УДК 004.416.2:004.415
DOI https://doi.org/10.32838/2663-5941/2020.2-1/30

Струзік В.А.
«Інженер-програміст ТОВ «ПЕРША УКРАЇНСЬКА ЛІЗИНГОВА КОМПАНІЯ»»

Грибков С.В.
Національний університет харчових технологій

Чобану В.В.
Національний університет харчових технологій

СПЕЦИФІКАЦІЯ СЕМАНТИЧНОГО ВЕРСІОНУВАННЯ БАЗИ ДАНИХ

У статті проаналізовано необхідність використання версіонування під час розробки та підтримки програмного забезпечення. Описані наявні схеми версіонування та особливості кожної з них. Окремо виділено семантичне версіонування, тому що ця схема є найпопулярнішою серед компаній, що займаються розробкою програмного забезпечення. У статті розглянуто основні правила зміни версії програмного продукту відповідно до специфікації семантичного версіонування.

У статті проведено класифікацію баз даних за способом доступу до даних, а саме: файл-серверні, клієнт-серверні та вбудовані. Так само як програмний код, база даних в процесі розробки та підтримки зазнає змін. Для можливості контролювати ці зміни важливим є впровадження версіонування бази даних. Проте сьогодні не існує загальновідомої специфікації версіонування баз даних, зазвичай правила версіонування встановлюються на рівні компанії або команди розробників у вигляді маніфесту. Тому автори статті запропонували специфікацію версіонування баз даних, побудовану на базі семантичного версіонування. У статті наведені основні рекомендації щодо зміни версії бази даних відповідно до специфікації семантичного версіонування баз даних, а також показано вплив тих чи інших змін елементів бази даних на стан версії.

Детально проаналізовано наявні схеми версіонування, а саме: послідовність номеру, десятковий дріб, послідовність чисел, буква як молодша версія, вказівка стадії розробки, алфавітно-цифрова назва, дата, внутрішні версії. Також проаналізовано доцільність використання схеми версіонування та наведено інформацію щодо проблем, з якими може стикнутись команда розробників у разі відмови від використання версіонування. Детально описана найпопулярніша специфікація версіонування програмного коду – специфікація семантичного версіонування, та подано її основні положення.

Ключові слова: *версіонування, база даних, специфікація семантичного версіонування, семантичне версіонування бази даних, система управління базою даних.*

Постановка проблеми. Під час розробки програмне забезпечення постійно еволюціонує, адже додаються нові функції та виправляються знайдені помилки. Щоб полегшити оновлення програмного продукту всі зміни найчастіше об'єднують та випускають єдиним пакетом. Проте сьогодні практикується впровадження декількох релізів одного й того самого програмного забезпечення, тому важливо розуміти, яка функціональність та які помилки присутні в кожному конкретному випуску. Щоб відрізняти ці випуски, команди розробників використовують схеми версіонування для їх маркування. Варто зазначити, що важливим є версіонування не лише програмного коду, а й версіонування схеми бази даних. Найпопулярнішою схемою версіонування програмного коду сьогодні є семантичне версіонування, проте не існує специфікації версіонування баз даних, у зв'язку з чим кожній компанії, що займається розробкою програмного забезпечення, доводиться вводити власний регламент версіонування баз даних.

Аналіз останніх досліджень і публікацій. Автором роботи [1] створено специфікацію семантичного версіонування, яка описує набір правил та вимог, що визначають те, як саме призначається та збільшується значення версії програмного продукту. Автором зазначено, що відповідно до специфікації семантичного версіонування, номери версій і спосіб їх зміни передають інформацію про вихідний код і про те, що змінено від попередньої до нової версії. Проте інформація щодо версіонування баз даних у даній специфікації відсутня.

Постановка завдання. Основною ціллю статті є створення специфікації «семантичне

версіонування бази даних» для ефективного контролю зміни схеми бази даних.

Виклад основного матеріалу дослідження.

Необхідність використання версіонування

Версіонування є невіддільним інструментом протягом всього процесу розробки програмного забезпечення. Всі сучасні компанії, які займаються розробкою програмного забезпечення, впроваджують схеми версіонування, які націлені не лише на визначення та організацію роботи команди розробників, а й на встановлення взаємозв'язку між компанією та користувачами. Шляхом використання маркування версії програмного продукту розробники можуть легко донести до користувачів, чого саме очікувати в будь-якому конкретному випуску. Правильно обрана схема версіонування повинна надати користувачам можливість зрозуміти, які функції та помилки присутні в тому випуску, який використовується. Коли користувач стикається з певною помилкою у відповідній версії програмного продукту, він може:

− оновити програмний продукт до новішої версії, в якій ця помилка виправлена;

− у разі відсутності виправлень, звернутися до компанії розробника з чітким формулюванням помилки та дочекатися нової версії.

