Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 22.Б09-мм

Разработка Moodle плагина проверки SQL запросов к СУБД PostgreSQL

Николаев Даниил Владимирович

Отчёт по учебной практике
в форме «Решение»

Научный руководитель:
доцент кафедры информационно-аналитических систем, к. ф.-м. н., Е. Г. Михайлова

Санкт-Петербург
2025

Оглавление

Введение

Moodle — это платформа для организации учебных курсов, которая используется в университетах и школах для создания заданий, тестирования и управления обучением. Она позволяет преподавателям размещать материалы, проводить тесты и оценивать знания студентов. Одно из главных преимуществ Moodle — возможность создавать собственные плагины, которые расширяют функциональность системы, например, добавляют новые типы вопросов или инструменты для проверки заданий.

Для обучения SQL в Moodle нет встроенного инструмента, который бы позволял проверять запросы к базе данных PostgreSQL. Единственное доступное решение — плагин CodeRunner, который по умолчанию работает только с SQLite. CodeRunner поддерживает выполнение SQL-запросов, но не имеет глубокой интеграции с PostgreSQL, что ограничивает возможности для создания сложных заданий и обеспечения безопасности.

В рамках учебной практики пятого семестра было разработано расширение для CodeRunner, добавляющее поддержку PostgreSQL. Для работы с PostgreSQL был обновлен Docker-образ Jobe-сервера, включая установку необходимых пакетов и настройку базы данных. Однако это решение оказалось неполноценным из-за ограниченной гибкости.

В данной курсовой работе предлагается новый плагин для Moodle — PostgreSQL Runner. Он позволяет студентам писать SQL-запросы к PostgreSQL, выполняет их в безопасной среде с временными ролями и схемами, а затем сравнивает результаты с правильными. Плагин поддерживает банк вопросов с параметризацией, настройку базы данных, валидацию запросов и гибкие критерии оценки. Это решение обеспечивает безопасность, удобство для преподавателей и полноценный инструмент для обучения SQL.

1. Постановка задачи

Целью работы является разработка Moodle-плагина для проверки SQL-запросов к СУБД PostgreSQL.

Для достижения этой цели были поставлены следующие **задачи**:

- Изучить архитектуру Moodle и существующие плагины для проверки кода, включая CodeRunner, чтобы определить их возможности и ограничения.
- Разработать архитектуру плагина PostgreSQL Runner, обеспечивающего интеграцию с PostgreSQL и безопасное выполнение запросов.
- Реализовать функционал плагина, включая поддержку банка вопросов с параметризацией, настройку окружения базы данных и различные варианты оценки результатов.
- Обеспечить безопасность выполнения запросов через валидацию, изолированные роли и схемы, а также ограничение доступа к системным таблицам.
- Протестировать плагин на различных сценариях, включая корректность проверки запросов и устойчивость к ошибкам.
- Задокументировать процесс установки, настройки и использования плагина для преподавателей и администраторов Moodle.

2. Обзор

2.1. CodeRunner в Moodle

CodeRunner — плагин для Moodle, расширяющий возможности системы за счёт автоматической проверки программного кода, включая SQL-запросы. Он интегрируется как тип вопроса в тестах Moodle, позволяя преподавателям задавать язык программирования, тестовые примеры, ожидаемые результаты и правила оценки. Студенты вводят код в текстовое поле, после чего он отправляется на сервер CodeRunner, который использует Jobe-сервер для выполнения в изолированной среде [1]. Результаты возвращаются в Moodle, и студент получает оценку.

