

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»
Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта

КУРСОВАЯ РАБОТА
УПРАВЛЕНИЕ МЕТАДААННЫМИ РЕЛЯЦИОННОЙ СИСТЕМЫ
УПРАВЛЕНИЯ БАЗЫ ДАННЫХ POSTGRESQL

Выполнил:
студент гр. 5140201/50301

А.С. Тимофеев

Преподаватель:
доцент ВШТИИ ИКНК

С.Г. Попов

Санкт-Петербург
2025

СОДЕРЖАНИЕ

| | |
|---|----|
| ПОСТАНОВКА ЗАДАЧИ | 4 |
| 1 ОПИСАНИЕ ЗАДАННОГО ОКРУЖЕНИЯ | 5 |
| 2 ХРАНИЛИЩЕ МЕТАДАННЫХ | 9 |
| 2.1 Схема базы данных | 9 |
| 2.2 Запросы к information schema | 11 |
| 2.3 Использование построенных запросов..... | 14 |
| 3 ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС..... | 16 |
| 3.1 Генерация SELECT-запросов | 16 |
| 3.2 Особенности пользовательского интерфейса..... | 19 |
| ЗАКЛЮЧЕНИЕ | 23 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ | 24 |
| ПРИЛОЖЕНИЕ А..... | 25 |

ПОСТАНОВКА ЗАДАЧИ

В рамках работы необходимо реализовать приложение для работы с метаданными реляционной системы управления базой данных PostgreSQL и реализовать генератор подмножества SELECT запросов для заданных баз данных, а также хранение истории сгенерированных запросов. Для решения данной задачи выделены следующие подзадачи:

1. Описание заданного окружения.
2. Создание хранилища метаданных:
 - проектирование схемы базы данных метаданных;
 - составление запросов к information schema;
 - реализация сохранения метаданных по DSN-строке подключения к СУБД;
3. Реализация графического интерфейса.
4. Разработка генератора подмножества SELECT.
5. Организация хранения истории запросов и возможности их повторного выполнения.

1 ОПИСАНИЕ ЗАДАННОГО ОКРУЖЕНИЯ

Общая схема сетевого взаимодействия сущностей

В текущей работе рассматриваются три схемы и три СУБД расположенных удаленном сервере в отдельных Docker-контейнерах:

- База данных метаданных;
- База данных по баскетболу;
- База данных по игре Dota 2;

На **Рисунке 1** представлена схема взаимодействия приложения управления метаданными и заданными базами данных.

