

МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное
учреждение высшего образования
«Национальный исследовательский университет «МЭИ»

Отчет по курсовой работе
по дисциплине «Базы данных»
по теме №19:
Разработка базы данных отдела техобслуживания

Работу выполнил
студент группы А-136-20
Бегунов Никита
Преподаватель:
Сидорова Наталья Петровна

Москва 2023

Оглавление

Задание.....	3
Введение	4
1. Анализ предметной области	5
1.1 Описание предметной области	5
1.2 Анализ функций процесса	5
2. Концептуальное проектирование базы данных.....	7
2.1 ER-модель	7
2.2 Проектирование правил целостности	9
3. Реализация базы данных	11
4. Программная реализация	15
4.1. Интеграция с базой данных	15
4.2. Примеры экранных форм интерфейса	15
Выводы по работе	19
Список литературы	20
Приложение 1. Код программы.....	21

Задание

Цель работы: разработка БД отдела техобслуживания компании. Компания имеет несколько филиалов, в каждом из которых есть несколько отделов. В каждом отделе находится техника различного типа (ПК, принтеры и д.). Отдел техобслуживания принимает заявки от руководителей филиалов на ремонт оборудования различного типа. Каждый вид ремонта имеет свой нормативный срок выполнения.

БД должна поддерживать выполнение следующих функций:

- Учет заявок на ремонтные работы
- Контроль выполнения заявок по исполнителям;
- Составление отчета невыполненных заявок по исполнителям;
- Составление отчета по заявкам, выполненным с превышением нормативного срока;
- Составление отчета о количестве заявок заданного типа;
- Изменение состава типа заявок;
- Изменение данных о руководителях филиалов.

Введение

Целью курсовой работы является разработка моделей базы данных (БД) и интерфейсных средств для выполнения указанных в задании функций. БД обеспечивает хранение информации и представляет собой поименованную совокупность данных, организованных по определенным правилам, включающим общие принципы описания, хранения и манипулирования данными. БД является информационной моделью предметной области, в которой решаются поставленные задачи.

Этапы выполнения курсовой работы:

- Анализ предметной области;
- Проектирование БД;
- Реализация БД средствами выбранной СУБД;
- Программная реализация интерфейса с БД.

Проектирование моделей БД выполняется с помощью CASE-средства AllFusion ERwin Data Modeler, входящий в состав пакета AllFusion Modeler Suite. Реализация БД осуществляется при помощи СУБД PostgreSQL. В качестве языка реализации выбран Python с использованием библиотеки PyQt5 в среде PyCharm.

1. Анализ предметной области

1.1 Описание предметной области

Предметная область курсовой работы: отдел техобслуживания компании.

В компании есть несколько филиалов, в каждом филиале есть несколько отделов. В каждом отделе находится техника различного типа (ПК, принтеры и другие). Отдел техобслуживания принимает заявки от руководителей филиалов на ремонт оборудования различного типа. Каждый вид ремонта имеет свой нормативный срок выполнения. Выполнение заявки по ремонту какой-то техники осуществляет один из сотрудников отдела техобслуживания.

В организации процесса выполнения заявок определены следующие правила:

- У каждого филиала есть руководитель;
- Техника из отделов относится хотя бы к одному из видов;
- У каждого вида ремонта есть нормативный срок выполнения;
- Выполнение заявки осуществляет один сотрудник.

1.2 Анализ функций процесса

Анализ функций позволил выделить сущности в заданной предметной области. Состав и характеристики сущностей приведены в таблице 1.2.1. Характеристика связей между сущностями приведена в таблице 1.2.2.

Таблица 1.2.1 – Описание сущностей и их атрибутов.

№ п/п	Имя сущности	Описание	Атрибуты
1	Филиал	Содержит информацию о филиале	Название филиала, ФИО руководителя
2	Отдел	Содержит данные об отделе	Филиал (FK), название отдела
3	Тип техники	Содержит информацию о типах техники	Наименование типов техники
4	Техника	Содержит информацию о технике в отделах	Название техники, тип техники (FK), филиал (FK), отдел (FK)
5	Вид ремонта	Содержит информацию о видах ремонтов	Название вида ремонта (FK), нормативный срок выполнения
6	Исполнитель	Содержит информацию об исполнителях ремонтов	ФИО исполнителя, должность

7	Заявка на ремонт	Содержит информацию о заявках на ремонт	Исполнитель (FK), филиал (FK), отдел (FK), статус заявки, дата заявки, техника (FK), вид ремонта (FK)
---	------------------	---	---

Таблица 1.2.2 – Отношение связей между сущностями.

№ п/п	Имя отношения	Описание
1	Филиал – отдел	Отдел находится в одном из филиалов
2	Отдел – техника	Техника находится в одном из отделов
3	Техника – тип техники	Техника относится к одному из видов
4	Филиал – заявка на ремонт	Руководитель филиала оставляет заявку на ремонт техники
5	Заявка на ремонт – техника	В заявке указывается техника для ремонта
6	Заявка на ремонт – вид ремонта	В заявке указывается вид ремонта
7	Заявка на ремонт – исполнитель	Исполнитель выполняет ремонт техники по заявке

2. Концептуальное проектирование базы данных

2.1 ER-модель

На основе выделенных сущностей и связей между ними была построена реляционная ER-модель средствами AllFusion ERwin Data Modeler.

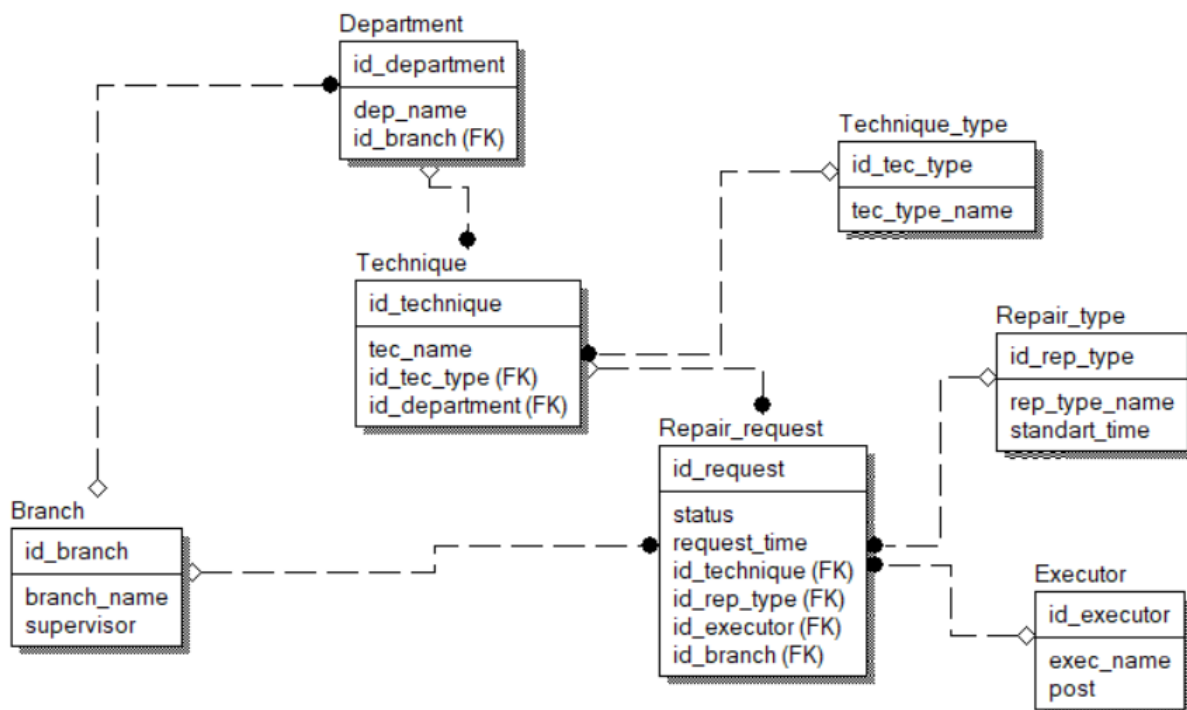


Рис. 2.1. Реляционная ER-модель БД

Описание атрибутов сущностей приведены в таблицах 2.1.1 – 2.1.7

Таблица 2.1.1 – Описание атрибутов таблицы Branch

Имя атрибута	Описание
id_branch	Идентификатор филиала (PK)
branch_name	Название филиала
supervisor	ФИО руководителя филиала

Таблица 2.1.2 – Описание атрибутов таблицы Department

Имя атрибута	Описание
id_department	Идентификатор отдела (PK)
dep_name	Название отдела
id_branch	Идентификатор филиала (FK)

Таблица 2.1.3 – Описание атрибутов таблицы Technique_type

Имя атрибута	Описание
id_tec_type	Идентификатор типа техники (PK)
tec_type_name	Название типа техники

Таблица 2.1.4 – Описание атрибутов таблицы Repair_type

Имя атрибута	Описание
id_rep_type	Идентификатор вида ремонта (PK)
rep_type_name	Название вида ремонта
standart_time	Нормативный срок выполнения

Таблица 2.1.5 – Описание атрибутов таблицы Technique

Имя атрибута	Описание
id_technique	Идентификатор техники (PK)
tec_name	Название техники
id_tec_type	Идентификатор типа техники (FK)
id_department	Идентификатор отдела (FK)

Таблица 2.1.6 – Описание атрибутов таблицы Executor

Имя атрибута	Описание
id_executor	Идентификатор исполнителя
exec_name	Имя исполнителя
post	Должность

Таблица 2.1.7 – Описание атрибутов таблицы Repair_request

Имя атрибута	Описание
id_request	Идентификатор заявки на ремонт (PK)
status	Статус ремонта
request_time	Дата и время заявки на ремонт
id_technique	Идентификатор техники из заявки (FK)
id_rep_type	Идентификатор вида ремонта техники (FK)
id_executor	Идентификатор исполнителя (FK)
id_branch	Идентификатор филиала, откуда поступила заявка (FK)

Для избавления от избыточности данных в базе данных необходимо, чтобы она находилась в третьей нормальной форме. Для это необходимо обеспечить атомарность атрибутов, выделить функциональные зависимости между атрибутами и проанализировать их. Функциональные зависимости между атрибутами в таблицах приведены в таблице 2.1.8.

