2. Моделирование базы данных

2.1 Создание концептуальной схемы

Концептуальная модель базы данных это некая наглядная диаграмма, нарисованная в принятых обозначениях и подробно показывающая связь между объектами и их характеристиками. Создается концептуальная модель для дальнейшего проектирования базы данных и перевод ее, например, в реляционную базу данных. На концептуальной модели в визуально удобном виде прописываются связи между объектами данных и их характеристиками.

**Объект или сущность -** Это фактическая вещь или объект (для людей) за которой пользователь (заказчик) хочет наблюдать.

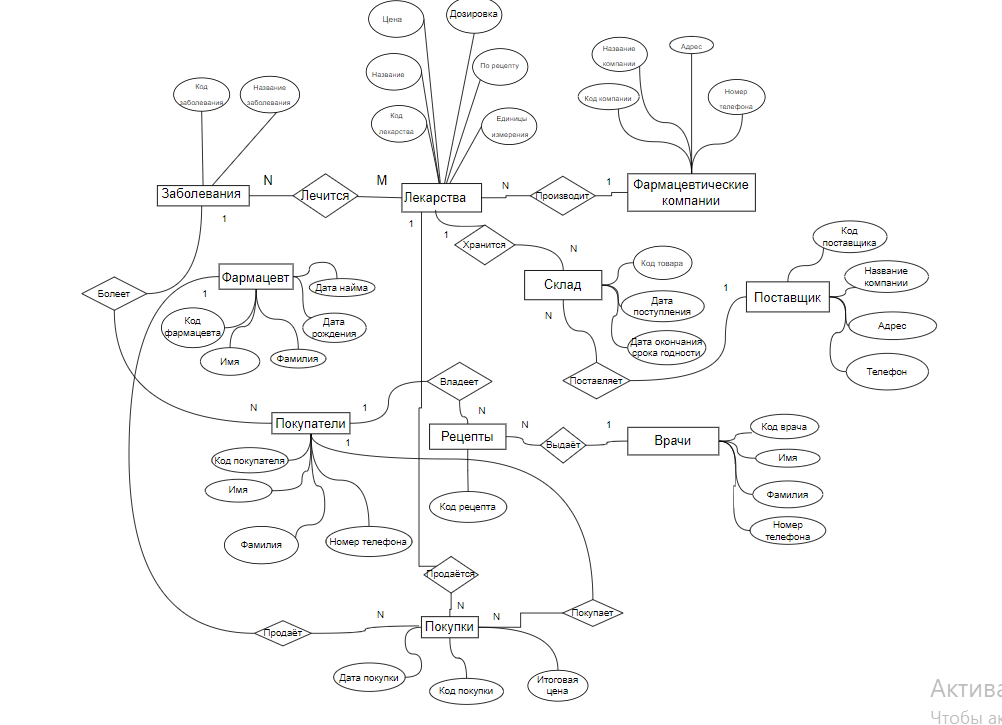
**Атрибут** - это характеристика объекта, соответствующая его сущности.

Третье понятие в проектировании концептуальной базы данных это связь или отношения между объектами.

Определим сущности, которые будут использоваться для создания концептуальной схемы:

* Поставщики
* Лекарства
* Заболевания
* Фармацевтические компании
* Склад
* Фармацевты
* Покупки
* Покупатели
* Рецепты
* Врачи

На основании определённых выше сущностей создадим концептуальную схему разрабатываемой базы данных:



Следующим шагом после создания концептуальной схемы базы данных будет разработка логической схемы базы данных

2.2 Разработка физической схемы базы данных

**На этапе логического проектирования разрабатывается логическая схема, определяются сущности (таблицы), атрибуты (столбцы таблицы), а также указывается вид связей между таблицами.**Логическая модель данных является универсальной и никак не связана с конкретной реализацией СУБД.

Различают два вида сущностей: сильные и слабые.