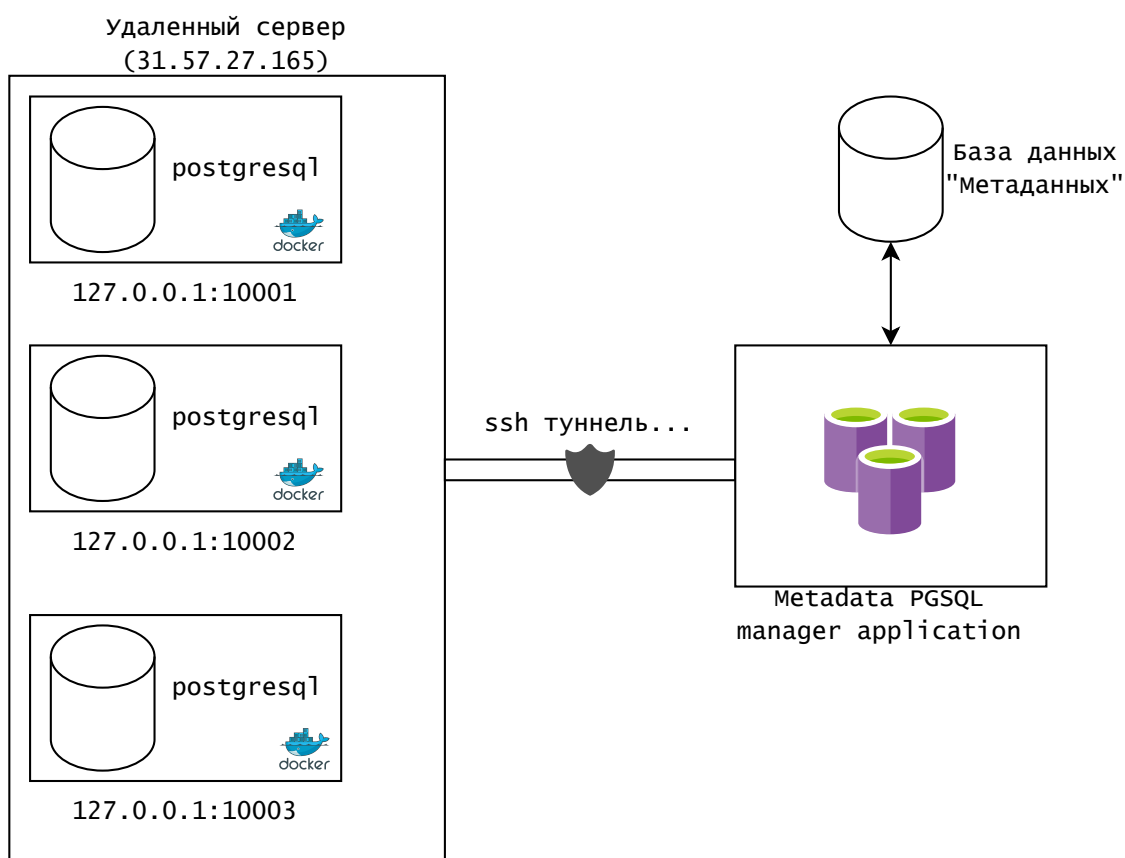


Рис. 1 — Схема сетевого взаимодействия приложения и баз данных

В рамках данной работы соединение с удаленным сервером с развернутыми СУБД выполнено через SSH-туннель.

Развертывание и установка соединения с БД

Для развертывания Docker-контейнеров были использованы Docker-compose скрипты представленные в [Листинге 1.1](#)

```
1 services:
2 metadata-db:
3     image: postgres:16
4     container_name: metadata-db
5     restart: unless-stopped
6     env_file: .env
7     ports:
8         - "127.0.0.1:${VPS_PORT}:5432"
9     volumes:
10         - metadata_db_data:/var/lib/postgresql/data
11 volumes:
12     metadata_db_data:
```

Листинг 1.1 — docker-compose.yml файл для создания контейнера с СУБД

Параметр VPS-PORT задается в файле переменных окружения .env, пример файла представлен в [Листинге 1.2](#)

```
1 POSTGRES_DB=app
2 POSTGRES_USER=appuser
3 POSTGRES_PASSWORD=ChangeMe_12345
4 VPS_PORT=10001
```

Листинг 1.2 — .env файл для создания контейнера с СУБД

В .env файле дополнительно настраиваются переменные окружения для подключения к базе данных: POSTGRES_DB, POSTGRES_USER, POSTGRES_PASSWORD.

Для подключения к базе данных используется DSN (Data source name) по следующему шаблону:

postgresql://[user[:password]@][host][:port]/[database_name].

Для установки соединения необходимо настройку SSH-туннеля для портов, соответствующих контейнеров на удаленном сервере.

Описание взаимодействия внутренних компонент

В рамках курсовой работы были выделены следующие основные компоненты:

1. Metadata Manager - приложение (web-сервер).
2. Metadata Database - СУБД Postgresql с базой данных для хранения метаданных.
3. Функциональные компоненты взаимодействия с СУБД:
 - Metadata Extractor Service - извлечение метаданных из базы данных;
 - Metadata Writer Service - запись в Metadata Database;
 - Metadata Reader Service - чтение из Metadata Database;
 - Meta-select executor service - выполнение запросов к базам данных;
4. Web-UI - пользовательский графический веб-интерфейс.

На **Рисунке 2** представлена схема взаимодействия основных компонент приложения.

