УДК 004.62 DOI: 10.30748/soi.2024.177.07

Ю.Е. Парфьонов

Харківський національний економічний університет ім. С. Кузнеця, Харків

ПИТАННЯ МІГРАЦІЇ СХЕМИ БАЗИ ДАНИХ ПІД ЧАС СУПРОВОДУ ВЕБ-ЗАСТОСУНКІВ НА БАЗІ ФРЕЙМВОРКУ DJANGO

Статтю присвячено особливостям виконання міграції схеми наявної бази даних у веб-застосунках на базі фреймворку Django. Розглянуто проблеми, пов'язані з розширенням функціональності застосунків під час їх супроводу. Проведено аналіз головних елементів процесу міграції схем баз даних. Досліджено роль Django ORM (Object Relational Mapping) щодо управління міграціями, вплив міграцій на роботу застосунків та практичні аспекти їх застосування. Виконано опис головних міграційних команд та доцільних сценаріїв їх використання. Наведено поради щодо виконання процесу міграції.

***Ключові слова**: міграція схеми бази даних; модель даних; супровід веб-застосунків; Django; ORM; SQL-запит.*

Вступ

Постановка проблеми. Життєвий цикл програмного забезпечення (ПЗ) відбиває його еволюцію від задуму до виведення з експлуатації. Міжнародний стандарт ISO/IEC/IEEE 12207:2017 [1] визначає чотири групи процесів життєвого циклу ПЗ: процеси угоди, процеси організаційного забезпечення проєкту, процеси технічного управління та технічні процеси. Відповідно до [1] є чотирнадцять технічних процесів, які охоплюють різноманітну діяльність від бізнес-аналізу до вилучення ПЗ з експлуатації.

Одним із важливих елементів життєвого циклу ПЗ є його супровід – сукупність дій із забезпечення роботи, внесення змін при виявленні помилок, адаптації ПЗ до нового середовища функціонування, а також підвищення продуктивності або поліпшення деяких характеристик ПЗ [2].

Процес супроводу ПЗ визначається міжнародним стандартом ISO/IEC/IEEE 14764:2022 [3]. Він спрямований на забезпечення стабільної роботи ПЗ, задоволення потреб користувачів та підтримку бізнес-процесів, для яких було розроблено програмний продукт (ПП). Цей процес пов'язаний з відстежуванням здатності ПП надавати послуги, вживанням коригувальних, адаптивних, удосконалювальних та профілактичних заходів.

Оскільки вимоги до ПП досить часто змінюються в процесі його експлуатації, дуже важливими є заходи, спрямовані саме на його вдосконалення.

Аналіз останніх досліджень і публікацій. Удосконалювальний супровід – це процес модифікації елементів програмної системи для покращення її роботи та продуктивності. Він забезпечує сприйнятливість ПП та зручність його використання. Цей процес може містити зміну поточної функціональності ПП шляхом покращення, видалення або додавання нових функціональних можливостей, підвищення його продуктивності, покращення користувацьких інтерфейсів, зручності використання тощо.

Яскравим прикладом такого ПП є "Knowledge Assessment System for World Wide Web" (KnAS4Web) [4], призначений для комп'ютерного оцінювання рівня навчальних досягнень студентів. KnAS4Web активно використовується впродовж останніх п'яті років. Протягом цього часу цей застосунок зазнав декілька змін для покращення його функціональності та задоволення вимог користувачів.

Під час супроводу сучасних веб-застосунків на базі серверного фреймворку Django [5], таких як KnAS4Web, також виникає проблема міграції схем наявних баз даних.

Відомо, що міграція схеми бази даних – це процес зміни її структури для адаптації до нових вимог, зокрема, – до вдосконалення програмного забезпечення. Він може містити додавання нових таблиць, зміну атрибутів або типів даних, видалення непотрібних елементів та інші модифікації, що впливають на структуру бази даних [6].

Схеми баз даних доводиться модифікувати в разі появи нових функціональних або нефункціональних вимог, рефакторингу бази даних тощо [7]. Коли застосовуються зміни схеми бази даних, зазвичай, необхідна сумісна еволюція даних і коду [8].