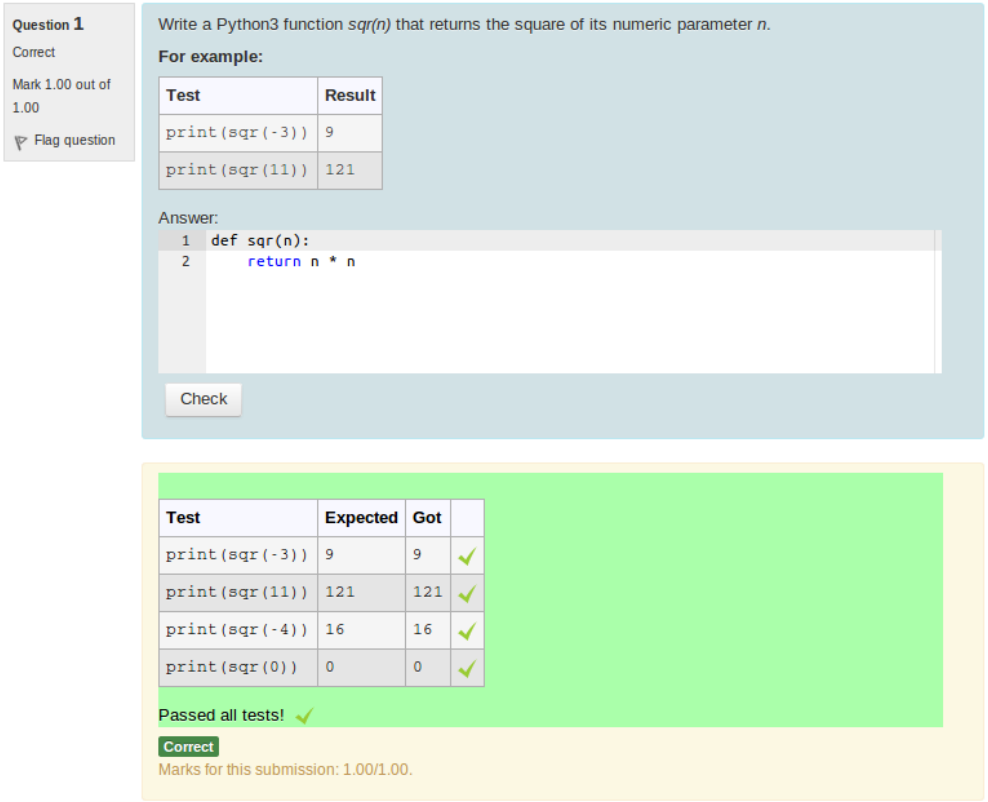

Рис. 1: Пример интерфейса вопроса CodeRunner в Moodle

По умолчанию CodeRunner поддерживает SQLite для SQL-запросов, что ограничивает его возможности для работы с PostgreSQL. В рамках учебной практики пятого семестра было разработано расширение для CodeRunner, добавляющее поддержку PostgreSQL. Основные изменения включали:

- использование утилиты `psql` для выполнения запросов;
- реализацию транзакций и точек сохранения (`SAVEPOINT`) для изоляции изменений;
- добавление функции `sort_lines_and_columns` для сортировки строк и столбцов, что позволило гибко сравнивать результаты;
- модификацию Docker-образа Jobe-сервера, включающего установку PostgreSQL и библиотеки `psycopg2`.

Расширение позволило выполнять SQL-запросы в PostgreSQL, но выявило ряд ограничений, которые стали основой для разработки нового плагина.

2.2. Недостатки CodeRunner

- Недостаточная гибкость из-за большой кодовой базы: CodeRunner имеет обширную кодовую базу, поддерживающую множество языков программирования. Добавление новых функций, таких как параметризация заданий или банк вопросов, запрошенных преподавателем, оказалось трудоёмким. Поиск подходящего места для реализации изменений занимал значительное время, что снижало эффективность разработки.
- Нагруженный интерфейс: интерфейс CodeRunner содержит множество полей, предназначенных для настройки заданий на языках программирования, таких как Python или C++, но не относящихся к SQL. Это создаёт неудобства для преподавателей, которым приходится игнорировать лишние настройки при создании SQL-заданий.
- Зависимость от Jobe-сервера: CodeRunner использует Jobe-сервер для выполнения кода. Публичный Jobe-сервер, размещённый в Университете Кентербери, может вызывать задержки и проблемы с конфиденциальностью данных [1]. Локальный Jobe-сервер требует

настройки, но его масштабируемость при одновременной работе множества пользователей с PostgreSQL остаётся под вопросом, так как нагрузочное тестирование не проводилось.

Эти недостатки подтолкнули к созданию специализированного плагина, ориентированного на SQL-запросы к PostgreSQL.

2.3. Аналогичные сервисы за пределами Moodle

Существует множество онлайн-платформ для практики SQL, таких как SQL Fiddle, SQLZoo, LeetCode и StrataScratch, которые позволяют пользователям выполнять запросы в браузере [6, 7]. Однако их подходы к обеспечению безопасности и изоляции запросов редко документированы публично. На основе анализа технической документации и обсуждений на форумах, таких как Moodle.org и Habr, можно выделить два основных метода изоляции пользовательских запросов в образовательных системах [2]:

- Использование схем баз данных: в PostgreSQL схемы позволяют создавать изолированные пространства для данных, где каждый пользователь или сессия работает в своей схеме [4]. Это обеспечивает безопасность, предотвращая доступ к данным других пользователей, и является менее ресурсоёмким подходом. Например, обсуждения на Moodle.org упоминают использование схем для разделения данных в кастомных решениях [3]. Этот метод эффективен для образовательных платформ, где требуется поддержка множества пользователей без значительных затрат ресурсов.