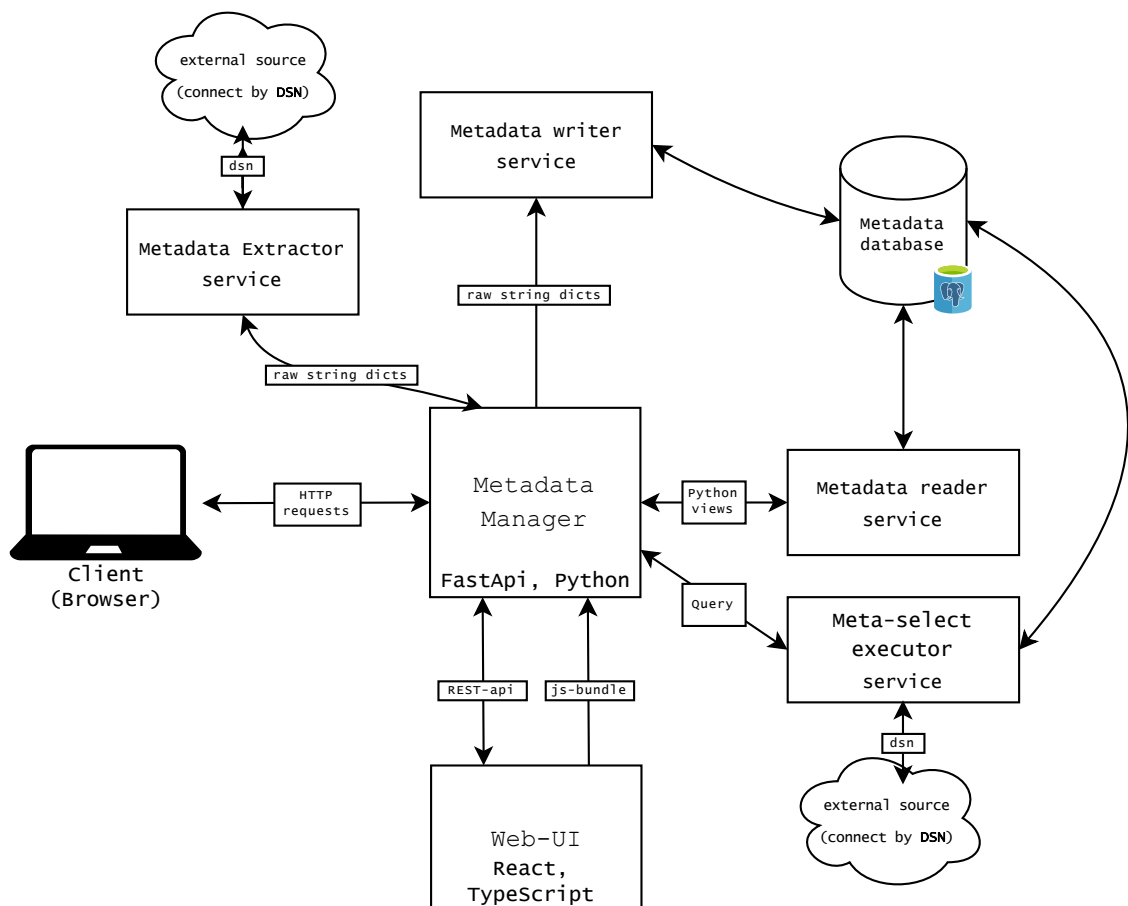


Рис. 2 — Схема взаимодействия основных компонент приложения

Для реализации приложения использованы следующие языки программирования и технологии:

- Python 3.12;
- TypeScript;
- React
- FastApi, uvicorn web server;
- PostgreSQL;

2 ХРАНИЛИЩЕ МЕТАДААННЫХ

Метаданные - информация о другой информации, или данные, относящиеся к дополнительной информации о содержимом или объекте. Метаданные раскрывают сведения о признаках и свойствах, характеризующих какие-либо сущности, позволяющие автоматически искать и управлять ими в больших информационных потоках. [1]

2.1 Схема базы данных

Для реляционной базы данных, рассматриваемой в курсовой работе, выбран следующий набор метаданных, характеризующих базу данных:

1. Имя базы данных.
2. Секреты для подключения к базе данных:
 - сетевой адрес;
 - порт;
 - имя пользователя;
 - пароль;
3. Набор таблиц.
4. Набор колонок.
5. Набор первичных ключей.
6. Набор внешних ключей.

Данный набор сущностей позволяет реализовать генератор подмножества SELECT-запросов.

На **Рисунке 4** представлена реляционная схема базы данных метаданных, хранящая вышеописанные сущности.

SQL-скрипт создания представленной схемы приведен в **Приложении А**.

