МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Институт информационных технологий

Кафедра/департамент «Информационные системы»

Пояснительная записка

к расчетно-графической работе

по дисциплине «Управление данными»

на тему «База данных ателье»

Выполнил: обучающийся 3 курса

группы: ИС/б-20-2-о

Направления подготовки

09.03.02

Информационные системы и технологии

Филозоп А.Н.

« __» _____20 22 г.

Научный руководитель:

«__» _____20 22 г.

Оценка_____

« __» _____ 20 22 г.

Севастополь 2022

СОДЕРЖАНИЕ

Содержание.....2
1 Аналитическая часть.....4
1.1. Анализ предметной области.....4
1.2. Постановка задачи.....4
2 Разработка логической модели базы данных.....6
2.1. Построение простой и сложной сетевых моделей.....6
2.2. Построение диаграммы «сущность-связь» в нотации П.Чена.....7
2.3. Построение модели основанной на ключах.....8
2.4. Построение полной атрибутивной модели в нотации IDEF1X.....9
3 Разработка физической модели базы данных.....11
3.1. Выбор аппаратной и программной платформы для реализации БД.....11
3.2. Реализация базы данных.....11
3.3. Тестирование базы данных.....13
4 Разработка клиентского приложения.....21
4.1. Обоснование выбора языка программирования.....21
4.2. Разработка интерфейса.....21
4.3. Алгоритм работы модулей.....21
4.4. Тестирование работы приложения.....21

ВВЕДЕНИЕ

В рамках настоящей работы ведется разработка веб-приложения для работы с базой данных на тему «База данных ателье» на основании документа – техническое задание и в рамках организации – Севастопольский государственный университет. Дата выдачи задания: 17.09.2022.

Целью расчетно-графической работы является систематизация, закрепление и углубление знаний в области управления данными на SQL-подобных языках и их совершенствование путем применения при разработке комплексного веб-приложения.

Для достижения цели на разных этапах курсового проектирования должны быть решены следующие задачи:

- анализ предметной области;
- разработка логической модели базы данных;
- построение физической модели базы данных;
- разработка клиентского приложения.

1 АНАЛИТИЧЕСКАЯ ЧАСТЬ

1.1. Анализ предметной области

Заказчиком базы данных и прикладного приложения — бот для Telegram – является некоторое ателье. Ателье занимается пошивом и ремонтом одежды. Были выделены следующие основные объекты: заказ, клиент и работник.

В ходе исследования были установлены следующие связи между объектами. Клиент оформлять несколько заказов, при этом заказ может быть привязан только к одному клиенту. Работник может одновременно работать над несколькими заказами, но над заказом может работать только один работник.

Были выделены основные свойства каждого из объектов. Клиента характеризуют такие свойства, как: имя, фамилия, номер телефона, имя пользователя в Telegram, абстрактный идентификатор. Работника характеризуют следующие свойства: имя, фамилия, номер телефона, имя пользователя в Telegram, абстрактный идентификатор и принадлежность к одной из групп пользователей БД. Заказ характеризуют: номер, клиент, исполнитель (работник), описание, цена, состояние, время начала, время последнего изменения состояния, время завершения и список дополнительных требований.

1.2. Постановка задачи

В ходе анализа предметной области с помощью документации, расположенной в открытом доступе, реальных прототипов приложений и информации специалистов были выявлены основные связи и объекты рассматриваемой базы данных.

Основными объектами базы данных являются: заказ; клиент; работник.

Пользователями базы данных являются клиент, работник и администратор.

Клиенты имеют возможность просматривать все или активные на данный момент заказы, а также формировать требования, например, об ускорении выполнения заказа за дополнительную плату.

Работники имеют возможность добавлять заказ и изменять его состояние, а также отвечать на поступившие запросы от клиентов либо отказом, либо подтверждением с установлением размера дополнительной платы.

Администратор, помимо возможностей работника, имеет возможность добавлять пользователей БД.

По результатам анализа предметной области были выделены основные сущности и связи между ними, что будет необходимо на логоческом этапе разработки базы данных.

2 РАЗРАБОТКА ЛОГИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ

2.1. Построение простой и сложной сетевых моделей

Для отображения связей между объектами базы данных была построена диаграмма сложной сетевой модели (рисунок 2.1). Так как между объектами отсутствуют сложные связи, то диаграмма простой сетевой модели (рисунок 2.2) идентична диаграмме сложной сетевой модели.

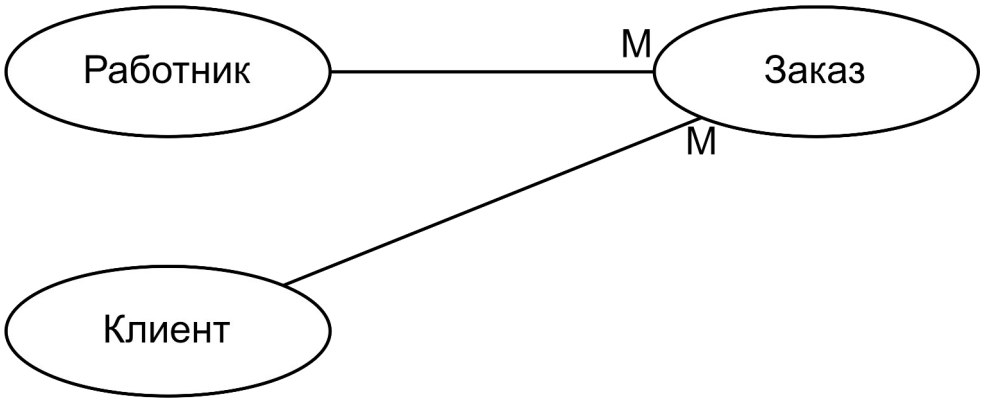

Рисунок 2.1 — Диаграмма сложной сетевой модели

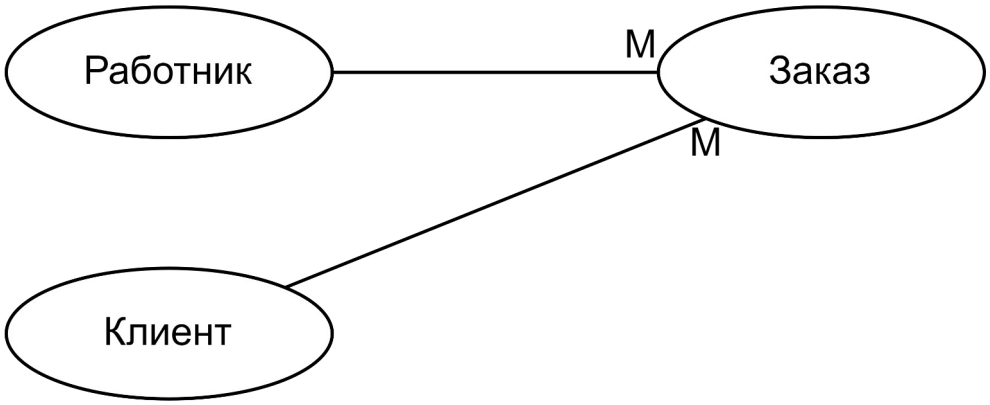

Рисунок 2.2 — Диаграмма простой сетевой модели

Построенные на данном этапе диаграммы будут участвовать при создании диаграммы «сущность-связь».

2.2. Построение диаграммы «сущность-связь» в нотации П.Чена

Выделенные сущности были повторно проанализированы с целью выделения первичных ключей. Таким образом в «Заказе» в качестве первичного ключа был выделен атрибут «Номер», в «Клиент» — атрибут «Ид», в «Работник» — атрибут «Ид». После чего была построена диаграмма «сущность-связь» в нотации П.Чена (рисунок 2.3).

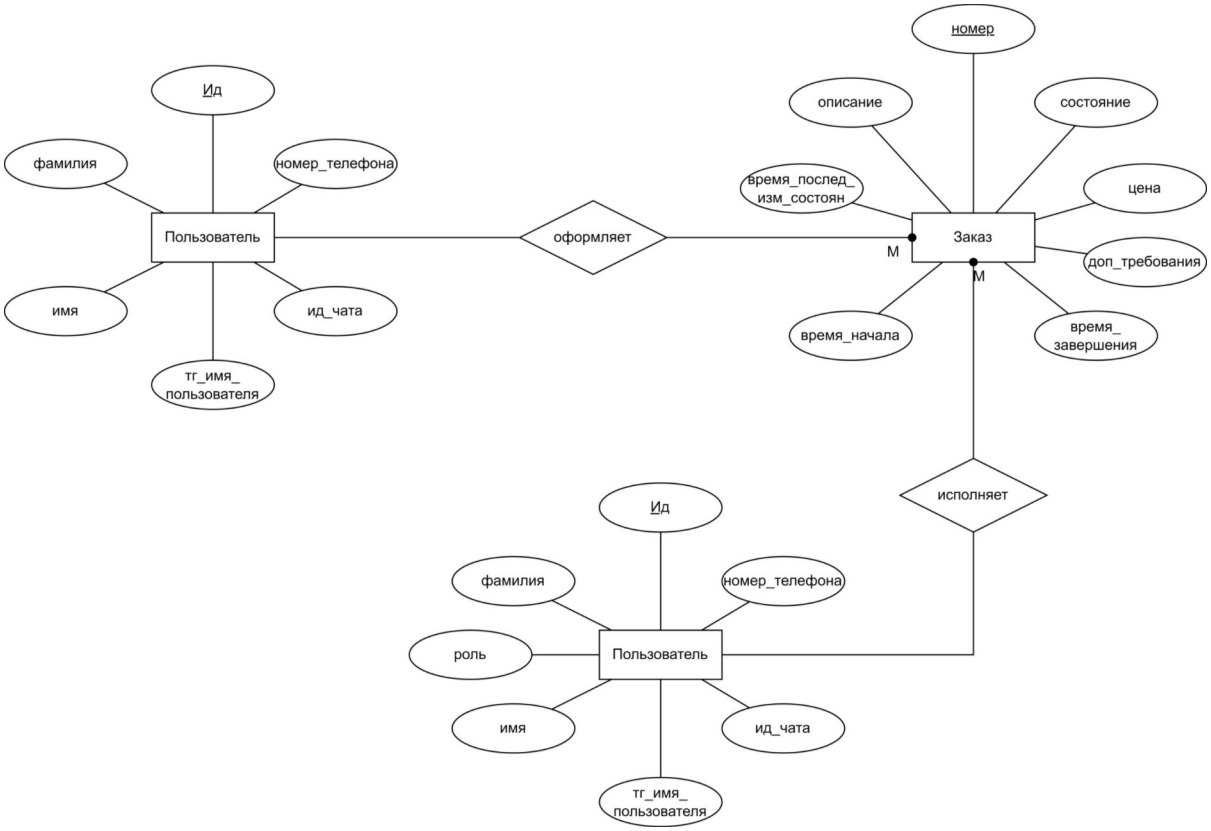

Рисунок 2.3 — Диаграмма «сущность-связь»

Построенные на данном этапе диаграмма будут участвовать при создании модели, основанной на ключах.