Таблица 2.1.8 – Функциональные зависимости между атрибутами

Таблица	Зависимости
Branch	id_branch → branch_name id_branch → supervisor
Department	id_department → dep_name id_department → id_branch
Technique_type	id_tec_type → tec_type_name
Technique	id_technique → tec_name

	id_technique → id_tec_type id_technique → id_department
Repair_type	id_rep_id → rep_type_name id_rep_id → standart_time
Executor	id_executor → exec_name id_executor → post
Repair_request	id_request → status id_request → request_time id_request → id_technique id_request → id_rep_type id_request → id_executor id_request → id_branch

Обоснование нахождения базы данных в третьей нормальной форме:

1. Все таблицы базы данных находятся в 1НФ, так как все атрибуты являются простыми и все домены содержат только скалярные значения;
2. Все таблицы базы данных находятся в 2НФ, так как они находятся в 1НФ и каждый не ключевой атрибут зависит от первичного ключа (что видно из таблицы 2.1.8 в функциональных зависимостях между атрибутами);
3. Все таблицы базы данных находятся в 3НФ, так как они находятся в 2НФ и каждый не ключевой атрибут не транзитивно зависит от ключа.

Таким образом, база данных находится в третьей нормальной форме.

2.2 Проектирование правил целостности

Общие правила целостности:

- Все первичные ключи – натуральные числа, NOT NULL;
- Текстовые атрибуты – максимальная длина ограничена 128 символами, но может быть и меньше.

Правила сущностной целостности соблюдены наличием в каждой из таблиц уникального первичного ключа. Ограничения на значения атрибутов приведены в таблице 2.2.1.

Таблица 2.2.1 – Ограничения на значения атрибутов

Таблица	Атрибут	PK	FK	Ограничения
Branch	id_branch	Yes	No	INTEGER NOT NULL
	branch_name	No	No	VARCHAR(64) NOT NULL
	supervisor	No	No	VARCHAR(64) NOT NULL
Department	id_department	Yes	No	INTEGER NOT NULL
	dep_name	No	No	VARCHAR(64) NOT NULL
	id_branch	No	Yes	INTEGER NOT NULL

Technique_type	id_tec_type	Yes	No	INTEGER NOT NULL
	tec_type_name	No	No	VARCHAR(64) NOT NULL
Technique	id_technique	Yes	No	INTEGER NOT NULL
	tec_name	No	No	VARCHAR(64) NOT NULL
	id_tec_type	No	Yes	INTEGER NOT NULL
	id_department	No	Yes	INTEGER NOT NULL
Repair_type	id_rep_type	Yes	No	INTEGER NOT NULL
	rep_type_name	No	No	VARCHAR(128) NOT NULL
	standart_time	No	No	INTERVAL
Executor	id_executor	Yes	No	INTEGER NOT NULL
	exec_name	No	No	VARCHAR(64) NOT NULL
	post	No	No	VARCHAR(128)
Repair_request	id_request	Yes	No	INTEGER NOT NULL
	status	No	No	VARCHAR(32) status IN ('заявка получена', 'ремонт', 'готов', 'отмена')
	request_time	No	No	TIMESTAMP
	id_technique	No	Yes	INTEGER NOT NULL
	id_rep_type	No	Yes	INTEGER NOT NULL
	id_executor	No	Yes	INTEGER NOT NULL
	id_branch	No	Yes	INTEGER NOT NULL

Правила ссылочной целостности приведены в таблице 2.2.2.

Таблица 2.2.2 – Правила ссылочной целостности

Имя отношения	Parent		Child	
	Update	Delete	Insert	Update
Branch- Department	CASCADE	SET NULL	RESTRICT	RESTRICT
Department- Technique	CASCADE	SET NULL	RESTRICT	RESTRICT
Technique_type- Technique	CASCADE	SET NULL	RESTRICT	RESTRICT
Branch- Repair_request	CASCADE	SET NULL	RESTRICT	RESTRICT
Repair_type- Repair_request	CASCADE	SET NULL	RESTRICT	RESTRICT
Executor- Repair_request	CASCADE	SET NULL	RESTRICT	RESTRICT
Technique- Repair_request	CASCADE	CASCADE	RESTRICT	RESTRICT

3. Реализация базы данных

В качестве СУБД для реализации базы данных в курсовом проекте была выбрана PostgreSQL. Эта система управления базами данных является мощным инструментом с открытым исходным кодом, разработанным сообществом и поддерживаемым в коммерческих целях. PostgreSQL также следует реляционной модели данных и поддерживает широкий спектр функций для разработки и управления данными.

Основной язык запросов, используемый в PostgreSQL, — это SQL (Structured Query Language), а также присутствуют расширения языка SQL, специфичные для PostgreSQL. Система поддерживает множество функций и процедур, которые могут быть написаны на различных языках программирования, таких как PL/pgSQL, PL/Python, PL/Perl и других.

PostgreSQL может быть использована для работы с разнообразными базами данных, от небольших до крупномасштабных проектов. Ниже приведено описание структуры базы данных с использованием SQL-запросов в PostgreSQL.

```
CREATE TABLE IF NOT EXISTS Branch
(
    id_branch SERIAL PRIMARY KEY,
    branch_name VARCHAR(64) NOT NULL,
    supervisor VARCHAR(64) NOT NULL
);

CREATE TABLE IF NOT EXISTS Department
(
    id_department SERIAL PRIMARY KEY,
    dep_name VARCHAR(64) NOT NULL,
    id_branch INTEGER NOT NULL REFERENCES Branch(id_branch)
);

CREATE TABLE IF NOT EXISTS Technique_type
(
    id_tec_type SERIAL PRIMARY KEY,
    tec_type_name VARCHAR(64) NOT NULL
);

CREATE TABLE IF NOT EXISTS Technique
(
    id_technique SERIAL PRIMARY KEY,
    tec_name VARCHAR(64) NOT NULL,
    id_department INTEGER NOT NULL REFERENCES Department(id_department),
    id_tec_type INTEGER NOT NULL REFERENCES Technique_type(id_tec_type)
);
```

```

CREATE TABLE IF NOT EXISTS Repair_type
(
    id_rep_type SERIAL PRIMARY KEY,
    rep_type_name VARCHAR(128) NOT NULL,
    standart_time INTERVAL
);

CREATE TABLE IF NOT EXISTS Executor
(
    id_executor SERIAL PRIMARY KEY,
    exec_name VARCHAR(64) NOT NULL,
    post VARCHAR(128)
);

CREATE TABLE IF NOT EXISTS Repair_request
(
    id_request SERIAL PRIMARY KEY,
    status VARCHAR(32),
    request_time TIMESTAMP,
    id_technique INTEGER NOT NULL REFERENCES Technique(id_techique),
    id_rep_type INTEGER NOT NULL REFERENCES Repair_type(id_rep_type),
    id_executor INTEGER NOT NULL REFERENCES Executor(id_executor),
    id_branch INTEGER NOT NULL REFERENCES Branch(id_branch),
    CHECK (status = 'заявка получена' OR
           status = 'ремонт' OR
           status = 'готов' OR
           status = 'отмена')
);

```

Хранимые процедуры

```

-- Заявки на ремонтные работы
CREATE OR REPLACE FUNCTION get_requests()
RETURNS TABLE (
    id_request INTEGER,
    exec_name VARCHAR,
    post VARCHAR,
    tec_name VARCHAR,
    tec_type_name VARCHAR,
    repair_type VARCHAR,
    request_time TIMESTAMP,
    deadline TIMESTAMP)
AS $$
BEGIN
    RETURN QUERY
    SELECT rr.id_request,
           exe.exec_name,
           exe.post,
           tec.tec_name,
           tt.tec_type_name,
           rt.rep_type_name,

