УДК 004.65

5.2.2. Математические, статистические и инструментальные методы в экономике (экономические науки)

РАЗРАБОТКА ОБРАЗОВАТЕЛЬНОЙ ПЛАТФОРМЫ

Авакимян Наталья Николаевна
К.ф.–м.н., доцент
РИНЦ SPIN–код: 6082–4770
email: avnatali@mail.ru

Гучинский Никита Сергеевич
студент группы БИ2201
email: guchinsky@yandex.ru
ФГБОУ «Кубанский государственный аграрный университет», 350044, Россия, г. Краснодар, ул. Калинина 13

В современных условиях стремительного развития технологий и увеличения объемов информации повышается актуальность использования баз данных. В сфере образовательных платформ, направленных на изучение английского языка, работа с большими объемами данных играет ключевую роль в обеспечении качественного обучения и эффективного управления процессами. Компании, создающие образовательные платформы, сталкиваются с необходимостью обработки данных о пользователях, учебных материалах, расписании занятий, результатах тестов и взаимодействии между учениками и преподавателями. Это требует применения автоматизированных систем, которые упрощают работу с данными, ускоряют аналитические процессы и минимизируют вероятность ошибок. Одной из главных задач образовательных платформ является обеспечение персонализированного подхода к обучению и отслеживание прогресса пользователей. Недостаток актуальных данных о пользователях может затруднить адаптацию учебных материалов к их потребностям, тогда как избыточная информация может усложнить процесс анализа. Кроме того, в образовательной сфере важно учитывать такие аспекты, как эффективность уроков, актуальность учебных материалов, удобство интерфейса платформы и качество обратной связи с пользователями. Это делает управление данными сложным процессом, который требует четкой структуры и интеграции всех элементов системы. Применение базы данных позволяет решить указанные проблемы, обеспечивая систематизацию данных и поддержку принятия решений на основе точной информации

Ключевые слова: ОБРАЗОВАТЕЛЬНАЯ ПЛАТФОРМА, РЕЛЯЦИОННЫЕ БАЗЫ ДАННЫХ, ФОРМАТ JSON

UDC 004.65

5.2.2. Mathematical, statistical and instrumental methods in economics (Economic sciences)

DEVELOPMENT OF AN EDUCATIONAL PLATFORM

Avakimyan Natalia Nikolaevna
Cand.Phys.–Math.Sci., associate professor
RSCI SPIN–code: 6082–4770
email: avnatali@mail.ru.

Guchinsky Nikita Sergeevich
student of group BI2201
email: guchinsky@yandex.ru
Kuban State Agricultural university, 350044, Russia, Krasnodar, Kalinina, 13

In modern conditions of rapid development of technologies and an increase in the volume of information, the relevance of using databases is increasing. In the field of educational platforms aimed at learning English, working with large amounts of data plays a key role in ensuring high-quality learning and effective process management. Companies creating educational platforms face the need to process data about users, educational materials, class schedules, test scores, and interactions between students and teachers. This requires the use of automated systems that simplify data management, speed up analytical processes, and minimize the likelihood of errors. One of the main tasks of educational platforms is to provide a personalized approach to learning and tracking user progress. A lack of up-to-date user data can make it difficult to adapt training materials to their needs, while excessive information can complicate the analysis process. In addition, in the educational field, it is important to take into account such aspects as the effectiveness of lessons, the relevance of educational materials, the convenience of the platform interface and the quality of user feedback. This makes data management a complex process that requires a clear structure and integration of all system elements. The use of a database allows you to solve these problems by providing systematization of data and decision-making support based on accurate information

Keywords: EDUCATIONAL PLATFORM, RELATIONAL DATABASES, JSON FORMAT

http://dx.doi.org/10.21515/1990-4665-210-056

Компании, создающие образовательные платформы, сталкиваются с необходимостью обработки данных о пользователях, учебных материалах, расписании занятий, результатах тестов и взаимодействии между учениками и преподавателями. Это требует применения автоматизированных систем, которые упрощают работу с данными, ускоряют аналитические процессы и минимизируют вероятность ошибок.

