

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
1 ПОСТРОЕНИЕ ИНФОЛОГИЧЕСКОЙ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ	5
1.1 Анализ предметной области и выявление необходимого набора сущностей.....	5
1.2 Обоснование требуемого набора атрибутов для каждой сущности и выделение идентифицирующих атрибутов	6
1.3 Определение связей между объектами	7
1.4 Описание полученной модели на языке инфологического проектирования.....	8
2 ПОСТРОЕНИЕ СХЕМЫ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ	9
2.1 Построение набора необходимых отношений базы данных	9
2.2 Задание первичных и внешних ключей определенных отношений.....	9
2.3 Третья нормальная форма	10
2.4 Определение ограничений целостности для внешних ключей отношений и для отношений в целом.....	11
2.5 Графическое представление связей между внешними и первичными ключами.....	11
3 СОЗДАНИЕ СПРОЕКТИРОВАННОЙ БАЗЫ ДАННЫХ	12
4 ЗАПИСЬ ВЫРАЖЕНИЙ УКАЗАННЫХ В ВАРИАНТЕ ЗАДАНИЯ	
ТИПОВ ЗАПРОСОВ НА ЯЗЫКЕ SQL	16
5 ВЫБОР И ОСНОВАНИЕ СРЕДСТВ РАЗРАБОТКИ ПРИЛОЖЕНИЯ	23
6 РЕАЛИЗАЦИЯ ЗАКОНЧЕННОГО ПРИЛОЖЕНИЯ, РАБОТАЮЩЕГО С СОЗДАННОЙ БАЗОЙ ДАННЫХ	24
6.1 Разработка и построение интерфейса главной и рабочих форм.....	24
6.2 Построение главного меню и кнопок панели инструментов.....	24
6.3 Выполнение программного кода на языке Java	25
ЗАКЛЮЧЕНИЕ	27
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	28
ПРИЛОЖЕНИЕ А	29
ПРИЛОЖЕНИЕ Б.....	30
ПРИЛОЖЕНИЕ В	31
ПРИЛОЖЕНИЕ Г.....	32
ПРИЛОЖЕНИЕ Д	34

					ЯАС.1610574.ПЗ		
Изм	Лист	№ докум.	Подпись	Дата			
Разраб.		Яблонский А.С			Разработка информационной системы студенческого общежития	Лит.	Лист
Провер.		Роголев В.С.					Листов
Реценз.							3
Н. Контр.							34
Утверд.						Учреждение образования «Полоцкий государственный университет», группа 16-ИТ-3	

ВВЕДЕНИЕ

Актуальностью данной работы является увеличение спроса студентов на проживание в общежитиях. Приток студентов с каждым годом увеличивается, потому что в современном мире наличие высшего образования стало необходимым, молодое поколение стремится к саморазвитию и высокому уровню жизни.

Тема курсового проекта – проектирование реляционной базы данных информационной системы студенческого общежития и разработка приложения для взаимодействия с созданной базой данных.

Объектом исследования данной курсовой работы является студенческое общежитие. Студенческое общежитие является важной составляющей любого учебного заведения. Данное предприятие должно обеспечивать студентов, проживающих в общежитии, всем необходимым для проживания в нем.

Для обеспечения функционала, а также для удобства пользования информационной системой необходимо разработать приложение, которое позволит добавлять, удалять, редактировать договоры, клиентов и просматривать информацию о них. Приложение должно быть простым в использовании, которым могли бы пользоваться даже неквалифицированные сотрудники.

Аналогов данной программы не мало. Создаваемая информационная система не привязывается к существующему общежитию, и создается на основе не существующего общежития.

Для создания информационной базы данных будет использоваться СУДБ Oracle. Для создания приложения – среда JavaFX.

					ЯАС.1610574.ПЗ	Лист
						4
Изм.	Лист	№ докум.	Подпись	Дата		

1 ПОСТРОЕНИЕ ИНФОЛОГИЧЕСКОЙ КОНЦЕПТУАЛЬНОЙ МОДЕЛИ

1.1 Анализ предметной области и выявление необходимого набора сущностей

В ходе анализа знаний и разработке базы данных были выявлены следующие основные сущности:

Сущность Сотрудник представляет собой сотрудника общежития, который будет закреплен за определенной комнатой, так же у сотрудника должна быть должность.

Сущность Сотрудник университета это проживающий в общежитии сотрудник.

Сущность Факультет описывает факультет университета.

Сущность Группа описывает группы университета и содержит данные о факультете, к которому она относится.

Сущность Инвентарь характеризует инвентарь закрепленный за комнатой.

Сущность Нежилые комнаты это комнаты, в которых работает персонал общежития, такие как кладовая и т.п..

Сущность Должность описывает должность сотрудника общежития.

Сущность Имущество описывает имущество, которое имеется в общежитии, которое впоследствии может быть закреплено за проживающим.

Сущность Комната представляет из себя данные о комнате и блоке в котором она находится.

Сущность Студент содержит дополнительную информацию о проживающей категории студент.

Сущность Житель полная информация о проживающем в общежития, так же эта информация общая как для студента так и для сотрудника.

Сущность Блок содержит блоки из которых состоит общежитие.

При начальном анализе также полезно строить диаграмму потоков данных (*англ.* Data Flow Diagram, DFD-diagram). Такая диаграмма описывает внешние по отношению к системе источники и адресаты данных, логические функции, потоки данных и хранилища данных, к которым осуществляется доступ. Диаграмма потоков данных для предметной области представлена на рисунке 1.1.

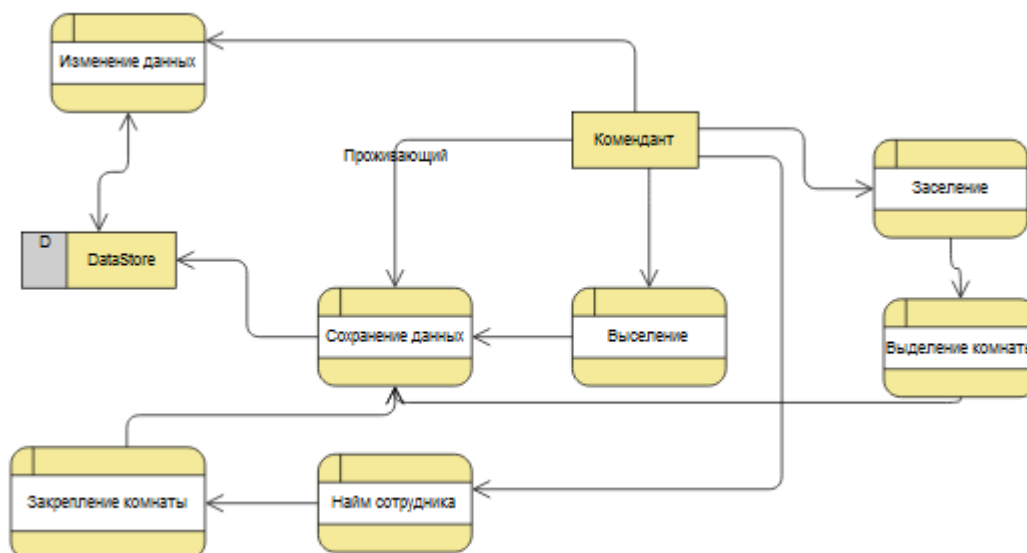


Рисунок 1.1 – Диаграмма потоков данных

1.2 Обоснование требуемого набора атрибутов для каждой сущности и выделение идентифицирующих атрибутов

Для построения инфологической концептуальной модели необходимо для каждой сущности, выявленной в предыдущем пункте, определить требуемый набор атрибутов. Атрибутом является поименованная характеристика сущности. Его наименование должно быть уникальным для конкретного типа сущности, но может быть одинаковым для различного типа сущностей. Атрибуты используются для определения того, какая информация должна быть собрана о сущности.