Подальше вдосконалення подібних ПП, підвищення вимог до їх безпеки та швидкодії неминуче ставить питання про ефективність процесу міграції схеми баз даних. Це актуально як для професійних веб-розробників, так і для початківців, які вивчають розробку веб-застосунків на базі Django.

Метою статті є аналіз головних елементів процесу міграції схем баз даних у веб-застосунках, побудованих на основі фреймворку Django. Об'єктом дослідження є сам процес міграції, його інструменти, вплив на роботу застосунків та практичні аспекти виконання цього процесу.

Виклад основного матеріалу

Фреймворк Django є одним з популярних інструментів для швидкого розроблення серверної частини веб-застосунків мовою програмування Python. Він надає повний стек інструментів для створення веб-застосунків, зокрема вбудований ORM-фреймворк.

Django ORM надає розробникам такі переваги:

1. Спрощення роботи з базою даних.

Він дозволяє взаємодіяти з базою даних з використанням об'єктно-орієнтованого підходу, що робить цей процес інтуїтивно зрозумілим. Замість написання складних SQL-запитів, розробник може працювати з моделями даних Python, що підвищує його продуктивність.

2. Уникнення повторення коду.

Django ORM дозволяє використовувати одну і ту ж модель даних для взаємодії з різними СУБД, що робить програмний код більш уніфікованим.

3. Захист від SQL-ін'єкцій.

SQL-ін'єкція – це один з найбільш поширених видів атак на веб-сервери. Її головна ідея полягає в додаванні шкідливого коду під час виконання SQL-запиту [9]. Django дозволяє значно знизити імовірність виникнення вразливостей, пов'язаних із SQL-ін'єкціями, оскільки цей фреймворк автоматично генерує параметризовані SQL-запити.

4. Підтримка міграцій схеми бази даних.

Django ORM допомагає автоматизувати процес міграції схеми бази даних, забезпечуючи створення міграційних скриптів та їх автоматичне застосування до бази даних.

У Django кожна модель даних, визначена класом Python, відображається на таблицю бази даних. Атрибути цього класу відповідають полям таблиці, а методи – операціям з даними. ORM у Django також забезпечує можливість використання відношень між моделями, таких як "Один до одного", "Один до багатьох", "Багато до багатьох" тощо.

Під час змін в моделях даних, таких як додавання нових полів або зміна типів вже наявних, Django ORM допомагає генерувати міграційні скрипти відповідно до змін у моделях, а також виконувати ці міграції та оновлювати схему бази даних.

Отже, ORM відіграє ключову роль у процесі міграції схеми бази даних у Django-застосунках. Він автоматизує створення міграційних скриптів, забезпечує узгодженість між моделями даних та схемою бази даних, спрощує виконання змін в схемі бази даних.

Django має механізми управління міграціями, що дозволяють створювати, застосовувати та скасовувати міграції, а також відстежувати їх стан.

Міграція схеми бази даних у фреймворку Django передбачає виконання певних команд для управління цим процесом [10; 11].

Першим кроком є створення міграційних скриптів, які описують майбутні зміни в структурі бази даних. Вони містять Python-код, що визначає необхідні зміни, такі як створення нових таблиць, додавання або зміна полів, видалення таблиць тощо. Ці скрипти генеруються автоматично на основі змін у моделях даних. Для створення міграційних файлів використовується команда *makemigrations*.

Після цього міграційні скрипти необхідно застосувати до бази даних за допомогою команди *migrate*. Вона також дозволяє визначити конкретну версію міграцій, які мають бути застосовані до бази даних. Це дозволяє контролювати процес міграції та забезпечувати узгодженість бази даних.

Django також дозволяє перевіряти, які міграції були застосовані до бази даних, а які ще ні (команда *showmigrations*), що допомагає управляти процесом розробки та підтримки веб-застосунків.

В цілому, процес міграції схеми бази даних у фреймворку Django є автоматизованим та досить ефективним, забезпечуючи зручний механізм для управління змінами в структурі бази даних. Але, під час супроводу веб-застосунків можуть виникати різноманітні проблеми щодо виконання міграції схеми наявної бази даних, які можуть ускладнити цей процес.