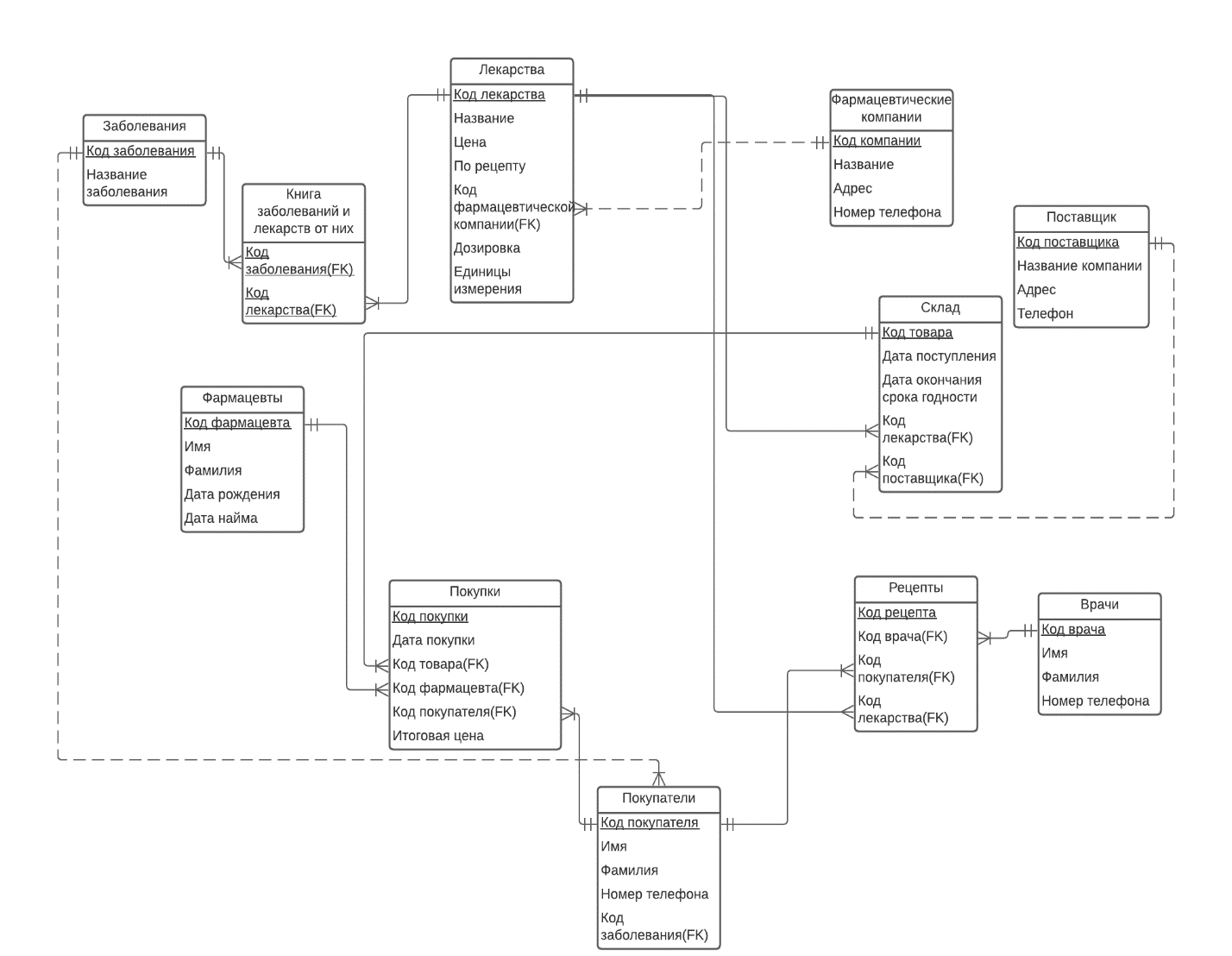
Сильные (стержневые сущности) - существуют объективно, и их существование не зависит от какой-либо другой сущности.

Слабая сущность - связана со стержневой сущностью и экземпляр слабой сущности не может существовать, если не существует экземпляр стержневой сущности.