Ниже представлены сущности и определенные для них атрибуты, а также ключи. Имена сущностей и атрибутов указываются, как они будут определены в созданной базе данных, в скобках указывается перевод либо описание:

1. employee:
 - **id** (код сотрудника) ;
 - fio (ФИО);
 - post_id (занимаемая должность);
 - room_id (закрепленная комната);
2. faculty:
 - **id** (код факультета);
 - name (наименование факультета);
3. groups
 - **id** (код группы);
 - name (название группы);
 - faculty_id (факультет к которому относится)

4. inventory:
 - **id** (код инвенторя);
 - name (название);
5. other_rooms:
 - **id** (код нежилой комнаты);
 - name (название);
6. payment (оплата):
 - **id** (код оплаты);
 - tenant_id (проживающий);
 - date_payment (дата оплаты);
7. post (должность):
 - **id** (коддолжности);
 - name (название);
8. property (имущество):
 - **id** (код имущества);
 - name (название);
9. room (комната):
 - **id** (код комнаты);
 - square (площадь);
 - count_bets (количество мест);
 - busy_bets (количество занятых мест);
 - unit_id (блок);
10. student (студент):
 - **Tenant_id** (код проживающего);
 - Group_id (код группы);
11. tenant (проживающий):
 - **Number_contract**(номер контракта);
 - birthday (дата рождения);
 - check_date (дата заселения);
 - eviction_date (дата выселения);
 - room_id (комната);
 - fio (ФИО);
12. unit (блок):
 - **id** (номер блока)

1.3 Определение связей между объектами

Кроме атрибутов каждой сущности модель данных должна определять связи между сущностями. На концептуальном уровне связи представляют собой простые ассоциации между сущностями.

					ЯАС.1610574.ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

Связь – это ассоциирование двух или более сущностей. Если бы назначением базы данных было только хранение отдельных, не связанных между собой данных, то ее структура могла бы быть очень простой. Однако, одно из основных требований к организации базы данных – это обеспечение возможности отыскания одних сущностей по значениям других, для чего необходимо установить между ними определенные связи. А так как в реальных базах данных нередко содержатся десятки или даже сотни сущностей, то между ними может быть установлено великое множество связей. Наличие такого множества связей и определяет сложность инфологических моделей.

Для реализации информационной системы общежития необходимо установить все связи между объектами. А именно, нужно рассмотреть всю информационную систему общежития в совокупности и определить взаимное влияние объектов, составляющих систему.

Этот процесс представлен в приложении А.

1.4 Описание полученной модели на языке инфологического проектирования

Проектирование инфологической модели предметной области – частично формализованное описание объектов предметной области в терминах некоторой семантической модели, например, в терминах ER-модели (*англ.* entity-relationship model). По правилам построения ER-диаграмм в нотации Питера Чена, сущности изображаются прямоугольниками, их атрибуты – овалами, отношения – ромбами. Связи между объектами изображаются линиями [2].

На основе проведенного проектирования, в частности на основе инфологической схемы, получим ER-диаграмму базы данных городской телефонной сети, представленную в Приложении Б.

					ЯАС.1610574.ПЗ	Лист
						8
Изм.	Лист	№ докум.	Подпись	Дата		

2 ПОСТРОЕНИЕ СХЕМЫ РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ

2.1 Построение набора необходимых отношений базы данных

Чтобы построить схему реляционной базы данных необходимо определить совокупность отношений, которые составляют базу данных. Эта совокупность отношений будет содержать всю информацию, которая должна храниться в базе данных.

В предыдущем пункте мы создали инфологическую концептуальную модель базы данных городской телефонной сети, построенной с помощью языка «Таблицы-связи». На основе полученной концептуальной модели можно определить набор необходимых отношений базы данных. На рисунке 2.1 представлены отношения для базы данных городской телефонной сети.

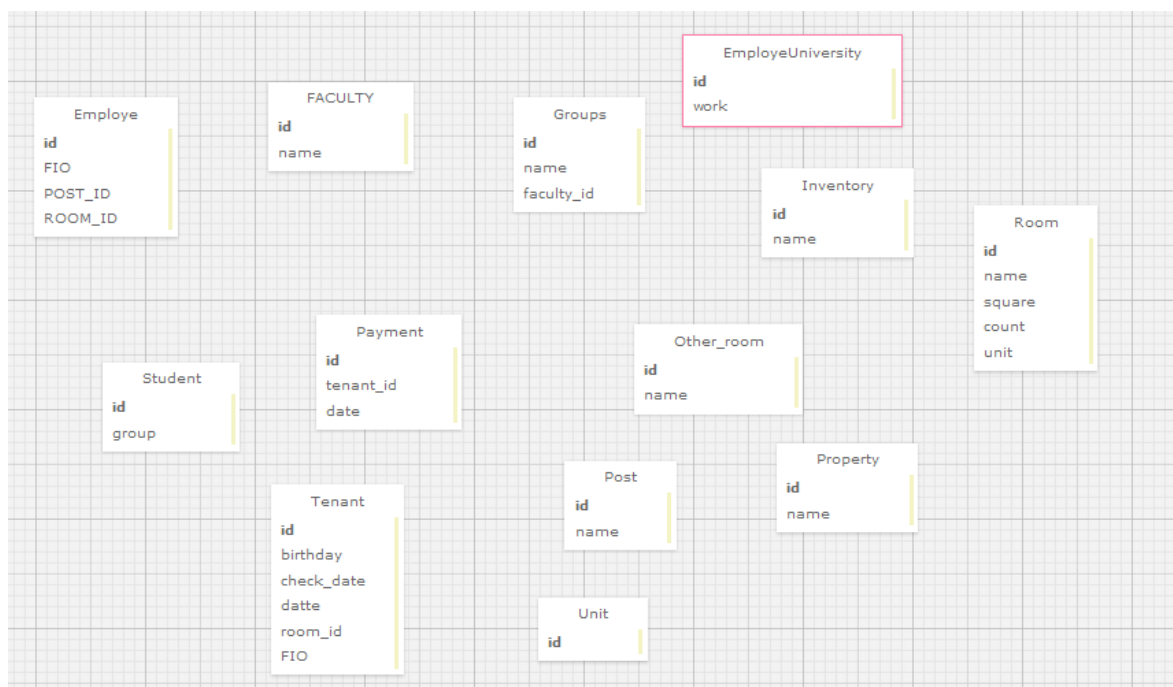


Рисунок 2.1 – Набор необходимых отношений базы данных

2.2 Задание первичных и внешних ключей определенных отношений

В реляционной базе данных каждому объекту и сущности реального мира соответствуют кортежи отношений. И любое отношение должно обладать первичным ключом. Ключ – это минимальный набор атрибутов, по значениям которых можно однозначно найти требуемый экземпляр сущности. Минимальность означает, что исключение из набора любого атрибута не позволяет идентифицировать сущность по оставшимся. Каждое отношение должно обладать хотя бы одним ключом. В таблице 2.1 определены первичные и внешние ключи для отношений.