- Контейнеризация с Docker: некоторые платформы, такие как DataCamp, используют Docker-контейнеры для создания отдельных экземпляров баз данных для каждого пользователя или задания [5]. Docker обеспечивает полную изоляцию, так как каждый контейнер работает независимо, но требует больше ресурсов и сложной настройки. Например, проект `moodle-docker` использует контейнеры для тестирования Moodle с PostgreSQL, но не для

проверки студенческих запросов [8]. Этот подход менее практичен для образовательных систем с ограниченными ресурсами.

- Временные таблицы или ограниченные роли: в некоторых случаях платформы, такие как SQL Fiddle, могут использовать временные таблицы или ограниченные учётные записи с минимальными привилегиями для изоляции [6]. Однако такие методы менее гибки и могут быть уязвимы при неправильной настройке.

Таблица 1 сравнивает эти подходы по ключевым критериям.

Таблица 1: Сравнение методов изоляции пользовательских SQL-запросов

Метод	**Безопасность**	**Ресурсоёмкость**	**Простота настройки**
Схемы PostgreSQL	Высокая	Низкая	Средняя
Docker-контейнеры	Высокая	Высокая	Низкая
Временные таблицы	Средняя	Средняя	Высокая

2.4. Выводы

Обзор выявил, что CodeRunner, несмотря на свою универсальность, не оптимален для проверки SQL-запросов к PostgreSQL из-за низкой гибкости, сложного интерфейса и потенциальных проблем с Jobe-сервером. Предыдущая работа автора по расширению CodeRunner подтвердила эти ограничения, особенно при попытке внедрения новых функций. Анализ альтернативных платформ показал, что использование схем PostgreSQL для изоляции запросов является эффективным и ресурсосберегающим подходом, подходящим для образовательных целей. Поэтому для плагина PostgreSQL Runner выбран метод изоляции с помощью схем, что обеспечивает безопасность, удобство и совместимость с Moodle.

3. Описание решения

Разработанный Moodle-плагин PostgreSQL Runner позволяет студентам выполнять SQL-запросы к базе данных PostgreSQL в рамках тестов Moodle, а преподавателям — создавать задания с динамическими параметрами и гибкими настройками оценки. Плагин решает проблемы, выявленные в CodeRunner, такие как сложный интерфейс, низкая гибкость и ограниченная поддержка PostgreSQL.

3.1. Архитектура плагина

Плагин PostgreSQL Runner интегрируется с Moodle как новый тип вопроса, следуя стандартной структуре Moodle-плагинов. Основные компоненты включают:

- **Класс вопроса (`question.php`)**: Реализует логику обработки попыток, выполнения запросов и оценки ответов. Поддерживает рандомизацию заданий из банка вопросов.
- **Класс типа вопроса (`questiontype.php`)**: Определяет поведение вопроса, включая взаимодействие с базой данных и экспорт/импорт вопросов.
- **Рендерер (`renderer.php`)**: Отвечает за отображение интерфейса вопроса и обратной связи, включая таблицы результатов.
- **Форма редактирования (`edit_postgresqlrunner_form.php`)**: Позволяет преподавателям создавать и редактировать вопросы, задавать SQL-код, банк вопросов и настройки оценки.
- **Классы безопасности (`connection_manager.php`, `sql_validator.php`)**: Обеспечивают безопасное подключение к PostgreSQL и валидацию запросов.
- **Конфигурация базы данных (`db/install.xml`, `init.sql`)**: Определяет структуру таблиц и начальные настройки PostgreSQL.

Архитектура плагина представлена на рисунке 2. Компоненты взаимодействуют через Moodle Question Engine, обеспечивая интеграцию с тестами и другими активностями Moodle.