Характеристикой - называется слабая сущность, являющаяся дополнением стержневой сущности.

Ассоциативная сущность - это сущность которая служит для обеспечения взаимодействия между двумя и более другими сущностями.

На основании концептуальной схемы и вышесказанного создадим логическую схему базы данных.



1. Сущность фармацевтические компании описывает данные о фармацевтических компаниях, которые производят лекарства

|  |  |
| --- | --- |
| Название характеристики | Тип характеристики |
| **Код компании** | **INTEGER** |
| Номер телефона | VARCHAR |
| Название | VARCHAR |
| Адрес | VARCHAR |

1. Сущность лекарства содержит данные о лекарствах.

|  |  |
| --- | --- |
| Название характеристики | Тип характеристики |
| **Код лекарства** | **INTEGER** |
| Название | VARCHAR |
| Цена | INTEGER |
| По рецепту | TINYINT(1) |
| Код фармацевтической компании | INTEGER |
| Дозировка | INTEGER |
| Единицы измерения | VARCHAR |

1. Сущность заболевания содержит данные о заболеваниях

|  |  |
| --- | --- |
| Название характеристики | Тип характеристики |
| **Код заболевания** | **INTEGER** |
| Номер название заболевания | VARCHAR |

1. Сущность Книга заболеваний и лекарств от них осуществляет Many-To-Many связь между заболеваниями и лекарствами

|  |  |
| --- | --- |
| Название характеристики | Тип характеристики |
| **Код заболевания** | **INTEGER** |
| **Код лекарства** | **INTEGER** |

1. Сущность Фармацевты содержит данные о фармацевтах, работающих в аптеке

|  |  |
| --- | --- |
| Название характеристики | Тип характеристики |
| **Код фармацевта** | **INTEGER** |
| Имя | VARCHAR |
| Фамилия | VARCHAR |
| Дата рождения | DATE |
| Дата найма | DATE |

1. Сущность поставщик содержит данные о поставщиках

|  |  |
| --- | --- |
| Название характеристики | Тип характеристики |
| **Код поставщика** | **INTEGER** |
| Телефон | VARCHAR |
| Название | VARCHAR |
| Адрес | VARCHAR |

1. Сущность склад хранит содержит информацию о товарах, хранимых на складе

|  |  |
| --- | --- |
| Название характеристики | Тип характеристики |
| **Код товара** | **INTEGER** |
| Код лекарства | INTEGER |
| Код поставщика | INTEGER |
| Дата поступления | DATE |
| Дата окончания срока годности | DATE |

1. Сущность врачи содержит информацию о врачах

|  |  |
| --- | --- |
| Название характеристики | Тип характеристики |
| **Код врача** | **INTEGER** |
| Номер телефона | VARCHAR |
| Имя | VARCHAR |
| Фамилия | VARCHAR |

1. Сущность покупатели содержит данные о покупателях в аптеке

|  |  |
| --- | --- |
| Название характеристики | Тип характеристики |
| **Код покупателя** | **INTEGER** |
| Имя | VARCHAR |
| Фамилия | VARCHAR |
| Номер телефона | VARCHAR |
| Код заболевания | INTEGER |

1. Сущность рецепты содержит информацию о рецептах выданных покупателям врачами

|  |  |
| --- | --- |
| Название характеристики | Тип характеристики |
| **Код рецепта** | **INTEGER** |
| Код врача | INTEGER |
| Код покупателя | INTEGER |
| Код лекарства | INTEGER |

1. Сущность покупки содержит информацию об покупках, осуществлённых в аптеке

|  |  |
| --- | --- |
| Название характеристики | Тип характеристики |
| **Код покупки** | **INTEGER** |
| Код фармацевта | INTEGER |
| Код покупателя | INTEGER |
| Код товара | INTEGER |
| Итоговая цена | INTEGER |
| Дата покупки | DATETIME |

Проверим полученную логическую схему на соответствие первым трем нормальным формам [5].

