УДК 004.65
Струзік В.А., Грибков С.В., Литвин А.О.
Національний університет харчових технологій

# ДОСЛІДЖЕННЯ МЕТОДІВ І ПІДХОДІВ ПРОВЕДЕННЯ РЕФАКТОРІНГУ БАЗ ДАНИХ

**Струзік В.А., Грибков С. В., Литвин А. О. Дослідження методів і підходів проведення рефакторінгу баз даних.** В роботі проведено дослідження методів і підходів проведення рефакторингу баз даних. Досліджено основні ознаки при яких необхідно проводити рефакторинг баз даних, а також розглянуто основні операції рефакторингу.
**Ключові слова:** рефакторинг, ітераційні процеси, розробка, інформаційні системи.

**Струзик В.А., Грибков С. В., Литвин А. О. Исследование методов и подходов проведение рефакторинга баз данных.** В роботе проведено исследование методов и подходов проведения рефакторингу баз данных. Исследовано основные показатели при которых необходимо проводить рефакторинг баз данных, а также рассмотрено основные операции рефакторингу.
**Ключевые слова:** рефакторинг, итерационные процессы, разработка, информационные системы.

**Struzik V.A., Hrybkov S.V., Lytvyn A.O. Study of the methods and approaches of databases refactoring** In this paper the research of methods and approaches of databases refactoring is carried out. The main features under which it is necessary to refactor databases are examined, as well as the basic operations of refactoring.
**Keywords:** refactoring, databases, information systems.

**Актуальність.** Розвиток інформаційних технологій спонукає до постійного розвитку та удосконалення методологій по створенню інформаційних систем. Постійне удосконалення методологій спрямовано не тільки для покращення показників програмного забезпечення, а й на зменшення часу розробки, і головне – забезпечення якості менеджменту процесу створення програмного забезпечення. Всі сучасні методології розробки програмного забезпечення за своїм характером є еволюційними, тому для них характерне постійний ітеративний процес на якому постійно відбувається накопичення та вдосконалення функціоналу. До прикладів типових процесів відносяться: Rational Unified Process (RUP); Extreme Programming (XP); Scrum; Dynamic System Development Method (DSDM); сімейство методів Crystal; Team Software Process (TSP); Agile Unified Process (AUP); Enterprise Unified Process (EUP); Feature-Driven Development (FDD); Rapid Application Development (RAD). На сьогоднішній день, незалежно від життєвого циклу програмного забезпечення етап розробки програмного забезпечення є ітеративним, що забезпечує: гнучке підпорядкування вимог клієнтів; швидке розширення функціоналу програмного забезпечення; просту взаємодію розробників; зменшення .

Популярність ітеративного підходу розроблення програмного забезпечення обумовлено його ефективністю, а саме: врахування усіх вимог замовника, що формуються послідовно; швидку зміну функціоналу системи; ефективну взаємодію між елементами програмного забезпечення; продуктивну роботу розробників без додаткових витрат ресурсів та з мінімізацією часу на розробку. Але дана методологія потребує високого рівня комунікації між членами команди розробників. Цей процес відрізняється від послідовного підходу, в якому виявляються всі вимоги, що підлягають реалізації – створюється детальний проект, який реалізується, виконується його перевірка і, нарешті, відбувається розгортання готової системи.

Слід зазначити, що більшість компаній на практиці не завжди дотримуються однієї чіткої методології. Але виникають ситуації, коли замовник висуває вимоги до дотримання певної методології, тому для спеціалістів, які працюють в IT сфері необхідно знати основні методології по створенню програмного забезпечення. Все це обумовлено тим, що замовники висувають вимоги по сертифікації відповідно до конкретної методології.

Постійний розвиток інформаційних систем спонукає до постійного саморозвитку. Чітке розуміння та знання методологій дозволяє краще розуміти процес створення програмного забезпечення та організацію на різних стадіях його життєвого циклу. Методології частково перетинаються у різних автоматизованих засобах по проектуванню та створенню програмного забезпечення.

