МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ

КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка інформаційно-довідкової системи обробки даних з використанням принципів нормалізації табличних значень»

Рівень вищої освіти другий (магістерський)
Спеціальність 122 Комп'ютерні науки
Освітня програма Комп'ютерні науки

Виконав: студент групи МгІТ-1-22

Козак Даниїл Олександрович

Науковий керівник д.т.н., професор Володимир ЩЕРБАНЬ

Рецензент _____

Київ 2023

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ

ФАКУЛЬТЕТ МЕХАТРОНІКИ ТА КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

Спеціальність **122 Комп'ютерні науки**
Освітня програма **Комп'ютерні науки**

ЗАТВЕРДЖУЮ

завідувач кафедри КНТ
_____ проф. Володимир ЩЕРБАНЬ
"_____" _____2023 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Козаку Даниїлу Олександровичу

1. Тема роботи Розробка інформаційно-довідкової системи обробки даних з використанням принципів нормалізації табличних значень
Науковий керівник роботи: Щербань Володимир Юрійович, завідувач кафедри
затверджені наказом КНУТД від "_12_"вересня_2023 року №_210-уч_

2. Вихідні дані до роботи: Розробки кафедри комп'ютерних наук; рекомендована література, додатки.

3. Зміст дипломної роботи: Вступ; РОЗДІЛ 1 Постановка задачі; РОЗДІЛ 2 Проектування; РОЗДІЛ 3 Програмна реалізація; Висновки; Список літератури; ДОДАТОК А Окремі фрагменти програмного коду; ДОДАТОК Б Презентація.

4. Дата видачі завдання_1 вересня 2023

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапу кваліфікаційної роботи	Орієнтовний термін виконання	Примітка про виконання
1	Вступ	*15.09.2023*	
2	Розділ 1. Постановка задачі	*20.09.2023*	
3	Розділ 2 Проектування	*30.09.2023*	
4	Розділ 3. Програмна реалізація	*10.10.2023*	
5	Висновки	*25.10.2023*	
6	Оформлення (чистовий варіант)	*1.11.2023*	
7	Подача кваліфікаційної роботи науковому керівнику для відгуку (за 14 днів до захисту)	*4.11.2023*	
8	Подача кваліфікаційної роботи для рецензування (за 12 днів до захисту)	*6.11.2023*	
9	Перевірка кваліфікаційної роботи на наявність ознак плагіату (за 10 днів до захисту)	*8.11.2023*	
10	Подання кваліфікаційної роботи на затвердження завідувачу кафедри (з 7 днів до захисту)	*10.11.2023*	

З завданням ознайомлений:

Студент _____ Даниїл КОЗАК

Науковий керівник _____ Володимир ЩЕРБАНЬ

АНОТАЦІЯ

Козак Д.О. Розробка інформаційно-довідкової системи обробки даних з використанням принципів нормалізації табличних значень. – Рукопис.

Кваліфікаційна магістерська робота за спеціальністю 122 – «Комп'ютерні науки ». – Київський національний університет технологій та дизайну, Київ, 2023 рік.

Дипломну магістерську роботу присвячено дослідженню, розробці та реалізації інформаційно-довідкової системи обробки даних з використанням принципів нормалізації табличних значень.

Запропоновано основні напрями удосконалення системи обробки даних з використанням принципів нормалізації табличних значень. Подальший розвиток та впровадження цієї інформаційної системи матиме велике значення для ефективної роботи різноманітних галузей бізнесу та допоможе вирішити проблеми, пов'язані з управлінням даними.

Для реалізації поставленого завдання було підготовлено матеріал, який складається з кількох основних розділів, які спрямовані на дослідження, проектування, реалізацію та тестування інформаційно-довідкової системи, а також на аналіз отриманих результатів та формулювання висновків щодо подальшого розвитку даної теми.

Ключові слова: інформаційна система, набір засобів, зберігання методів.

ANNOTATION

Kozak D.O. Development of an information and reference system for data processing using the principles of normalization of table values. - Manuscript.

Qualifying master's thesis in specialty 122 - "Computer science". - Kyiv National University of Technology and Design, Kyiv, 2023.

The master's thesis is devoted to the research, development and implementation of an information and reference system for data processing using the principles of normalization of table values.

The main directions for improving the data processing system using the principles of table value normalization are proposed. The further development and implementation of this information system will be of great importance for the effective operation of various business sectors and will help solve problems related to data management.

To accomplish this task, we have prepared material that will consist of several main sections aimed at researching, designing, implementing and testing the information and reference system, as well as analyzing the results and drawing conclusions on the further development of this topic.

Keywords: information system, set of tools, storage of methods.

ЗМІСТ

ВСТУП

Сучасний світ вимагає надійних та ефективних систем обробки даних, що стає важливим завданням для багатьох галузей та організацій. Розробка інформаційно-довідкової системи з використанням принципів нормалізації табличних значень є актуальною та важливою темою, оскільки вона спрямована на покращення збереження та обробки даних з урахуванням сучасних стандартів та методів роботи з базами даних.

В інформатиці поняття "система" найчастіше використовують стосовно набору технічних засобів і програм. Системою називають також апаратну частину комп'ютера. Доповнення поняття "система" словом інформаційна" відображає мету її створення і функціонування. Формаційна система - взаємозв'язана сукупність засобів, методів і персоналу, використовувана для зберігання, оброблення та видачі інформації з метою вирішення конкретного завдання.

Мета даної дипломної роботи полягає у розробці інформаційної системи, яка забезпечить ефективне збереження та обробку даних шляхом використання принципів нормалізації табличних значень. Ця система буде спрямована на вирішення потреб сучасних підприємств у керуванні та обробці інформації.

Дипломна робота буде присвячена вивченню та аналізу сучасних принципів нормалізації даних, розробці структури бази даних, створенню функціональності системи обробки даних, а також тестуванню та апробації розробленої системи.

Результати даної роботи можуть бути корисними для підприємств, які мають потребу в ефективній системі управління даними, а також для дослідників, які цікавляться питаннями розробки інформаційних систем з використанням принципів нормалізації табличних значень.

Розробка інформаційно-довідкової системи обробки даних з використанням принципів нормалізації табличних значень є важливим кроком у забезпеченні ефективної та точної обробки даних. Нормалізація даних дозволяє зменшити залежність між таблицями та уникнути дублювання інформації, що може призвести до неправильних аналітичних результатів та втрати ефективності системи.

Враховуючи важливість теми та актуальність розв’язання поставленої проблеми, дана дипломна робота має перспективи для подальших досліджень та розвитку в області інформаційних технологій та обробки даних.

РОЗДІЛ 1. Постановка задачі

1.1. Роль та важливість обробки даних у сучасному світі

Щоб відповісти на питання "Що таке обробка даних?", необхідно зрозуміти, як відбувається обробка даних. Обробка даних перетворює дані в цінний і прийнятний формат відповідно до заздалегідь визначеної послідовності дій. Це перетворення можна зробити вручну або механічно. Багато фахівців обробляють дані за допомогою комп'ютерів. Оброблені дані можна відображати в різних форматах, наприклад, у вигляді зображень, графіків, таблиць і діаграм. Програма або метод обробки даних, який ви використовуєте, може визначати макет ваших даних.

Обробка даних є важливою операцією для будь-якого бізнесу, оскільки вона допомагає витягувати потрібну інформацію для подальшого використання. Багато галузей, як-от банківська, фінансова, освітня та інші великі корпорації, потребують зберігання точної інформації у своїх системах для негайного або майбутнього використання.

Інформаційно-довідкові системи (ІДС) - це засоби, які забезпечують доступ користувача до різноманітної інформації та довідкових матеріалів. Їх головна мета - надання конкретних відповідей на запитання користувача, пошук потрібної інформації, підтримка процесу прийняття рішень.

Використання ІДС є широким і розповсюдженим в різних сферах діяльності. Для бізнесу вони допомагають вирішувати питання пошуку ринку, аналізу конкурентів, розробки стратегій. Державні структури використовують ІДС для забезпечення доступу до нормативно-правової бази, статистичних даних та іншої інформації. В освіті ІДС допомагають студентам та вчителям швидко знайти необхідну літературу, матеріали для викладання.

ІДС можуть бути представлені у вигляді баз даних, електронних довідників, пошукових систем, енциклопедій, електронних бібліотек та інших довідкових ресурсів. Вони можуть бути доступні як онлайн-сервіси, програмні додатки або інтегровані в інші програми.

Використання ІДС дозволяє ефективно використовувати час та зусилля в процесі пошуку необхідної інформації, забезпечує швидкий доступ до актуальних даних і сприяє прийняттю обґрунтованих рішень. Однак, для успішного використання ІДС необхідно мати навички роботи з ними, вміння правильно формулювати запити та аналізувати результати.

Етапи обробки даних

Будь-яка діяльність, що передбачає збір даних, вимагає їхньої обробки. Зібрана інформація включає в себе збір, сортування, обробку, аналіз та представлення. Ми можемо розділити всю процедуру на шість основних етапів, які включають

1. Збір даних

Початковим етапом обробки даних є збір даних. Після цього можна збирати дані з різних джерел, таких як оперативні бази даних і сховища даних. Збір даних з авторитетних джерел може забезпечити точність і надійність даних.

2. Сортування даних

Після збору даних наступним етапом є їх сортування. Упорядкування даних передбачає вилучення та структурування необроблених даних для підготовки їх до наступного кроку. На цьому етапі вам, можливо, доведеться перевірити необроблені дані на наявність помилок. Метою сортування даних є усунення надлишкових, неповних або неправильних даних. Кінцевий продукт дає високоякісні дані, які є релевантними для бізнес-аналітики.

3. Введення даних

Введення даних - це перший крок у перетворенні необроблених даних на придатну для використання інформацію. На цьому етапі ви можете передати чисті дані за призначенням. Цей етап також передбачає перетворення даних на придатну для використання інформацію, яку може зрозуміти програмне забезпечення для управління взаємовідносинами з клієнтами або сховища даних.

4. Обробка даних

Комп'ютер обробляє дані, введені на попередньому етапі, для інтерпретації. Хоча процедура може дещо відрізнятися залежно від джерела

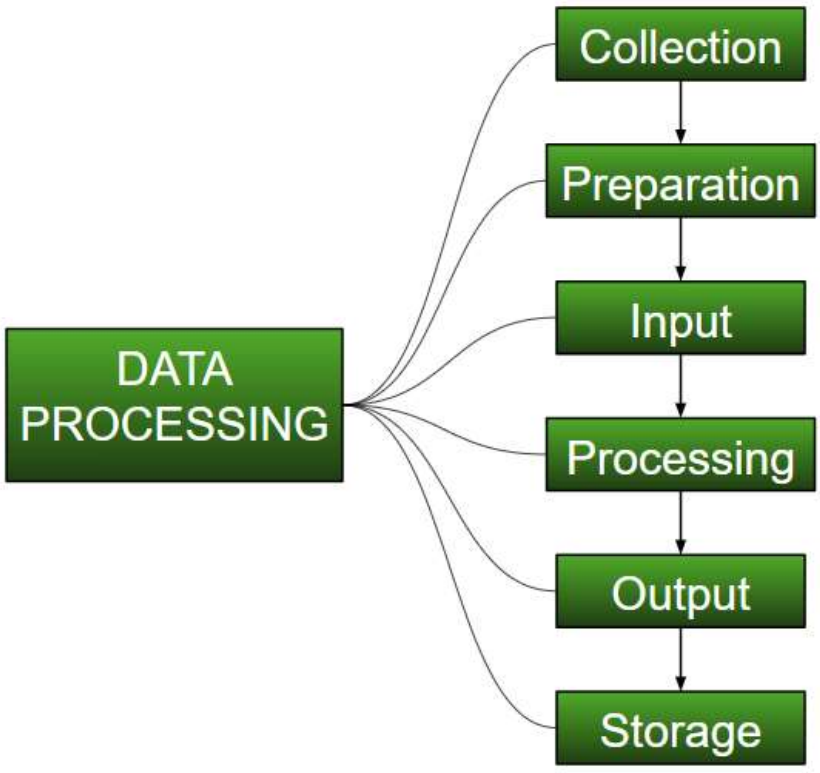
даних і мети, алгоритми машинного навчання часто завершують цей етап. Наприклад, визначення потреб клієнтів, вивчення рекламних тенденцій і встановлення медичного діагнозу на основі звітів про дані - це різні цілі обробки даних.

5. Виведення даних

На етапі виведення даних аналітики, які не є аналітиками даних, можуть використовувати та інтерпретувати дані. Ці фахівці часто перетворюють дані на зрозумілу інформацію, яка часто складається з графіків, відео, фотографій, текстів та інших форматів. Співробітники організації можуть використовувати цю інформацію для своїх проектів.

6. Зберігання даних

Зберігання даних - це останній етап обробки даних. Після аналізу та обробки всіх даних ви можете зберегти їх для подальшого використання. Законодавство про захист даних, наприклад, Загальний регламент про захист даних (GDPR), якого дотримуються компанії, вимагає правильного зберігання даних. Тому, коли ви зберігаєте дані правильно, люди можуть швидко і легко отримати до них доступ, коли вони цього побажають.



