

Государственное АВТОНОМНОЕ образовательное

учреждение высшего образования

**Московский городской университет управления Правительства Москвы ИМЕНИ Ю.М. ЛУЖКОВА**

Отделение прикладного и академического бакалавриата

Кафедра экономики городского хозяйства и жилищного права

Направление: 38.03.04 «Государственное и муниципальное управление»

Профиль «Управление цифровыми проектами города»

Форма обучения: очная

**Курсовая работа**

по учебной дисциплине

«Система управления базами данных»

на тему: «Построение базы данных по учету актуализации информации на портале открытых данных Правительства Москвы»

|  |  |  |  |
| --- | --- | --- | --- |
| Группа/ курс | 21ГМУ-УЦП11.1 / 3 курс |  |  |
| Студент |  |  | Шумский И.Н. |
| Подпись  руководителя о  допуске к защите | (подпись) |  | канд. пед. наук, доц. Филимонова Е.В. |
| Дата защиты | (подпись)  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_ 2023 года |  |  |
| Оценка |  |  |  |

Москва 2024

Оглавление

[ВВЕДЕНИЕ 3](#_Toc155621546)

[РАЗДЕЛ 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ. ПОСТРОЕНИЕ КОНЦЕПТАУЛЬНОЙ МОДЕЛИ БАЗЫ ДАННЫХ 6](#_Toc155621547)

[1.1. Портал открытых данных Правительства Москвы 6](#_Toc155621548)

[1.2. Построение концептуальной модели в нотации Питера Чена 10](#_Toc155621549)

[РАЗДЕЛ 2. НОРМАЛИЗАЦИЯ И ПОСТРОЕНИЕ ДАТАЛОГИЧЕСКОЙ МОДЕЛИ 13](#_Toc155621550)

[2.1. Нормализация. Построение даталогической модели 13](#_Toc155621551)

[2.2 Добавление справочных таблиц 20](#_Toc155621552)

[РАЗДЕЛ 3. построение физической модели базы данных 22](#_Toc155621553)

[3.1. Создание таблиц базы данных 22](#_Toc155621554)

[3.2. Построение типовых запросов на выборку данных 29](#_Toc155621555)

[ЗАКЛЮЧЕНИЕ 43](#_Toc155621556)

[СПИСОК ЛИТЕРАТУРЫ 44](#_Toc155621557)

[Приложение А 47](#_Toc155621558)

[Приложение Б 49](#_Toc155621560)

[Приложение В 51](#_Toc155621562)

[Приложение Г 53](#_Toc155621564)

[Приложение Д 58](#_Toc155621566)

ВВЕДЕНИЕ

**Актуальность темы** **исследования** обусловлена тем, что в условиях нарастающей сложности социальных процессов и новых экономических вызовов серьезной задачей для органов исполнительной власти является необходимость формирования моделей принятия решений и реализации государственных функций, основанных на активном участии гражданского общества в управлении государством, а также на использовании современных механизмов общественного контроля.[[1]](#footnote-1) Для реализации данных целей, в 2014 году в России была принята концепция открытости федеральных органов исполнительной власти.

Открытость данных не только затрагивает вопросы прозрачности деятельности органов власти, но и позволяет создавать проекты общественного контроля для взаимодействия государства и бизнеса, а также для взаимодействия разных ветвей и уровней власти. За последние годы в фокусе оказались усилия по стандартизации данных в конкретных направлениях и по централизованному раскрытию данных из разных стран.

Примером развития темы открытых данных в мире является расширение числа открытых репозиториев научных данных, публикуемых на условиях открытого доступа (Open Access). Это более 2677 репозиториев открытых данных[[2]](#footnote-2), на которых ученые из исследовательских центров по всему миру раскрывают данные о результатах исследований в соответствии с принципами воспроизводимости результатов исследований (FAIR – Findability, Accessibility, Interoperability and Reuse)[[3]](#footnote-3).

Учитывая вышеизложенное, тема работы «Построение базы данных по учету актуализации информации на портале открытых данных Правительства Москвы» является значимой и актуальной.

**Объектом** **исследования** является процесс учета актуализации информации на портале открытых данных Правительства Москвы

**Предметом** **исследования** Построение базы данных по учету актуализации информации на портале открытых данных Правительства Москвы**.**

**Целью** **исследования** – на основе проведенного анализа разработать и построить физическую модель базы данных по учету актуализации информации на портале открытых данных Правительства Москвы**.**

Поставленная цель и определила необходимость решения следующих **задач**:

1. Анализ портала открытых данных Правительства Москвы;
2. Разработка и построение концептуальной модели в нотации Питера Чена;
3. Построение и нормализация даталогической модели;
4. Построение физической модели базы данных;

Во введении сформулированы цели, задачи, объект, предмет, актуальность, изученность темы, содержание работы.

В первой главе проведен анализ портала открытых данных Правительства Москвы и построение концептуальной модели.

Вторая глава посвящена построению и нормализации даталогической модели.

В третьей главе представлена физическая модель базы данных, продемонстрирован пример работы с базой данных.

В заключении представлены основные выводы по курсовому исследованию.

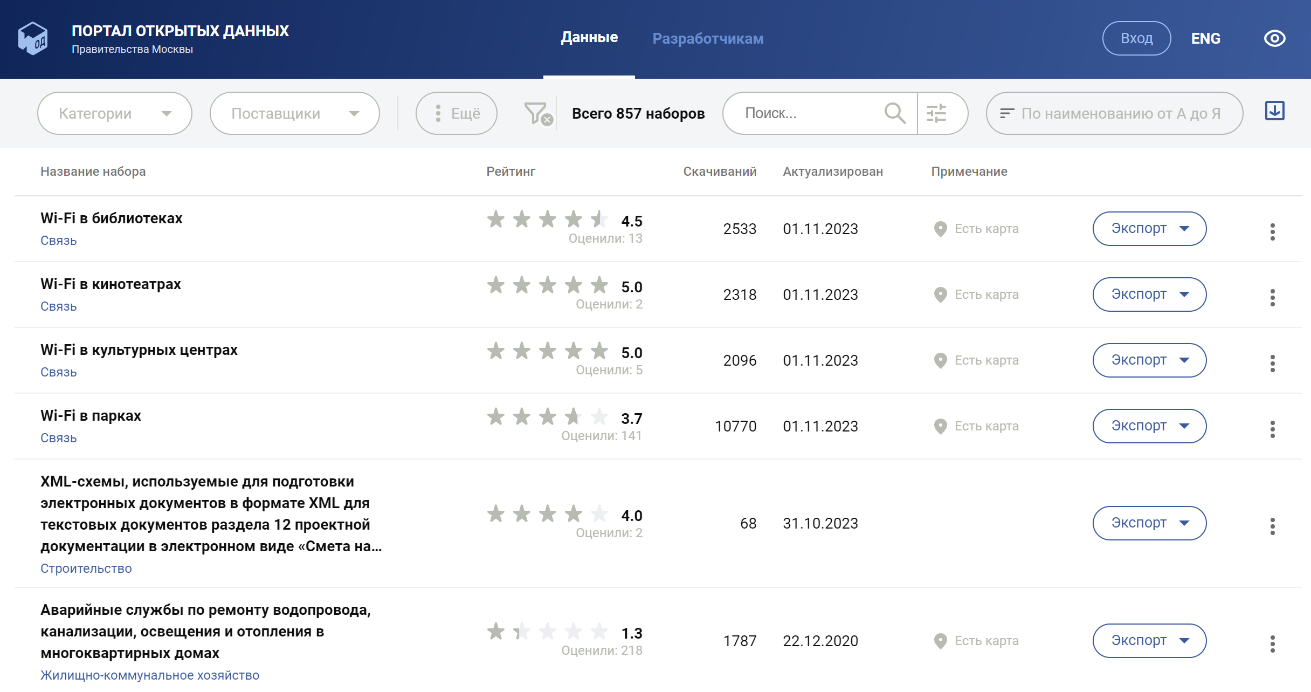
При написании курсовой работы были использованы следующие источники: нормативно-правовые акты, периодические издания, официальные интернет-ресурсы.

РАЗДЕЛ 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ. ПОСТРОЕНИЕ КОНЦЕПТАУЛЬНОЙ МОДЕЛИ БАЗЫ ДАННЫХ

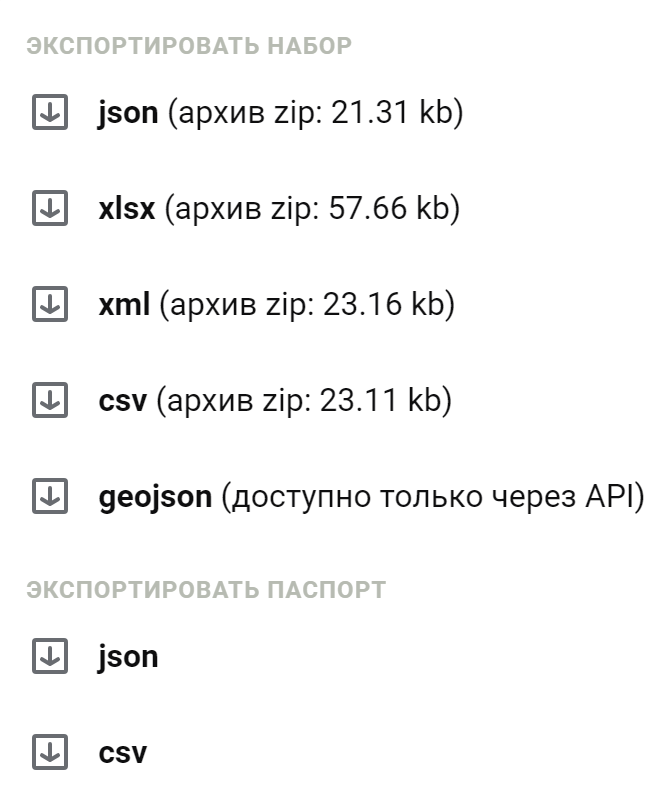
1.1. Портал открытых данных Правительства Москвы

Портал открытых данных Правительства Москвы является первым порталом открытых данных в России и был запущен для широкого круга пользователей 29 января 2013 года. На данный момент на портале опубликовано более 1000 наборов данных и справочников, в которых представлены сведения по различным направлениям: от объектов городской инфраструктуры до расписаний ближайших мероприятий, проводимых в городе Москва.

Также необходимо отметить то, что данный портал также предоставляет возможность частным разработчикам настроить информационный обмен, получая информацию из наборов данных посредством API (application programming interface).

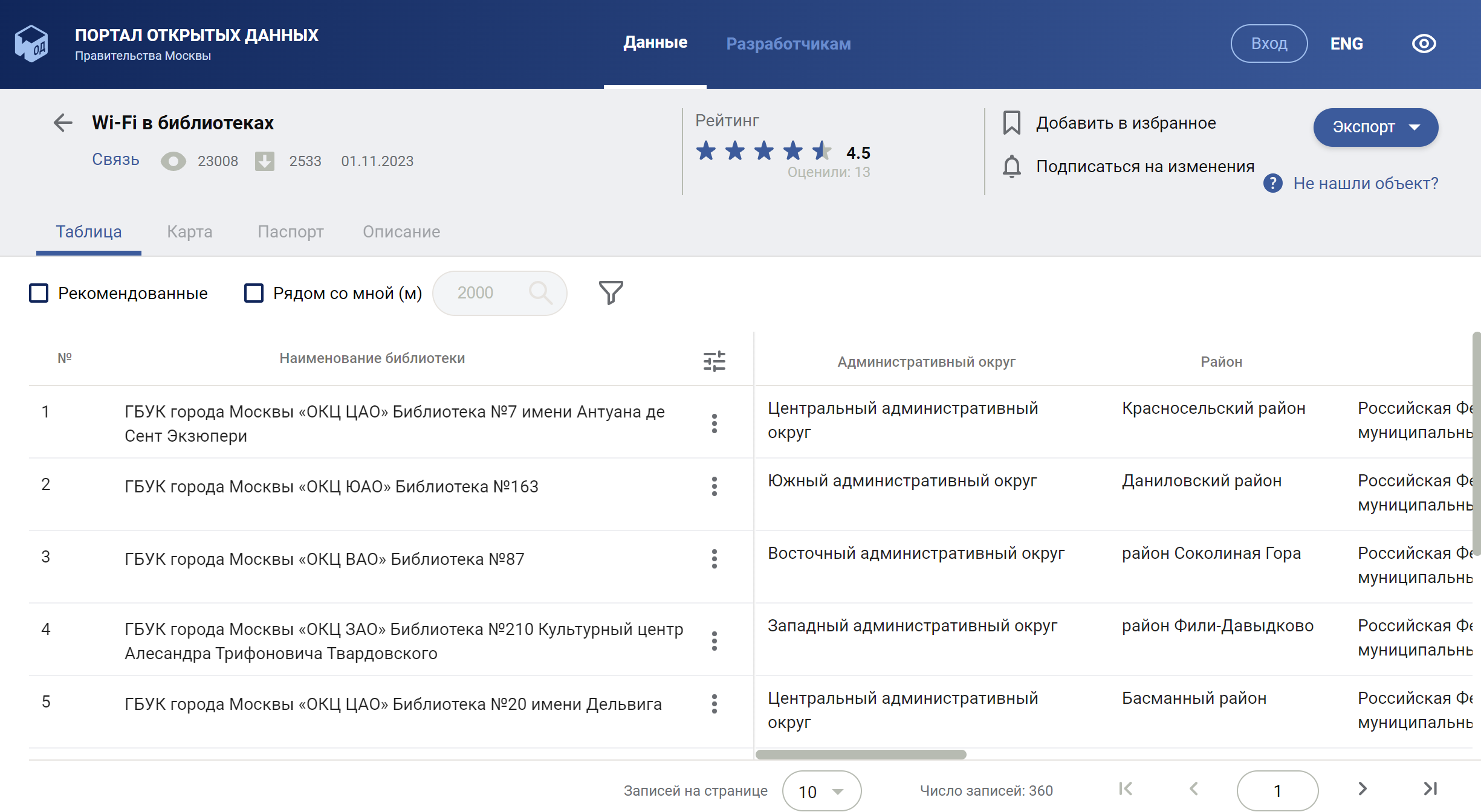
На текущий момент на портале представлено 857 активных наборов, поставщиками которых являются 48 органов исполнительной власти города Москвы (рисунок 1).

**Рисунок 1.** Главная страница Портала.

Как следует из рисунка 1, каждый набор данных имеет следующие основные поля: наименование, категорию, рейтинг, количество скачиваний, дату актуализации, примечание, а также возможность экспорта в различных форматах (рисунок 2).

**Рисунок 2.** Функция экспорта набора и паспорта

Для получения доступа к набору данных, необходимо перейти на соответствующую страницу, на которой представлена более детальная информация о наборе: таблица данных в веб-интерфейсе, карта (при наличии), описание, а также паспорт набора (рисунок 3).



