Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Технологии разработки программного обеспечения

ОТЧЁТ

по лабораторной работе

на тему

Проектирование и создание БД

Выполнил

Студент гр. 053502

Макаро М. В.

Проверил

Ассистент кафедры информатики

Гриценко Н.Ю.

Минск 2023

**СОДЕРЖАНИЕ**

[1 Цель работы 3](#_Toc148368332)

[2 Краткие теоретические сведения 4](#_Toc148368333)

[2.1 DF-диаграмма 4](#_Toc148368334)

[2.2 ER-диаграмма 4](#_Toc148368335)

[2.3 IDEF1.X-диаграмма 5](#_Toc148368336)

[3 Разработка структуры базы данных 6](#_Toc148368337)

[Выводы 10](#_Toc148368338)

[Приложение А 11](#_Toc148368339)

# 1 ЦЕЛЬ РАБОТЫ

Необходимо разработать структуру базы данных. Создать логическую и физическую модели БД. Использовать DFD, ERD и IDEF1.X диаграммы для визуализации и представления структуры (не менее 10 сущностей). Сравнить данные методы. Сгенерировать DDL скрипты. Создать БД. Подготовить отчет по результатам выполнения лабораторной работы.

# 2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

2.1 DF-диаграмма

DFD диаграммы в отличии от других нотаций позволяют визуально показать все процессы с точки зрения данных.

Выделяют 4 элемента в диаграмме:

* Процесс. Процессы, при которых идет изменение потока данных (обработка, трансформация и др. изменения). Процесс, как и в других диаграммах, обычно прописывается с помощью глагола, например: “Отправка заполненной формы”.
* Внешняя сущность. Сущность (объект), которая получает или отправляете данные при взаимодействии с описанным процессом.
* Хранилище данных. Все хранилища данных или отдельные файлы, которые хранят исходные или выходные данные, а также все промежуточные хранилища.
* Поток данных. Поток данных, который отображает направление и сами данные, которые перемещаются между внешними сущностями и хранилищами данных с помощью процессов.

Диаграмма позволяет визуализировать как движение данных между объектами системы, так и преобразования данных, которые могут применяться на разных шагах процесса.

Самые распространенные системы нотации DFD-схем названы в честь их создателей:

* Йордон и Коуд;
* Йордон и Де Марко;
* Гейн и Сарсон.

Основное различие между этими системами заключается в том, что методы Йордона-Коуда и Йордона-Де Марко для обозначения процессов применяют круги, а метод Гейна-Сарсона — прямоугольные блоки со скругленными углами (которые иногда называют «конфетками»).

## 2.2 ER-диаграмма

Схема «сущность-связь» (также ERD или ER-диаграмма) – это разновидность блок-схемы, где показано, как разные «сущности» (люди, объекты, концепции и так далее) связаны между собой внутри системы. ER-диаграммы чаще всего применяются для проектирования и отладки реляционных баз данных в сфере образования, исследования и разработки программного обеспечения и информационных систем для бизнеса. ER-диаграммы (или ER-модели) полагаются на стандартный набор символов, включая прямоугольники, ромбы, овалы и соединительные линии, для отображения сущностей, их атрибутов и связей. Эти диаграммы устроены по тому же принципу, что и грамматические структуры: сущности выполняют роль существительных, а связи – глаголов.

ER-диаграммы – «родственники» схем структуры данных (DSD), где вместо связей между самими сущностями отображаются отношения между элементами внутри них. ER-диаграммы часто используются в сочетании с диаграммами DFD, которые схематично показывают движение потоков информации в рамках процесса или системы.

## 2.3 IDEF1.X-диаграмма

IDEF1X является методом для разработки реляционных баз данных и использует условный синтаксис, специально разработанный для удобного построения концептуальной схемы. Концептуальной схемой мы называем универсальное представление структуры данных в рамках коммерческого предприятия, независимое от конечной реализации базы данных и аппаратной платформы. Будучи статическим методом разработки, IDEF1X изначально не предназначен для динамического анализа по принципу "AS IS", тем не менее, он иногда применяется в этом качестве, как альтернатива методу IDEF1. Использование метода IDEF1X наиболее целесообразно для построения логической структуры базы данных после того, как все информационные ресурсы исследованы (скажем с помощью метода IDEF1) и решение о внедрении реляционной базы данных, как части корпоративной информационной системы, было принято. Однако не стоит забывать, что средства моделирования IDEF1X специально разработаны для построения реляционных информационных систем, и если существует необходимость проектирования другой системы, скажем объектно-ориентированной, то лучше избрать другие методы моделирования.