При разработке образовательной платформы, в первую очередь, нужно установить и инициализировать проект Prisma Client для упрощенной работы с моделями в базе данных. Для этого в терминале Visual Studio Code используется команда: npm install @prisma/client npx prisma init

Структура файла schema.prisma довольно простая, сначала идет подключение к базе данных: generator client {

provider = "prisma-client-js"

} datasource db {

provider = "postgresql"

url= env("DATABASE_URL")}

Сама база данных передается в url через файл environment(env), после успешного подключения в самой базе данных также будет уведомление об этом.

После этого, можно начинать создавать модели (таблицы) с их переменными.

Написание таблиц на Typescript отличается от MySQL незначительно.

Например, если на MySQL таблицы создаются через create table profile(), то на Typescript через “model Profile{}”, в данном случае, слово «модель» отвечает за таблицу, а дальше идет просто название таблицы.

Таким образом, готовый вариант таблиц Profile и Server будет выглядеть так:

model Profile {

id String @id @default(uuid()) userId String @unique name String imageUrl String @db.Text email String @db.Text servers Server[] members Member[] createdAt DateTime @default(now()) updatedAt DateTime @updatedAt

} model Server { id String @id @default(uuid()) name String inviteCode String @unique profileId String profile Profile @relation(fields: [profileId], references: [id], onDelete: Cascade)

members Member[] createdAt DateTime @default(now()) updatedAt DateTime @updatedAt

@@index([profileId])}

Данный код показывает создание двух моделей данных Profile и Server, которые используются в рамках ORM Prisma для взаимодействия с базой данных.

В данном примере модель Profile используется для представления информации о профиле пользователя. Каждая запись этой модели включает уникальный идентификатор id, который автоматически генерируется в формате UUID. Также в модели присутствует поле userId, которое обеспечивает уникальную связь профиля с пользователем, например, в рамках внешней системы аутентификации. Поля name, imageUrl и email предназначены для хранения имени пользователя, ссылки на его аватар и адреса электронной почты соответственно. Особенностью является использование аннотации @db.Text, которая позволяет хранить текстовые данные большого объема.

Дополнительно, в модели Profile предусмотрено поле servers, которое представляет собой массив объектов модели Server. Это поле описывает связь профиля с несколькими серверами, на которых пользователь может быть активен. Также модель включает поля createdAt и updatedAt, которые

автоматически фиксируют время создания и последнего обновления записи.

Модель Server, в свою очередь, описывает структуру данных для хранения информации о серверах. Здесь также используется уникальный идентификатор id, автоматически генерируемый в формате UUID. Поле name предназначено для хранения названия сервера, а поле inviteCode – для уникального кода приглашения, который обеспечивает доступ пользователей к серверу.

Связь между моделью Server и моделью Profile реализована через поле profileId, которое ссылается на идентификатор профиля. Эта связь дополнительно описана с помощью аннотации @relation, которая уточняет, что поле profileId в модели Server связано с полем id в модели Profile. При этом атрибут onDelete: Cascade задает каскадное удаление – если профиль пользователя будет удален, все связанные с ним серверы также автоматически удалятся.

Для оптимизации запросов в модели Server создан индекс на поле profileId. Поля createdAt и updatedAt, аналогичные таковым в модели Profile, фиксируют время создания и изменения записи.

Таким образом, данный код демонстрирует гибкую и эффективную схему для работы с реляционными базами данных, где пользователи (профили) могут быть связаны с несколькими серверами, а сами модели обладают встроенными механизмами обеспечения целостности данных и поддержки каскадных операций.

Примеры SQL-запросов:

– создание нового профиля:

INSERT INTO “Profile” (id, userId, name, imageUrl, email, createdAt, updatedAt)

VALUES(‘uuid-generated’, ‘user-id-123’, ‘John Doe’, ‘http://image.url’, ‘john@example.com’, NOW(), NOW());

– создание нового сервера, связанного с профилем:

INSERT INTO “Server” (id, name, inviteCode, profileId, createdAt, updatedAt) VALUES (‘uuid-generated’, ‘My Server’, ‘unique-invite-code’, ‘profile-id’, NOW(), NOW());

– удаление профиля и связанных серверов:

DELETE FROM "Profile" WHERE id = 'profile-id';

Так как установлен onDelete: Cascade, то все связанные серверы будут удалены автоматически при удалении профиля.

