

NovaData LMS: создание SQL-заданий

Целевая аудитория: авторы курсов и методисты

Документ описывает создание SQL-заданий в LMS: какие поля заполнять, как готовить окружение, как задавать эталонный запрос и требования к решению.

Связанные документы:

- README_DOCUMENTATION.md - индекс комплекта документации и рекомендуемый порядок чтения.
- PROJECT_DOCUMENTATION.md - общая техническая документация проекта.
- API_CONTRACTS.md - API-контракты LMS/checkers.
- DEPLOYMENT_GUIDE.md - инструкция по развертыванию.
- ENVIRONMENT_VARIABLES.md - схема переменных окружения.
- OPERATIONS_RUNBOOK.md - регламент эксплуатации.
- PROGRAMMING_TASK_AUTHOR_GUIDE.md - руководство по созданию programming-заданий.

1. Назначение SQL-заданий

SQL-задание используется, когда студент должен написать SQL-запрос. LMS отправляет запрос студента и эталонный запрос во внешний SQLChecker, который выполняет оба запроса и сравнивает результат.

Поддерживаемые диалекты в модели:

- PostgreSQL;
- ClickHouse.

2. Где создается задание

В интерфейсе автора:

1. Открыть курс.
2. Перейти к структуре курса.
3. Открыть нужный модуль и урок.
4. Добавить шаг типа `sql`.

5. Открыть редактирование SQL-шага.

6. Заполнить форму и сохранить.

После создания шага LMS создает связанную запись StepContentsSQL .

3. Основные поля формы

Поле	Обязательность	Описание
Условие задачи	Да	Текст задачи для студента
Диалект СУБД	Да	PostgreSQL или ClickHouse
Тип окружения	Да	Сейчас основной режим - создание окружения
SQL скрипт для создания окружения	Да, если нет dump-файла	DDL/DML для подготовки таблиц и данных
Файл дампа SQL	Да, если нет setup script	Файл .sql , .dump или .txt
Контрольный запрос	Да	Эталонное решение автора
Требования к решению	Нет	Включение обязательных ключевых слов
Обязательные ключевые слова	Да, если включены требования	Например JOIN, GROUP BY
Таймаут выполнения	Да	Таймаут SQL-запроса
Максимальное количество строк	Да	Лимит preview результата
Баллы	Да	Баллы за задание
Максимум попыток	Да	Сколько раз студент может отправить решение
Подсказка	Нет	Подсказка для студента
Объяснение решения	Нет	Пояснение правильного решения

4. Подготовка окружения

Для SQL-задания нужно подготовить данные, на которых будут выполняться:

- эталонный запрос автора;
- запрос студента.

Есть два варианта:

- указать setup script прямо в форме;
- загрузить dump-файл.

Если указаны оба варианта, в интерфейсе и модели предусмотрен приоритет dump-файла.

5. Setup script

Setup script должен создать таблицы и наполнить их тестовыми данными.

Пример PostgreSQL:

```
CREATE TABLE users (  
    id INTEGER,  
    name TEXT,  
    age INTEGER  
);  
  
INSERT INTO users (id, name, age) VALUES  
    (1, 'Alice', 25),  
    (2, 'Bob', 17),  
    (3, 'Charlie', 30);
```

Пример ClickHouse:

```
CREATE TABLE users (  
    id UInt32,  
    name String,  
    age UInt32  
) ENGINE = Memory;  
  
INSERT INTO users VALUES  
    (1, 'Alice', 25),  
    (2, 'Bob', 17),  
    (3, 'Charlie', 30);
```

Рекомендации:

- использовать небольшие тестовые наборы данных;
- явно задавать имена колонок;
- не использовать production-данные;
- избегать случайных значений;
- добавлять строки для граничных случаев;
- проверять setup script отдельно до публикации.

6. Dump-файл

Поддерживаемые расширения:

- .sql ;
- .dump ;
- .txt .

Ограничение размера файла:

10 MB

Dump-файл подходит, если окружение большое или неудобно вставлять setup script вручную.

Рекомендации:

- не загружать секретные или персональные данные;
- очищать дампы от лишних таблиц;
- проверять, что дампы совместимы с выбранным диалектом;
- указывать в условии задачи только те таблицы и колонки, которые нужны студенту.

7. Контрольный запрос

Контрольный запрос - эталонное решение автора. SQLChecker сравнивает результат запроса студента с результатом контрольного запроса.

Пример:

```
SELECT id, name
FROM users
WHERE age >= 18
ORDER BY id;
```

Рекомендации:

- контрольный запрос должен возвращать только нужные колонки;
- порядок колонок важен для строгой проверки в LMS;
- если порядок строк важен, добавляйте `ORDER BY` ;
- не используйте `SELECT *` , если задача требует конкретные поля;
- проверяйте, что результат не зависит от случайного порядка строк.

8. Требования к решению

Если включить Требования к решению , можно указать обязательные ключевые слова.

Пример:

```
JOIN, GROUP BY, HAVING
```

Эти требования передаются в SQLChecker как `required_functions` .

Используйте требования, если важно проверить не только результат, но и подход. Например:

- задача на JOIN должна содержать `JOIN` ;
- задача на агрегацию должна содержать `GROUP BY` ;
- задача на фильтрацию агрегатов должна содержать `HAVING` .