**Рисунок 3.** Набор данных

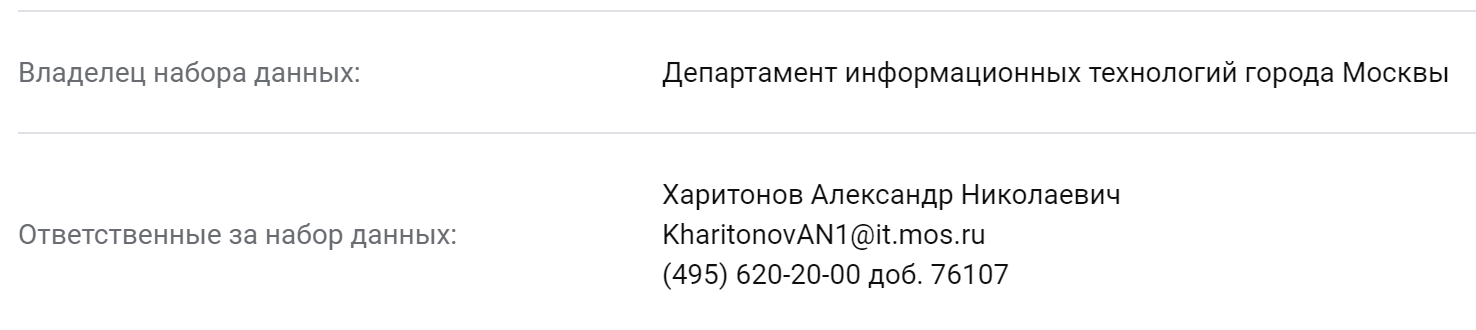
Перечень атрибутов набора данных, с указанием типа данных и обязательности заполнения (таблица 1) указан в Распоряжении Правительства Москвы №64-16-426/13 «Об утверждении регламента актуализации открытых данных в автоматизированной системе «Общегородской платформы открытых данных».

**Таблица 1. Пример паспорта набора данных.**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Название атрибута** | **Обязательность заполнения** | **Типа данных** |
| 1. | Название набора данных | Да | STRING |
| 2. | Тематическая категория | Да | STRING |
| 3. | Описание набора данных | Да | STRING |
| 4. | Владелец данных (ОИВ) | Да | STRING |
| 5. | Код ОИВ | Да | INT |
| 6. | Информационная система ОИВ | Да | STRING |
| 7. | Периодичность обновления данных | Да | STRING |
| 8. | Ключевые слова | Да | STRING |
| 9. | Начало отчетного периода публикации | Нет | DATE |
| 10. | Конец отчетного периода публикации | Да | DATE |
| 11. | Наличие географической привязки | Да | STRING |
| 12. | Режим публикации и обновления | Да | STRING |

*Источник: Распоряжение Правительства Москвы №64-16-426/13 «Об утверждении регламента актуализации открытых данных в автоматизированной системе «Общегородской платформы открытых данных».*

Также стоит отметить, что помимо владельца данных, каждый набор имеет ответственного сотрудника, с указанием его контактных данных (фамилии, имени, отчества, адреса электронной почты, а также контактного номера телефона) (рисунок 4).

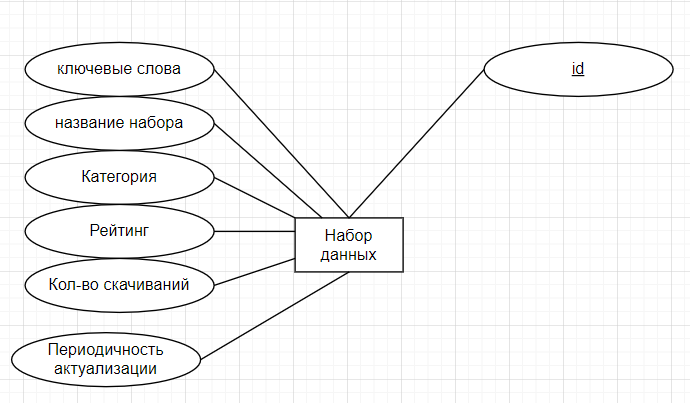


**Рисунок 4.** Информация о владельце набора и ответственном сотруднике.

Также на портале реализована функция поиска набора данных при помощи существующих фильтров. Все наборы разделены на 24 категории. Также возможен поиск по поставщику данных, видам и статусам набора, а также сезонности (рисунок 2).

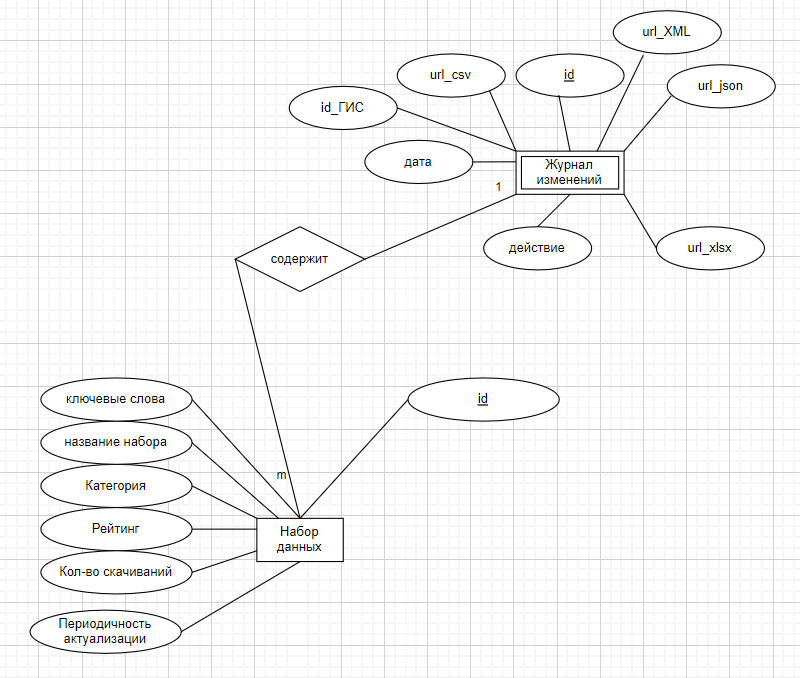
1.2. Построение концептуальной модели в нотации Питера Чена

После анализа предметной области перейдем к построению концептуальной модели в нотации Питера Чена. Для этого выделим основные сущности в рассматриваемой предметной области, их атрибуты, а также их возможные связи друг с другом.

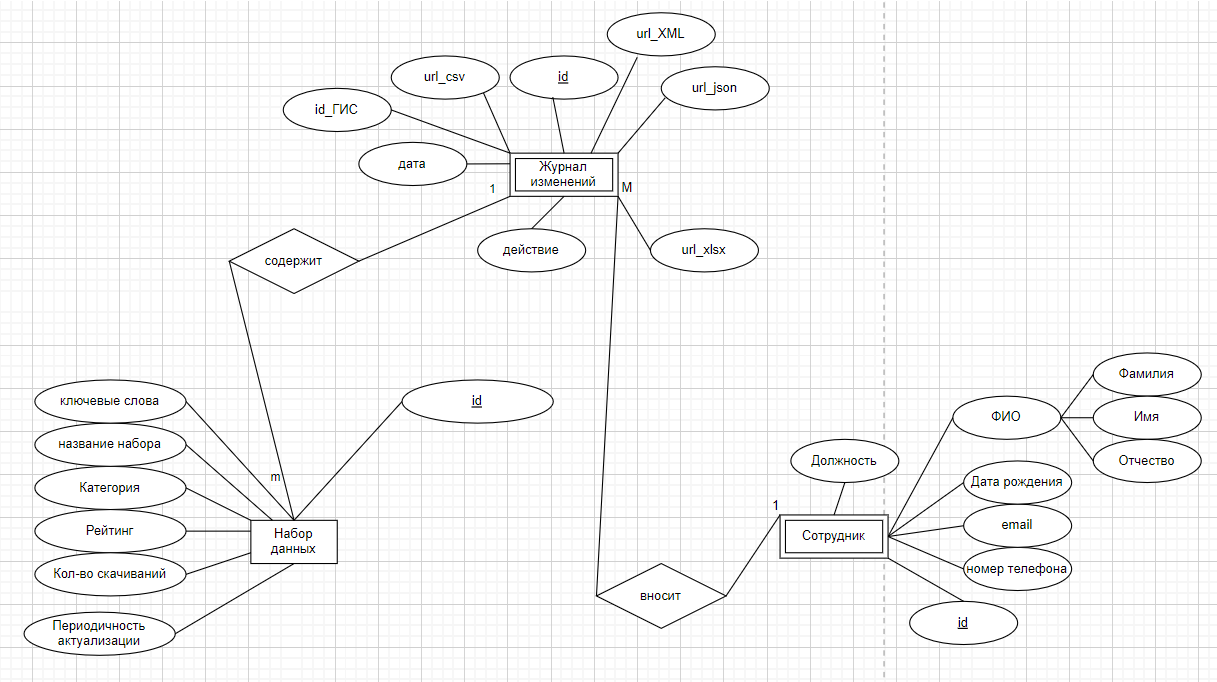
Прежде всего необходимо выделить сущность «Набор данных», со следующим набором атрибутов: id (идентификатор), ключевые слова, название набора, категория, рейтинг, количество скачиваний и периодичность актуализации.

**Рисунок 5.** Сущность «Набор данных»

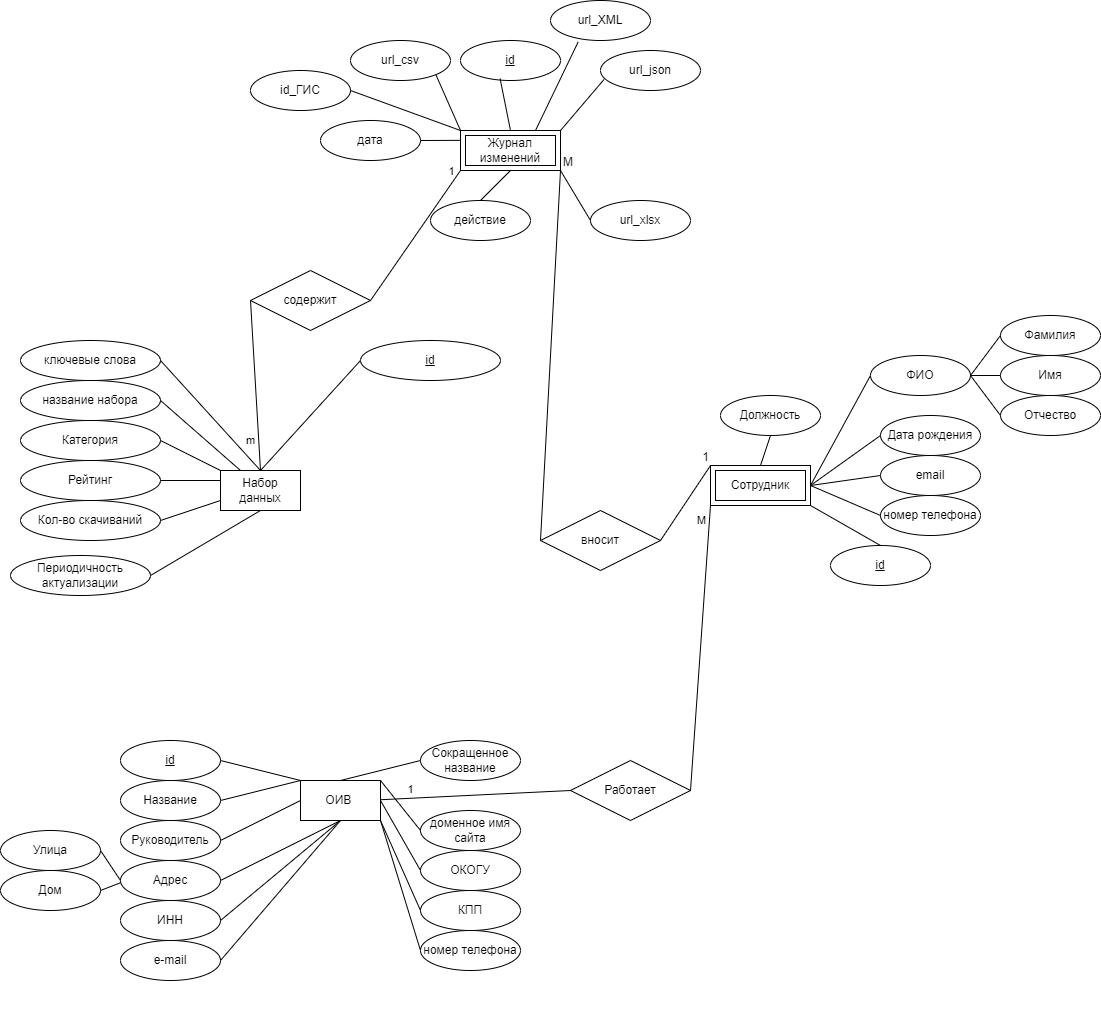
С набором данных может быть совершен ряд действий определенным сотрудником, поэтому выделяется еще одна сущность «Журнал изменений». Поскольку в данной информационной системе журнал изменений не может существовать без набора данных, сущность «журнал изменений» является слабой. Выделим следующий состав ее атрибутов: id (идентификатор), id\_набора, id\_ГИС (идентификатор информационной системы ОИВ), дата, действие, а также атрибуты, в которых хранятся ссылки на набор данных в различных форматах: url\_CSV, url\_XLSX, url\_JSON и url\_XML.



**Рисунок 6.** Сущности «Набор данных», «Журнал изменений»

 Тем не менее, сущность «Журнал изменений» не может существовать без сущности «Сотрудник», которая имеет следующий набор атрибутов: id (идентификатор), фамилия, имя, отчество, дата рождения, должность, номер телефона, адрес электронной почты.

**Рисунок 7.** Сущности «Набор данных», «Журнал изменений», «Сотрудник»

Сущность «сотрудник» является слабой сущностью, так как не может существовать без места работы, в нашем случае сущность место работы называется «ОИВ», и имеет следующие атрибуты: id (идентификатор), наименование, сокращенное наименование, руководитель, адрес, ИНН (идентификационный номер налогоплательщика), адрес электронной почты, доменное имя сайта, номер телефона, ОКОГУ (общероссийский классификатор органов государственной власти и управления) , КПП (Код причины постановки на учет) (рисунок 8).

**Рисунок 8.** Концептуальная модель базы данных

Раздел 2. нормализация и построение даталогической модели

2.1. Нормализация. Построение даталогической модели

Перед построением даталогической модели базы данных, необходимо выполнить ряд действий: определение связей между сущностями, выбор вида базы данных, приведение модели базы данных к нормальным формам.

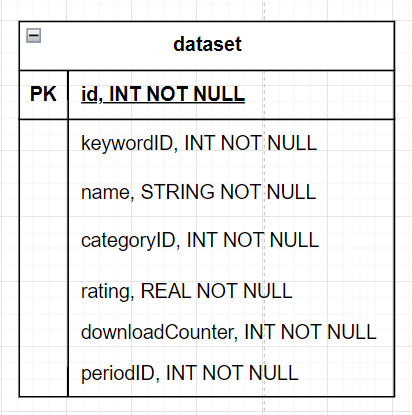
При дальнейшем проектирование и построении базы данных будет использована реляционная база данных – одна из наиболее распространенных видов.

