КОМП'ЮТЕРНІ НАУКИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

УДК 004.4'22:004.652 **DOI: 10.30977/BUL.2219-5548.2023.100.0.7**

МЕТОД ГЕНЕРАЦІЇ ПРОГРАМНОГО КОДУ ORM-МОДЕЛЕЙ НА ОСНОВІ СХЕМ РЕЛЯЦІЙНИХ БАЗ ДАНИХ

Долгополов К. В., Імангулова З. А.
Харківський національний університет радіоелектроніки

Анотація. *Визначено завдання розроблення методу для автоматичної генерації програмного коду ORM-моделей (Object-Relational Mapping). Здійснено аналіз можливих способів та загальної структури презентації схеми даних в текстовому форматі. Сформовано схему даних бази даних конфігураційної інформації, необхідної для генерації коду. Подано етапи здійснення генерації програмного коду для ORM-моделей. Результати проектування ORM-моделей можуть використовуватися розробниками програмного забезпечення під час реалізації рівня даних інформаційних систем, що дасть змогу зекономити час, що витрачається на створення віртуальної бази даних, а також уникнути потенційних помилок у коді.*
Ключові слова: *генерація коду, конфігураційний файл, реляційні бази даних, схема даних, ORM-бібліотека.*

Вступ

З плином часу збільшилися темпи розвитку технологій, настала нова епоха технічної еволюції – цифрова ера, яка привнесла значні зміни в галузь інформаційно-комп'ютерних технологій, зокрема в галузь опрацювання та зберігання інформації.

Майже кожна програмна система працює з безпосереднім використанням бази даних, яка є тим самим «сховищем зі звітами». Спектр застосування систем управління базами даних (СУБД) доволі широкий. Бази даних використовуються під час розроблення програмних застосунків у багатьох сферах, зокрема у виробництві, промисловості, маркетингу, у мобільних пристроях, фінансовій та банківській сферах, на телебаченні, у телекомунікаціях та рекламі.

Крім переваг використання БД, є й недоліки, зокрема процес їхнього проектування та інтеграції у програмний код. Цю проблему намагаються вирішувати через використання ORM-бібліотек, які надають зручний програмний інтерфейс для здійснення базових запитів. Але сам процес налаштування ORM-бібліотеки в проекті зазвичай є трудомістким та тривалим, зокрема багато зусиль вимагає написання програмного коду ORM-моделей.

Аналіз публікацій

На стадії аналізу проблеми часто здійснюють пошук та аналіз наявних аналогів подібних систем. Метою цього етапу є аналіз роботи конкурентів. Результати такого аналізу застосовуються на стадіях проектування та розроблення з метою отримання системи, яка буде позбавлена об'єктивних недоліків та міститиме найкращий функціонал з конкурентних систем. Під час аналізу можуть бути визначені певні бізнес-функції, критичні баги або так звані «killer features» [1].

Процес аналізу систем з можливістю генерації ORM-моделей був складним через наявність таких факторів: по-перше, було визначено, що наявні платформи для генерації моделей є певним консольним рішенням у межах бібліотеки конкретної мови програмування, а це ускладнює їх практичне дослідження [2, 3]; по-друге, серед засобів, які мають хоча б якийсь графічний інтерфейс для роботи с БД, реалізована генерація лише для однієї, конкретної ORM-бібліотеки, замість певної глобальної реалізації [4]; потретє, важливим фактором, що блокує можливість дослідження подібних систем, є те, що часто процес генерації файлів інкапсульований всередині самого засобу генерації, код застосунку занадто обфускований або складається з великої кількості файлів [5, 6, 7].

Таким чином, на основі здійсненого аналізу, було зроблено висновок, що рішенням проблеми, яка вирішується, буде розроблення вебзастосунку, який дозволить забезпечити IT-фахівців єдиною цілісною платформою з можливістю моделювання схеми даних для всіх сучасних реляційних баз даних (РБД), та подальша автоматизована генерація коду для заздалегідь вибраної ORM-бібліотеки [8].

