Использование метаданных для проверки SQL DDL-запросов в обучающих системах

Анастасия Дмитриевна Плужникова
Студент
Национальный исследовательский университет ИТМО
Санкт-Петербург, Россия
pluzhnikova.01@yandex.com
ORCID 0000-0000-0000-0000

Наталья Генриховна Графеева
Кандидат физико-математических наук, доцент
Национальный исследовательский университет ИТМО
Санкт-Петербург, Россия
nggrafeeva@itmo.ru
ORCID 0000-0002-9483-4748

Ольга Борисовна Егорова
Кандидат филологиеских наук, доцент ВЩЦК
Национальный исследовательский университет ИТМО
Санкт-Петербург, Россия
oegorova@itmo.ru
ORCID 0000-0002-5967-5638

Поступила в редакцию 08.12.2024
Принята 30.01.2025
Опубликована 28.02.2025

УДК 004.421.2:004.771.1(075.8)
DOI 10.25726/f9812-1752-7733-p
EDN BHXWIX
ВАК 5.8.1. Общая педагогика, история педагогики и образования (педагогические науки)
OECD 05.03.HA. EDUCATION & EDUCATIONAL RESEARCH

Аннотация

В данной работе рассматривается проблема автоматизированной проверки SQL DDL-запросов при обучении работе с базами данных в онлайн курсах или интерактивных тренажерах. Предлагается использование метаданных, доступных в СУБД PostgreSQL через системный каталог и информационную схему. Проводится сравнительный анализ этих способов, рассматриваются их преимущества и недостатки для различных типов объектов базы данных. Описывается реализация проверочных функций и использование расширения PostgreSQL для их объединения.

Ключевые слова

PostgreSQL, SQL DDL-запросы, онлайн-курсы, тренажеры, метаданные.

Введение

В последние десятилетия активно развивается цифровизация образования благодаря распространению интернет-технологий и цифровых устройств. Появляются инновационные платформы, онлайн-курсы и интерактивные ресурсы, охватывающие самые разные дисциплины. Высшие учебные заведения активно включаются в этот процесс, разрабатывая собственные онлайн-курсы и интегрируя их в учебные планы основных образовательных программ (Михайлова, 2020). Такой подход не только

повышает привлекательность программ для абитуриентов и студентов, но и позволяет оптимизировать учебный процесс, сэкономив временные и финансовые ресурсы как обучающихся, так и преподавателей. Например, в Университете ИТМО модули по Цифровой культуре и Прикладному искусственному интеллекту, включающие в себя более 40 различных образовательных траекторий каждый семестр проходят более 4000 студентов бакалавриата и магистратуры. В случае отсутствия автоматизированной проверки, реализация такого рода модулей потребовала бы огромный штат преподавателей.

Одним из ключевых навыков, который получают студенты технических специальностей, является навык работы с базами данных. Поэтому многие вузы, реализующие программы в области аналитики больших данных и других технических областях, заинтересованы в разработке онлайн-курсов и интерактивных тренажеров для отработки этого навыка.

Как правило, центральное место в курсах по основам работы с базами данных занимает работа с языком SQL (Structured Query Language), в особенности двум его подмножествам – DDL (Data Definition Language) и DML (Data Manipulation Language). SQL DDL операторы используются для создания, изменения и удаления объектов базы данных, таких как таблицы, представления, индексы и процедуры, а SQL DML операторы работают непосредственно с данными в базе.

Материалы и методы исследования

Важнейшим аспектом процесса обучения является проверка знаний. Она позволяет оценить прогресс учащегося, указать на его сильные и слабые стороны, закрепить изученный материал. Контрольные задания в онлайн формате позволяют объективно интерпретировать результаты, а также снимают с преподавателя объемную работу по проверке и обработке ответов студентов.

В системе контроля знаний существуют два основных типа тестовых заданий. Задания закрытого типа, где нужно выбрать правильный ответ из предложенных, удобны для автоматизированной проверки, но их главный недостаток – невозможность полноценно оценить глубину понимания материала, так как они часто сводятся к угадыванию.