```

```

        rr.request_time,
        rr.request_time + rt.standart_time
FROM repair_request rr
JOIN executor exe ON rr.id_executor = exe.id_executor
JOIN repair_type rt ON rr.id_rep_type = rt.id_rep_type
JOIN technique tec ON rr.id_technique = tec.id_technique
JOIN technique_type tt ON tec.id_tec_type = tt.id_tec_type
ORDER BY id_request ASC;
END;
$$ LANGUAGE plpgsql;

-- Контроль выполнения заявок по исполнителям
CREATE OR REPLACE FUNCTION get_requests_by_executor(
    IN executor_param VARCHAR)
RETURNS TABLE (
    id_request INTEGER,
    exec_name VARCHAR,
    post VARCHAR,
    tec_name VARCHAR,
    tec_type_name VARCHAR,
    repair_type VARCHAR,
    request_time TIMESTAMP,
    deadline TIMESTAMP)
AS $$
BEGIN
    RETURN QUERY
    SELECT rr.id_request,
           exe.exec_name,
           exe.post,
           tec.tec_name,
           tt.tec_type_name,
           rt.rep_type_name,
           rr.request_time,
           rr.request_time + rt.standart_time
    FROM repair_request rr
    JOIN executor exe ON rr.id_executor = exe.id_executor
    JOIN repair_type rt ON rr.id_rep_type = rt.id_rep_type
    JOIN technique tec ON rr.id_technique = tec.id_technique
    JOIN technique_type tt ON tec.id_tec_type = tt.id_tec_type
    WHERE exe.exec_name = executor_param
    ORDER BY id_request ASC;
END;
$$ LANGUAGE plpgsql;

-- Отчет о количестве заявок заданного типа
CREATE OR REPLACE FUNCTION get_requests_by_type()
RETURNS TABLE (
    rep_type_name VARCHAR,
    count_by_type BIGINT)
AS $$
BEGIN

```

```

RETURN QUERY
SELECT rt.rep_type_name,
       COUNT(*) AS count_by_type
FROM repair_request rr
JOIN repair_type rt ON rr.id_rep_type = rt.id_rep_type
GROUP BY rt.rep_type_name
ORDER BY count_by_type DESC, rt.rep_type_name ASC;
END;
$$ LANGUAGE plpgsql;

-- Отчет о количестве заявок заданного типа
CREATE OR REPLACE FUNCTION get_requests_deadline()
RETURNS TABLE (
    id_request INTEGER,
    status VARCHAR,
    request_time TIMESTAMP,
    standart_time INTERVAL,
    delay INTERVAL,
    tec_name VARCHAR,
    tec_type_name VARCHAR,
    exec_name VARCHAR
)
AS $$
BEGIN
    RETURN QUERY
    SELECT rr.id_request,
           rr.status,
           rr.request_time,
           rt.standart_time,
           now() - (rr.request_time + rt.standart_time) AS delay,
           tec.tec_name,
           tt.tec_type_name,
           ex.exec_name
    FROM repair_request rr
    JOIN repair_type rt ON rr.id_rep_type = rt.id_rep_type
    JOIN executor ex ON rr.id_executor = ex.id_executor
    JOIN technique tec ON rr.id_technique = tec.id_technique
    JOIN technique_type tt ON tec.id_tec_type = tt.id_tec_type
    WHERE DATE_PART('microseconds', now() - (rr.request_time + rt.standart_time)) > 0
    AND rr.status IN ('заявка получена', 'ремонт')
    ORDER BY rr.status ASC, delay DESC;
END;
$$ LANGUAGE plpgsql;

```

4. Программная реализация

4.1. Интеграция с базой данных

Psycopg2 — это библиотека для Python, которая обеспечивает взаимодействие с базами данных PostgreSQL. Она предоставляет разработчикам набор инструментов и классов для работы с данными, выполнения запросов к базе данных, установления соединений и выполнения других операций.

Важно отметить, что psycopg2 ориентирован на работу с конкретной базой данных - PostgreSQL, хотя сама библиотека обеспечивает унифицированный интерфейс для взаимодействия с ней. Она предоставляет возможность использовать стандартные объекты, такие как:

- Connection: для установки и управления соединением с базой данных PostgreSQL;
- Cursor: позволяет выполнять SQL-запросы и манипулировать данными;
- ResultProxy: предоставляет доступ к результатам выполнения SQL-запросов;
- Pool: для управления пулом соединений с базой данных.

Хотя psycopg2 ориентирован на PostgreSQL, он предоставляет разработчикам удобный и унифицированный интерфейс для работы с данными и выполнения операций независимо от конкретной СУБД.

4.2. Примеры экранных форм интерфейса

Пользовательский интерфейс реализуется на языке программирования Python с использованием среды разработки PyCharm. Для реализации окон используется библиотека PyQt5. Ниже показаны примеры окон пользовательского интерфейса.

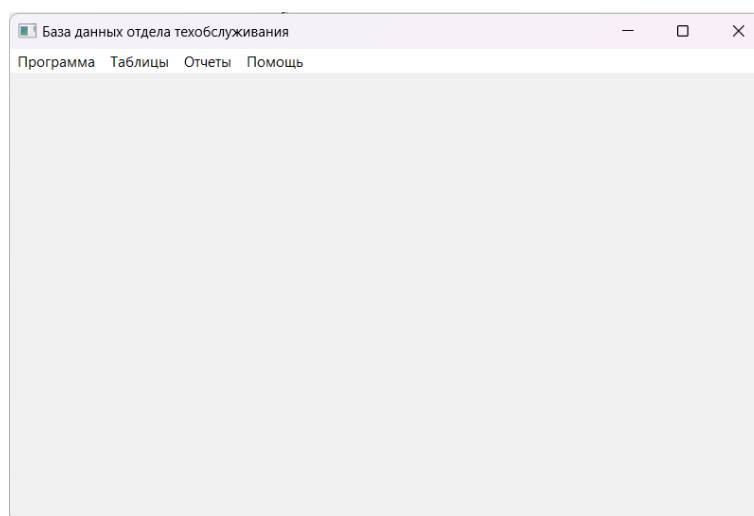


Рис. 4.2.1. Главная форма приложения

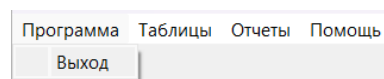


Рис. 4.2.2. Меню «Программа»

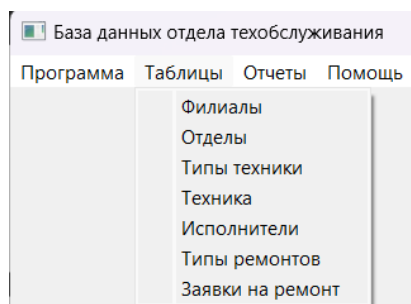


Рис. 4.2.3. Меню «Таблицы»

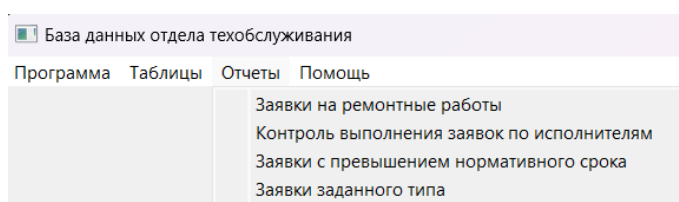


Рис. 4.2.4. Меню «Отчеты»

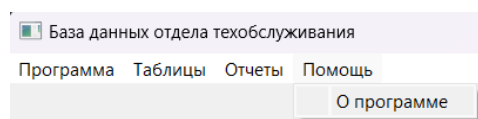


Рис. 4.2.5. Меню «Помощь»