**Первая нормальная форма** (1НФ) говорит, что каждый атрибут отношения должен хранить атомарное значение, каждое отношение (строка в таблице) должно содержать одинаковое количество атрибутов (столбцов), т.е.:

* запрещает повторяющиеся столбцы (содержащие одинаковую по смыслу информацию)
* запрещает множественные столбцы (содержащие значения типа списка и т.п.)
* требует определить первичный ключ для таблицы, то есть тот столбец или комбинацию столбцов, которые однозначно определяют каждую строку.

**Вторая нормальная форма** (2НФ) говорит, что отношение находится во второй нормальной форме, если оно находится в 1НФ, и при этом все не ключевые атрибуты зависят только от первичного ключа, т.е.:

* Вторая нормальная форма требует, чтобы не ключевые столбцы таблиц зависели от первичного ключа в целом, но не от его части.
* Если таблица находится в первой нормальной форме и первичный ключ у нее состоит из одного столбца, то она автоматически находится и во второй нормальной форме [2].

Отношение находится в **третьей нормальной форме** (3НФ), если оно находится во второй нормальной форме и каждый не ключевой атрибут зависит только от первичного ключа и не зависят друг от друга [4].

Переменная отношения находится в BCNF тогда и только тогда, когда каждая её нетривиальная и неприводимая слева функциональная зависимость имеет в качестве своего детерминанта некоторый потенциальный ключ.

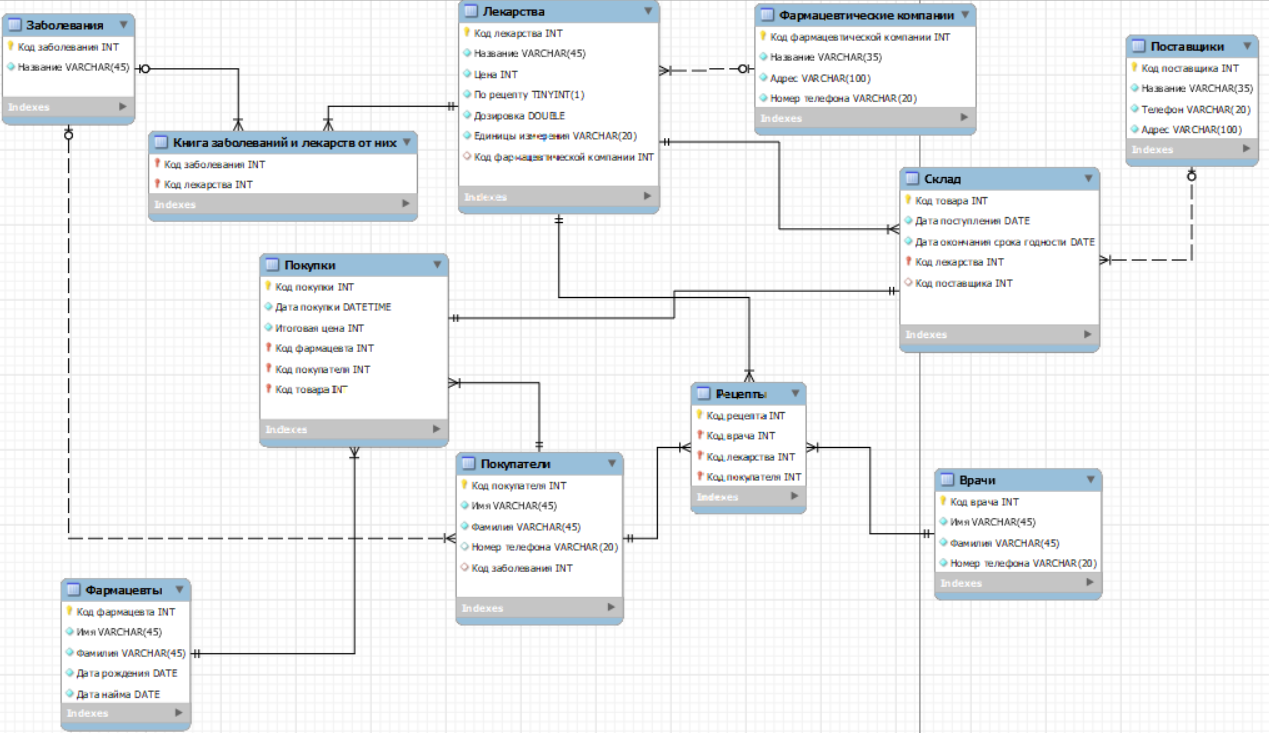
Так как все вышеперечисленные условия соблюдены, то разрабатываемая база данных находится в нормальной форме Бойса-Кодда.