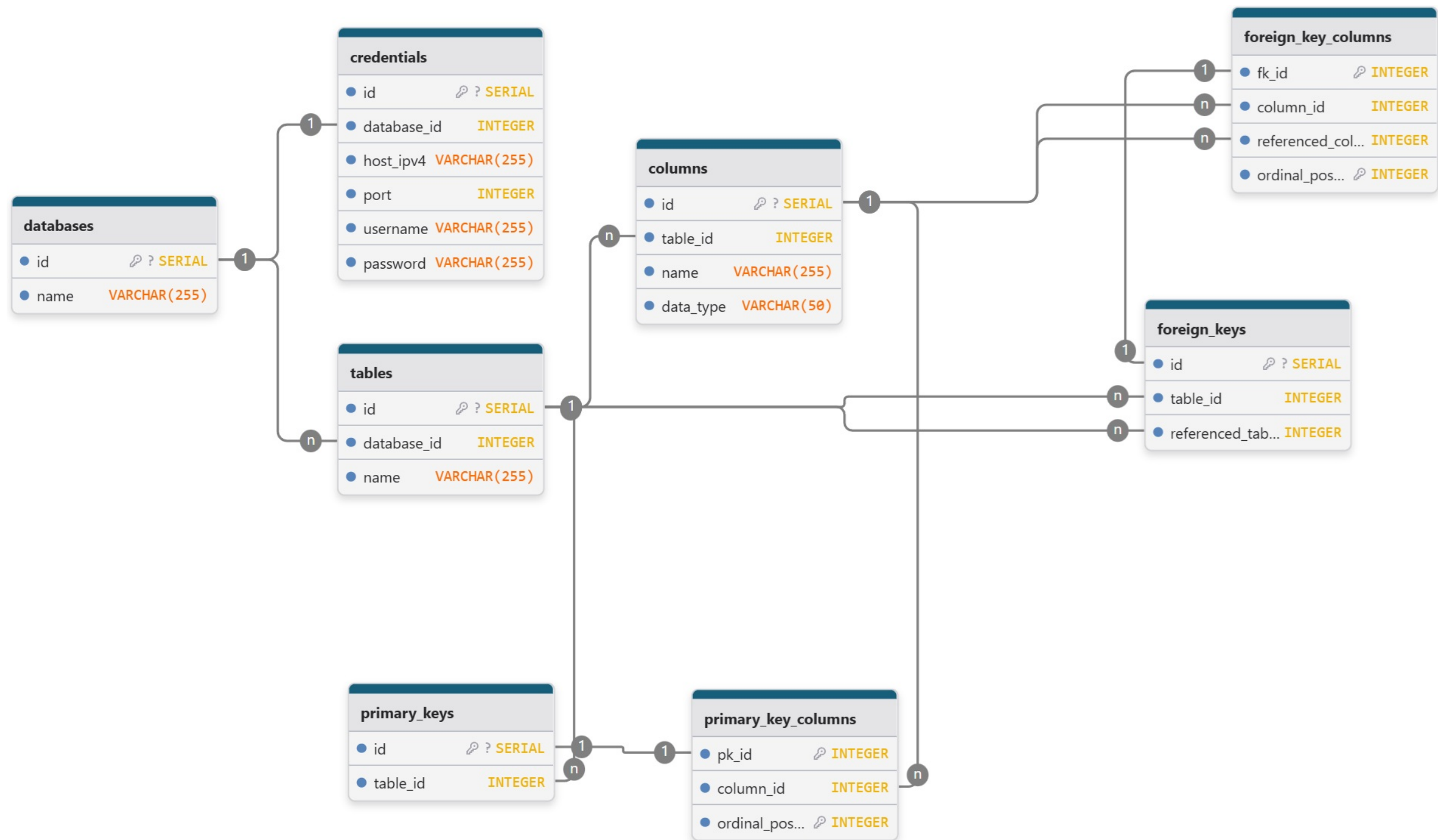


Рисунок 4 — Схема базы данных метаданных

2.2 Запросы к information schema

Каждая созданная в СУБД PostgreSQL база данных содержит отдельную информационную схему, называющуюся `information schema`. Данная схема содержит таблицы, колонки и связи созданные пользователем. [2]

Получение списка таблиц

Представление `information_schema.tables` содержит в себе таблицы в текущей базе данных. При обращении к ней есть только те таблицы, которые доступны текущему пользователю. [3]

В Листинге 2.1 приведен пример выполнения запроса к `information schema` для получения имен таблиц.

```
1 SELECT
2 table_name
3 FROM information_schema.tables
4 WHERE table_schema = 'public' AND table_type='BASE TABLE'
5 ORDER BY table_name;
6
7      table_name
8  -----
9 columns
10 credentials
11 databases
12 foreign_key_columns
13 foreign_keys
14 primary_key_columns
15 primary_keys
16 saved_queries
17 tables
```

Листинг 2.1 — Пример выполнения запроса для получения списка таблиц

Получение списка колонок

Представление `information_schema.columns` содержит в себе колонки в текущей базе данных. При обращении к ней есть только те таблицы, которые доступны текущему пользователю.

В **Листинге 2.2** приведен пример выполнения запроса к `information schema` для получения имен колонок, типов данных и порядка колонки.

```
1 SELECT
2 ordinal_position,
3 column_name,
4 data_type
5 FROM information_schema.columns
6 WHERE table_schema = 'public'
7 AND table_name = 'credentials'
8 ORDER BY ordinal_position;
```

| ordinal_position | column_name | data_type |
|------------------|-------------|-------------------|
| 1 | id | integer |
| 2 | database_id | integer |
| 3 | host_ipv4 | character varying |
| 4 | port | integer |
| 5 | username | character varying |
| 6 | password | character varying |

Листинг 2.2 — Пример выполнения запроса для получения списка колонок для таблицы `credentials`

Получение списка первичных ключей

Представление `information_schema.table_constraints` содержит в себе необходимую информацию о первичных и внешних ключах таблицы. В представлении `information_schema.key_column_usage` содержится информация о используемых ключах.