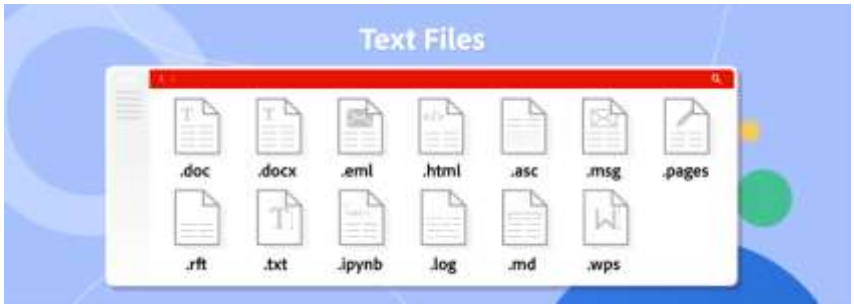
Етапи обробки даних

Форми виведення оброблених даних

Існує декілька форм зберігання даних, деякі з яких ми розглянемо нижче:

- Прості текстові файли

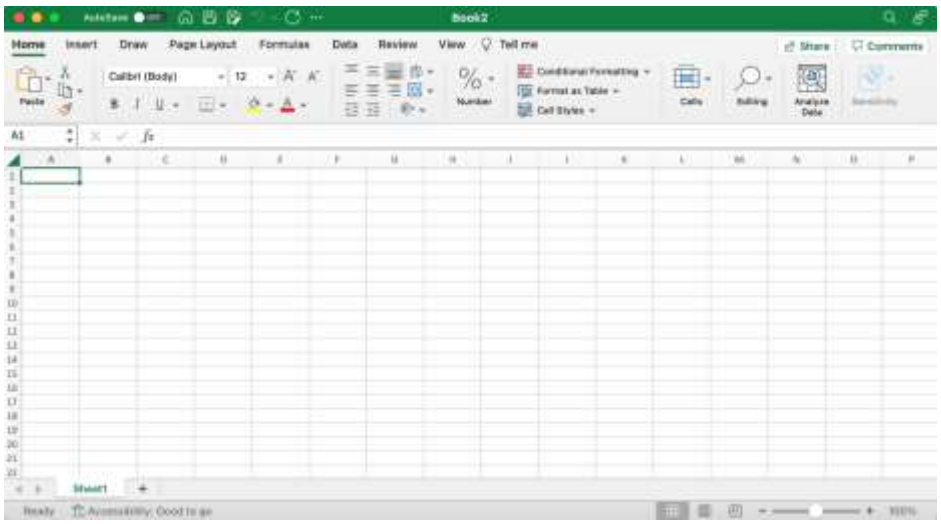
Це найпростіші правила виведення оброблених даних. Багато з цих файлів є зручними для користувача і простими для інтерпретації. Ці файли не потребують додаткової обробки, і ви можете зберігати їх як текстові файли, які займають мало місця на диску.



Прості текстові файли

- Електронна таблиця

Це загальновживаний формат для числових даних. Користувач може виконувати різні операції з цифрами в рядках і стовпчиках. Наприклад, фільтрація та сортування даних у певному порядку може покращити їхнє розуміння читачем і полегшити застосування цієї інформації. Читачі також можуть виконувати різні математичні завдання, використовуючи вихідні дані цього файлу.



Електронна таблиця

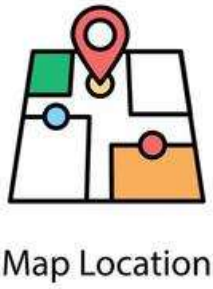
- Діаграми та графіки

Створення вихідних даних у вигляді діаграм і графіків є корисною функцією, яка зараз включена до багатьох програм. Такий підхід корисний при роботі з числами, які ілюструють закономірності, зростання або падіння. Існує багато діаграм і графіків на вибір для виконання різних функцій. Деякі клієнти або користувачі також надають рекомендації щодо конкретних графіків для візуалізації даних. Ви також можете спроектувати і створити власну діаграму, якщо немає вбудованого графіка або діаграми.

Діаграми та графіки

- Карти, векторні та графічні файли

Під час роботи з просторовими даними, тобто даними, які пов'язані з певним місцем на Землі, можливість експортувати оброблені дані у вигляді карт, векторів і графічних файлів може допомогти багатьом фахівцям. Багато фахівців використовують карти, вектори та файли зображень для виконання своїх обов'язків, наприклад, лісівники, кліматологи та картографи. Для містобудівників, які працюють з різними картами, наявність вашої інформації на картах є корисною для планування та проектування громад.



Карти, векторні та графічні файли

1.2. Актуальність теми

Перше на що хочеться звернути увагу, що вивчення баз даних вважається однією з найважливіших основ для будь-якого комп'ютерного фахівця, який прагне розвинути набір навичок, необхідних для сучасного бізнесу, а життєва важлива роль баз даних у багатьох аспектах повсякденного життя зробила їх однією з найбільш затребуваних спеціальностей на ринку праці.

Хоча загальна думка всієї комп'ютерної спільноти стверджує, що важливість вивчення БД в комп'ютерній галузі є беззаперечним питанням, це пов'язано з тим, наскільки глибоко БД пов'язані з іншими галузями, наприклад: розробник програмного забезпечення повинен володіти знаннями про те, як дані, отримані з веб-сайтів і додатків, повинні зберігатися і організовуватися, а також про те, який тип технологій БД підходить для їх бізнес-додатків.

Загалом, система баз даних допомагає організаціям і фірмам зберігати, управляти і оптимально використовувати дані, що зберігаються, за допомогою декількох інтегрованих технологій. Експоненціальне зростання обсягів даних, що генеруються щодня, підкреслює необхідність їхнього використання у добре структурованому вигляді. Це спричинило появу нових досліджень, які випливають з ядра баз даних, таких як машинне навчання та наука про дані. Через швидкий розвиток цих нових технологій як малі, так і великі компанії почали покладатися на великі дані у прийнятті рішень, що має неоціненний вплив на зростання бізнесу.

SQL вважається офіційною розмовною мовою будь-якої системи управління базами даних, і будь-який фахівець з баз даних повинен мати ґрунтовні знання про SQL. Згідно з опитуванням розробників Stack Overflow 2019, SQL посідає третє місце за популярністю серед професійних розробників, і він знаходиться в топ-5 з 2015 року. Це показує, наскільки важливо вивчити SQL, щоб мати справу з великими обсягами даних і допомогти вирішити ряд бізнес-кейсів, пов'язаних з бізнесом.

Також хочеться зауважити, що важливість обробки даних покращує доступність даних. Тобто дані є активом для будь-якої організації, і їх зберігання

після збору забезпечує легкий доступ до них для подальшого використання. Крім того, це зменшує потребу в регулярному зборі даних.

Прискорює виконання повторюваних завдань: Обробка даних полегшує перевірку операцій, модифікацій та транзакцій. Завдяки ретельно обробленим даним деякі організації, наприклад, страхові компанії, можуть обробляти та врегульовувати страхові випадки.

Покращує завдання управління записами: Завдяки пакетній обробці та надійній системі баз даних ви можете зберігати, керувати та створювати електронні медичні записи. Крім того, ви можете додати до обробки даних обробку зображень для створення візуально привабливих і всеосяжних графічних зображень для візуалізації даних.

Підвищує читацьку привабливість: Обробка даних, як і обробка тексту, може допомогти у створенні публікацій, які є привабливими для читачів. Це може допомогти покращити залучення читачів, що може покращити показники продажів вашої організації.

Сприяє збору даних опитувань: Уряд може використовувати отримані дані для економії часу на проведення опитувань, що може допомогти йому надавати індивідуальні послуги в різних географічних точках. Ви можете швидко створювати звіти та швидше вирішувати проблеми, маючи безпечні та надійні дані.

1.3. Поняття нормалізації табличних значень

Нормалізація - це процес організації даних у базі даних. Він включає створення таблиць і встановлення в’язків між цими таблицями відповідно до правил, призначених як для захисту даних, так і для того, щоб зробити базу даних гнучкішою, усунувши надмірність і непослідовну залежність.

Нормалізація бази даних - це процес організації даних у таблиці таким чином, щоб результати використання бази даних завжди були однозначними і відповідали призначенню. Така нормалізація є невід'ємною частиною теорії реляційних баз даних. Вона може призвести до дублювання даних у базі даних і часто призводить до створення додаткових таблиць.

Концепція нормалізації баз даних, як правило, бере свій початок від Е.Ф. Кодда, дослідника IBM, який у 1970 році опублікував статтю, що описує модель реляційної бази даних. Те, що Кодд назвав "нормальною формою для в’язків між базами даних", стало важливим елементом реляційної техніки. Така нормалізація даних знайшла готову аудиторію в 1970-х і 1980-х роках - час, коли дискові накопичувачі були досить дорогими, а високоефективні засоби для зберігання даних були вкрай необхідні. З того часу інші методи, зокрема де нормалізація, також знайшли прихильників.

Хочемо виділити декілька правил нормалізації даних. Хоча правила нормалізації даних, як правило, збільшують дублювання даних, вони не призводять до надмірності даних, яка є непотрібним дублюванням. Нормалізація бази даних, як правило, є процесом уточнення після початкового визначення об'єктів даних, які повинні бути в реляційній базі даних, виявлення їхніх в’язків і визначення необхідних таблиць і стовпців у кожній таблиці. Приклад наведено у табл. 1.1

Таблиця 1.1

Покупець	Придбаний товар	Ціна
Марко	Товар 1	560 грн
Оксана	Товар 1	254 грн
Ефім	Товар 2	59 грн
Мелана	Товар 3	275 грн

Якщо ця таблиця використовується для відстеження цін на товари і користувач хоче видалити одного з клієнтів, він також видалить ціну. Нормалізація даних означитиме розуміння цього і вирішення проблеми шляхом розділення цієї таблиці на дві таблиці, одна з яких міститиме інформацію про кожного покупця і товар, який він придбав, а друга - про кожен товар і його ціну. Додавання або видалення даних в одній таблиці не вплине на іншу.

Визначено ступені нормалізації таблиць реляційних баз даних, які включають

Перша нормальна форма (1 NF). Це "базовий" рівень нормалізації бази даних, і він в цілому відповідає визначенню будь-якої бази даних, а саме:

- Вона містить двовимірні таблиці з рядками та стовпчиками.
- Кожен стовпець відповідає під об’єкту або атрибуту об'єкта, представленого всією таблицею.
- Кожен рядок представляє унікальний екземпляр цього під об’єкта або атрибута і повинен відрізнятися від будь-якого іншого рядка (тобто дублікати рядків неможливі).
- Всі записи в будь-якому стовпчику повинні бути одного типу. Наприклад, у стовпчику з назвою "Клієнт" дозволено вводити лише імена або номери клієнтів.

Отже, перша нормальна форма (1НФ) є ключовою ідеєю в архітектурі реляційних баз даних. Вона гарантує, що дані організовані таким чином, щоб полегшити їх обробку, усунути надмірність і підтримати цілісність даних. 1НФ створює основу для більш складних стратегій нормалізації, які ще більше підвищують коректність та ефективність систем баз даних, накладаючи атомарні значення та забороняючи повторювані групування всередині рядків.

Друга нормальна форма (2 NF). На цьому рівні нормалізації кожен стовпець таблиці, який не є визначником вмісту іншого стовпця, повинен сам по собі бути функцією від інших стовпців таблиці. Наприклад, у таблиці з трьома стовпчиками, що містять ідентифікатор клієнта, проданий товар і ціну товару при продажу, ціна буде функцією від ідентифікатора клієнта (який має право на знижку) і конкретного товару. У цьому випадку кажуть, що дані у третьому стовпчику залежать від даних у першому та другому стовпчиках. У випадку з 1NF такої залежності немає.

Стовпець з ідентифікатором клієнта вважається первинним ключем, оскільки він однозначно ідентифікує рядки в цій таблиці і відповідає іншим прийнятим вимогам стандартної схеми управління базами даних: Він не має значень NULL і його значення не змінюються з часом.

Отже, 2NF - це фундаментальна концепція нормалізації баз даних, яка допомагає усунути часткові залежності у вашій реляційній базі даних. Дотримання правил 2NF допомагає організувати вашу базу даних, щоб уникнути аномалій і забезпечити цілісність даних, полегшуючи їх зберігання та пошук.

Третя нормальна форма (3 NF). У другій нормальній формі модифікації все ще можливі, оскільки зміна одного рядка в таблиці може вплинути на дані, які посилаються на цю інформацію з іншої таблиці. Наприклад, у щойно наведеній таблиці клієнтів видалення рядка, що описує покупку клієнта (можливо, через повернення), також призведе до видалення факту, що продукт має певну ціну. У третій нормальній формі ці таблиці були б розділені на дві таблиці, щоб ціни на продукти відстежувалися окремо.

Третя нормальна форма (3НФ) вважається адекватною для нормального проектування реляційних баз даних, оскільки більшість таблиць 3НФ вільні від аномалій вставки, оновлення та видалення. Крім того, 3НФ завжди забезпечує збереження функціональних залежностей та відсутність втрат.

Розширення базових нормальних форм включають нормалізовану форму домен/ключ, в якій ключ однозначно ідентифікує кожен рядок у таблиці, та нормальну форму Бойса-Кодда (BCNF), яка вдосконалює та покращує методи, що використовуються в 3NF для обробки деяких типів аномалій.

Здатність нормалізації баз даних уникати або зменшувати аномалії, надмірність і дублювання даних, одночасно підвищуючи цілісність даних, зробила її важливою частиною інструментарію розробників даних протягом багатьох років. Це одна з відмінних рис реляційної моделі даних.