Таблица 2.1 – Первичные и внешние ключи отношений.

№ п/п	Название таблицы	Первичный ключ	Внешние ключи
1	employee	id	Post_id, room_id
2	faculty	id	
3	groups	id	faculty_id
4	inventory	id	
5	room	id	Unit_id
6	unit	id	
7	Other_toom	id	
8	tenant	id	Room_id
9	post	id	
10	property	id	

В дальнейшем построении схемы реляционной базы данных ключи будут служить для организации связей между отношениями.

Таким образом, ненормализованная схема базы данных представлена в приложении В, на рисунке В1.

2.3 Третья нормальная форма

Процесс преобразования базы данных к виду, отвечающему нормальным формам, называется нормализацией. Нормализация предназначена для приведения структуры базы данных к виду, обеспечивающему минимальную избыточность, то есть нормализация не имеет целью уменьшение или увеличение производительности работы или же уменьшение, или увеличение объёма БД. Конечной целью нормализации является уменьшение потенциальной противоречивости хранимой в БД информации.

Для реляционных баз данных необходимо, чтобы все отношения базы данных обязательно находились в 1НФ. Нормальные формы более высокого порядка могут использоваться разработчиками по своему усмотрению. Однако грамотный специалист стремится к тому, чтобы довести уровень нормализации базы данных хотя бы до 3НФ, тем самым, исключив из базы данных избыточность и аномалии обновления.

Определение 3НФ – неключевые атрибуты не должны определять другие неключевые атрибуты.

2.4 Определение ограничений целостности для внешних ключей отношений и для отношений в целом

Ограничение целостности отношений заключается в том, что в любом отношении должны отсутствовать записи с одним и тем же значением первичного ключа. Конкретно требование состоит в том, что любая запись любого отношения должна быть отличной от любой другой записи этого отношения. Это требование автоматически удовлетворяется, если в системе не нарушаются базовые свойства отношений.

Ограничение целостности для внешних ключей состоит в том, что значение внешнего ключа должно быть равным значению первичного ключа цели; либо быть полностью неопределенным, т.е. каждое значение атрибута, участвующего во внешнем ключе должно быть неопределенным.

Условиями целостности называется набор правил, используемых для поддержания допустимых межтабличных связей и запрета на случайное изменение или удаление связанных данных. Следует устанавливать целостность данных только при выполнении следующих условий: связываемое поле из главной таблицы является полем первичного ключа и имеет уникальный индекс, связанные поля имеют один и тот же тип данных.

Для автоматического обновления связанных полей (удаления записей) при обновлении (удалении) в главной таблице, следует устанавливать обеспечение целостности данных и каскадное обновление связанных полей (каскадное удаление связанных записей).

Ограничение целостности, накладываемые на разрабатываемую систему:

- ключевое поле отношения должно быть уникальным;
- внешний ключ должен быть повторяющимся, то есть соответствовать уникальному ключу в своем отношении.

Для удовлетворения требования ограничения целостности для внешних ключей отношений и для отношений в целом необходимо, чтобы выполнялось соответствие между типами вводимых данных и типами столбцов в таблицах, а так же чтобы были заполнены все обязательные поля в таблицах, т.е. те поля которые не могут содержать значения NULL.

2.5 Графическое представление связей между внешними и первичными ключами

По результатам нормализации, определении первичных и внешних ключей, связей между сущностями, была получена схема реляционной базы данных, представленная в приложении В, на рисунке В.1. На ней изображаются все отношения базы данных, а также связей между внешними и первичными ключами.

					ЯАС.1610574.ПЗ	Лист
						11
Изм.	Лист	№ докум.	Подпись	Дата		

3 СОЗДАНИЕ СПРОЕКТИРОВАННОЙ БАЗЫ ДАННЫХ

Для реализации спроектированной базы данных была выбрана система управления базами данных Oracle. Это обусловлено тем, что, данная СУБД имеет большую функциональность, множество средств для поддержки и работы с ней, развитую инфраструктуру для интеграции баз данных в пользовательские приложения. Большое количество дополнений - несмотря на огромное количество встроенных функций, существует очень много дополнений, позволяющих разрабатывать данные для этой СУБД и управлять ими.

В создаваемой базе данных будут использоваться следующие типы данных [3]:

1. INT – Целочисленный тип. Размер – 4 байта
2. VARCHAR – Строковый тип
3. FLOAT – Вещественный тип
4. BOOLEAN – Целочисленный тип размером в 1 байт.

Определяется как логический тип.

5. DATETIME – Тип, определяющий дату и время.
6. DATE – Тип, определяющий дату.

Опишем все таблицы, которые будут созданы в базе данных.

Таблица *employee* содержит сотрудников. Ее структура приведена в таблице 3.1.

Таблица 3.1 – Характеристика атрибутов таблицы *employee*.

Имя атрибута	Тип	Описание
id	INT	Идентификатор сотрудника. Ключевой атрибут
fio	VARCHAR()	ФИО
Post_id	INT	Занимаемая должность
Room_id	INT	Комната за которой закреплен

Таблица *employee_university* содержит всех проживающих сотрудников университета. Ее структура приведена в таблице 3.2.

Таблица 3.2 – Характеристика атрибутов таблицы *employee_university*.

Имя атрибута	Тип	Описание
<u>Tenant_id</u>	INT	Идентификатор проживающего
work	VARCHAR()	Место работы

Таблица *faculty* характеризует факультет. Ее структура приведена в таблице 3.3.

Таблица 3.3 – Характеристика атрибутов таблицы *faculty*.

Имя атрибута	Тип	Описание
id	INT	Идентификатор факультета. Ключевой атрибут
name	VARCHAR()	Название

Таблица *groups* характеризует группы. Ее структура приведена в таблице 3.4.

Таблица 3.4 – Характеристика атрибутов таблицы *groups*.