В **Листинге 2.3** приведен пример выполнения запроса к `information schema` для получения списка первичных ключей для таблицы `databases`.

```
1 SELECT
2 kcu.column_name AS column_name,
3 kcu.ordinal_position AS position_in_pk
4 FROM information_schema.table_constraints AS tc
5 JOIN information_schema.key_column_usage AS kcu`
6 ON tc.constraint_name = kcu.constraint_name
7 AND tc.table_schema = kcu.table_schema
8 WHERE tc.constraint_type = 'PRIMARY_KEY'
```

```

9 AND tc.table_schema = 'public'
10 AND tc.table_name = 'databases'
11 ORDER BY kcu.ordinal_position;
12
13 column_name | position_in_pk
14 -----+-----
15 id          | 1

```

Листинг 2.3 — Пример выполнения запроса для получения списка первичных ключей для таблицы databases

Получение списка внешних ключей

В [Листинге 2.4](#) приведен пример выполнения запроса к information schema для получения списка внешних ключей для таблицы databases.

```

1 SELECT
2 kcu.table_name          AS src_table,
3 kcu.column_name        AS src_column,
4 ccu.table_name          AS tgt_table,
5 ccu.column_name        AS tgt_column,
6 kcu.ordinal_position    AS position
7 FROM information_schema.table_constraints AS tc
8 JOIN information_schema.key_column_usage AS kcu
9 ON tc.constraint_name = kcu.constraint_name
10 AND tc.table_schema = kcu.table_schema
11 JOIN information_schema.constraint_column_usage AS ccu
12 ON ccu.constraint_name = tc.constraint_name
13 AND ccu.constraint_schema = tc.table_schema
14 WHERE tc.constraint_type = 'FOREIGN_KEY'
15 AND kcu.table_schema = 'public'
16 AND kcu.table_name = 'credentials'
17 ORDER BY position;
18 src_table | src_column | tgt_table | tgt_column | position
19 -----+-----+-----+-----+-----
20 credentials | database_id | databases | id          | 1

```

Листинг 2.4 — Пример выполнения запроса для получения списка внешних ключей для таблицы credentials

2.3 Использование построенных запросов

Чтение метаданных

Полученные запросы используются в `core/extractor-service` для получения метаданных из внешней базы данных. Для подключения к внешней базе данных используется DSN-строка, например: `postgresql://appuser:password123@31.124.33.3:5432/databasename`. Полученные данные представляются как есть в виде словарей.

Реализованный модуль для извлечения метаданных содержит следующие основные методы:

- `list_tables(database)` - получение списка таблиц;
- `list_columns(database, table)` - получение списка колонок;
- `list_primary_keys(database, table, column)` - получение списка первичных ключей;
- `list_foreign_keys(database, table, column)` - получение списка внешних ключей;

В [Листинге 2.5](#) приведен пример ”псевдовызова” метода и результат вызова.

```
1 > extractor('postgresql://appuser:password123@31.124.33.3:5432/
   databasename')
2   .list_columns('users')
3 >
4 [
5   {'name': 'id', 'data_type': 'integer'},
6   {'name': 'name', 'data_type': 'text'},
7   {'name': 'age', 'data_type': 'integer'},
8   {'name': 'is_active', 'data_type': 'boolean'}
9 ]
```

Листинг 2.5 — Пример выполнения запроса из `core/extractor-service`

Запись метаданных

Модуль `core/writer-service` используются для записи метаданных из внешней базы данных в локальную базу данных метаданных. Входные данные ожидаются как есть в виде словарей.

Реализованный модуль для извлечения метаданных содержит следующие основные методы:

- `ensure_database(database_name)` - запись информации о базе данных;
- `ensure_credentials(database_name, dsn)` - запись секретов о базе данных;
- `ensure_tables(database, tables)` - запись таблиц;
- `ensure_columns(database, table, columns)` - запись колонок;
- `ensure_primary_keys(database, table, pkeys)` - запись первичных ключей;
- `ensure_foreign_keys(database, table, fkeys)` - запись внешних ключей;

Чтение метаданных

Модуль `core/reader-service` используются для чтения метаданных из локальной базы данных метаданных. Выходные данные ожидаются представление ”один ко дному” данных на уровне программы.

Реализованный модуль для чтения метаданных содержит следующие основные методы:

- `list_database(database_name)` - чтение информации о базе данных;
- `list_credentials(database_name)` - чтение секретов о базе данных;
- `list_tables(database, tables)` - чтение таблиц;
- `list_columns(database, table)` - чтение колонок;
- `list_primary_keys(database, table)` - чтение первичных ключей;
- `list_foreign_keys(database, table)` - чтение внешних ключей;

Таким образом, реализованные модули чётко разделяют зоны ответственности, а их взаимодействие через определённые контракты обеспечивает соблюдение уровней абстракции внутри приложения `Manager Metadata`.

3 ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

В курсовой работе реализован графический пользовательский веб-интерфейс, доступный например через веб-браузер.

Для программной реализации был использован язык программирования: TypeScript и библиотека React. Для сборки js-bundle использовалась утилита eslint.

3.1 Генерация SELECT-запросов

Генерация SELECT-запросов выполняется на уровне клиента (пользователя). Для данной работы выбрано следующее подмножество SELECT-запросов:

1. Выбор одной или нескольких колонок.
2. Агрегатные функции для колонок.
3. Единственное условие WHERE (опционально).