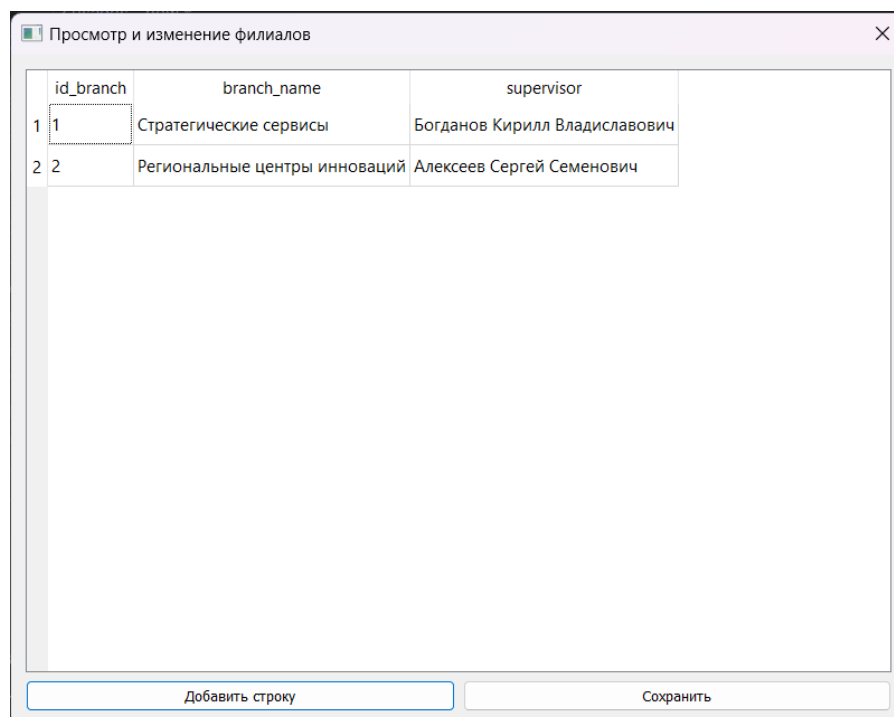


Рис. 4.2.6. Просмотр и изменение филиалов

Заявки на ремонтные работы								
	id_request	exec_name	post	tec_name	tec_type_name	repair_type	request_time	deadline
1	1	Иванов Алексей Петрович	Инженер по обслуживанию систем	PowerTech ProTower	Системный блок	Замена жесткого диска	2023-11-15 11:01:00	2023-11-15 12:01:00
2	2	Иванов Алексей Петрович	Инженер по обслуживанию систем	PowerTech ProTower	Системный блок	Замена жесткого диска	2023-11-15 11:01:00	2023-11-15 12:01:00
3	3	Смирнова Екатерина Андреевна	Специалист технической поддержки	PrintPro Xpress	Принтер	Замена картриджа	2023-11-16 15:35:53	2023-11-16 16:35:53
4	4	Петров Дмитрий Игоревич	Технический ассистент	PrintPro Xpress	Принтер	Ремонт печатающего механизма	2023-11-17 08:12:45	2023-11-20 08:12:45
5	5	Козлова Ольга Васильевна	Инженер-технолог по ремонту оборудования	EliteForce Desktop	Системный блок	Ремонт материнской платы	2023-11-19 16:06:10	2023-11-20 16:06:10
6	6	Петров Дмитрий Игоревич	Технический ассистент	HyperStream Workstation	Системный блок	Замена жесткого диска	2023-11-19 17:01:00	2023-11-19 18:01:00
7	7	Смирнова Екатерина Андреевна	Специалист технической поддержки	InkJet ElitePrint	Принтер	Замена картриджа	2023-11-16 19:41:55	2023-11-16 20:41:55
8	8	Никитин Артем Сергеевич	Специалист по сервисной поддержке	VisionPeak Display	Монитор	Замена матрицы	2023-11-10 16:25:25	2023-11-12 16:25:25
9	9	Иванов Алексей Петрович	Инженер по обслуживанию систем	UltraView Monitor	Монитор	Замена матрицы	2023-11-13 14:03:13	2023-11-15 14:03:13
10	10	Никитин Артем Сергеевич	Специалист по сервисной поддержке	LaserX ProSeries	Принтер	Ремонт печатающего механизма	2023-11-14 12:32:11	2023-11-17 12:32:11
11	11	Смирнова Екатерина Андреевна	Специалист технической поддержки	EliteForce Desktop	Системный блок	Ремонт материнской платы	2023-11-16 16:11:54	2023-11-17 16:11:54

Рис. 4.2.7. Заявки на ремонтные работы

Контроль выполнения заявок по исполнителям

ФИО исполнителя: Иванов Алексей Петрович

Ввод

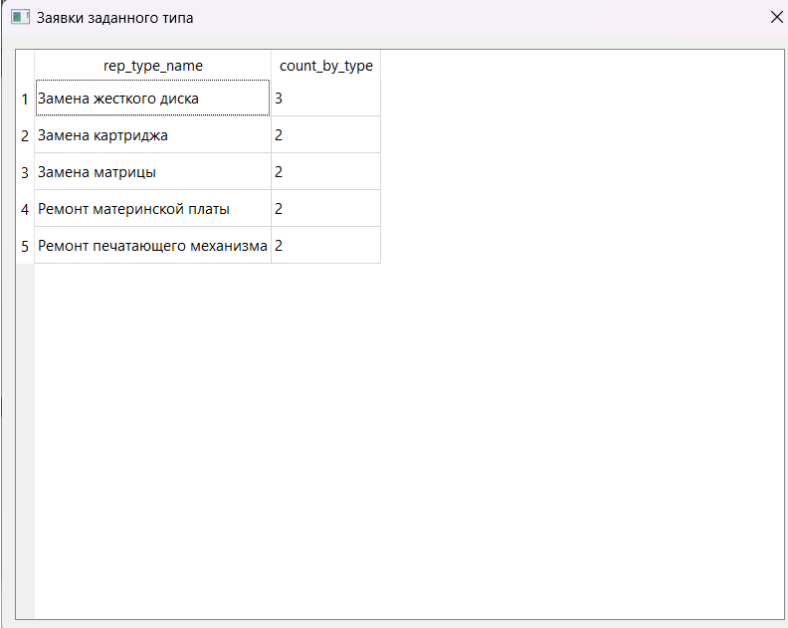
	id_request	exec_name	post	tec_name	tec_type_name	repair_type	request_time	deadline
1	1	Иванов Алексей Петрович	Инженер по обслуживанию систем	PowerTech ProTower	Системный блок	Замена жесткого диска	2023-11-15 11:01:00	2023-11-15 12:01:00
2	2	Иванов Алексей Петрович	Инженер по обслуживанию систем	PowerTech ProTower	Системный блок	Замена жесткого диска	2023-11-15 11:01:00	2023-11-15 12:01:00
3	9	Иванов Алексей Петрович	Инженер по обслуживанию систем	UltraView Monitor	Монитор	Замена матрицы	2023-11-13 14:03:13	2023-11-15 14:03:13

Рис. 4.2.8. Контроль выполнения заявок по исполнителям

Заявки с превышением нормативного срока

	id_request	status	request_time	standart_time	delay	tec_name	tec_type_name	exec_name
1	2	заявка получена	2023-11-15 11:01:00	1:00:00	7 days, 9:39:29.176335	PowerTech ProTower	Системный блок	Иванов Алексей Петрович
2	1	заявка получена	2023-11-15 11:01:00	1:00:00	7 days, 9:39:29.176335	PowerTech ProTower	Системный блок	Иванов Алексей Петрович
3	9	заявка получена	2023-11-13 14:03:13	2 days, 0:00:00	7 days, 7:37:16.176335	UltraView Monitor	Монитор	Иванов Алексей Петрович
4	7	ремонт	2023-11-16 19:41:55	1:00:00	6 days, 0:58:34.176335	Inklet ElitePrint	Принтер	Смирнова Екатерина Андреевна
5	11	ремонт	2023-11-16 16:11:54	1 day, 0:00:00	5 days, 5:28:35.176335	EliteForce Desktop	Системный блок	Смирнова Екатерина Андреевна
6	6	ремонт	2023-11-19 17:01:00	1:00:00	3 days, 3:39:29.176335	HyperStream Workstation	Системный блок	Петров Дмитрий Игоревич
7	4	ремонт	2023-11-17 08:12:45	3 days, 0:00:00	2 days, 13:27:44.176335	PrintPro Xpress	Принтер	Петров Дмитрий Игоревич

Рис. 4.2.9. Заявки с превышением нормативного срока



Заявки заданного типа

	rep_type_name	count_by_type
1	Замена жесткого диска	3
2	Замена картриджа	2
3	Замена матрицы	2
4	Ремонт материнской платы	2
5	Ремонт печатающего механизма	2

Рис. 4.2.10. Заявки заданного типа

Выводы по работе

В результате выполнения курсовой работы:

1. Проведен анализ отдела техобслуживания компании:
 - Выделены и сформулированы ключевые бизнес-правила;
 - Выделены сущности, участвующие в процессе, и их атрибуты.
2. Спроектирована модель БД, состоящая из семи таблиц:
 - Разработаны ER-модель и реляционная модель данных;
 - Модель данных доведена до требуемой 3НФ.
3. Разработаны программные средства интерфейса для взаимодействия с БД, реализующая поставленные в введении требования.

По итогам проделанной работы были сформированы и закреплены необходимые навыки разработки баз данных, а также интерфейса к ним для удобства использования.