Имя атрибута	Тип	Описание
<u>id</u>	INT	Идентификатор группы. Ключевой атрибут
name	VARCHAR()	Название
Faculty_id	INT	Факультет

Таблица *inventory* характеризует инвентарь общежития. Ее структура приведена в таблице 3.5.

Таблица 3.5 – Характеристика атрибутов таблицы *inventory*.

Имя атрибута	Тип	Описание
<u>id</u>	INT	Идентификатор предмета. Ключевой атрибут
name	VARCHAR()	Название

Таблица *other_room* содержит информацию о нежилых комнатах. Ее структура приведена в таблице 3.6.

Таблица 3.6 – Характеристика атрибутов таблицы *other_room*.

Имя атрибута	Тип	Описание
<u>id</u>	INT	Идентификатор комнаты. Ключевой атрибут
name	VARCHAR()	Название

Таблица *payment* содержит информацию обо всех оплатах за проживание. Ее структура приведена в таблице 3.7.

Таблица 3.7 – Характеристика атрибутов таблицы *payment*.

Имя атрибута	Тип	Описание
<u>id</u>	INT	Идентификатор оплаты. Ключевой атрибут
Tenant_id	INT	Проживающий, который оплатил проживание
Date_payment	DATE	Дата оплаты

Таблица *post* содержит основную информацию о должностях сотрудников. Ее структура приведена в таблице 3.8.

Таблица 3.8 – Характеристика атрибутов таблицы *post*.

Имя атрибута	Тип	Описание
<u>id</u>	INT	Идентификатор должности. Ключевой атрибут
name	VARCHAR()	Название

Таблица *property* содержит перечень всех имуществ. Ее структура приведена в таблице 3.9.

Таблица 3.9 – Характеристика атрибутов таблицы *property*.

Имя атрибута	Тип	Описание
<u>id</u>	INT	Идентификатор имущества. Ключевой атрибут
name	VARCHAR()	Название имущества

Таблица *room* описывает комнату для проживания. Ее структура приведена в таблице 3.10.

Таблица 3.10 – Характеристика атрибутов таблицы *room*.

Имя атрибута	Тип	Описание
<u>id</u>	INT	Идентификатор комнаты. Ключевой атрибут
square	FLOAT	Площадь
Count_bets	INT	Количество спальных мест
Busy_bets	INT	Занято мест
Unit_id	INT	Номер блока

Таблица *student* содержит перечень всех студентов проживающих в общежитии, расположенных в городе. Ее структура приведена в таблице 3.11.

Таблица 3.11 – Характеристика атрибутов таблицы *student*.

Имя атрибута	Тип	Описание
<u>Tenant_id</u>	INT	Идентификатор студента. Ключевой атрибут
Group_id	INT	Идентификатор группы

Таблица *tenant* содержит информацию о жильце. Ее структура приведена в таблице 3.12.

Таблица 3.12 – Характеристика атрибутов таблицы *tenant*.

Имя атрибута	Тип	Описание
<u>Number_contract</u>	INT	Идентификатор жильца. Ключевой атрибут
birthday	DATE	День рождения
Check_date	DATE	Дата заселения
Eviction_date	DATE	Дата выселения
Room_id	INT	Идентификатор комнаты
fio	VARCHAR()	ФИО

Таблица *unit* описывает блок. Ее структура приведена в таблице 3.13.

Таблица 3.13 – Характеристика атрибутов таблицы *unit*.

Имя атрибута	Тип	Описание
id	INT	Номер блока

Таблица *room_inventory* содержит связь между комнатой и инвентарём в ней. Ее структура приведена в таблице 3.14.

Таблица 3.14 – Характеристика атрибутов таблицы *room_inventory*.

Имя атрибута	Тип	Описание
<u>Room_id</u>	INT	Идентификатор комнаты
<u>Inventory_id</u>	INT	Идентификатор инвентаря

Таблица *property_tenant* описывает связь между имуществом и жильцом. Ее структура приведена в таблице 3.15.

Таблица 3.15 – Характеристика атрибутов таблицы *property_tenant*.

Имя атрибута	Тип	Описание
<u>Property_id</u>	INT	Идентификатор имущества
<u>Tenant_id</u>	INT	Идентификатор жильца

4 ЗАПИСЬ ВЫРАЖЕНИЙ УКАЗАННЫХ В ВАРИАНТЕ ЗАДАНИЯ ТИПОВ ЗАПРОСОВ НА ЯЗЫКЕ SQL

Класс, выполняющий взаимодействие с жителями представлен в листинге 4.1.

Листинг 4.1 – Класс TenantController

```
1: public class TenantController extends TabController {
2:
3:     private TableView table;
4:
5:     public TenantController(TableView table) {
6:         super(table);
7:         this.table = table;
8:
9:         ObservableList<TableColumn> columns =
            table.getColumns();
10:        columns.get(0).setCellValueFactory(new
            PropertyValueFactory<Tenant, String>("FIO"));
11:        columns.get(1).setCellValueFactory(new
            PropertyValueFactory<Tenant, String>("unit"));
12:        columns.get(2).setCellValueFactory(new
            PropertyValueFactory<Tenant, String>("type"));
13:    }
14:
15:    @Override
16:    public void add() {
17:        try {
18:            Stage stage =
                modalWindow("FXML/tenant/add.fxml", "Добавить жильца");
19:            Object data = stage.getUserData();
20:            if (data != null && (Integer) data ==
                Window.CLICK_ADD) updateView();
21:        } catch (Exception ex) {
22:            AlertWindow.errorAlert(ex.getMessage());
23:            ex.printStackTrace();
24:        }
25:    }
26:
27:    @Override
28:    public void remove() {
29:    }
30:
31:
32:    @Override
33:    public void updateView() {
34:        TenantMapper tenantMapper =
            ctx.getBean("tenantMapper", TenantMapper.class);
35:        ObservableList<EmployeeTenant> employees =
            FXCollections.observableArrayList(tenantMapper.getEmployeeTenant());
```

					ЯАС.1610574.ПЗ	Лист
						16
Изм.	Лист	№ докум.	Подпись	Дата		

Продолжение листинга 4.1

```

36:         ObservableList<Student>          students          =
FXCollections.observableArrayList(tenantMapper.getStudent
());
37:         ObservableList                    list              =
FXCollections.observableArrayList();
38:         list.addAll(employees);
39:         list.addAll(students);
40:         table.setItems(list);
41:     }
42:
43:     @Override
44:     public void update() {
45:         try {
46:             Object                      obj                  =
table.getSelectionModel().getSelectedItem();
47:             if (obj == null) {
48:                 AlertWindow.errorAlert("Нет  выбранного
элемента");
49:                 return;
50:             }
51:             Stage update = new Stage();
52:             LoadFXML loader  = ctx.getBean("loader",
LoadFXML.class);
53:             UpdateController          controller            =
(UpdateController)
loader.loadModal("fxml/tenant/update.fxml",
"Редактировать жильца", update, stage);
54:             controller.setTenant((Tenant) obj);
55:             update.showAndWait();
56:             Object data = update.getUserData();
57:             if (data != null && (Integer) data ==
Window.CLICK_EDIT) updateView();
58:         } catch (Exception ex) {
59:             AlertWindow.errorAlert(ex.getMessage());
60:             ex.printStackTrace();
61:         }
62:     }
63: }

```

Класс, выполняющий выборку факультетов представлен в листинге 4.2.