У багатьох різних схемах версіонування є свої особливості, але найчастіше основною метою є надання інформації про різницю між двома різними версіями.

Є багато підходів до версіонування: послідовність номеру, десятковий дріб, послідовність чисел, буква в ролі молодшої версії, вказівка стадії розробки, алфавітно-цифрова назва, дата, внутрішні версії тощо.

Використання схеми «послідовність номеру» передбачає нумерацію програмних додатків числами 1, 2, 3 тощо. Послідовність цих номерів можуть відповідати якому-небудь технічному лічильнику, наприклад, номеру версії в системі управління версіями (англ. Version Control System, VCS або Revision Control System). На цей час таким чином позначають лише ті програмні продукти, що оновлюються достатньо рідко, проте стабільно. В таких програмах зазвичай невеликі технічні зміни не описуються наявно, лише під час перегляду через, наприклад, меню «Про програму». У разі випуску версії з новою функціональністю, що недостатньо значна для нової версії, можуть використовувати позначення десятковим дробом.

Першим способом нумерації версій, що розділяв невеликі та серйозні зміни, стала схема «десятковий дріб». Номер версії в такому разі – це десятковий дріб в американському форматі, тобто розділений крапкою. Нумерація йде таким чином:

− перша версія має номер 1.0;

− наступна версія – 1.1;

− версія, що зазнала незначних змін – 1.11, у разі внесення нової функціональності – 2.0;

− час від часу розробники можуть «перескочити», наприклад, з версії 2.0 відразу до версії 2.5, щоб вказати на додавання декількох значних змін у програмі, що недостатньо значні для переходу до версії 3.0;

− для версій, що виходять до офіційного релізу, використовуються значення менше 1, наприклад, 0.1 або 0.5.

Версія в схемі «послідовність чисел» має складається з декількох чисел (найчастіше трьох), розділених крапкою, де перше значення – старша версія, друге – молодша, третє – дрібні зміни. У разі збільшення одного із значень всі числа, що йдуть після збільшеного, обнулюються. Подібна схема використовується в специфікації семантичного версіонування. Крім того, іноді четвертим числом додають номер пакету з наскрізною нумерацією – це значення може збільшуватись на одиницю з кожним випуском або обирається за певним технічним лічильником. Так само як і у варіанті версіонування десятковим дробом, попередні випуски мають нуль як перше значення версії (0.1 або 0.5).

Іноді під час використання схеми «буква в ролі молодшої версії» замість третього числа використовується буква. Це може вказувати, наприклад, на те, що програма не змінила функціональності, проте було виправлено певні помилки.

У разі варіанту версіонування «вказівка стадії розробки» до номеру версії додається стадія розробки: альфа-версія (alpha), бета-версія (beta), випуск-кандидат (release candidate), кінцевий випуск (general available), виправлення помилок (service pack) тощо. Є різні види схеми позначення стадії розробки, наприклад третє число може означати:

− 0 – альфа;

− 1 – бета;

− 2 – випуск-кандидат;

− 3 – кінцевий випуск.

Всередині компанії також може додаватись стадія розробки. Проте при офіційному випуску, що отримує кінцевий користувач, цього немає з метою виключити плутанину серед тестувальників.

Найчастіше «алфавітно-цифрова назва» використовується для програмного забезпечення

з довгою історією та версіями, що рідко виходять. Наприклад, коли рахунок версій зайшов надто далеко та потрібно його обнулити: Adobe Photoshop 7.0 < CS < CS2 < … < CS6 < CC < CC 2014. Іноді на доповнення до звичайної версії використовуються алфавітно-цифрова підназва: Ubuntu 9.04 Jaunty Jackalope, Embarcadero Delphi 10.2 Tokyo.

Варіант версіонування «дата» зазвичай використовується для версіонування програмного забезпечення, що рідко оновлюються. У разі використання дати в нумерації версії варто користуватись схемою ISO «рік-місяць-день» (дефіс можна викинути), щоб полегшити порівняння версій.

Часто програмний продукт має і публічну (маркетингову) назву, і так звану «внутрішню версію». Внутрішня версія складається за правилами. Зазвичай внутрішню версію використовують у випадках, коли виникає помилка і необхідно вказати розробникам, у якій саме версії виникла помилка.