Наиболее эффективными с педагогической точки зрения являются открытые задания, требующие самостоятельного формулирования ответа (например, в виде полноценного запроса). Такие задания наиболее точно выявляют пробелы в знаниях, поскольку требуют глубокого понимания материала, а не механического запоминания. Кроме того, они максимально приближены к профессиональной деятельности, где специалистам регулярно приходится составлять и оптимизировать запросы, находя нестандартные решения сложных задач. При этом такие задания сложнее проверять в рамках автоматизированной проверки системы онлайн курсов. Поэтому в работе будут рассмотрены задания открытого типа, где студентам предлагается написать в качестве ответа полноценный SQL-запрос.

Результаты и обсуждение

Проверить задания, где необходимо написать запрос на языке SQL DML, вполне реально: достаточно сравнить результирующую выборку по запросу студента с правильным ответом. А вот проверять SQL DDL-запросы может быть затруднительно, так как существует множество способов реализовать одно и то же задание и при проверке только текста некоторые правильные ответы могут быть не учтены. Например, решение следующего задания имеет множество вариантов текстового написания, например, корректными являются следующие варианты и не только (рис. 1). Создайте таблицу employees с полями: emp_id типа SERIAL, emp_name типа VARCHAR(100), position типа VARCHAR(50) и salary типа DECIMAL(10, 2). Установите первичный ключ по полю emp_id.

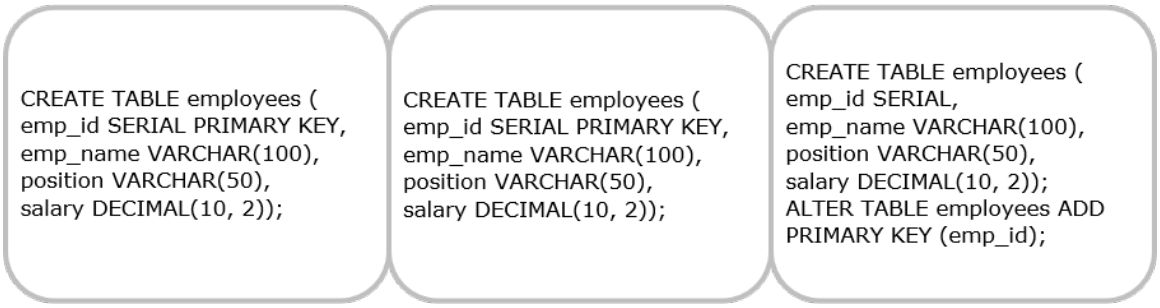

Рисунок 1. Возможные варианты решения задания на создание таблицы employees.

Именно поэтому для проверки запросов такого рода полезно сравнивать не текст запроса, а использовать метаданные. Метаданные базы данных – это информация о самой базе данных, ее структуре, объектах, их свойствах и связях между ними. Они обеспечивают ценную информацию для администраторов баз данных и разработчиков, помогая им отслеживать производительность, анализировать статистику и текущее состояние баз данных и мониторить активность пользователей.

Рассмотрим, как можно использовать метаданные для проверки SQL DDL запросов. Так как детали метаданных могут варьироваться для разных СУБД, для определенности выберем одну – PostgreSQL. Эта надежная транзакционная СУБД с открытым исходным кодом активно используется, в том числе, в онлайн-курсах и тренажерах во многих вузах.

PostgreSQL предлагает два основных подхода к работе с метаданными базы данных, каждый из которых имеет свои особенности и область применения (рис. 2) (PostgreSQL Documentation: 17, 2025).

Рисунок 2. Схемы для просмотра метаданных в PostgreSQL.

Системный каталог (схема pg_catalog) состоит из набора системных таблиц и представлений с префиксом ’pg_’, например, системное представление pg_tables содержит информацию о таблицах в текущей базе данных, а системное представление pg_indexes – об индексах. Названия столбцов обычно имеют свой 3-х буквенный префикс, соответствующий имени таблицы или представления каталога: например, все столбцы в представлении pg_attribute получили префикс ’att’. Названия объектов в таблицах словаря хранятся в нижнем регистре. Такая система наименований обеспечивает единообразие и предсказуемость структуры. Также все таблицы и представления системного каталога имеют специальный столбец с уникальным идентификатором с именем OID (Object Identifier) и одноименным типом данных. Тип oid представляет собой целочисленный тип данных с разрядностью 32 бита и автоинкрементом (Рогов, 2022).