Листинг 4.2 – Класс FacultyController

```

1:     public class FacultyController extends TabController
{
2:
3:     private TableView table;
4:
5:     public FacultyController(TableView table) {
6:         super(table);
7:         this.table = table;

```

Продолжение листинга 4.2

```

8:         ObservableList<TableColumn> columnsFaculty =
table.getColumns();
9:         columnsFaculty.get(0).setCellValueFactory(new
PropertyValueFactory<Faculty, Long>("id"));
10:        columnsFaculty.get(1).setCellValueFactory(new
PropertyValueFactory<Faculty, String>("name"));
11:    }
12:
13:    @Override
14:    public void add() {
15:        try {
16:            Stage stage =
modalWindow("FXML/faculty/add.fxml", "Добавить
факультет");
17:            Object data = stage.getUserData();
18:            if (data != null && (Integer) data ==
Window.CLICK_ADD) updateView();
19:        } catch (Exception ex) {
20:            AlertWindow.errorAlert(ex.getMessage());
21:            ex.printStackTrace();
22:        }
23:    }
24:
25:    @Override
26:    public void remove() {
27:
28:    }
29:
30:    @Override
31:    public void update() {
32:        try{
33:            Object obj =
table.getSelectionModel().getSelectedItem();
34:            if (obj == null) {
35:                AlertWindow.errorAlert("Нет выбранного
элемента");
36:                return;
37:            }
38:            Stage update = new Stage();
39:            LoadFXML loader = ctx.getBean("loader",
LoadFXML.class);
40:            UpdateController controller =
(UpdateController) loader.loadModal("FXML/faculty/update.f
xml", "Редактировать факультет", update, stage);
41:            controller.setFaculty((Faculty) obj);
42:            update.showAndWait();
43:            Object data = update.getUserData();
44:            if (data != null && (Integer) data ==
Window.CLICK_EDIT) updateView();
45:        } catch (Exception ex){
46:            AlertWindow.errorAlert(ex.getMessage());

```

					ЯАС.1610574.ПЗ	Лист
						18
Изм.	Лист	№ докум.	Подпись	Дата		

Окончание листинга 4.2

```
47:                                     ex.printStackTrace();
48:
49:     }
50: }
51:
52:     @Override
53:     public void updateView() {
54:         FacultyMapper facultyMapper =
55:             ctx.getBean("facultyMapper", FacultyMapper.class);
56:         ObservableList<Faculty> faculties =
57:             FXCollections.observableArrayList(facultyMapper.getFaculty());
58:         table.setItems(faculties);
59:     }
60: }
```

Слушатель, выполняющий добавление факультета представлен на листинге 4.3.

Листинг 4.3 – метод add

```
1: @FXML
2:     private void add() {
3:         try {
4:             FacultyMapper facultyMapper =
5:                 ctx.getBean("facultyMapper", FacultyMapper.class);
6:             facultyMapper.insertFaculty(nameText.getText());
7:             close(Window.CLICK_ADD);
8:         } catch (Exception ex) {
9:             AlertWindow.errorAlert(ex.getMessage());
10:            ex.printStackTrace();
11:        }
12:    }
```

Класс-слушатель представленный в листинге 4.4 выполняет:

1. Удаление должности;
2. Обновление данных о должности;
3. Добавление должности.

Листинг 4.4 – класс PostController

```
1: public class PostController extends TabController {
2:
3:     @Override
4:     public void add() {
5:         try {
6:             Stage stage =
modalWindow("FXML/post/add.fxml", "Добавить должность");
7:             Object data = stage.getUserData();
8:             if (data != null && (Integer) data ==
Window.CLICK_ADD) updateView();
9:         } catch (Exception ex) {
10:             AlertWindow.errorAlert(ex.getMessage());
11:             ex.printStackTrace();
12:         }
13:     }
14:
15:     @Override
16:     public void update() {
17:         try{
18:             Object obj =
table.getSelectionModel().getSelectedItem();
19:             if (obj == null) {
20:                 AlertWindow.errorAlert("Нет выбранного
элемента");
21:                 return;
22:             }
23:             Stage update = new Stage();
24:             LoadFXML loader = ctx.getBean("loader",
LoadFXML.class);
25:             UpdateController controller =
(UpdateController) loader.loadModal("FXML/post/update.fxml
", "Редактировать должность", update, stage);
26:             controller.setPost((Post) obj);
27:             update.showAndWait();
28:             Object data = update.getUserData();
29:             if (data != null && (Integer) data ==
Window.CLICK_EDIT) updateView();
30:         } catch (Exception ex){
31:             AlertWindow.errorAlert(ex.getMessage());
32:             ex.printStackTrace();
33:         }
34:     }
35: }
```

Запросы указанные в варианте задания:

1 Запрос, возвращающий данные о жильцах указанной комнаты
представлен в листинге 4.5.

					ЯАС.1610574.ПЗ	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

Листинг 4.5 – Запрос возвращающий данные о жильцах

```
1: SELECT *
2: FROM TENANT
3: WHERE
4: ROOM_ID = #{roomId}
```

2 Запрос, высчитывающий на текущий момент общую используемую площадь представлен в листинге 4.6.

Листинг 4.6 – Запрос, высчитывающий общую используемую площадь

```
1: SELECT SUM(square)
2: FROM ROOM
3: WHERE
4: ID IN (SELECT ROOM_ID FROM TENANT)
```

3 Запрос, получающий общее число свободных комнат на текущий момент представлен в листинге 4.7.

Листинг 4.7 - Запрос, получающий общее число свободных комнат

```
1: SELECT ID
2: FROM ROOM
3: WHERE
4: ID NOT IN (SELECT ROOM_ID FROM TENANT)
```

4 Запрос, возвращающий инвентарь закрепленный за определенной комнатой представлен в листинге 4.8

Листинг 4.8 - Запрос, возвращающий инвентарь закрепленный за определенной комнатой

```
1: SELECT
2: r.id as "room_id",
3: i.name as "inventory_name"
4: FROM ROOM r
5: INNER JOIN ROOM_INVENTORY ri ON ri.room_id = r
6: INNER JOIN INVENTORY i ON i.id = ri.inventory_id
7: WHERE
8: r.id = #{id}
```

5 Запрос, возвращающий белье закрепленное за определенным студентом представлен в листинге 4.9

Листинг 4.9 - Белье, закреплённое за определённым студентом

```
1: SELECT
2: p.name as "property_name"
3: FROM PROPERTY p
4: INNER JOIN PROPERTY_TENANT pt ON pt.property_id = p.id
5: INNER JOIN TENANT t ON t.number_contract = pt.tenant_id
6: WHERE
7: t.number_contract = #{number_contract}
```

6 Запрос на вывод жильцов, живущих в определенной комнате представлен в листинге 4.10.