Підхід використаний у системі комп'ютерної верстки TeX, що був запропонований Дональдом Кнутом, полягає у вказівці версії відповідно до значень наближеними до числа π: 3.0 < 3.1 < 3.14 тощо. Номер останнього стабільного випуску – 3.141592653.

При виборі варіанту версіонування важливо враховувати таке [2]:

- версія повинна мати сенс;
- версія повинна відображати сумісність;
- версії мають бути легко порівнюваними;
- версії мають бути зрозумілими для користувачів.

Специфікація семантичного версіонування

Найпопулярнішою специфікацією версіонування є семантичне версіонування. Автором специфікації семантичного версіонування є Том Престон-Вернер, засновник Gravatars та співзасновник GitHub. Відповідно до неї, версія програмного продукту містить у собі три числа, розділених крапкою, де кожне з чисел (X.Y.Z) має відповідну назву – мажорна, мінорна, патч.

Патч-версія повинна збільшуватись, якщо в програмний продукт були внесені зворотно сумісні баг-фікси. Відповідно до [1] баг-фікс означає внутрішні зміни, які виправляють певну помилку поведінки системи.

Мінорна версія повинна збільшуватись, якщо в систему була внесена певна зворотно сумісна функціональність, а також якщо деяка функціональність відмічена як застарівша. Мінорна версія також може збільшуватись у разі реалізації нової функціональності або удосконалення в приватному коді. У разі збільшення мінорної версії патч-версія повинна обнулитися.

Мажорна версія збільшується у разі внесення зворотно несумісної функціональності. Вона також може включати в себе зміни, які характерні для мінорної версії та патч-версії. У разі збільшення мажорної версії мінорна версія та патч-версія повинні обнулятись.

Також можливо позначити передрелізну версію, додаючи безпосередньо після патч-версії дефіс та ідентифікатор предрелізної версії, наприклад: 1.0.0-alpha, 1.0.0-0.3.7. Крім того, можна також позначити метадані у версії – необхідно поставити знак плюс після номеру патч-версії або передрелізної версії та додати ідентифікатор метаданих. В обох випадках ідентифікатори можуть містити лише алфавітно-цифрові символи ASCII та дефіс і можуть бути розділені крапкою.

Класифікація баз даних за типом їх розміщення

Важливою частиною інформаційної системи є база даних, але вона не може існувати без СУБД (системи управління базами даних). Едгар Ф. Кодд запропонував такі функції та сервіси, які повинні мати СУБД загального призначення [3]:

- зберігання, пошук та оновлення даних;
- доступний для користувачів каталог або словник даних, що описують метадані;
- підтримка транзакцій та конкурентного доступу;
- засоби для відновлення бази даних у разі її пошкодження;
- підтримка авторизації для надання доступу на читання та запис;
- можливість віддаленого доступу;
- підтримка обмежень для забезпечення відповідності даних, що зберігаються, до певних правил.

Класифікувати СУБД можна за різними критеріями, але доцільно виділити класифікацію за способом доступу до бази даних, а саме вони розділяються на файл-серверні, клієнт-серверні та вбудовані.

Файл-серверні СУБД. В цьому разі база даних знаходиться централізовано на файл-сервері. Доступ СУБД до даних виконується через локальну мережу. Зчитування та оновлення виконується з використанням блокування файлів. Такі СУБД застосовуються зазвичай у локальних додатках, що використовують функції управління базами даних, а також у системах з низькою інтенсивністю обробки даних і низьким піковим навантаженням на базу даних.

Перевагою цієї архітектури є низьке навантаження на процесор файлового серверу.

Проте файл-серверні СУБД мають велику кількість недоліків:

− низька ефективність роботи в рамках комп'ютерної мережі;

− складність або неможливість централізованого управління;

− складність або неможливість таких важливих характеристик, як висока надійність, висока доступність та висока безпека.

На цей час використання файл-серверної технології недоцільно, адже така технологія вважається застарілою.

Клієнт-серверні СУБД розміщуються на сервері разом з базою даних і виконують доступ до бази даних безпосередньо, в монопольному режимі. Всі клієнтські запити на обробку даних виконуються клієнт-серверною СУБД централізовано. Недоліком таких СУБД є високі вимоги до серверу. Натомість клієнт-серверні СУБД мають достатньо переваг:

− висока ефективність у рамках комп'ютерної мережі;

− зручність централізованого управління;

− зручність забезпечення високої надійності, високої доступності та високої безпеки.

Вбудована СУБД. Така система управління базами даних може постачатися як частина якогонебудь програмного продукту та не потребує процедури самостійного встановлення. Вбудовані СУБД призначені для локального збереження даних своєї системи та не розраховані на колективне використання в мережі.