На різних етапах життєвого циклу програмного забезпечення виникає задача рефакторингу його елементів та складових з метою підвищення ефективності та якості функціонування кінцевого

продукту, забезпечення сумісності з рішеннями інших розробників. Окрему нішу займає процес рефакторингу баз даних, що потребує детальне вивчення та класифікацію підходів.

**Аналіз останніх досліджень і публікацій.** Авторами роботи [1] розглядаються стратегії рефакторингу та складності його проведення при створенні інформаційних систем, але більше уваги приділено рефакторингу коду, а не структури бази даних. Робота [2] присвячена еволюційному підходу розробки програмних засобів, а також висвітлено ефективність даного підходу при проведенні рефакторингу. У праці [3] наведено аспекти проведення рефакторингу, але не розглянуто рефакторинг структури баз даних, а тільки окремих елементів. Роботи [4, 5] описують приклади проведення рефакторингу, але більша увага направлена на рефакторинг програмного коду. Автори праці [6] наводять основні алгоритми та підходи проведення рефакторингу усіх елементів системи, описано програмні засоби для підтримки процесу рефакторингу. Більше уваги приділено на оптимізацію процесу рефакторингу потоків даних, а не на структури зберігання даних. Робота [7] більш присвячена рефакторингу баз даних з підтримкою SQL та такі, що мають реляційну структуру. Автори роботи [8] висвітлили проблеми проектування інформаційних систем, а також який вплив має обрання технологій та підходів розробки на проведення рефакторингу.

**Виділення невирішених раніше частин загальної проблеми.** На даний момент підходи рефакторингу програмного коду описані в більшості фундаментальних літературних джерелах, існують багато підходів та автоматизованих програмних засобів для його проведення. Процес рефакторингу програмного коду в теорії не чим не відрізняється від практичного застосування [1]. Але особливу увагу необхідно виділити рефакторингу баз даних, що направлений на підвищення ефективності роботи з нею із забезпеченням збереження її функціональної та інформаційної семантики.

**Формлювання мети дослідження**. Необхідно провести чітку класифікацію та дослідити етапи рефакторингу баз даних, що дасть можливість підвищити ефективність його проведення.

**Виклад основного матеріалу досліджень.** Рефакторингом бази даних називають зміну в схемі базі даних, що сприяє поліпшенню проекту і в той же час забезпечує збереження функціональної та інформаційної семантики бази даних. Іншими словами, проведення операцій рефакторинга не повинно призводити до додавання нових функцій або порушення роботи існуючих, а також не повинно бути направлене на додавання нових даних або зміну існуючих даних. Схема бази даних включає структурні аспекти, такі як визначення таблиць і представлень, а також функціональні аспекти, такі як збережені процедури й тригери.

Але особливу увагу необхідно виділити рефакторингу баз даних, що направлений на підвищення ефективності роботи з нею із забезпеченням збереження її функціональної та інформаційної семантики. Необхідно відмітити, що проведення рефакторінгу бази даних залежить від того, на якій стадії знаходиться програмне забезпечення, адже чим ближче воно до стадії впровадження та експлуатації, тим буде складнішим це здійснити, а також буде збільшуватися витрати часу. Рефакторинг баз даних набагато складніший, ніж рефакторинг програмного коду, тому що база даних містить у собі [1–2]:

- структуровані данні, збережені у відповідності з певною схемою;
- програмні конструкції, такі як збережені процедури та тригери, що забезпечують виконання бізнес-логіки.

Рефакторинг збережених програмних конструкцій бази даних здійснюється за стандартними підходами рефакторингу програмного коду та підлягає автоматизованому тестуванню. Але будь-які зміни у базі даних стосуються не тільки форматів та структур збереження інформації в ній, а впливають на підходи та функціонал програмного забезпечення у цілому. Особливу проблему при рефакторингу бази даних створюють наявні дані.

