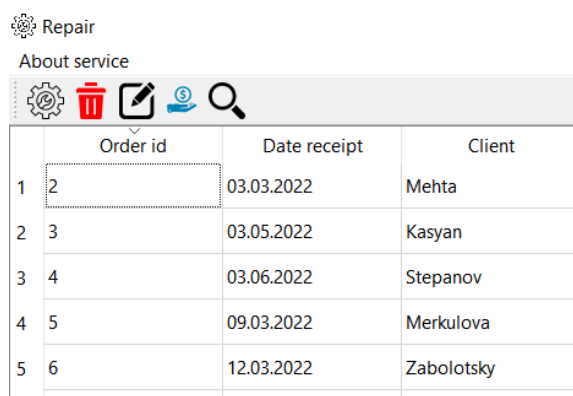


Рис. 4: Tool bar

На рисунке 4 на tool bar-е расположены иконки для создание, удаление, изменение заказа и поиска по таблице.

14	15	05.01.2022	Zabolotsky	tablet	Apple
15	16	01.05.2022	Stepanov	tablet	Aplle
16	35	22.12.2022	Stepanov	tablet	HP
17	36	22.12.2022	Orlova	keyboard	HP

All orders

Diagnostic

Ready orders

Waiting for component

Рис. 5: Состояние заказа

На рисунке 5 в левом нижнем углу расположены кнопки для сортировки по состоянию заказа.

Рис. 6: Создание заказа

На рисунке 6 окно для создание заказа.

Рис. 7: Состояние заказа

На рисунке 7 окно для поиска данных по параметрам.



# ЗАКЛЮЧЕНИЕ

Таким образом, в работе было показано, как создать базу данных сервиса по ремонту техники «Орбита-сервис» для повышения эффективности работы.

Были реализованы возможности:

1. Добавление, изменение, удаление заказов.
2. Подсчет заработной платы мастеров.
3. Сортировка заказов по разным параметрам.
4. Подсчет трат на содержание помещения мастерской.

# СПИСОК ЛИТЕРАТУРЫ

1. Баженова И.Ю. Основы проектирования приложений баз данных (2-е изд.) - М.: НОУ "Интуит 2016.- 237с.
2. Карпова И.П. Базы данных. Учебное пособие. – Московский государственный институт электроники и математики (Технический университет). – М., 2009.
3. Коннолли Т., Бегг К. Базы данных: проектирование, реализация, сопровождение. Теория и практика. – 3-е изд.: Пер. с англ.: Уч. пос. – М.: Изд. дом "Вильямс 2003. – 1440 с.

# ПРИЛОЖЕНИЕ

## ПРИЛОЖЕНИЕ А

---

```
1
2 BEGIN;
3
4
5 CREATE TABLE IF NOT EXISTS public.client
6 (
7     idclient int GENERATED ALWAYS AS IDENTITY NOT NULL,
8     fio character varying(60) NOT NULL,
9     tel_num character varying(13) NOT NULL,
10    dop_tel_num character varying(13),
11    CONSTRAINT PK_client_idclient PRIMARY KEY (idclient)
12 );
13
14 CREATE TABLE IF NOT EXISTS public.technic
15 (
16     idtechnic int GENERATED ALWAYS AS IDENTITY NOT NULL,
17     type character varying(45) NOT NULL,
18     model character varying(45) NOT NULL,
19     brand character varying(45) NOT NULL,
20     imei character varying(45) NOT NULL,
21     comment character varying(45),
22     CONSTRAINT PK_technic_idtechnic PRIMARY KEY (idtechnic)
23 );
24
25 CREATE TABLE IF NOT EXISTS public.orders
26 (
27     idorders int GENERATED ALWAYS AS IDENTITY NOT NULL,
28     date_get date NOT NULL,
29     condition character varying(45),
30     total_price int NOT NULL,
31     clientid int ,
32     technicid int UNIQUE,
33
34     CONSTRAINT PK_orders_idorders PRIMARY KEY (idorders),
35     CONSTRAINT FK_orders_client FOREIGN KEY (clientid) REFERENCES public.client (idclient) ON DELETE
↵ CASCADE,
36     CONSTRAINT FK_orders_technic FOREIGN KEY (technicid) REFERENCES public.technic (idtechnic) ON
↵ DELETE CASCADE
37
38 );
39
40 CREATE TABLE IF NOT EXISTS public.rent
```

```

41 (
42     idrent int GENERATED ALWAYS AS IDENTITY NOT NULL,
43     rent_premises integer NOT NULL,
44     utilites integer NOT NULL,
45     advertisement integer,
46     CONSTRAINT PK_rent_idrent PRIMARY KEY (idrent)
47 );
48
49 CREATE TABLE IF NOT EXISTS public.service
50 (
51     idservice int GENERATED ALWAYS AS IDENTITY NOT NULL,
52     address character varying(100),
53     rentid int ,
54
55     CONSTRAINT PK_service_idservice PRIMARY KEY (idservice),
56     CONSTRAINT FK_service_rent FOREIGN KEY (rentid) REFERENCES public.rent (idrent) ON DELETE CASCADE,
57
58 );
59
60 CREATE TABLE IF NOT EXISTS public.issue
61 (
62     idissue int GENERATED ALWAYS AS IDENTITY NOT NULL,
63     description character varying(100) NOT NULL,
64     CONSTRAINT PK_issue_idissue PRIMARY KEY (idissue)
65 );
66
67 CREATE TABLE IF NOT EXISTS public.component
68 (
69     idcomponent int GENERATED ALWAYS AS IDENTITY NOT NULL,
70     description character varying(100) NOT NULL,
71     address character varying(100),
72     CONSTRAINT PK_component_idcomponent PRIMARY KEY (idcomponent)
73 );
74
75 CREATE TABLE IF NOT EXISTS public.master
76 (
77     idmaster int GENERATED ALWAYS AS IDENTITY NOT NULL,
78     fio character varying(60) NOT NULL,
79     CONSTRAINT PK_master_idmaster PRIMARY KEY (idmaster)
80 );
81
82 CREATE TABLE IF NOT EXISTS public.master_orders
83 (
84     master_idmaster integer NOT NULL,
85     orders_idorders integer NOT NULL,
86     master_price integer
87 );