Реляционные базы данных основаны на реляционной модели — интуитивно понятном, наглядном табличном способе представления данных. Каждая строка, содержащая в таблице такой базы данных, представляет собой запись с уникальным идентификатором, который называют ключом. Столбцы таблицы имеют атрибуты данных, а каждая запись обычно содержит значение для каждого атрибута, что дает возможность легко устанавливать взаимосвязь между элементами данных.[[4]](#footnote-4)

Нормализация – это процесс последовательного преобразования таблиц базы данных к виду, принятому в реляционной модели данных.

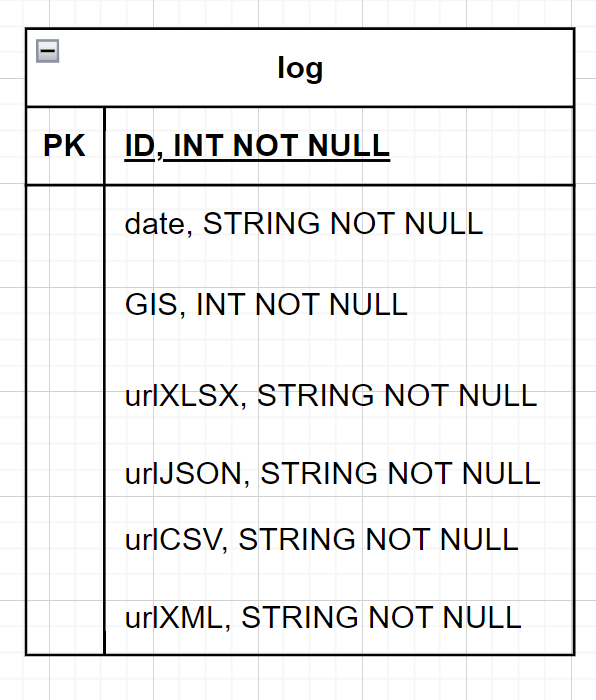
Специалисты различают пять последовательных этапов (форм) приведения данных к реляционному виду от первой нормальной формы (First Normal Form, 1NF) до пятой (5NF). На каждой из ступеней нормализации таблица приобретает новые черты, которые потребуются для перехода к очередному этапу.

Для приведения базы данных к первой нормальной форме (1NF), необходимо соблюдение атомарности всех атрибутов таблиц. В данном случае, под атомарностью понимается целостность и неделимость данных в базе данных. Для удобства в каждой реляционной таблице будем использовать только суррогатные простые первичные ключи, это позволит автоматически обеспечить 2 NF и соответственно полностью пропустить этот этап нормализации.

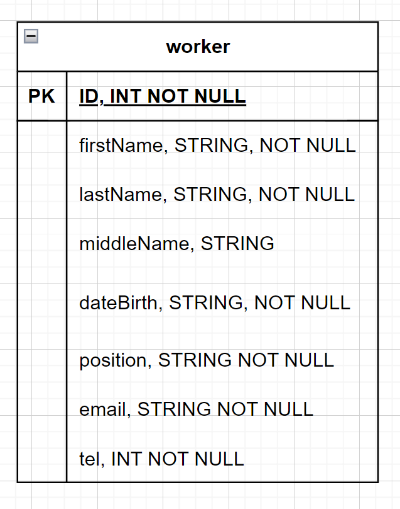
Сущность «набор данных» – таблица «dataSet». Определим первичный ключ id, типа значения INT. Атрибут «ключевое слово» обозначим **keywordID**. Чтобы использовать данное поле как внешний ключ справочной таблицы, обозначим его тип как INT. Атрибут «название» обозначим **name**, с типом данных STRING. Для атрибутов «категория» и «период актуализации» необходимо совершить действия, аналогичные для атрибута «ключевое слово», а именно обозначить их в качестве **categoryID**, **periodID** соотвественно, тип данных INT. Атрибут «рейтинг» назовем rating, с типом данных REAL. Атрибут «количество скачиваний» – **downloadCounter**, тип данных INT.

**Рисунок 9.** Реляционная таблица сущности «набор данных»

Сущность «журнал изменений» – таблица «log». Определим первичный ключ id, типа значения INT. Атрибут «дата» обозначим **date,** тип данных DATE. Атрибут «id\_ГИС» хранит в себе номер информационной системы, используемый ведомством при подготовке набора данных, поэтому обозначим его как **GIS**, тип данных INT. Следующие атрибуты хранят ссылки на набор данных в различных форматах и все имеют тип данных STRING, обозначим их соответственно: **urlCSV, urlXLSX, urlJSON и urlXML.**

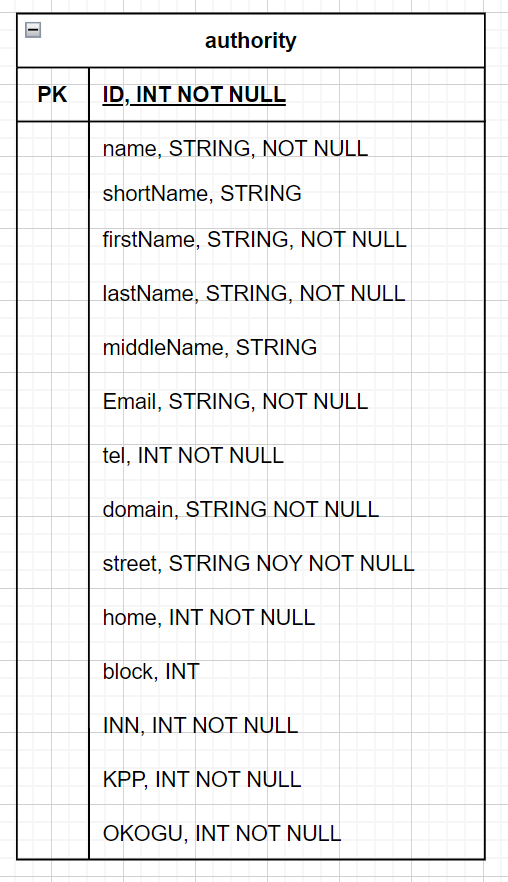


**Рисунок 10.** Реляционная таблица сущности «журнал изменений»

Сущность «сотрудник – таблица «worker». Определим первичный ключ id, типа значения INT. Атрибут «ФИО», для соблюдения атомарности, необходимо разделить на 3 отдельных атрибута с типами данных STRING, фамилия – **lastName,** имя – **firstName**  и отчество – **middleName**. Атрибут «дата рождения» обозначим **dateBirth**, тип данных DATE. Атрибут «должность» – **position**, тип данных STRING, атрибут «электронная почта» назовем **email**, тип данных STRING. И последний атрибут в данной таблице – «номер телефона» обозначим **tel**, тип данных INT.

**Рисунок 11.** Реляционная таблица сущности «сотрудник»

Сущность «ОИВ – таблица «authority». Определим первичный ключ id, типа значения INT. Атрибут «название» обозначим **name**, тип данных STRING. Атрибут «сокращенное название» назовем **shortName**, тип данных STRING. Атрибут «Руководитель», для соблюдения атомарности, необходимо разделить на 3 отдельных атрибута с типами данных STRING, фамилия – **lastName,** имя – **firstName**  и отчество – **middleName**. Атрибут «электронная почта» назовем **email**, тип данных STRING. И последний атрибут в данной таблице – «номер телефона» обозначим **tel**, тип данных INT. Атрибут «доменное имя сайта» обозначим **domain**, тип данных STRING. Атрибут «Адрес», для соблюдения атомарности, необходимо разделить на 3 отдельных атрибута: улица – **street,** тип данных STRING,номер дома – **home,** тип данных INT и корпус – **block,** тип данных STRING**.** Также атрибуты ИНН, КПП, ОКОГУ обозначим в виде **INN, KPP, OGOKU,** соответственно, с типом данных INT.



**Рисунок 12.** Реляционная таблица сущности «ОИВ»

Таким образом, этап приведения к 1NF завершен.

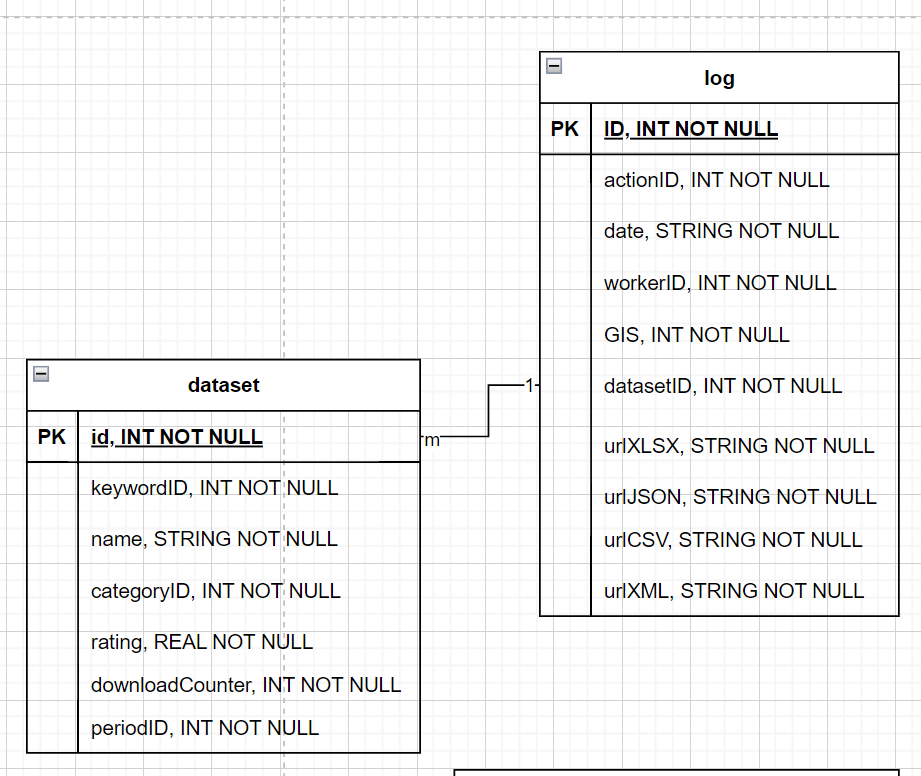
Приведение ко второй нормальной форме направлено на устранение частичных зависимостей, которые могут возникать между частями составного первичного ключа и остальными атрибутами. Поскольку на предыдущем этапе первичные ключи всех таблиц были выбраны простыми база данных уже приведена ко второй нормальной форме.

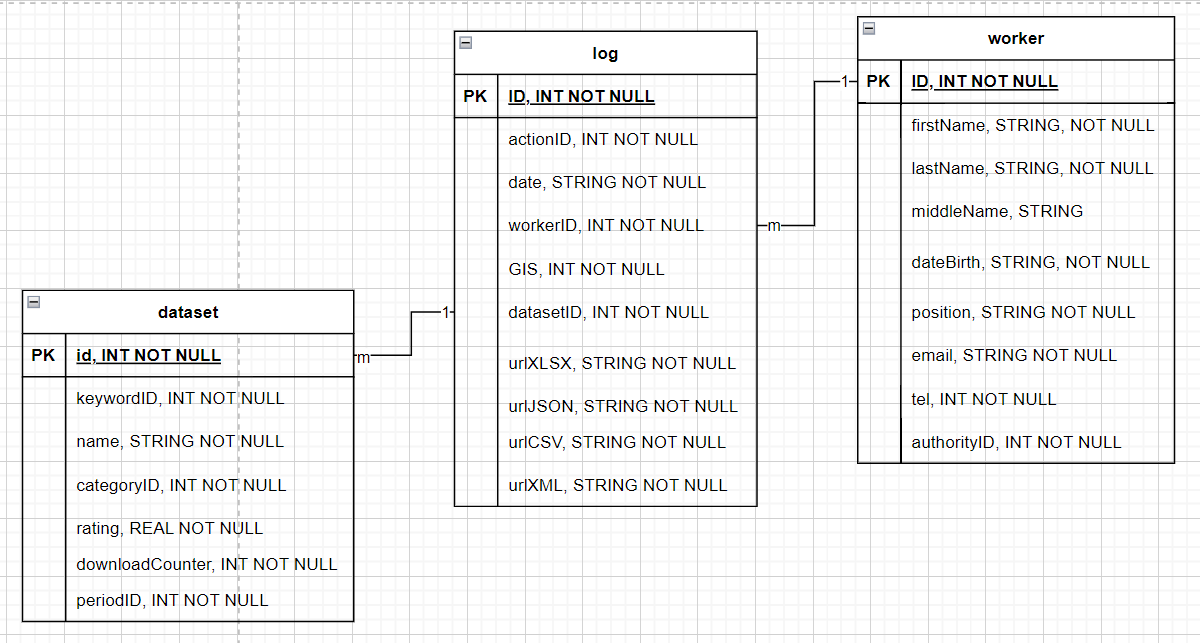
Таблица находится в третьей нормальной форме 3NF, если между ее атрибутами нет транзитивных связей (внутри приведенной к 3NF таблицы должны отсутствовать связи «один ко многим»).

Таблица приведена к четвертой нормальной форме, если она соответствует 3NF и в ней отсутствуют многозначные зависимости (многие ко многим).

Для приведения таблиц к 3NF и 4NF, необходимо реализовать связи между сущностями.

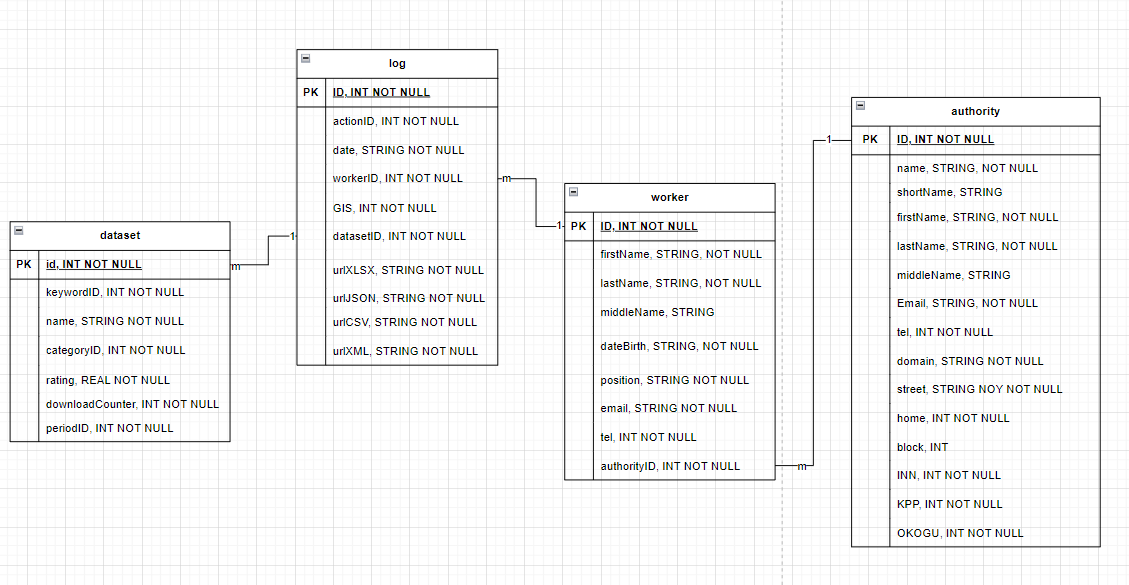
Определение связи между таблицами «log» и «datSet». В журнале изменений фиксируется изменение с конкретным набором данных в конкретный момент времени, следовательно у одного такого действия может быть только 1 набор данных, однако с 1 набором данных может совершаться ряд действий в разные периоды. Это связь «один ко многим», т.е. необходимо добавить в таблицу «log» внешний ключ, с обозначением **dataSetID,** тип данных INT.