Так, існує ризик втрати даних, особливо якщо міграції виконуються не коректно. Наприклад, це може трапитися через неправильно сконструйовані міграційні файли або некоректно налаштовані команди міграції. В найгіршому випадку втрата певних даних може призвести до неможливості використання застосунку за призначенням.

Також, потенційні конфлікти між різними версіями міграційних файлів можуть спричинити некоректне застосування міграцій та, як наслідок, – неочікуваний стан бази даних і помилки у функціонуванні веб-застосунку.

Однією з проблем є накопичення в проєкті великої кількості міграційних файлів, що ускладнює розуміння структури бази даних та управління процесом міграції. Через це може спостерігатися зниження продуктивності механізму міграцій, підвищення ймовірності конфліктів між ними, складнощі з тестуванням і розгортанням веб-застосунку.

В деяких випадках може знадобитися "відкотити" зміни, зроблені під час міграції. Це може бути складним процесом, особливо якщо до цього відбулося видалення даних або зміна структури бази даних.

Таким чином, аналіз ризиків та можливих наслідків помилок під час міграції є важливою складовою процесу розроблення та супроводу веб-застосунків на базі фреймворку Django.

Існують кращі практики, яких доцільно дотри-

муватися під час розроблення та виконання міграцій схеми наявної бази даних [10–13].

Зокрема, до початку міграції важливо провести аналіз потреб та зрозуміти, які саме зміни необхідно внести до схеми бази даних. Це допоможе зберегти час та ресурси для розроблення міграційних файлів, забезпечити ефективність та безпеку процесу міграції. Також важливо зробити резервну копію бази даних, що дозволить відновити попередній стан бази даних у випадку виникнення проблем під час міграції.

Для ефективного керування версіями міграційних файлів, вирішення конфліктів та відстеження змін рекомендується використовувати одну із систем контролю версій, таких як Git.

Важливо здійснювати моніторинг процесу міграцій та своєчасно реагувати на будь-які помилки або проблеми, які можуть виникнути під час цього процесу.

Необхідно документувати процес міграції та зміни, внесені до схеми бази даних. Ця документація допоможе розробникам краще розуміти поточний стан схеми та вирішувати можливі проблеми в майбутньому.

У випадку необхідності суттєвої модифікації схеми бази даних рекомендується розбити процес міграції на кілька етапів та поступово впроваджувати зміни. Завдяки цьому можна зменшити ризик виникнення помилок та спростити їх виявлення.

Також для більш швидкого та ефективного реагування на будь-які непередбачені ситуації під час міграції добре мати резервний план дій.

Дотримання цих рекомендацій дозволить забезпечити успішне та безпечне впровадження міграцій схеми наявної бази даних у застосунках на базі фреймворку Django.

Далі розглянемо питання управління міграціями схеми бази даних на прикладі супроводу ПП KnAS4Web [4].

KnAS4Web використовується в навчальному процесі з 2020 року. На теперішній час цей вебзастосунок має декілька версій, а в його програмний код внесено більше 95 змін. Кількість змін за роками наведено на рис. 1.

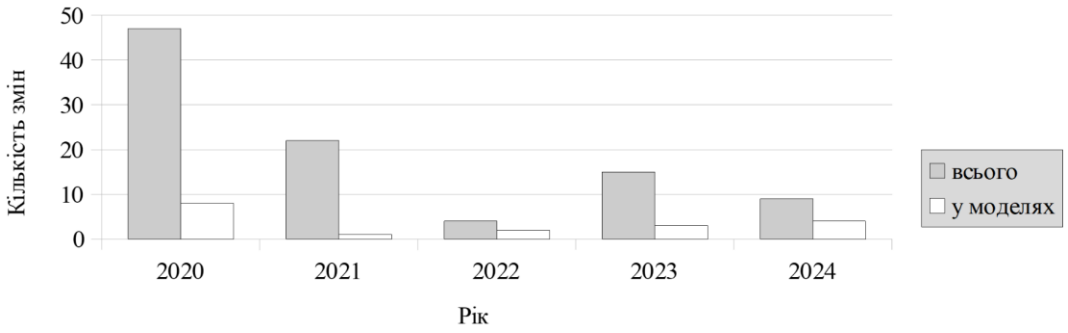

Рис. 1. Зміни в програмному коді KnAS4Web
Джерело: розроблено автором.