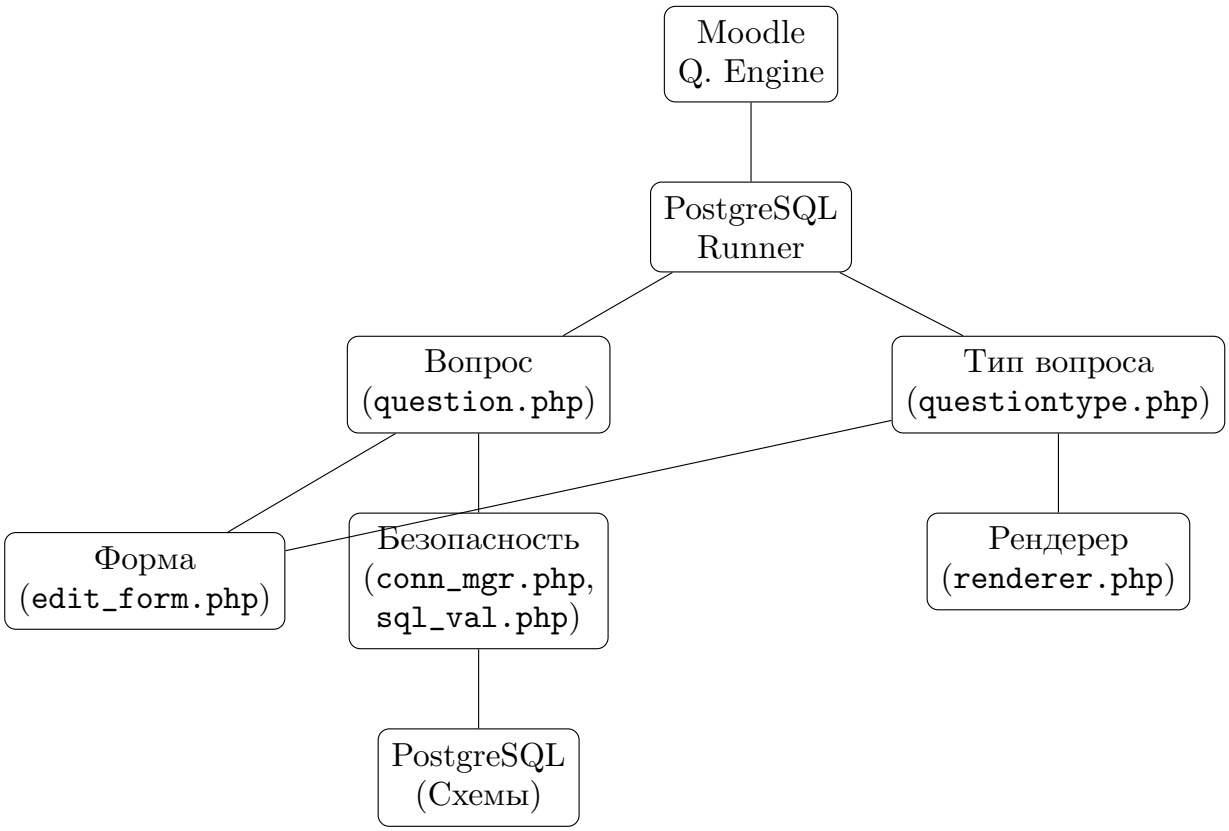

Рис. 2: Архитектура плагина PostgreSQL Runner

3.2. Безопасность

Безопасность — ключевой аспект плагина, так как выполнение пользовательских SQL-запросов сопряжено с рисками. Реализованы следующие меры:

- **Валидация запросов (`sql_validator.php`)**: Класс проверяет запросы на наличие разрешенных команд (`SELECT`, `INSERT`, `UPDATE`, `CREATE TABLE`, `BEGIN`, `COMMIT`, `ROLLBACK`, `SAVEPOINT`) и запрещает доступ к системным таблицам (например, `pg_catalog`, `information_schema`). Ограничиваются длина запроса (2048 символов) и количество токенов (200) для предотвращения атак на ресурсы.

- **Изолированные схемы и роли (`connection_manager.php`)**: Для каждой сессии создается уникальная роль (`student_<user_id>_q<question_id>_<hash>`) с ограниченными правами и схема (`student_<user_id>_q<question_id>`). Роль имеет доступ только к своей схеме и публичным таблицам для чтения, а срок действия ограничен 30 минутами.

- **Управление транзакциями**: Запросы выполняются в транзакции, которая откатывается (`ROLLBACK`) после выполнения, предотвращая постоянные изменения в базе данных.

- **Логирование ошибок**: Ошибки выполнения запросов записываются в таблицу `qtype_postgresqlrunner_security_log` для мониторинга безопасности.

Эти меры обеспечивают безопасное выполнение запросов, минимизируя риски для базы данных и других пользователей.

3.3. Выполнение и оценка запросов

Процесс выполнения и оценки запросов включает следующие этапы, реализованные в `question.php`:

1. **Инициализация окружения**: Перед выполнением запроса выполняется SQL-код из поля `environment_init`, создающий таблицы и данные. Если используется банк вопросов, выбирается случайное задание, а параметры заменяются значениями (например, `{{salary}}` на `50000`).

2. **Валидация запроса**: Запрос студента проверяется классом `sql_validator.php` на безопасность.

3. **Выполнение запроса**: Запрос выполняется в изолированной схеме через `connection_manager.php`. Для `SELECT`-запросов результаты извлекаются напрямую, для других типов запросов может выполняться дополнительный `SELECT`-запрос из поля `extra_code`.