Существует несколько очевидных причин, по которым IDEF1X не следует применять в случае построения нереляционных систем. Во-первых, IDEF1X требует от проектировщика определить ключевые атрибуты, для того чтобы отличить одну сущность от другой, в то время как объектно-ориентированные системы не требуют задания ключевых ключей, в целях идентифицирования объектов. Во-вторых, в тех случаях, когда более чем один атрибут является однозначно идентифицирующим сущность, проектировщик должен определить один из этих атрибутов первичным ключом, а все остальные вторичными. И, таким образом, построенная проектировщиком IDEF1X-модель и переданная для окончательной реализации программисту является некорректной для применения методов объектно-ориентированной реализации, и предназначена для построения реляционной системы.

# 3 РАЗРАБОТКА СТРУКТУРЫ БАЗЫ ДАННЫХ

Было выбрано использовать sql-базу данных, т.к. язык SQL универсален для всех реляционных хранилищ и в случае смены СУБД не придётся переписывать весь код. Преимущества РСУБД – соответствия базы данных требованиям ACID, целостность данных, структурированность. Механизм SQL позволяет воспользоваться тем, что заложено в базу данных на этапе её проектирования, а NoSQL решения не требуют определять схему базы данных перед началом работы, поэтому в процессе разработки можно наткнуться на непредвиденные трудности, которые могут привести к отказу от данного NoSQL решения.

Было решено использовать MySQL, как одна из самых распространенных СУБД, имеющая следующие плюсы:

* Высокая скорость работы;
* Поддержка практически всех CMS;
* Бесплатная лицензия;
* Надежная и простая система безопасности;
* Плагины, позволяющие упростить и настроить работу под себя;
* В одной таблице может содержаться несколько миллионов записей.

При выполнении данной лабораторной работы необходимо было создать логическую и физическую модели базы данных разрабатываемого программного продукта, используя DFD, ERD и IDEF1.X диаграммы для визуализации.

DFD диаграмма на логическом уровне в нотации Гейна-Сарсона представлена на рисунках 3.1 и 3.2.