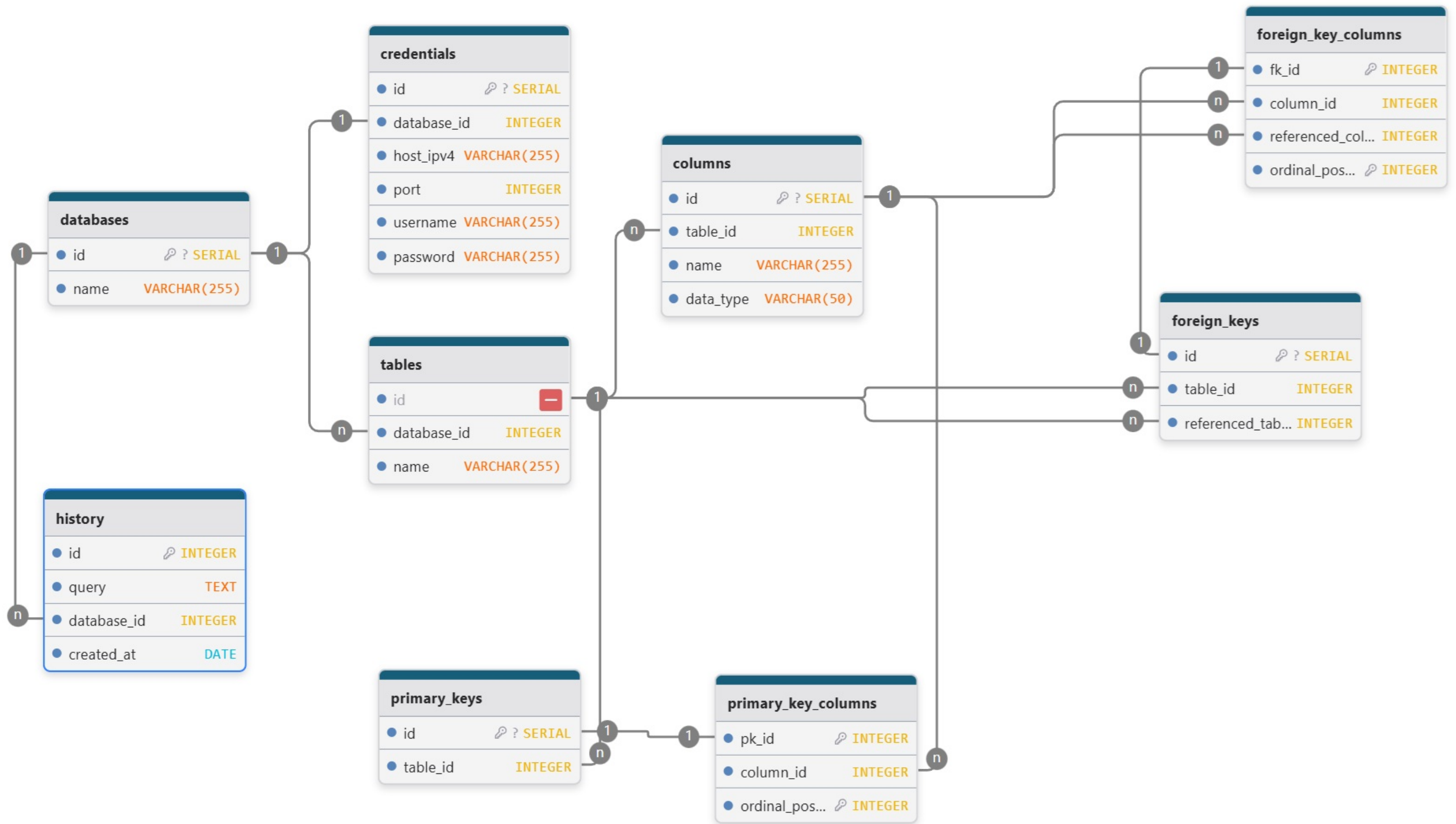
Также для генератора запроса выдвинуты дополнительные требования:

- наполнение данных в форме при генерация если выбрать таблицу или колонку синхронизируется.
- сгенерированные запросы сохраняются в приложении.

Сохранение SELECT-запросов

Для реализации сохранения сгенерированных SELECT-запросов была введена дополнительная таблица в базу данных метаданных history. При выполнении запроса всегда фиксируем историю всех запросов в базе данных для воспроизведения их в дальнейшем.

На **Рисунке 6** представлена обновленная реляционная схема базы данных метаданных, хранящая историю запросов.



С точки зрения абстракции не совсем правильно хранить в базе данных метаданных запросы, однако для более удобной разработки в курсовой работе данные сущности располагаются рядом.

3.2 Особенности пользовательского интерфейса

В пользовательском интерфейсе определены следующие основные части:

1. Список подключенных баз данных.
2. Форма добавления новой базы данных по DSN.
3. Генератор SELECT-запроса.
4. История сгенерированных SELECT-запросов.
5. Модальная форма результата SELECT-запроса.