Реляційна модель виникла в епоху, коли ділова документація велася насамперед на папері. Використання таблиць у ній частково було спробою відобразити тип таблиць, що використовувалися на папері, які виступали в якості первинного представлення даних (переважно бухгалтерських). Потреба у підтримці такого типу представлення зменшилася, оскільки на зміну паперовим записам прийшло цифрове представлення даних.

Але інші фактори також сприяли тому, що домінування нормалізації баз даних було поставлено під сумнів.

З часом постійне зниження вартості зберігання даних на дисках, а також нові аналітичні архітектури підірвали панування нормалізації. Де нормалізація як альтернатива почала серйозно розвиватися з появою сховищ даних, починаючи з 1990-х років. Пізніше з'явилися документ-орієнтовані бази даних NoSQL; ці та інші нереляційні системи часто використовують не диск орієнтовані типи сховищ. Зараз, більше ніж у минулому, архітектори та розробники даних балансують між нормалізацією та де нормалізацією даних під час проектування своїх систем.

1.4 Методи нормалізації даних

Нормалізація даних - важливий елемент аналізу даних. Це те, що дозволяє аналітикам збирати та порівнювати числа різного розміру, отримані з різних джерел даних. Це важливо, оскільки багато алгоритмів машинного навчання чутливі до масштабу вхідних даних і можуть давати кращі результати, коли дані нормалізовані.

Нормалізація зазвичай потрібна, коли ми маємо справу з атрибутами в різних масштабах, інакше це може призвести до розмивання ефективності важливого не менш важливого атрибуту (на нижчій шкалі) через інший атрибут, що має значення на більшій шкалі. Простіше кажучи, коли є декілька атрибутів, але вони мають значення на різних шкалах, це може призвести до поганих моделей даних при виконанні операцій з інтелектуального аналізу даних. Тому їх нормалізують, щоб привести всі атрибути до одного масштабу.

Причина недооцінки нормалізації, ймовірно, пов'язана з плутаниною навколо того, що це таке. Існують прості методи нормалізації, такі як видалення десяткових знаків, і складні методи нормалізації, такі як нормалізація за допомогою z-рахунку.

Нормалізація даних передбачає масштабування значень атрибутів таким чином, щоб вони чисельне лежали в одному інтервалі/масштабі, а отже, мали однакову важливість. Оскільки SVM створюють кращі моделі, коли дані

нормалізовані, всі дані повинні бути нормалізовані або стандартизовані перед класифікацією. Існує три методи нормалізації:

1. Нормалізація за Z-рахунком
2. Нормалізація за міні мак сним значенням
3. Нормалізація за допомогою десяткового масштабування.

Різниці між цими трьома методами немає. Більш детальніше опишемо нижче.

Нормалізація за Z-рахунком:

Нормалізація за Z-рахунком: Ця методика масштабує значення ознаки так, щоб її середнє значення дорівнювало 0, а стандартне відхилення - 1. Це робиться шляхом віднімання середнього значення ознаки від кожного значення, а потім ділення на стандартне відхилення.

Наприклад, якщо у вас є значення від 10 до 100 в одному вимірі, а в іншому - від 100 до 100 000, важко порівняти відносні зміни в обох вимірах. Коли ми стикаємося з цією проблемою, рішенням є нормалізація.

Мабуть, найпоширенішим типом нормалізації є z-рахунки. Простіше кажучи, z-рахунок нормалізує кожну точку даних до стандартного відхилення. Формула виглядає наступним чином:

де X - значення даних, μ - середнє значення набору даних, а σ - стандартне відхилення.

Нормалізація за допомогою десяткового масштабування:

Цей метод масштабує значення ознаки шляхом ділення значень ознаки на ступінь 10.

Нормалізація десяткових розрядів відбувається в таблицях даних з числовими типами даних. Якщо ви коли-небудь працювали з Excel, ви знаєте, як це відбувається. За замовчуванням Excel ставить дві цифри після коми для звичайних чисел, розділених комами. Ви маєте вирішити, скільки десяткових знаків вам потрібно, і масштабувати їх по всій таблиці.

Нормалізація за міні мак сним значенням:

Лінійна нормалізація є, мабуть, найпростішим і найгнучкішим методом нормалізації. Простіше кажучи, вона полягає у встановленні нової "бази" відліку для кожної точки даних. Цей метод, який часто називають "максимально-мінімальною" нормалізацією, дозволяє аналітикам взяти різницю між максимальним і мінімальним значенням x у наборі даних і встановити базу.

Нормалізація Min-Max: Ця методика масштабує значення ознаки до діапазону від 0 до 1. Це робиться шляхом віднімання мінімального значення ознаки від кожного значення, а потім ділення на діапазон ознаки.

Це гарна стартова стратегія, і насправді аналітики можуть нормалізувати точки даних до будь-якої бази, як тільки вони завершили лінійну нормалізацію. Ось формула лінійної нормалізації:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Якщо вони хочуть отримати основу 100, наприклад, це питання простої арифметики. Наприклад, уявіть, що у вас значення "x" дорівнює 20, максимальне число - 55, а мінімальне - 5. Щоб нормалізувати це число, давайте почнемо з обчислення основи 50 (55-5). Тепер нам просто потрібно змінити чисельник з тією ж ідеєю: x - min. У цьому випадку він стає 15 (20-5). Отже, наше стандартизоване x, або x', дорівнює 15/50 = 0,3.

Звичайно, якщо ми хочемо нормалізувати до 100, нам просто потрібно помножити або поділити дріб на число, необхідне для того, щоб знаменник став рівним 100. У цьому випадку це множення на 2. Ми множимо 50*2, щоб отримати 100, і 15*2, щоб отримати 30. Стандартизація становить 30/100 = 0,3. Зробіть це з усіма числами в наборі даних, щоб отримати базову стандартизацію 100.

Отже, нормалізація є важливим кроком в інтелектуальному аналізі даних, оскільки вона може допомогти поліпшити продуктивність алгоритмів машинного навчання, масштабуючи вхідні ознаки до загальної шкали. Це може допомогти зменшити вплив викидів і підвищити точність моделі.

Нормалізація використовується для масштабування даних атрибута так, щоб вони потрапляли в менший діапазон, наприклад, від -1.0 до 1.0 або від 0.0 до 1.0. Це зазвичай корисно для алгоритмів класифікації.

1.5. Приклади використання нормалізації в різних галузях

У нормалізованій базі даних загальна інформація зберігається в одній таблиці, а інформація, специфічна для кожного примірника (наприклад, ідентифікаційний номер), - в іншій таблиці. Це усуває необхідність зберігати ту саму інформацію кілька разів і полегшує відстеження того, яка інформація належить до якого примірника книги. Іншим прикладом є база даних клієнтів компанії, де кожен клієнт має унікальний ідентифікатор клієнта, ім'я, адресу та іншу особисту інформацію.

Компанія також може мати окрему таблицю для замовлень клієнтів, яка включатиме ідентифікатор клієнта, ідентифікатор замовлення та деталі замовлення. Нормалізувавши дані таким чином, компанія може легко отримати інформацію про конкретного клієнта та його замовлення без необхідності шукати всю інформацію про клієнта для кожного замовлення. Варто зазначити, що нормалізація - це процес, який включає кілька рівнів нормальних форм, кожен з яких має власний набір правил. Кожен рівень нормалізації спрямований на усунення різних типів надмірності даних.

Управління базами даних може підвищити ефективність бізнес-операцій, засвідчуючи точність інформації в базі даних.

Ведення структурованої бази даних може допомогти відділам використовувати і розуміти інформацію з бази даних та приймати практичні рішення. Впровадження методів нормалізації даних може дозволити вам покращити цілісність даних у базах даних на вашому робочому місці.

Нормалізація бази даних полягає в тому, як уникнути аномалій при оновленні таблиці. Отже, для сутності в таблиці має бути один і тільки один унікальний ідентифікатор.

Найочевиднішим прикладом є таблиця працівників у кадровій базі даних. Більшість сучасних кадрових систем містять згенерований системою

ідентифікаційний номер, який використовується в кожній таблиці для ідентифікації окремого працівника. Цей ідентифікатор залишається незмінним незалежно від зміни імені, адреси, військового статусу, номера телефону чи чогось іншого, що може змінитися з часом. Цей ідентифікатор є первинним ключем (PK) у всіх таблицях співробітників.

Всі поля в цих таблицях покладаються виключно на цей ключ, і будь-які зміни, внесені в будь-яке інше поле, можуть бути відображені у всій базі даних, оскільки єдиний первинний ключ (PK) пов'язує всі таблиці разом.

Коротше кажучи, можна сказати, що таблиця в реляційній базі даних знаходиться в нормальній формі Бойса-Кодда, коли всі поля в таблиці покладаються на ПК і тільки на первинний ключ.

Також як приклад із ще однієї галузі хочемо виділити роздрібний магазин.

Розглянемо базу даних, яка зберігає інформацію про клієнтів роздрібного магазину.

У ненормалізованій базі даних інформація про клієнтів може зберігатися в одній таблиці зі стовпчиками для імені, адреси, номера телефону, електронної пошти та історії покупок.

Такий дизайн призведе до дублювання даних, оскільки при зміні адреси або номера телефону покупця їх доведеться оновлювати в декількох місцях таблиці.

Нижче наведено приклад ненормалізованої бази даних, яка зберігає всю інформацію в табл. 1.2.

Таблиця 1.2 Інформація про клієнті

Інформація про клієнтів				
Ім’я	E-mail	Номер телефону	Дата	Номер у системі
Анна	E-mail@gmail.com	+(380)97 877 67 91	11.09.2019	101
Василь	E-mail@gmail.com	+(380)99 947 97 84	24.06.2022	102
Даша	E-mail@gmail.com	+(380)66 367 51 46	05.07.2023	1103

На противагу цьому, нормалізована база даних матиме окремі таблиці для інформації про клієнтів та історії покупок, з унікальним ідентифікатором, що пов'язує ці дві таблиці. Такий дизайн усуває дублювання даних і забезпечує їхню цілісність. Нижче наведено приклад нормалізованої бази даних у табл. 1.3.

Таблиця 1.3. Приклад нормалізованої бази даних

Інформація про клієнтів					Інформація про купівлю		
Ім’я	E-mail	Номер телефону	Дата	Номер у системі	Дата	Купівля	Номер у системі
Анна	E-mail@gmail.com	+(380)97 877 67 91	11.09.2019	101	11.09.2019	1000 грн	101
Василь	E-mail@gmail.com	+(380)99 947 97 84	24.06.2022	102	24.06.2022	500 Грн	102
Даша	E-mail@gmail.com	+(380)66 367 51 46	05.07.2023	1103	05.07.2023	1465 грн	1103

Загалом, нормалізація важлива для баз даних, оскільки вона допомагає забезпечити узгодженість і точність даних. Дотримуючись правил нормалізації, дані організовуються таким чином, щоб мінімізувати дублювання даних і забезпечити їхню цілісність, що полегшує підтримку та оновлення бази даних з часом.

Практично кожен бізнес використовує ту чи іншу форму збору даних, незалежно від того, наскільки він великий чи малий. У той час як великі підприємства мають більш усталені методи збору, зберігання та аналізу даних, менші компанії та Стартапи також починають розуміти цінність збору та аналізу даних для того, щоб інформувати про бізнес-рішення, сприяти зростанню.

Це особливо актуально в епоху великих даних і демократизації даних, коли ми маємо більше можливостей для аналізу даних, ніж будь-коли раніше.

Більшість підприємств вже збирають дані та керують ними за допомогою баз даних, CRM-платформ або систем автоматизації, але дані в різних формах і типах введення можуть призвести до суперечливої або дублюючої (надлишкової) інформації. Більш ефективний збір даних вимагає більш впорядкованого процесу управління даними. Саме тут на допомогу приходить нормалізація даних.

1.6. Висновок до розділу

З усього матеріалу можемо зробити висновок, що нормалізація даних - це процес перетворення даних у послідовний і порівнянний формат, який може покращити якість даних, їх аналіз та інтеграцію.

Нормалізацію даних можна визначити ще як процес, покликаний сприяти створенню більш зв'язної форми введення даних, по суті, "очищенню" даних. Коли ви нормалізуєте набір даних, ви реорганізуєте його, щоб видалити будь-які неструктуровані або надлишкові дані, щоб забезпечити кращий, більш логічний спосіб зберігання цих даних.

Основною метою нормалізації даних є досягнення стандартизованого формату даних у всій вашій системі. Це дозволяє легше запитувати та аналізувати дані, що може призвести до прийняття кращих бізнес-рішень.

Однак існують різні методи нормалізації, які мають різні переваги та недоліки залежно від даних і цілей.

Переваги:

1. Покращена продуктивність алгоритмів машинного навчання: Нормалізація може допомогти поліпшити продуктивність алгоритмів машинного навчання, масштабуючи вхідні характеристики до загального масштабу. Це може допомогти зменшити вплив викидів і підвищити точність моделі.

2. Краща обробка викидів: Нормалізація може допомогти зменшити вплив викидів, масштабуючи дані до загальної шкали, що може зробити викиди менш впливовими.

3. Покращена інтерпретація результатів: Нормалізація може полегшити інтерпретацію результатів моделі машинного навчання, оскільки вхідні дані будуть в єдиному масштабі.