Основними причинами проведення рефакторингу баз даних є наявність наступних недоліків [2]:

- багатоцільові стовпці, що використовуються в декількох цілях та в різних додатках;
- багатоцільові таблиці, що використовуються для зберігання даних про сутності декількох різних типів;
- надлишковість даних, що є суттєвою проблемою в повсякденній експлуатації бази даних, оскільки при зберіганні одних і тих же даних в декількох місцях виникає ймовірність порушення відповідності та узгодженості;

- таблиці з великою кількістю стовпців, що свідчить про наявність полів, які можуть відноситись до різних сутностей;
- багатозначні стовпці, що вміщують в різних позиціях кілька різних фрагментів інформації;
- багатозначні стовпців, всередині яких в різних позиціях представлено кілька різних фрагментів інформації.

Розглянемо детально основні причини для проведення рефакторингу.

*Багатоцільові стовпці.* Якщо стовпець використовується в декількох цілях, то велика ймовірність, що існує додатковий код, який призначений для забезпечення використання вихідних даних за призначенням, шляхом перевірки значень в одному або декількох інших стовпцях.

*Багатоцільові таблиці* – якщо таблиця використовується для зберігання даних про сутності кількох різних типів, що не можливо допускати для коректної роботи з базою даних. Прикладом може слугувати таблиця Клієнт, яка використовується для зберігання інформації про фізичних осіб та фірми. Недоліком такого підходу є те, що інформації про фізичних осіб і фірми мають різні структури даних, наприклад, для фізичних осіб необхідно зберігати відомості про прізвища, імені та по-батькові, а для фірм вказується юридична назва. За таким випадком неминуча поява значення NULL у певних стовпчиках, що відносяться до одного з типів клієнтів.

*Надлишкові дані.* Наявність надлишкових даних у базі даних, що експлуатується, неминуче призведе до порушення цілісності, тому що не доцільно в різних місцях зберігати однакову інформацію. Наприклад, у багатьох організаціях інформація про клієнтів зберігається декілька разів з різною уточнюючою інформацією. Наприклад клієнт міг змінити адресу проживання, номер мобільного телефону тощо, а в таблиці буде кількість записів, що мають різні значення певних атрибутів.

*Таблиці з великою кількістю стовпців.* Якщо в таблиці є багато стовпців, це можна розглядати як ознаку відсутності неподільності в структурі таблиці. В такій таблиці можуть бути представлені дані, що відносяться до кількох різних сутностей, що потребує проведення нормалізації структури.

*Таблиці з великою кількістю записів.* Наявність великих таблиць слугує ознакою проблеми продуктивності. Наприклад, при пошуку в таблиці, що вміщує понад міліонну записів, не можливо без втрати часу. У зв'язку з цим, доцільно частину записів, що застарілі чи неактуальні, перенести в архів.

*Багатозначні стовпці.* Багатозначними називаються стовпці, що поєднують кілька різних фрагментів інформації. Наприклад, багатозначним стовпцем є ідентифікатор клієнта, в якому перші чотири цифри ідентифікатору клієнта позначають головне відділення компанії цього клієнта, оскільки для виявлення додаткової інформації доводиться виконувати синтаксичний аналіз значень з цього стовпчика. На практиці, такі стовпці розбиваються на окремі поля з даними, щоб можна було простіше проводити обробку цих полів у вигляді окремих елементів.

*Наявність нереалізованих змін.* Якщо зміни в схемі бази даних давно не проводилися, з тієї причини, що можуть виникнути якісь порушення в роботі. Подібні побоювання перед можливими порушеннями в роботі явно свідчать про постійне зростання ризику повної відмови системи, а така ситуація з часом тільки погіршується.

При проведенні рефакторингу бази даних не повинно відбуватися додавання нових функціональних можливостей або порушуватися робота існуючих, а також не повинні додаватися нові дані або змінюватися існуючі, але при цьому можливі зміни як у схемі бази даних так і в модулях забезпечення бізнес-логіки обробки інформації, тому що вони прив'язані до схеми [1].

Також необхідно відмітити, що іноді необхідно провести достатньо глибокий рефакторинг, що буде стосуватися дуже суттєвих змін з метою оптимізації та вдосконалення, при збережені функціональності програмного забезпечення.

При проведені рефакторингу баз даних доцільно виділити 6 основних операцій, а саме:

- рефакторинг структури БД;
- рефакторинг якості даних;
- рефакторинг зв'язків та цілісності;
- рефакторинг архітектури;
- рефакторинг методів;
- зміни, що не входять до операцій рефакторингу.