2.3. Построение модели основанной на ключах

На основе диаграммы «сущность-связь» была построена модель, основанная на ключах (рисунок 2.4).

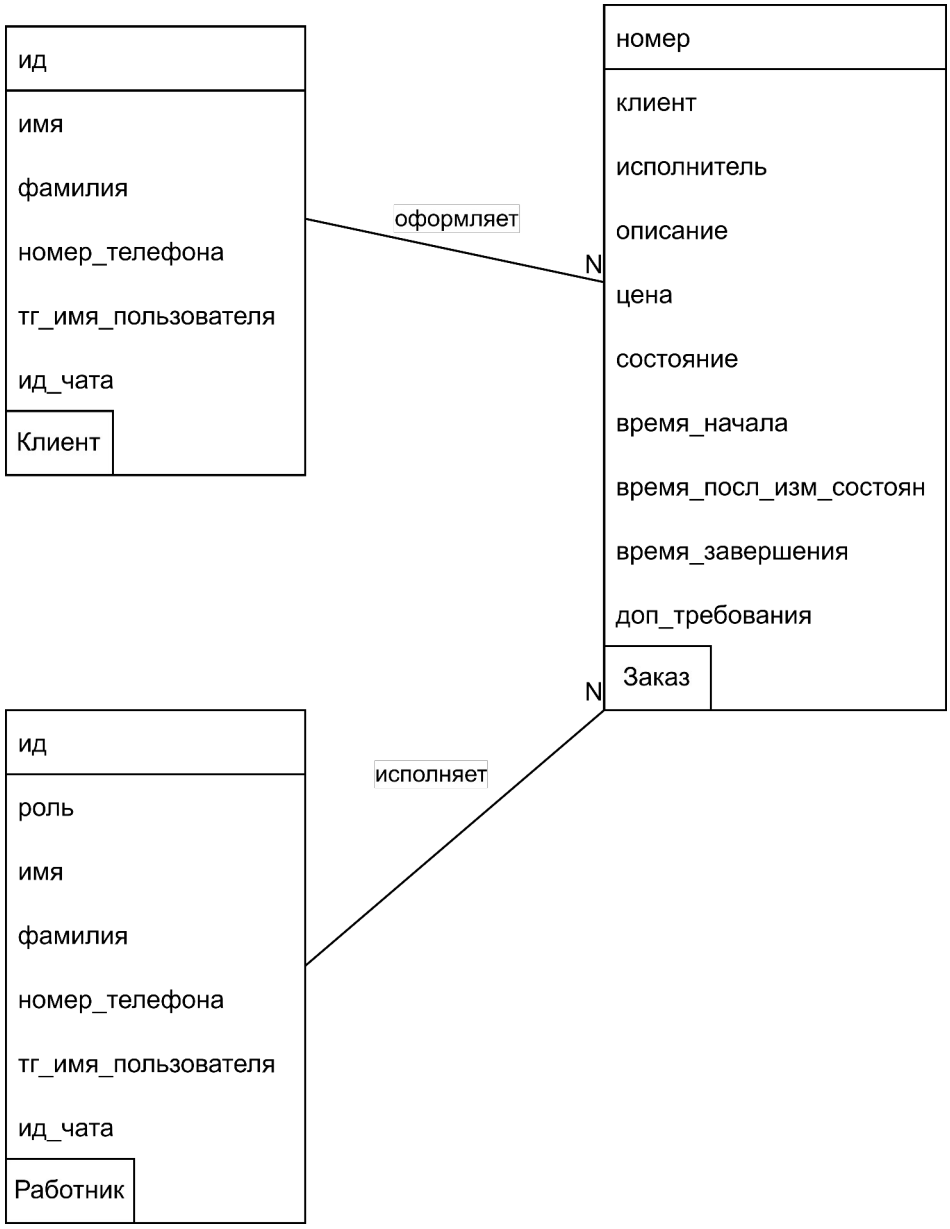

Рисунок 2.4 — Модель, основанная на ключах

Данная модель будет в дальнейшем нормализирована до 3НФ.

2.4. Построение полной атрибутивной модели в нотации IDEF1X

Полученная в разделе 2.3модель была подвергнута нормализации до 3НФ. Ниже описано приведение исходной модели базы данных к каждой из нормальных форм.

Первая нормальная форма требует атомарности каждого из атрибутов отношения. Так как заказ может иметь множество дополнительных требований, то отношение «Заказ» не находится в 1НФ. Для приведения к 1НФ требуется разбить это отношение на два: Заказ (Номер, Клиент, Исполнитель, Описание, Цена, Состояние, Время_начала, Время_последнего_изменения_состояния, Время_завершения); Доп_требования(Заказ, Время, Тип, Описание, Состояние, Изменение_цены). Теперь отношения находятся в 1НФ.

Вторая нормальная форма требует нахождения отношений в 1НФ и полной зависимости каждого неключевого атрибута от каждого ключа. В каждом отношении каждый атрибут зависит от каждого ключа, значит отношения находятся в 2НФ.

Третья нормальная форма требует нахождения отношения в 2НФ и нетранзитивной зависимости каждого неключевого атрибута от первичного ключа. В каждом отношении отсутствуют транзитивные зависимости, значит они находятся в 3НФ.

После проведения нормализации была построена полная атрибутивная модель в нотации IDEF1X (рисунок 2.5).

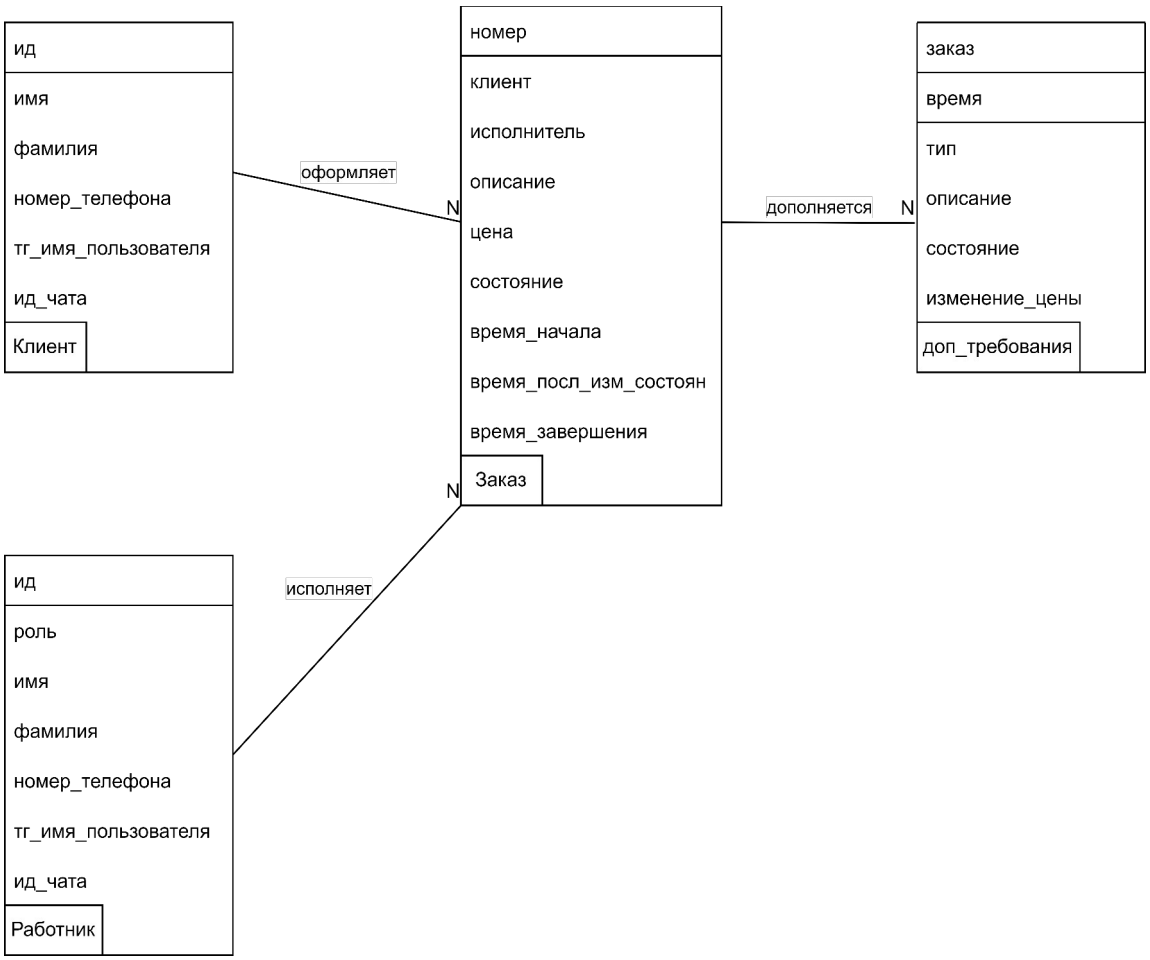

Рисунок 2.5 — Полная атрибутивная модель

По составленной полной атрибутивной модели будет построена физическая база данных.

3 РАЗРАБОТКА ФИЗИЧЕСКОЙ МОДЕЛИ БАЗЫ ДАННЫХ

3.1. Выбор аппаратной и программной платформы для реализации БД

Для реализации физической модели базы данных была выбрана платформа SQLite. Причина данного выбора: легковесность базы данных, высокая скорость работы на небольшом наборе данных.

3.2. Реализация базы данных

Таблицы 3.1 и 3.2 содержат информацию о реализации выделенных сущностей в базе данных.

Отдельного комментария заслуживают атрибуты, связанные с датой. Так как SQLite не предоставляет отдельного типа данных, позволяющего хранить дату, то было принято решение хранить дату в виде строки TEXT в соответствии со стандартом ISO 8601-2:2019.