**Рисунок 13.** Связь между сущностями «набор данных» - «журнал изменений»  
 Как было указано ранее, в журнале изменений фиксируется конкретное действие, совершенное конкретным сотрудником, следовательно у каждой записи в журнале изменений может быть только один сотрудник, в то время как у одного сотрудника может быть несколько записей. Это связь «один ко многим», т.е. необходимо добавить в таблицу «log» внешний ключ, с обозначением **workerID,** тип данных INT.



**Рисунок 14.** Связь между сущностями «набор данных», «журнал изменений» и «сотрудник»

Каждый сотрудник имеет 1 место работы, в то время как в 1 органе исполнительной власти работает множество сотрудников. Это также пример связи «один ко многим», т.е. необходимо добавить в таблицу «worker» внешний ключ, с обозначением **authorityID,** тип данных INT.

****

**Рисунок 15.** Основные связи между сущностями

2.2 Добавление справочных таблиц

Для завершения этапа даталогического проектирования остается рассмотреть структуру и наполнение справочных таблиц: **keywordList, categoryList, periodList, actionList.**

Справочная таблица **keywordList** включает в себя следующие поля:

Первичный ключ таблицы **id**, тип значения INT, поле **keyword**, тип значения string. Таблица содержит следующие записи: летние, зимние, всесезонные.

Справочная таблица **categoryList** включает в себя следующие поля:

Первичный ключ таблицы **id**, тип значения INT, поле **category**, тип значения string. Таблица содержит следующие записи: Активный гражданин, выборы, дороги и транспорт, жилищно-коммунальное хозяйство, здравоохранение, культура, общественное питание, пешеходная инфраструктура, религиозные объекты, социальная среда, строительство, торговля, безопасность, государственные услуги, досуг и отдых, забота о животных, земля и имущество, образование, органы государственной власти, предпринимательство, связь, справочная информация, территориальное деление, трудоустройство.

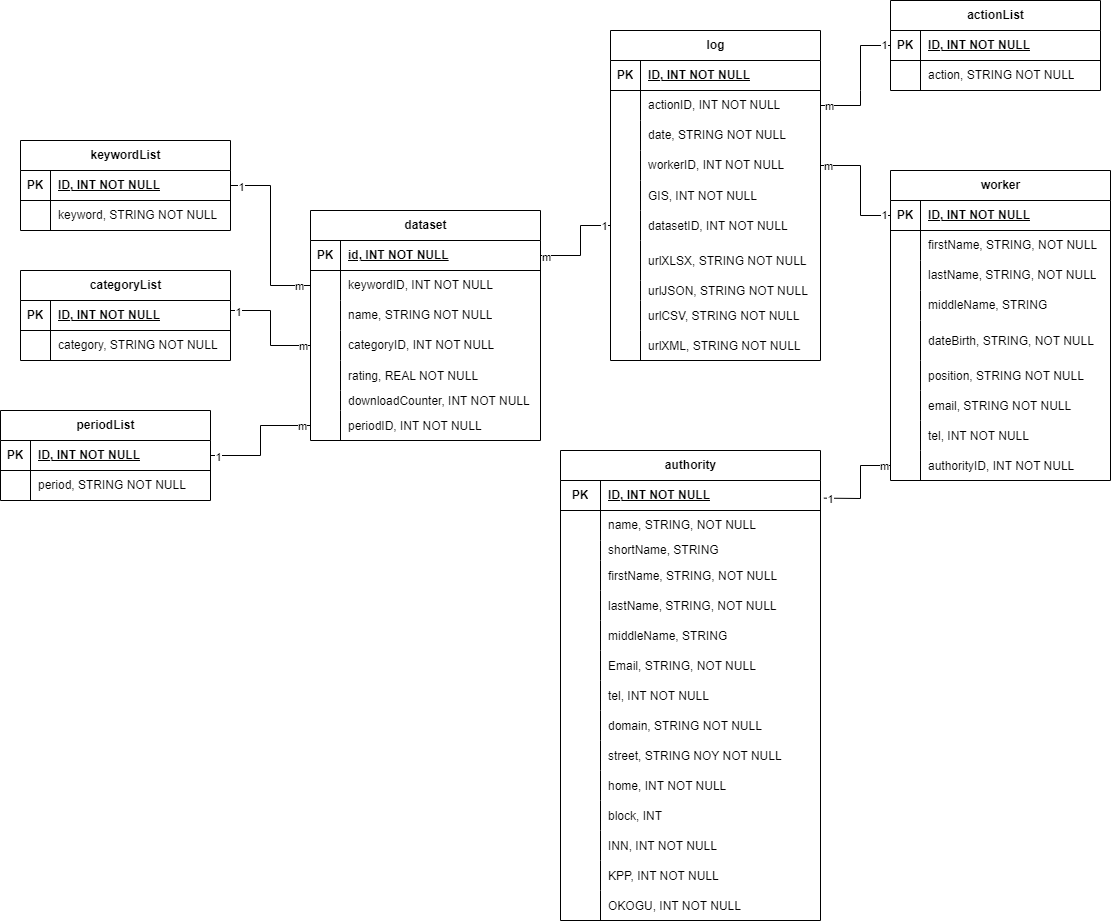
Справочная таблица **periodList** включает в себя следующие поля:

Первичный ключ таблицы **id**, тип значения INT, поле **period**, тип значения string. Таблица содержит следующие записи: ежемесячно, ежеквартально, ежегодно.

Справочная таблица **actionList** включает в себя следующие поля:

Первичный ключ таблицы **id**, тип значения INT, поле **action**, тип значения string. Таблица содержит следующие записи: опубликован, архивирован, изъят из публичного доступа.

Окончательный вид даталогической модели представлен на рисунке 16.



**Рисунок 16.** Даталогическая модель базы данных

РАЗДЕЛ 3. построение физической модели базы данных

3.1. Создание таблиц базы данных

Построение физической модели базы данных также включает в себя выбор конкретной системы управления базой данных (СУБД). Для построения в данной работе будет использована библиотека SQLite, которая является одной из самых широко-используемых СУБД в мире, которую используют такие крупные компании, как Adobe, Apple, Google и другие.[[5]](#footnote-5),[[6]](#footnote-6)

Первый этап построения физической модели включает в себя создание пустой базы данных на сервере СУБД. Помимо использования библиотеки sqlite3, для работы в среде jupyter-lab, будет использован python.

В результате выполнения программного кода, представленного в листинге 1, создается файл базы данных.

**Листинг 1.** Создание базы данных shumskiiIN.db

1. import sqlite3 # импортируем библиотеку по работе с базой данных
2. try:
3. sqlite\_connection = sqlite3.connect('shumskiiIN.db') #создаем переменную подключения к БД
4. cursor = sqlite\_connection.cursor() #создаем специальную переменную по работе с БД
5. print("База данных создана и успешно подключена к SQLite")
6. sqlite\_select\_query = "SELECT sqlite\_version();" #пишем запрос получение версии библиотеки sqlite
7. cursor.execute(sqlite\_select\_query) #выполняем запрос, после чего в буфере содержится текст результата
8. record = cursor.fetchall() #читаем все содержимое буфера в переменную record
9. print("Версия базы данных SQLite: ", record)
10. cursor.close() #закрываем курсор по работе с БД
11. except sqlite3.Error as error:
12. print("Ошибка при подключении к sqlite", error) # если при попытке подключения к БД возникли ошибки, выводим их
13. finally:
14. if (sqlite\_connection):
15. sqlite\_connection.close() #если соединение

База данных создана и успешно подключена к SQLite Версия базы данных SQLite: [('3.43.1',)]

Соединение с SQLite закрыто

После создания базы данных можно приступать к созданию и заполнению таблиц. Вначале целесообразно создать все справочные таблицы, поскольку они, как правило, не зависят от наличия - отсутствия других таблиц.

В результате выполнения программного кода, представленного в листинге 2, создаются таблицы **categoryList, actionList, keywordList, actionList**, все они имеют одинаковую структуру.

**Листинг 2.** Создание справочных таблиц

1. import sqlite3
2. try:
3. sqlite\_connection = sqlite3.connect('shumskiiIN.db')
4. # Создаем sql-запросы на создание таблиц, записываем для удобства их в переменные с одноименным названием
5. periodList = 'CREATE TABLE periodList (id INTEGER PRIMARY KEY NOT NULL, period VARCHAR(30) NOT NULL);'
6. categoryList = 'CREATE TABLE categoryList (id INTEGER PRIMARY KEY NOT NULL, category VARCHAR(30) NOT NULL);' actionList = 'CREATE TABLE actionList (id INTEGER PRIMARY KEY NOT NULL, action VARCHAR(30) NOT NULL);'
7. keywordList = 'CREATE TABLE keywordList (id INTEGER PRIMARY KEY NOT NULL, keyword VARCHAR(30) NOT NULL);'
8. cursor = sqlite\_connection.cursor()
9. print("База данных подключена к SQLite") cursor.execute(periodList)
10. sqlite\_connection.commit()
11. print("Таблица periodList создана")
12. cursor.execute(categoryList) sqlite\_connection.commit()
13. print("Таблица categoryList создана")
14. cursor.execute(actionList) sqlite\_connection.commit()
15. print("Таблица actionList создана")
16. cursor.execute(keywordList) sqlite\_connection.commit()
17. print("Таблица keywordList создана") cursor.close()
18. except sqlite3.Error as error:
19. print("Ошибка при подключении к sqlite", error)
20. finally:
21. if (sqlite\_connection):
22. sqlite\_connection.close()
23. print("Соединение с SQLite закрыто")

База данных подключена к SQLite

Таблица periodList создана

Таблица categoryList создана

Таблица actionList создана

Таблица keywordList создана

Соединение с SQLite закрыто

После создания справочных таблиц, необходимо реализовать запросы на заполнение справочных таблиц (листинги 3-6).

**Листинг 3.** Заполнение таблицы periodList

1. # Заполнение таблицы periodList
2. import sqlite3
3. try:
4. sqlite\_connection = sqlite3.connect('shumskiiIN.db')
5. filling\_periodList = '''INSERT INTO periodList (id, period) VALUES
6. (1, 'Ежемесячно'),
7. (2, 'Ежеквартально'),
8. (3, 'Ежегодно'); '''
9. cursor = sqlite\_connection.cursor() cursor.execute(filling\_periodList)
10. sqlite\_connection.commit()
11. cursor.close()
12. except sqlite3.Error as error:
13. print("Ошибка при подключении к sqlite", error)
14. finally:
15. if (sqlite\_connection):
16. sqlite\_connection.close()
17. print("Соединение с SQLite закрыто")

Соединение с SQLite закрыто

**Листинг 4.** Заполнение таблицы catregoryList

1. # Заполнение таблицы categoryList
2. import sqlite3
3. try:
4. sqlite\_connection = sqlite3.connect('shumskiiIN.db')
5. filling\_categoryList = '''INSERT INTO categoryList (id, category) VALUES (1, 'Активный гражданин'),
6. (2, 'Выборы'),
7. (3, 'Дороги и транспорт'),
8. (4, 'Жилищно-коммунальное хозяйство'),
9. (5, 'Здравоохранение'),
10. (6, 'Культура'),
11. (7, 'Общественное питание'),
12. (8, 'Пешеходная инфраструктура'),
13. (9, 'Религиозные объекты'),
14. (10, 'Социальная среда'),
15. (11, 'Строительство'),
16. (12, 'Торговля'),
17. (13, 'Безопасность'),
18. (14, 'Государственные услуги'), (15, 'Досуг и отдых'),
19. (16, 'Забота о животных'), (17, 'Земля и имущество'), (18, 'Образование'),
20. (19, 'Органы государственной власти'), (20, 'Предпринимательство'),
21. (21, 'Связь'),
22. (22, 'Справочная информация'),
23. (23, 'Территориальное деление'),
24. (24, 'Трудоустройство'); '''
25. cursor = sqlite\_connection.cursor() cursor.execute(filling\_categoryList)
26. sqlite\_connection.commit()
27. cursor.close()
28. except sqlite3.Error as error:
29. print("Ошибка при подключении к sqlite", error)
30. finally:
31. if (sqlite\_connection):
32. sqlite\_connection.close()
33. print("Соединение с SQLite закрыто")

Соединение с SQLite закрыто

**Листинг 5.** Заполнение таблицы **actionList**

1. # Заполнение таблицы actionList
2. import sqlite3
3. try:
4. sqlite\_connection = sqlite3.connect('shumskiiIN.db')
5. filling\_actionList = '''INSERT INTO actionList (id, action) VALUES
6. (1, 'Опубликован'),
7. (2, 'Архивирован'),
8. (3, 'Изъят из публичного доступа'); '''
9. cursor = sqlite\_connection.cursor() cursor.execute(filling\_actionList)
10. sqlite\_connection.commit()
11. cursor.close()
12. except sqlite3.Error as error:
13. print("Ошибка при подключении к sqlite", error)
14. finally:
15. if (sqlite\_connection):
16. sqlite\_connection.close()
17. print("Соединение с SQLite закрыто")

Соединение с SQLite закрыто

**Листинг 6.** Заполнение таблицы **keywordList**

1. # Заполнение таблицы keywordList
2. import sqlite3
3. try:
4. sqlite\_connection = sqlite3.connect('shumskiiIN.db')
5. filling\_keywordList = '''INSERT INTO keywordList (id, keyword) VALUES (1, 'Зимние'),
6. (2, 'Летние'),
7. (3, 'Всесезонные'); '''
8. cursor = sqlite\_connection.cursor() cursor.execute(filling\_keywordList)
9. sqlite\_connection.commit()
10. cursor.close()
11. except sqlite3.Error as error:
12. print("Ошибка при подключении к sqlite", error)
13. finally:
14. if (sqlite\_connection):
15. sqlite\_connection.close()
16. print("Соединение с SQLite закрыто")

Соединение с SQLite закрыто

Итак, справочные таблицы созданы и заполнены необходимой информацией. Для просмотра результата заполнения справочных таблиц используем функцию, представленную в листингах 7-8, с использованием дополнительных библиотек numpu и pandas.[[7]](#footnote-7),[[8]](#footnote-8) Результат выполнения запроса, представленного в листинге 8 представлен в Приложении А.