Не стоит включать требования, если одну и ту же задачу можно корректно решить разными SQL-конструкциями.

9. Как происходит проверка

При отправке решения:

1. LMS получает `step_id` и SQL-запрос студента.
2. LMS загружает настройки `StepContentsQL`.
3. LMS формирует запрос к `SQLChecker`.
4. `SQLChecker` создает временное окружение.
5. `SQLChecker` выполняет `setup script` или `dump`.
6. `SQLChecker` выполняет контрольный запрос.
7. `SQLChecker` выполняет запрос студента.
8. LMS сравнивает `preview` результата и дополнительные признаки.
9. LMS обновляет попытки, баллы и статус прогресса.

10. Что сравнивается

В текущей логике учитываются:

- синтаксическая корректность;
- совпадение результата выполнения;
- совпадение колонок;
- совпадение количества строк;
- поэлементное совпадение значений;
- обязательные ключевые слова, если они включены;
- количество попыток;
- баллы.

11. Настройки выполнения

Рекомендуемые значения:

Тип задачи	Timeout	Max rows
Простая SELECT-задача	15-30 сек	100-1000
JOIN/GROUP BY	30-60 сек	1000
ClickHouse аналитика	30-60 сек	1000-10000

Тип задачи	Timeout	Max rows
Большой dump	60-300 сек	1000-10000

`max_result_rows` влияет на объем preview, который возвращается и отображается студенту.

12. Пример полной PostgreSQL-задачи

Условие:

Выведите `id` и `name` всех совершеннолетних пользователей. Отсортируйте результат по `id`.

Диалект:

PostgreSQL

Setup script:

```
CREATE TABLE users (  
    id INTEGER,  
    name TEXT,  
    age INTEGER  
);  
  
INSERT INTO users (id, name, age) VALUES  
    (1, 'Alice', 25),  
    (2, 'Bob', 17),  
    (3, 'Charlie', 30);
```

Контрольный запрос:

```
SELECT id, name  
FROM users  
WHERE age >= 18  
ORDER BY id;
```

Ожидаемый результат:

1 | Alice
3 | Charlie

Настройки:

Timeout: 30
Max rows: 1000
Баллы: 5
Максимум попыток: 3

13. Пример задачи на JOIN

Setup script:

```
CREATE TABLE users (  
  id INTEGER,  
  name TEXT  
);
```

```
CREATE TABLE orders (  
  id INTEGER,  
  user_id INTEGER,  
  amount INTEGER  
);
```

```
INSERT INTO users VALUES  
  (1, 'Alice'),  
  (2, 'Bob');
```

```
INSERT INTO orders VALUES  
  (1, 1, 100),  
  (2, 1, 200),  
  (3, 2, 50);
```

Контрольный запрос:

```
SELECT u.name, SUM(o.amount) AS total_amount
FROM users u
JOIN orders o ON o.user_id = u.id
GROUP BY u.name
ORDER BY u.name;
```

Обязательные ключевые слова:

JOIN, GROUP BY

14. Чек-лист перед публикацией

Перед публикацией SQL-задания проверьте:

- условие задачи понятно студенту;
- выбран правильный диалект СУБД;
- setup script или dump-файл указаны;
- setup script создает все нужные таблицы;
- данные покрывают граничные случаи;
- контрольный запрос выполняется без ошибок;
- контрольный запрос возвращает только нужные колонки;
- при необходимости указан ORDER BY ;
- обязательные ключевые слова включены только там, где это действительно нужно;
- выставлены баллы и число попыток;
- timeout соответствует сложности запроса;
- max rows не слишком мал для результата;
- dump-файл не содержит лишних или чувствительных данных.

15. Типовые ошибки автора

Ошибка	Последствие	Как исправить
Нет setup script и dump-файла	Форма не сохранится	Указать script или загрузить dump
Пустой контрольный запрос	Форма не сохранится	Добавить эталонное решение

Ошибка	Последствие	Как исправить
В контрольном запросе нет ORDER BY , но порядок важен	Студент может получить несовпадение	Добавить ORDER BY
Используется SELECT *	Проверка становится хрупкой	Явно перечислить колонки
Включены лишние required keywords	Правильные альтернативные решения отклоняются	Оставить только действительно необходимые требования
Dump не соответствует выбранному диалекту	SQLChecker вернет ошибку	Проверить dump в нужной СУБД
Слишком большой dump	Файл не загрузится или проверка будет медленной	Уменьшить данные
Нет граничных данных	Проверка слабая	Добавить данные для крайних случаев

16. Рекомендации по качеству SQL-задач

Хорошая SQL-задача должна:

- иметь небольшую, понятную схему данных;
- проверять один основной навык;
- содержать достаточные тестовые данные;
- иметь детерминированный результат;
- не зависеть от production-таблиц;
- иметь понятное условие;
- иметь объяснение решения для разбора после прохождения.

Для задач на агрегации, JOIN и оконные функции желательно отдельно проверять:

- несколько групп;
- пустые или отсутствующие связи;
- одинаковые значения сортировки;
- строки, которые должны быть отфильтрованы;
- пограничные числовые значения.