Удобно, что схема pg_catalog включена в поиск по умолчанию, то есть можно обращаться к объектам в pg_catalog без явного указания схемы (Рогов, 2024). Системный каталог оптимизирован для PostgreSQL, поэтому обеспечивает быстрый доступ к системной информации, и отображает специфичные особенности этой СУБД.

Альтернативой системному каталогу служит стандартизированная информационная схема information_schema. Этот подход, определенный в SQL-стандарте (первоначально была стандартизирована в ISO/IEC 9075-11:1999, актуальная версия в ISO/IEC 9075-11:2023 Part 11), предлагает более универсальный интерфейс для работы с метаданными. Можно ожидать, что она будет более переносимой и стабильной, а запросы, обращающиеся к информационной схеме, будут работать

в других СУБД. Однако за счет своей универсальности information_schema не раскрывает всех возможностей PostgreSQL (например, как будет рассмотрено в пункте 2.1, не отображает никаких данных о материализованных представлениях) и может работать несколько медленнее (особенно в больших базах данных), поскольку реализована как набор представлений над системными таблицами pg_catalog.

Для доступа к метаданным используются обычные запросы SQL. При помощи команд SELECT можно получить описание любых объектов, а при помощи команд DDL можно добавлять и изменять объекты. Например, для получения списка всех таблиц в базе, можно выполнить такой запрос через pg_catalog:

```
SELECT relname FROM pg_class WHERE relkind = 'r';
```

Рисунок 3. Листинг 1. Запрос для получения списка всех таблиц в базе через pg_catalog

И такой через информационную схему:

```
SELECT table_name FROM information_schema.tables;
```

Рисунок 4. Листинг 2. Запрос для получения списка всех таблиц в базе через information_schema.

Далее будут рассмотрены различные объекты баз данных и через какие таблицы или представления можно получить их метаданные.

Основную информацию о таблицах можно получить из pg_catalog.pg_tables и information_schema.tables, а о колонках таблицы из pg_catalog.pg_attribute и information_schema.columns. Также в представлении pg_catalog.pg_class описываются отношения – таблицы и прочие объекты, имеющие столбцы или каким-то образом подобные таблицам (Рогов, 2024). В отличии от pg_tables, из pg_class можно получить OID таблицы и использовать его для связи с другими объектами (например, колонками). В следующем запросе отбираются имена всех колонок таблицы clients. Для этого таблица pg_attribute связывается с pg_class через oid. Поле relkind таблицы pg_class определяет вид отношения: r = обычная таблица, i = индекс, S = последовательность, t = TOAST таблица, v = представление, m = материализованное представление, c = составной тип, f = внешняя таблица, p = секционированная таблица, I = секционированный индекс. Условие attstattarget != 0 позволяет отфильтровать только пользовательские столбцы, исключив скрытые системные атрибуты, которые PostgreSQL добавляет к каждой таблице.

```
SELECT attname
FROM   pg_attribute t
WHERE  attrelid = (SELECT oid
                   FROM   pg_class
                   WHERE  relname = 'employees'
                          AND relkind = 'r')
       AND attstattarget != 0
```

Рисунок 5. Листинг 3. Запрос для получения имен полей таблицы employees.

Посмотреть все ограничения целостности таблицы можно с помощью information_schema.table_constraints. Тип ограничения содержится в колонке constraint_type, при этом ограничение NOT NULL будет записано как CHECK, но в constraint_name будет указание, что это именно NOT NULL. Для таблицы employees из примера отобразится следующее (цифра 1 перед not_null в имени ограничения, указывает, что это ограничение первого столбца):

Таблица 1. Описание ограничений из таблицы table_constraints

constraint_name	constraint_type
employees_pkey	PRIMARY KEY
2200_123726_1_not_null	CHECK

Также в информационной схеме есть представление information_schema.constraint_column_usage, в котором можно узнать, к каким столбцам принадлежат имеющиеся у таблицы ограничения. Ограничение NOT NULL здесь указано не будет, для информации о нем можно обратиться к представлению information_schema.columns: если столбец таблицы не может содержать NULL значения, то атрибут is_nullable этого столбца будет содержать значение NO. В pg_catalog данные о всех ограничениях, кроме NOT NULL, содержатся в представлении pg_constraint, где столбец contype описывает тип ограничения: c = CHECK, f = FOREIGN KEY, p = PRIMARY KEY, u = UNIQUE и др. Установленное ограничение NOT NULL отображается через колонку attnotnull представления pg_attribute. Также может быть полезной функция pg_get_constraintdef(oid), которая реконструирует SQL-код создания ограничения по его OID. Например, с ее помощью можно увидеть, какое условие задано в CHECK.