Біля 19 % з них було певною мірою пов’язано з необхідністю міграції схеми наявної бази даних. Протягом останніх п’яти років такі зміни відбувалися щорічно, а їхня кількість та складність залежала від різних факторів, зокрема – від планованого розширення функціональності застосунку.

Крім того, за цей час в базі даних накопичився досить великий об’єм критично важливих даних. Їх втрата може спричинити тривале припинення використання KnAS4Web за призначенням.

Таким чином, супровід ПП, подібних KnAS4Web, вимагає періодичної міграції схеми бази даних. Цей процес ускладнюється необхідністю збереження наявних даних. Тому вкрай важливо застосовувати кращі практики щодо виконання міграції схеми бази даних, зважаючи на особливості KnAS4Web.

Зазвичай, механізм міграцій Django працює коректно та автоматично забезпечує виконання необхідних змін у структурі бази даних. Але нерідко виникають проблеми, які потребують втручання в процес міграції.

Розглянемо деякі практичні ситуації.

По-перше, доцільно запобігати накопиченню в проєкті великої кількості міграційних файлів. Для цього рекомендується періодично проводити очищення історії міграцій. Цей процес зазвичай складається з таких етапів [12]:

1. Забезпечення відповідності наявних моделей поточній схемі бази даних.

Це можна перевірити шляхом виконання команди *makemigrations*. Тут може бути корисним використання параметра *--dry-run*, який показує, які міграції буде виконано, не створюючи жодного міграційного файлу.

Якщо виявиться, що є заплановані міграції, треба спочатку виконати їх (команда *migrate*).

2. Очищення історії міграції для кожного засто-

сунку проєкту.

Виконується командою *migrate* з параметром *--fake* із застосуванням імені *zero*. Наприклад, для застосунку *myapp* це виглядає так:

migrate --fake myapp zero.

Тут треба бути дуже обережним, щоб не забути вказати параметр *--fake* та ім'я цільового застосунку. В протилежному випадку може бути очищена історія міграцій усіх застосунків, зокрема системних. Тоді для відновлення коректності виконання міграцій знадобиться редагування схеми бази даних "вручну", що несе додаткові ризики. На цьому етапі також доцільно застосовувати команду *showmigrations*, яка показує всі міграції з позначенням їх поточного стану.

3. Видалення наявних міграційних файлів.

Необхідно видалити всі файли, за винятком *__init__.py*, з каталогів *migrations* застосунків проєкту. Для цього можна застосувати команди цільової операційної системи з консолі або використати певний файловий менеджер.

4. Створення початкових міграцій (команда *makemigrations*).

5. Імітація застосування початкової міграції.

Мета цього етапу – змінити стан початкової міграції на "застосовано". Жодних змін в базі даних не відбувається, оскільки вони вже були зроблені раніше. Виконується командою *migrate* з параметром *--fake-initial*.

Після застосування всіх наведених етапів у проєкті створюється тільки один новий міграційний файл, який об'єднує всі попередні міграції. Змін у наявних даних не відбулося, база даних знаходиться в узгодженому стані, веб-застосунок працює коректно. Загалом це дозволяє зменшити розмір кодової бази, зробити історію змін більш читабельною та зрозумілою, спростити процес знаходження та виправлення помилок у міграціях, а також покращити швидкість їх виконання.

Іноді може знадобитися внести зміни в схему бази даних шляхом безпосереднього виконання SQL-запитів. Це може стати ще однією проблемою, оскільки в такому випадку маємо ситуацію неузгодженості стану бази даних та механізму міграцій Django.

Для відновлення працездатності веб-застосунку, скоріше за все, прийдеться виконати певні міграції "вручну", використовуючи мову SQL. Рекомендується дотримуватися послідовності дій [13], яка перевірена на практиці під час супроводу веб-застосунку KnAS4Web:

1. Вивести на консоль SQL-команди для певної міграції шляхом виконання команди *sqlmigrate*:

sqlmigrate myapp 0002,

де *myapp* – ім'я цільового застосунку в проєкті;

0002 – унікальний префікс імені міграційного файлу, такого як *0002_remove_item.py*.