4. **Сравнение результатов**: Результаты сравниваются с ожидаемыми:

   - **Точное совпадение**: Проверяется идентичность имен полей и данных с учетом чувствительности к регистру и порядка строк.
   - **Частичное совпадение**: Игнорируются имена полей, но данные должны совпадать.

   Настройки `case_sensitive` и `allow_ordering_difference` определяют, учитываются ли регистр и порядок строк.

5. **Оценка и обратная связь**: На основе сравнения присваивается оценка. Обратная связь включает таблицы результатов и сообщения об ошибках.

Рисунок 3 иллюстрирует процесс выполнения и оценки.

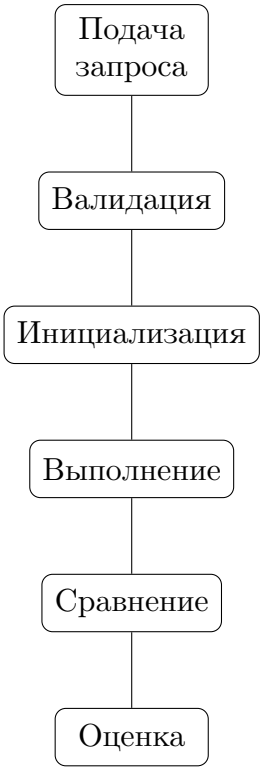

Рис. 3: Процесс выполнения и оценки SQL-запроса

3.4. Интерфейс

Плагин предоставляет два интерфейса:

- **Интерфейс преподавателя (`edit_postgresqlrunner_form.php`)**: Форма позволяет задавать SQL-код, банк вопросов в формате JSON, код инициализации окружения, дополнительный код для оценки и настройку (тип оценки, чувствительность к регистру, порядок строк). Кнопка «Проверить SQL» использует AJAX (`validate_sql.js`, `validate_sql.php`) для проверки синтаксиса и безопасности запросов.

- **Интерфейс студента (`renderer.php`)**: Студенты видят текст вопроса, текстовое поле для ввода запроса (с шаблоном, если задан) и кнопку отправки. После выполнения отображается обратная связь с таблицами результатов или сообщениями об ошибках, стилизованными через `styles.css`.