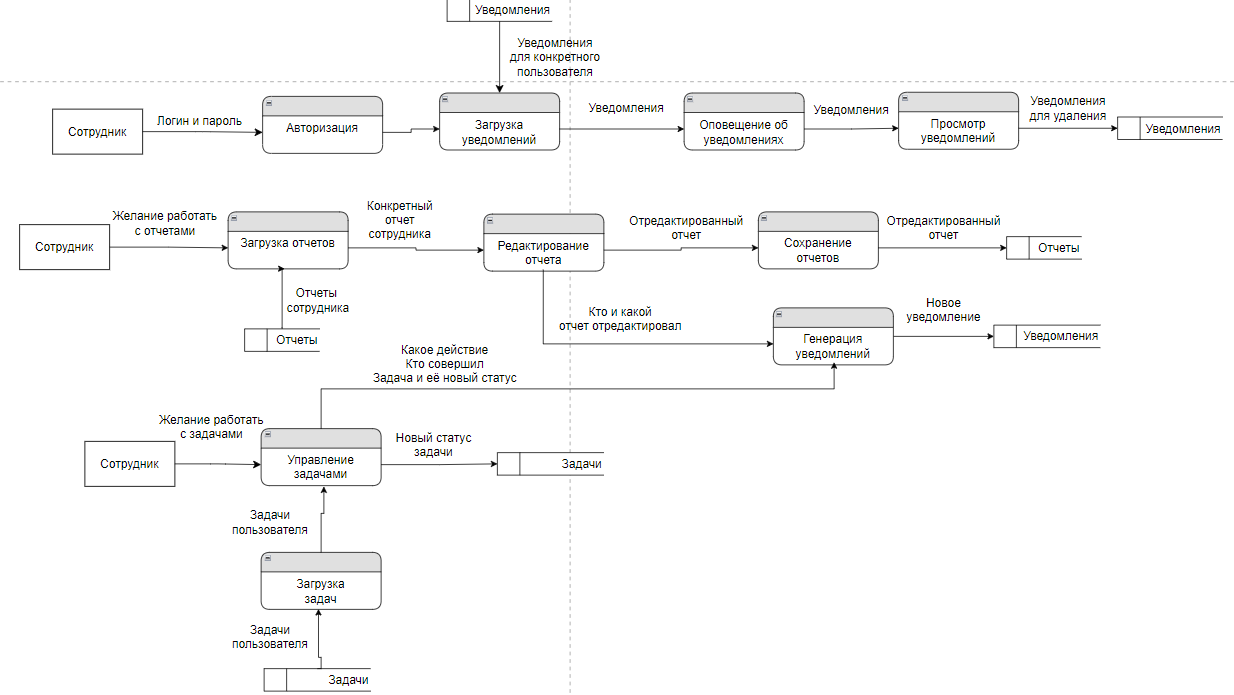


Рисунок 3.1 – DFD диаграмма (часть 1)

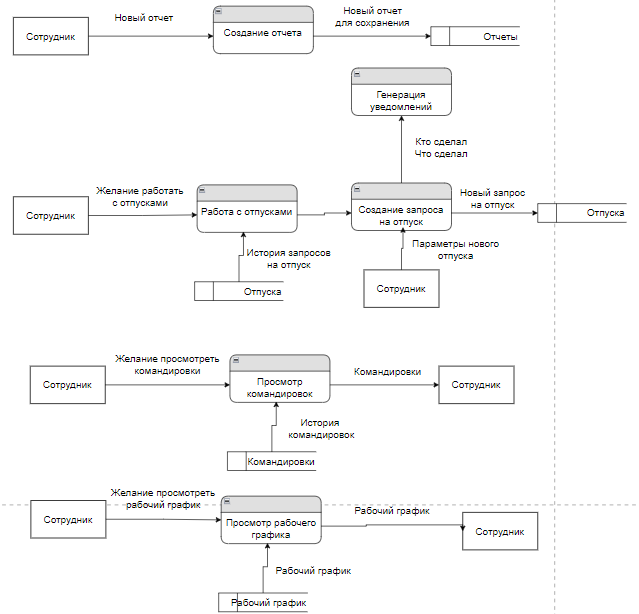


Рисунок 3.2 – DFD диаграмма (часть 2)