Мета та постановка завдання

Метою роботи є розроблення алгоритму автоматичної генерації програмного коду для ORM-моделей на основі вибраної ORM-бібліотеки та схеми даних, поданої як конфігураційний файл. Цей підхід дасть можливість кінцевому користувачу економити час, що витрачається на створення віртуальної бази даних, а також уникнути потенційних помилок у коді.

Для досягнення поставленої мети необхідно визначити формат та структуру презентації схеми даних у текстовому форматі, вибрати принцип збереження необхідних конфігураційних даних для генерації коду та описати етапи самого процесу генерації.

Текстова презентація схеми даних

Існує низка форматів серіалізації загального призначення, які можуть подавати складні структури даних у текстовому форматі. Найбільш відомими є XML, JSON та YAML [9, 10]. Кожен з них має свої переваги та недоліки, особливості використання та синтаксису.

XML є видом мови розмічування, що розширюється. Такі файли формату є документами, що використовують теги з метою визначення об'єктів, а також їхніх атрибутів. Формат XML, на відміну від HTML, надає користувачу можливість самостійно задавати теги, які застосовує мова XML [11]. Формат XML як формат наведення даних є всесвітньо відомий потужний та поширений інструмент, який надає користувачу великий набір можливостей для наведення даних у потрібному виді, зокрема атрибутів, тегів, областей імен тощо. Але його «багатогранність» у деяких випадках є недоліком, адже це призводить до значних проблем з парсингом файлу, його розмірами, складною валідацією та роботою загалом [12].

Функцію наведення структурованих даних на основі синтаксису JavaScript здійснює стандартний текстовий формат під назвою JSON (JavaScript Object Notation) [13]. JSON більш простий та «легкий» формат наведення даних на відміну від XML. Він пропонує біль простий, але достатній функціонал для наведення ієрархії даних. Це збільшує швидкість створення, валідації, опрацювання та парсингу файлу цього формату. Крім того, JSON-формат є одним з найбільш позитивних рішень для вебплатформ завдяки його рівню крос-браузерності, вбудованим інструментам роботи в рушії браузерів та мові програмування JavaScript.

YAML, або «YAML Ain't Markup Language», – це мова для зберігання інформації у форматі, що є зрозумілим для людини. Його назва розшифровується як «Ще одна мова розмітки». Однак пізніше назву змінили на «YAML не мова розмітки», щоб відрізняти його від справжніх мов розмітки [14]. Завдяки своїм можливостям серіалізації ця мова є гідною заміною таким мовами, як JSON і XML. До речі, YAML v1.2 є надмножиною JSON. YAML був створений з метою замінити JSON-формат. Він повністю підтримує його функціонал, бо є його надмножиною, виправив деякі недоліки свого попередника, має додаткові можливості, є більш лаконічним. Але, незважаючи на це, сам синтаксис мови став складнішим через наявність значної кількості системних символів та їхніх комбінацій для забезпечення нового функціонала. Це значною мірою вплинуло на швидкість серіалізації та десеріалізації.

Серед розглянутих варіантів форматів для наведення схеми даних було вибрано JSON. Це пояснюється тим, що це зручний і простий для використання формат, який забезпечує високу швидкість генерації, парсингу та передачі даних мережею. Цей формат можна використовувати під час роботи з вебсистемами, а його інтегрованість в більшість мов програмування робить реалізацію підходу генерації конфігураційних даних простим процесом.

Структура конфігураційного файлу має бути єдиною та незмінною для будь-яких РБД, що були використані на етапі моделювання. Така уніфікованість забезпечить швидкість розроблення та використання даних з конфігураційного файлу, адже не буде необхідності створювати різні алгоритми парсингу під різні бази даних, а подібна структура більшості РБД тільки сприятиме цьому. Загальну структуру конфігураційного файлу наведено на рис. 1.