Результат виведення даних щодо кожної операції міграції складається з коментаря, що описує її призначення, після якого відображається відповідна послідовність операторів SQL. Для виконання команди *sqlmigrate* потрібне активне з'єднання з базою даних. Рекомендується, щоб це була саме та база даних, до якої планується застосувати певну міграцію.

2. Послідовне виконання всіх SQL-команд. Спочатку обов'язково треба порівняти схему наявної бази даних із змінами, що плануються в кожній міграційній операції. Можливо, певні елементи схеми вже знаходяться в необхідному стані. В такому разі деякі окремі SQL-команди або їх послідовності треба пропустити.

3. Імітація застосування міграції. Потрібно виконати фейкову міграцію, оскільки це дозволить фреймворку Django "вважати", що зміни вже були застосовані до бази даних, навіть якщо насправді вони були внесені вручну. Це допоможе уникнути конфліктів під час застосування нових міграцій, які можуть виникнути через вже існуючі зміни в схемі бази даних. Виконується за допомогою команди *migrate* аналогічно зазначеному вище випадку імітації початкової міграції. Замість параметру *--fake-initial* треба використовувати *--fake*. Наприклад, для застосунку *myapp* та міграції *0002_remove_item.py* це виглядає так:

migrate --fake myapp 0002.

Розглянутий процес з мінімальними змінами можна застосувати і в зворотному порядку, тобто для скасування міграцій "уручну". Зокрема, знадобиться виконати команду *sqlmigrate* з параметром *--backwards*.

Висновки

Таким чином, дуже важливим є удосконалювальний супровід ПП, зокрема, тих, що базуються на серверному веб-фреймворку Django. Часто він пов'язаний з необхідністю міграції схеми наявної бази даних. Цей процес є критично важливим елементом супроводу Django-застосунків, що вимагає ретельного планування та тестування.

Django має потужні інструменти для ефективного управління міграцією схем баз даних. Але під час міграції схеми наявної бази даних можуть виникати проблеми, такі як втрата даних, зниження продуктивності тощо.

Застосування кращих практик, зокрема, версіонування схеми та регулярне резервне копіювання допоможе уникнути багатьох з них.

В цілому, запорукою успішної міграції схеми бази даних під час супроводу веб-застосунків на базі веб-фреймворку Django є комплексний підхід, а також адаптація до специфіки конкретного проєкту.

Список літератури

1. ISO/IEC/IEEE 12207:2017. Systems and software engineering – Software life cycle processes. [Active November, 15, 2017]. Geneva : ISO, 145 p. https://doi.org/10.1109/IEEESTD.2017.8100771.

2. Величко О. М., Грабовський О. В., Гордієнко Т. Б. Особливості процесів життєвого циклу програмного забезпечення для засобів вимірювальної техніки. *Збірник наукових праць Одеської державної академії технічного регулювання та якості*. 2020. № 2(17). С. 37–45. https://doi.org/10.32684/2412-5288-2020-2-17-37-45.

3. ISO/IEC/IEEE 14764:2022. Software engineering – Software life cycle processes – Maintenance. [Active January, 21, 2022]. Geneva : ISO, 46 p. https://doi.org/10.1109/IEEESTD.2022.9690131.

4. Парфьонов Ю. Е. Комп'ютерна програма "Knowledge Assessment System for c Wide Web (KnAS4Web)". Свідоцтво про реєстрацію авторського права на твір від 24.02.2020 № 96294. *Авторське право і суміжні права* : офіційний бюлетень. Київ : ДП "Український інститут інтелектуальної власності", 2020. Бюл. № 57. С. 494. URL: https://ukrpatent.org/atachs/BULETEN_Avt_Pravo_%E2%84%96%2057-2020.pdf (дата звернення: 07.05.2024).

5. Django Software Foundation. *Django* : web site. URL: https://www.djangoproject.com/foundation (accessed 07.05.2024).

6. Database Schema Migration (Schema Change). *Liquibase* : web site. URL: https://www.liquibase.com/resources/guides/database-schema-migration (accessed 07.05.2024).