ERD диаграмма на уровне физической модели данных представлена на рисунке 3.3

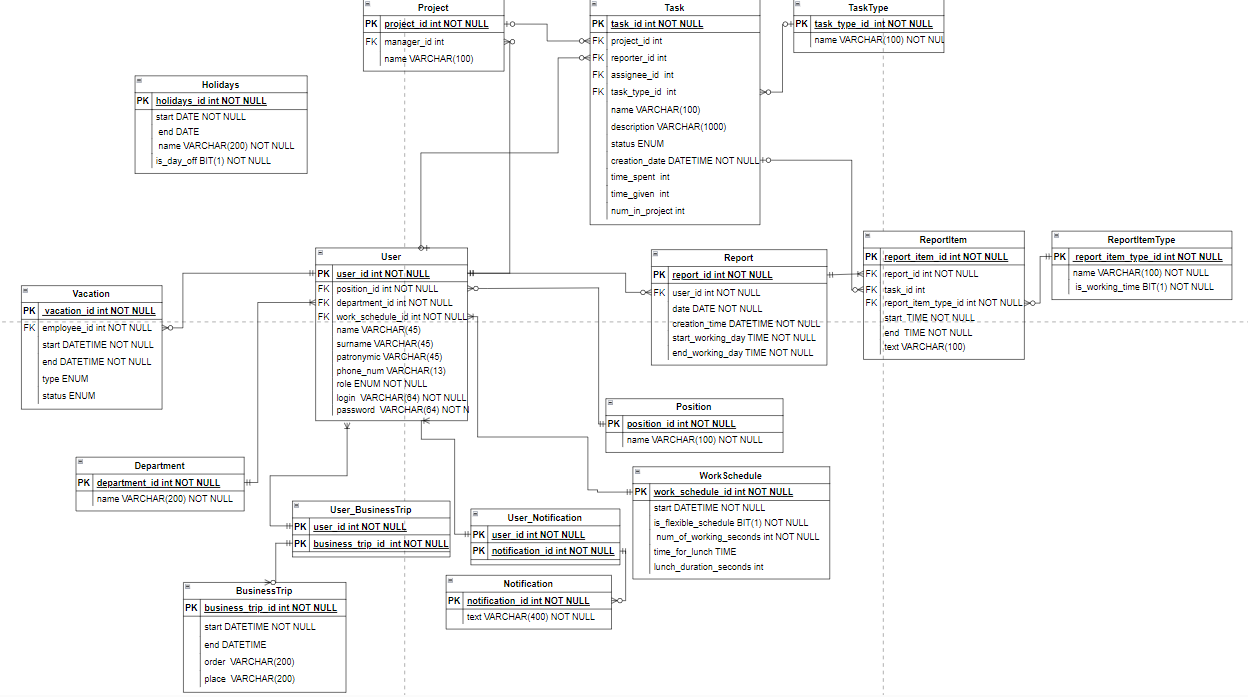


Рисунок 3.3 – ERD диаграмма базы данных на физическом уровне

ERD диаграмма на уровне логической модели данных представлена на рисунке 3.4

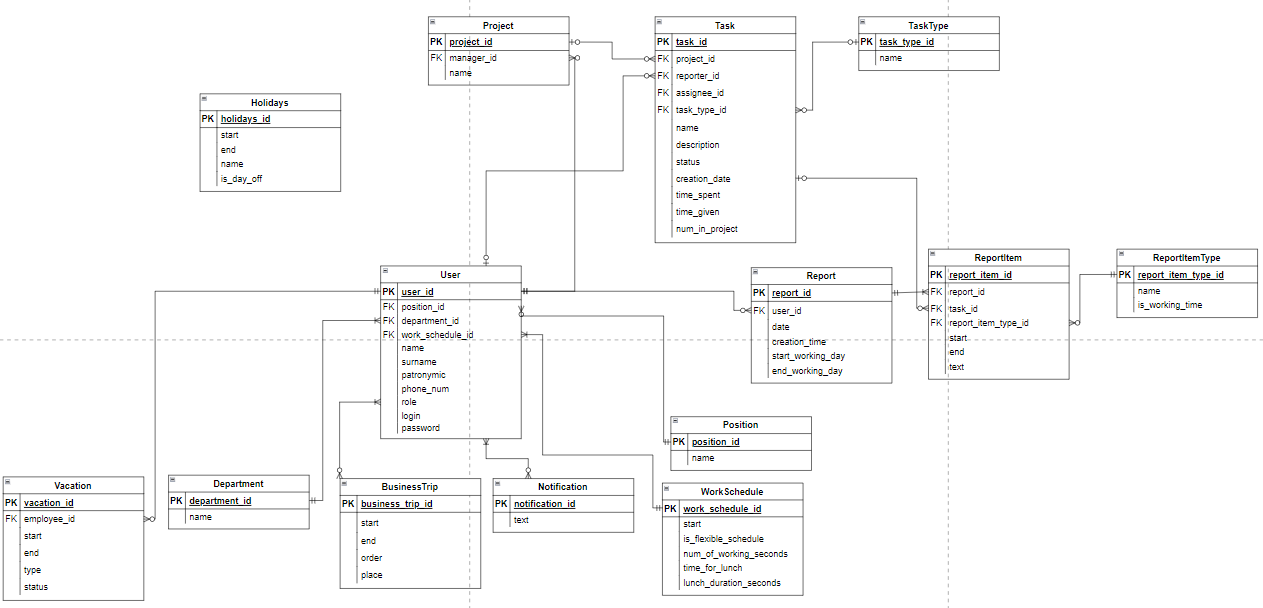


Рисунок 3.4 – ERD диаграмма базы данных на логическом уровне

ERD диаграмма на уровне концептуальной модели данных представлена на рисунке 3.5

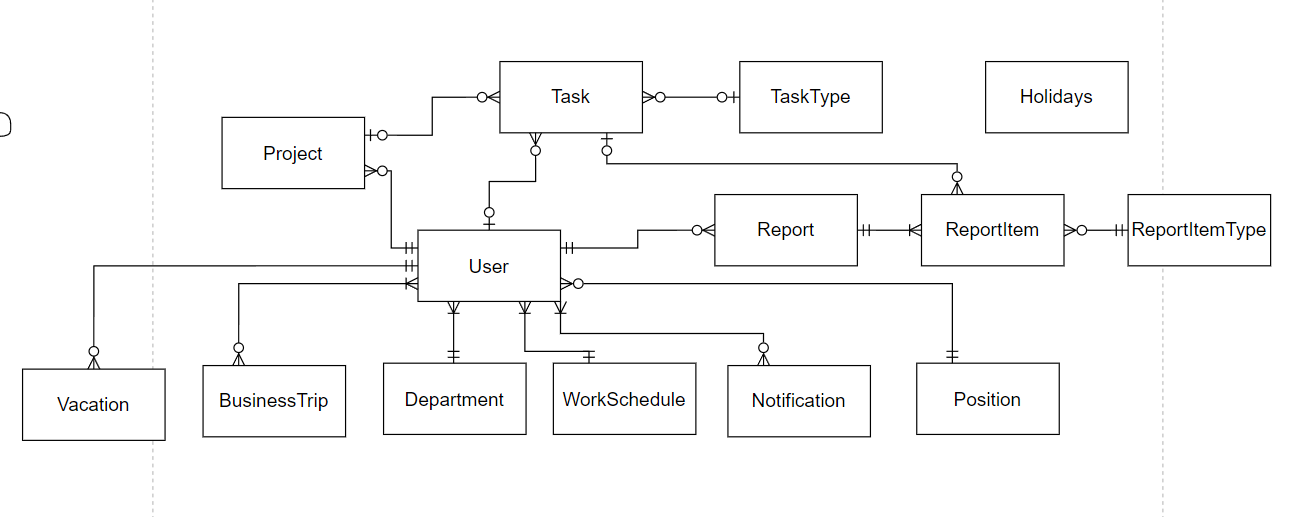


Рисунок 3.5 – ERD диаграмма базы данных на концептуальном уровне

IDEF1.X диаграмма представлена на рисунке 3.6

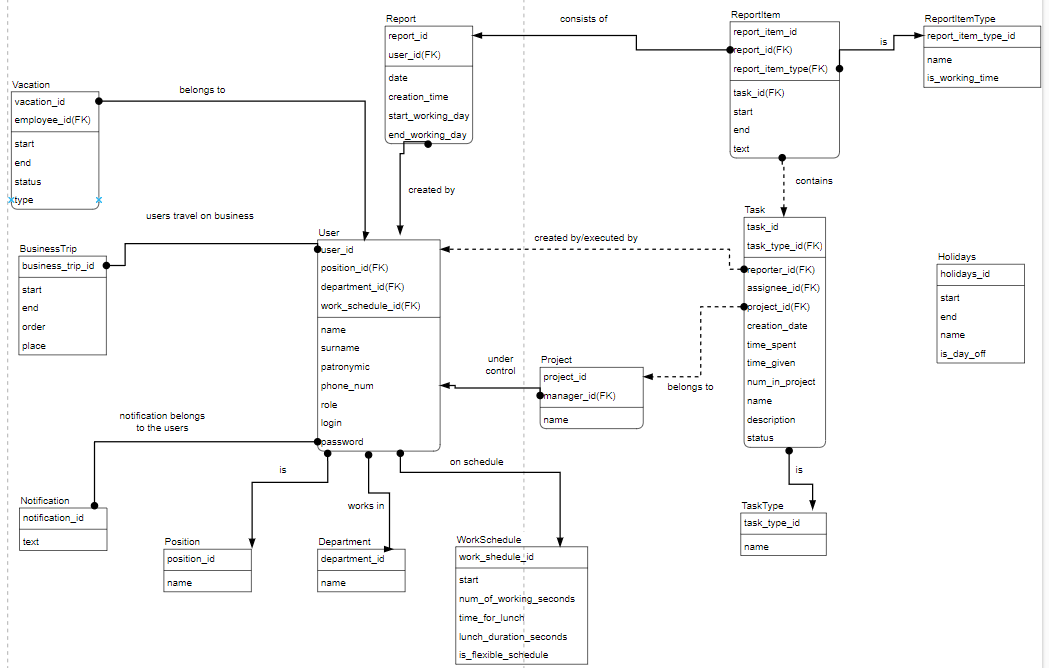


Рисунок 3.6 – IDEF1.X диаграмма базы данных

# ВЫВОДЫ

Таким образом, в ходе лабораторной работы были изучены понятия логической и физической модели базы данных, разобраны отличия DFD, ERD и IDEF1.X диаграмм. Была создана база данных для разрабатываемого приложения на основе схем, составленных в предыдущей и этой лабораторной. Было замечено, что для создание базы данных очень упрощается, при использовании ERD и IDEF1.X диаграмм, а диаграмма DFD способствует лучшему пониманию происходящих внутри системы процессов с данными.