Під час аналізу структури файлу можна визначити три основні компоненти:

– «metadata». Компонента зберігає ключову інформацію про конфігураційний файл, користувача системи, який його згенерував, та ключову інформацію про обрані технології – ORM-бібліотеку, РБД та мову програмування;

– «schemaData». Одна з найважливіших частин файлу, адже саме тут зберігається вся інформація про схему даних. Структура сформована у такий спосіб, щоб універсально зберігати інформацію про схему, зокрема таблиці, їхні поля, зв’язки між ними та обмежен-

ня. Кожен з ключових об'єктів схеми даних визначається власним UUID-ідентифікатором для забезпечення його унікальності та можливості посилання на нього [15];

– «settings». Вона визначається тим, що містить ключову інформацію, яка необхідна для безпосереднього процесу генерації моделей. Це можуть бути налаштування визначення імен у моделі чи параметри мови програмування.

```
{
  "metadata": {
    "createdAt": "09-12-2022T22:08:37Z",
    "userId": "USER_ID",
    "rdb": {
      "version": "14.5.1",
      "type": "postgresql"
    },
    "orm": {
      "version": "6.0",
      "type": "sequelize"
    },
    "pl": {
      "type": "nodejs",
      "version": "18.12.1",
      "ext": ".js"
    }
  },
  "schemaData": {
    "tables": [{
      "_id": "TABLE_ID",
      "tableName": "user",
      "modelName": {
        "singularName": "user",
        "pluralName": "users"
      },
      "fields": [{
        "_id": "FIELD_ID",
        "domainType": "string ",
        "dataType": "varchar",
        "name": "email",
        "isPrimaryKey": true,
        "isNull": false,
        "isAutoIncrement": true,
        "defaultValue": "test@mail.com"
      }],
      "constraints": [{
        "_id": "CONSTRAINT_ID",
        "type": "unique",
        "fields": ["FIELD_ID_1",
"FIELD_ID_2"]
      }]
    }],
    "relations": [{
      "_id": "RELATION_ID",
      "type": "ONE_TO_MANY",
      "sourceTableId": "TABLE_ID",
      "targetTableId": "TABLE_ID",
      "onDelete": "cascade",
      "onUpdate": "restrict"
    }]
  },
  "settings": {
    "templates": {
      "pkConstraint": "pk_name",
      "fkConstraint": "fk_name",
      "uniqueConstraint": "uq_name",
      "indexConstraint": "idx_name",
      "file": "file_name",
      "archive": "archive_name"
    },
    "indetation": {
      "type": "space",
      "count": 2
    },
    "export": "named",
    "archive": "rar"
  }
}
```

Рис. 1. Лістинг схеми конфігураційного файлу

Сховище конфігураційних даних

Описуючи етапи генерації файлу, необхідно визначитись зі сховищем конфігураційних даних сервісу, зокрема вибрати спосіб зберігання та загальну структуру цього сховища. Конфігураційними даними сервісу є шаблони коду, перелік ключових програмних слів, списки співвідношень структур бібліотеки та інші елементи, необхідні для безпосереднього конструювання коду моделі. Для розроблення схеми сховища цієї інформації спочатку потрібно вибрати один зі способів збереження. Як варіанти було розглянуто реляційну базу даних, документоорієнтовану базу даних та локальний конфігураційних файл. Перевагу було надано документоорієнтованій базі даних на кшталт MongoDB. Сховища такого типу дозволяють створювати колекції документів будь-якої структури з можливістю посилатись на інші об'єкти. Така гнучкість є перевагою для збереження складних програмних параметрів. Завдяки швидкості операцій зчитування та зручності побудови запитів ця база даних є інструментом для пошуку та отримання потрібної інформації на етапах генерації коду [16]. База даних web-застосунку для генерації ORM-моделей містить 11 колекцій даних:

– «OrmSettingsTypes». Колекція зберігає всі характеристики та налаштування, які можуть вплинути на код чи його структуру під час генерації. Наприклад, це можуть бути версія ORM-бібліотеки, особливості мови програмування, тип наведення моделі тощо;

– «OrmSettingsValues». Колекція містить всі можливі значення налаштувань, визначених у колекції «OrmSettingsTypes»;

– «OrmDataTypes». Колекція зберігає всі можливі значення типів даних, які підтримує бібліотека. Крім самих типів, вона зберігає і фрагмент коду, який потрібно додати до файлу для відповідного типу даних;

– «DataTypesMapping». Колекція здійснює функцію визначення типу даних в ORM-бібліотеці для поля на основі типу домену та типу даних поля, отриманих зі схеми бази даних відповідного атрибута змісту. Процес отримання типу даних бібліотеки буде описано в цій роботі;