4. Краще узагальнення: Нормалізація може допомогти поліпшити узагальнення моделі, зменшуючи вплив викидів і роблячи модель менш чутливою до масштабу вхідних даних.

Недоліки:

1. Втрата інформації: Нормалізація може призвести до втрати інформації, якщо початковий масштаб вхідних ознак є важливим.
2. Вплив на пропуски: Нормалізація може ускладнити виявлення пропусків, оскільки вони будуть масштабовані разом з рештою даних.
3. Вплив на інтерпретацію: Нормалізація може ускладнити інтерпретацію результатів моделі машинного навчання, оскільки вхідні дані будуть мати загальну шкалу, яка може не збігатися з початковою шкалою даних.
4. Додаткові обчислювальні витрати: Нормалізація може додати додаткові обчислювальні витрати до процесу інтелектуального аналізу даних, оскільки вимагає додаткового часу для масштабування даних.

Міні мак нормалізація - це проста техніка, яка змінює масштаб значень даних до діапазону від 0 до 1, використовуючи мінімальні та максимальні значення вихідних даних. Цей метод зберігає відносний порядок і відстань між точками даних, але також зменшує дисперсію і збільшує вплив пропусків. Міні мак нормалізація корисна, коли дані мають фіксований діапазон, наприклад, оцінки або відсотки, але вона може спотворити дані, якщо є екстремальні значення або різні шкали.

Нормалізація Z-рахунку - це метод, який стандартизує значення даних шляхом віднімання середнього значення і ділення на стандартне відхилення вихідних даних. Цей метод перетворює дані на нормальний розподіл із середнім значенням 0 і стандартним відхиленням 1, що полегшує порівняння та аналіз різних змінних. Однак нормалізація за z-критерієм також змінює початковий масштаб і діапазон даних, і на них можуть впливати викиди та асиметрія. Нормалізація за Z-критерієм корисна, коли дані мають розподіл Гауса або коли масштаб і діапазон не є важливими, але вона може ввести в оману, якщо дані мають інший розподіл або якщо викиди мають значення.

Нормалізація десяткового масштабування - це метод, який зміщує десяткову крапку значень даних, щоб зменшити їхню величину, використовуючи коефіцієнт 10. Цей метод зберігає відносний порядок і пропорцію точок даних, але також змінює масштаб і діапазон даних. Нормалізація десяткового масштабування

корисна, коли дані мають великий діапазон значень або коли величина не є важливою, але вона може бути проблематичною, якщо дані мають невеликий діапазон значень або якщо масштаб і діапазон мають значення.

Отже, нормалізація даних може мати як переваги, так і недоліки. Вона може покращити продуктивність алгоритмів машинного навчання і полегшити інтерпретацію результатів. Однак вона також може призвести до втрати інформації та ускладнити виявлення викидів. Важливо зважити всі "за" і "проти" нормалізації даних і ретельно оцінити ризики та переваги, перш ніж впроваджувати її.

Процес нормалізації даних може потребувати часу та зусиль, але переваги нормалізації даних значно переважають недоліки. Без нормалізації даних, які ви збираєте з різних джерел, більша частина цих даних не матиме реального значення або призначення для вашої організації.

Під час нормалізації стовпці, атрибути та таблиці бази даних, або зв'язки, організовуються відповідно до набору правил нормальної форми. Ці нормальні форми діють як своєрідна система стримувань і противаги для збереження цілісності в’язків між характеристиками та відношеннями і є тим, що спрямовує процес нормалізації. За допомогою набору правил (які називаються "нормальні форми") процес нормалізації прагне гарантувати, що узгодженість бази даних зберігається незалежно від того, чи змінюються, додаються або знищуються будь-які дані.

Хоча бази даних і системи можуть еволюціонувати, дозволяючи зберігати менше даних, все одно важливо враховувати стандартизований формат для ваших даних, щоб уникнути дублювання, аномалій або надмірності, щоб підвищити загальну цілісність ваших даних. Нормалізація даних розкриває бізнес-потенціал, підвищуючи функціональність і можливості зростання будь-якої організації. З цієї причини нормалізація даних - одна з найкращих речей, які ви можете зробити для свого підприємства сьогодні.

РОЗДІЛ 2. Проектування.

2.1. Визначення основних функціональних вимог до системи.

Функціональні вимоги визначають базову поведінку системи. Якщо функціональні вимоги не виконані, система не працюватиме. Ці вимоги визначають, що система може робити і коли саме вона повинна це робити - функціональність системи.

Функціональні вимоги - це специфікації того, що система повинна робити, а не як вона повинна це робити. Вони визначають можливості, функції та поведінку системи, які відповідають потребам та очікуванням користувачів і зацікавлених сторін. Визначення функціональних вимог є вирішальним кроком у розробці системи, оскільки воно впливає на дизайн, тестування та впровадження системи.

Функціональні вимоги визначають поведінку "якщо/то" і включають розрахунки, введення даних та бізнес-процеси. Цей тип вимог детально описує функції, які дозволяють системі функціонувати так, як було задумано. Не плутайте цей тип вимог з нефункціональними вимогами.

Функціональні вимоги можна вважати загальною категорією, що містить різні інші типи вимог. Ці типи включають в себе

- Вимоги користувача
- Бізнес-вимоги
- Адміністративні функції
- Системні вимоги

Вони пов'язані з функціями продукту, в той час як нефункціональні вимоги стосуються властивостей продукту. Простіше кажучи, функціональні вимоги - це що, а нефункціональні - як. Щоб ефективно виявити, проаналізувати та задокументувати функціональні вимоги існує кілька порад і методів, які можуть допомогти у цьому процесі.

1. Розуміння проблеми

Перш ніж приступити до визначення функціональних вимог, необхідно зрозуміти проблему, яку повинна вирішити система. Це означає визначення

поточної ситуації, цілей і завдань, обсягу і меж, а також ризиків і припущень проекту. Для з'ясування проблеми та її контексту можна використовувати такі методи, як постановка задачі, статут проекту, визначення обсягу робіт та аналіз ризиків.

2. Залучайте зацікавлені сторони

Зацікавлені сторони - це люди, які мають інтерес або вплив на систему, наприклад, користувачі, клієнти, менеджери, спонсори або регулятори. Вони є джерелом функціональних вимог, оскільки виражають свої потреби, вподобання та очікування щодо системи, тому потрібно залучати їх протягом усього процесу, від планування до варіації, щоб переконатися, що ви зафіксували і визначили пріоритетність їхніх вимог. Можна використовувати такі методи, як інтерв'ю, опитування, фокус-групи, семінари та спостереження, щоб отримати від них інформацію та зворотній зв'язок.

3.Моделювання вимог

Моделювання - це процес представлення функціональних вимог у візуальній або текстовій формі, наприклад, у вигляді діаграм, таблиць, графіків або історій. Моделювання допомагає вам організувати, проаналізувати та донести вимоги більш чітко та ефективно. Воно також допомагає виявити прогалини, невідповідності та конфлікти у вимогах. Для моделювання вимог можна використовувати такі методи, як варіанти використання, історії користувачів, сценарії, блок-схеми, діаграми станів та фрейми.

4.Валідація вимог

Валідація - це процес перевірки того, що функціональні вимоги є правильними, повними, послідовними, здійсненними і такими, що можуть бути перевірені. Валідація допомагає вам переконатися, що вимоги відповідають потребам і очікуванням зацікавлених сторін і системи. Вона також допомагає уникнути помилок, змін і переробок на більш пізніх етапах циклу розробки. Для валідації вимог можна використовувати такі методи, як огляди, перевірки, покрокові огляди, створення прототипів і тестування.

5.Документація вимог

Документування - це процес запису функціональних вимог у формальний або неформальний спосіб, наприклад, у вигляді документа, бази даних або інструменту. Документація допомагає вам спілкуватися і ділитися вимогами з командою розробників і зацікавленими сторонами. Вона також допомагає відстежувати та керувати вимогами протягом усього циклу розробки. Для документування вимог можна використовувати такі методи, як специфікація вимог, матриця відстеження вимог, інструмент управління вимогами та контроль версій.

6.Керування вимогами

Управління - це процес контролю та моніторингу функціональних вимог протягом усього циклу розробки. Управління допомагає впоратися зі змінами, проблемами та залежністю, які можуть виникнути у вимогах. Воно також допомагає вам переконатися, що вимоги узгоджені з цілями і завданнями проекту. Для управління вимогами можна використовувати такі методи, як контроль змін, відстеження проблем, управління конфігурацією та звітування про стан.

Правильне визначення вимог - це ключ до успіху будь-якого проекту. Неможливість точно визначити та задокументувати їх неминуче призводить до непорозумінь між зацікавленими сторонами, постійних переглядів та непотрібних затримок. Дослідження показують, що нечіткі або погано задокументовані вимоги можуть збільшити терміни і бюджет проекту на 60%.

Зі зростанням популярності гнучкого підходу до документування деякі команди почали нехтувати документуванням вимог - зрештою, це ж "робоче програмне забезпечення, а не вичерпна документація", чи не так?

На жаль, це поширена помилка, і відмова від належної внутрішньої документації може бути особливо шкідливою, коли мова йде про вимоги.

2.2. Виявлення основних потреб користувачів

Вимоги користувачів, які часто називають потребами користувачів, описують те, що користувач робить з системою, наприклад, які дії користувачі

повинні вміти виконувати. Вимоги користувача, як правило, документуються в документі про вимоги користувача (URD) з використанням описового тексту. Вимоги користувача, як правило, підписуються користувачем і використовуються як основний вхідний матеріал для створення системних вимог.

Не існує загальноприйнятого визначення понять "вимога користувача" і "потреба користувача", в результаті чого ці терміни використовуються як взаємозамінні і мають різні визначення в залежності від уподобань організації.

Найчастіше слово "потреби" використовується для опису бажань і очікувань користувача до того, як буде визначено обсяг проекту і вони будуть пріори тезовані в інтегрований набір вимог. Після того, як основні результати проекту будуть узгоджені, прийняті потреби користувача стануть вимогами користувача.

Простіший спосіб зрозуміти цей тип функціональних вимог - розглядати аспект користувача як атрибут стандартної функціональної вимоги. Вимога користувача - це функціональна вимога, в якій необхідність, пріоритет і важливість вимоги визначаються тим, що кінцевий користувач бажає або очікує цього. Якщо вимога користувача не буде виконана, система не буде працювати для користувача. Якщо система або продукт не працює для користувача, то чи працює вона взагалі?

Вимоги користувача можуть бути менш формальними, але після доопрацювання вони повинні працювати так само, як і стандартні функціональні вимоги.

Аналіз зацікавлених сторін необхідний для того, щоб визначити, на кого вплине система і як залучити людей, на яких вона вплине, щоб отримати їхні вимоги користувачів.

Важливим і складним етапом проектування програмного продукту є визначення того, що користувач насправді хоче, щоб він робив. Це пов'язано з тим, що користувач часто не в змозі передати всі свої потреби і бажання, а інформація, яку він надає, може бути неповною, неточною і суперечливою. Відповідальність за повне розуміння того, чого хоче клієнт, лягає на бізнес-

аналітика. Ось чому вимоги користувачів зазвичай розглядаються окремо від системних вимог. Бізнес-аналітик ретельно аналізує вимоги користувачів і ретельно конструює та документує набір високоякісних системних вимог, гарантуючи, що вимоги відповідають певним характеристикам якості.

Специфікація користувацьких вимог описує бізнес-потреби щодо того, що користувачі вимагають від системи. Специфікації користувацьких вимог складаються на початку процесу валідації, як правило, до створення системи. Вони пишуться власником системи і кінцевими користувачами за участю відділу забезпечення якості. Вимоги, викладені в URS, зазвичай перевіряються під час кваліфікаційного тестування продуктивності або тестування прийнятності для користувача. Специфікація користувацьких вимог не є технічним документом; читачі, які мають лише загальні знання про систему, повинні бути в змозі зрозуміти вимоги, викладені в URS.

Як правило, URS є документом планування, який створюється, коли компанія планує придбати систему і намагається визначити конкретні потреби. Коли система вже створена або придбана, або для менш складних систем, специфікація вимог користувача може бути об'єднана з документом функціональних вимог.

Вимоги користувачів

Хороші вимоги є об'єктивними та перевіреними. Наприклад:

- Екран A приймає інформацію про виробництво, включаючи партію, номер продукту і дату.
- Система B створює Зведений звіт лабораторії.
- Двадцять користувачів можуть одночасно використовувати Систему C без помітних затримок у роботі системи.
- Система D може друкувати дані з екрану на принтер.
- Система E відповідає вимогам 21 CFR 11.

Вимоги повинні включати в себе:

- Вступ - включаючи сферу застосування системи, ключові цілі проекту та відповідні регуляторні питання

- Програмні вимоги - функції та робочий процес, які повинна виконувати система
- Вимоги до даних - тип інформації, яку система повинна бути здатна обробляти
- Вимоги до життєвого циклу - включаючи те, як система буде підтримуватися і як користувачі будуть навчатися.

Вимоги зазвичай надаються з унікальним ідентифікатором, таким як ID#, щоб полегшити відстеження протягом усього процесу валідації.