**Листинг 7.** Реализация функции запроса на чтение базы данных

1. #Функция реализует запрос на чтение к базе данных
2. def query\_sql (db, query):
3. sqlite\_connection = sqlite3.connect(db)
4. cursor = sqlite\_connection.cursor()
5. cursor.execute(query)
6. sqlite\_connection.commit()
7. rows = cursor.fetchall()
8. names = tuple(map(lambda x: x[0], cursor.description))
9. cursor.close()
10. sqlite\_connection.close()
11. import pandas as pd
12. import numpy as np
13. aa=np.array(rows)
14. dd=pd.DataFrame(aa, columns=names)
15. display(dd)
16. **Листинг 7.** Реализация функции запроса на чтение базы данных

**Листинг 8**. SQL-запрос на просмотр содержимого справочных таблиц

1. query\_sql('shumskiiIN.db','SELECT \* FROM periodList;')
2. query\_sql('shumskiiIN.db','SELECT \* FROM actionList;')
3. query\_sql('shumskiiIN.db','SELECT \* FROM categoryList;')
4. query\_sql('shumskiiIN.db','SELECT \* FROM keywordList;')

После создания справочных таблиц целесообразно приступить к заполнению так называемых целевых таблиц, которые содержат информацию, на хранение которой и нацелена база данных. Поскольку заполнение целевых таблиц происходит инкрементно, для удобства дальнейшей работы с базой данных, будут использованы экранные формы, созданные при использовании дополнительной библиотеки ipywidgets.[[9]](#footnote-9)

Перед созданием запросов на создание целевых таблиц, необходимо создать обработчик событий, для корректного выполнения запросов типа SELECT, INSERT, не выводя дополнительных сообщений. Пример данного обработчика представлен в листинге 9.

**Листинг 9.** Обработчик событий

1. def query\_sql (db, query, visible = True):
2. import sqlite3
3. sqlite\_connection = sqlite3.connect(db)
4. cursor = sqlite\_connection.cursor()
5. cursor.execute(query)
6. sqlite\_connection.commit()
7. if visible:
8. rows = cursor.fetchall()
9. names = tuple(map(lambda x: x[0], cursor.description))
10. import pandas as pd
11. import numpy as np
12. aa=np.array(rows)
13. dd=pd.DataFrame(aa, columns=names)
14. return dd
15. cursor.close()
16. sqlite\_connection.close()

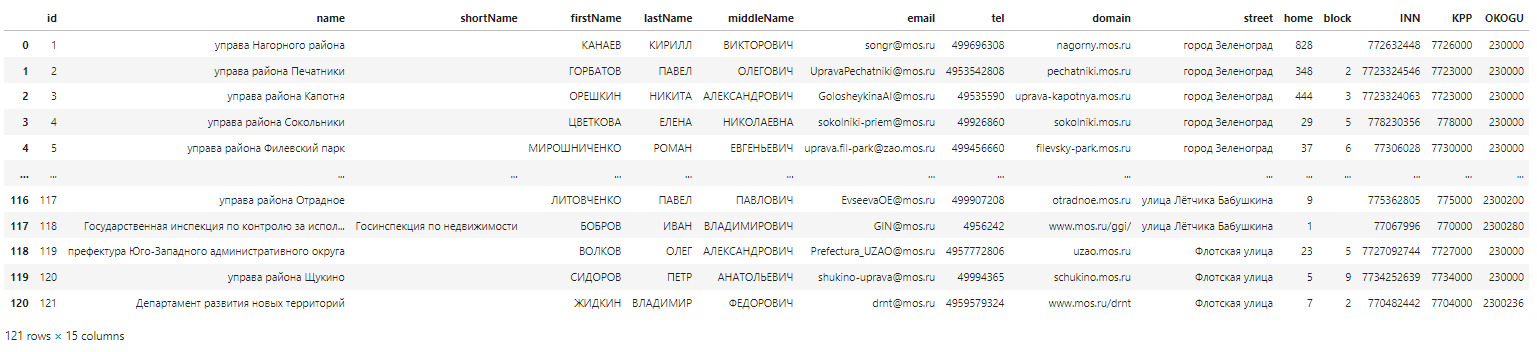
После этого, создадим таблицу authority, так как она не требует дополнительного подключения справочных таблиц. В результате выполнения кода, представленного в листинге 10, создается незаполненная таблица.

**Листинг 10.** Запрос на создание таблицы authority

1. #создание таблицы authority
2. query\_sql('shumskiiIN.db','''
3. CREATE TABLE authority (id INTEGER PRIMARY KEY NOT NULL,
4. name VARCHAR(50) NOT NULL,
5. shortName VARCHAR(30),
6. firstName VARCHAR(15) NOT NULL, lastName VARCHAR(15) NOT NULL,
7. middleName VARCHAR(20),
8. email VARCHAR(30) NOT NULL, tel INTEGER NOT NULL,
9. domain VARCHAR(30) NOT NULL, street VARCHAR(30) NOT NULL, home INTEGER NOT NULL,
10. block INTEGER,
11. INN INTEGER NOT NULL, KPP INTEGER NOT NULL,
12. OKOGU INTEGER NOT NULL);
13. ''', False)

После создания необходимой таблицы, приступим к написанию экранной формы. В результате выполнения вышеуказанного кода, представленного в Приложении Б, создается экранная форма, при заполнении которой, соответствующие данные вносятся в таблицу authority.

Заполним целевую таблицу при помощи экранной формы.

**Рисунок 17.** Результат заполнения таблицы authority

При заполнении остальных таблиц требуется делать подстановки значений из справочных таблиц. Покажем реализацию на примере таблицы worker. Для подстановки используем виджет widgets.Dropdown, но вначале создадим данную таблицу. В результате работы программного кода, представленного в листинге 11, создается пустая таблица worker.

**Листинг 11.** Создание таблицы worker

1. #создание табдицы worker
2. query\_sql('shumskiiIN.db','''
3. CREATE TABLE worker (id INTEGER PRIMARY KEY NOT NULL,
4. firstName VARCHAR(15) NOT NULL, lastName VARCHAR(15) NOT NULL,
5. middleName VARCHAR(20),
6. dateBirth VARCHAR(10) NOT NULL, position VARCHAR(30) NOT NULL, email VARCHAR(30) NOT NULL,
7. tel INTEGER NOT NULL,
8. authorityID INTEGER NOT NULL REFERENCES authority (id) ON DELETE NO ACTION ON UPDATE NO ACTION);
9. ''', False)

После создания целевой таблиц, можно приступить к созданию экранной формы. Программный код и экранная форма представлены в Приложении В.

Заполним целевую таблицу при помощи экранной формы.

**Рисунок 18.** Результат заполнения таблицы worker

По аналогии с таблицей worker, создадим таблицы, экранные формы и заполним таблицы log и dataset (Приложение Г).

Таким образом, все таблицы базы данных построены и заполнены.

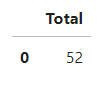
3.2. Построение типовых запросов на выборку данных

Для демонстрации работы с базой данных построим ряд типовых запросов.

Для реализации запроса типа LIKE, рассчитаем количество департаментов, наборы данных которых скачали более 2000 раз. Результат выполнения программного кода, представленного в листинге 12, представлен на рисунке 19.

**Листинг 12.** Запрос типа LIKE

1. #Расчитать количество департементов, чьи наборы скачали более 2000 раз
2. query\_sql('shumskiiin.db', '''SELECT COUNT(authority.id) as 'Total'
3. FROM authority, worker, dataSet, log
4. WHERE authority.name LIKE "%Департамент%" AND
5. worker.authorityID = authority.ID AND
6. log.workerID = worker.id AND
7. log.dataSetID = dataSet.id AND
8. dataSet.downloadCounter > 2000;
9. '''
10. )

**Рисунок 19.** Результат выполнения запроса типа LIKE

Для реализации запроса типа BETWEEN, который используется для выборки данных из диапазона с указанием начала и конца этого диапазона, получим список ответственным сотрудников, наборы которых имеют рейтинг выше 7,00. Результат выполнения программного кода, представленного в листинге 13, представлен на рисунке 20.

**Листинг 13.** Запрос типа BETWEEN

1. #Получить список сотрудников, чьи наборы данных имеют рейтинг от 7.00 до 10.00
2. query\_sql('shumskiiin.db', '''SELECT worker.lastName, worker.firstName, worker.middleName
3. FROM worker, dataSet, log
4. WHERE worker.id = log.workerID AND
5. log.dataSetID = dataSet.id AND
6. dataSet.rating BETWEEN 7.0 AND 10.0;
7. '''
8. )

**Рисунок 20.** Результат выполнения запроса типа BETWEEN

Для реализации запроса типа IN, который позволяет отобрать элементы из заданного списка, получим список наборов, с ключевым словом «летний» или «зимний». Результат выполнения программного кода, представленного в листинге 14, представлен на рисунке 21.

**Листинг 14.** Запрос типа IN

1. #Получить список наборов, которые содержат ключевые слова летний или зимний
2. query\_sql('shumskiiin.db', '''SELECT dataSet.name, keywordList.keyword
3. FROM dataSet, keywordList
4. WHERE dataSet.keywordID = keywordList.id AND
5. keywordList.keyword IN ('Зимние', 'Летние');
6. '''
7. )

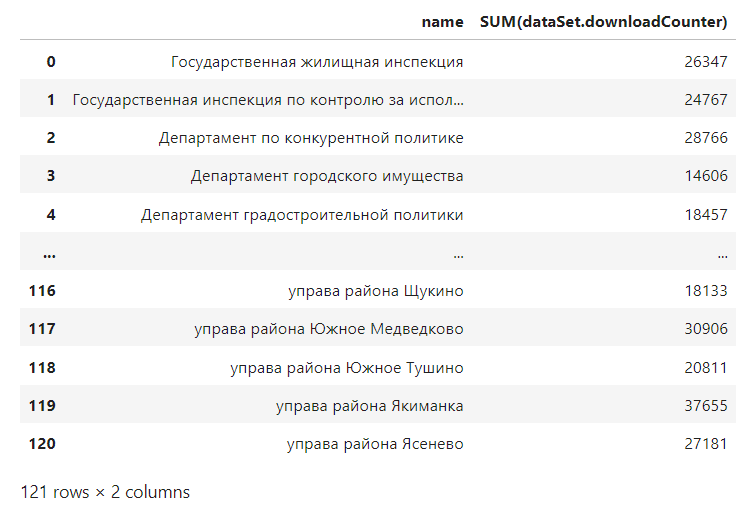
**Рисунок 21.** Результат выполнения запроса типа IN

Для реализации запроса типа SUM, получим суммарное количество скачиваний наборов в разрезе органов исполнительной власти. Результат выполнения программного кода, представленного в листинге 15, представлен на рисунке 22.

**Листинг 15.** Запрос типа SUM

1. #Рассчитать суммарное кол-во скачиваний в разрезе ОИВ
2. query\_sql('shumskiiin.db', '''SELECT authority.name, SUM(dataSet.downloadCounter)
3. FROM authority, dataSet, log, worker
4. WHERE authority.id = worker.authorityID AND
5. worker.id = log.workerID AND
6. log.dataSetID = dataSet.id
7. GROUP BY authority.name;
8. '''
9. )

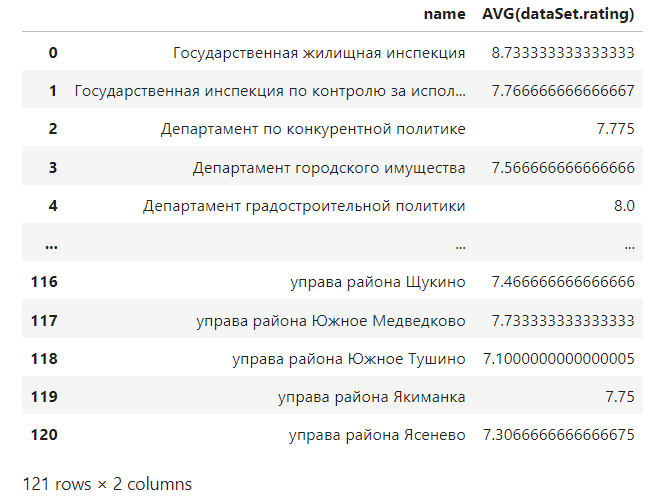
**Рисунок 22.** Результат выполнения запроса типа SUM

****

Для реализации запроса типа AVG, рассчитаем средний рейтинг наборов в разрезе органов исполнительной власти. Результат выполнения программного кода, представленного в листинге 16, представлен на рисунке 23.

**Листинг 16.** Запрос типа AVG

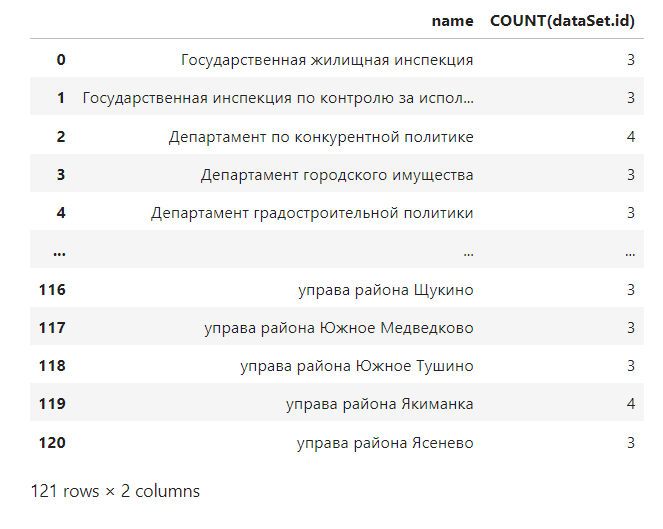
1. #Рассчитать средний рейтинг наборов в разрезе ОИВ
2. query\_sql('shumskiiin.db', '''SELECT authority.name, AVG(dataSet.rating)
3. FROM authority, dataSet, log, worker
4. WHERE authority.id = worker.authorityID AND
5. worker.id = log.workerID AND
6. log.dataSetID = dataSet.id
7. GROUP BY authority.name;
8. '''
9. )

**Рисунок 23.** Результат выполнения запроса типа AVG

Для реализации запроса типа COUNT, рассчитаем количество наборов на каждый орган исполнительной власти. Результат выполнения программного кода, представленного в листинге 17, представлен на рисунке 24.

**Листинг 17.** Запрос типа COUNT

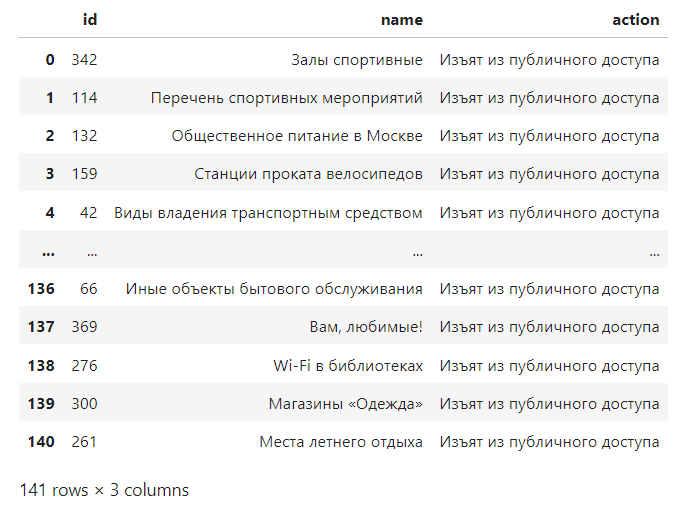
1. #Рассчитать кол-во наборов на каждый ОИВ
2. query\_sql('shumskiiin.db', '''SELECT authority.name, COUNT(dataSet.id)
3. FROM authority, dataSet, log, worker
4. WHERE authority.id = worker.authorityID AND
5. worker.id = log.workerID AND
6. log.dataSetID = dataSet.id
7. GROUP BY authority.name;
8. '''
9. )

**Рисунок 24.** Результат выполнения запроса типа COUNT

Для реализации запроса типа WHERE, получим список наборов данных, которых имеют статус «изъят из публичного доступа». Результат выполнения программного кода, представленного в листинге 18, представлен на рисунке 25.