Доцільно розглянути їх детальніше.

Операції рефакторингу структури направлені на зміну структур однієї чи декількох таблиць або представлень. Прикладом такої операції може бути переміщення стовпця з однієї таблиці в іншу або розбиття багатоцільового стовпчика на кілька окремих стовпців, кожен з яких виконує окреме призначення.

Операції рефакторингу направлені на якість даних в першу чергу повинні забезпечити ефективність збереження та роботи з такими даними, що досягається за рахунок додавання правил валідації, форматів введення даних, забезпечення неможливості залишати порожніми поля та використання значень за замовчуванням та ін.

Проведення операції рефакторингу направлених на зв'язки та цілісність даних забезпечують зв'язаність даних, що знаходяться в різних таблицях зв'язаних різними видами зв'язків. При таких операціях доцільним є створення тригерів для: реалізації каскадного типу зв'язку; ведення історії проведених операцій; збереження видалених чи змінених записів у додаткових таблицях та ін.

Операції рефакторингу архітектури направлені на розбиття складних операцій бізнес-логіки, що реалізовані у клієнтських додатках, на простіші та реалізації їх у збережених процедурах та функціях, що дозволяє уніфікувати подальше їх використання у різних програмних модулях та додатках.

Операції рефакторингу методів полягає у модифікації збережених процедур, функцій та тригерів направлених на оптимізацію та покращення якості їх виконання. При виконанні даних операцій діють правила рефакторингу програмного коду, але необхідно враховувати усі наслідки таких змін. Найпростішими прикладами є зміна назв збережених процедури з метою спрощення розуміння її призначення, а більш складним прикладом може бути заміна алгоритму обробки інформації.

Остання група операцій перетворення, що не входять до операцій рефакторингу, направлена на змін її семантики, адже вони можуть бути виправдані тільки в тому разі, якщо узгоджуються із замовником програмного забезпечення.

Одним з підходів уникнення проблем рефакторингу є використання тестів. Фактично доцільно виділити напрямки написання тестів:

- перевірка схеми бази даних;
- перевірка спосіб використання структур бази даних у програмних додатках;
- контроль введених даних;
- перевірка процедур та функцій, що зберігаються, на повну реалізацію покладеної на них бізнес логіки.

Для багатьох фахівців підхід, який передбачає тестування бази даних, є незвичним, тому їм доводиться стикатися з певними проблемами підчас проведення рефакторингу бази даних. Одніє з проблем є не відповідність персоналу з методикою тестування баз даних. Другою основною проблемою є недостатній асортимент інструментальних засобів для тестування бази даних. Такий інструментарій надається розробниками СУБД, але не завжди є можливість його придбати. Але є програмні засоби з відкритим кодом (Open Source Software - OSS), а саме:

- DBUnit (dbunit.sourceforge.net), що дозволяє управляти тестовими даними;
- SQLUnit, призначеного для тестування збережених процедур.

При виконанні рефакторингу бази даних не можливо обійтись без використання контролю версій. При кожній ітерації змін необхідне проведення контролю версій, адже це забезпечить відновлення втрачених файлів, повернення бази до робочої версії, надає можливість порівняння версій. Контроль версій існує двох типів – централізована та розподілена.

Традиційні системи контролю версій використовують централізовану модель, де всі функції контролю версій виконуються на загальному сервері. Якщо два розробника спробують змінити один і той же файл одночасно, без будь-якого методу управління доступом розробники можуть переписати роботу один одного. Централізовані системи контролю версій вирішують цю проблему за допомогою одного з двох варіантів «моделей управління версіями»: блокування файлів або злиття версій.

Розподілені системи контролю версій (DRCS) використовують одноранговий підхід, а не клієнт-серверний підхід централізованих систем. Вони, в загальному випадку, не потребують централізованому сховищі: вся історія зміни документів зберігається на кожному комп'ютері, в локальному сховищі, і при необхідності окремі фрагменти історії локального сховища

синхронізуються з аналогічним сховищем на іншому комп'ютері. У деяких таких системах локальне сховище розташовується безпосередньо в каталогах робочої копії. Розподілений контроль версій здійснює синхронізацію шляхом обміну патчами (змінами) від тимчасових вузлів.