Информацию о представлении можно получить с помощью таблиц pg_catalog.pg_views или information_schema.views. Таблица views немного шире pg_views, но обе включают поля с указанием имени схемы, представления, владельца и SQL-запроса, определяющего представление. Представление считается отношением, поэтому его также можно найти в pg_class, а данные о его столбцах в pg_attribute или information_schema.columns.

Данные о материализованных представлениях указаны в pg_catalog.pg_matviews. Это представление содержит поля с именем схемы, мат. представления, его владельца, табличного пространства, а также отображает наличие индексов, заполнено ли представление и SQL-запрос, определяющий это мат. представление. Аналогично обычным представлениям, они также есть в pg_class. В информационной схеме материализованные представления не указаны.

Посмотреть информацию об индексах можно через таблицы pg_catalog.pg_index и pg_catalog.pg_class (в PostgreSQL индекс считается отношением – в B-деревьях узлы состоят из строк, содержащих индексированные значения и ссылки на другие узлы или на табличные строки (Рогов, 2024)). Среди всех полей наиболее актуальным при проверке решений будет indkey, содержащее массив чисел, ссылающихся на номера колонок таблицы. Для того, чтобы получить имена колонок, достаточно соединить pg_index с pg_attribute. Например employees:

```
SELECT attname
FROM  pg_attribute t
LEFT JOIN pg_index i
ON attnum = indkey[0] AND attrelid = indrelid
WHERE indrelid = (SELECT oid FROM pg_catalog.pg_class
                  WHERE relname = 'employees' AND relkind = 'r')
```

Рисунок 6. Листинг 4. Запрос для получения имен полей с одноколоночными индексами таблицы.

Информационная схема данных об индексах не отображает. Сведения о триггерах хранятся в information_schema.triggers и pg_catalog.pg_trigger. При этом, чтобы определить условия возникновения триггера (BEFORE UPDATE, AFTER INSERT и т.п.) через pg_catalog, нужно обратиться к полю tgtype, содержащем число – битовую маску условий. Также может быть полезна функция pg_get_triggerdef(oid), которая воссоздает команду триггера по его OID.

```
SELECT tgname,
       Pg_get_triggerdef(oid)
FROM   pg_trigger
WHERE  tgname = 'check_update'
```

Рисунок 7. Листинг 5. Запрос для получения имени триггера и его определяющей команды.

Таблица 2. Пример получения имени триггера и его определяющей команды

Tgname	pg_get_triggerdef
check_update	CREATE TRIGGER check_update BEFORE UPDATE ON public.employees FOR EACH ROW EXECUTE FUNCTION process_emp_audit()

А в information_schema условия хранятся в более удобном для чтения пользователем формате - полях event_manipulation (INSERT, UPDATE или DELETE) и action_timing (BEFORE, AFTER или INSTEAD OF).

Метаданные по процедурам и функциям записаны в представлениях pg_catalog.pg_proc и information_schema.routines. В pg_proc столбец prokind описывает тип процедуры, например f – обычная функция, p – процедура, w – оконная функция и другие. В представлении routines есть столбец routine_type, который может принимать два значения: PROCEDURE (процедура) и FUNCTION (функция). Полный код создания процедуры можно получить через pg_catalog с помощью функции pg_get_functiondef(oid), а внутренний код процедуры в поле prosrc представления pg_proc или в поле routine_definition представления routines.