– «Constraints». Колекція містить інформацію про обмеження, яких має дотримуватись система, та параметри генерації, щоб отримати фрагмент коду, який є валідним та функціональним для зазначених параметрів. Саме в цій колекції використовуються посилання на характеристики та їхні значення з документів колекцій «OrmSettingsTypes» та «OrmSettingsValues»;

– «RelationTypeMapping». Колекція містить співвідношення між типами відносин, що зберігаються в згенерованій схемі бази даних та підтримуються бібліотекою. Водночас у цій колекції застосовується посилання обмеження, адже вона зберігає і фрагмент коду, що відповідає певному типу зв'язку;

– «TemplateTypes». Колекція містить перелік типів шаблонів коду, які застосовуються в процесі генерації моделі, зокрема «import», «export», «class», «classProperty», «field», «index», «constraint», «relation»;

– «Markers». Колекція зберігає значення маркерів коду, які застосовуються у відповідних шаблонах коду. Маркери використовують для додавання згенерованого коду до шаблону;

– «Keywords». Колекція здійснює функцію збереження ключових слів, які можуть застосовуватись в шаблоні коду. Ключові слова є відповідними місцями вставки назв моделей, обмежень, значень параметрів, типів даних тощо;

– «Properties». Колекція містить програмні назви, які використовуються для зазначення параметрів полів, наприклад назви параметрів, що відповідають за тип даних, автоінкремент, первинний ключ тощо;

– «Templates». Одна з найважливіших колекцій бази даних, адже зберігає документи з шаблонами коду для тих частин файлу моделі, які зазначені в колекції «TemplateTypes». Кожен шаблон також визначається своїми обмеженнями використання.

Технологія генерації коду

Генерація коду для ORM-моделей – складний і заплутаний процес, що містить велику кількість залежностей, в якому потрібно дотримуватись багатьох передумов та розв'язання необхідних задач. Різноманітність ORM-бібліотек для мов програмування має тільки ускладнити підготовку алгоритмів для генерації файлів. Різні мови програмування та бібліотеки обумовлюють відмінності хоча б щодо ключових програмних слів, підтримуваних структур даних або синтаксису. Проте, як і реляційні бази даних, ORM-бібліотеки майже не відрізняються одна від одної за принципом дії та за структурою моделі. Тому й алгоритми для побудови моделей будуть подібні за принципом дії, а сховища з конфігураційними даними – за своєю схемою. Це дозволяє узагальнити та уніфікувати етапи процесу генерації коду для всіх бібліотек, водночас усі технічні відмінності та особливості будуть інкапсульовані в межах самого сервісу відповідної ORM-бібліотеки чи даних його сховища.

Сервіс звертається до конфігураційного файлу зі схемою бази даних з метою отримання масиву таблиць. Процес формування моделей здійснюється ітеративно для кожної таблиці. Він складається з кількох послідовних кроків: спочатку з масиву таблиць вибирають об'єкт таблиці, для якого формується відповідний файл. Як назву файлу використовується шаблон, визначений користувачем на етапі налаштування процесу, в якому ключові слова необхідно замінити відповідними значеннями. Розширення для файлу також вибирають з конфігураційного файлу схеми, зокрема з компоненти «metadata». На основі комбінації назви та розширення створюється файл у заздалегідь створеній теці, до якої буде записуватись код моделі.

Після створення файлу моделі здійснюється генерація коду моделі. Сервіс отримує з конфігураційного файлу налаштування мови програмування, особливості синтаксису та інші правила, які впливають на кінцевий код. Використовуючи цю інформацію, сервіс звертається до бази даних з конфігураційними даними з метою отримання фрагмента коду, який відповідає за базові імпорти сторонніх модулів та бібліотек, необхідних для правильної роботи моделі. Отримавши пот-

рібний фрагмент коду з імпортами, він записує цей код до створеного файлу моделі.