**Листинг 18.** Запрос типа WHERE

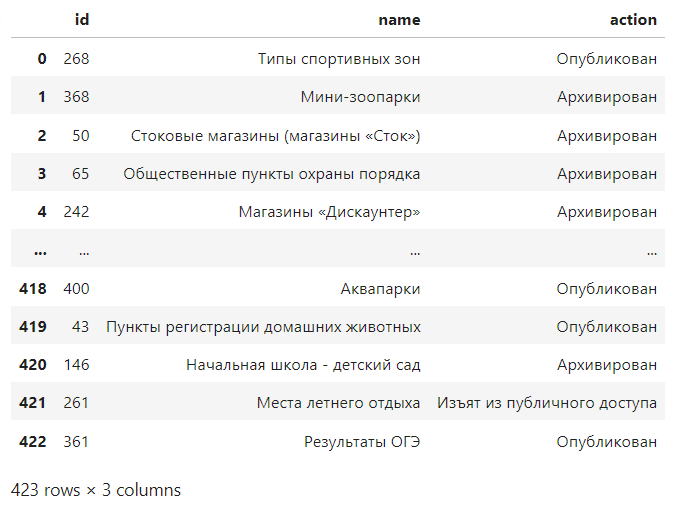
1. #Вывести все наборы со статусом "изъят из публичного доступа"
2. query\_sql('shumskiiin.db', '''SELECT dataSet.id, dataSet.name, actionList.action
3. FROM dataSet, log, actionList
4. WHERE dataSet.id = log.dataSetID AND
5. log.actionID = actionList.id AND
6. actionList.action = 'Изъят из публичного доступа';
7. '''
8. )

**Рисунок 25.** Результат выполнения запроса типа WHERE

Также получим список наборов и их статус. Результат выполнения программного кода, представленного в листинге 19, представлен на рисунке 26.

**Листинг 19.** Запрос на получение списка наборов и их статусов

1. #Название набора и его статус
2. query\_sql('shumskiiin.db', '''SELECT dataSet.id, dataSet.name, actionList.action
3. FROM dataSet, log, actionList
4. WHERE log.dataSetID = dataSet.id AND
5. log.actionID = actionList.id
6. ;
7. '''
8. )

**Рисунок 26.** Результат выполнения запрос на получение списка наборов и их статусов

На примере этого же запроса посмотрим на реализации с помощью операторов INNER JOIN. Результат выполнения программного кода, представленного в листинге 20, представлен на рисунке 27.

**Листинг 20.** Запрос типа INNER JOIN

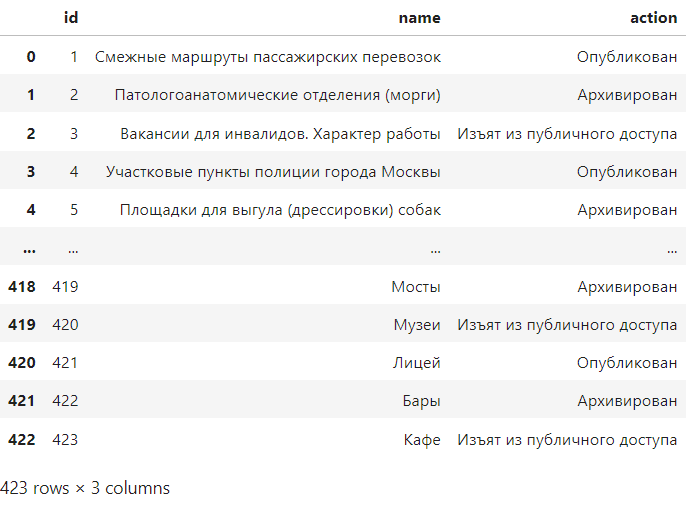
1. query\_sql('shumskiiin.db', '''SELECT dataSet.id, dataSet.name, actionList.action
2. FROM dataSet
3. INNER JOIN log
4. ON dataSet.id = log.dataSetID
5. INNER JOIN actionList
6. ON log.actionID = actionList.id
7. ;
8. '''
9. )

**Рисунок 27.** Результат выполнения запроса типа INNER JOIN

На примере этого же запроса посмотрим на реализации с помощью операторов LEFT JOIN. Результат выполнения программного кода, представленного в листинге 21, представлен на рисунке 28.

**Листинг 21.** Запрос типа LEFT JOIN

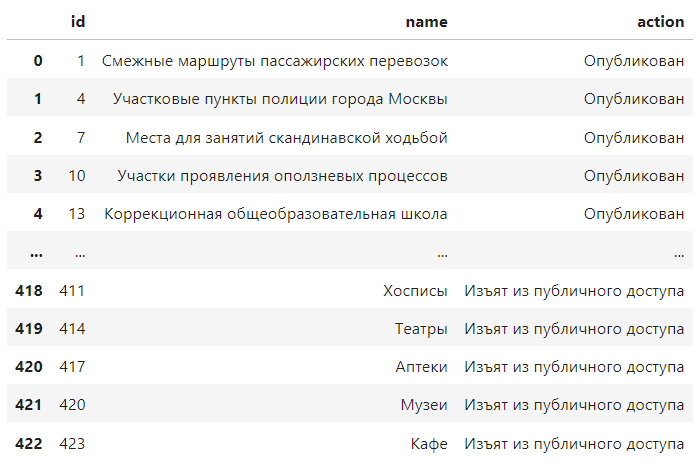
1. query\_sql('shumskiiin.db', '''SELECT dataSet.id, dataSet.name, actionList.action
2. FROM dataSet
3. LEFT JOIN log
4. ON dataSet.id = log.dataSetID
5. LEFT JOIN actionList
6. ON log.actionID = actionList.id
7. ;
8. '''
9. )

**Рисунок 28.** Результат выполнения запроса типа LEFT JOIN

Заметим также, что запросы типа RIGHT JOIN sqlite не поддерживает в чистом виде. Для этого просто изменим порядок таблиц. Результат выполнения программного кода, представленного в листинге 22, представлен на рисунке 29.

**Листинг 29.** Результат выполнения запроса типа LEFT JOIN с измененным порядком таблиц

1. query\_sql('shumskiiin.db', '''SELECT dataSet.id, dataSet.name, actionList.action
2. FROM actionList
3. LEFT JOIN log
4. ON log.actionID = actionList.id
5. LEFT JOIN dataSet
6. ON dataSet.id = log.dataSetID
7. ;
8. '''
9. )

**Рисунок 28.** Результат выполнения запроса типа LEFT JOIN с измененным порядком таблиц

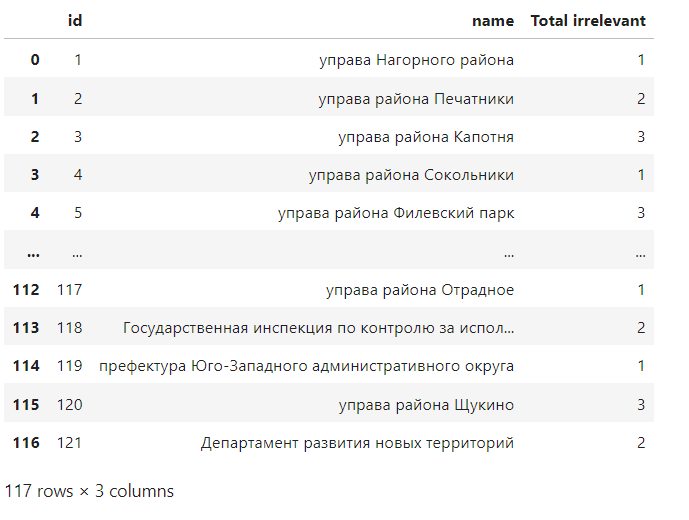
Исходя из результатов выполнения запросом типа INNER, LEFT, RIGTH JOIN, можно сделать вывод о том, что результат остается одинаковым, поскольку каждой записи из одной таблицы соответствует запись в другой таблице.

Подзапросы обычно используются, когда требуется выполнить более сложный запрос, включающий в себя информацию, которая может быть получена из другого запроса. Рассмотрим два таких примера.

Рассчитаем отношение неактуальных наборов данных (имеют статус «архивирован» или «изъят из публичного доступа») в разрезе органов исполнительной власти. Для этого сначала получим количество неактуальных наборов данных в разрезе органов исполнительной власти. Результат выполнения программного кода, представленного в листинге 23, представлен на рисунке 30.

**Листинг 23.** Запрос на получение количества неактуальных наборов данных в разрезе органов исполнительной власти

1. query\_sql('shumskiiin.db', '''SELECT authority.id, authority.name, COUNT(dataSet.id) as "Total irrelevant"
2. FROM actionList
3. LEFT JOIN log
4. ON log.actionID = actionList.id
5. LEFT JOIN dataSet
6. ON dataSet.id = log.dataSetID
7. LEFT JOIN worker
8. ON log.workerID = worker.ID
9. LEFT JOIN authority
10. ON worker.authorityID = authority.id
11. WHERE
12. actionList.action IN ('Архивирован', 'Изъят из публичного доступа')
13. GROUP BY authority.id, authority.name;
14. '''
15. )

**Рисунок 30.** Результат выполнения запроса на получение количества неактуальных наборов данных в разрезе органов исполнительной власти

Далее получим количество наборов данных в разрезе органов исполнительной власти. Результат выполнения программного кода, представленного в листинге 24, представлен на рисунке 31.

**Листинг 24.** Запрос на получение количества наборов данных в разрезе органов исполнительной власти

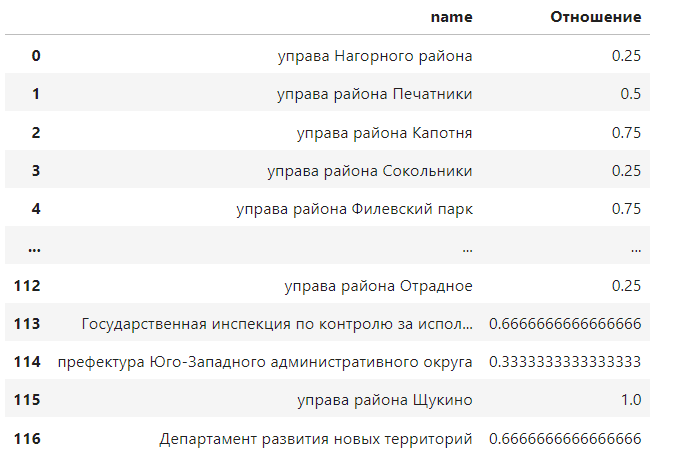
1. query\_sql('shumskiiin.db', '''SELECT authority.id, authority.name, COUNT(dataSet.id) as "Total dataSet"
2. FROM actionList
3. LEFT JOIN log
4. ON log.actionID = actionList.id
5. LEFT JOIN dataSet
6. ON dataSet.id = log.dataSetID
7. LEFT JOIN worker
8. ON log.workerID = worker.ID
9. LEFT JOIN authority
10. ON worker.authorityID = authority.id
11. GROUP BY authority.id, authority.name;
12. '''
13. )

**Рисунок 31.** Результат выполнения запроса на получение количества наборов данных в разрезе органов исполнительной власти

Объединим теперь эти два запроса в один общий запрос (заметим, что при делении целых чисел sqlite выполняет целочисленное деление, поэтому предварительно необходимо или числитель, или знаменатель преобразовать в дробное число, это можно сделать умножением его на 1.0). Результат выполнения программного кода, представленного в листинге 25, представлен на рисунке 32.

**Листинг 25.** Запрос на получение отношения неактуальных наборов данных в разрезе ОИВ

1. query\_sql('shumskiiin.db', '''SELECT a.name, 1.0 \*`Total irrelevant` / `Total dataSet` as 'Отношение'
2. FROM
3. (SELECT authority.id, authority.name, COUNT(dataSet.id) as "Total irrelevant"
4. FROM actionList
5. LEFT JOIN log
6. ON log.actionID = actionList.id
7. LEFT JOIN dataSet
8. ON dataSet.id = log.dataSetID
9. LEFT JOIN worker
10. ON log.workerID = worker.ID
11. LEFT JOIN authority
12. ON worker.authorityID = authority.id
13. WHERE
14. actionList.action IN ('Архивирован', 'Изъят из публичного доступа')
15. GROUP BY authority.id, authority.name) a,
16. (SELECT authority.id, authority.name, COUNT(dataSet.id) as "Total dataSet"
17. FROM actionList
18. LEFT JOIN log
19. ON log.actionID = actionList.id
20. LEFT JOIN dataSet
21. ON dataSet.id = log.dataSetID
22. LEFT JOIN worker
23. ON log.workerID = worker.ID
24. LEFT JOIN authority
25. ON worker.authorityID = authority.id
26. GROUP BY authority.id, authority.name) b
27. WHERE a.id = b.id;
28. '''
29. )

**Рисунок 32.** Результат выполнения запроса на получение отношения неактуальных наборов данных в разрезе ОИВ

Аналогично предыдущим запросам, рассчитаем отношение неактуальных наборов данных (имеют статус «архивирован» или «изъят из публичного доступа») в разрезе сотрудников. Для этого также поочередно выполним каждый из запросов, а потом объединим в один общий запрос. Программный код запросов, а также промежуточные результаты представлены в Приложении Е.

ЗАКЛЮЧЕНИЕ

В работе была поставлена цель – разработать и построить физическую модель базы данных по учету актуализации информации на портале открытых данных Правительства Москвы, для достижения которой необходимо решить следующие задачи: провести анализ портала открытых данных Правительства Москвы, построить даталогическую и концептуальные модели базы данных, а также разработать физическую модель базы данных, с построением типовых запросов на выборку.

Для достижения цели работы в первой главе был проведен анализ нормативно-правовых актов, а также работы портала открытых данных, связанных с содержанием наборов данных, а также способом их публикации.

В работе произведена нормализация отношений, заключающаяся в добавлении новых атрибутов, для соблюдения атомарности, определении связей между сущностями, добавления справочных таблиц.

На основе представленной даталогической модели разработана физическая модель, с экранными формами, которая позволяет не только вносить данные в базу данных через графический интерфейс, но и строить различные типовые запросы на выборку данных, необходимые для дальнейшего анализа.