Список подключенных баз данных

На **Рисунке 7** представлен внешний вид компонентов пользовательского интерфейса:

- список подключенных баз данных;
- форма добавления новой базы данных;

Current databases

| Name | Address |
|------------|-----------------|
| app | 127.0.0.1:10002 |
| metadata | localhost:10001 |
| basketball | 127.0.0.1:10003 |

Add new

Рис. 7 — Список подключений и форма создания нового подключения

Генератор SELECT-запроса

На **Рисунке 8** представлен внешний вид генератора SELECT-запроса.

The screenshot shows a web interface titled "Select from meta-storage". It contains several input fields and a button:

- SELECT**: A dropdown menu with "COUNT" selected.
- name [player] (app)**: A dropdown menu with "name [player] (app)" selected.
- + add column**: A button with a dropdown arrow.
- FROM**: A dropdown menu with "player (app)" selected.
- WHERE**: A dropdown menu with "rating [player] (app)" selected.
- ≥**: A dropdown menu with "≥" selected.
- 10000**: A text input field containing the value "10000".
- Execute**: A blue button.

Below the input fields, there is a text box showing the generated SQL query:

```
SELECT COUNT(name) FROM
player WHERE rating >=
10000;
```

Рис. 8 — Генератор SELECT-запроса

В генераторе запроса пользователю доступен выбор:

- одной или нескольких колонок;
- опциональные агрегатные функции (COUNT, SUM, MIN, MAX, AVG);
- опциональные условия WHERE;

Наполнение select-box'ов зависит от выбранных колонок или таблицы и синхронизируется в зависимости выбора пользователя.

История сгенерированных SELECT-запросов

На **Рисунке 9** представлен внешний вид истории сгенерированных SELECT-запросов.

| database_name | sql_query | created_at | execute? |
|---------------|--|----------------------|--------------------------|
| app | SELECT COUNT(name) FROM player WHERE rating >= 10000; | 02.11.2025, 14:16:14 | <button>Execute</button> |
| app | SELECT COUNT(name) FROM player WHERE rating >= 10000; | 02.11.2025, 14:16:08 | <button>Execute</button> |
| app | SELECT * FROM hero; | 02.11.2025, 14:12:58 | <button>Execute</button> |
| metadata | SELECT name FROM columns; | 02.11.2025, 14:12:55 | <button>Execute</button> |
| basketball | SELECT COUNT(game_id), game_data FROM game WHERE game_time = '18:00:00'; | 30.10.2025, 11:10:05 | <button>Execute</button> |
| basketball | SELECT COUNT(game_id) FROM game WHERE game_time = '18:00:00'; | 30.10.2025, 11:09:11 | <button>Execute</button> |
| basketball | SELECT COUNT(game_id) FROM game WHERE game_time = '12:00:00'; | 30.10.2025, 11:09:06 | <button>Execute</button> |
| basketball | SELECT COUNT(game_id) FROM game WHERE game_time > '12:00:00'; | 30.10.2025, 11:03:09 | <button>Execute</button> |
| basketball | SELECT * FROM game; | 30.10.2025, 11:01:53 | <button>Execute</button> |
| app | SELECT * FROM hero; | 30.10.2025, 10:50:00 | <button>Execute</button> |
| basketball | SELECT * FROM foultype; | 30.10.2025, 09:26:03 | <button>Execute</button> |

Рис. 9 — История SELECT-запросов

Модальное окно результата SELECT-запроса

На **Рисунке 10** представлен внешний вид модального окна выполненного SELECT-запроса.

| foultype_id | foultype_name | foultype_weight |
|-------------|------------------------|-----------------|
| 1 | Обоюдный фол | 1 |
| 2 | Технический фол | 1 |
| 3 | Умышленный фол | 1 |
| 4 | Дисквалифицирующий фол | 5 |
| 5 | Персональный фол | 1 |

Рис. 10 — Результат SELECT-запроса

Пользовательский интерфейс позволяет выполнять генерацию SELECT-запросов и их выполнение для добавленных баз данных. За счет разделения сущностей пользовательский интерфейс обращается через API к Metadata Manager веб-серверу и не выполняет напрямую манипуляции с внешними базами данных.

ЗАКЛЮЧЕНИЕ

В результате курсовой работы разработано приложение для управления работы с метаданными реляционной системы управления базой данных PostgreSQL и реализован генератор подмножества `SELECT` запросов для заданных баз данных, а также хранение истории сгенерированных запросов.

Для решения данной задачи были выполнены следующие подзадачи:

1. Описание заданного окружения.
2. Создание хранилища метаданных:
 - проектирование схемы базы данных метаданных;
 - составление запросов к `information schema`;
 - реализация сохранения метаданных по DSN-строке подключения к СУБД;
3. Реализация графического интерфейса.
4. Разработка генератора подмножества `SELECT`.
5. Организация хранения истории запросов и возможности их повторного выполнения.