Таблица 3.1 — Реализация сущностей в базе данных SQLite (часть 1)

Сущность	Название	Название атрибута в БД SQLite	Тип данных
Клиент	Ид	id	INTEGER
	Имя	first_name	TEXT
	Фамилия	last_name	TEXT
	Номер телефона	tel_number	TEXT
	Имя пользователя Telegram	tg_username	TEXT
	Ид чата Telegram	tg_chat_id	INTEGER

Таблица 3.2 — Реализация сущностей в базе данных SQLite (часть 2)

Сущность	Название	Название атрибута в БД SQLite	Тип данных
Заказ	Ид	id	INTEGER
	Ид клиента	customer_id	INTEGER
	Ид исполнителя	worker_id	INTEGER
	Описание	description	TEXT
	Цена	price	INTEGER
	Состояние	state	TEXT
	Время начала выполнения	begin_time	TEXT
	Время последнего изменения состояния	last_state_update_time	TEXT
	Время завершения	end_time	TEXT
Доп. требования	Ид заказа	order_id	INTEGER
	Время добавления	instant	TEXT
	Тип	type	TEXT
	Описание	description	TEXT
	Состояние	state	TEXT
	Изменение цены	price_delta	INTEGER

Таблица 3.3 — Реализация сущностей в базе данных SQLite (часть 3)

Сущность	Название	Название атрибута в БД SQLite	Тип данных
Работник	Ид	id	INTEGER
	Роль	role	TEXT
	Имя	first_name	TEXT
	Фамилия	last_name	TEXT
	Номер телефона	tel_number	TEXT
	Имя пользователя Telegram	tg_username	TEXT
	Ид чата Telegram	tg_chat_id	INTEGER

Запросы SQL для создания всех необходимых для работы базы данных таблиц описаны в приложении А.

3.3. Тестирование базы данных

Было проведено тестирование базы данных. До проведения испытаний база данных была заполнена некоторыми данными. Также была создано 15 тестовых запросов, представленных ниже.

Запрос 1: вывести всех клиентов. Код запроса представлен в листинге 3.1. Результат выполнения — на рисунке 3.1.

Листинг 3.1 — Текст запроса

```
SELECT * FROM "customer";
```

```
1   SELECT * FROM "customer";
```

	id	first_name	last_name	tel_number	tg_username	tg_chat_id
1	1	Alex	Filozop	+79788033293	cory7666	0
2	2	Cory	Lasky	none	corylasky	-1
3	3	Иван	Иванов	+79788555555	ivan_username	0
4	4	Александр	Прохорчук	+11253695689	alex89	0
5	5	Денис	Мимин	+55861235822	den89	0
6	6	Алиса	Джугашвили	+78035968596	alice_uHome	0

Рисунок 3.1 — Результат выполнения запроса №1

Запрос 2: вывести всех администраторов базы данных. Код запроса представлен в листинге 3.2. Результат выполнения — на рисунке 3.2.

Листинг 3.2 — Текст запроса

```
SELECT * FROM "worker" WHERE "role" = 'ADMIN';
```

```
1  SELECT * FROM "worker" WHERE "role" = 'ADMIN';
```

	id	role	first_name	last_name	tel_number	tg_username	tg_chat_id
1	1	ADMIN	Cory	Lasky	+11111155446	cory7666	0
2	2	ADMIN	Александр	Нонейм	79785555555	JohnyHarr	-1

Рисунок 3.2 — Результат выполнения запроса №2

Запрос 3: вывести все активные заказы (заказы, находящиеся на стадиях «В процессе» и «Готов»). Код запроса представлен в листинге 3.3. Результат выполнения — на рисунке 3.3.

Листинг 3.3 — Текст запроса

```
SELECT * FROM "order" WHERE "state" = 'IN_PROGRESS' OR "state" = 'READY';
```

```
1  SELECT * FROM "order" WHERE "state" = 'IN_PROGRESS' OR "state" = 'READY';
```

	id	customer_id	worker_id	description	price	state	begin_time	last_state_update_time	end_time
1	1	1	1	Ремонт футболки	200	IN_PROGRESS	2022-11-28T21:52:28Z	2022-12-21T07:49:55Z	2022-11-29T21:52:28Z
2	2	1	1	Подшить штаны.	10000	READY	2022-11-28T22:08:56Z	2022-12-21T12:39:12Z	2022-11-30T21:52:28Z
3	3	1	1	Красная шляпа, похожая на шляпу ...	80050	IN_PROGRESS	2022-12-21T08:25:11Z	2022-12-21T09:21:18Z	2022-12-21T12:00:00Z
4	4	2	2	Шуба из норки.	8000054	IN_PROGRESS	2022-12-21T09:20:33Z	2022-12-21T09:20:33Z	2022-12-25T15:00:00Z
5	5	1	1	Подшить штаны.	30032	READY	2022-12-21T12:32:43Z	2022-12-21T22:45:44Z	2022-12-21T16:00:00Z
6	6	3	3	Пошив штор.	30000	IN_PROGRESS	2022-12-21T23:14:33Z	2022-12-21T23:14:33Z	2022-12-25T02:13:00Z
7	7	1	1	Пошив штор.	50090	IN_PROGRESS	2022-12-21T23:17:01Z	2022-12-21T23:17:01Z	2022-12-22T05:00:00Z

Рисунок 3.3 — Результат выполнения запроса №3

Запрос 4: вывести все активные заказы вместе с информацией о клиентах. Код запроса представлен в листинге 3.4. Результат выполнения — на рисунке 3.4.

Листинг 3.4 — Текст запроса

```
SELECT * FROM "order" INNER JOIN "customer" ON "customer_id" = "customer"."id";
```

```
SELECT * FROM "order" INNER JOIN "customer" ON "customer_id" = "customer"."id";
```

	id	customer_id	worker_id	description	price	state	begin_time	last_state_update_time	end_time	id	first_name	last_name	tel_number	tg_usern
1	1	1	1	Ремонт футболки	200	IN_PROGRESS	2022-11-28T21:52:28Z	2022-12-21T07:49:55Z	2022-11-29T21:52:28Z	1	Alex	Filozop	+79788033293	cory7666
2	2	1	1	Подшить штаны.	10000	READY	2022-11-28T22:08:56Z	2022-12-21T12:39:12Z	2022-11-30T21:52:28Z	1	Alex	Filozop	+79788033293	cory7666
3	3	1	1	Красная шляпа, похожая на шляпу ...	80050	IN_PROGRESS	2022-12-21T08:25:11Z	2022-12-21T09:21:18Z	2022-12-21T12:00:00Z	1	Alex	Filozop	+79788033293	cory7666
4	4	2	2	Шуба из норки.	8000054	IN_PROGRESS	2022-12-21T09:20:33Z	2022-12-21T09:20:33Z	2022-12-25T15:00:00Z	2	Cory	Lasky	none	corylasky
5	5	1	1	Подшить штаны.	30032	READY	2022-12-21T12:32:43Z	2022-12-21T22:45:44Z	2022-12-21T16:00:00Z	1	Alex	Filozop	+79788033293	cory7666
6	6	3	3	Пошив штор.	30000	IN_PROGRESS	2022-12-21T23:14:33Z	2022-12-21T23:14:33Z	2022-12-25T02:13:00Z	3	Иван	Иванов	+79788555555	ivan_usern
7	7	1	1	Пошив штор.	50090	IN_PROGRESS	2022-12-21T23:17:01Z	2022-12-21T23:17:01Z	2022-12-22T05:00:00Z	1	Alex	Filozop	+79788033293	cory7666

Рисунок 3.4 — Результат выполнения запроса №4

Запрос 5: вывести все заказы, к которым были предъявлены дополнительные требования, и собственно дополнительные требования. Код запроса представлен в листинге 3.5. Результат выполнения — на рисунке 3.5.

Листинг 3.5 — Текст запроса

```
SELECT "id", "order"."description" as "order_description",
"additional_requirement"."description" as "ar_description"
       FROM "order" INNER JOIN "additional_requirement" ON "order"."id" =
"additional_requirement"."order_id";
```

```
SELECT "id", "order"."description" as "order_description", "additional_requirement"."description" as "ar_description"
    FROM "order" INNER JOIN "additional_requirement" ON "order"."id" = "additional_requirement"."order_id";
```

	id	order_description	ar_description
1	1	Ремонт футболки	Awersome description.
2	5	Подшить штаны.	Хочу уменьшить величину в два раза.

Рисунок 3.5 — Результат выполнения запроса №5

Запрос 6: вывести клиентов, номер телефона которых начинается с «+7978». Код запроса представлен в листинге 3.6. Результат выполнения — на рисунке 3.6.

Листинг 3.6 — Текст запроса

```
SELECT * FROM "customer" WHERE "tel_number" LIKE '+7978%';
```

```
1   SELECT * FROM "customer" WHERE "tel_number" LIKE '+7978%';
```

	id	first_name	last_name	tel_number	tg_username	tg_chat_id
1	1	Alex	Filozop	+79788033293	cory7666	0
2	3	Иван	Иванов	+79788555555	ivan_username	0
3	7	Дмитрий	Рогозин	+79786675643	dmitry_rogozin	-1

Рисунок 3.6 — Результат выполнения запроса №6

Запрос 7: вывести среднюю цену выполнения всех заказов. Код запроса представлен в листинге 3.7. Результат выполнения — на рисунке 3.7.

Листинг 3.7 — Текст запроса

```
SELECT SUM("price") FROM "order";
```

```
1   SELECT SUM("price") FROM "order";
```

	SUM("price")
1	8200426

Рисунок 3.7 — Результат выполнения запроса №7

Запрос 8: вывести максимально дорогой заказ. Код запроса представлен в листинге 3.8. Результат выполнения — на рисунке 3.8.

Листинг 3.8 — Текст запроса

```
SELECT MAX("price") FROM "order";
```

```
1   SELECT MAX("price") FROM "order";
```

	MAX("price")
1	8000054

Рисунок 3.8 — Результат выполнения запроса №8

Запрос 9: вывести все отклонённые дополнительные требования. Код запроса представлен в листинге 3.9. Результат выполнения — на рисунке 3.9.

Листинг 3.9 — Текст запроса

```
SELECT * FROM "additional_requirement" WHERE "state" = 'CANCELLED';
```

```
1   SELECT * FROM "additional_requirement" WHERE "state" = 'CANCELLED';
```

	order_id	instant	type	description	state	price_delta
1	5	2022-12-22T02:17:18Z	NOTE	Хочу уменьшить величину в два раза.	CANCELLED	0

Рисунок 3.9 — Результат выполнения запроса №9

Запрос 10: вставить в таблицу «Клиент» некоторого пользователя. Код запроса представлен в листинге 3.10. Результат выполнения — на рисунке 3.10.

Листинг 3.10 — Текст запроса

```
INSERT INTO "customer" ("first_name", "last_name", "tel_number", "tg_username",
"tg_chat_id")
  VALUES ('Александр', 'Альбред', '+79882323354', 'aalexander', 0);
```

```
1  INSERT INTO "customer" ("first_name", "last_name", "tel_number", "tg_username", "tg_chat_id")
2    VALUES ('Александр', 'Альбред', '+79882323354', 'aalexander', 0);
3  SELECT * FROM "customer" WHERE "tg_username" = 'aalexander';
```

	id	first_name	last_name	tel_number	tg_username	tg_chat_id
1	8	Александр	Альбред	+79882323354	aalexander	0

Рисунок 3.10 — Результат выполнения запроса №10

Запрос 11: вывести всех пользователей БД, которые являются одновременно и клиентами, и работниками. Код запроса представлен в листинге 3.11. Результат выполнения — на рисунке 3.11.