Список литературы

1. Сидорова Н.П. Базы данных. Практикум по проектированию реляционных баз данных. М.: издательство «Директ-медиа», 2020. – 93с.
2. PyQt5 Reference Guide // Riverbank Computing сайт. – URL: <https://www.riverbankcomputing.com/static/Docs/PyQt5/>

Приложение 1. Код программы

Файл main.py

```
import sys
from MainWindow import MainWindow
from PyQt5.QtWidgets import QApplication

def main():
    app = QApplication(sys.argv)
    win = MainWindow()
    win.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    main()
```

Файл MainWindow.py

```
from PyQt5.QtWidgets import QWidget, QMainWindow, QAction, QVBoxLayout
from AboutWindow import AboutDialog
from TableEditWindows.BranchEditWindow import BranchEditDialog
from TableEditWindows.DepartmentEditWindow import DepartmentEditDialog
from TableEditWindows.ExecutorEditWindow import ExecutorEditDialog
from TableEditWindows.RepairRequestEditWindow import RepairRequestEditDialog
from TableEditWindows.RepairTypeEditWindow import RepairTypeEditDialog
from TableEditWindows.TechniqueEditWindow import TechniqueEditDialog
from TableEditWindows.TechniqueTypeEditWindow import TechniqueTypeEditDialog
from ReportWindows.RequestsWindow import RequestsDialog
from ReportWindows.RequestsByExecutorWindow import RequestsByExecutorDialog
from ReportWindows.RequestsByTypeWindow import RequestsByTypeDialog
from ReportWindows.RequestsDeadlineWindow import RequestsDeadlineDialog

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('База данных отдела техобслуживания')
        self.resize(800, 500)

        layout = QVBoxLayout()

        self.setLayout(layout)
        central_widget = QWidget()
        central_widget.setLayout(layout)
        self.setCentralWidget(central_widget)

        self.create_menu_bar()

    def create_menu_bar(self):
        menu_bar = self.menuBar()

        file_menu = menu_bar.addMenu('Программа')
        close_action = QAction('Выход', self)
        close_action.triggered.connect(self.close_action)
        file_menu.addAction(close_action)

        tables_edit = menu_bar.addMenu('Таблицы')
        table_branch = QAction('Филиалы', self)
        table_branch.triggered.connect(self.table_branch_edit_action)
```

```

        tables_edit.addAction(table_branch)
        table_department = QAction('Отделы', self)
        table_department.triggered.connect(self.table_department_edit_action)
        tables_edit.addAction(table_department)
        table_technique_type = QAction('Типы техники', self)

    table_technique_type.triggered.connect(self.table_tec_type_edit_action)
        tables_edit.addAction(table_technique_type)
        table_technique = QAction('Техника', self)
        table_technique.triggered.connect(self.table_technique_edit_action)
        tables_edit.addAction(table_technique)
        table_executor = QAction('Исполнители', self)
        table_executor.triggered.connect(self.table_executor_edit_action)
        tables_edit.addAction(table_executor)
        table_repair_type = QAction('Типы ремонтов', self)

    table_repair_type.triggered.connect(self.table_repair_type_edit_action)
        tables_edit.addAction(table_repair_type)
        table_repair_request = QAction('Заявки на ремонт', self)

    table_repair_request.triggered.connect(self.table_repair_request_edit_action)
        tables_edit.addAction(table_repair_request)

        reports = menu_bar.addMenu('Отчеты')
        table_requests = QAction('Заявки на ремонтные работы', self)
        table_requests.triggered.connect(self.table_requests_action)
        reports.addAction(table_requests)
        table_requests_by_executor = QAction('Контроль выполнения заявок по
исполнителям', self)

    table_requests_by_executor.triggered.connect(self.table_requests_by_executor_
action)
        reports.addAction(table_requests_by_executor)
        table_requests_deadline = QAction('Заявки с превышением нормативного
срока', self)

    table_requests_deadline.triggered.connect(self.table_requests_deadline_action
)
        reports.addAction(table_requests_deadline)
        table_requests_by_type = QAction('Заявки заданного типа', self)

    table_requests_by_type.triggered.connect(self.table_requests_by_type_action)
        reports.addAction(table_requests_by_type)

        help_menu = menu_bar.addMenu('Помощь')
        about = QAction('О программе', self)
        about.triggered.connect(self.about_action)
        help_menu.addAction(about)

    @staticmethod
    def close_action():
        exit()

    @staticmethod
    def about_action():
        about = AboutDialog()
        about.exec_()

    @staticmethod
    def table_branch_edit_action():
        br = BranchEditDialog()

```

```

        br.exec_()

    @staticmethod
    def table_department_edit_action():
        dep = DepartmentEditDialog()
        dep.exec_()

    @staticmethod
    def table_tec_type_edit_action():
        tt = TechniqueTypeEditDialog()
        tt.exec_()

    @staticmethod
    def table_technique_edit_action():
        tec = TechniqueEditDialog()
        tec.exec_()

    @staticmethod
    def table_executor_edit_action():
        executor = ExecutorEditDialog()
        executor.exec_()

    @staticmethod
    def table_repair_type_edit_action():
        rep_type = RepairTypeEditDialog()
        rep_type.exec_()

    @staticmethod
    def table_repair_request_edit_action():
        rep_request = RepairRequestEditDialog()
        rep_request.exec_()

    @staticmethod
    def table_requests_action():
        rd = RequestsDialog()
        rd.exec_()

    @staticmethod
    def table_requests_by_executor_action():
        rbed = RequestsByExecutorDialog()
        rbed.exec_()

    @staticmethod
    def table_requests_deadline_action():
        rdd = RequestsDeadlineDialog()
        rdd.exec_()

    @staticmethod
    def table_requests_by_type_action():
        rbtd = RequestsByTypeDialog()
        rbtd.exec_()

```

Файл AboutWindow.py

```

from PyQt5.QtWidgets import QDialog, QLabel, QVBoxLayout, QPushButton
from PyQt5.QtGui import QFont
from PyQt5.QtCore import Qt

class AboutDialog(QDialog):
    def __init__(self):
        super().__init__()

```

```

        self.setWindowFlags(self.windowFlags() &
(~Qt.WindowContextHelpButtonHint))
        self.setWindowTitle("О программе")
        self.resize(400, 300)

        layout = QVBoxLayout()

        label1 = QLabel("База данных отдела техобслуживания")
        label1.setFont(QFont('Arial', 14))
        label1.setAlignment(Qt.AlignHCenter | Qt.AlignVCenter)
        label2 = QLabel('Курсовая работа по дисциплине "Базы данных"\n'
                        'Работу выполнил студент группы А-13б-20\n'
                        'Бегунов Никита Сергеевич\n'
                        'Тема №19')
        label2.setFont(QFont('Arial', 10))
        layout.addWidget(label1)
        layout.addWidget(label2)

        close_button = QPushButton("Заккрыть")
        close_button.clicked.connect(self.close)
        layout.addWidget(close_button)

        self.setLayout(layout)

```

В папке TableEditWindows файл BranchEditWindow.py

```

from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QHBoxLayout,
                             QPushButton, QTableWidgetItem,
                             QMessageBox)
from PyQt5.QtCore import Qt
import psycopg2

class BranchEditDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.conn =
psycopg2.connect('postgresql://postgres:postgres@localhost:5432/course_work_d
b')

        self.index_column = 0

        self.setWindowFlags(self.windowFlags() &
(~Qt.WindowContextHelpButtonHint))
        self.setWindowTitle("Просмотр и изменение филиалов")
        self.resize(800, 600)

        cursor = self.conn.cursor()
        cursor.execute('SELECT * FROM branch ORDER BY id_branch')
        self.rows = cursor.fetchall()
        columns = [desc[0] for desc in cursor.description]
        cursor.close()

        layout = QVBoxLayout()

        self.table = QTableWidgetItem(cursor.rowcount, len(columns))
        self.table.setHorizontalHeaderLabels(columns)
        self.table.itemChanged.connect(self.item_changed)
        layout.addWidget(self.table)

        layout_buttons = QHBoxLayout()
        add_line_button = QPushButton('Добавить строку')

```



```

add_line_button.clicked.connect(self.add_line_btn_clicked)
layout_buttons.addWidget(add_line_button)
save_button = QPushButton('Сохранить')
save_button.clicked.connect(self.save_btn_clicked)
layout_buttons.addWidget(save_button)
layout.addLayout(layout_buttons)

self.setLayout(layout)

for i in range(len(self.rows)):
    for j in range(len(columns)):
        item = QTableWidgetItem(str(self.rows[i][j]))
        if j == self.index_column:
            item.setFlags(item.flags() & ~Qt.ItemIsEditable)
        self.table.setItem(i, j, item)
self.table.resizeColumnsToContents()

def add_line_btn_clicked(self):
    row_position = self.table.rowCount()
    self.table.insertRow(row_position)
    try:
        if row_position == 0:
            pred_index = 0
        else:
            pred_index = int(self.table.item(row_position-1, 0).text())
        item = QTableWidgetItem(str(pred_index + 1))
        item.setFlags(item.flags() & ~Qt.ItemIsEditable)
        self.table.setItem(row_position, 0, item)
    except:
        pass

def item_changed(self):
    self.table.resizeColumnsToContents()

def save_btn_clicked(self):
    try:
        data = []
        for r in range(self.table.rowCount()):
            row = []
            for c in range(self.table.columnCount()):
                row.append(self.table.item(r, c).text())
            data.append(row)
        for r in range(len(self.rows)):
            the_same = True
            for c in range(len(self.rows[r])):
                if str(data[r][c]) != str(self.rows[r][c]):
                    the_same = False
            if not the_same:
                cur = self.conn.cursor()
                cur.execute(f"""UPDATE branch SET
branch_name='{data[r][1]}',
supervisor='{data[r][2]}' WHERE
id_branch={data[r][0]}""")
                self.conn.commit()
                cur.close()
            if len(self.rows) < len(data):
                for r in range(len(self.rows), len(data)):
                    cur = self.conn.cursor()
                    cur.execute(f"""INSERT INTO branch (branch_name,
supervisor)
VALUES ('{data[r][1]}', '{data[r][2]}')""")

```

```

        self.conn.commit()
        cur.close()
        QMessageBox.about(self, 'Изменения сохранены', 'Изменения успешно
сохранены')
    except:
        QMessageBox.about(self, 'Ошибка', 'Неверное значение в таблице')