Специфікація користувацьких вимог (URS) - це плановий документ, який визначає, що користувач хоче, щоб продукт або система робили. Власники бізнесу, менеджери та кінцеві користувачі зазвичай впроваджують URS, коли купують обладнання, наприклад, програмне забезпечення. Команда забезпечення якості також робить свій внесок у написання URS. Зазвичай вони аналізують вимоги в URS під час прийняття користувачем або тестування на відповідність експлуатаційним характеристикам. Зазвичай це простий документ, в якому не використовується багато технічної інформації, щоб відобразити точку зору кінцевого користувача.

Специфікації користувацьких вимог повинні бути підписані власником системи, ключовими кінцевими користувачами та відділом якості. Після затвердження URS (Специфікація вимог користувача) зберігається відповідно до практики зберігання документів, прийнятої у вашій організації.

Коли система створюється, специфікація вимог користувача є цінним інструментом для забезпечення того, що система буде робити те, що потрібно користувачам. При ретроспективній валідації, коли валідації підлягає існуюча система, вимоги користувача еквівалентні функціональним вимогам: ці два документи можуть бути об'єднані в один документ.

Альтернативні назви та скорочення документів:

Іноді використовуються наступні терміни або абревіатури: Специфікація вимог користувача, специфікація вимог користувача, вимоги користувача, специфікації користувача, URS, UR, US.

Ресурси документів з валідації:

- Генеральні плани валідації (VMP)
- Плани валідації (VP)
- Оцінка ризиків (RA)
- Специфікації вимог користувача (User Specs, URS)
- Функціональні вимоги (Functional Requirement Specifications, Functional Specs, FRS, FS)
- Специфікація дизайну (DS)
- План випробувань / протокол випробувань
- Кваліфікація установки (IQ)
- Кваліфікація експлуатації (OQ)
- Кваліфікація продуктивності (PQ)
- Матриця відстеження вимог (Trace Matrix, RTM, TM)
- Відхилення протоколу випробувань
- Підсумковий звіт про валідацію (Validation Report, Summary Report, VR, SR)
- Контроль змін для валідованих систем
- Термінологія валідації
- Валідація FAQ (Часті запитання про валідацію)
- Валідація комп'ютерних систем
- Аудит та оцінювання

Документ URS важливий з наступних причин:

Збирає інформацію про користувача: URS допомагає продавцю чи постачальнику зрозуміти вимоги клієнта. Інформація, що міститься в URS, спрямовує їх на розробку системи спеціалізованого обладнання, яке відповідає потребам клієнта.

Визначає потреби: URS допомагає організації визначити свої вимоги до системи. Розробники ретельно аналізують їхні потреби і можуть співпрацювати з різними відділами, щоб визначити та адекватно повідомити про основні вимоги.

Заощаджує час і ресурси: Документ URS допомагає постачальнику та організації-клієнту заощадити час і ресурси, мінімізуючи кількість змін у системі. Повідомляючи про свої потреби, організації допомагають постачальникам постачати системи або обладнання, які відповідають стандартам.

Дотримання нормативних вимог: Деякі компанії чи галузі вимагають від персоналу написання документів URS перед придбанням обладнання. Окрім відповідності вимогам, документ URS є основою для валідації, наприклад, для отримання кваліфікації з експлуатації та інсталяції.

Допомагає вирішувати суперечки: Оскільки URS діє як точка відліку, він може допомогти в управлінні будь-якими суперечками, що виникають. Це важливо для збереження професійних відносин між організаціями.

Що потрібно включити в URS. Інформація в URS варіюється залежно від проекту. Складний проект вимагає більше специфікацій, ніж простий. Нижче наведені деякі загальні характеристики, які слід включити в URS:

1.Автори

Вкажіть авторів документа, щоб допомогти його ідентифікувати. Вкажіть ім'я, посаду, назву організації та підписи авторів. Ви також можете додати контактну інформацію, наприклад, електронну пошту та номери телефонів, щоб допомогти постачальникам або розробникам зв'язатися з відповідною особою із запитами.

2.Мета

Розділ "Мета" створює перше враження та надає базовий огляд вимог замовника. Він описує все, що клієнт хоче отримати від постачальника. Інформація в розділі "Мета" включає сферу застосування URS, потужність або продуктивність продукту та відповідність нормативним вимогам, якщо такі є.

Приклад: Компанія має на меті створити централізований додаток для смартфонів для своїх численних ігор. Вона хоче, щоб додаток дозволяв користувачам створювати профілі, завантажувати фотографії та взаємодіяти з іншими користувачами з будь-якої частини світу. Компанія також хоче, щоб додаток мав систему оцінювання та відгуків.

3.Вимоги

У розділі вимог описуються конкретні потреби клієнта в системі. Якщо специфікацій небагато, дехто пише цю частину документа в описовій формі. Інші перераховують вимоги або створюють категорії, щоб вичерпно пояснити та ефективно передати найважливіші деталі. Наведемо деякі з типів вимог, які можна включити в URS-документ: функціональний, оперативний, дані, технічний, інтерфейс, безпека, доступність, екологічний, обслуговування, регуляторний, життєвий цикл, міграція електронних даних, необхідні обмеження.

4.Супровідні документи

Додайте супровідні документи, які можуть знадобитися користувачам. Деякі приклади документації, яку слід додати до URS, включають електричну схему з інструкцією з експлуатації, схеми трубопроводів і приладів та інші документи, що мають відношення до робочого механізму системи. Розгляньте можливість включення зображень або ескізів концепцій, пов'язаних з проектом, прикладів програмного забезпечення і систем, які вже відповідають деяким вимогам організації, а також посилань на веб-сайти, які надають додаткові моделі для ознайомлення.

5.Встановлення та введення в експлуатацію

У цьому розділі детально описані процедури встановлення та експлуатації обладнання або системи. Він визначає, за що відповідає кожна сторона під час встановлення. Наприклад, роз'яснюється роль виробника та кроки, які слід зробити, якщо система не запускається.

6.Поставка та навчання

Цей розділ описує умови доставки продукту. Наприклад, визначається, чи клієнт забирає продукт самостійно, чи виробник доставляє його на територію клієнта. Розділ "Навчання" висвітлює процедуру навчання персоналу роботі з системою після встановлення та введення в експлуатацію.

7.Глосарій та іменний покажчик

Якщо документ містить технічний або нетехнічний жаргон, скорочення або абревіатури, важливо пояснити їх у розділі глосарію та покажчика. Це допоможе

кінцевому користувачеві краще зрозуміти УРП. Подумайте про те, щоб включити індекс, список, який організовує зміст відповідно до номерів сторінок, для довгих документів URS.

8.Комерційні умови

Комерційні умови описують фінансові деталі проекту. Вони також обговорюють, як постачальник і замовник розподіляють ризики. Наприклад, вони можуть розмежовувати ціни на гарантійні зобов'язання та банківську інформацію.

9.Розділ затвердження

Розділ затвердження є одним з останніх розділів URS. Він включає всі підписи від відповідних відділів. Затвердження надходять від ключових відділів, таких як виробничий інжиніринг та забезпечення якості, а також від керівника об'єкта, операції або організації.

2.3. Формування комплексної структури системи

Однією з необхідних умов вивчення інформаційної структури є визначення її меж і, зокрема, надання чіткого набору критеріїв, які відрізняють інформаційну структуру від інших функціональних доменів мови. Інше, але пов'язане з цим питання полягає в тому, як визначити, яка пара форма-функція в мові є частиною інформаційної структури, а яка не є частиною інформаційної структури.

Одне з основних питань щодо інформаційної структури полягає в тому, що є частиною інформаційної структури, а що ні. Якщо погодитися з Ламбрехтом (Lambrecht, 1994) та Ван Валіном (Van Valin, 2016), який включив підхід Ламбрехта до інформаційної структури у свою "Рольову та еталонну граматику", то кожне висловлювання має певні елементи інформаційної структури. Як зазначає Ван Валін, Ламбрехт розрізняє фокус присудка, фокус речення і вузький фокус. Прикладом фокусу на присудку є наступна відповідь на питання «Що сталося з вашим автомобілем?»:

«Моя машина зламалася.» На думку Ван Валіна, предикативний фокус відповідає традиційному розрізненню теми і коментаря.

Фокус речення, на думку Ван Валіна, ілюструється наступною відповіддю на питання "Що сталося?": «Моя машина зламалася.»

Прикладом вузького фокусу є відповідь на питання "Ваш мотоцикл зламався?". «Ні, зламався мій автомобіль» або «Це моя машина зламалася».

Таким чином, підхід до інформаційної структури, який обстоює Ван Валін, передбачає, що кожне висловлювання містить певну функцію, яка належить до сфери інформаційної структури. Емпіричні факти в різних мовах не підтверджують цього припущення. Наприклад, спонукальне речення може мати тему, але не обов'язково.

Типові компоненти та структура інформаційних систем

Комп’ютерні інформаційні системи інформаційних систем належать до категорії складних систем , у якій взаємодіють багато різноманітних компонентів. Досі не існує загальноприйнятої повної класифікацій елементів інформаційних систем. Найчастіше в структурах іс виділяються компоненти, які вважаються неподільними. Компонент (підсистема) інтелектуальної системи це компонент інтелектуальної системи виокремлений за зазначеною ознакою або сукупністю ознак і розглядається як самостійне ціле.

Компоненти поділяються на:

1. забезпечувальні та функціональні за своїм призначенням забезпечувальні компоненти а саме види забезпечення включають інформаційних систем;
2. технічне - це сукупність усіх технічних засобів які використовуються під час роботи системи;
3. програмне забезпечення це набір програм на носіях даних і пригримни документів призначених для від лагодження функціонування та перевірки робото здатності інформаційних систем;
4. математичне - це сукупність математичних методів моделей і алгоритмів розв’язування задач, які застосовуються в інформаційних системах. Інформаційних систем моделі та алгоритми, які розвиваються в інструменти що входять до цього типу програмного забезпечення;
5. інформаційні ресурси як предмет праці, методи та засоби ведення інформаційної бази; організація — сукупність документів, що описують

технологію функціонування інформаційних систем, а також методи, за допомогою яких користувачі вибирають ці технології для досягнення конкретних результатів під час функціонування;

6. інформаційне забезпечення включає документи, стандарти та рішення щодо обсягу, розміщення та виду інформації, яка використовується в інформаційних системах під час їх функціонування;

7. лінгвістичне — це сукупність методів і правил формалізації природного мовлення, які використовують користувачі та експлуатаційний персонал ІС під час використання комплексу технічних засобів для спілкування;

8. право - це система правових правил, що регулюють правові відносини, що виникають під час роботи інформаційних систем, а також юридичний статус результатів такої роботи;

9. ергономічна — це сукупність інструментів і методів, які створюють найсприятливіші умови для праці людини в інтелектуальних системах, а також умови для взаємодії людини та електронної машини. Для підвищення продуктивності, надійності та безпеки функціонування систем «людина — машина» ергономічні вимоги визначаються характеристиками людини та середовища.

2.4. Висновки до розділу

Зробивши висновок з усього матеріалу, можемо зазначити, що визначаючи вимоги, багато людей плутаються, намагаючись розмежувати бізнес-вимоги, вимоги користувачів та вимоги до програмного забезпечення. Всі три типи вимог відрізняються і служать різним цілям.

Вимоги користувачів - це очікування і потреби кінцевих користувачів і зацікавлених сторін проекту з розробки системи. Вони визначають, що повинна робити система, як вона повинна працювати і які переваги вона повинна надавати. Задоволення вимог користувачів є важливим для створення успішної системи, яка відповідає цілям проекту і задовольняє потреби користувачів. Однак вимоги користувачів можуть бути складними, розпливчастими, мінливими і

суперечливими, що ускладнює їх виявлення, аналіз, уточнення, перевірку та управління ними.

1.З'ясування потреб користувачів

Першим кроком на шляху до задоволення вимог користувачів є їх з'ясування у користувачів та зацікавлених сторін. З'ясування - це процес збору інформації про проблемну область, потреби користувачів, цілі системи, а також обмеження та припущення.

2.Проаналізуйте вимоги користувачів

Другим кроком у задоволенні вимог користувачів є їх аналіз на предмет ясності, повноти, узгодженості, здійсненності та пріоритетності. Аналіз - це процес вивчення, уточнення, організації та моделювання вимог користувача, щоб переконатися, що вони чітко визначені, зрозумілі та піддаються тестуванню. У процесі аналізу учасники повинні підтвердити своє розуміння та згоду.

3.Визначення нефункціональних вимог

Третій крок - визначення нефункціональних вимог до системи. Це атрибути якості та обмеження, які впливають на продуктивність, зручність використання, надійність, безпеку та ремонтопридатність системи.

4.Визначення вимог до інтерфейсу користувача

Четвертим кроком є визначення вимог до інтерфейсу користувача системи. Це аспекти системи, які стосуються зовнішнього вигляду, компонування, навігації та взаємодії інтерфейсу користувача.

5.Перевірка вимог користувача

П'ятий крок - перевірка вимог користувача до системи. Це означає перевірку того, що вимоги користувача є правильними, повними, послідовними, здійсненними і такими, що можуть бути протестовані. Для перевірки користувацьких вимог можна використовувати різні методи, такі як огляди, перевірки, покрокові інструкції або тестування.

6.Керування вимогами користувачів

Останнім кроком є управління вимогами користувачів до системи. Це означає контроль за змінами, оновленнями та переглядами вимог користувачів

протягом усього проекту системи. Ви можете використовувати різні інструменти для управління вимогами користувачів, такі як депозитарії, матриці відстеження або системи управління змінами.