```

```

88
89 CREATE TABLE IF NOT EXISTS public.master_service
90 (
91     master_idmaster integer NOT NULL,
92     service_idservice integer NOT NULL
93 );
94
95 CREATE TABLE IF NOT EXISTS public.technic_issue
96 (
97     technic_idtechnic integer NOT NULL,
98     issue_idissue integer NOT NULL
99 );
100
101 CREATE TABLE IF NOT EXISTS public.technic_component
102 (
103     technic_idtechnic integer NOT NULL,
104     component_idcomponent integer NOT NULL
105 );
106
107
108 ALTER TABLE IF EXISTS public.master_orders
109     ADD FOREIGN KEY (master_idmaster)
110     REFERENCES public.master (idmaster) MATCH SIMPLE ON DELETE CASCADE
111     ON UPDATE NO ACTION
112     ON DELETE NO ACTION
113     NOT VALID;
114
115
116 ALTER TABLE IF EXISTS public.master_orders
117     ADD FOREIGN KEY (orders_idorders)
118     REFERENCES public.orders (idorders) MATCH SIMPLE ON DELETE CASCADE
119     ON UPDATE NO ACTION
120     ON DELETE NO ACTION
121     NOT VALID;
122
123
124 ALTER TABLE IF EXISTS public.master_service
125     ADD FOREIGN KEY (master_idmaster)
126     REFERENCES public.master (idmaster) MATCH SIMPLE
127     ON UPDATE CASCADE
128     ON DELETE CASCADE
129     NOT VALID;
130
131
132 ALTER TABLE IF EXISTS public.master_service
133     ADD FOREIGN KEY (service_idservice)
134     REFERENCES public.service (idservice) MATCH SIMPLE

```

```

135     ON UPDATE CASCADE
136     ON DELETE CASCADE
137     NOT VALID;
138
139
140 ALTER TABLE IF EXISTS public.technic_issue
141     ADD FOREIGN KEY (technic_idtechnic)
142     REFERENCES public.technic (idtechnic) MATCH SIMPLE
143     ON UPDATE CASCADE
144     ON DELETE CASCADE
145     NOT VALID;
146
147
148 ALTER TABLE IF EXISTS public.technic_issue
149     ADD FOREIGN KEY (issue_idissue)
150     REFERENCES public.issue (idissue) MATCH SIMPLE
151     ON UPDATE CASCADE
152     ON DELETE CASCADE
153     NOT VALID;
154
155
156 ALTER TABLE IF EXISTS public.technic_component
157     ADD FOREIGN KEY (technic_idtechnic)
158     REFERENCES public.technic (idtechnic) MATCH SIMPLE
159     ON UPDATE CASCADE
160     ON DELETE CASCADE
161     NOT VALID;
162
163
164 ALTER TABLE IF EXISTS public.technic_component
165     ADD FOREIGN KEY (component_idcomponent)
166     REFERENCES public.component (idcomponent) MATCH SIMPLE
167     ON UPDATE CASCADE
168     ON DELETE CASCADE
169     NOT VALID;
170
171 END;

```

---