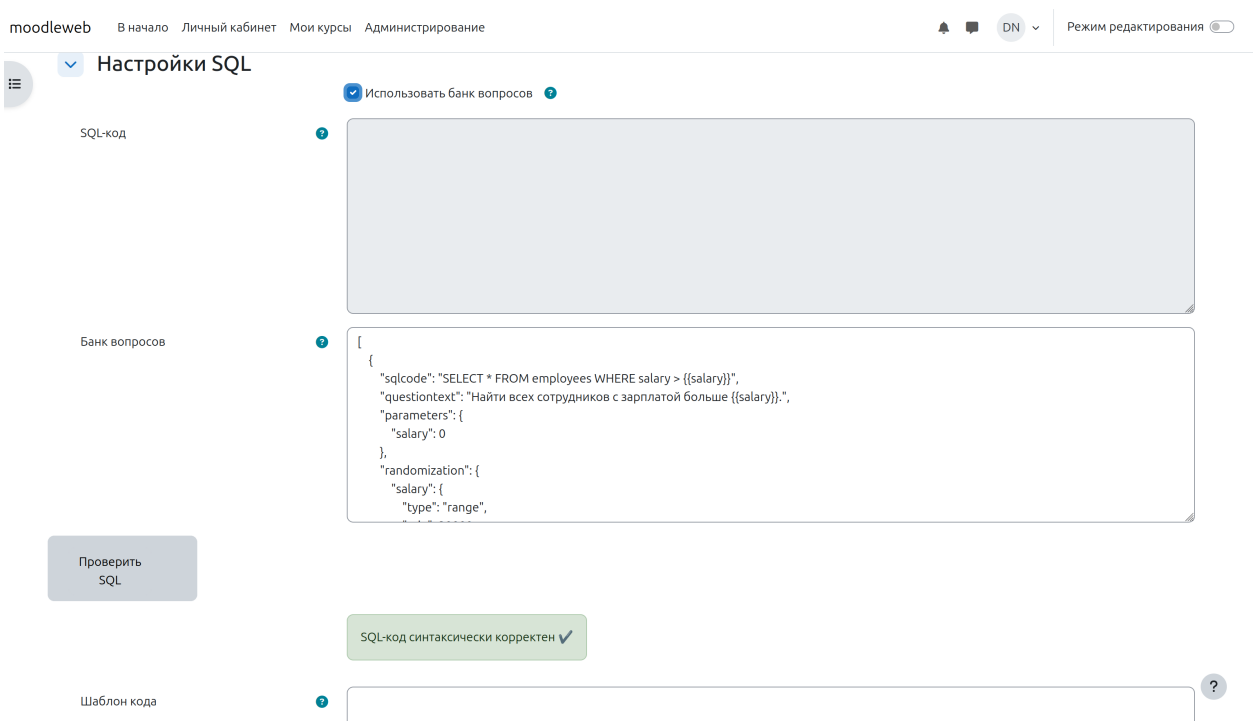

Рис. 4: Пример заполнения формы вопроса в PostgreSQL Runner

3.5. Конфигурация и развертывание

Для использования плагина администраторы выполняют следующие шаги:

1. **Установка**: Плагин устанавливается через интерфейс Moodle, загружая ZIP-архив.

2. **Настройка базы данных**: Файл `config.php` содержит параметры подключения к PostgreSQL (хост, имя базы, пользователь, пароль). Скрипт `init.sql` настраивает начальные роли и права.

3. **Создание таблиц**: Таблицы `qtype_postgresqlrunner_options`, `qtype_postgresqlrunner_session` и `qtype_postgresqlrunner_roles` создаются автоматически через `install.php` и `upgrade.php`.

4. Использование

Как уже было описано ранее, плагин позволяет преподавателям формировать задания для проверки знаний студентов, а студентам — выполнять собственные SQL-запросы к PostgreSQL без установки локального сервера или дополнительных зависимостей.

4.1. Создание вопроса

Преподаватели создают вопросы через интерфейс Moodle, следуя этим шагам:

1. Войдите в курс Moodle как преподаватель и перейдите в банк вопросов.

2. Выберите тип вопроса «PostgreSQL Runner».

3. Укажите название вопроса и текст, например, «Найдите сотрудников с зарплатой выше заданного значения».

4. Выберите, использовать ли одиночный SQL-код или банк вопросов. Для банка вопросов введите JSON-массив задач, каждая из которых содержит `sqlcode`, `questiontext` и `parameters` для рандомизации. Пример:

```
[
  {
    "sqlcode": "SELECT * FROM employees WHERE salary > {{salary}}",
    "questiontext": Найти" всехсотрудниковсзарплатойбольше     {{salary}}.",
    "parameters": {
      "salary": 0
    },
    "randomization": {
      "salary": {
        "type": "range",
        "min": 30000,
        "max": 70000
      }
    }
  },
  {
```

```
    "sqlcode": "SELECT * FROM employees WHERE department = {{department}}",
    "questiontext": Найти" всехсотрудниковвотделе    {{department}}.",
    "parameters": {
      "department": ""
    },
    "randomization": {
      "department": {
        "type": "list",
        "options": ["HR", "IT"]
      }
    }
  }
]
```

Листинг 1: Пример банка вопросов

5. Задайте SQL-код инициализации окружения для создания таблиц и данных, например:

```sql
CREATE TABLE employees (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100),
  salary INTEGER,
  department VARCHAR(50)
);

INSERT INTO employees (name, salary, department) VALUES
  ('Alice', 60000, 'IT'),
  ('Bob', 45000, 'HR'),
  ('Charlie', 55000, 'IT'),
  ('David', 40000, 'Marketing');
```

Листинг 2: Пример инициализации окружения

6. Укажите шаблон кода (опционально), который будет предзаполнен для студентов.

7. Настройте параметры оценки: точное или частичное совпадение, чувствительность к регистру, учет порядка строк.

8. Нажмите кнопку «Проверить SQL» для проверки синтаксиса и безопасности через AJAX (`validate_sql.js`, `validate_sql.php`).

9. Сохраните вопрос для использования в тестах.

4.2. Взаимодействие студентов

Студенты взаимодействуют с вопросами PostgreSQL Runner в тестах Moodle следующим образом:

1. В тесте отображается текст вопроса с замененными параметрами, например, «Найдите сотрудников с зарплатой выше 50000».

2. Студент вводит SQL-запрос в текстовое поле, которое может быть предзаполнено шаблоном, например:

```
SELECT * FROM employees WHERE salary >
```

Листинг 3: Пример шаблона для студента

3. После отправки запрос выполняется в изолированной схеме PostgreSQL. Плагин сравнивает результат с ожидаемым, используя заданные параметры оценки.

4. Студент получает обратную связь:

   - При правильном ответе: «Правильно! Ваш запрос дал ожидаемый результат.» и таблица результатов.
   - При неправильном: «Неправильный результат.» с таблицами фактического и ожидаемого результатов.
   - При ошибке: сообщение, например, «Ошибка запроса: Запрещенная команда».