Следующим шагом после создания логической схемы будет создание физической схемы разрабатываемой базы данных.

2.3 Создание физической схемы базы данных

**Для построения физической модели данных необходимо указать физическую структуру базы данных, типы атрибутов и связи. Основное отличие физической модели от логической модели в том, что физическая модель привязана к конкретной СУБД**, т.е. одной логической модели может соответствовать несколько различных физических моделей. В физической модели важно описать всю информацию о конкретных таблицах, колонках, индексах (например, явно указывается тип данных каждого атрибута).

Для разработки физической схемы базы данных будет использован MySQL Workbench 8.0, а именно Data Modeler.



3. Реализация базы данных

3.1 Создание базы данных

После создания физической схемы следует перейти к реализации базы данных .

Для того чтобы создать базу данных pharmacy необходимо выполнить следующую SQL-команду:

CREATE DATABASE pharmacy;

После создания команда 'use pharmacy' будет обозначать то, что следующие за ней команды будут выполняться в созданной базе данных.

Следующим шагом будет создание таблиц. Для того чтобы создать таблицу pharmacists необходимо выполнить следующую SQL команду:

CREATE TABLE pharmacy.pharmacists (

id INTEGER UNSIGNED AUTO\_INCREMENT UNIQUE PRIMARY KEY NOT NULL,

firstname VARCHAR(45) NOT NULL,

lastname VARCHAR(45) NOT NULL,

date\_of\_birth DATE NOT NULL,

hire\_date DATE NOT NULL

);

Здесь:

* INTEGER – целочисленный тип данных;
* UNSIGNED – числа больше или равные нулю;
* AUTO\_INCREMENT – обозначение того, что данное поле будет автоматически увеличиваться при создании новой записи, без явного указания чему равно поле;
* UNIQUE – обозначение того, что все записи в таблице должны быть уникальны по данному полю;
* PRIMARY KEY – обозначение того, что данное поле будет являться первичным ключом в таблице;
* NOT NULL – обозначение того, что данное поле не может быть равно NULL;
* VARCHAR(m) – строка с переменной длинной(не дополняется до максимальной длины пробелами в отличии от CHAR(m)), где m-максимальная длина строки;
* DATE- тип данных, обозначающий дату.

Кроме того при создании таблиц будет использоваться механизм внешних ключей. Для того чтобы создать внешний ключ необходимо при создании таблицы после объявления поля, которое будет являться внешним ключом написать:

FOREIGN KEY (название\_поля\_которое\_будет\_являться\_ внешним\_ ключом) REFERENCES название\_таблицы (название\_поля\_на\_которое\_будет\_ссылаться\_внешний\_ключ) [ON DELETE {опция}] [ON UPDATE {опция}];

В качестве опций будут использованы:

* SET NULL – установить значение поля внешнего ключа в дочерней таблице равному NULL при удалении(обновлении) связанной записи в родительской таблице;
* CASCADE – удалять(изменять) запись в дочерней таблице при удалении(изменении) связанной записи в родительской таблице.

Следующим шагом после создания таблиц будет написание триггеров для этих таблиц.

3.2 Создание триггеров в базе данных

Триггер – это [хранимая процедура](https://dic.academic.ru/dic.nsf/ruwiki/91859) особого типа, которую пользователь не вызывает непосредственно, а исполнение которой обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определенном столбце заданной таблицы [реляционной базы данных](https://dic.academic.ru/dic.nsf/ruwiki/8508). Триггеры применяются для обеспечения целостности данных и реализации сложной [бизнес-логики](https://dic.academic.ru/dic.nsf/ruwiki/93540).