Бізнес-вимоги описують, чому організація береться за проект. Вони вказують на переваги, які організація-розробник або її клієнти очікують отримати від продукту. Бізнес-вимоги можуть бути викладені в декількох документах, таких як статут проекту, бізнес-кейс, бачення та опис проекту.

Бізнес-вимоги ставлять власника проекту, зацікавлені сторони та команду проекту на один щабель. Але ви не можете створити програмне забезпечення на основі такої високо рівневі інформації.

Вимоги користувачів, які часто називають потребами користувачів, описують те, що користувач робить з системою, наприклад, які дії користувачі повинні вміти виконувати.

Вимоги користувача, як правило, документуються в документі про вимоги користувача (URD) з використанням описового тексту. Вимоги користувача, як правило, підписуються користувачем і використовуються як первинна інформація для створення системних вимог.

Системні вимоги - це будівельні блоки, з яких розробники будують систему. Це традиційні твердження "повинен", які описують, що система "повинна робити". Системні вимоги поділяються на функціональні та додаткові.

Функціональні вимоги визначають те, що потрібно користувачеві для виконання своєї роботи. Наприклад, система може бути необхідна для введення та друку кошторисів витрат; це функціональна вимога. Додаткові або нефункціональні вимоги визначають всі інші вимоги, які не охоплені функціональними вимогами. Я вважаю за краще використовувати термін "додаткові вимоги" замість "нефункціональні вимоги"; хто хоче, щоб його називали нефункціональним? Додаткові вимоги іноді називають вимогами до якості обслуговування. План реалізації функціональних вимог детально описується в проекті системи.

Для досягнення оптимальних результатів набір вимог до продукту повинен бути перевірений на відповідність характеристикам добре сформованих вимог (наприклад, необхідних, однозначних, повних, узгоджених, правильних, здійсненних, перевірених) і підтверджений, що вони відображають наміри потреб, з яких вони були перетворені.

РОЗДІЛ 3 Програмна реалізація.

3.1. Вибір технології розробки додатку та бази даних.

C# (вимовляється як See Sharp) - це об'єктно-орієнтована мова програмування загального призначення, розроблена компанією Microsoft. Назва натхненна символом "диез", який використовується в нотному записі для підвищення ноти на півтону. Якщо придивитися уважніше, знак # складається з чотирьох плюсів, розміщених у сітці два на два. Це вказує на те, що C# є інкрементом C++. Випущена у 2000 році, C# залишається основною технологією для створення десктопних додатків на Windows.

Додатки, написані на C#, використовують середовище виконання .NET, бібліотеки класів і, по суті, сам фреймворк .NET, тому обидві технології часто розглядаються як нероздільні. На сьогоднішній день .NET є платформою та програмним середовищем для крос-платформної розробки.

Перша ітерація C# 1.0, що постачалася з Visual Studio та .NET у 2002 році, була багато в чому схожа на Java, що також зазначено в документації. "Якщо повернутися назад і подивитися, то C# версії 1.0, випущена з Visual Studio .NET 2002, була дуже схожа на Java.

У той час схожість на Java означала, що вона досягла тих ранніх цілей проектування". У той же час, C# не вистачало такої функціональності, як асинхронне програмування, підтримка багатопотоковості та фільтрація винятків, щоб можна було порівняти її безпосередньо з Java. C#, як і .NET, залишалася технологією лише для Windows.

Тим не менш, C# 9.0 є останньою версією мови, яка використовується разом з .NET 5 та .NET Core.

Отже, давайте поговоримо про екосистему C#, яка робить її універсальним рішенням для загальних цілей програмування.

Платформа .NET. C# невіддільна від платформи .NET. Вона компілює та запускає програми за допомогою

- Common Language Runtime (CLR). Остання версія платформи .NET - .NET 5.

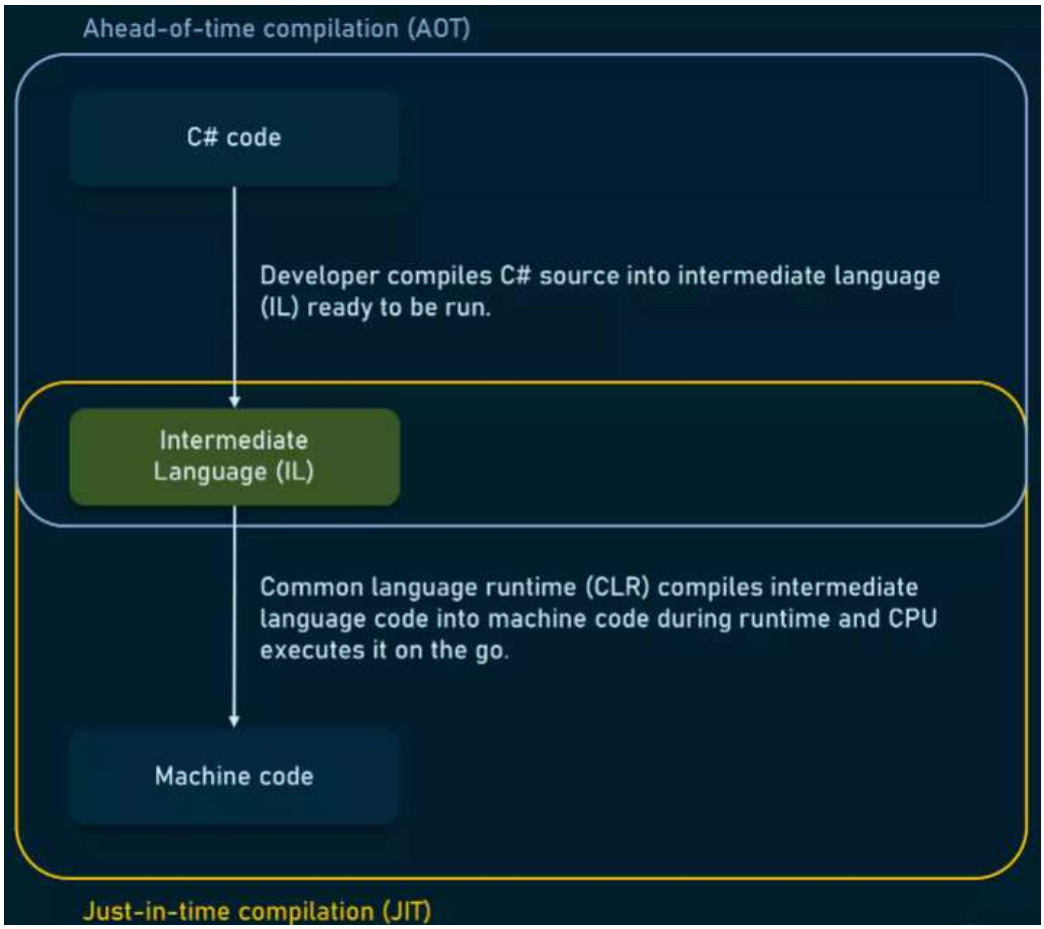
- Фреймворк .NET Core. Раніше Core був крос-платформною версією старої версії .NET. Він надає бібліотеки та елементи багаторазового використання для прискорення розробки додатків. Оскільки .NET 5 вже має всі крос-платформні можливості Core, .NET 5 можна вважати єдиним новим продуктом .NET.

- Visual Studio. Це нативне середовище розробки для C#, яке дозволяє розробникам завантажувати та встановлювати мову, писати код, налагоджувати та запускати/компілювати його.

- .NET SDK (читається як C# SDK). Набір інструментів та бібліотек створений для розробки C# додатків або написання нових бібліотек.

Компіляція та виконання C#

C# - це мова, що компілюється. Щоб запустити програму, зазвичай розробник компілює вихідний код C# у проміжну мову (IL). Таким чином, він може виконуватися на різних цільових системах. Під час виконання код на IL далі компілюється в машинний код поточної цільової системи, і процесор виконує його на ходу.



Це стандартний сценарій, і можуть бути варіації залежно від цільової платформи та типу програми

Компіляція Just-in-time (JIT) від IL забезпечує універсальність C#-додатків, які можна запускати скрізь, від Windows до PlayStation. Але це також призводить до певного зниження продуктивності порівняно, скажімо, з C++, який компілюється в машинний код цільової системи одразу, перед виконанням. Але не впадайте у відчай, середовища виконання, що використовуються з C#, підтримують ряд випадків, коли IL компілюється перед виконанням, щоб забезпечити найкращу можливу продуктивність. Наприклад, додатки для iOS вимагають попередньої компіляції.

Існує два основних середовища виконання для C#.

- .NET Common Language Runtime - використовується для запуску і компіляції веб-додатків з версіями .NET 5 і Core. Підтримує Windows, Linux, MacOS, Android, iOS - практично всі основні цільові платформи.

- Mono Runtime - доповнення, яке спочатку постачалося з Xamarin для роботи над мобільними додатками (iOS/Android), Linux та macOS. Сьогодні Mono використовується для додатків, які мають критичні вимоги до продуктивності, таких як ігри або мобільні додатки. Він також підтримує всі основні операційні системи, а також деякі ігрові консолі.

Тепер давайте розглянемо переваги та недоліки програмування на C#.

Переваги програмування мовою C#

C# вважається чудовим вибором для настільних додатків Windows, корпоративних рішень і навіть для розробки ігор, оскільки ігровий рушій Unity побудований на C#. Отже, що робить C# перевагою як основної мови програмування?

Об'єктно-орієнтоване програмування

Від самого початку C# базувалася на принципах об'єктно-орієнтованого програмування (ООП). Ця концепція кодування передбачає, що ви можете визначити тип і структуру даних, застосувати до них набір стандартних функцій. ООП збирає дані в об'єкти, що полегшує розбиття програми на менші частини, які швидше створювати, керувати та комбінувати.

Використовуючи ООП, об'єктами можна керувати без взаємодії з їхніми внутрішніми атрибутами, описуючи поведінку об'єктів через оголошення класів. ООП мови створюють програми, які легше тестувати і читати, дозволяють реагувати на будь-які проблеми, що виникають, і в цілому означають більш ощадливий підхід до написання коду.

Мова високого рівня з можливостями доступу до пам'яті

Мова C# вважається мовою високого рівня, оскільки її синтаксис нагадує людську мову. Іншими словами, вона має високий рівень абстракції від машинного коду, тому код, написаний на C#, потрібно компілювати, щоб апаратне забезпечення розуміло його команди.

Високо рівневі мови корисні для розробників, оскільки вони мають простіший синтаксис для розуміння та управління, на відміну від низько рівневі мов, таких як C.

Але це питання є дещо суперечливим. Однією з важливих особливостей таких мов, як C, є те, що вони можуть отримувати доступ до пам'яті безпосередньо, використовуючи специфічні типи команд, які називаються вказівниками. Хоча C# є мовою значно вищого рівня, розробникам все ще доступний обмежений набір функцій вказівників.

C# є безпечною за типом, що означає, що змінна не може змінювати свій тип у коді. Наприклад, якщо ви оголосили змінну GoodDay як ціле число, ви можете присвоювати їй тільки точні числові значення, а рядкові значення, такі як субота або неділя, є неприйнятними.

Безпека типів гарантує, що змінна поводитиметься передбачувано, і будь-які операції над нею будуть можливі лише за умови відповідності типу. Ніяких сюрпризів! Код, що містить змінну GoodDay, буде звертатися тільки до тих комірок пам'яті, які призначені для цілих чисел. Такий підхід робить вихідний код загалом менш схильним до помилок.

За замовчуванням C# виконує перевірку типів під час компіляції, що називається статичною типізацією. В результаті, типові помилки виявляються якомога раніше, до того, як вони потраплять у середовище виконання. Однак, починаючи з версії 4, мова також підтримує динамічну типізацію. Ви можете створювати об'єкти, позначені як динамічні, що дозволяє їм обходити перевірку типів під час компіляції. Помилка, якщо вона виникне, буде виявлена під час виконання.

Динамічна опція надає розробникам більшої гнучкості та спрощує взаємодію з фрагментами коду, що надходять з інших середовищ виконання. Наприклад, ваш об'єкт може отримати своє значення з динамічної мови, такої як IronPython.

Де використовується C#: найпоширеніші програми

Тепер, коли ми обговорили основні переваги та недоліки, залишилося відповісти на останнє питання: коли має сенс обрати C#? Хоча мову відносять до універсальних, є кілька сфер, де вона найбільш доречна.