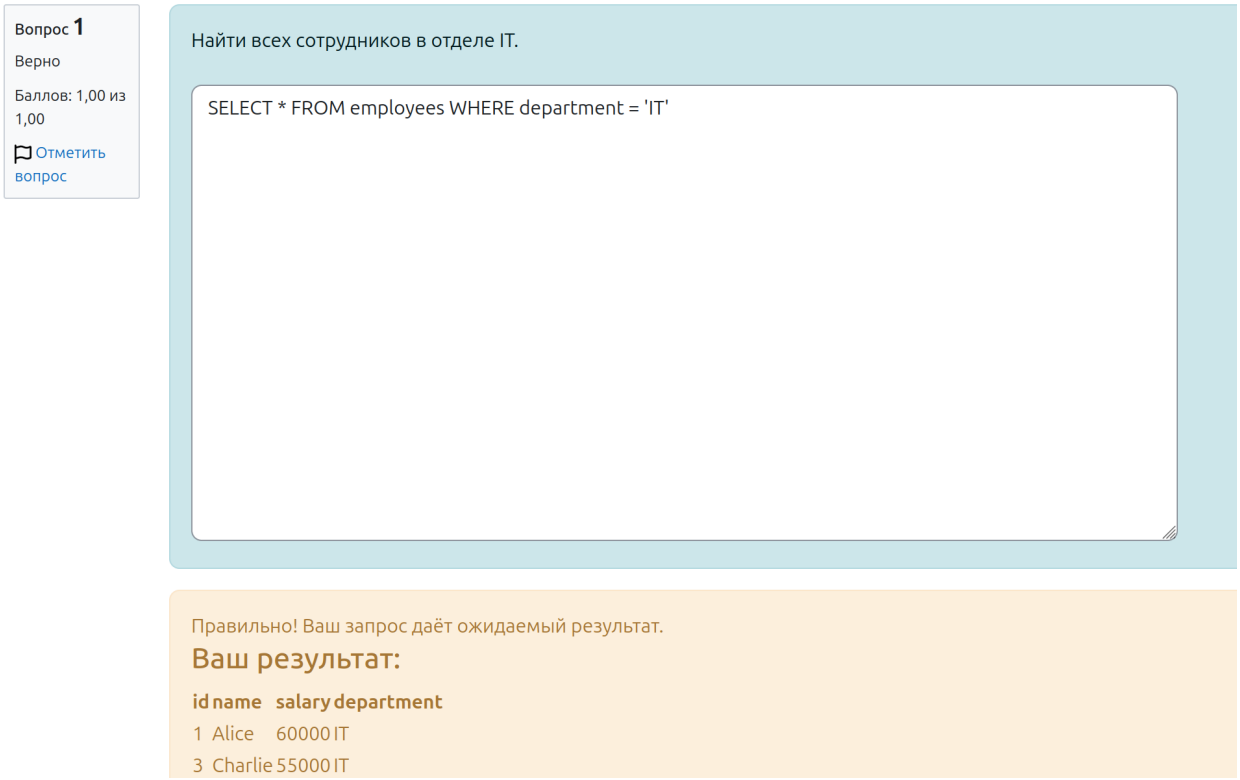

Рис. 5: Пример обратной связи для студента

4.3. Проверка уязвимостей

- **Запрещённые команды.**
  Валидатор (`sql_validator.php`) отслеживает попытки выполнения потенциально опасных операторов, таких как `DROP DATABASE`, `ALTER ROLE`, `COPY TO PROGRAM`, `SET ROLE`, обращения к `pg_catalog` и `information_schema`. При обнаружении запрета студенту выводится сообщение об ошибке, а событие заносится в журнал безопасности. На рис. 6 показан пример, где валидатор блокирует оператор `DROP TABLE`.

- **Изоляция сессий студентов.**
  Каждому пользователю при ответе назначается уникальная роль и создаётся отдельная схема вида `student_<user_id>_q<question_id>_<hash>`.

  Use-case. Студент 1 выполняет `INSERT INTO employees VALUES (...)`. Одновременно студент 2 отправляет `SELECT * FROM employees`. Благодаря раздельным схемам второй студент

видит лишь исходное окружение, а изменения первого не затрагивают его попытку.

Защита от пересечений. Попытка студента 1 сослаться на схему студента 2 приведёт к ошибке «permission denied for schema».

Очистка. Схемы и роли удаляются плановым скриптом через 30 минут после создания, обеспечивая экономию ресурсов и конфиденциальность данных.