Для того чтобы создать триггер необходимо выполнить следующую SQL-команду:

CREATE TRIGGER название\_триггера {BEFORE|AFTER} {UPDATE|INSERT|DELETE} ON название\_таблицы FOR EACH ROW

BEGIN

….

END;

Здесь:

* BEFORE означает, что триггер сработает до выполнения основной команды.
* AFTER означает, что триггер сработает после выполнения основной команды

Перед выполнением этой команды необходимо поменять разделитель команд. Это делается с помощью DELIMITER // (разделитель меняется с ; на //).

Скрипт создания триггера, который проверяет корректность введённых данных при выполнении INSERT в таблице pharmacists:

DELIMITER //

CREATE TRIGGER insert\_pharmacist BEFORE INSERT ON pharmacy.pharmacists FOR EACH ROW

BEGIN

IF (NEW.hire\_date < NEW.date\_of\_birth) THEN

signal sqlstate '45000' set message\_text = 'birth date cant be less then hire date';

END IF;

IF (NEW.date\_of\_birth > current\_date() OR NEW.hire\_date > current\_date()) THEN

signal sqlstate '45000' set message\_text = 'birth date and hire date cant be on future';

END IF;

END; //

DELIMITER ;

Следующим шагом после создания триггеров будет заполнение таблиц данными.

3.3 Заполнение таблиц данными

Для добавления записей в таблицу используется SQL-команда INSERT INTO название\_таблицы (название\_колонки\_1,...,название\_колонки\_n) VALUES (значение\_1,…,значение\_n),…;

Для того чтобы добавить записи в таблицу pharmacists необходимо выполнить следующую команду:

INSERT INTO pharmacy.pharmacists (firstname,lastname,date\_of\_birth,hire\_date) VALUES ('Sergey','Sergeev','2000-01-01', current\_date()), ('Fedor','Fedorov','1995-08-03',current\_date()), ('Vladislav','Kuznetsov', '1997-05-30',current\_date()), ('Irina','Sidorova','1993-04-05',current\_date());

Следующим шагом после заполнения таблиц в базе данных будет разработка запросов к базе данных.

3.4 Разработка запросов к базе данных

Для создания запроса к базе данных используется оператор SELECT.

Например, для получения кодов просроченных товаров на складе, необходимо написать следующую SQL-команду:

SELECT id FROM pharmacy.stock WHERE expiry\_date<current\_date();

Следующим шагом после разработки запросов к базе данных будет создание хранимых процедур.

3.5 Создание хранимых процедур

Хранимая процедура - объект [базы данных](https://dic.academic.ru/dic.nsf/ruwiki/7390), представляющий собой набор [SQL](https://dic.academic.ru/dic.nsf/ruwiki/4818)-инструкций, который компилируется один раз и хранится на сервере. Хранимые процедуры очень похожи на обыкновенные процедуры языков высокого уровня, у них могут быть входные и выходные параметры и локальные переменные, в них могут производиться числовые вычисления и операции над символьными данными, результаты которых могут присваиваться переменным и параметрам. В хранимых процедурах могут выполняться стандартные операции с базами данных. Кроме того, в хранимых процедурах возможны циклы и ветвления, то есть в них могут использоваться инструкции управления процессом исполнения.

Для создания хранимой процедуры необходимо использовать команду:

CREATE PROCEDURE название\_процедуры(параметры)

BEGIN

….

END;

Например, для того чтобы создать процедуру, удаляющую все просроченные товары со склада необходимо выполнить следующие команды:

DELIMITER //

CREATE PROCEDURE pharmacy.delete\_all\_expired()

BEGIN

DELETE FROM pharmacy.stock WHERE current\_date()>expiry\_date;

END;//

DELIMITER ;

Чтобы вызвать процедуру необходимо выполнить команду:

CALL название\_процедуры(параметры);