7. Chillón A. H., Klettke M., Ruiz D. S., Molina J. G. A Generic Schema Evolution Approach for NoSQL and Relational Databases. *IEEE Transactions on Knowledge and Data Engineering*. 2024. Vol. 36. P. 2774–2789. https://doi.org/10.1109/TKDE.2024.3362273.

8. Chillón A. H., Molina J. G., Hoyos J. R., Ortín-Ibáñez M. J. Propagating Schema Changes to Code: An Approach Based on a Unified Data Model. *Workshop Proceedings of the EDBT/ICDT 2023 Joint Conference*. Ioannina, Greece, 28-31 March. 2023. URL: https://ceur-ws.org/Vol-3379/CoMoNoS_2023_id251_Alberto_Hernandez_Chillon.pdf (accessed 07.05.2024).

9. Duisebekova K., Khabirov R., Zholzhan A. Django as Secure Web-Framework in Practice. *The Bulletin of Kazakh Academy of Transport and Communications named after M. Tynyshpayev.* 2021. Vol. 116. No. 1. P. 275–281. https://doi.org/10.52167/1609-1817-2021-116-1-275-281.

10. Documentation: Migrations. *Django* : web site. URL: https://docs.djangoproject.com/en/5.0/topics/migrations/ (accessed 07.05.2024).

11. Як зробити міграцію Django – поради експерта NIX. *Nix Solutions* : web site. URL: https://www.nixsolutions.com/ua/blog/it-novosti/yak-zrobiti-migraciyu-django-poradi-eksperta-nix/ (accessed 07.05.2024).

12. Freitas V. How to Reset Migrations. *Simple Complex* : web site. URL: https://simpleisbetterthancomplex.com/tutorial/2016/07/26/how-to-reset-migrations.html (accessed 07.05.2024).

13. Johnson A. Django: Run a migration "by hand". *Adam Johnson* : web site. URL: https://adamj.eu/tech/2022/06/29/run-a-django-migration-by-hand/ (accessed 07.05.2024).

Надійшла до редколегії 07.05.2024
Схвалена до друку 28.08.2024

Відомості про автора:

Парфьонов Юрій Едуардович
кандидат технічних наук старший науковий співробітник
доцент
Харківського національного економічного
університету ім. С. Кузнеця,
Харків, Україна
https://orcid.org/0000-0003-3943-3201

Information about the author:

Yurii Parfonov
PhD in Engineering Senior Researcher
Associate Professor
of Simon Kuznets Kharkiv
National University of Economics,
Kharkiv, Ukraine
https://orcid.org/0000-0003-3943-3201

ISSUES OF DATABASE SCHEMA MIGRATION WHEN MAINTAINING DJANGO-BASED WEB APPLICATIONS

Yu. Parfonov

The process of software maintenance is aimed at ensuring its stable operation, meeting the needs of users, and supporting the business processes for which the software product was developed. It is associated with monitoring the software's ability to provide services, taking corrective, adaptive, additive, and preventive actions. When maintaining modern web applications based on the Django server framework, the problem of migrating existing database schemas also arises. The paper provides an examination of database schema migrations within web applications using the Django framework. It navigates through the complexities of extending application functionality during maintenance, highlighting the inherent challenges and strategies for overcoming them. The paper dissects the core components of the database schema migration process, shedding light on its intricacies and potential stumbling blocks. A focus of the paper is to highlight the critical role that Django's ORM plays in orchestrating and managing migrations. The paper emphasizes the need for careful planning and execution to mitigate any negative impact on user experience. Central to the paper's discourse are the primary migration commands essential for executing database schema changes. In addition to theoretical insights, the paper provides practical guidance on how to effectively implement migration processes. Drawing on real-world experience, it examines various implementation strategies, highlighting best practices and common pitfalls to avoid. The paper concludes with valuable recommendations for optimizing the migration process. These recommendations provide pragmatic guidance for developers embarking on migration projects. In summary, this work provides a thorough analysis of the migration process, explores the role of Django ORM, and offers practical implementation insights. It equips developers with the knowledge and tools necessary to overcome migration challenges and drive the evolution of their applications with confidence.

Keywords: *data model; database schema migration; Django; ORM; SQL query; web application maintenance.*