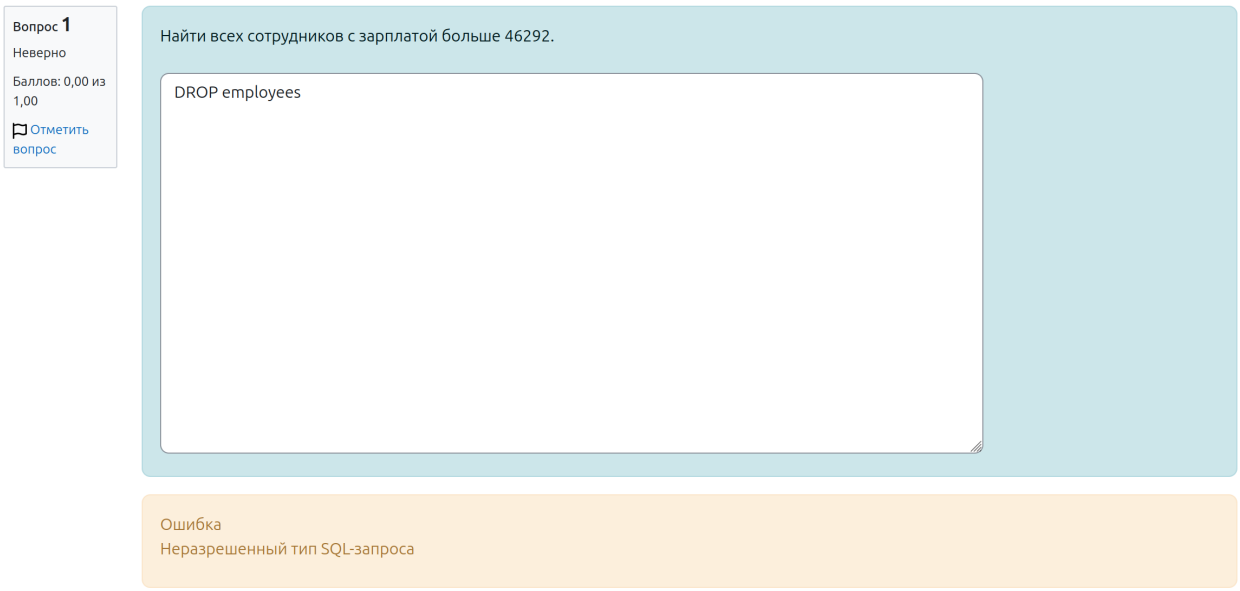

Рис. 6: Работа валидатора: попытка выполнить запрещённую команду

Примечание. Плагин развёрнут локально на `localhost`. В следующем семестре планируется провести нагрузочное тестирование для оценки масштабируемости и производительности.

Заключение

В ходе работы были выполнены следующие задачи:

- Изучена архитектура Moodle и плагина CodeRunner.
- Спроектирован и реализован плагин *PostgreSQL Runner*.
- Добавлены: банк вопросов с параметризацией, AJAX-валидация, изоляция схем, валидатор запрещённых команд.
- Проведено функциональное тестирование.
- Подготовлена инструкция по установке и использованию.

Код доступен на GitHub[1].

[1] https://github.com/niklvdanya/postgresqlrunner

Список литературы

[1] CodeRunner - Moodle question type plugin. — URL: https://moodle.org/plugins/qtype_coderunner (дата обращения: 25 июня 2025 г.).

[2] Moodle Community, Moodle Documentation. — URL: https://docs.moodle.org (дата обращения: 25 июня 2025 г.).

[3] Moodle forum: PostgreSQL, multiple schemas, my cheat is failing. — URL: https://moodle.org/mod/forum/discuss.php?d=267626 (дата обращения: 25 июня 2025 г.).

[4] PostgreSQL Documentation: Schemas. — URL: https://www.postgresql.org/docs/current/ddl-schemas.html (дата обращения: 25 июня 2025 г.).

[5] PostgreSQL in Docker: A Step-by-Step Guide for Beginners. — URL: https://www.datacamp.com/tutorial/postgresql-docker (дата обращения: 25 июня 2025 г.).

[6] SQL Fiddle - Online SQL Compiler for learning practice. — URL: https://sqlfiddle.com (дата обращения: 25 июня 2025 г.).

[7] SQLZoo. — URL: https://sqlzoo.net (дата обращения: 25 июня 2025 г.).

[8] moodlehq/moodle-docker: A docker environment for moodle developers. — URL: https://github.com/moodlehq/moodle-docker (дата обращения: 25 июня 2025 г.).