Наступним етапом є запис у файлі наступного основного фрагмента коду – самої структури моделі. Оскільки ORM-бібліотеки побудовані на принципах об'єктно-орієнтованого підходу, модель зазвичай подана як клас або як функції-конструктор. Сервіс звертається до конфігураційної бази даних та отримує «wireframe», тобто так званий каркас майбутньої моделі. Цей каркас використовують для того, щоб можна було без зайвих проблем додавати поля, зв'язки та характеристики моделі. Крім цього, каркас моделі вже містить всі необхідні символи-роздільники та ключові слова, щоб компілятор чи інтерпретатор відповідної мови програмування вважав його валідним. Назвою класу є назва моделі, зазначена в об'єкті таблиці зі схеми бази даних. Цей каркас містить поля-параметри характеристик моделі. Наприклад, це може бути конвертація імені моделі в множинну форму чи можливість автоматичного створення полів для збереження часової мітки створення та останньої зміни запису в таблиці. Всі можливі поля-характеристики моделі, які підтримує бібліотека, зберігаються всередині бази даних, а потенційні значення для них передаються всередині конфігураційного файлу бази даних.

До каркаса моделі також додається програмний опис полів таблиці. Поля, як і таблиці, отримують як масив об'єктів, кожен з об'єктів поля опрацьовується окремо. Шаблон для створення коду поля так само зберігається в базі даних сервісу. Отримавши цей фрагмент коду, сервіс починає заповнювати його даними (назва поля та його характеристики). Крім того, необхідно визначити тип даних поля в моделі, адже кожна ORM-бібліотека містить свої типи даних. З цією метою всі сервіси генерації керуються загальним алгоритмом переведення типу даних в БД до відповідного типу даних моделі, який використовує значення їхнього типу даних, типу зазначених в об'єкті поля та перелік підтримуваних бібліотекою значень. Принцип роботи алгоритму визначення типу даних ORM-бібліотеки наведено на рис. 2. Визначений тип даних розміщується у фрагменті коду поля. Після опрацювання всіх полів таблиці та генерації їхнього коду його записують у каркас моделі на визначене місце.

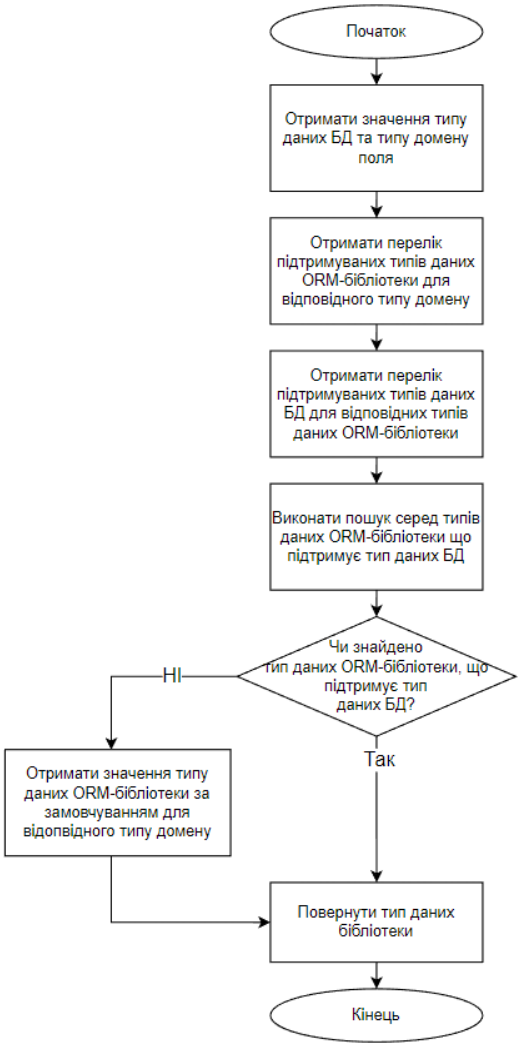

Рис. 2. Алгоритм вибору типу даних ORM-бібліотеки

Наступним етапом є визначення обмежень та індексів таблиці. Їхні дані, як і іншу інформацію для генерації ORM-моделі, отримують з конфігураційного файлу. Після отримання переліку обмежень та індексів для них формується відповідна назва згідно з типом обмеження та шаблоном назви, наведеним у конфігураційному файлі. Після генерації назви сервіс знову звертається до конфігураційного файлу для отримання об'єктів полів. Для цього використовуються ідентифікатори об'єктів полів, які збережені у відповідному масиві. Отримавши всю необхідну інформацію, сервіс звертається до бази даних, отримує шаблон коду, до якого додає необхідну інформацію. Після завершення підготовки коду для всіх обмежень та індексів фрагменти коду об'єднують та записують на визначене місце в каркасі моделі.