```

В папке TableEditWindows файл DepartmentEditWindow.py

```

from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QHBoxLayout,
                             QPushButton, QTableWidgetItem,
                             QMessageBox)
from PyQt5.QtCore import Qt
import psycopg2

class DepartmentEditDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.conn =
psycopg2.connect('postgresql://postgres:postgres@localhost:5432/course_work_d
b')

        self.index_column = 0

        self.setWindowFlags(self.windowFlags() &
(~Qt.WindowContextHelpButtonHint))
        self.setWindowTitle("Просмотр и изменение отделов")
        self.resize(800, 600)

        cursor = self.conn.cursor()
        cursor.execute('SELECT * FROM department ORDER BY id_department')
        self.rows = cursor.fetchall()
        columns = [desc[0] for desc in cursor.description]
        cursor.close()

        layout = QVBoxLayout()

        self.table = QTableWidgetItem(cursor.rowcount, len(columns))
        self.table.setHorizontalHeaderLabels(columns)
        self.table.itemChanged.connect(self.item_changed)
        layout.addWidget(self.table)

        layout_buttons = QHBoxLayout()
        add_line_button = QPushButton('Добавить строку')
        add_line_button.clicked.connect(self.add_line_btn_clicked)
        layout_buttons.addWidget(add_line_button)
        save_button = QPushButton('Сохранить')
        save_button.clicked.connect(self.save_btn_clicked)
        layout_buttons.addWidget(save_button)
        layout.addLayout(layout_buttons)

        self.setLayout(layout)

        for i in range(len(self.rows)):
            for j in range(len(columns)):
                item = QTableWidgetItem(str(self.rows[i][j]))
                if j == self.index_column:
                    item.setFlags(item.flags() & ~Qt.ItemIsEditable)
                self.table.setItem(i, j, item)
            self.table.resizeColumnsToContents()

    def add_line_btn_clicked(self):

```

```

row_position = self.table.rowCount()
self.table.insertRow(row_position)
try:
    if row_position == 0:
        pred_index = 0
    else:
        pred_index = int(self.table.item(row_position-1, 0).text())
    item = QTableWidgetItem(str(pred_index + 1))
    item.setFlags(item.flags() & ~Qt.ItemIsEditable)
    self.table.setItem(row_position, 0, item)
except:
    pass

def item_changed(self):
    self.table.resizeColumnsToContents()

def save_btn_clicked(self):
    try:
        data = []
        for r in range(self.table.rowCount()):
            row = []
            for c in range(self.table.columnCount()):
                row.append(self.table.item(r, c).text())
            data.append(row)
        for r in range(len(self.rows)):
            the_same = True
            for c in range(len(self.rows[r])):
                if str(data[r][c]) != str(self.rows[r][c]):
                    the_same = False
            if not the_same:
                cur = self.conn.cursor()
                cur.execute(f"""UPDATE department SET
dep_name='{data[r][1]}',
                                id_branch={data[r][2]} WHERE
id_department={data[r][0]}""")
                self.conn.commit()
                cur.close()
            if len(self.rows) < len(data):
                for r in range(len(self.rows), len(data)):
                    cur = self.conn.cursor()
                    cur.execute(f"""INSERT INTO department (dep_name,
id_branch)
                                VALUES ('{data[r][1]}', {data[r][2]})""")
                    self.conn.commit()
                    cur.close()
                QMessageBox.about(self, 'Изменения сохранены', 'Изменения успешно
сохранены')
    except:
        QMessageBox.about(self, 'Ошибка', 'Неверное значение в таблице')

```

В папке TableEditWindows файл ExecutorEditWindow.py

```

from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QHBoxLayout,
                             QPushButton, QTableWidgetItem,
                             QMessageBox)
from PyQt5.QtCore import Qt
import psycopg2

class ExecutorEditDialog(QDialog):
    def __init__(self):
        super().__init__()

```

```

        self.conn =
psycopg2.connect('postgresql://postgres:postgres@localhost:5432/course_work_d
b')

        self.index_column = 0

        self.setWindowFlags(self.windowFlags() &
(~Qt.WindowContextHelpButtonHint))
        self.setWindowTitle("Просмотр и изменение исполнителей")
        self.resize(800, 600)

        cursor = self.conn.cursor()
        cursor.execute('SELECT * FROM executor ORDER BY id_executor')
        self.rows = cursor.fetchall()
        columns = [desc[0] for desc in cursor.description]
        cursor.close()

        layout = QVBoxLayout()

        self.table = QTableWidgetItem(cursor.rowcount, len(columns))
        self.table.setHorizontalHeaderLabels(columns)
        self.table.itemChanged.connect(self.item_changed)
        layout.addWidget(self.table)

        layout_buttons = QHBoxLayout()
        add_line_button = QPushButton('Добавить строку')
        add_line_button.clicked.connect(self.add_line_btn_clicked)
        layout_buttons.addWidget(add_line_button)
        save_button = QPushButton('Сохранить')
        save_button.clicked.connect(self.save_btn_clicked)
        layout_buttons.addWidget(save_button)
        layout.addLayout(layout_buttons)

        self.setLayout(layout)

        for i in range(len(self.rows)):
            for j in range(len(columns)):
                item = QTableWidgetItem(str(self.rows[i][j]))
                if j == self.index_column:
                    item.setFlags(item.flags() & ~Qt.ItemIsEditable)
                self.table.setItem(i, j, item)
        self.table.resizeColumnsToContents()

    def add_line_btn_clicked(self):
        row_position = self.table.rowCount()
        self.table.insertRow(row_position)
        try:
            if row_position == 0:
                pred_index = 0
            else:
                pred_index = int(self.table.item(row_position-1, 0).text())
            item = QTableWidgetItem(str(pred_index + 1))
            item.setFlags(item.flags() & ~Qt.ItemIsEditable)
            self.table.setItem(row_position, 0, item)
        except:
            pass

    def item_changed(self):
        self.table.resizeColumnsToContents()

    def save_btn_clicked(self):
        try:

```

```

data = []
for r in range(self.table.rowCount()):
    row = []
    for c in range(self.table.columnCount()):
        row.append(self.table.item(r, c).text())
    data.append(row)
for r in range(len(self.rows)):
    the_same = True
    for c in range(len(self.rows[r])):
        if str(data[r][c]) != str(self.rows[r][c]):
            the_same = False
    if not the_same:
        cur = self.conn.cursor()
        cur.execute(f"""UPDATE executor SET
exec_name='{data[r][1]}',
                                post='{data[r][2]}' WHERE
id_executor={data[r][0]}""")
        self.conn.commit()
        cur.close()
    if len(self.rows) < len(data):
        for r in range(len(self.rows), len(data)):
            cur = self.conn.cursor()
            cur.execute(f"""INSERT INTO executor (exec_name, post)
                                VALUES ('{data[r][1]}', '{data[r][2]}')""")
            self.conn.commit()
            cur.close()
    QMessageBox.about(self, 'Изменения сохранены', 'Изменения успешно
сохранены')
except:
    QMessageBox.about(self, 'Ошибка', 'Неверное значение в таблице')

```

В папке TableEditWindows файл RepairRequestEditWindow.py

```

from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QHBoxLayout,
                             QPushButton, QTableWidgetItem,
                             QMessageBox)
from PyQt5.QtCore import Qt
import psycopg2

class RepairRequestEditDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.conn =
psycopg2.connect('postgresql://postgres:postgres@localhost:5432/course_work_d
b')

        self.index_column = 0

        self.setWindowFlags(self.windowFlags() &
(~Qt.WindowContextHelpButtonHint))
        self.setWindowTitle("Просмотр и изменение заявок на ремонт")
        self.resize(800, 600)

        cursor = self.conn.cursor()
        cursor.execute('SELECT * FROM repair_request ORDER BY id_request')
        self.rows = cursor.fetchall()
        columns = [desc[0] for desc in cursor.description]
        cursor.close()

        layout = QVBoxLayout()

        self.table = QTableWidgetItem(cursor.rowcount, len(columns))

```

```

self.table.setHorizontalHeaderLabels(columns)
self.table.itemChanged.connect(self.item_changed)
layout.addWidget(self.table)

layout_buttons = QHBoxLayout()
add_line_button = QPushButton('Добавить строку')
add_line_button.clicked.connect(self.add_line_btn_clicked)
layout_buttons.addWidget(add_line_button)
save_button = QPushButton('Сохранить')
save_button.clicked.connect(self.save_btn_clicked)
layout_buttons.addWidget(save_button)
layout.addLayout(layout_buttons)

self.setLayout(layout)

for i in range(len(self.rows)):
    for j in range(len(columns)):
        item = QTableWidgetItem(str(self.rows[i][j]))
        if j == self.index_column:
            item.setFlags(item.flags() & ~Qt.ItemIsEditable)
        self.table.setItem(i, j, item)
self.table.resizeColumnsToContents()

def add_line_btn_clicked(self):
    row_position = self.table.rowCount()
    self.table.insertRow(row_position)
    try:
        if row_position == 0:
            pred_index = 0
        else:
            pred_index = int(self.table.item(row_position-1, 0).text())
            item = QTableWidgetItem(str(pred_index + 1))
            item.setFlags(item.flags() & ~Qt.ItemIsEditable)
            self.table.setItem(row_position, 0, item)
    except:
        pass

def item_changed(self):
    self.table.resizeColumnsToContents()

def save_btn_clicked(self):
    try:
        data = []
        for r in range(self.table.rowCount()):
            row = []
            for c in range(self.table.columnCount()):
                row.append(self.table.item(r, c).text())
            data.append(row)
        for r in range(len(self.rows)):
            the_same = True
            for c in range(len(self.rows[r])):
                if str(data[r][c]) != str(self.rows[r][c]):
                    the_same = False
            if not the_same:
                cur = self.conn.cursor()
                cur.execute(f"UPDATE repair_request SET
status='{data[r][1]}',
                                request_time='{data[r][2]}',
id_technique={data[r][3]},
                                id_rep_type={data[r][4]},
id_executor={data[r][5]}