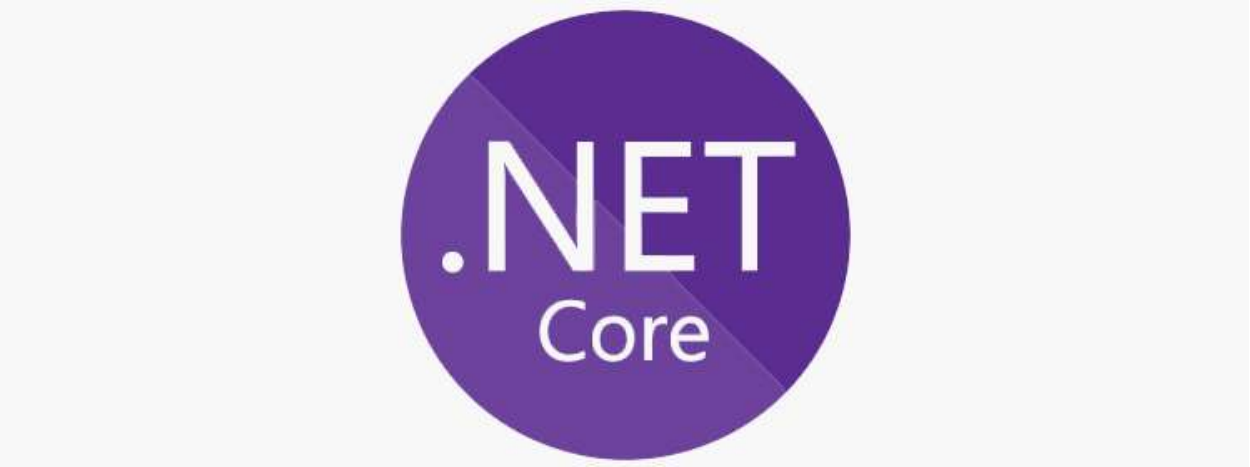
- Десктопні додатки Windows. C# є стандартним вибором для Windows-додатків завдяки вбудованій підтримці фрейм ворків .NET. Вона надає безліч бібліотек, компонентів, бібліотек класів інтерфейсу користувача та інших ресурсів, які пришвидшують розробку.
- Веб-сервіси та додатки. За допомогою .NET Core також можна розробляти надійні веб-сервіси, використовуючи ті ж ресурси платформи .NET.
- Додатки для Linux і macOS. Використовуючи середовище виконання Mono, одні й ті ж додатки можна оптимізувати для систем на базі Unix і пристроїв на базі macOS. У багатьох випадках Mono демонструє результати продуктивності, порівнянні з середовищем виконання .NET. Це означає, що немає суттєвої різниці в роботі C#-додатків на різних платформах.
- Мобільна розробка стає можливою завдяки Xamarin, який використовує C#. Xamarin - це крос платформний фрейм ворку, який обертає компоненти та бібліотеки у шар .Net для створення додатків для Android та iOS. Це дозволяє розробникам повторно використовувати до 90 відсотків коду на двох основних мобільних платформах.
- Розробка ігор.

Для реалізації проектів на C# існують різні механізмі та підходи написання як малих, так і великих бізнес проектів, основними та найпоширенішими з яких є .NET Framework та .NET Core.

Що таке .NET Core?

ASP.NET, також відомий як .NET (вимовляється як дот-нет) - це безкоштовна високопродуктивна платформа з відкритим вихідним кодом, яка підтримується корпорацією Майкрософт.

Вона пропонує крос-платформну основу для створення сучасних, підключених до Інтернету, хмарних додатків, які можуть працювати на операційних системах Mac OS, Linux та Windows. Крім того, .Net Core забезпечує поліпшення стабільності та чудову продуктивність порівняно з Mono, яка витягується зі спільної крос-платформної кодової бази та підтримується активною, чуйною та добре забезпеченою ресурсами командою розробників.



.Net Core кодується з нуля, що робить його швидким, легким та модульним фрейм ворком. Це дозволяє нам пережити захоплюючий період розробки веб- та серверних додатків на .Net, різновид .Net, якого ми не бачили раніше. Крім того, цей додаток прискорює виконання, його легко підтримувати і, крім того, він зменшує обсяг пам'яті. Завдяки цьому його гнучкість підвищує цінність поточної частки .Net, а також робить його привабливим для широкого і високопродуктивного середовища, яке раніше не розглядало .Net як варіант.

Особливості .NET Core:

- Користуватися інструментами розробки для Mac OS, Windows та Linux на свій вибір.
- Розробляти веб-додатки та сервіси, Інтернет речей (IoT) та мобільні бескеди.
- Впроваджувати їх у хмарі або локально.
- Запускати та виконувати на .Net Core.

ASP.NET Framework (вимовляється як дот-нет) - це середовище розробки, яке використовується для створення та запуску різних програмних додатків у Windows. Цей фреймворк складається з інструментів, мов програмування та бібліотек, які допомагають у створенні додатків. .Net Framework спроектований, розроблений і керований компанією Microsoft.

Його перша бета-версія (1.0) була випущена в 2002 році. .Net Framework має різні додатки, які дозволяють запускати код .Net на багатьох інших платформах, або ми можемо сказати, що це крос-платформне програмне

забезпечення, яке працює на Windows, Linux, Mac OS, Android, iOS та багатьох інших.

Що знову ж таки дає перевагу над іншим програмним забезпеченням платформи Mono. Він також використовується для розробки додатків для Windows, телефонів або Інтернету. Простими словами можна сказати, що це свого роду віртуальна машина для компіляції та виконання кодів, написаних різними мовами, такими як C#, VB.Net або іншими.

.Net Framework поділяється на дві частини, або можна сказати, що він має дві основні частини, а саме: Common Language Runtime (CLR) та бібліотека класів Framework (FCL). Common Language Runtime (CLR) - це середовище виконання, в якому виконуються програми, написані на .Net. Бібліотека класів Framework Class Library (FCL) складається з великої кількості бібліотек класів, які містить фреймворк .Net. Таким чином, можна сказати, що CLR та FCL відіграють важливу та основну роль у фреймворку. Такі мови, як C#, F#, Perl, COBOL, ML, VB.NET, Python, Oberon, Cobra, Pascal, ADA, Eiffel підтримуються .Net Framework та багато інших.

Було сказано, що .Net Framework підтримує більше 60 мов програмування, з яких 11 мов програмування були розроблені і спроектовані Microsoft, інші мови, які не були розроблені Microsoft, поки що підтримуються фрейм ворком.

Ми можемо використовувати .Net Framework для розробки додатків на основі форм або веб-додатків, а також веб-сервісів. Він відповідає галузевим стандартам, а також надає багато функціональних можливостей.

Різниця між .NET Core та .NET Framework

Наразі ми маємо певні знання про .Net core та .Net framework, тож давайте обговоримо деякі ключові відмінності між цими платформами.

Перш за все, ми знаємо, що .Net core - це безкоштовна платформа розробки з відкритим вихідним кодом, спроектована і розроблена компанією Microsoft, яка використовується для розробки хмарних програмних додатків загального призначення, які є крос-платформними і можуть виконуватися на Mac OS, Windows і Linux.

Фреймворк .Net - це платформа розробки для кодування та виконання додатків на Windows. Цей фрейм ворки складається з різних деталей, таких як інструменти розробника, мови програмування та бібліотеки для розробки веб та додатків, він також має макет основних вимог до розробки додатків, таких як підключення до бази даних, інтерфейс користувача, сервіси тощо.

Звідси можна сказати, що ядро .Net - це платформа, але не повноцінний фрейм ворки. Або більш простими словами можна сказати, що .Net Core - це лише підмножина .Net Framework. При цьому .Net Core - це остання версія .Net Framework, яка є відкритим вихідним кодом і крос-платформою, розробленою для сучасних додатків.

Отже, тепер ми маємо гарне уявлення та розуміння .Net Core та .Net Framework. Ядро .Net - це лише підмножина фреймворку .Net. Ядро .Net - це також найновіша версія фреймворку .Net.

Фреймворк .Net - це безкоштовна платформа для розробки з відкритим вихідним кодом (доступно декілька пакетів). Ядро .Net є крос-платформою, яка може бути використана для створення, проектування та розробки різних додатків для Windows, Mac OS та Linux.

Він також може бути використаний при проектуванні, кодуванні та виконанні ряду додатків або програмного забезпечення, які можуть включати настільні, мобільні, машинне навчання, онлайн, мікро сервіси, ігри, IoT тощо.

Як ми також знаємо, створюючи будь-який проект з нуля, ми будемо використовувати ядро .Net, яке також надає нам легкий інтерфейс командної мови CLI (Command Language Interface), якому більшість розробників віддають

перевагу перед IDE (Integrated Development Environment). Фрейм ворки .Net - це програмне забезпечення, в якому додатки розробляються і виконуються тільки на базі Windows.

Також, як ми знаємо, ядро .Net підтримує мікро сервіси, в результаті чого ми можемо розробляти програмне забезпечення або додатки на основі мікро сервісів, в той час як ці сервіси не підтримуються фрейм ворком .Net, тому в ньому відсутня така розробка. З точки зору безпеки ми можемо повністю довіряти .Net Framework, який надає нам послуги безпеки, на відміну від ядра .Net, яке цього не робить.

Використовуючи платформу .NET для написання проектів різного рівня складності також не варто забувати про використання певних стандартів та шаблонів, або як їх прийнято називати в програмування «патери».

Основними патерами на платформні .NET є MVC (Model, view, control) MVVM (Model, view, view model).

Розробники програмного забезпечення часто використовують ці патери та інструменти, які допомагають їм у розробці та тестуванні веб-сайтів і додатків.

Основна відмінність між MVVM та MVC полягає у їх використанні та моделях розробки. Хоча вони в чомусь схожі, оскільки обидва допомагають розробникам розробляти та тестувати функціонал, між ними є деякі ключові відмінності.

MVVM розділяє різні компоненти процесу розробки на три категорії: модель, подання та ViewModel. Зазвичай це стосується розмітки коду або графічних інтерфейсів користувача (GUI). MVC, або модель-вид-управління - це спосіб, за допомогою якого розробники розділяють програми на ці три компоненти. Це допомагає відокремити бізнес-вимоги та правила від того, як користувачі працюють з додатком.

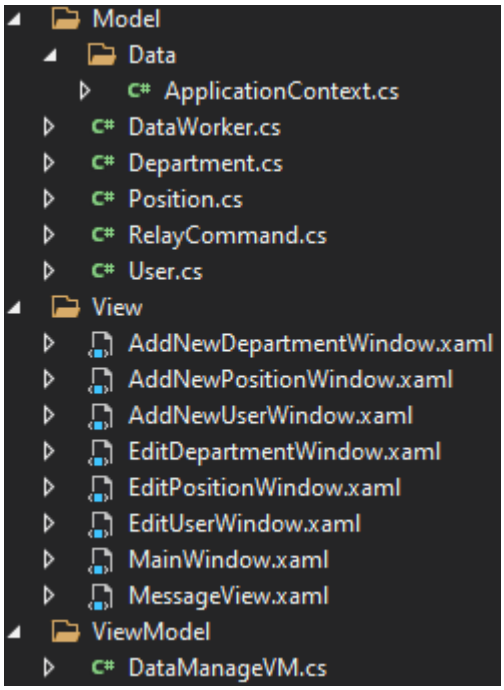
Шаблон MVC полягає в наступному:

- Користувач взаємодіє з певним представленням (view).
- Представлення (view) зв'язується з контролером (controller), щоб викликати подію.

- Контролер (controller) оновлює модель (model).
- Модель (model) надсилає повідомлення про те, що представлення (view) змінилося.
- Представлення (view) витягує дані з моделі (model) та оновлює себе.

У MVVM користувач взаємодіє безпосередньо з представленням (view), яке працює з моделлю представлення (view model). Якщо є якісь зміни, то вони відбуваються безпосередньо між представленням-моделлю (view model) та самою моделлю (model).

Перевагою партерна MVVM над MVC є більш чітко розгалужена та «чиста» структура проекту, з якою програмісту зручно та зрозуміло працювати, швидко орієнтуючись в програмному коді.



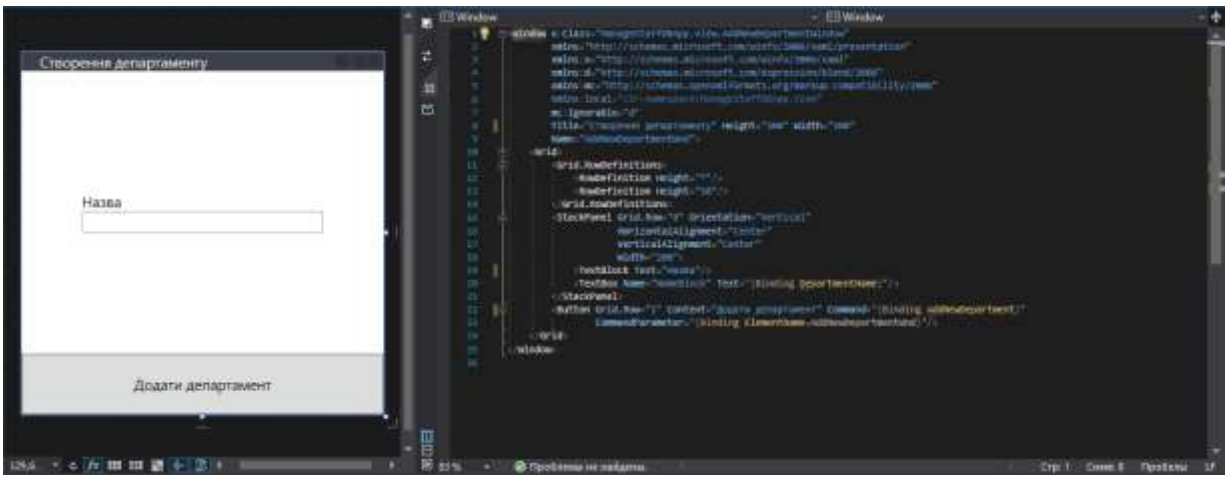
Структура проекту, поділена на теки за патером MVVM

Так, зважаючи на зображення, наведене вище, інженер-розробник, що буде працювати з вашим проектом через деякий час після релізу з певних причин, основними з яких можуть бути додавання розширеного функціоналу або зміна логіки обробки даних тощо, буде розуміти підхід, застосований нами під час розробки, адже проект структурно поділений на наступні теки:

- Тека «Model» містить основні файли (класи), що відповідають безпосередньо за можливість структурованого доступу до даних. Наприклад, клас «Department», тобто «Департамент» містить основні «поля» «Id», «Name» в які дані будуть передаватись з таблиці бази даних:

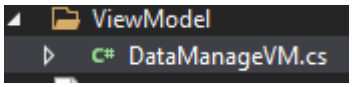
```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
namespace ManageStaffDBApp.Model
{
    Ссылок: 19
    public class Department
    {
        Ссылок: 6
        public int Id { get; set; }
        Ссылок: 6
        public string Name { get; set; }
        Ссылок: 0
        public List<Position> Positions { get; set; }
        [NotMapped]
        Ссылок: 0
        public List<Position> DepartmentPositions
        {
            get
            {
                return DataWorker.GetAllPositionsByDepartmentId(Id);
            }
        }
    }
}
```

- Тека «View» містить файли-представлення (view), або іншими словами колекцію елементів (вікон), з якими користувач може взаємодіяти напряму з нашим додатком, наприклад:



Вікно «AddNewDepartmentWindow.xaml», або вікно «Додавання нового департаменту» та друга частина – програмний код вікна

- Тека «ViewModel» містить основний файл (клас), в якому реалізована основна логіка обробки взаємодії між нашим представленням (view) і моделлю (model) у вигляді програмного коду:



Тека «ViewModel» та клас «DataManageVM»

Щодо роботи з даними існує три основних підходи:

- «Code First»

Підхід "Code First" полягає у створенні бази даних за допомогою класів C#, а потім у використанні фрейм ворку сутностей для "міграції" бази даних за допомогою команд.