На рисунке 1 представлена схема базы данных, состоящая из трех основных таблиц: Profile, Server и Member, которые структурируют информацию о пользователях системы, серверах и участниках этих серверов.

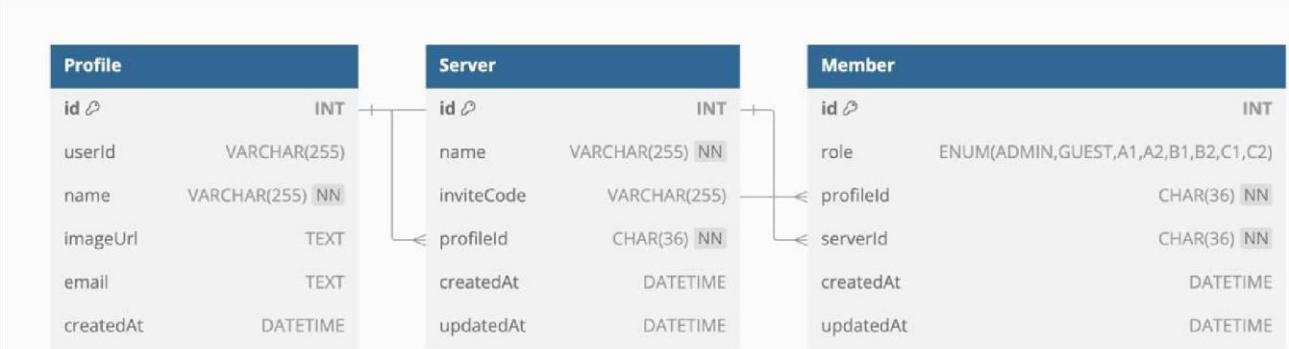

Рисунок 1 – ER-диаграмма концептуальной модели базы данных

Каждая таблица выполняет свою роль в организации данных и связей между ними.

Таблица Profile хранит основную информацию о пользователях системы. Каждая запись идентифицируется уникальным первичным ключом id, представленным типом данных INT. Помимо этого, в таблице есть поле userId типа VARCHAR(255), которое является уникальным идентификатором пользователя. Поля name (имя пользователя), imageUrl (ссылка на изображение профиля) и email (электронная почта) хранят текстовые данные о пользователе. Для отслеживания временных

изменений предусмотрены поля createdAt и updatedAt, которые фиксируют дату создания и последнего обновления записи. В итоге данная таблица служит основой для хранения персональных данных пользователей.

Таблица Server отвечает за хранение информации о серверах, которые могут быть привязаны к пользователям. Каждый сервер имеет уникальный первичный ключ id и поля name и inviteCode, хранящие название сервера и его уникальный код приглашения. Связь между сервером и пользователем реализована через внешнее ключевое поле profileId, которое ссылается на поле id таблицы Profile.

Данные о серверах в базе данных приведены на рисунке 2.

id	name	inviteCode
0b9b1547-c55e-42c0-99f…	LF	c6b61a85-99e3-4246-834…

profileId	createdAt	updatedAt
80a0dae1-e343-4327-828…	2024-10-28 23:46:06.56	2024-10-28 23:46:06.56

Рисунок 2 – Данные о серверах в базе данных

На рисунке 3 приведены данные о пользователях в базе данных.

id	userId	name	imageUrl
80a0dae1-e343-4327-828…	user_2XqzWGqpBOoSaxDqL…	GN 28	https://img.clerk.com/…

email	createdAt	updatedAt
nikguchi28@gmail.com	2024-10-28 23:45:46.556	2024-10-28 23:45:46.556

Рисунок 3 – Данные о пользователях в базе данных

Для того, чтобы данные о профиле попали в базу данных, написан небольшой запрос для сохранения пользователя.

import { auth } from "@clerk/nextjs/server"; import { db } from "@/lib/db"; export const currentProfile = async () => {

const { userId } = auth(); if(!userId) { return null;

} const profile = await db.profile.findUnique({ where: { userId}

}); return profile;}

В первой строке происходит импорт функции auth из модуля @clerk/nextjs/server. Эта библиотека предоставляет возможности аутентификации пользователей. Во второй строке импортируется объект db из модуля @/lib/db, который представляет собой экземпляр базы данных.