Останньою частиною коду, яку необхідно записати до моделі, є співвідношення між таблицями. Сервіс знову звертається до кон-

фігураційного файлу з метою отримання переліку всіх співвідношень між таблицями. Для кожного співвідношення сервіс здійснює пошук двох об'єктів таблиць – дочірньої та батьківської. Пошук здійснюють за допомогою ідентифікаторів об'єктів таблиць, які зберігаються в описі співвідношення. На основі інформації з цих об'єктів таблиць формується сама назва співвідношення, а також визначаються назви моделей. Також визначається ключове слово, яке описує тип зв'язку. Для цього використовується одна з таблиць конфігураційної бази даних, що зберігає співвідношення між типами зв'язків у схемі бази даних та типами, які підтримуються ORM-бібліотекою.

Останнім етапом є визначення поведінки співвідношення в процесі операцій з видалення та зміни записів таблиці. Для цих операцій також існує системна таблиця в конфігураційній базі даних, яка відповідає за трансформацію типу поведінки зі схеми бази даних у тип, що підтримується бібліотекою. Після опрацювання всіх співвідношень з конфігураційного файлу всі фрагменти коду комбінуються та записуються до клас моделі у відповідному виді.

Останнім, але необов'язковим етапом генерації коду є оформлення коду для експорту класу моделі з файлу. Його здійснюють лише для тих мов програмування, для яких потрібно писати певний код для експорту класу задля його подальшого використання. Для цього сервіс звертається до конфігураційного файлу та зчитує параметри експорту, якщо такі підтримуються мовою. На основі параметрів експортування та параметрів самої мови програмування він звертається до бази даних сервісу, щоб отримати необхідний фрагмент коду. До цього фрагмента підставляється назва класу, якщо це необхідно, наприклад у процесі іменованого експорту в JavaScript [17, 18]. В іншому випадку здійснюють неіменований експорт або експорт за замовчуванням.

Після завершення етапу генерації коду здійснюють процес чистки коду та його форматування, тобто видаляють всі текстові ключові мітки, які вказували на місце майбутнього розміщення згенерованих фрагментів коду. Мітки в коді легко знайти та видалити в автоматичному режимі, оскільки вони мають унікальну комбінацію символів і розміщені на окремому рядку. Процес форматування складається із застосування зазначених користувачем параметрів відступів. Під час здійснення генерації код файлу форматується з використанням двох пробільних символів на один відступ. Сервіс звертається до конфігураційного файлу для отримання параметрів форматування, які були зазначені користувачем. Якщо параметри тотожні з тими, що застосовувалися в процесі генерації, то процес форматування не здійснюють. В іншому випадку здійснюють пошук символів відступу та їхню заміну на потрібну послідовність символів. Генерація коду є завершеною після створення файлів моделей для всіх таблиць зі схеми БД.

Висновки

Було визначено підхід до наведення схеми реляційної бази даних у текстовому виді за допомогою конфігураційного файлу формату JSON та особливості цього формату. Наведено перелік компонент конфігураційного файлу та визначено їхній основний зміст.

Обґрунтовано необхідність використання, а також описано структуру конфігураційної бази даних для збереження інформації, потрібної для генерації коду ORM-моделей. Вибрано та обґрунтовано вибір документоорієнтованої бази даних як типу такого сховища.