В результате был получен ценный практический и теоретический опыт работы с базами данных. Это знание может быть полезным для дальнейшей работы в разработке приложений.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Код DDL скриптов для создания базы данных**

create database timeKeeping;

use timeKeeping;

CREATE TABLE Position(

position\_id INTEGER NOT NULL AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

PRIMARY KEY(position\_id ),

);

CREATE TABLE Project(

project\_id INTEGER NOT NULL AUTO\_INCREMENT,

manager\_id INTEGER NOT NULL,

PRIMARY KEY(project\_id ),

FOREIGN KEY(manager\_id) REFERENCES User(user\_id)

);

CREATE TABLE Task(

task\_id INTEGER NOT NULL AUTO\_INCREMENT,

project\_id INTEGER NOT NULL,

reporter\_id INTEGER NOT NULL,

assignee\_id INTEGER NOT NULL,

task\_type\_id INTEGER NOT NULL,

PRIMARY KEY(task\_id ),

FOREIGN KEY(project\_id) REFERENCES Project(project\_id),

FOREIGN KEY(reporter\_id) REFERENCES User(user\_id),

FOREIGN KEY(assignee\_id) REFERENCES User(user\_id),

FOREIGN KEY(task\_type\_id) REFERENCES TaskType(task\_type\_id)

);

CREATE TABLE TaskType(

task\_type\_id INTEGER NOT NULL AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

PRIMARY KEY(task\_type\_id),