Далее определяется асинхронная функция currentProfile, которая возвращает профиль текущего пользователя. Внутри функции вызывается метод auth(), который возвращает информацию об авторизованном пользователе. Эта информация нужна для получения значения userId – уникального идентификатора пользователя.

Следующим шагом проверяется, существует ли userId. Если пользователь не авторизован или идентификатор не был получен, функция возвращает null. Это предотвращает выполнение ненужных запросов к базе данных для неавторизованных пользователей.

Если userId доступен, выполняется асинхронный запрос к базе данных с использованием функции db.profile.findUnique. В данном случае происходит поиск записи в таблице profile, где поле userId совпадает с переданным значением. Метод findUnique возвращает единственный результат, если он существует, или null, если совпадений не найдено. Запрос выполняется с использованием условия в параметре where, что позволяет точно указать критерий поиска.

Следующим шагом будет добавление «функции участников», так как к курсам должны прикрепляться определенные студенты. В английском языке много уровней владения, поэтому очень важно сделать разделение по ним, а также назначить администраторов и просто гостей на данной платформе.

Самым лучшим способом для реализации служит создание «ролей» на сервере:

enum MemberRole {

ADMIN

GUEST

A1

A2

B1

B2

C1

C2 } model Member {

id String @id @default(uuid()) role MemberRole @default(GUEST) profileId String profile Profile @relation(fields: [profileId], references: [id], onDelete: Cascade) serverId String server Server @relation(fields: [serverId], references: [id], onDelete: Cascade) createdAt DateTime @default(now()) updatedAt DateTime @updatedAt

@@index([profileId])

@@index([serverId])}

Таким образом, представлена модель Member и перечисление ролей MemberRole. Эти модели описывают участников системы, их роли и связь с профилями пользователей и сервером.

Модель Member отвечает за хранение информации о пользователях, которые являются участниками сервера. Каждый участник на платформе имеет уникальный идентификатор id, который автоматически генерируется. Поле role определяет роль участника, для которой используется перечисление MemberRole. Это нужно, в первую очередь, для перечисления определенных ролей, таких как ADMIN, GUEST и другие, что позволяет строго контролировать допустимые значения. По умолчанию, при заходе и регистрации на платформе, участник имеет роль: GUEST.

Перечисление MemberRole представляет собой список доступных ролей, которые может занимать участник. В него включены роли ADMIN и

GUEST, а также дополнительные категории (A1, A2, B1, B2, C1, C2), которые нужны для более точной системы определения уровня знаний каждого студента. Админы сервера смогут менять роли каждого конкретного студента, в специальном модальном окне, в котором также будет показываться и пользователи платформы, в том числе и сами администраторы. Система поможет учителям сразу видеть уровень всех студентов, их фамилии и почту в одном месте, помимо этого, в случае необходимости, смогут и исключить участника с сервера. Такое перечисление делает управление ролями более простым.

Связь участника с другими таблицами базы данных реализована через поля profileId и serverId. Поле profileId отвечает за связь участника с профилем пользователя, а serverId указывает на сервер, к которому он принадлежит. Эти поля являются внешними ключами.

Индексы для полей profileId и serverId нужны для более быстрой работы платформы и запросов, так как данных в базе будет очень много.

Для того чтобы изменить роль участников сервера, будет использоваться данный API код:

```
const server = await db.server.update({ where: {
id: serverId, profileId: profile.id, }, data: {
members: { update: {
where: { id: params.memberId, profileId: {
not: profile.id}
}, data: {
role}}}
}, include: { members: {
include: {profile: true,},
orderBy: {role: "asc"}}}});
```

Принцип действия данного запроса достаточно прост, db.server.update используется для обновления записи в таблице server.