Листинг 4.10 – Запрос на жильцов, живущих в определенной комнате

```
1: SELECT
2: *
3: FROM TENANT t
4: WHERE
5: t.room_id = #{roomId}
```

7 Запрос на вывод всех договор заключенные с 1 в указанном году представлен в листинге 4.11.

Листинг 4.11

```
1: SELECT
2: t.number_contract
3: FROM TENANT t
4: WHERE
5: extract(year from t.eviction_date) = #{year}
6: AND extract(month from t.eviction_date)>8
```

8 Запрос на число студентов живущих в данный момент в общежитии представлен в листинге 4.12

Листинг 4.12 - Запрос на число студентов живущих в данный момент в общежитии

```
1: SELECT
2: COUNT(t.number_contract)
3: FROM TENANT t
4: WHERE
5: t.eviction_date IS NULL;
```

9 Запрос на определение количества свободных комнат для проживания представлен в листинге 4.13

Листинг 4.13 - Запрос на определение количества свободных комнат для проживания

```
1: SELECT COUNT(ID)
2: FROM ROOM
3: WHERE
4: ID NOT IN (SELECT ROOM_ID FROM TENANT)
```

5 ВЫБОР И ОСНОВАНИЕ СРЕДСТВ РАЗРАБОТКИ ПРИЛОЖЕНИЯ

Для создания приложения используется JavaFX. JavaFX — платформа на основе Java для создания приложений с насыщенным графическим интерфейсом. Может использоваться как для создания настольных приложений, запускаемых непосредственно из-под операционных систем, так и для интернет-приложений (RIA), работающих в браузерах, и для приложений на мобильных устройствах. JavaFX призвана заменить использовавшуюся ранее библиотеку Swing. Платформа JavaFX конкурирует с Microsoft Silverlight, Adobe Flash и аналогичными системами.

Для реализации задачи по разработке информационной системы была выбрана СУБД Oracle 11g r2. Oracle Database - это объектно-реляционная система поддерживающая некоторые технологии, реализующие объектно-ориентированный подход, то есть обеспечивающих управление создания и использования баз данных.

Ключевые возможности Oracle Database:

Real Application Cluster (RAC) обеспечивает работу одного экземпляра базы данных на нескольких узлах grid, позволяя управлять нагрузкой и гибко масштабировать систему в случае необходимости.

Automatic Storage Management (ASM) позволяет автоматически распределять данные между имеющимися ресурсами систем хранения данных, что повышает отказоустойчивость системы и снижает общую стоимость владения (TCO).

Производительность. Oracle Database позволяет автоматически управлять уровнями сервиса и тиражировать эталонные конфигурации в рамках всей сети.

Простые средства разработки. Новый инструмент разработки приложений HTML DB позволит простым пользователям создавать эффективные приложения для работы с базами данных в короткие сроки.

Самоуправление. Специальные механизмы Oracle Database позволяют самостоятельно перераспределять нагрузку на систему, оптимизировать и корректировать SQL-запросы, выявлять и прогнозировать ошибки.

Большие базы данных. Теперь максимальный размер экземпляра базы данных Oracle может достигать 8 экзбайт.

Недорогие серверные системы. Oracle Database может использовать недорогие однопросессорные компьютеры или модульные системы из “серверов-лезвий”.

					ЯАС.1610574.ПЗ	Лист
						23
Изм.	Лист	№ докум.	Подпись	Дата		

6 РЕАЛИЗАЦИЯ ЗАКОНЧЕННОГО ПРИЛОЖЕНИЯ, РАБОТАЮЩЕГО С СОЗДАННОЙ БАЗОЙ ДАННЫХ

6.1 Разработка и построение интерфейса главной и рабочих форм

Создание всех компонентов приложения происходит с помощью FXML. Для упрощения организации и работы с интерфейсом в JavaFX может использоваться язык разметки FXML, который создан на основе XML. FXML позволяет определить интерфейс приложения декларативным способом подобно тому, как веб-страницы определяются с помощью HTML.

Рабочие формы приложения представлены в виде диалоговых окон. Данные диалоги предназначены для выбора параметров запросов, добавления, редактирования, удаления записей.

Скриншоты главной и некоторых диалоговых окон представлены в приложении Г.

6.2 Построение главного меню и кнопок панели инструментов

Главное меню программы представлено двумя разделами: таблицы и запросы.

Таблицы содержат следующие разделы:

1. Сотрудники;
2. Должности;
3. Комнаты;
4. Жители;
5. Нежилые комнаты;
6. И др.

Создание подменю `add_room` представлено в листинге 6.1.

Листинг 6.1 – Создание подменю `add_room`

```
1: <AnchorPane prefHeight="200.0" prefWidth="268.0"
2: xmlns="http://javafx.com/javafx/11.0.1"
3: xmlns:fx="http://javafx.com/fxml/1"
4: fx:controller="by.pdu.dormitory.ui.room.AddController">
5: <children>
6: <Button layoutX="180.0" layoutY="161.0"
7: mnemonicParsing="false" onAction="#add"
8: prefHeight="25.0" prefWidth="74.0" text="Добавить" />
9: <ComboBox fx:id="unitBox" layoutX="30.0"
10: layoutY="119.0" prefHeight="25.0"
11: prefWidth="209.0" promptText="Блок" />
12: <TextField fx:id="countBetsField"
13: layoutX="30.0" layoutY="75.0" prefHeight="25.0"
14: prefWidth="209.0" promptText="Кол. коек" />
15: </children>
16: </AnchorPane>
```

					ЯАС.1610574.ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		24

6.3 Выполнение программного кода на языке Java

Для взаимодействия mybatis с базой данных необходимо его сконфигурировать, так параметры представлены в листинге 6.2

Листинг 6.2 – mybatis-config.xml

```
1: <properties
2:     resource="application.properties"> <!--ссылка
    на файл со свойствами(ссылка на СУБД, логин, пароль и
    тп.)-->
3: </properties>
4:
5: <environments default="development"><!--в данном
    блоке настраиваются подключения к БД-->
6:     <environment id="development">
7:         <transactionManager type="JDBC"/>
8:         <dataSource type="POOLED">
9:             <property name="driver"
    value="\${db.driver}"/>
10:            <property name="url"
    value="\${db.url}"/>
11:            <property name="username"
    value="\${username}"/>
12:            <property name="password"
    value="\${password}"/>
13:        </dataSource>
14:    </environment>
15: </environments>
16:     <mapper
    class="by.pdu.dormitory.mapper.TenantMapper"/>
17:     <mapper
    class="by.pdu.dormitory.mapper.InventoryMapper"/>
18:     <mapper
    class="by.pdu.dormitory.mapper.FacultyMapper"/>
19:     <mapper
    class="by.pdu.dormitory.mapper.PropertyMapper"/>
20:     <mapper
    class="by.pdu.dormitory.mapper.PostMapper"/>
21:     <mapper
    class="by.pdu.dormitory.mapper.OtherRoomMapper"/>
22:     <mapper
    class="by.pdu.dormitory.mapper.UnitMapper"/>
23:     <mapper
    class="by.pdu.dormitory.mapper.GroupMapper"/>
24:     <mapper
    class="by.pdu.dormitory.mapper.RoomMapper"/>
25:     <mapper
    class="by.pdu.dormitory.mapper.EmployeeMapper"/>
26:
27: </mappers>
28: </configuration>
```

					ЯАС.1610574.ПЗ	Лист
						25
Изм.	Лист	№ докум.	Подпись	Дата		

MyBatis позволяет выполнять запросы на SQL, который он берет из маппера.