Листинг 3.11 — Текст запроса

```
SELECT "customer"."first_name", "customer"."last_name" FROM "customer"
INNER JOIN "worker" ON "customer"."tg_username" = "worker"."tg_username";
```

```
1  SELECT "customer"."first_name", "customer"."last_name" FROM "customer"
2  INNER JOIN "worker" ON "customer"."tg_username" = "worker"."tg_username";
```

	first_name	last_name
1	Alex	Filozop

Рисунок 3.11 — Результат выполнения запроса №11

Запрос 12: вывести все заказы, связанные с ремонтом некоторой вещи. Код запроса представлен в листинге 3.12. Результат выполнения — на рисунке 3.12.

Листинг 3.12 — Текст запроса

```
SELECT * FROM "order" WHERE "description" LIKE 'Ремонт%';
```

```
1  SELECT * FROM "order" WHERE "description" LIKE 'Ремонт%';
```

	id	customer_id	worker_id	description	price	state	begin_time	last_state_update_time	end_time
1	1	1	1	Ремонт футболки	200	IN_PROGRESS	2022-11-28T21:52:28Z	2022-12-21T07:49:55Z	2022-11-29T21:52:28Z

Рисунок 3.12 — Результат выполнения запроса №12

Запрос 13: вывести все заказы, в которых участвовали шторы. Код запроса представлен в листинге 3.13. Результат выполнения — на рисунке 3.13.

Листинг 3.13 — Текст запроса

```
SELECT * FROM "order" WHERE "description" LIKE '%штор%';
```

```
1  SELECT * FROM "order" WHERE "description" LIKE '%штор%';
```

	id	customer_id	worker_id	description	price	state	begin_time	last_state_update_time	end_time
1	6	3	3	Пошив штор.	30000	IN_PROGRESS	2022-12-21T23:14:33Z	2022-12-21T23:14:33Z	2022-12-25T02:13:00Z
2	7	1	1	Пошив штор.	50090	IN_PROGRESS	2022-12-21T23:17:01Z	2022-12-21T23:17:01Z	2022-12-22T05:00:00Z

Рисунок 3.13 — Результат выполнения запроса №13

Запрос 14: изменить состояние заказа 4 с «В процессе» на «Готов» с автоматическим обновлением даты последнего изменения. Код запроса представлен в листинге 3.14. Результат выполнения — на рисунке 3.14.

Листинг 3.14 — Текст запроса

```
UPDATE "order"
  SET "state" = 'READY', "last_state_update_time" = (strftime('%Y-%m-%dT%H:%M:%SZ', 'now'))
  WHERE "id" = 4;
```

```
1  UPDATE "order"
2    SET "state" = 'READY', "last_state_update_time" = (strftime('%Y-%m-%dT%H:%M:%SZ', 'now'))
3    WHERE "id" = 4;
```

	id	customer_id	worker_id	description	price	state	begin_time	last_state_update_time	end_time
1	4	2	2	Шуба из норки.	8000054	READY	2022-12-21T09:20:33Z	2022-12-22T07:43:15Z	2022-12-25T15:00:00Z

Рисунок 3.14 — Результат выполнения запроса №14

Запрос 15: подтвердить дополнительное требование для заказа №5 и выставить изменение цена на 90 руб (9000 коп.). Код запроса представлен в листинге 3.15. Результат выполнения — на рисунке 3.15.

Листинг 3.15 — Текст запроса

```
UPDATE "additional_requirement"
       SET    "state" = 'APPROVED',
              "price_delta" = 9000
       WHERE "order_id" = 5;
```

```
1   UPDATE "additional_requirement"
2       SET "state" = 'APPROVED',
3           "price_delta" = 9000
4       WHERE "order_id" = 5;
5   SELECT * FROM "additional_requirement" WHERE "order_id" = 5;
```

	order_id	instant	type	description	state	price_delta
1	5	2022-12-22T02:17:18Z	NOTE	Хочу уменьшить величину в два раза.	APPROVED	9000

Рисунок 3.15 — Результат выполнения запроса №15

По результатам проведённых тестов можно сделать вывод, что как SQL-запросы, так и таблицы в базе данных, созданы правильно и работают корректно.

4 РАЗРАБОТКА КЛИЕНТСКОГО ПРИЛОЖЕНИЯ

4.1. Обоснование выбора языка программирования

В качества языка для разработки клиентской программы был выбран язык программирования Java. Данный выбор может быть обоснован тем, что Java зачастую выбирается в качестве языка для разработки на стороне сервера. В качестве IDE используется Eclipse for Java Developers.

4.2. Разработка интерфейса

Для отрисовки интерфейса на стороне сервера была использована библиотека TelegramAPI. Программа взаимодействует с базой данных посредством библиотеки SQLite-JDBC через интерфейс Java DataBase Connector (JDBC).

4.3. Алгоритм работы модулей

Алгоритм работы программы построен на основе архитектуры MVP. При получении текстовых данных объект класса бота передаёт полученные данные в Presenter, предварительно обработав их. Presenter в свою очередь определяет текущее состояние диалога с пользователем и определяет, какой текст отобразить и как взаимодействовать с данными. Классы уровня модели обеспечивают взаимодействие программы с базой данных.

4.4. Тестирование работы приложения

В самом начале общения с ботом каждый пользователь получает сообщение-справку. Одновременно с этим сервер устанавливает определённую клавиатуру в зависимости от прав доступа (рисунки 4.1 — 4.3).

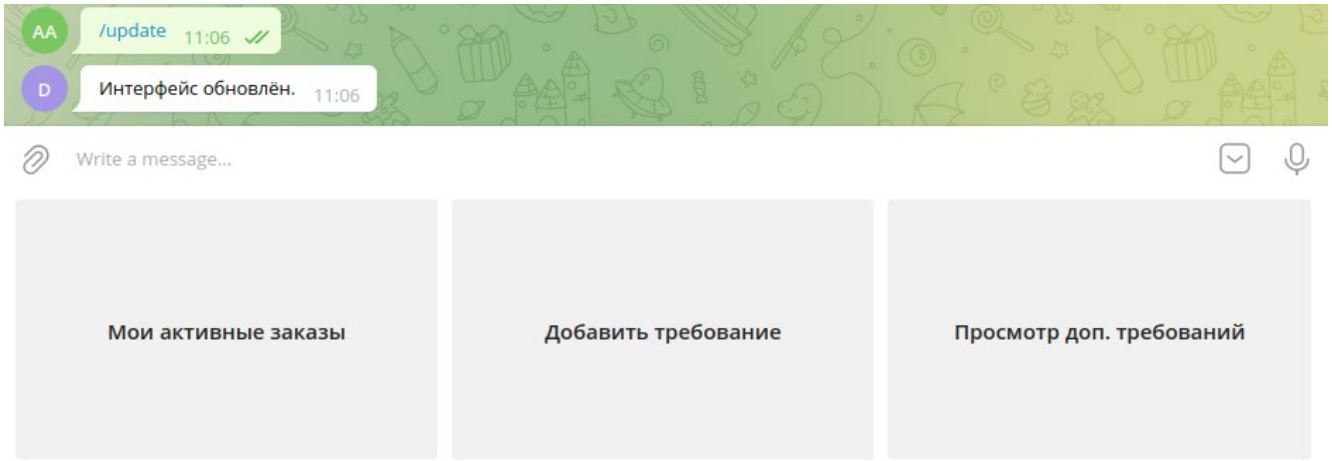

Рисунок 4.1 — Клавиатура клиента

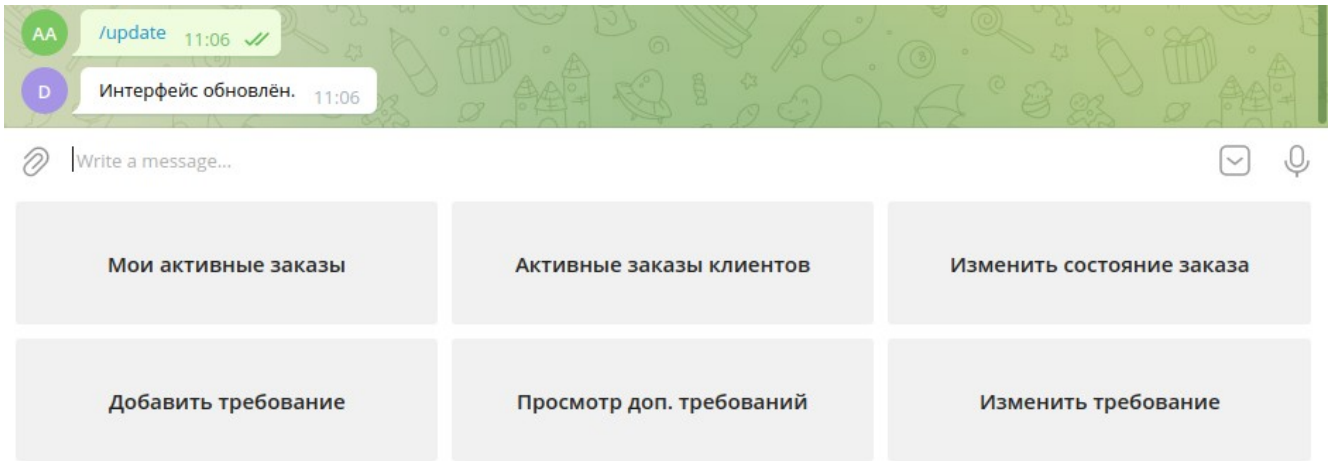

Рисунок 4.2 — Клавиатура обычного работника

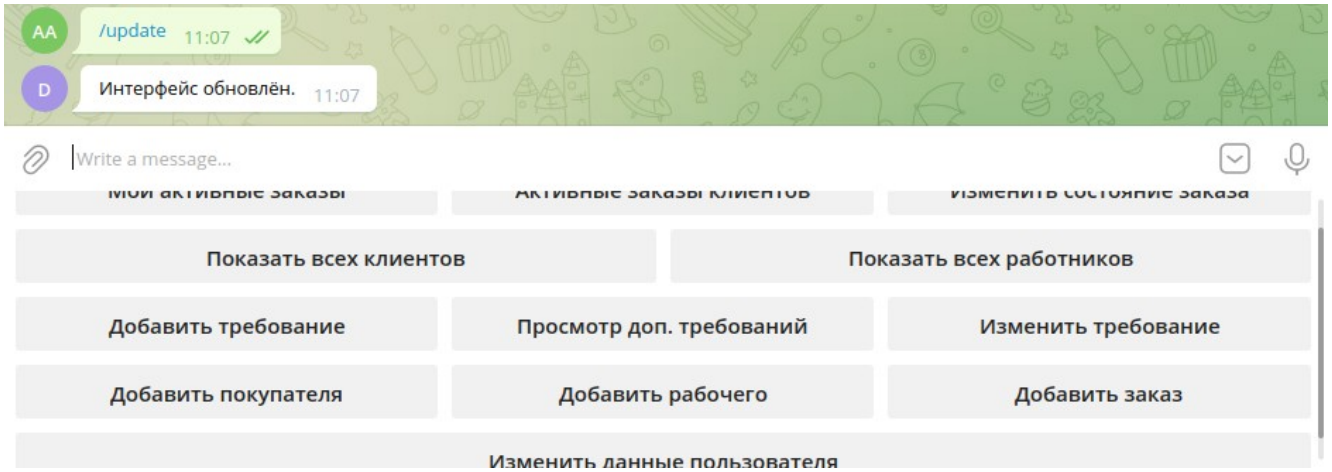

Рисунок 4.3 — Клавиатура администратора базы данных

Например, для добавления заказа работнику необходимо нажать на кнопку «Добавить заказ», затем вводить необходимые данные. Диалог представлен на рисунке 4.4.