Література

1. Портер М. Конкурентна стратегія. Техніки аналізу галузей і конкурентів. Київ: Наш Формат, 2020. 424 с.
2. Sequelize. Version 6. Migrations. URL: https://sequelize.org/docs/v6/other-topics/migrations/#creating-the-first-model-and-migration (дата звернення: 18.01.23).
3. Typeorm-model-generator. Homepage. URL: https://github.com/Kononnable/typeorm-model-generator (дата звернення: 18.01.23).
4. Prisma. Documentation. Manage data with Data Browser. URL: ttps://www.prisma.io/docs/data-platform/data-browser (дата звернення: 27.12.22).
5. Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design / Torres A., Galante R., Pimenta M. S, Martins A. J. B. *Information and Software Technology*. 2017. Volume 82. P. 1–18. URL: https://www.sciencedirect.com/science/article/abs/pii/S0950584916301859 (Last accessed: 27.12.22).
6. Hibernate Tools ORM. URL: https://github.com/hibernate/hibernate-tools/tree/main/orm (дата звернення: 27.12.22).
7. The Symfony MakerBundle. Documentation. URL:https://symfony.com/bundles/SymfonyMakerBundle/current/index.html (дата звернення: 27.12.22).
8. Долгополов К. В. Розробка веб-платформи для генерації ORM-моделей на основі схеми реляційної БД. *Концепт науки XXI: стратегії, методи*

та наукові інструменти: *матеріали II Міжнародної студентської наукової конф.,* м. Черкаси, 21 жовт. 2022 р. Вінниця, 2022. С.158–159.
9. Chaudhari A. YAML vs JSON vs XML. What is the Difference Between Them? URL: https://www.csestack.org/yaml-vs-json-vs-xml-difference/ (дата звернення: 27.12.22).
10. Bhalla S. YAML vs JSON vs XML: Which One to Choose? URL: https://javascript.plainenglish.io/yaml-vs-json-vs-xml-what-to-choose-4c7a72417ff4/ (дата звернення: 27.12.22).
11. Harold E. R., Means W. S. XML in a Nutshell. O'Reilly Media, 2004. 714 p.
12. Singh Ch. Advantages and Disadvantages of XML. URL: https://beginnersbook.com/2018/10/advantages-and-disadvantages-of-xml/ (дата звернення: 27.12.22).
13. Smith B. Beginning JSON. Apress Berkeley, CA, 2015. 324 p.
14. Dionysia Lemonaki. What is YAML? The YML file format. URL: https://www.freecodecamp.org/news/what-is-yaml-the-yml-file-format/ (дата звернення: 27.12.22).
15. Leach P., Mealling M., Salz R. A Universally Unique IDentifier (UUID) URN Namespace. The Internet Society, 2005. URL: https://www.ietf.org/rfc/rfc4122.txt (дата звернення: 27.12.22).
16. Fowler M., Sadalage P. J. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional, 2012. 347 p.
17. MDN web docs. JavaScript. Statements and declarations. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export (дата звернення: 27.12.22).
18. Megida D. Module.exports – How to Export in Node.js and JavaScript. URL: https://www.freecodecamp.org/news/module-exports-how-to-export-in-node-js-and-javascript/ (дата звернення: 27.12.22).