Маппер факультета представлен в листинге 6.3.

Листинг 6.3 – FacultyMapper.xml

```
1: <mapper
  namespace="by.pdu.dormitory.mapper.FacultyMapper">
2:   <resultMap id="facultyResultMap"
  type="by.pdu.dormitory.domain.Faculty">
3:     <id column="faculty_id" property="id"/>
4:     <result column="faculty_name" property="name"/>
5:   </resultMap>
6:   <select id="getFaculty" resultMap="facultyResultMap">
7:     SELECT
8:       id as "faculty_id",
9:       name as "faculty_name"
10:    FROM
11:      FACULTY
12:    ORDER BY id ASC
13:   </select>
14:   <insert id="insertFaculty">
15:     INSERT INTO FACULTY (NAME)
16:     VALUES (#{name})
17:   </insert>
18:   <update id="updateFaculty">
19:     UPDATE FACULTY
20:     SET NAME =#{name}
21:     WHERE id = #{id}
22:   </update>
23: </mapper>
```

Так же, загрузка информации о сущности tenant представлена в листинге 6.4.

Листинг 6.4 – Загрузка информации о сущности tenant

```
1: @Override
2: public void updateView() {
3:     TenantMapper tenantMapper =
  ctx.getBean("tenantMapper", TenantMapper.class);
4:     ObservableList<EmployeeTenant> employees =
  FXCollections.observableArrayList(tenantMapper.getEmployeeTenant());
5:     ObservableList<Student> students =
  FXCollections.observableArrayList(tenantMapper.getStudent());
6:     ObservableList list =
  FXCollections.observableArrayList();
7:     list.addAll(employees);
8:     list.addAll(students);
9:     table.setItems(list);
10: }
```


ЗАКЛЮЧЕНИЕ

Разработка больших проектов, к которым, вне всякого сомнения, относятся системы огромных корпораций по созданию продуктов программного обеспечения, представляет собой задачу высокой сложности. Такая система разрабатывается командами, и процесс разработки зачастую занимает огромное количество времени. Требования к специалисту, участвующему в подобных разработках, довольно высокие, поскольку он должен иметь навыки работы в команде, обладать высокими знаниями в области разработки как прикладного, так и системного программного обеспечения, иметь фундаментальные и практические навыки работы с СУБД, сетевыми технологиями, технологиями защиты данных и т. п.

В результате выполненной работы, была создана база данных для информационной системы общежития, а так же эффективно работающее с этой базой данных приложение. Полученная комбинация представляет собой информационную систему общежития.

Разработанная база данных удовлетворяет всем требованиям, предъявленным в задании, и позволяет без проблем хранить и извлекать требуемую информацию.

В процессе выполнения данной курсовой работы были закреплены навыки в программировании на Java SE 8.0, проектировании баз данных и реализации их в СУБД Oracle.

					ЯАС.1610574.ПЗ	Лист
						27
Изм.	Лист	№ докум.	Подпись	Дата		

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Data flow diagram - Wikipedia, the free encyclopedia / Многоязычная общедоступная свободно распространяемая энциклопедия, публикуемая в Интернете Википедия. Режим доступа: http://en.wikipedia.org/wiki/Data_flow_diagram

2 Entity-relationship model – Wikipedia, the free encyclopedia / Многоязычная общедоступная свободно распространяемая энциклопедия, публикуемая в Интернете Википедия. Режим доступа: http://en.wikipedia.org/wiki/Entity-relationship_model

3 PostgreSQL Manual [Электронный ресурс] / Официальный сайт PostgreSQL. Документация по PostgreSQL. Режим доступа:

4 <http://dev.postgresql.com/doc/refman/5.0/en/index.html>

5 Документация по языку Java от компании Sun [Электронный ресурс]. – Электрон. текстовые дан. (270 Мб). – Sun Microsystems, Inc., 2006

6 Документация по PostgreSQL Connector/J [Электронный ресурс]. – Электрон. текстовые дан. (3 Мб). – Sun Microsystems, Inc., 2008

7 Unified Modeling Language – Wikipedia, the free encyclopedia / Многоязычная общедоступная свободно распространяемая энциклопедия, публикуемая в Интернете Википедия. Режим доступа: http://en.wikipedia.org/wiki/Unified_Modeling_Language

					ЯАС.1610574.ПЗ	Лист
						28
Изм.	Лист	№ докум.	Подпись	Дата		

ПРИЛОЖЕНИЕ А

(обязательное)