Для работников по нажатию кнопки «Все активные заказы клиентов» бот отправит сообщение, в котором будут указаны все активные заказы для отправителя сообщения (рисунок 4.5).

Уникальной возможностью администратора является редактирование информации о пользователях. На рисунке 4.6 представлен пример диалога с ботом, в котором администратор изменяет Telegram username у пользователя-клиента на введённый в последнем сообщении.

Добавлять дополнительные требования может только клиент при нажатии на соответствующую кнопку, а изменять состояние требования — только работник, работающий с соответствующим заказом, по нажатию кнопки. Соответствующие диалоги продемонстрированы на рисунках 4.7 и 4.8.

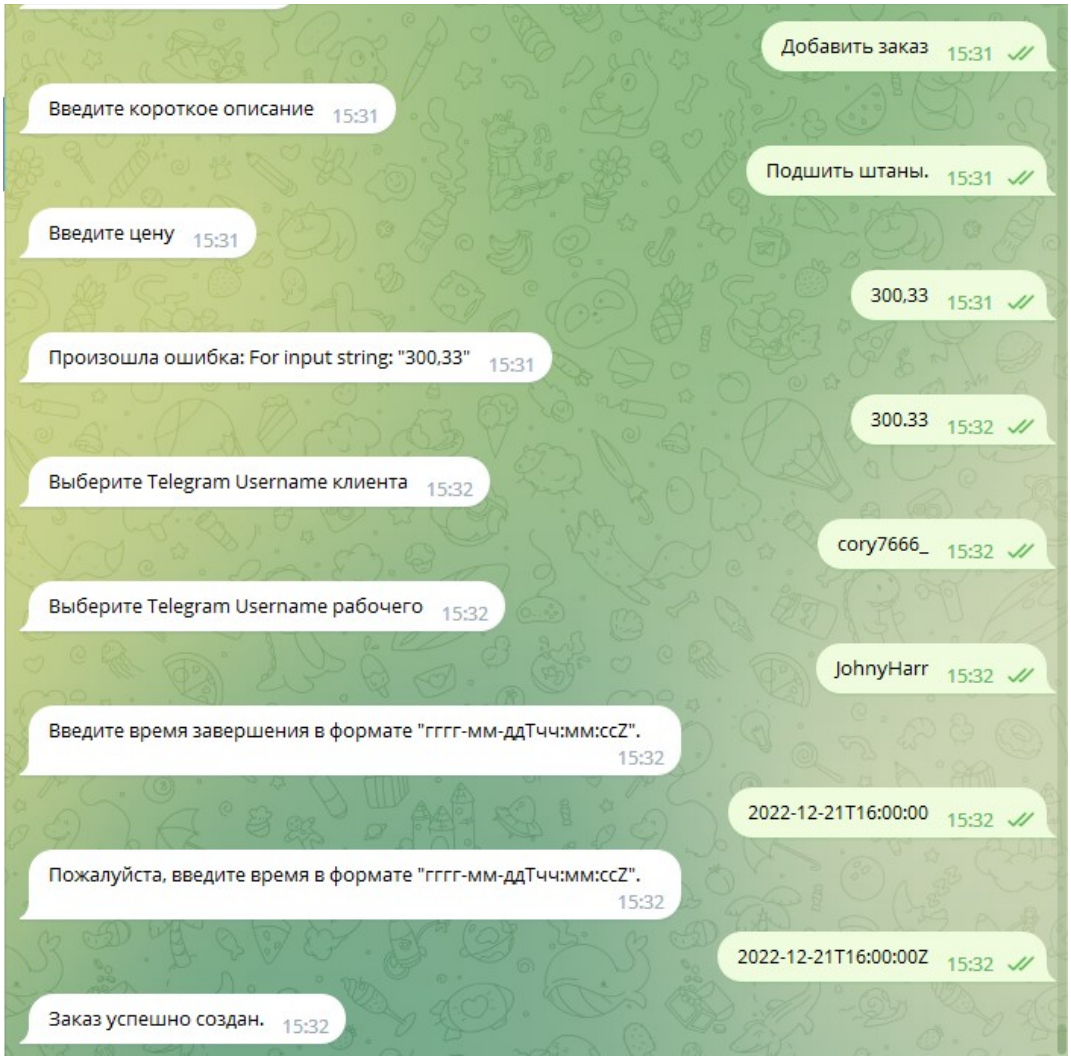

Рисунок 4.4 — Создание работником нового заказа

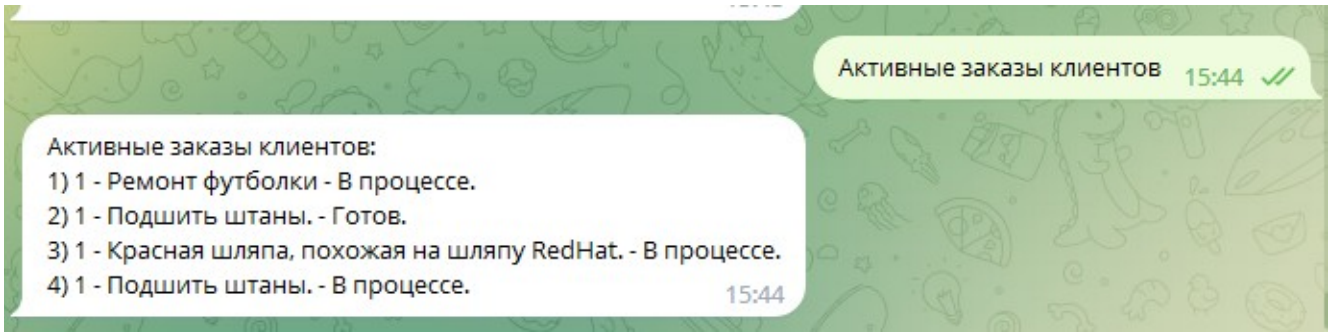

Рисунок 4.5 — Просмотр активных заказов работника

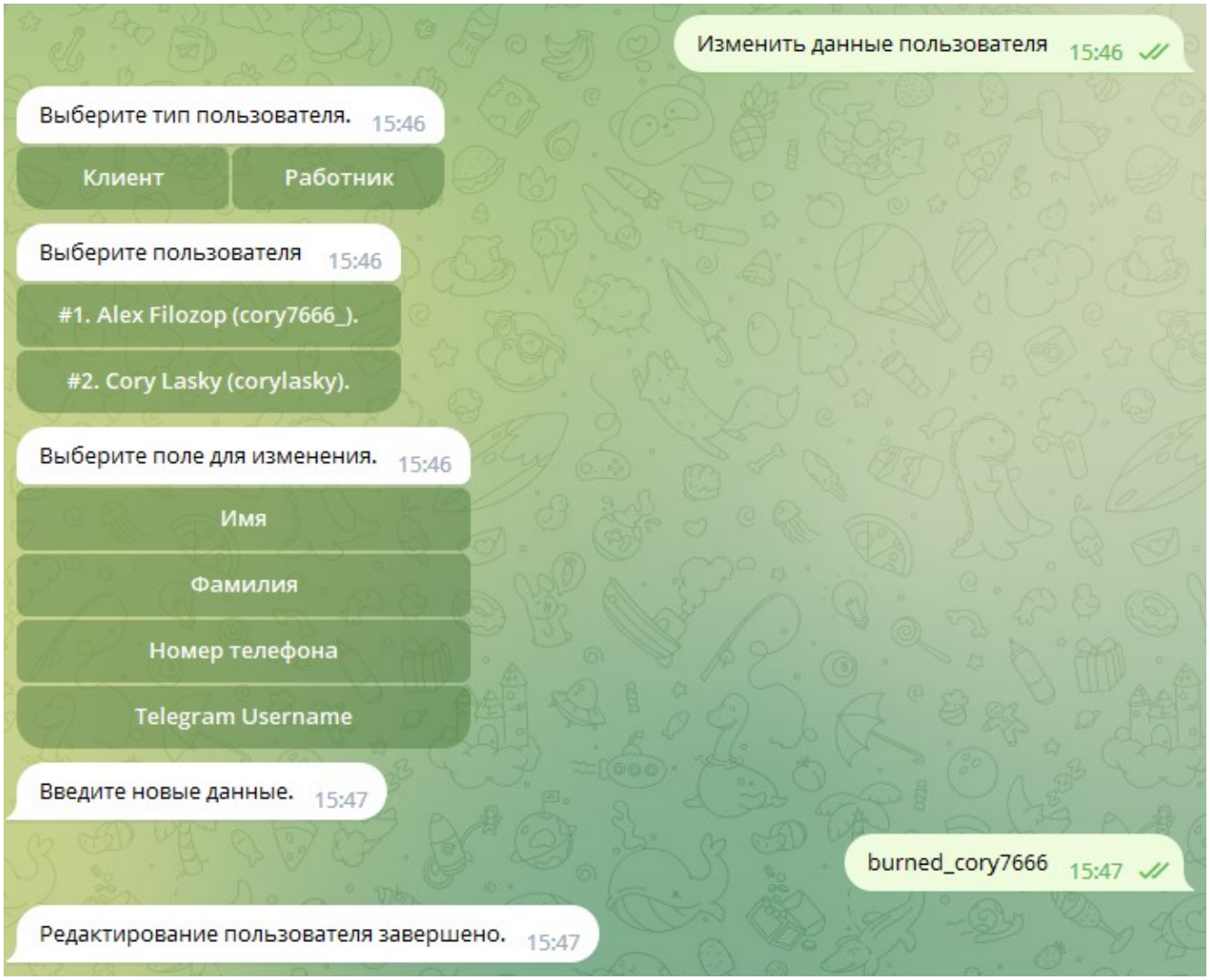

Рисунок 4.6 — Редактирование информации пользователя

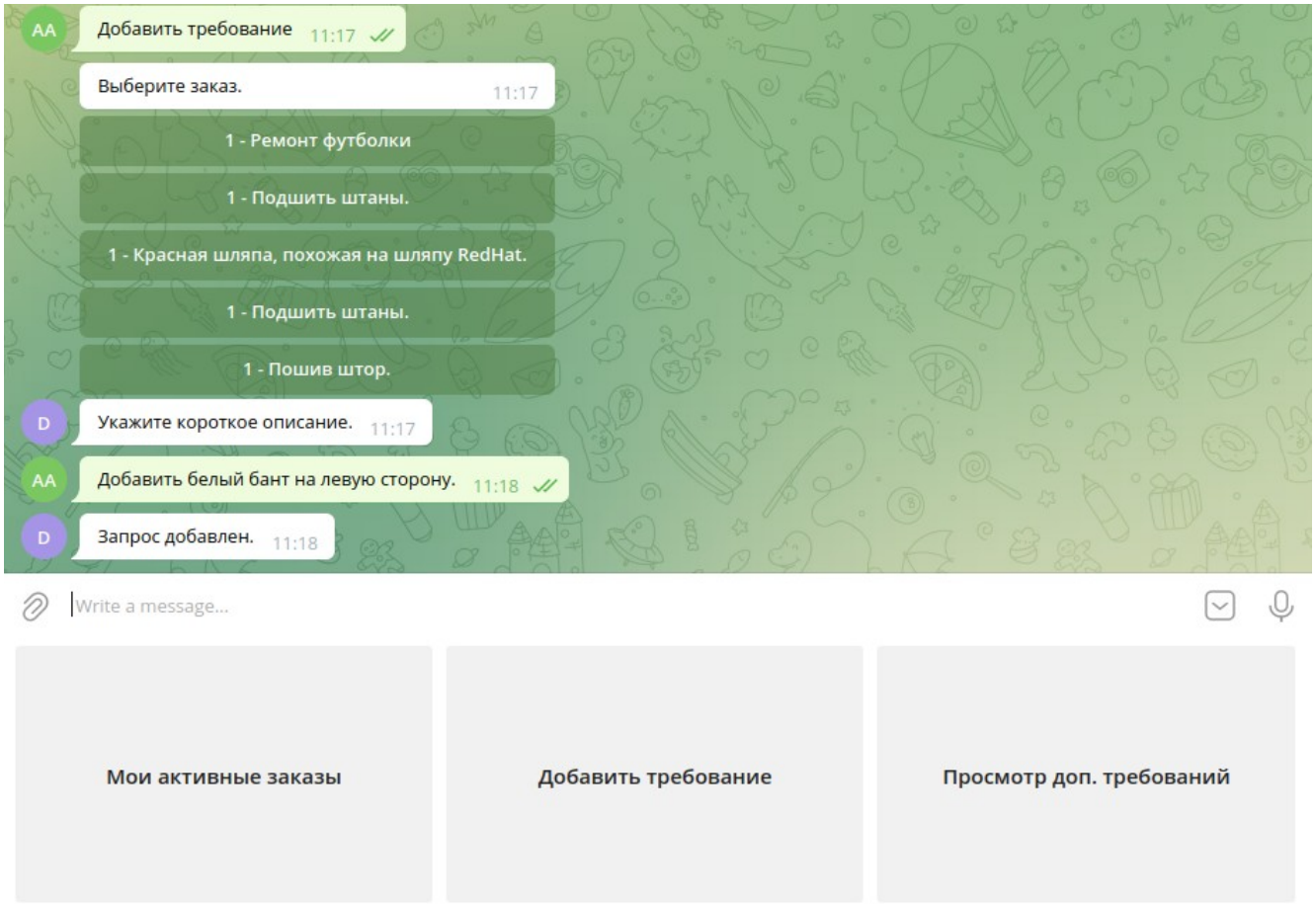

Рисунок 4.7 — Добавление требования посредством диалога

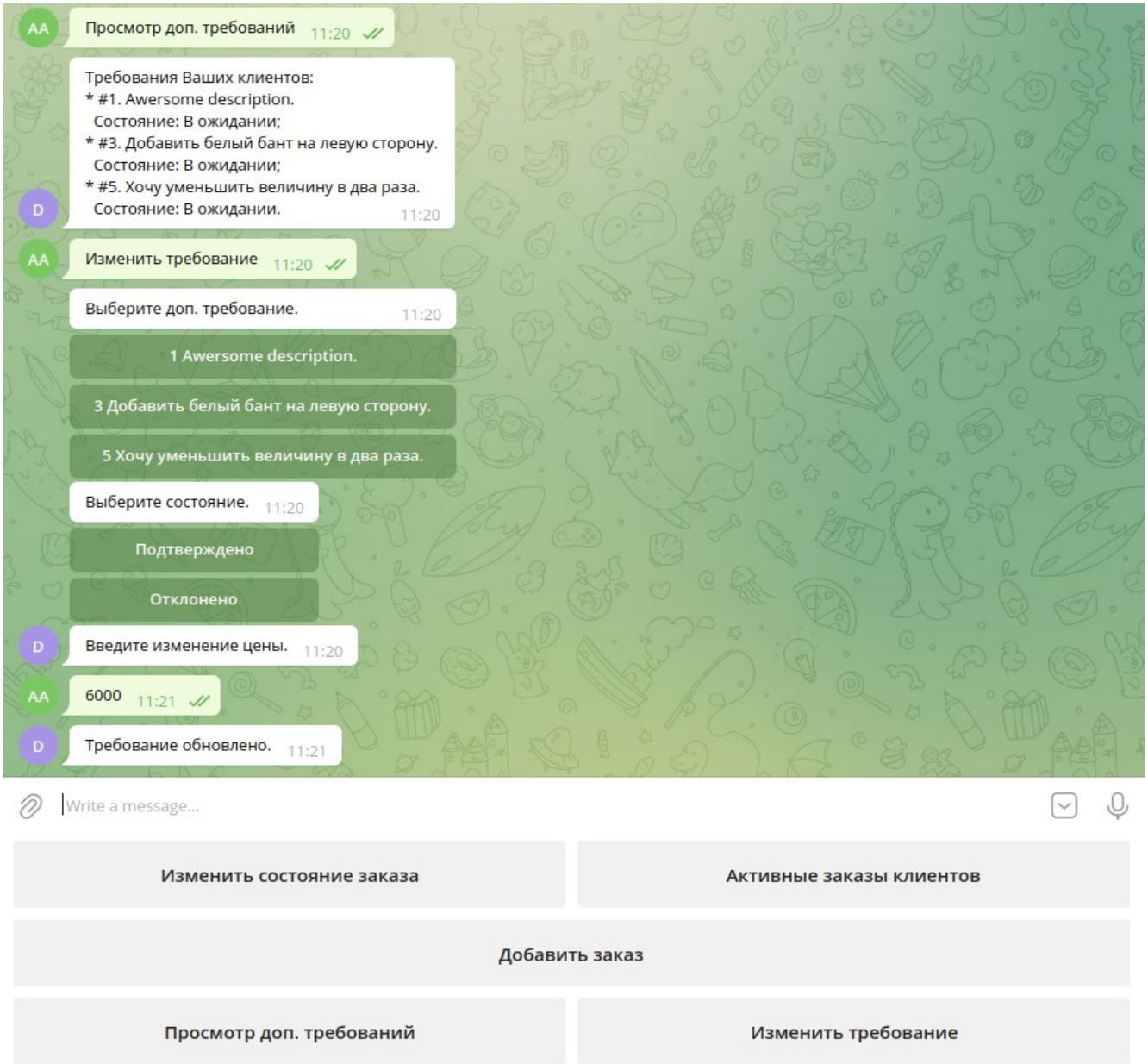

Рисунок 4.8 — Изменение состояния требования и изменения цены с помощью диалога

По результатам тестирования можно сделать вывод, что программа работает так, как задумывалось на этапе изучения предметной области.

ЗАКЛЮЧЕНИЕ

В соответствии с вариантом задания была разработана база данных, позволяющая добавлять, редактировать, просматривать и удалять данные. При построении базы данных были пройдены такие этапы проектирования как: концептуальное проектирование, логическое проектирование, физическое проектирование и использования полученной физической модели в разрабатываемом приложении. На этапе концептуального проектирования были рассмотрены все необходимые для данной предметной области сущности, их связи между собой. На этапе логического проектирования была спроектирована полная атрибутивная модель базы данных, удовлетвояющая нотации IDEF1X. Для этого при логическом проектировании были пройдены этапы создания моделей сущность-связь, создания модели, основанной на ключах с последующей нормализацией данной модели. В результате по полученной логической модели была создана физическая модель, используемая в программе.