);

CREATE TABLE Vacation(

vacation\_id INTEGER NOT NULL AUTO\_INCREMENT,

employee\_id INTEGER NOT NULL,

start DATETIME NOT NULL,

end DATETIME NOT NULL,

type ENUM('Working vacation', 'Unpaid leave', 'Paid leave', 'Maternity leave')

status ENUM('disapprove', 'approve'),

PRIMARY KEY(vacation\_id ),

FOREIGN KEY(employee\_id) REFERENCES User(user\_id)

);

CREATE TABLE BusinessTrip(

business\_trip\_id INTEGER NOT NULL AUTO\_INCREMENT,

start DATETIME NOT NULL,

end DATETIME,

order VARCHAR(200),

place VARCHAR(200),

PRIMARY KEY(business\_trip\_id),

);

CREATE TABLE Department(

department\_id INTEGER NOT NULL AUTO\_INCREMENT,

name VARCHAR(200) NOT NULL,

PRIMARY KEY(department\_id),

);

CREATE TABLE Report(

report\_id INTEGER NOT NULL AUTO\_INCREMENT,

user\_id INTEGER NOT NULL,

date DATE NOT NULL,

creation\_time DATETIME NOT NULL,

start\_working\_day TIME NOT NULL,

end\_working\_day TIME NOT NULL,

PRIMARY KEY(report\_id ),

FOREIGN KEY(user\_id) REFERENCES User(user\_id)

);

CREATE TABLE ReportItemType(

report\_item\_type\_id INTEGER NOT NULL AUTO\_INCREMENT,

name VARCHAR(100) NOT NULL,

is\_working\_time BIT(1) NOT NULL,

PRIMARY KEY(report\_item\_type\_id),

);