В СУБД PostgreSQL последовательности рассматриваются как особый вид отношений, содержащих одну колонку и одну строку, поэтому часть информации можно получить в pg_class, а посмотреть основную информацию о последовательностях можно через pg_catalog.pg_sequences и information_schema.sequences. Из обеих таблиц можно получить начальное, минимальное и максимальное значения, инкремент и условие цикличности, но вот последнее значение последовательности, записанное на диск, хранится только в pg_sequences. Так как у схемы pg_catalog было найдено больше преимуществ, чем у информационной схемы, при составлении проверочных функций использовалась схема pg_catalog.

Далее рассмотрим функцию для проверки SQL DDL запросов. Для того чтобы не проверять метаданные вручную по очереди, было решено создать функции на языке PL/pgSQL, где функция бы не привязывалась к конкретным названиям объектов и, тем самым, могла быть использована при проверки нескольких вариантов заданий одного типа. Спецификация функции подразумевает передачу трех параметров: наименование схемы студента, схемы преподавателя и любой вариант SQL-запроса, реализующий поставленную задачу.

```
SELECT check_table_structure(
    'student_schema',
    'teacher_schema',
    'CREATE TABLE employees (
        emp_id SERIAL PRIMARY KEY,
        emp_name VARCHAR(100),
        position VARCHAR(50),
        salary DECIMAL(10, 2))'
);
```

Рисунок 8. Листинг 6. Пример запуска проверочной функции.

По этому запросу в схеме преподавателя создастся эталонный объект, с метаданными которого будут сравниваться метаданные в схеме студента. В данном случае по таблице, которая создалась у преподавателя будет осуществляться поиск аналогичной таблицы в схеме студента и будут сверяться поля и правила целостности.

Существует два варианта результата, который могла бы выдавать проверочная функция – логическое значение, соответствующие правильному или неправильному ответу, и дифференцированная отметка по итогам выполнения задания. Например, целесообразно возвращать полный балл за полностью выполненное задание и частичный балл, если задание выполнено не полностью. Для задания из описанного выше примера (см. рис. 1) вернем 0.2 балла за существование таблицы с правильным названием, еще 0.2 балла за правильный первичный ключ и по 0.6/n за каждую корректную колонку, где n – число колонок. При этом, если студент выполнил больше требуемого, например, добавил дополнительную колонку в таблицу, это не будет считаться ошибкой.

Аналогичным образом, был создан набор типовых заданий, направленных на работу разными типами объектов, а также написаны соответствующие им проверочные функции.

Далее рассмотрим расширение CHECK_DLL_EXT. PostgreSQL предоставляет возможность создания расширений – дополнительных модулей, которые добавляют в PostgreSQL новые функции, типы данных, операторы, агрегатные функции, процедуры и даже целые движки хранения. Они позволяют расширять функциональность СУБД без необходимости модифицировать ее ядро (Новиков, 2020). Например, одним из популярных расширений является PostGIS (PostGIS, 2024), работающий с пространственными и географическими данными.

В общем случае, расширения в PostgreSQL состоят из двух файлов:

– <extension_name>.control – control-файл, содержащий основные сведения о расширении (краткое описание, автор, название, версия, зависимости и пр.) в формате «<ключ> = <значение>»;

– <extension_name>–<version>.sql – скрипт, содержащий саму логику расширения.

Из множества упомянутых выше проверочных функций было создано расширение check_ddl_ext, состоящее из следующих файлов:

– check_ddl_ext.control;

– check_ddl_ext–1.0.sql.

Для подключения расширения эти файлы нужно добавить в директорию с остальными расширениями PostgreSQL (для Linux $(pg_config –sharedir)/extension) и по команде CREATE EXTENSION check_ddl_ext; функции добавятся в базу.

С примерами заданий, их решений и проверочными функциями для разных типов объектов можно ознакомиться в репозитории GitHub (PostgreSQL extension for ddl-queries validation, 2025).

Заключение

Таким образом, используя метаданные базы данных, можно значительно упростить процесс проверки заданий при работе с SQL DDL-запросами. При этом проверочные функции могут быть использованы для обработки нескольких вариантов одного задания, а также для реализации механизма дифференцированного оценивания в зависимости от степени выполнения задания.