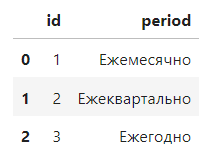
На основании вышеизложенного можно считать, что задачи, поставленные в курсовой работе, решены и цель работы достигнута.

СПИСОК ЛИТЕРАТУРЫ

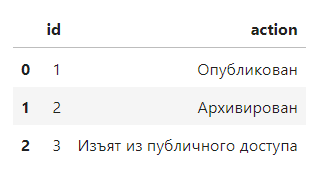
1. Федеральный закон Российской Федерации от 27 июля 2006 г. N 149-ФЗ Об информации, информационных технологиях и о защите информации // СПС КонсультантПлюс
2. Федеральный закон от 09.02.2009 № 8-ФЗ «Об обеспечении доступа к информации о деятельности государственных органов и органов местного самоуправления» // СПС КонсультантПлюс
3. Распоряжение Правительства РФ от 30.01.2014 № 93-р «Об утверждении концепции открытости федеральных органов исполнительной власти» // Правительство РФ URL: http://government.ru/docs/10122/ (дата обращения: 06.01.2024).
4. Постановление Правительства Москвы № 187-ПП от 02.04.2013 «Об информационных системах, обеспечивающих деятельность Открытого правительства города Москвы» // СПС КонсультантПлюс
5. Постановление Правительства Москвы № 234-ПП от 15.04.2013 «Об интерактивном взаимодействии органов исполнительной власти города Москвы с населением города Москвы» // СПС КонсультантПлюс
6. Распоряжение Правительства Москвы №64-16-426/13 «Об утверждении регламента актуализации открытых данных в автоматизированной системе «Общегородской платформы открытых данных».
7. Распоряжение Правительства Москвы об утверждении регламента обработки открытых данных в автоматизированной информационной системе "Общегородская платформа открытых данных" // Портал открытых данных Правительства Москвы URL: https://data.mos.ru/documents (дата обращения: 06.01.2024).
8. Единые требования к размещению и наполнению подразделов официальных сайтов федеральных государственных органов, посвященных вопросам противодействия коррупции, утвержденные Минтрудом России от 26 ноября 2012 г. // СПС КонсультантПлюс
9. Методические рекомендации по публикации открытых данных государственными органами и органами местного самоуправления и техническими требованиями к публикации открытых данных» // СПС КонсультантПлюс
10. Регламент ведения и актуализации сезонных наборов данных для публикации на общегородских порталах Портал открытых данных Правительства Москвы URL: https://data.mos.ru/documents (дата обращения: 06.01.2024).
11. Регламент обработки сообщений граждан, поступающих в систему обработки электронных сообщений граждан через Единую городскую автоматизированную систему обеспечения поддержки деятельности Открытого правительства города Москвы Портал открытых данных Правительства Москвы URL: https://data.mos.ru/documents (дата обращения: 06.01.2024).
12. Регламент раскрытия и актуализации наборов данных в Единой городской автоматизированной системе обеспечения поддержки деятельности Открытого правительства города Москвы Портал открытых данных Правительства Москвы URL: https://data.mos.ru/documents (дата обращения: 06.01.2024).
13. Родионов А. Н. Критерии качества даталогических схем. Полнота моделей данных // Вестник Хабаровского государственного университета экономики и права. 2011. №2. URL: https://cyberleninka.ru/article/n/kriterii-kachestva-datalogicheskih-shem-polnota-modeley-dannyh (дата обращения: 07.01.2024).
14. Смирнов М.В., Толмасов Р.С. ГРАФИЧЕСКАЯ НОТАЦИЯ МОДЕЛИРОВАНИЯ ДОКУМЕНТНЫХ БАЗ ДАННЫХ // Открытое образование. 2021. №5. URL: https://cyberleninka.ru/article/n/graficheskaya-notatsiya-modelirovaniya-dokumentnyh-baz-dannyh (дата обращения: 07.01.2024).
15. Базы данных // Oracle.com URL: https://www.oracle.com/cis/database/what-is-a-relational-database/ (дата обращения: 06.01.2024).
16. Данные // Портал открытых данных Правительства Москвы URL: https://data.mos.ru/opendata (дата обращения: 06.01.2024).
17. Документация // Портал открытых данных Правительства Москвы URL: https://data.mos.ru/documents (дата обращения: 06.01.2024).
18. About SQLite // SQLite.org URL: https://www.sqlite.org/about.html (дата обращения: 07.01.2024).
19. Documentation // SQLite.org URL: https://www.sqlite.org/docs.html (дата обращения: 06.01.2024).
20. FAIR Principles // go-fair.org URL: https://www.go-fair.org/fair-principles (дата обращения: 06.01.2024).
21. International Council for Science World Data System // re3data.org URL: https://www.re3data.org/search (дата обращения: 06.01.2024).
22. NumPy documentation // NumPy URL: https://numpy.org/doc/stable/ (дата обращения: 07.01.2024).
23. Pandas documentation // pandas URL: https://pandas.pydata.org/docs/ (дата обращения: 07.01.2024).
24. Well-Known Users of SQLite // SQLite.org URL: https://www.sqlite.org/famous.html (дата обращения: 07.01.2024).
25. Widget List // Ipywidgets URL: https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20List.html (дата обращения: 07.01.2024).

приложение а

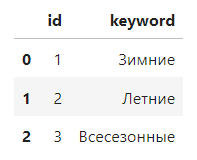
Содержимое справочных таблиц



**Рисунок 1.** Содержимое справочной таблицы **periodList**

****

**Рисунок 2.** Содержимое справочной таблицы **actiomList**



**Рисунок 3.** Содержимое справочной таблицы **keywordList**

****

**Рисунок 3.** Содержимое справочной таблицы **categoryList**

приложение Б

Создание экранной формы для таблицы authority

**Листинг 1.** Программный код

1. #authority-form
2. **import** ipywidgets **as** widgets #Подключаем библиотеку виджетов
3. **from** IPython.display **import** display # импортируем функцию для визуализации виджетов
4. style **=** {'description\_width': 'initial'} #Делаем установку ширины поля discription под размер текста в нем
5. id **=** widgets**.**IntText(value**=**1, description**=**'id:', disabled**=False**)
6. name **=** widgets**.**Text(value**=**'Название ОИВ', description**=**'Наименование:', disabled**=False**)
7. shortName **=** widgets**.**Text(value**=**'Сокращенное название', description**=**'Сокращенное наименование:', disabled**=False**) lastName **=** widgets**.**Text(value**=**'Иванов', description**=**'Фамилия:', disabled**=False**)
8. firstName **=** widgets**.**Text(value**=**'Иван', description**=**'Имя:', disabled**=False**)
9. middleName **=** widgets**.**Text(value**=**'Иванович', description**=**'Отчество:', disabled**=False**) email **=** widgets**.**Text(value**=**'sample@mos.ru', description**=**'E-mail:', disabled**=False**)
10. tel **=** widgets**.**IntText(value**=**79123456789, description**=**'Номер телефона:', disabled**=False**, style**=**style) domain **=** widgets**.**Text(value**=**'mos.ru', description**=**'Домен:', disabled**=False**)
11. street **=** widgets**.**Text(value**=**'Тверская', description**=**'Улица:', disabled**=False**)
12. home **=** widgets**.**IntText(value**=**13, description**=**'Номер дома:', disabled**=False**, style**=**style)
13. block **=** widgets**.**IntText(value**=**1, description**=**'Строение/корпус:', disabled**=False**, style**=**style) INN **=** widgets**.**IntText(value**=**770000000000, description**=**'ИНН:', disabled**=False**)
14. KPP **=** widgets**.**IntText(value**=**770000000, description**=**'КПП:', disabled**=False**)
15. OKOGU **=** widgets**.**IntText(value**=**770000000, description**=**'ОКГУ:', disabled**=False**)
16. button **=** widgets**.**Button(description**=**'Записать в БД', disabled**=False**, button\_style**=**'')
17. print('Заполнение таблицы authority')
18. display(id, name, shortName, lastName, firstName, middleName, email, tel, domain, street, home, block, INN, KPP, OKOGU, button)
19. **def** button\_clicked(b):
20. query **=** '''INSERT INTO authority (id, name, shortName, lastName, firstName, middleName, email, tel, domain, street, home, block, INN, KPP, OKOGU) VALUES '''
21. query **+=** '(' **+** str(id**.**value) **+** ',' **+** '\''**+** name**.**value **+** '\'' **+** ',' **+** '\''**+** shortName**.**value **+** '\'' **+** ',' query **+=** '\''**+** lastName**.**value **+** '\'' **+** ',' **+** '\''**+** firstName**.**value **+** '\'' **+** ','
22. query **+=** '\'' **+** middleName**.**value **+** '\'' **+** ',' **+** '\''**+** email**.**value **+** '\'' **+** ',' **+** str(tel**.**value) **+** ','
23. query **+=** '\'' **+** domain**.**value **+** '\'' **+** ',' **+** '\''**+** street**.**value **+** '\'' **+** ',' **+** str(home**.**value) **+** ','
24. query **+=** str(block**.**value) **+** ',' **+** str(INN**.**value) **+** ',' **+** str(KPP**.**value) **+** ',' **+** str(OKOGU**.**value) **+** ');' print(query\_sql)
25. query\_sql('shumskiiIN.db', query, **False**)
26. button**.**on\_click(button\_clicked)

Заполнение таблицы authority

IntText(value=1, description='id:')

Text(value='Название ОИВ', description='Наименование:')

Text(value='Сокращенное название', description='Сокращенное наименование:') Text(value='Иванов', description='Фамилия:')

Text(value='Иван', description='Имя:')

Text(value='Иванович', description='Отчество:')

Text(value='sample@mos.ru', description='E-mail:')

IntText(value=79123456789, description='Номер телефона:', style=DescriptionStyle(description\_width='initial')) Text(value='mos.ru', description='Домен:')

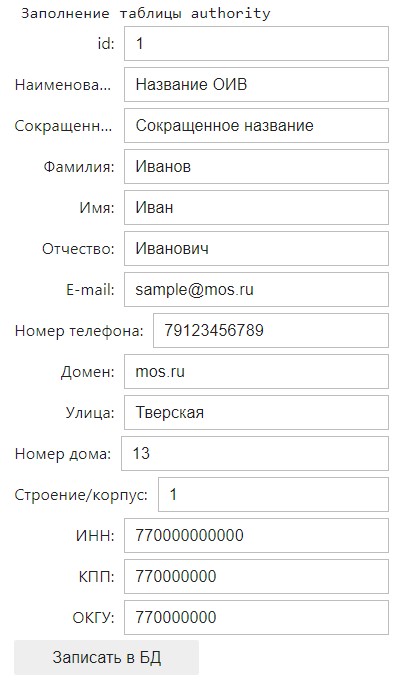
Text(value='Тверская', description='Улица:')

IntText(value=13, description='Номер дома:', style=DescriptionStyle(description\_width='initial'))

IntText(value=1, description='Строение/корпус:', style=DescriptionStyle(description\_width='initial')) IntText(value=770000000000, description='ИНН:')

IntText(value=770000000, description='КПП:') IntText(value=770000000, description='ОКГУ:')

Button(description='Записать в БД', style=ButtonStyle())

**Рисунок 1.** Экранная форма таблицы authority

приложение в

Создание экранной формы для таблицы worker

**Листинг 1.** Программный код

1. #worker-form
2. import ipywidgets as widgets #Подключаем библиотеку виджетов
3. from IPython.display import display # импортируем функцию для визуализации виджетов
4. style = {'description\_width': 'initial'} #Делаем установку ширины поля discription под размер текста в нем
5. id = widgets.IntText(value=1, description='id:', disabled=False)
6. lastName = widgets.Text(value='Иванов', description='Фамилия:', disabled=False) firstName = widgets.Text(value='Иван', description='Имя:', disabled=False)
7. middleName = widgets.Text(value='Иванович', description='Отчество:', disabled=False)
8. dateBirth = widgets.DatePicker(description='Дата рождения:', disabled=False, style=style) email = widgets.Text(value='sample@mos.ru', description='E-mail:', disabled=False)
9. tel = widgets.IntText(value=79123456789, description='Номер телефона:', disabled=False, style=style) R=query\_sql('shumskiiIN.db','select \* from authority;')
10. authorityID=widgets.Dropdown(
11. options=[tuple([R.iloc[i][1], int(R.iloc[i][0])]) for i in range(len(R['id']))], value = 1,
12. description='Организация:', disabled=False
13. )
14. position = widgets.Text(value='Ведущий специалист', description='Должность:', disabled=False)
15. button = widgets.Button(description='Записать в БД', disabled=False, button\_style='') print('Заполнение таблицы worker')
16. display(id, lastName, firstName, middleName, dateBirth, email, tel, authorityID, position, button)
17. def button\_clicked(b):
18. query = '''INSERT INTO worker (id, lastName, firstName, middleName, dateBirth, email, tel, authorityID, position) VALUES '''
19. query += '(' + str(id.value) + ',' + '\''+ lastName.value + '\'' + ',' + '\''+ firstName.value + '\'' + ',' query += '\'' + middleName.value + '\'' + ',' + str(dateBirth.value) + ',' + '\''+ email.value + '\'' + ',' query += str(tel.value) + ',' + str(authorityID.value) + ',' + '\''+ position.value + '\');'
20. print(query\_sql)
21. query\_sql('shumskiiIN.db', query, False)
22. button.on\_click(button\_clicked)

IntText(value=1, description='id:')

Text(value='Иванов', description='Фамилия:') Text(value='Иван', description='Имя:')

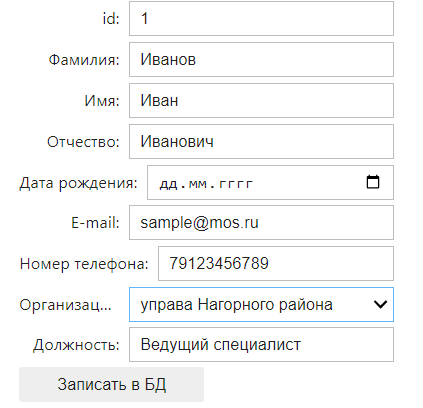
Text(value='Иванович', description='Отчество:')

DatePicker(value=None, description='Дата рождения:', step=1, style=DescriptionStyle(description\_width='initial… Text(value='sample@mos.ru', description='E-mail:')

IntText(value=79123456789, description='Номер телефона:', style=DescriptionStyle(description\_width='initial')) Dropdown(description='Организация:', options=(('Название ОИВ', 1),), value=1)

Text(value='Ведущий специалист', description='Должность:') Button(description='Записать в БД', style=ButtonStyle())

**Рисунок 1.** Экранная форма



приложение г

Создание и заполнение таблиц, создание экранных форм для таблиц dataSet, log

**Листинг 1.** Создание таблицы dataSet

1. #создание таблицы dataSet
2. query\_sql('shumskiiIN.db','''
3. CREATE TABLE dataSet (id INTEGER PRIMARY KEY NOT NULL,
4. name VARCHAR(40) NOT NULL,
5. keywordID INTEGER NOT NULL REFERENCES keywordList (id) ON DELETE NO ACTION ON UPDATE NO ACTION,
6. categoryID INTEGER NOT NULL REFERENCES categoryList (id) ON DELETE NO ACTION ON UPDATE NO ACTION,
7. rating REAL NOT NULL,
8. downloadCounter INTEGER NOT NULL,
9. periodID INTEGER NOT NULL REFERENCES periodList (id) ON DELETE NO ACTION ON UPDATE NO ACTION);
10. ''', False)