Таким образом, была достигнута цель расчетно-графической работы – были получены и закреплены практические и теоретические навыки концептуального, логического и физического проектирования баз данных а также использования внутри разрабатываемых приложений СУБД, использующих принципы SQL.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. https://habr.com/ru/post/432548/
2. https://github.com/pengrad/java-telegram-bot-api

ПРИЛОЖЕНИЕ А

Запросы создания таблиц на языке SQL

```
CREATE TABLE "service_add_req_state" (
"name"TEXT,
PRIMARY KEY("name")
) WITHOUT ROWID;

CREATE TABLE "service_order_state" (
"name"TEXT,
PRIMARY KEY("name")
) WITHOUT ROWID;

CREATE TABLE "service_worker_role" (
"name"TEXT,
PRIMARY KEY("name")
) WITHOUT ROWID;

CREATE TABLE "service_stage_type" (
"type"TEXT,
PRIMARY KEY("type")
) WITHOUT ROWID;

CREATE TABLE "customer" (
"id"INTEGER,
"first_name"TEXT NOT NULL,
"last_name"TEXT NOT NULL,
"tel_number"TEXT NOT NULL,
"tg_username"TEXT NOT NULL,
"tg_chat_id"INTEGER NOT NULL,
PRIMARY KEY("id" AUTOINCREMENT)
);

CREATE TABLE "worker" (
"id"INTEGER,
"role"TEXT NOT NULL,
"first_name"TEXT NOT NULL,
"last_name"TEXT NOT NULL,
"tel_number"TEXT NOT NULL,
"tg_username"TEXT NOT NULL,
"tg_chat_id"INTEGER NOT NULL,
FOREIGN KEY("role") REFERENCES "service_worker_role"("name") ON UPDATE CASCADE ON DELETE
CASCADE,
PRIMARY KEY("id" AUTOINCREMENT)
);

CREATE TABLE "order" (
"id"INTEGER,
"customer_id"INTEGER NOT NULL,
"worker_id"INTEGER NOT NULL,
"description"TEXT NOT NULL,
"price"INTEGER NOT NULL,
"state"TEXT NOT NULL,
"begin_time"TEXT NOT NULL DEFAULT (strftime('%Y-%m-%dT%H:%M:%SZ', 'now')),
"last_state_update_time"TEXT NOT NULL DEFAULT (strftime('%Y-%m-%dT%H:%M:%SZ', 'now')),
"end_time"TEXT NOT NULL,
FOREIGN KEY("state") REFERENCES "service_order_state"("name") ON UPDATE CASCADE ON DELETE
CASCADE,
FOREIGN KEY("worker_id") REFERENCES "worker"("id") ON UPDATE CASCADE ON DELETE CASCADE,
FOREIGN KEY("customer_id") REFERENCES "customer"("id") ON UPDATE CASCADE ON DELETE CASCADE,
PRIMARY KEY("id" AUTOINCREMENT)
);
```

```
CREATE TABLE "additional_requirement" (
"order_id"INTEGER,
"instant"TEXT DEFAULT (strftime('%Y-%m-%dT%H:%M:%SZ', 'now')),
"type"TEXT NOT NULL,
"description"TEXT NOT NULL,
"state"TEXT NOT NULL,
"price_delta"INTEGER NOT NULL,
FOREIGN KEY("state") REFERENCES "service_add_req_state"("name") ON UPDATE CASCADE ON DELETE
CASCADE,
PRIMARY KEY("order_id","instant")
);

CREATE TABLE "stage" (
"id"TEXT,
"type"TEXT,
"data"TEXT,
PRIMARY KEY("id"),
FOREIGN KEY("type") REFERENCES "service_stage_type"("type") ON UPDATE CASCADE ON DELETE
CASCADE
) WITHOUT ROWID;
```