Следует отметить, что одним из недостатков рассматриваемой реализации является хранение истории запросов в базе данных метаданных, что частично нарушает разделения абстракций. Данный подход был выбран для упрощения реализации на этапе разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Шеймович А.В.* Электронные базы метаданных и построение диалектологического атласа // Родной язык. — 2022. — № 2. — URL: <https://cyberleninka.ru/article/n/elektronnye-bazy-metadannyh-i-postroenie-dialektologicheskogo-atlasa>; (дата обращения: 01.11.2025).
2. *PostgreSQL Global Development Group.* PostgreSQL Documentation: Glossary. — 2025. — URL: <https://www.postgresql.org/docs/18/glossary.html>; (дата обращения: 01.11.2025).
3. *PostgreSQL Global Development Group.* PostgreSQL Documentation: 35.54. tables. — 2025. — URL: <https://www.postgresql.org/docs/18/infoschema-tables.html>; (дата обращения: 01.11.2025).

ПРИЛОЖЕНИЕ А

```
1  -- =====
2  -- DROP EXISTING TABLES
3  -- =====
4  DROP TABLE IF EXISTS
5  credentials,
6  foreign_key_columns,
7  foreign_keys,
8  primary_key_columns,
9  primary_keys,
10 columns,
11 tables,
12 databases
13 CASCADE;
14
15 -- =====
16 -- DATABASES
17 -- =====
18 CREATE TABLE databases (
19 id SERIAL PRIMARY KEY,
20 name VARCHAR(255) NOT NULL
21 );
22
23 -- =====
24 -- TABLES
25 -- =====
26 CREATE TABLE tables (
27 id SERIAL PRIMARY KEY,
28 database_id INT NOT NULL REFERENCES databases(id) ON DELETE
    CASCADE,
29 name VARCHAR(255) NOT NULL
30 );
31
32 -- =====
33 -- COLUMNS
34 -- =====
35 CREATE TABLE columns (
36 id SERIAL PRIMARY KEY,
37 table_id INT NOT NULL REFERENCES tables(id) ON DELETE CASCADE,
```



```

38 name VARCHAR(255) NOT NULL,
39 data_type VARCHAR(50) NOT NULL
40 );
41
42 -- =====
43 -- PRIMARY KEYS
44 -- =====
45 CREATE TABLE primary_keys (
46 id SERIAL PRIMARY KEY,
47 table_id INT NOT NULL REFERENCES tables(id) ON DELETE CASCADE
48 );
49
50 CREATE TABLE primary_key_columns (
51 pk_id INT NOT NULL REFERENCES primary_keys(id) ON DELETE CASCADE,
52 column_id INT NOT NULL REFERENCES columns(id) ON DELETE CASCADE,
53 ordinal_position INT NOT NULL,
54 PRIMARY KEY (pk_id, ordinal_position),
55 UNIQUE (pk_id, column_id)
56 );
57
58 -- =====
59 -- FOREIGN KEYS
60 -- =====
61 CREATE TABLE foreign_keys (
62 id SERIAL PRIMARY KEY,
63 table_id INT NOT NULL REFERENCES tables(id) ON DELETE CASCADE,
64 referenced_table_id INT NOT NULL REFERENCES tables(id) ON DELETE
    CASCADE
65 );
66
67 CREATE TABLE foreign_key_columns (
68 fk_id INT NOT NULL REFERENCES foreign_keys(id) ON DELETE CASCADE,
69 column_id INT NOT NULL REFERENCES columns(id) ON DELETE CASCADE,
70 referenced_column_id INT NOT NULL REFERENCES columns(id) ON
    DELETE CASCADE,
71 ordinal_position INT NOT NULL,
72 PRIMARY KEY (fk_id, ordinal_position),
73 UNIQUE (fk_id, column_id, referenced_column_id)
74 );
75
76 -- =====
77 -- CREDENTIALS

```

```

78 -- =====
79 CREATE TABLE credentials (
80 id SERIAL PRIMARY KEY,
81 database_id INT NOT NULL REFERENCES databases(id) ON DELETE
    CASCADE,
82 host_ipv4 VARCHAR(255) NOT NULL,
83 port INT NOT NULL CHECK (port > 0 AND port <= 65535),
84 username VARCHAR(255) NOT NULL,
85 password VARCHAR(255) NOT NULL
86 );
87
88 -- =====
89 -- SAVED QUERIES (history)
90 -- =====
91 CREATE TABLE history (
92 id SERIAL PRIMARY KEY,
93 database_id INTEGER NOT NULL REFERENCES databases(id) ON DELETE
    CASCADE,
94 sql_query TEXT NOT NULL,
95 created_at TIMESTAMP DEFAULT NOW()
96 );

```

Листинг 3.1 — SQL-скрипт создания схемы базы данных