**Листинг 2.** Создание таблицы log

1. #создание таблицы log
2. query\_sql('shumskiiIN.db','''
3. CREATE TABLE log (id INTEGER PRIMARY KEY NOT NULL,
4. actionID INTEGER NOT NULL REFERENCES actionList (id) ON DELETE NO ACTION ON UPDATE NO ACTION,
5. date VARCHAR(10) NOT NULL,
6. workerID INTEGER NOT NULL REFERENCES worker (id) ON DELETE NO ACTION ON UPDATE NO ACTION,
7. GIS INTEGER NOT NULL,
8. dataSetID INTEGER NOT NULL REFERENCES dataSet (id) ON DELETE NO ACTION ON UPDATE NO ACTION,
9. urlXLSX VARCHAR(80), urlJSON VARCHAR(80), urlCSV VARCHAR(80), urlXML VARCHAR(80));
10. ''', False)

**Листинг 3.** Создание экранной формы для таблицы dataSet

1. #dataSet-form
2. import ipywidgets as widgets #Подключаем библиотеку виджетов
3. from IPython.display import display # импортируем функцию для визуализации виджетов
4. style = {'description\_width': 'initial'} #Делаем установку ширины поля discription под размер текста в нем
5. id = widgets.IntText(value=1, description='id:', disabled=False)
6. name = widgets.Text(value='Название набора:', description='Наименование:', disabled=False)
7. R=query\_sql('shumskiiIN.db','select \* from categoryList;') categoryID=widgets.Dropdown(
8. options=[tuple([R.iloc[i][1], int(R.iloc[i][0])]) for i in range(len(R['id']))], value = 1,
9. description='Категория:', disabled=False
10. )
11. R=query\_sql('shumskiiIN.db','select \* from keywordList;') keywordID=widgets.Dropdown(
12. options=[tuple([R.iloc[i][1], int(R.iloc[i][0])]) for i in range(len(R['id']))], value = 1,
13. description='Ключевое слово:', disabled=False
14. )
15. R=query\_sql('shumskiiIN.db','select \* from periodList;') periodID=widgets.Dropdown(
16. options=[tuple([R.iloc[i][1], int(R.iloc[i][0])]) for i in range(len(R['id']))], value = 1,
17. description='Период актуализации:', disabled=False
18. )
19. rating = widgets.FloatSlider(value=5.5, min=0, max=10.0, step=0.1, description='Рейтинг:', disabled=False, continuous\_update=False, orientation='horizontal', readout=True, readout\_format='.1f downloadCounter = widgets.IntText(value=10, description='Количество скачиваний', disabled=False, style=style)
20. button = widgets.Button(description='Записать в БД', disabled=False, button\_style='') print('Заполнение таблицы dataSet')
21. display(id, name, categoryID, keywordID, periodID, rating, downloadCounter, button)
22. def button\_clicked(b):
23. query = '''INSERT INTO dataSet (id, name, categoryID, keywordID, periodID, rating, downloadCounter) VALUES '''
24. query += '(' + str(id.value) + ',' + '\''+ name.value + '\'' + ','
25. query += str(categoryID.value) + ',' + str(keywordID.value) + ',' + str(periodID.value) + ',' query += str(rating.value) + ',' + str(downloadCounter.value) + ');'
26. print(query\_sql)
27. query\_sql('shumskiiIN.db', query, False)
28. button.on\_click(button\_clicked)

IntText(value=1, description='id:')

Text(value='Название набора:', description='Наименование:')

Dropdown(description='Категория:', options=(('Активный гражданин', 1), ('Выборы', 2), ('Дороги и транспорт', 3… Dropdown(description='Ключевое слово:', options=(('Зимние', 1), ('Летние', 2), ('Всесезонные', 3)), value=1)

Dropdown(description='Период актуализации:', options=(('Ежемесячно', 1), ('Ежеквартально', 2), ('Ежегодно', 3)… FloatSlider(value=5.5, continuous\_update=False, description='Рейтинг:', max=10.0, readout\_format='.1f')

IntText(value=10, description='Количество скачиваний', style=DescriptionStyle(description\_width='initial')) Button(description='Записать в БД', style=ButtonStyle())

**Листинг 4.** Создание экранной формы для таблицы log

1. #log-form
2. import ipywidgets as widgets #Подключаем библиотеку виджетов
3. from IPython.display import display # импортируем функцию для визуализации виджетов
4. style = {'description\_width': 'initial'} #Делаем установку ширины поля discription под размер текста в нем
5. id = widgets.IntText(value=1, description='id:', disabled=False)
6. R=query\_sql('shumskiiIN.db','select \* from actionList;') actionID=widgets.Dropdown(
7. options=[tuple([R.iloc[i][1], int(R.iloc[i][0])]) for i in range(len(R['id']))], value = 1,
8. description='Действие:', disabled=False
9. )
10. date = widgets.DatePicker(description='Дата:', disabled=False, style=style) R=query\_sql('shumskiiIN.db','select \* from worker;')
11. workerID=widgets.Dropdown(
12. options=[tuple([R.iloc[i][2] + ' ' + R.iloc[i][1] + ' ' + R.iloc[i][3], int(R.iloc[i][0])]) for i in range(len(R['id']))], value = 1,
13. description='Сотрудник:', disabled=False
14. )
15. GIS = widgets.IntText(value=1, description='ИД ГИС:', disabled=False)
16. =query\_sql('shumskiiIN.db','select \* from dataSet;') dataSetID=widgets.Dropdown(
17. options=[tuple([R.iloc[i][1], int(R.iloc[i][0])]) for i in range(len(R['id']))], value = 1,
18. description='Набор данных:', disabled=False
19. )
20. urlXLSX = widgets.Text(value='//localhost:8888/', description='XLSX-формат:', disabled=False) urlJSON = widgets.Text(value='//localhost:8888/', description='JSON-формат:', disabled=False) urlCSV = widgets.Text(value='//localhost:8888/', description='CSV-формат:', disabled=False)
21. urlXML = widgets.Text(value='//localhost:8888/', description='XML-формат:', disabled=False)
22. button = widgets.Button(description='Записать в БД', disabled=False, button\_style='') print('Заполнение таблицы dataSet')
23. display(id, dataSetID, workerID, actionID, date, GIS, urlXLSX, urlJSON, urlCSV, urlXML, button)
24. def button\_clicked(b):
25. query = '''INSERT INTO log (id, dataSetID, workerID, actionID, date, GIS, urlXLSX, urlJSON, urlCSV, urlXML) VALUES '''
26. query += '(' + str(id.value) + ',' + str(dataSetID.value) + ','
27. query += str(workerID.value) + ',' + str(actionID.value) + ',' + str(date.value) + ',' + str(GIS.value) + ',' query += '\''+ urlXLSX.value + '\'' + ',' + '\''+ urlJSON.value + '\'' + ','
28. query += '\''+ urlCSV.value + '\'' + ',' + '\''+ urlXML.value + '\'' + ');' print(query\_sql)
29. query\_sql('shumskiiIN.db', query, False)
30. button.on\_click(button\_clicked)

IntText(value=1, description='id:')

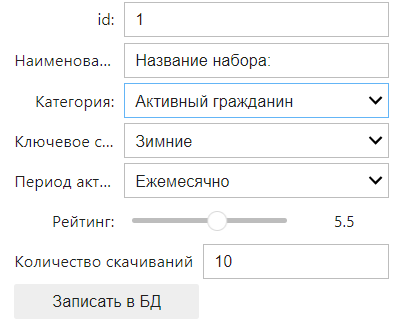
Dropdown(description='Набор данных:', options=(('Название набора:', 1),), value=1) Dropdown(description='Сотрудник:', options=(('Иванов Иван Иванович', 1),), value=1)

Dropdown(description='Действие:', options=(('Опубликован', 1), ('Архивирован', 2), ('Изъят из публичного досту… DatePicker(value=None, description='Дата:', step=1, style=DescriptionStyle(description\_width='initial'))

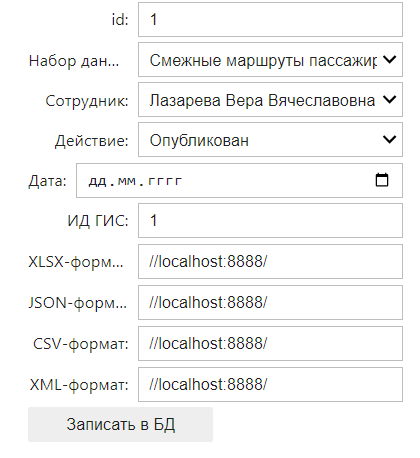
IntText(value=1, description='ИД ГИС:')

Text(value='//localhost:8888/', description='XLSX-формат:') Text(value='//localhost:8888/', description='JSON-формат:') Text(value='//localhost:8888/', description='CSV-формат:') Text(value='//localhost:8888/', description='XML-формат:') Button(description='Записать в БД', style=ButtonStyle())

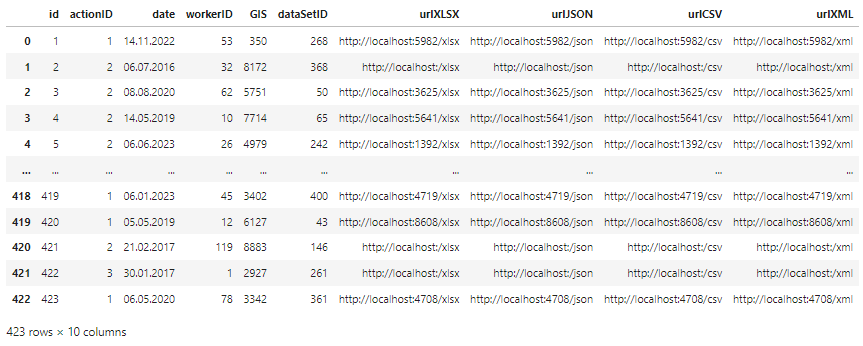
**Рисунок 1.** Экранная форма таблицы dataSet



**Рисунок 2.** Экранная форма таблицы log



**Рисунок 3.** Результат заполнения таблицы dataSet

**Рисунок 4.** Результат заполнения таблицы log

приложение Д

Написание запроса на расчет отношения неактуальных наборов данных в разрезе сотрудников

**Листинг 1.** Запрос на получение количества неактуальных наборов данных в разрезе сотрудников

1. query\_sql('shumskiiin.db', '''SELECT worker.id, worker.lastName, worker.firstName, worker.middleName, COUNT(dataSet.id) as "Total irrelevant"
2. FROM actionList
3. LEFT JOIN log
4. ON log.actionID = actionList.id
5. LEFT JOIN dataSet
6. ON dataSet.id = log.dataSetID
7. LEFT JOIN worker
8. ON log.workerID = worker.ID
9. WHERE
10. actionList.action IN ('Архивирован', 'Изъят из публичного доступа')
11. GROUP BY worker.id, worker.lastName, worker.firstName, worker.middleName
12. '''
13. )

**Рисунок 1.** Результат выполнения запроса на получение количества неактуальных наборов данных в разрезе сотрудников

**Листинг 2.** Запрос на получение количества наборов данных в разрезе сотрудников

1. query\_sql('shumskiiin.db', '''SELECT worker.id, worker.lastName, worker.firstName, worker.middleName, COUNT(dataSet.id) as "Total dataSet"
2. FROM actionList
3. LEFT JOIN log
4. ON log.actionID = actionList.id
5. LEFT JOIN dataSet
6. ON dataSet.id = log.dataSetID
7. LEFT JOIN worker
8. ON log.workerID = worker.ID
9. GROUP BY worker.id, worker.lastName, worker.firstName, worker.middleName
10. '''
11. )

**Рисунок 2.** Результат выполнения запроса на получение количества наборов данных в разрезе сотрудников

**Листинг 3.** Запрос на получение отношения неактуальных наборов данных в разрезе сотрудников

1. query\_sql('shumskiiin.db', '''SELECT a.lastName, a.firstName, a.middleName, 1.0 \* `Total irrelevant` / `Total dataSet` as 'Отношение'
2. FROM
3. (SELECT worker.id, worker.lastName, worker.firstName, worker.middleName, COUNT(dataSet.id) as "Total irrelevant"
4. FROM actionList
5. LEFT JOIN log
6. ON log.actionID = actionList.id
7. LEFT JOIN dataSet
8. ON dataSet.id = log.dataSetID
9. LEFT JOIN worker
10. ON log.workerID = worker.ID
11. WHERE
12. actionList.action IN ('Архивирован', 'Изъят из публичного доступа')
13. GROUP BY worker.id, worker.lastName, worker.firstName, worker.middleName) a,
14. (SELECT worker.id, worker.lastName, worker.firstName, worker.middleName, COUNT(dataSet.id) as "Total dataSet"
15. FROM actionList
16. LEFT JOIN log
17. ON log.actionID = actionList.id
18. LEFT JOIN dataSet
19. ON dataSet.id = log.dataSetID
20. LEFT JOIN worker
21. ON log.workerID = worker.ID
22. GROUP BY worker.id, worker.lastName, worker.firstName, worker.middleName) b
23. WHERE a.id = b.id;
24. '''
25. )

**Рисунок 3.** Результат выполнения запроса на получение отношения неактуальных наборов данных в разрезе сотрудников

1. Распоряжение Правительства РФ от 30.01.2014 № 93-р «Об утверждении концепции открытости федеральных органов исполнительной власти» // Правительство РФ URL: http://government.ru/docs/10122/ (дата обращения: 06.01.2024). [↑](#footnote-ref-1)
2. International Council for Science World Data System // re3data.org URL: https://www.re3data.org/search (дата обращения: 06.01.2024). [↑](#footnote-ref-2)
3. FAIR Principles // go-fair.org URL: https://www.go-fair.org/fair-principles (дата обращения: 06.01.2024). [↑](#footnote-ref-3)
4. Базы данных // Oracle.com URL: https://www.oracle.com/cis/database/what-is-a-relational-database/ (дата обращения: 06.01.2024). [↑](#footnote-ref-4)
5. About SQLite // SQLite.org URL: https://www.sqlite.org/about.html (дата обращения: 07.01.2024). [↑](#footnote-ref-5)
6. Well-Known Users of SQLite // SQLite.org URL: https://www.sqlite.org/famous.html (дата обращения: 07.01.2024). [↑](#footnote-ref-6)
7. NumPy documentation // NumPy URL: https://numpy.org/doc/stable/ (дата обращения: 07.01.2024). [↑](#footnote-ref-7)
8. Pandas documentation // pandas URL: https://pandas.pydata.org/docs/ (дата обращения: 07.01.2024). [↑](#footnote-ref-8)
9. Widget List // Ipywidgets URL: https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20List.html (дата обращения: 07.01.2024). [↑](#footnote-ref-9)