ПРИЛОЖЕНИЕ Б

Листинг основных классов

Листинг 1 — Код интерфейса DatabaseConnectable

```
package com.undefinedgroup.epsilonbot.model.database;

import java.sql.PreparedStatement;
import java.sql.SQLException;

public interface DatabaseConnectable
        extends AutoCloseable
{
  PreparedStatement createPreparedStatement (String sql)
          throws SQLException;
}
```

Листинг 2 — Код класса SQLiteDBConnection

```
package com.undefinedgroup.epsilonbot.model.database;

import java.io.File;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class SQLiteDBConnection
        implements DatabaseConnectable
{
  private final File dbFile;
  private Connection connection;

  public SQLiteDBConnection (File file)
  {
    this.dbFile = file;
    this.connection = null;
  }

  @Override
  public PreparedStatement createPreparedStatement (String sql)
          throws SQLException
  {
    if (connection == null)
    {
      connection = DriverManager.getConnection(String.format("jdbc:sqlite:%s",
dbFile.toString()));
    }
    return connection.prepareStatement(sql);
  }

  @Override
  public void close ()
          throws Exception
  {
    final var conn = connection;
    connection = null;
    conn.close();
  }
}
```

Листинг 3 — Код интерфейса ICustomerRepository

```
package com.undefinedgroup.epsilonbot.model.repository;

import java.util.List;

public interface ICustomerRepository<T>
{
  T add (T data);

  T update (T newData);

  T findByTelephoneNumber (String telephoneNumber);

  T findByUsername (String username);

  List<T> getAll ();
}
```

Листинг 4 — Код интерфейса IOrderRepository

```
package com.undefinedgroup.epsilonbot.model.repository;

import com.undefinedgroup.epsilonbot.model.data.CustomerInfo;
import com.undefinedgroup.epsilonbot.model.data.WorkerInfo;

import java.util.List;

public interface IOrderRepository<T, K>
{
  T add (T data);

  T findById (Integer orderId);

  List<T> findAllByCustomerInfo (CustomerInfo customerInfo);

  List<T> findAllByWorkerInfo (WorkerInfo workerInfo);

  T updateState (T order, K newState);
}
```

Листинг 5 — Код интерфейса IWorkerRepository

```
package com.undefinedgroup.epsilonbot.model.repository;

import java.util.List;

public interface IWorkerRepository<T>
{
  T add (T data);

  T update (T newData);

  T findByTelephoneNumber (String telephoneNumber);

  T findByUsername (String username);

  List<T> getAll ();
}
```

Листинг 6 — Код класса SQLiteCustomerRepository

```
package com.undefinedgroup.epsilonbot.model.repository;

import com.undefinedgroup.epsilonbot.model.data.CustomerInfo;
import com.undefinedgroup.epsilonbot.model.database.DatabaseConnectable;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.LinkedList;
import java.util.List;

public class SQLiteCustomerRepository
        implements ICustomerRepository<CustomerInfo>
{
  private final DatabaseConnectable connection;

  public SQLiteCustomerRepository (DatabaseConnectable connection)
  {
    this.connection = connection;
  }

  @Override
  public CustomerInfo add (CustomerInfo data)
  {
    try (final var stm = connection.createPreparedStatement(insertNewCustomerPattern))
    {
      stm.setString(1, data.firstName);
      stm.setString(2, data.lastName);
      stm.setString(3, data.telephoneNumber);
      stm.setString(4, data.tgUsername);
      stm.setLong(5, data.tgChatId);
      stm.execute();

      return findByTelephoneNumber(data.telephoneNumber);
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  @Override
  public CustomerInfo update (CustomerInfo newData)
  {
    try (final var stm = connection.createPreparedStatement(updateWorkerInfoPattern))
    {
      stm.setString(1, newData.firstName);
      stm.setString(2, newData.lastName);
      stm.setString(3, newData.telephoneNumber);
      stm.setString(4, newData.tgUsername);
      stm.setLong(5, newData.tgChatId);
      stm.setInt(6, newData.id);
      stm.execute();
      return newData;
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  @Override
  public CustomerInfo findByTelephoneNumber (String telephoneNumber)
  {
    try (final var stm =
connection.createPreparedStatement(selectCustomerByTelephoneNumberPattern))
    {
```

```
      stm.setString(1, telephoneNumber);
      try (final var result = stm.executeQuery())
      {
        result.next();
        return createCustomerInfoFromResultSet(result);
      }
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  @Override
  public CustomerInfo findByUsername (String username)
  {
    try (final var stm =
connection.createPreparedStatement(selectCustomerByTgUsernamePattern))
    {
      stm.setString(1, username);
      try (final var result = stm.executeQuery())
      {
        result.next();
        return createCustomerInfoFromResultSet(result);
      }
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  @Override
  public List<CustomerInfo> getAll ()
  {
    try (
            final var stm = connection.createPreparedStatement(selectAllCustomers);
            final var results = stm.executeQuery()
    )
    {
      final List<CustomerInfo> customers = new LinkedList<>();
      while (results.next())
      {
        customers.add(createCustomerInfoFromResultSet(results));
      }
      return customers;
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  private CustomerInfo createCustomerInfoFromResultSet (ResultSet result)
          throws SQLException
  {
    final Integer id = result.getInt("id");
    final String firstName = result.getString("first_name");
    final String lastName = result.getString("last_name");
    final String telephoneNumber = result.getString("tel_number");
    final String tgUsername = result.getString("tg_username");
    final Long tgChatId = result.getLong("tg_chat_id");
    return new CustomerInfo(id, firstName, lastName, telephoneNumber, tgUsername,
tgChatId);
  }

  private static final String insertNewCustomerPattern = """
```

```
          INSERT INTO "customer" ("first_name", "last_name", "tel_number", "tg_username",
"tg_chat_id")
            VALUES (?, ?, ?, ?, ?);
          """, selectCustomerByTelephoneNumberPattern = """
          SELECT * FROM "customer" WHERE "tel_number" = ?;
          """, selectCustomerByTgUsernamePattern = """
          SELECT * FROM "customer" WHERE "tg_username" = ?;
          """, selectAllCustomers = """
          SELECT * FROM "customer";
          """, updateWorkerInfoPattern = """
          UPDATE "customer"
             SET    "first_name" = ?,
                          "last_name" = ?,
                          "tel_number" = ?,
                          "tg_username" = ?,
                          "tg_chat_id" = ?
             WHERE "id" = ?;
          """;
}
```

Листинг 7 — Код класса SQLiteOrderRepository

```
package com.undefinedgroup.epsilonbot.model.repository;

import com.undefinedgroup.epsilonbot.model.data.CustomerInfo;
import com.undefinedgroup.epsilonbot.model.data.OrderInfo;
import com.undefinedgroup.epsilonbot.model.data.WorkerInfo;
import com.undefinedgroup.epsilonbot.model.database.DatabaseConnectable;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.Instant;
import java.time.format.DateTimeFormatter;
import java.util.LinkedList;
import java.util.List;

public class SQLiteOrderRepository
        implements IOrderRepository<OrderInfo, OrderInfo.State>
{
  private final DatabaseConnectable connection;

  public SQLiteOrderRepository (DatabaseConnectable connection)
  {
    this.connection = connection;
  }

  private OrderInfo createOrderFromResultSet (ResultSet result)
          throws SQLException
  {
    final Integer id = result.getInt("id");
    final Integer customerId = result.getInt("customer_id");
    final Integer workerId = result.getInt("worker_id");
    final String description = result.getString("description");
    final Integer price = result.getInt("price");
    final String beginTime = result.getString("begin_time");
    final String lastUpdateTime = result.getString("last_state_update_time");
    final String end_time = result.getString("end_time");
    final OrderInfo.State state = switch (result.getString("state"))
            {
              case "READY" -> OrderInfo.State.READY;
              case "COMPLETE" -> OrderInfo.State.COMPLETE;
              default -> OrderInfo.State.IN_PROGRESS;
            };

    return new OrderInfo(id,
                         customerId,
```

```
                        workerId,
                        description,
                        price,
                        state,
                        Instant.from(DateTimeFormatter.ISO_INSTANT.parse(beginTime)),
                        Instant.from(DateTimeFormatter.ISO_INSTANT.parse(lastUpdateTime)),
                        Instant.from(DateTimeFormatter.ISO_INSTANT.parse(end_time))
    );
  }

  private static final String insertOrderPattern = """
          INSERT INTO "order" ("customer_id", "worker_id", "description", "price", "state",
"end_time") VALUES (?, ?, ?, ?, ?, ?);
          """, updateOrderStatePattern = """
          UPDATE "order"
            SET "state" = ?, "last_state_update_time" = (strftime('%Y-%m-%dT%H:%M:%SZ',
'now'))
            WHERE "id" = ?;
          """, selectOrderByIdPattern = """
          SELECT * FROM "order"
            WHERE "id" = ?;
          """, selectOrdersByCustomerIdPattern = """
          SELECT * FROM "order"
            WHERE "customer_id" = ?;
          """, selectOrdersByWorkerIdPattern = """
          SELECT * FROM "order"
            WHERE "worker_id" = ?;
          """, selectOrdersBySpecificDataPattern = """
          SELECT * FROM "order"
            WHERE "customer_id" = ?
              AND "worker_id" = ?
              AND "description" = ?
              AND "price" = ?
            ORDER BY "id" DESC;
          """;

  @Override
  public OrderInfo add (OrderInfo data)
  {
    try (var stm = connection.createPreparedStatement(insertOrderPattern))
    {
      stm.setInt(1, data.customerId);
      stm.setInt(2, data.workerId);
      stm.setString(3, data.description);
      stm.setInt(4, data.price);
      stm.setString(5, data.currentState.sqlName);
      stm.setString(6, DateTimeFormatter.ISO_INSTANT.format(data.endTime));
      stm.execute();
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }

    try (var stm = connection.createPreparedStatement(selectOrdersBySpecificDataPattern))
    {
      stm.setInt(1, data.customerId);
      stm.setInt(2, data.workerId);
      stm.setString(3, data.description);
      stm.setInt(4, data.price);
      try (var result = stm.executeQuery())
      {
        result.next();
        return createOrderFromResultSet(result);
      }
    }
    catch (Exception ex)
    {
```

```
      throw new RuntimeException(ex);
    }
  }

  @Override
  public OrderInfo findById (Integer orderId)
  {
    try (final var stm = connection.createPreparedStatement(selectOrderByIdPattern))
    {
      stm.setInt(1, orderId);
      try (final var result = stm.executeQuery())
      {
        result.next();
        return createOrderFromResultSet(result);
      }
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  @Override
  public List<OrderInfo> findAllByCustomerInfo (CustomerInfo customerInfo)
  {
    try (final var stm =
connection.createPreparedStatement(selectOrdersByCustomerIdPattern))
    {
      stm.setInt(1, customerInfo.id);
      try (final var queryResult = stm.executeQuery())
      {
        final List<OrderInfo> ret = new LinkedList<>();
        while (queryResult.next())
        {
          ret.add(createOrderFromResultSet(queryResult));
        }
        return ret;
      }
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  @Override
  public List<OrderInfo> findAllByWorkerInfo (WorkerInfo workerInfo)
  {
    try (final var stm = connection.createPreparedStatement(selectOrdersByWorkerIdPattern))
    {
      stm.setInt(1, workerInfo.id);
      try (final var queryResult = stm.executeQuery())
      {
        final List<OrderInfo> ret = new LinkedList<>();
        while (queryResult.next())
        {
          ret.add(createOrderFromResultSet(queryResult));
        }
        return ret;
      }
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  @Override
```

```
  public OrderInfo updateState (OrderInfo order, OrderInfo.State newState)
  {
    try (final var stm = connection.createPreparedStatement(updateOrderStatePattern))
    {
      stm.setString(1, order.currentState.sqlName);
      stm.setInt(2, order.id);
      stm.execute();
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }

    return findById(order.id);
  }
}
```