```

```

id_request={data[r][0]}"""
        self.conn.commit()
        cur.close()
        if len(self.rows) < len(data):
            for r in range(len(self.rows), len(data)):
                cur = self.conn.cursor()
                cur.execute(f"""INSERT INTO repair_request
                                (status, request_time, id_technique,
id_rep_type, id_executor, id_branch)
                                VALUES ('{data[r][1]}', '{data[r][2]}',
{data[r][3]},
                                {data[r][4]}, {data[r][5]}, {data[r][6]})""")
                self.conn.commit()
                cur.close()
                QMessageBox.about(self, 'Изменения сохранены', 'Изменения успешно
сохранены')
            except:
                QMessageBox.about(self, 'Ошибка', 'Неверное значение в таблице')

```

В папке TableEditWindows файл RepairTypeEditWindow.py

```

from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QHBoxLayout,
                             QPushButton, QTableWidgetItem,
QMessageBox)
from PyQt5.QtCore import Qt
import psycopg2

class RepairTypeEditDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.conn =
psycopg2.connect('postgresql://postgres:postgres@localhost:5432/course_work_d
b')

        self.index_column = 0

        self.setWindowFlags(self.windowFlags() &
(~Qt.WindowContextHelpButtonHint))
        self.setWindowTitle("Просмотр и изменение видов ремонта")
        self.resize(800, 600)

        cursor = self.conn.cursor()
        cursor.execute('SELECT * FROM repair_type ORDER BY id_rep_type')
        self.rows = cursor.fetchall()
        columns = [desc[0] for desc in cursor.description]
        cursor.close()

        layout = QVBoxLayout()

        self.table = QTableWidgetItem(cursor.rowcount, len(columns))
        self.table.setHorizontalHeaderLabels(columns)
        self.table.itemChanged.connect(self.item_changed)
        layout.addWidget(self.table)

        layout_buttons = QHBoxLayout()
        add_line_button = QPushButton('Добавить строку')
        add_line_button.clicked.connect(self.add_line_btn_clicked)
        layout_buttons.addWidget(add_line_button)
        save_button = QPushButton('Сохранить')
        save_button.clicked.connect(self.save_btn_clicked)
        layout_buttons.addWidget(save_button)

```

```

        layout.addLayout(layout_buttons)

    self.setLayout(layout)

    for i in range(len(self.rows)):
        for j in range(len(columns)):
            item = QTableWidgetItem(str(self.rows[i][j]))
            if j == self.index_column:
                item.setFlags(item.flags() & ~Qt.ItemIsEditable)
            self.table.setItem(i, j, item)
    self.table.resizeColumnsToContents()

def add_line_btn_clicked(self):
    row_position = self.table.rowCount()
    self.table.insertRow(row_position)
    try:
        if row_position == 0:
            pred_index = 0
        else:
            pred_index = int(self.table.item(row_position-1, 0).text())
            item = QTableWidgetItem(str(pred_index + 1))
            item.setFlags(item.flags() & ~Qt.ItemIsEditable)
            self.table.setItem(row_position, 0, item)
    except:
        pass

def item_changed(self):
    self.table.resizeColumnsToContents()

def save_btn_clicked(self):
    try:
        data = []
        for r in range(self.table.rowCount()):
            row = []
            for c in range(self.table.columnCount()):
                row.append(self.table.item(r, c).text())
            data.append(row)
        for r in range(len(self.rows)):
            the_same = True
            for c in range(len(self.rows[r])):
                if str(data[r][c]) != str(self.rows[r][c]):
                    the_same = False
            if not the_same:
                cur = self.conn.cursor()
                cur.execute(f"""UPDATE repair_type SET
rep_type_name='{data[r][1]}',
                                standart_time='{data[r][2]}' WHERE
id_rep_type={data[r][0]}""")
                self.conn.commit()
                cur.close()
            if len(self.rows) < len(data):
                for r in range(len(self.rows), len(data)):
                    cur = self.conn.cursor()
                    cur.execute(f"""INSERT INTO repair_type (rep_type_name,
standart_time)
                                VALUES ('{data[r][1]}', '{data[r][2]}')""")
                    self.conn.commit()
                    cur.close()
    QMessageBox.about(self, 'Изменения сохранены', 'Изменения успешно
сохранены')

```



```
except:
    QMessageBox.about(self, 'Ошибка', 'Неверное значение в таблице')
```

В папке TableEditWindows файл TechniqueEditWindow.py

```
from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QHBoxLayout,
                             QPushButton, QTableWidgetItem,
                             QMessageBox)
from PyQt5.QtCore import Qt
import psycopg2

class TechniqueEditDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.conn =
psycopg2.connect('postgresql://postgres:postgres@localhost:5432/course_work_d
b')

        self.index_column = 0

        self.setWindowFlags(self.windowFlags() &
(~Qt.WindowContextHelpButtonHint))
        self.setWindowTitle("Просмотр и изменение техники")
        self.resize(800, 600)

        cursor = self.conn.cursor()
        cursor.execute('SELECT * FROM technique ORDER BY id_technique')
        self.rows = cursor.fetchall()
        columns = [desc[0] for desc in cursor.description]
        cursor.close()

        layout = QVBoxLayout()

        self.table = QTableWidgetItem(cursor.rowcount, len(columns))
        self.table.setHorizontalHeaderLabels(columns)
        self.table.itemChanged.connect(self.item_changed)
        layout.addWidget(self.table)

        layout_buttons = QHBoxLayout()
        add_line_button = QPushButton('Добавить строку')
        add_line_button.clicked.connect(self.add_line_btn_clicked)
        layout_buttons.addWidget(add_line_button)
        save_button = QPushButton('Сохранить')
        save_button.clicked.connect(self.save_btn_clicked)
        layout_buttons.addWidget(save_button)
        layout.addLayout(layout_buttons)

        self.setLayout(layout)

        for i in range(len(self.rows)):
            for j in range(len(columns)):
                item = QTableWidgetItem(str(self.rows[i][j]))
                if j == self.index_column:
                    item.setFlags(item.flags() & ~Qt.ItemIsEditable)
                self.table.setItem(i, j, item)
        self.table.resizeColumnsToContents()

    def add_line_btn_clicked(self):
        row_position = self.table.rowCount()
        self.table.insertRow(row_position)
        try:
            if row_position == 0:
```

```

        pred_index = 0
    else:
        pred_index = int(self.table.item(row_position-1, 0).text())
        item = QTableWidgetItem(str(pred_index + 1))
        item.setFlags(item.flags() & ~Qt.ItemIsEditable)
        self.table.setItem(row_position, 0, item)
    except:
        pass

def item_changed(self):
    self.table.resizeColumnsToContents()

def save_btn_clicked(self):
    try:
        data = []
        for r in range(self.table.rowCount()):
            row = []
            for c in range(self.table.columnCount()):
                row.append(self.table.item(r, c).text())
            data.append(row)
        for r in range(len(self.rows)):
            the_same = True
            for c in range(len(self.rows[r])):
                if str(data[r][c]) != str(self.rows[r][c]):
                    the_same = False
            if not the_same:
                cur = self.conn.cursor()
                cur.execute(f"""UPDATE technique SET
tec_name='{data[r][1]}',
                                id_department={data[r][2]},
id_tec_type={data[r][2]}
                                WHERE id_rep_type={data[r][0]}""")
                self.conn.commit()
                cur.close()
            if len(self.rows) < len(data):
                for r in range(len(self.rows), len(data)):
                    cur = self.conn.cursor()
                    cur.execute(f"""INSERT INTO technique (tec_name,
id_department, id_tec_type)
                                VALUES ('{data[r][1]}', {data[r][2]},
{data[r][3]})""")
                    self.conn.commit()
                    cur.close()
        QMessageBox.about(self, 'Изменения сохранены', 'Изменения успешно
сохранены')
    except:
        QMessageBox.about(self, 'Ошибка', 'Неверное значение в таблице')

```

В папке TableEditWindows файл TechniqueTypeEditWindow.py

```

from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QHBoxLayout,
                             QPushButton, QTableWidgetItem,
                             QMessageBox)
from PyQt5.QtCore import Qt
import psycopg2

class TechniqueTypeEditDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.conn =
        psycopg2.connect('postgres://postgres:postgres@localhost:5432/course_work_d