КОНЦЕПТУАЛЬНАЯ СХЕМА БД

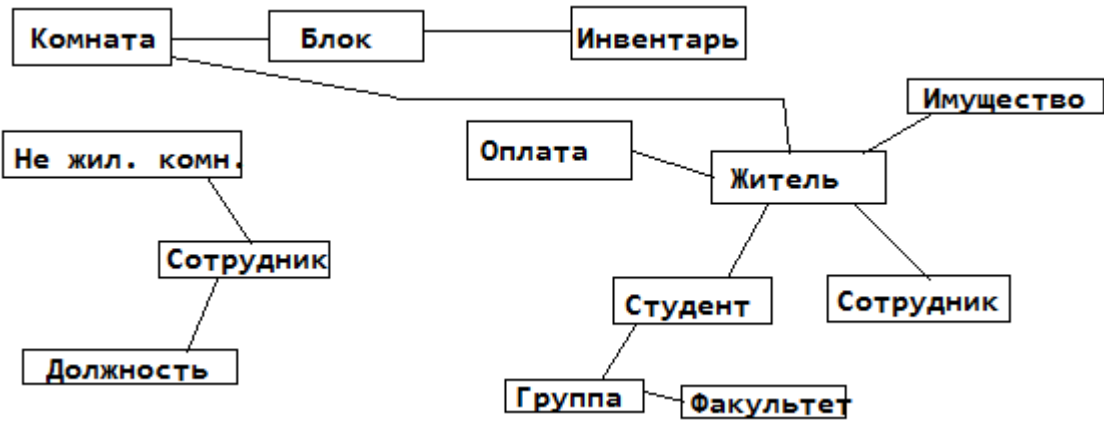


Рисунок А.1 – Концептуальная схема базы данных

ПРИЛОЖЕНИЕ Б

ER-диаграмма БД

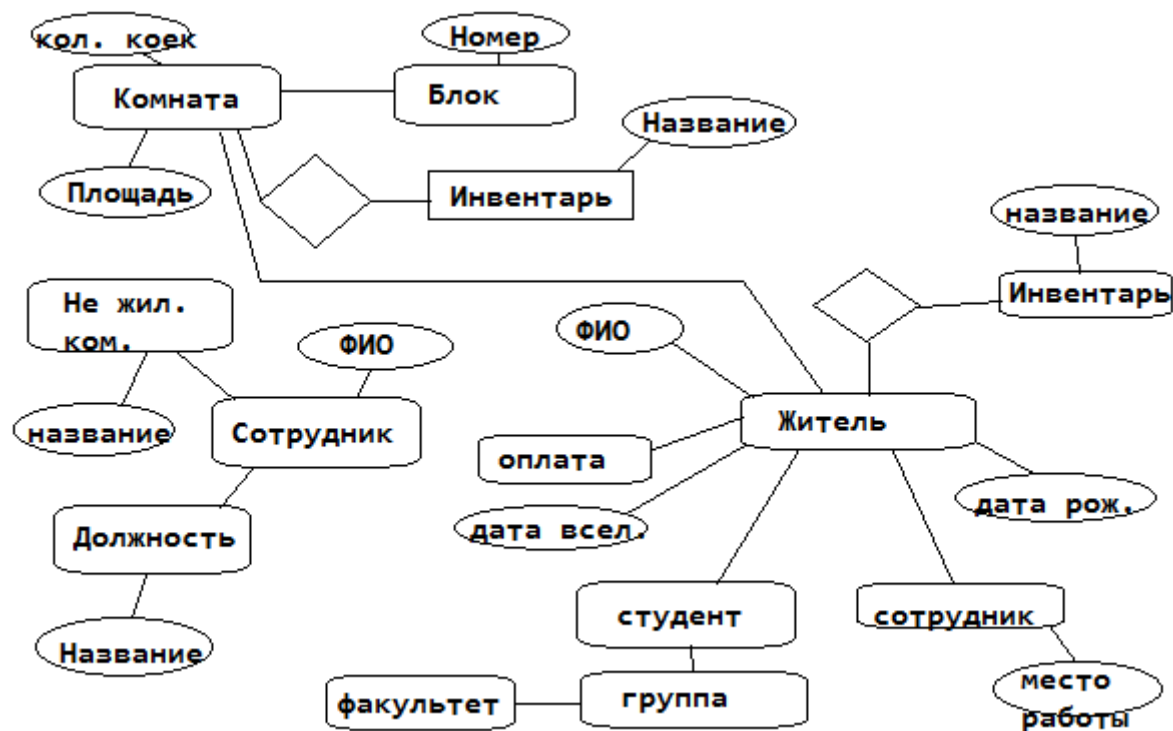


Рисунок Б.1 – ER-диаграмма проектируемой базы данных

ПРИЛОЖЕНИЕ В

(обязательное)

СХЕМА РЕЛЯЦИОННОЙ БАЗЫ ДАННЫХ

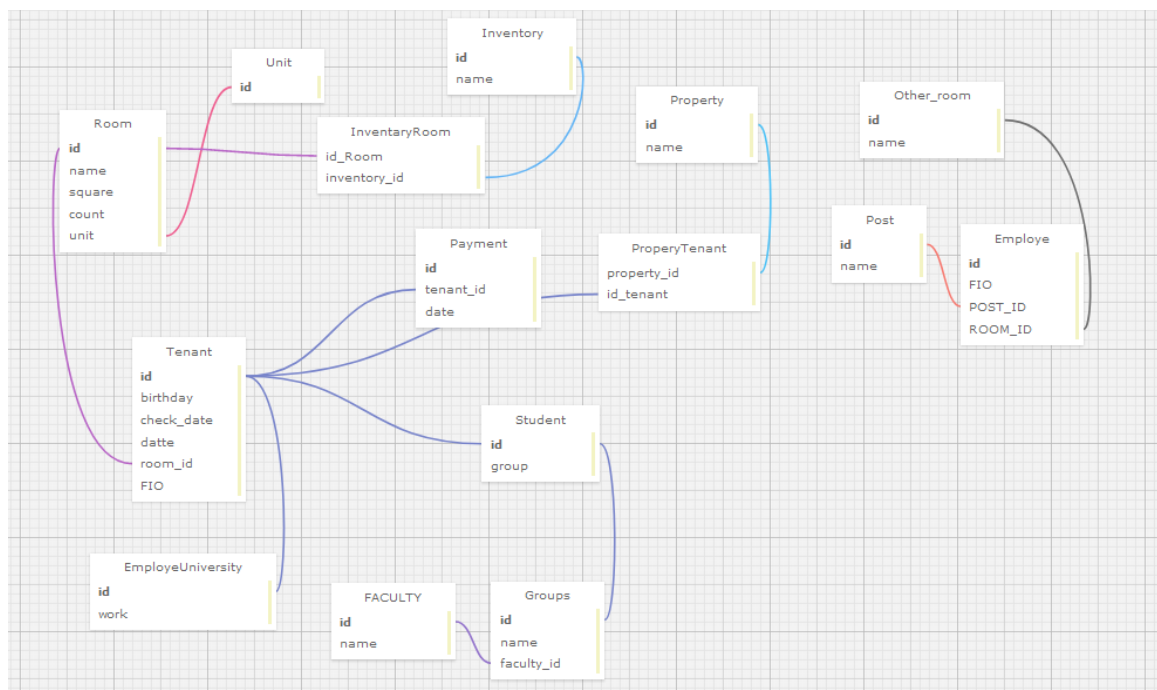


Рисунок В.1 – Схема реляционной базы данных

(обязательное)

ГЛАВНАЯ И РАБОЧИЕ ФОРМЫ ПРИЛОЖЕНИЯ

[illegible]

Рисунок Г.1 – Главная форма приложения

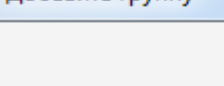


Рисунок Г.2 – Диалоговое окно «add_group»

Добавить комнату

Площадь

Кол. коек

Блок

Добавить

Рисунок Г.3 – Диалоговое окно «add_room»

Добавить жильца

Номер договора

ФИО

Комната

Дата рождения

Студент Сотрудник

Группа

Вещи	Выданные
Скатерть	No content in table
Полотенце махровое	

Добавить Удалить

Добавить

Рисунок Г.4 – Диалоговое окно «add_tenant»

ПРИЛОЖЕНИЕ Д

(обязательное)

ЛИСТИНГ ПРОГРАММЫ

Листинг программы представлен на диске.

					ЯАС.1610574.ПЗ	Лист
						34
Изм.	Лист	№ докум.	Подпись	Дата		