Листинг 8 — Код класса SQLiteWorkerRepository

```
package com.undefinedgroup.epsilonbot.model.repository;

import com.undefinedgroup.epsilonbot.model.data.WorkerInfo;
import com.undefinedgroup.epsilonbot.model.database.DatabaseConnectable;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.LinkedList;
import java.util.List;

public class SQLiteWorkerRepository
        implements IWorkerRepository<WorkerInfo>
{
  final DatabaseConnectable connection;

  public SQLiteWorkerRepository (DatabaseConnectable connection)
  {
    this.connection = connection;
  }

  @Override
  public WorkerInfo add (WorkerInfo data)
  {
    try (final var stm = connection.createPreparedStatement(insertWorkerPattern))
    {
      stm.setString(1, data.role.sqlName);
      stm.setString(2, data.firstName);
      stm.setString(3, data.lastName);
      stm.setString(4, data.telephoneNumber);
      stm.setString(5, data.tgUsername);
      stm.setLong(6, data.tgChatId);
      stm.execute();
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }

    try (final var stm =
connection.createPreparedStatement(selectWorkerBySpecificFieldsPattern))
    {
      stm.setString(1, data.telephoneNumber);
      try (final var resultSet = stm.executeQuery())
      {
        resultSet.next();
        return createWorkerFromResultSet(resultSet);
      }
```

```
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  @Override
  public WorkerInfo update (WorkerInfo newData)
  {
    try (final var stm = connection.createPreparedStatement(updateWorkerInfoPattern))
    {
      stm.setString(1, newData.role.sqlName);
      stm.setString(2, newData.firstName);
      stm.setString(3, newData.lastName);
      stm.setString(4, newData.telephoneNumber);
      stm.setString(5, newData.tgUsername);
      stm.setLong(6, newData.tgChatId);
      stm.setInt(7, newData.id);
      stm.execute();
      return newData;
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  @Override
  public WorkerInfo findByTelephoneNumber (String telephoneNumber)
  {
    try (final var stm =
connection.createPreparedStatement(selectWorkerByTelephoneNumberPattern))
    {
      stm.setString(1, telephoneNumber);
      try (final var resultSet = stm.executeQuery())
      {
        resultSet.next();
        return createWorkerFromResultSet(resultSet);
      }
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  @Override
  public WorkerInfo findByUsername (String username)
  {
    try (final var stm = connection.createPreparedStatement(selectWorkerByUsernamePattern))
    {
      stm.setString(1, username);
      try (final var resultSet = stm.executeQuery())
      {
        resultSet.next();
        return createWorkerFromResultSet(resultSet);
      }
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  @Override
  public List<WorkerInfo> getAll ()
  {
```

```
    try (
            final var stm = connection.createPreparedStatement(selectAllWorkers);
            final var resultSet = stm.executeQuery()
    )
    {
      final List<WorkerInfo> workers = new LinkedList<>();

      while (resultSet.next())
      {
        workers.add(createWorkerFromResultSet(resultSet));
      }

      return workers;
    }
    catch (Exception ex)
    {
      throw new RuntimeException(ex);
    }
  }

  private WorkerInfo createWorkerFromResultSet (ResultSet result)
          throws SQLException
  {
    final Integer id = result.getInt("id");
    final String firstName = result.getString("first_name");
    final String lastName = result.getString("last_name");
    final String telephoneNumber = result.getString("tel_number");
    final String tgUsername = result.getString("tg_username");
    final Long tgChatId = result.getLong("tg_chat_id");
    final WorkerInfo.Role role = switch (result.getString("role"))
            {
              case "ADMIN" -> WorkerInfo.Role.ADMIN;
              default -> WorkerInfo.Role.PLAIN;
            };

    return new WorkerInfo(id, role, firstName, lastName, telephoneNumber, tgUsername,
tgChatId);
  }

  private static final String insertWorkerPattern = """
          INSERT INTO "worker" ("role", "first_name", "last_name", "tel_number",
"tg_username", "tg_chat_id")
             VALUES (?, ?, ?, ?, ?, ?);
          """, selectWorkerByTelephoneNumberPattern = """
          SELECT * FROM "worker" WHERE "tel_number" = ?;
          """, selectWorkerByUsernamePattern = """
          SELECT * FROM "worker" WHERE "tg_username" = ?;
          """, selectAllWorkers = """
          SELECT * FROM "worker";
          """, selectWorkerBySpecificFieldsPattern = """
          SELECT * FROM "worker"
             WHERE "tel_number" = ?;
          """, updateWorkerInfoPattern = """
          UPDATE "worker"
            SET "role" = ?,
                    "first_name" = ?,
                "last_name" = ?,
                "tel_number" = ?,
                "tg_username" = ?,
                "tg_chat_id" = ?
            WHERE "id" = ?;
          """;
}
```

Листинг 9 — Код класса IAdditionalRequirementRepository

```
package com.undefinedgroup.epsilonbot.model.repository;
public interface IAdditionalRequirementRepository<T>
{
	T add (T x);
	T getByOrderId (Integer x);
	T update (T x);
}
```

Листинг 10 — Код класса SQLiteAdditionalRequirementRepository

```
package com.undefinedgroup.epsilonbot.model.repository;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.Instant;
import java.time.format.DateTimeFormatter;

import com.undefinedgroup.epsilonbot.model.data.AdditionalRequirement;
import com.undefinedgroup.epsilonbot.model.database.DatabaseConnectable;

public class SQLiteAdditionalRequirementRepository implements
IAdditionalRequirementRepository<AdditionalRequirement>
{
	private final DatabaseConnectable connection;

	public SQLiteAdditionalRequirementRepository (DatabaseConnectable connection)
	{
		this.connection = connection;
	}

	@Override
	public AdditionalRequirement add (AdditionalRequirement x)
	{
		try
		{
			try (final var stm =
connection.createPreparedStatement(insertInfoPattern))
			{
				stm.setInt(1, x.orderId);
				stm.setString(2, x.description);
				stm.setInt(3, x.priceDelta);
				stm.execute();
			}
			return getByOrderId(x.orderId);
		}
		catch (Exception ex)
		{
			throw new RuntimeException(ex);
		}
	}

	@Override
	public AdditionalRequirement getByOrderId (Integer x)
	{
		try (final var stm =
connection.createPreparedStatement(selectInfoByIdPattern))
		{
			stm.setInt(1, x);
			try (final var resultSet = stm.executeQuery())
			{
				resultSet.next();
				return buildFromResult(resultSet);
			}
		}
```

```
		catch (Exception ex)
		{
			throw new RuntimeException(ex);
		}
	}

	@Override
	public AdditionalRequirement update (AdditionalRequirement x)
	{
		try (final var stm = connection.createPreparedStatement(updateInfoPattern))
		{
			stm.setString(1, x.description);
			stm.setString(2, x.state.sqlName);
			stm.setInt(3, x.priceDelta);
			stm.setInt(4, x.orderId);
			stm.execute();

			return x;
		}
		catch (Exception ex)
		{
			throw new RuntimeException(ex);
		}
	}

	private AdditionalRequirement buildFromResult (ResultSet result) throws SQLException
	{
		try
		{
			final var orderId = result.getInt("order_id");
			final var instant =
Instant.from(DateTimeFormatter.ISO_INSTANT.parse(result.getString("instant")));
			final var type = result.getString("type");
			final var description = result.getString("description");
			final var state =
AdditionalRequirement.State.fromString(result.getString("state"));
			final var priceDelta = result.getInt("price_delta");
			return new AdditionalRequirement(orderId, instant, type, description,
state, priceDelta);
		}
		catch (NullPointerException ex)
		{
			throw new NonexistentRecordException(ex);
		}
	}

	private static final String insertInfoPattern = """
		INSERT INTO "additional_requirement"
			("order_id", "description", "price_delta")
			VALUES (?, ?, ?);
		""", selectInfoByIdPattern = "SELECT * FROM \"additional_requirement\"
WHERE \"order_id\" = ?;",
		updateInfoPattern = """
			UPDATE "additional_requirement"
				SET "description" = ?,
					"state" = ?,
					"price_delta" = ?
				WHERE "order_id" = ?;
			""";
}
```