Ітерація рефакторингу повинна закінчувати контролем версій, що забезпечить відновлення втрачених файлів, повернення бази до робочої версії, надає можливість порівняння версій.

Використання рефакторингу забезпечує «чистоту», тобто ненадлишковість, не тільки коду, а й структури, архітектури, даних та потоків даних програмного проекту. Для проведення успішного рефакторингу баз даних необхідно дотримуватися чотирьох, правил, а саме:

- створювати повну копію бази даних та всього проекту;
- використовувати засоби автоматизованого контролю версій;
- будь-які зміни вносити у початковий код по створенню БД;
- використовувати засоби автоматизованого тестування.

Тобто рефакторинг тісно зв'язаний методами еволюційної методології, такими як – тестування та версіонування.

**Висновки.** Розглянуті підходи для рефакторингу баз даних застосовуються при використанні реляційних та об'єктно-орієнтованих систем управляння баз даних. Їх обрання залежить від функцій програного забезпечення та пріоритетів, що висуває команда розробників.

Для спрощення процесу рефакторингу на етапі обрання СУБД доцільно обирати об'єктно-орієнтовані СУБД, адже вони підтримують еволюційний підхід для створення програмного забезпечення, а от реляційні СУБД орієнтуються на послідовний підхід розробки.

1. Sadalage P. J., Ambler S. W. Refactoring databases: evolutionary database design [Text] (Addison-Wesley Signature Series) – Addison-Wesley Professional, 2006. – 384 p.
2. Fowler M., Sadalage P. J Evolutionary Database Design [Electronic resource] martinfowler.com, May 2016: Proceedings. – Mode of access: http://martinfowler.com/articles/evodb.html - Last access: 2017. – Title from the screen.
3. Fowler M. Refactoring [Electronic resource] Refactoring.com, May 2017: Proceedings. – Mode of access: https://www.refactoring.com/ - Last access: 2017. – Title from the screen.
4. Robert C. Martin The Clean Coder: A Code of Conduct for Professional Programmers, 1st ed.; Prentice Hall, 2011. – 256p. – ISBN-13: 978-0137081073
5. Robert C. Martin Clean Architecture: A Craftsman's Guide to Software Structure and Design, 1st ed.; Prentice Hall, 2017. – 432p. – ISBN-13: 978-0134494166
6. Амблер С., Рефакторинг баз даннях. Эволюционное проектирование / С. Амблер, П. Дж. Садаладж; пер. с англ. К. Птицын. – М.: Вильямс, 2016. –368с. – ISBN-13: 978-5845911575
7. Лерми П. Рефакторинг SQL - приложений / П. Лерми, С. Фаро; пер. с англ. Ф. Гороховский – М.: Символ, 2009. – 336с. – ISBN-13: 978-5932861455
8. Макконнелл С. Совершенный код. Мастер-класс – М.: Русская Редакция, Microsoft Press, 2017. – 896 с. – ISBN-13: 978-5750200641
9. Gary McLean Hall Adaptive Code: Agile coding with design patterns and SOLID principles, 2nd ed.; Microsoft Press, 2017. – 448p. – ISBN-13: 978-1509302581
10. Месарош Д. Шаблоны тестирования xUnit. Рефакторинг кода тестов – М.: Вильямс, 2009. – 832с. – ISBN-13: 978-5845914484
11. Дэйт К. Дж. Введение в системы баз данных – М.: Вильямс, 2017. – 1328с. – ISBN-13: 978-5845907882
12. Тарасов С. СУБД для программиста. Базы данных изнутри – М.: Соломон, 2015. – 320с. – ISBN-13: 978-2746673830
13. Schwartz B., Zaitsev P., Tkachenko V. High Performance MySQL: Optimization, Backups, and Replication, 3rd ed.; O'Reilly Media, 2012. – 826p. – ISBN-13: 978-1449314286