```

```

b')

self.index_column = 0

self.setWindowFlags(self.windowFlags() &
(~Qt.WindowContextHelpButtonHint))
self.setWindowTitle("Просмотр и изменение видов техники")
self.resize(800, 600)

cursor = self.conn.cursor()
cursor.execute('SELECT * FROM technique_type ORDER BY id_tec_type')
self.rows = cursor.fetchall()
columns = [desc[0] for desc in cursor.description]
cursor.close()

layout = QVBoxLayout()

self.table = QTableWidgetItem(cursor.rowcount, len(columns))
self.table.setHorizontalHeaderLabels(columns)
self.table.itemChanged.connect(self.item_changed)
layout.addWidget(self.table)

layout_buttons = QHBoxLayout()
add_line_button = QPushButton('Добавить строку')
add_line_button.clicked.connect(self.add_line_btn_clicked)
layout_buttons.addWidget(add_line_button)
save_button = QPushButton('Сохранить')
save_button.clicked.connect(self.save_btn_clicked)
layout_buttons.addWidget(save_button)
layout.addLayout(layout_buttons)

self.setLayout(layout)

for i in range(len(self.rows)):
    for j in range(len(columns)):
        item = QTableWidgetItem(str(self.rows[i][j]))
        if j == self.index_column:
            item.setFlags(item.flags() & ~Qt.ItemIsEditable)
        self.table.setItem(i, j, item)
self.table.resizeColumnsToContents()

def add_line_btn_clicked(self):
    row_position = self.table.rowCount()
    self.table.insertRow(row_position)
    try:
        if row_position == 0:
            pred_index = 0
        else:
            pred_index = int(self.table.item(row_position-1, 0).text())
            item = QTableWidgetItem(str(pred_index + 1))
            item.setFlags(item.flags() & ~Qt.ItemIsEditable)
            self.table.setItem(row_position, 0, item)
    except:
        pass

def item_changed(self):
    self.table.resizeColumnsToContents()

def save_btn_clicked(self):
    try:
        data = []
        for r in range(self.table.rowCount()):

```

```

        row = []
        for c in range(self.table.columnCount()):
            row.append(self.table.item(r, c).text())
        data.append(row)
    for r in range(len(self.rows)):
        the_same = True
        for c in range(len(self.rows[r])):
            if str(data[r][c]) != str(self.rows[r][c]):
                the_same = False
        if not the_same:
            cur = self.conn.cursor()
            cur.execute(f"UPDATE technique_type SET
tec_type_name='{data[r][1]}'
                                WHERE id_tec_type={data[r][0]}")
            self.conn.commit()
            cur.close()
        if len(self.rows) < len(data):
            for r in range(len(self.rows), len(data)):
                cur = self.conn.cursor()
                cur.execute(f"INSERT INTO technique_type
(tec_type_name)
                                VALUES ('{data[r][1]}')")
                self.conn.commit()
                cur.close()
    QMessageBox.about(self, 'Изменения сохранены', 'Изменения успешно
сохранены')
except:
    QMessageBox.about(self, 'Ошибка', 'Неверное значение в таблице')

```

В папке ReportWindows файл RequestsByExecutorWindow.py

```

from PyQt5.QtWidgets import (QDialog, QVBoxLayout, QHBoxLayout, QLabel,
QLineEdit,
                                QPushButton, QTableWidgetItem)

from PyQt5.QtCore import Qt
import psycopg2

class RequestsByExecutorDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.conn =
psycopg2.connect('postgres://postgres:postgres@localhost:5432/course_work_d
b')

        self.setWindowFlags(self.windowFlags() &
(~Qt.WindowContextHelpButtonHint))
        self.setWindowTitle("Контроль выполнения заявок по исполнителям")
        self.resize(1300, 600)

        layout = QVBoxLayout()

        layout_enter_executor = QHBoxLayout()
        label_executor = QLabel('ФИО исполнителя: ')
        layout_enter_executor.addWidget(label_executor)
        self.executor_edit = QLineEdit()
        layout_enter_executor.addWidget(self.executor_edit)
        btn_enter_executor = QPushButton('Ввод')
        btn_enter_executor.clicked.connect(self.btn_enter_executor_action)
        layout_enter_executor.addWidget(btn_enter_executor)
        layout.addLayout(layout_enter_executor)

```

```

        self.table = QTableWidgetItem()
        self.table.setEditTriggers(QTableWidgetItem.NoEditTriggers)
        layout.addWidget(self.table)

    self.setLayout(layout)

    def btn_enter_executor_action(self):
        cursor = self.conn.cursor()
        cursor.execute(f"""SELECT * FROM
get_requests_by_executor('{self.executor_edit.text()}')""")
        rows = cursor.fetchall()
        columns = [desc[0] for desc in cursor.description]
        self.table.setRowCount(cursor.rowcount)
        self.table.setColumnCount(len(columns))
        self.table.setHorizontalHeaderLabels(columns)
        for i in range(len(rows)):
            for j in range(len(columns)):
                item = QTableWidgetItem(str(rows[i][j]))
                self.table.setItem(i, j, item)
        self.table.resizeColumnsToContents()
        cursor.close()

```

В папке ReportWindows файл RequestsByTypeWindow.py

```

from PyQt5.QtWidgets import (QDialog, QVBoxLayout,
                             QTableWidgetItem, QTableWidgetItem)
from PyQt5.QtCore import Qt
import psycopg2

class RequestsByTypeDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.conn =
        psycopg2.connect('postgres://postgres:postgres@localhost:5432/course_work_d
b')

        self.setWindowFlags(self.windowFlags() &
(~Qt.WindowContextHelpButtonHint))
        self.setWindowTitle("Заявки заданного типа")
        self.resize(800, 600)

        cursor = self.conn.cursor()
        cursor.execute('SELECT * FROM get_requests_by_type()')
        rows = cursor.fetchall()
        columns = [desc[0] for desc in cursor.description]
        cursor.close()

        layout = QVBoxLayout()

        self.table = QTableWidgetItem(cursor.rowcount, len(columns))
        self.table.setHorizontalHeaderLabels(columns)
        layout.addWidget(self.table)

        self.setLayout(layout)

        for i in range(len(rows)):
            for j in range(len(columns)):
                item = QTableWidgetItem(str(rows[i][j]))
                self.table.setItem(i, j, item)
        self.table.resizeColumnsToContents()
        self.table.setEditTriggers(QTableWidgetItem.NoEditTriggers)

```

В папке ReportWindows файл RequestsDeadlineWindow.py

```
from PyQt5.QtWidgets import (QDialog, QVBoxLayout,
                              QTableWidgetItem, QTableWidgetItem)
from PyQt5.QtCore import Qt
import psycopg2

class RequestsDeadlineDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.conn =
psycopg2.connect('postgres://postgres:postgres@localhost:5432/course_work_d
b')

        self.setWindowFlags(self.windowFlags() &
(~Qt.WindowContextHelpButtonHint))
        self.setWindowTitle("Заявки с превышением нормативного срока")
        self.resize(1300, 600)

        cursor = self.conn.cursor()
        cursor.execute('SELECT * FROM get_requests_deadline()')
        rows = cursor.fetchall()
        columns = [desc[0] for desc in cursor.description]
        cursor.close()

        layout = QVBoxLayout()

        self.table = QTableWidgetItem(cursor.rowcount, len(columns))
        self.table.setHorizontalHeaderLabels(columns)
        layout.addWidget(self.table)

        self.setLayout(layout)

        for i in range(len(rows)):
            for j in range(len(columns)):
                item = QTableWidgetItem(str(rows[i][j]))
                self.table.setItem(i, j, item)
        self.table.resizeColumnsToContents()
        self.table.setEditTriggers(QTableWidgetItem.NoEditTriggers)
```

В папке ReportWindows файл RequestsWindow.py

```
from PyQt5.QtWidgets import (QDialog, QVBoxLayout,
                              QTableWidgetItem, QTableWidgetItem)
from PyQt5.QtCore import Qt
import psycopg2

class RequestsDialog(QDialog):
    def __init__(self):
        super().__init__()
        self.conn =
psycopg2.connect('postgres://postgres:postgres@localhost:5432/course_work_d
b')

        self.setWindowFlags(self.windowFlags() &
(~Qt.WindowContextHelpButtonHint))
        self.setWindowTitle("Заявки на ремонтные работы")
        self.resize(1600, 600)

        cursor = self.conn.cursor()
```

```
cursor.execute('SELECT * FROM get_requests()')
rows = cursor.fetchall()
columns = [desc[0] for desc in cursor.description]
cursor.close()

layout = QVBoxLayout()

self.table = QTableWidgetItem(cursor.rowcount, len(columns))
self.table.setHorizontalHeaderLabels(columns)
layout.addWidget(self.table)

self.setLayout(layout)

for i in range(len(rows)):
    for j in range(len(columns)):
        item = QTableWidgetItem(str(rows[i][j]))
        self.table.setItem(i, j, item)
self.table.resizeColumnsToContents()
self.table.setEditTriggers(QTableWidgetItem.NoEditTriggers)
```