CREATE TABLE ReportItem(

report\_item\_id INTEGER NOT NULL AUTO\_INCREMENT,

report\_id INTEGER NOT NULL,

task\_id INTEGER NOT NULL,

report\_item\_type\_id INTEGER NOT NULL,

start TIME NOT NULL

end TIME NOT NULL

text VARCHAR(100)

PRIMARY KEY(report\_item\_id ),

FOREIGN KEY(report\_id) REFERENCES Report(report\_id)

FOREIGN KEY(task\_id) REFERENCES Task(task\_id)

FOREIGN KEY(report\_item\_type\_id) REFERENCES ReportItemType(report\_item\_type\_id)

);

CREATE TABLE WorkSchedule(

work\_schedule\_id INTEGER NOT NULL AUTO\_INCREMENT,

start DATETIME NOT NULL,

is\_flexible\_schedule BIT(1) NOT NULL,

num\_of\_working\_seconds INTEGER NOT NULL,

time\_for\_lunch TIME,

lunch\_duration\_seconds INTEGER,

PRIMARY KEY(work\_schedule\_id),

);

CREATE TABLE Holidays(

holidays\_id INTEGER NOT NULL AUTO\_INCREMENT,

start DATE NOT NULL,

end DATE,

name VARCHAR(200) NOT NULL,

is\_day\_off BIT(1) NOT NULL,

PRIMARY KEY(holidays\_id),

);

CREATE TABLE Notification(

notification\_id INTEGER NOT NULL AUTO\_INCREMENT,

text VARCHAR(400) NOT NULL

PRIMARY KEY(notification\_id),

);

CREATE TABLE User(

user\_id INTEGER NOT NULL AUTO\_INCREMENT,

position\_id INTEGER NOT NULL,

department\_id INTEGER NOT NULL,

work\_schedule\_id INTEGER NOT NULL,

name VARCHAR(45),

surname VARCHAR(45),

patronymic VARCHAR(45),

phone\_num VARCHAR(13),

role ENUM('Admin', 'Manager', 'Employee') NOT NULL

login VARCHAR(60) NOT NULL,

password VARCHAR(60) NOT NULL,

PRIMARY KEY(user\_id ),

FOREIGN KEY(position\_id) REFERENCES Position(position\_id),

FOREIGN KEY(department\_id) REFERENCES Department(department\_id),

FOREIGN KEY(work\_schedule\_id) REFERENCES WorkSchedule(work\_schedule\_id),

CONSTRAINT user\_login\_unique UNIQUE(login),

CONSTRAINT user\_login\_chk CHECK(LENGTH(login) BETWEEN 6 AND 60),

CONSTRAINT user\_name\_chk CHECK(name REGEXP '^([a-zA-Z-]{0,45}|[а-яА-ЯёЁ-]{0,45})$'),

CONSTRAINT user\_surname\_chk CHECK(surname REGEXP '^([a-zA-Z-]{0,45}|[а-яА-ЯёЁ-]{0,45})$'),

CONSTRAINT user\_patronymic\_chk CHECK(patronymic REGEXP '([a-zA-Z-]{0,45}|[а-яА-ЯёЁ-]{0,45})'),

CONSTRAINT user\_phone\_chk CHECK (phone\_num REGEXP '^([+]{1}375|80)(29|25|44|33)([0-9]{7})$'),

);

CREATE TABLE User\_BusinessTrip(

user\_business\_trip\_id INTEGER NOT NULL AUTO\_INCREMENT,

user\_id INTEGER NOT NULL,

business\_trip\_id INTEGER NOT NULL,

PRIMARY KEY(user\_business\_trip\_id),

FOREIGN KEY(user\_id) REFERENCES User(user\_id),

FOREIGN KEY(business\_trip\_id) REFERENCES BusinessTrip(business\_trip\_id)

);

CREATE TABLE User\_Notification(

user\_notification\_id INTEGER NOT NULL AUTO\_INCREMENT,

user\_id INTEGER NOT NULL,

notification\_id INTEGER NOT NULL,

PRIMARY KEY(user\_notification\_id),

FOREIGN KEY(user\_id) REFERENCES User(user\_id),

FOREIGN KEY(notification\_id) REFERENCES Notification(notification\_id)

);