Фізично вбудована СУБД зазвичай реалізується у вигляді бібліотеки. Доступ до даних з боку додатка може відбуватися через SQL або через спеціальні програмні інтерфейси. Прикладом використання вбудованих СУБД є поштові клієнти та сервіси обміну повідомленнями, медіапрогравачі тощо. Зазвичай у таких СУБД відсутні функції авторизації, адже більше ніж одного користувача бази даних для роботи не потрібно. Для ізоляції транзакції використовується блокування файлів через стандартні механізми операційної системи. Надійність бази даних залежить від надійності бібліотеки СУБД та файлової системи, на якій база даних знаходиться. Популярні вбудовані СУБД зазвичай добре протестовані, а сучасні файлові системи достатньо надійні, проте є багато способів втратити дані, тому такі рішення поступаються за надійністю клієнт-серверним СУБД.

Специфікація семантичного версіонування баз даних

Авторами статті запропоновано перелік правил та рекомендацій, керуючись якими можна легко здійснювати версіонування бази даних. Для цього переліку правил, в основі яких є специфікація семантичного версіонування, авторами запропоновано назву «семантичне версіонування баз даних».

Зміни в схемі бази даних доцільно розподілити на групи операцій, що виконуються на переліку елементів схеми, таблицях, полях таблиць, представленнях, збережуваних процедурах (функціях), тригерах.

На переліку елементів схеми можливі тільки дві операції: додавання нового або видалення наявного елементу. Додавання нового елементу є зворотно сумісною операцією, що відповідно до специфікації семантичного версіонування призводить до інкременту мінорної версії. Видалення елементу є зворотно несумісною операцією, що, відповідно, призведе до інкременту мажорної версії.

Над полями таблиць виконуються операції додавання, видалення, зміни. Попередньо варто зазначити, що визначення поля таблиці складається з назви поля, його типу даних, можливості збереження NULL-значення та наявності значення за замовчуванням. У разі додавання нового поля до таблиці виникає зворотно несумісна зміна структури, тому відбувається інкремент мажорної версії, і тільки якщо додаване поле має значення за замовчуванням, інкремент відбудеться за мінорною версією. Видалення стовпця в будь-якому разі є зворотно несумісною операцією, тому інкремент відбудеться у мажорній версії схеми бази даних. Під час редагування зміну кожного зі складників визначення поля таблиці варто виконувати окремим DDL-запитом. Зміну назви поля таблиці можна представити у вигляді додавання нового поля та видалення старого, а у разі видалення інкремент виконується за мажорною версією, тому і перейменування поля призводить до зміни за мажорною версією. У разі зміни типу даних необхідно враховувати, що початковий та кінцевий типи повинні бути сумісними. Сумісність типів описано в стандарті ISO/IEC 9075, останньою редакцією якого на момент написання статті є ISO/IEC 9075:2016 Information technology – Database languages – SQL, а саме у пункті 6.13 <cast specification> частини 2 Foundation (SQL/Foundation). Якщо типи є сумісними, то дану зміну можна виконувати з інкрементом мінорної версії, або патч-версії,

якщо виправляється помилка вибору типу даних для поля. Додаючи NULL-значення, виникає розширення множини інваріантів збережуваних типів полем таблиці і, як наслідок, цю операцію необхідно виконувати з інкрементом мінорної версії у разі додавання нової функціональності, або патч-версії у разі виправлення помилки проектування бази даних. Відповідно, у разі видалення NULL-значення інваріанти збережуваних типів даних колонки звужуються, що призводить до втрати зворотної сумісності, тому ця зміна схеми бази даних має виконуватись із інкрементом мажорної версії. Найчастіше розробники наявні NULL-значення прирівнюють до значення за замовчуванням. Аналогічним чином виконується додавання або видалення значення за замовчуванням. У разі видалення втрата зворотної сумісності викликана необхідністю явно вказувати значення поля у INSERT-запитах.

Операції додавання або видалення обмежень виконуються над полями та таблицями. У разі додавання обмежень виникає звуження множини варіантів значень даних, що можуть бути збережені у таблиці чи полі, тому зміни виконуються з інкрементом мажорної версії. А у разі видалення обмежень, розширюється набір варіантів значень, що зберігаються, тому зміни вносяться з інкрементом мінорної версії, або патч-версії, у разі виправлення помилки.