Обновляемая запись должна соответствовать id сервера (переменная serverId) и profileId, равному profile.id. В таком случае, изменения применятся к конкретному пользователю конкретного сервера.

Внутри data указывается новое значение для поля role, которое будет обновлено у выбранного нами участника (member). Также здесь присутствует проверка на то, чтобы никто не мог изменить роль администраторам. В данном случае проверкой служит условие «where:{profileId:{not: profile.id}}» Таким образом, этот запрос позволяет изменить роль конкретного участника сервера. Если у студента была роль с уровнем знания языка A1, но он перешел на уровень A2, учителя могут выбрать новую роль, соответствующую его уровню.

Для более легкого представления этого сложного и многоуровневого запроса, отразим запись этого запроса на MySQL:

– обновление роли члена сервера, который соответствует условиям:

UPDATE members SET role = ‘new_role’ - указываем новую роль студента WHERE id = ‘memberId’ AND profileId != 'profileId';

– включение профилей всех участников и сортировка по роли:

SELECT members*, profiles* FROM members JOIN profiles ON members.profileId = profiles.id

WHERE members.serverId = 'serverId' ORDER BY members.role ASC;

Такие параметры как memberId и serverId меняются идентификаторами определенных пользователей, роли которых нужно поменять.

Помимо функции изменения ролей, также нужно создать и функцию удаления пользователей:

```
const server = await db.server.update({ where: {
id: serverId,
profileId: profile.id,
}, data: {
```

members: { deleteMany: { id: params.memberId, profileId: { not: profile.id}

}}}, include: {

members: { include: { profile: true,},

orderBy: { role: "asc",},},},});

В данном случае значимых отличий не так много. Во-первых, отличается блок data, а во-вторых, методы функций. Здесь это DELETE, в первом же случае – PATCH.

Внутри блока data происходит операция deleteMany, которая удаляет все записи в таблице members, соответствующие идентификатором тех пользователей, которых хотим удалить. Аналогично реализована проверка администраторов, чтобы их нельзя было удалить с сервера.

Пример SQL-кода:

– удаление записей из таблицы 'members', которые соответствуют условиям:

DELETE FROM members WHERE id = 'memberId' AND profileId != 'profileId';

– получение обновленного списка членов сервера с их профилями, отсортированного по роли:

SELECT members*, profiles* FROM members JOIN profiles ON members.profileId = profiles.id WHERE members.serverId = 'serverId' ORDER BY members.role ASC;

Разработка описанной образовательной платформы обеспечит персонализированный подход к обучению и позволит отслеживать прогресс пользователей.

Список использованной литературы:

1. Дадян, Э. Г. Современные технологии программирования. Язык C#: учебник: в 2 томах. Том 2. Для продвинутых пользователей / Э.Г. Дадян. - Москва: ИНФРА-М, 2021. - 335 с.

2. Мартыненко, Т. В. Основы визуального программирования в среде Visual Studio на базе C#: учебное пособие / Т. В. Мартыненко, В. В. Турупалов, Н. К. Андриевская; под общ. ред. к. т. н., проф. В. В. Турупалова. - Москва; Вологда: Инфра-Инженерия, 2023. - 232 с.

3. Шустова, Л. И. Базы данных: учебник / Л.И. Шустова, О.В. Тараканов. - Москва: ИНФРА-М, 2023. - 304 с.

References:

1. Dadjan, Je. G. Sovremennye tehnologii programmirovanija. Jazyk S#: uchebnik: v 2 tomah. Tom 2. Dlja prodvinutyh pol'zovatelej / Je.G. Dadjan. - Moskva: INFRA-M, 2021. - 335 s.

2. Martynenko, T. V. Osnovy vizual'nogo programmirovanija v srede Visual Studio na baze C#: uchebnoe posobie / T. V. Martynenko, V. V. Turupalov, N. K. Andrievskaja; pod obshh. red. k. t. n., prof. V. V. Turupalova. - Moskva; Vologda: Infra-Inzhenerija, 2023. - 232 s.

3. Shustova, L. I. Bazy dannyh: uchebnik / L.I. Shustova, O.V. Tarakanov. - Moskva: INFRA-M, 2023. - 304 s.