References

1. Porter M. Competitive strategy. Techniques of industry and competitor analysis. Kyiv: Nash Format Publ., 2020. 424 p.
2. Sequelize. Version 6. Migrations. Available at: https://sequelize.org/docs/v6/other-topics/migrations/#creating-the-first-model-and-migration (accessed: 18.01.23).
3. Typeorm-model-generator. Homepage. Available at: https://github.com/Kononnable/typeorm-model-generator (accessed:18.01.23).
4. Prisma. Documentation. Manage data with Data Browser. Available at: ttps://www.prisma.io/docs/data-platform/data-browser (accessed: 27.12.22).
5. Twenty years of object-relational mapping: A survey on patterns, solutions, and their implications on application design / Torres A., Galante R., Pimenta M. S, Martins A. J. B. *Information and Software Technology,* 2017. Volume 82. pp. 1–18. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0950584916301859 (accessed: 27.12.22).
6. Hibernate Tools ORM. Available at: https://github.com/hibernate/hibernate-tools/tree/main/orm (accessed: 27.12.22).
7. The Symfony MakerBundle. Documentation. Available at: https://symfony.com/bundles/SymfonyMakerBundle/current/index.html (accessed: 27.12.22).
8. Dolhopolov K. V. Development of a Web Platform for Generating ORM Models Based on a Relational Database Scheme. *Science concept XXI: strategies, methods, and science instruments: materials of the II International Student Scientific Conference*. Vinnytsia, 2022. Pp. 158–159. [in Ukrainian]
9. Chaudhari A. YAML vs JSON vs XML. What is the Difference Between Them? Available at: https://www.csestack.org/yaml-vs-json-vs-xml-difference/ (accessed: 27.12.22).
10. Bhalla S. YAML vs JSON vs XML: Which One to Choose? Available at: https://javascript.plainenglish.io/yaml-vs-json-vs-xml-what-to-choose-4c7a72417ff4/ (accessed: 27.12.22).
11. Harold E.R., Means W. S. XML in a Nutshell. O'Reilly Media Publ., 2004. 714 p.
12. Singh Ch. Advantages and Disadvantages of XML. Available at: https://beginnersbook.com/2018/10/advantages-and-disadvantages-of-xml/ (accessed: 27.12.22).
13. Smith B. Beginning JSON. Apress Berkeley, CA, 2015. 324 p.
14. Dionysia Lemonaki. What is YAML? The YML file format. Available at: https://www.freecodecamp.org/news/what-is-yaml-the-yml-file-format/ (accessed: 27.12.22).
15. Leach P., Mealling M., Salz R. A Universally Unique IDentifier (UUID) URN Namespace. The Internet Society, 2005. Available at: https://www.ietf.org/rfc/rfc4122.txt (accessed: 27.12.22).
16. Fowler M., Sadalage P. J. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional Publ., 2012. 347 p.
17. MDN web docs. JavaScript. Statements and declarations. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/export (accessed: 27.12.22).
18. Megida D. Module.exports – How to Export in Node.js and JavaScript. Available at: https://www.freecodecamp.org/news/module-exports-how-to-export-in-node-js-and-javascript/ (accessed: 27.12.22).

Долгополов Кирило Вікторович, магістрант, кафедри системотехніки,
kyrylo.dolhopolov@nure.ua,
тел. +38 063-541-73-16,

Імангулова Зульфія Аліївна, к.т.н., доц., доцент кафедри системотехніки,
тел. +38 099-777-77-77,
zulfiia.imanhulova@nure.ua,
Харківський національний університет радіоелектроніки, пр. Науки, 14, м. Харків, 61166, Україна.

Method of generating ORM models software code based on relational database schemes

Abstract. Problem. *Databases and data stores are an important part of any system, regardless of the field of application. It is at this level that all system information is stored, such as a list of online store products, user personal data, system configuration parameters, etc. In addition, they allow to structure information in the desired format. Although the use of databases is associated with obvious advantages, it has some disadvantages, in particular the process of designing them and integrating their software code is labor-intensive and time-consuming. This problem is usually attempted to be circumvented by using ORM libraries, which provide a convenient programming interface for performing basic queries. But the process of setting up an ORM library including developing ORM models still takes a lot of time.* ***Goal.*** *The purpose of the work is to develop an algorithm for the automatic generation of software code for ORM models based on the selected ORM library and the data scheme presented in the form of a configuration file. To achieve the goal, it is necessary to establish the optimal representation of the data scheme in text format, to determine the structure of the data configuration scheme and to describe all stages of the generation of software code for ORM models.* ***Methodology.*** *System approach, analysis of existing methods of visual design, formats of text representation of configuration data and methods of configuration data storing.* ***Results.*** *The developed method for software code generation for the selected ORM library based on the scheme of the relational database provides the user with the opportunity to obtain a qualitatively designed software code with minimal time expenditure.* ***Originality.*** *The developed method implements a unique approach which allows to create program code for various ORM libraries, supports the syntax and features of a specific programming language, and provides work with data schemas from various relational databases.* ***Practical value.*** *Developed generation algorithm can be used as part of the software in which the automated process of preparing the data schema in the appropriate text formatted form will be implemented. As a result, such a web-application will represent a platform for working with the data layer for IT professionals.*

Key words: *code generation, configuration file, relational database, data schema, ORM library.*

Dolhopolov Kyrylo, Master's Degree student of the System Engineering Department,
kyrylo.dolhopolov@nure.ua, tel. +38 063-541-73-16,
Imanhulova Zulfiia, Ph.D., Associate Professor
zulfiia.imanhulova@nure.ua, tel. +38 067-999-57-57,
Kharkiv National University of Radio Electronics, Nauky Ave. 14, Kharkiv, 61166, Ukraine.