Вплив на версію бази даних змін в представленнях проходить аналогічно до змін полів таблиці. Тобто будь-яке додавання поля, розширення можливих значень поля, зворотно сумісна зміна типу даних поля – всі операції, що не порушують зворотної сумісності, – виконуються з інкрементом мінорної версії, або патч-версії. А своєю чергою операції, що призводять до порушення сумісності, мають бути виконані з інкрементом мажорної версії.

Розглядаючи вплив операцій над збережуваними процедурами на версію схеми бази даних, можна застосувати один із SOLID-принципів об'єктно-орієнтованого програмування, а саме принцип підстановки Барбари Лісков. Цей принцип декларує поняття заміщення: якщо S підтип T, тоді об'єкти типу T у програмі можуть бути заміщені об'єктами типу S без будь-яких змін бажаних властивостей цієї програми. Принцип підстановки Барбари Лісков близько стосується методології проектування за контрактом і веде до деяких обмежень на те, як контракти можуть взаємодіяти з наслідуванням [4]:

− передумови не можуть бути посилені в підтипі;

− післяумови не можуть бути послаблені в підтипі;

− інваріанти базового типу повинні виконуватись у підтипі;

− «історичне обмеження»: заборона модифікації стану об'єкта методами підтипу, які відсутні в базовому типі.

Екстраполюючи цей принцип на збережувані процедури, уявляючи, що змінена збережувана процедура є дочірньою до початкової, можна зазначити таке:

− передумовами є перелік, порядок та вимоги до вхідних параметрів;

− післяумовами є перелік, порядок та вимоги до вихідних параметрів;

− збереження поведінки є виконанням «історичного обмеження».

Якщо всі вимоги виконані, то збережена зворотна сумісність при внесенні змін, тому вони вплинуть на версію схеми бази даних інкрементом мінорної або патч-версії. У разі втрати зворотної сумісності зміни необхідно вносити з інкрементом мажорної версії.

Зміна тригерів не впливає на версію бази даних або може позначатися інкрементом мінорної версії у разі автоматичного виконання обов'язкових каскадних операцій.

Загальноприйнята розробниками стратегія видалення будь-яких елементів схеми бази даних відбувається через оголошення елементу застарілим (англ. deprecated) з інкрементом мінорної версії та наступним його видаленням з інкрементом мажорної версії.

Висновки. Авторами статті було запропоновано специфікацію семантичного версіонування баз даних, що включає в себе ряд правил щодо зміни версії схеми бази даних. Дотримання запропонованих правил забезпечує можливість:

− розробникам контролювати зміни у базі даних та порівнювати різні її версії;

− користувачам пересвідчитись в тому, що у разі оновлення бази даних програмні додатки, що її використовують, не втратять своєї працездатності;

− автоматично переходити за змінами в мінорному та патчевому сегменті версій.

Список літератури:

1. Том Престон-Вернер Семантичне Версіонування 2.0.0. URL: https://semver.org/.
2. Назаров К. Экстремально предвзятый взгляд на версионирование программных продуктов. URL: https://www.slideshare.net/racktear/semver.
3. Connolly T., Begg C. Database Systems – A Practical Approach to Design Implementation and Management. Pearson, 2015, pp. 97–102.
4. Janssen T. SOLID Design Principles Explained: The Liskov Substitution Principle with Code Examples. URL: https://stackify.com/solid-design-liskov-substitution-principle/.

Struzik V.A., Hrybkov S.V., Chobanu V.V. SEMANTIC VERSIONING OF DATABASE SCHEMA

Need for versioning in software development and software support is analyzes in the article. Existing versioning schemes and features of each are described here. Semantic versioning is highlighted separately because this scheme is the most popular among software development companies. This article describes the basic rules for changing a software version according to the semantic versioning specification.

The article describes the classification of databases by the access type, namely: file server, client server and embedded. Not only program code has changes in the process of development and support. Database has changes too. It is important to implement database versioning for changes control. However, there are no well-known database versioning specification. Usually versioning rules are applied by the development team as a manifest. Therefore, authors of the article propose the versioning specification of database which is based on semantic versioning. The article provides basic guidelines for changing a database version according the semantic versioning of database schema. There are shows the effect of some changes in a database schemes on the version value of the database.

Existing versioning schemes are analyzed in detail, namely: number of sequences, decimal numbers, incrementing sequences, letter as the lowest part in the version, designating development stage, alphanumeric name, date, internal versions. The article analyzes the appropriateness of using versions and describe the benefits of its usage. In addition, the article provides information about issues that the development team may encounter if they refuse usage of versioning. The most popular software version specification, semantic versioning, is described in detail and main points from its specification are outlined.

Key words: *versioning, database, semantic versioning specification, semantic versioning of database, database management system.*