Такая проверка заданий будет актуальна не только для онлайн курсов, но и для интерактивных тренажеров, позволяющих как разобраться в теоретических основах работы баз данных, так и основательно отработать полученные навыки на практике. Использование метаданных для проверки SQL DDL-запросов в интерактивных тренажерах позволяет анализировать ответ студента более детально и обеспечить предоставление обратной связи о том, что происходит во время выполнения запроса.

Список литературы

1. Mikhailova E., Boitsev A., Egorova O., Grafeeva N.G., Romanov A., Volchek D. Curriculum for Digital Culture at ITMO University // Advances in Intelligent Systems and Computing - 2020, Vol. 1161 AISC, pp. 235-244

2. Национальный исследовательский университет ИТМО. 2025. https://itmo.ru/
3. Новиков Б.А., Горшкова Е.А., Графеева Н.Г. Основы технологий баз данных: уч. пос. 2-е изд. М.: ДМК Пресс, 2020. 200 с.
4. Рогов Е., Лузанов П., Баштанов И. Организация данных. Системный каталог // Postgres professional. 2015-2022.
5. Рогов Е. PostgreSQL 17 изнутри // М.: ДМК Пресс, 2024. С. 24-28.
6. ISO/IEC 9075-11:2023 Information technology – Database languages SQL Part 11: Information and definition schemas (SQL/Schemata). 2023.
7. PostGIS. 2024. https://postgis.net/
8. PostgreSQL extension for ddl-queries validation. 2025. https://github.com/pluzhnikova/check_ddl_ext
9. PostgreSQL Documentation: 17 https://www.postgresql.org/docs/17/index.html

Using metadata to validate SQL DDL queries in training systems

Anastasia D. Pluzhnikova
Student
ITMO National Research University
Saint Petersburg, Russia
pluzhnikova.01@yandex.com
ORCID 0000-0000-0000-0000

Natalia G. Grafeeva
Candidate of Physico-Mathematical Sciences, Associate Professor
ITMO National Research University
Saint Petersburg, Russia
nggrafeeva@itmo.ru
ORCID 0000-0002-9483-4748

Olga B. Egorova
Candidate of Philological Sciences, Associate Professor at the Higher School of Digital Culture
ITMO National Research University
Saint Petersburg, Russia
oegorova@itmo.ru
ORCID 0000-0002-5967-5638

Received 08.12.2024
Accepted 30.01.2025
Published 28.02.2025

UDC 004.421.2:004.771.1(075.8)
DOI 10.25726/f9812-1752-7733-p
EDN BHXWIX
VAK 5.8.1. General pedagogy, history of pedagogy and education (pedagogical sciences)
OECD 05.03.HA. EDUCATION & EDUCATIONAL RESEARCH

Abstract

This paper examines the problem of automated verification of SQL DDL queries when learning to work with databases in online courses or interactive simulators. The use of metadata available in the PostgreSQL

database through the system catalog and information schema is proposed. A comparative analysis of these methods is carried out, their advantages and disadvantages for different types of database objects are considered. It describes the implementation of the verification functions and the use of the PostgreSQL extension to combine them.

Keywords
PostgreSQL, SQL DDL queries, online courses, simulators, metadata.

References

1. Mikhailova E., Boitsev A., Egorova O., Grafeeva N.G., Romanov A., Volchek D. Curriculum for Digital Culture at ITMO University // Advances in Intelligent Systems and Computing - 2020, Vol. 1161 AISC, pp. 235-244.
2. ITMO National Research University. 2025. https://itmo.ru/
3. Novikov B.A., Gorshkova E.A., Grafeeva N.G. Fundamentals of database technologies: a study guide. 2nd ed. M.: DMK Press, 2020. 200 p.
4. Rogov E., Luzanov P., Bashtanov I. Data organization. The // Postgres professional system directory. 2015-2022.
5. Rogov E. PostgreSQL 17 from the inside // M.: DMK Press, 2024. pp. 24-28.
6. ISO/IEC 9075-11:2023 Information technology – Database languages SQL Part 11: Information and definition schemas (SQL/Schemata). 2023.
7. PostGIS. 2024. https://postgis.net/
8. PostgreSQL extension for ddl-queries validation. 2025. https://github.com/pluzhnikova/check_ddl_ext
9. PostgreSQL Documentation: 17 https://www.postgresql.org/docs/17/index.html