Головне, що потрібно розуміти про «Code First» - це те, що оскільки ви створюєте базу даних за допомогою коду.

Переваги використання підходу «Code First»:

- ✓ Ваш код завжди звертається до схеми бази даних, яку він контролює, тому рідкою виникають "невідповідності" у типах стовпців, визначеннях таблиць тощо.
- ✓ У вас є нестандартний спосіб керувати міграціями баз даних (тому немає ніяких хитромудрих "запустіть цей скрипти перед запуском" типу "запустіть цей скрипти перед запуском")
- ✓ Майже будь-хто може створити базу даних, якщо він знає C#, навіть без особливих знань SQL в цілому (хоча... це може бути негативним моментом)
- ✓ Працює з різними базами даних, такими як Postgres, MySQL і т.д.

- «Database First»

Головною особливістю цього підходу є те, що його варто застосовувати, якщо у вас вже є створена база даних, що має певні таблиці для зберігання інформації.

Якщо є база даних виробничої системи, яка вже має, скажімо, 50 або більше таблиць, і кожна з цих таблиць має звичні стовпці, зовнішні ключі та обмеження. Якщо ми підходимо до роботи з кодом, то виникає кілька проблем:

- Нам доведеться вручну вводити кожне визначення таблиць, які ми хочемо використовувати, і ми повинні зробити все можливе, щоб вони були ідеально узгоджені.
- Ми не можемо використовувати міграцію коду, тому що база даних вже існує і, швидше за все, буде управлятися по-іншому. Це не проблема, але ми втрачаємо одну з величезних переваг використання підходу «Code First».

➢ «Model First»

Суть цього підходу полягає в тому, що спочатку робиться модель, а потім за нею створюється база даних.

Підхід Model-first передбачає створення моделі сутності (її концептуальної частини), а потім генерування контексту та прецедентів сутності, а також DDL для створення схеми бази даних на основі цієї моделі.

Базуючись на тому, що в нашому проекті немає попередньо створеної бази даних з відповідними таблицями і даними, ми будемо використовувати підхід «Code First», який зможе покрити всі наші вимоги:

✓ Створити базу даних

✓ Згенерувати необхідні таблиці

✓ Додати дані, згідно вимог: «Департамент», «Співробітник», «Посада»

3.2. Проектування структури бази даних

Наш проект – аналог певної CRM-системи (Customer Relationship Management Sytem) - інструментарій, які компанії використовують для управління та аналізу взаємодії з клієнтами або своїми співробітниками, їхніми даними протягом усього життєвого циклу компанії.

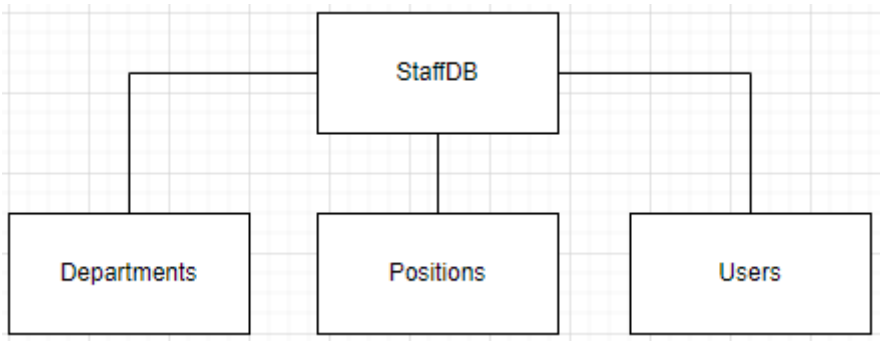
Згідно цього, основні дані, якими ми будемо оперувати, це:

✓ Дані про департамент

✓ Дані про співробітників

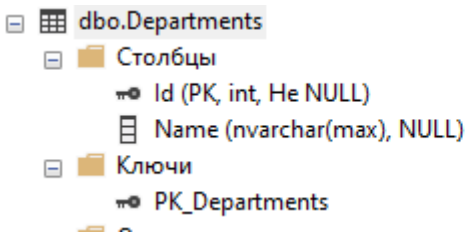
- ✓ Дані про посади

Відповідно, структура нашої бази даних виглядає наступним чином:



База даних «StaffDB» містить три таблиці: «Departments», «Positions», «Users».

Таблиця «Departments» містить колонку «Id», що виступає в ролі первинного ключа («Первинний ключ» (Primary Key) - який також називають первинним ключовим словом, - це стовпець у таблиці реляційної бази даних, який є відмінним для кожного запису і є унікальним ідентифікатором), а також – колонка «Name», що містить дані про назву наших департаментів.



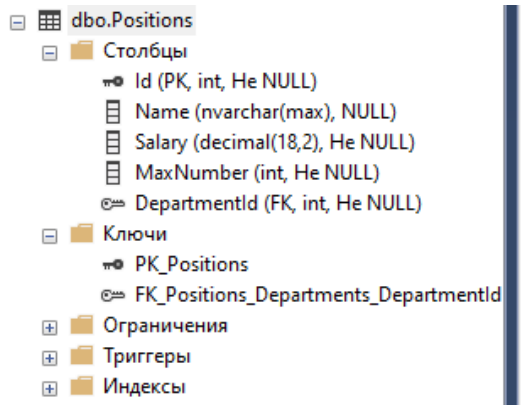
	Id	Name
1	1	IT - department
2	2	Salary department
3	3	Marketing department
4	4	Cleaning department
5	5	Accounting department

Структура таблиці «Departments», або «Департаменти»

Таблиця «Positions» містить в собі:

- ✓ Колонка «Id», що виступає в ролі первинного ключа
- ✓ «Name» містить дані про назву наших посад
- ✓ «Salary» - дані про розмір окладу
- ✓ «MaxNumber» вказує на максимальну кількість співробітників на посаді
- ✓ «DepartmentId» - відповідний департамент у вигляді зовнішнього ідентифікатора («Зовнішній ідентифікатор» (Foreign Key) – це поле (або набір полів) в одній таблиці, яке посилається на ПЕРВИННИЙ КЛЮЧ в іншій таблиці. Таблиця з зовнішнім ключем називається

дочірньою таблицею, а таблиця з первинним ключем називається таблицею, на яку посилаються, або батьківською таблицею) на таблицю «Departments»

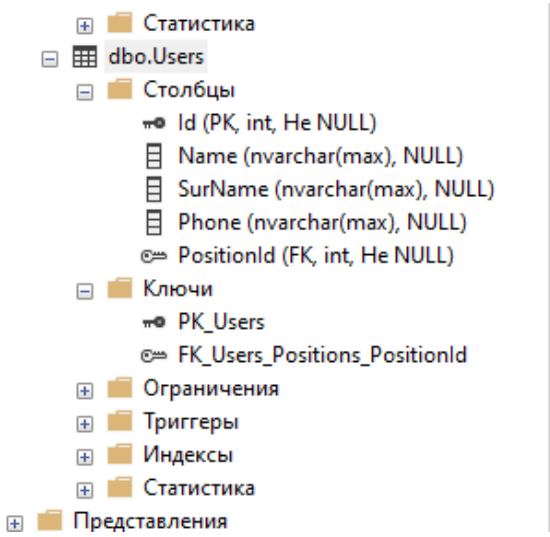


	Id	Name	Salary	MaxNumber	DepartmentId
1	1	Junior developer	1000.00	3	1
2	2	Middle developer	2000.00	4	1
3	3	Senior developer	3000.00	3	1
4	4	DevOps	5000.00	1	1
5	5	Payroll accountant	1000.00	2	2
6	6	Salary review manager	1500.00	1	2
7	7	Creative marketing manager	1300.00	1	3
8	8	Office cleaner	600.00	6	4
9	9	Accountant	1500.00	4	5
10	10	Accountant middle	2.00	2000	5
11	11	Accountant senior	3000.00	1	5

Структура таблиці «Positions», або «Посади»

Таблиця «Users» містить в собі:

- ✓ Колонка «Id», що виступає в ролі первинного ключа
- ✓ «Name» - ім’я співробітника
- ✓ «SurName» - прізвище співробітника
- ✓ «Phone» - контактний номер телефону
- ✓ «PositionId» - ідентифікатор посади, яке виступає у вигляді зовнішнього ключа на таблицю «Positions»



	Id	Name	SurName	Phone	PositionId
1	1	Daniil	Kozak	0982741229	1
2	2	Victoria	Petrova	3800671010101	7
3	3	Yurii	Ivanov	3800504440000	2
4	4	Mark	James	073494239432	4
5	5	Daniil	Kozak	0982741229	2
6	6	Aleksey	Aleksov	380904449999	3
7	7	Olena	Zelenska	3809991112233	5
8	8	Morgan	Johnson	1234567890	6
9	9	Sophia	Shevchenko	0987652211	1
10	10	Olga	Knopushko	7896754433	8
11	11	Ostap	Ostapenko	1234567899	8
12	12	Tetiana	Kozak	0677543455	9
13	13	Roman	Shuhevich	0504567899	10
14	14	Stepan	Giga	3456789900	11

Структура таблиці «Users», або «Співробітники»

Відповідно до наведених вище зображень, робимо висновок, що наші таблиці в базі даних мають другу ступінь нормалізації даних, що є цілком прийнятним для зберігання і відображення інформації.

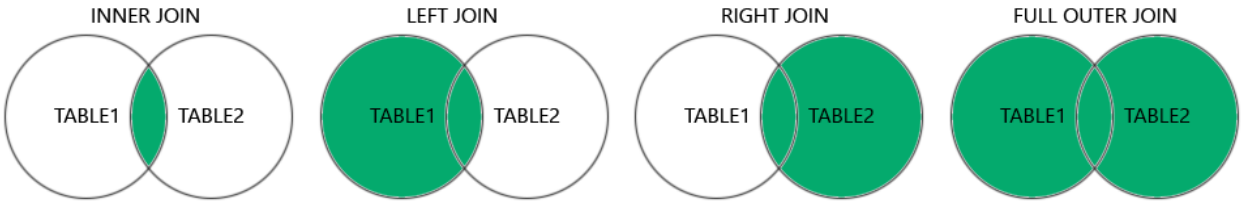
На практиці кожна компанія залишає за собою право нормалізувати або не нормалізувати свою базу даних, відповідно до умов з боку бізнесу, або користувачів – тобто програмістів, адже кінцеві користувачі, які будуть працювати з даними в CRM-системі не зможуть відрізнити нормалізовану таблицю в готовому вигляді від ненормалізованої.

Кінцева таблиця, яка доступна користувачам в програмі може як формуватись з одного запиту типу «SELECT», так і поєднуватись з багатьох таблиць за допомогою «комбінованих запитів»:

- (INNER) JOIN: Повертає записи, які мають однакові значення в обох таблицях
- LEFT (OUTER) JOIN: Повертає всі записи з лівої таблиці та відповідні записи з правої таблиці
- RIGHT (OUTER) JOIN: Повертає всі записи з правої таблиці та відповідні записи з лівої таблиці
- FULL (OUTER) JOIN: Повертає всі записи, якщо є збіг у лівій або правій таблиці

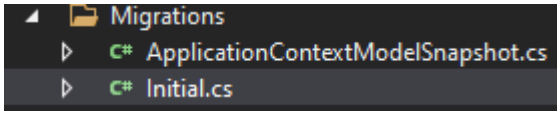
```
SELECT * FROM Customers;
```

Приклад звичайного запиту типу «SELECT» без додаткових умов

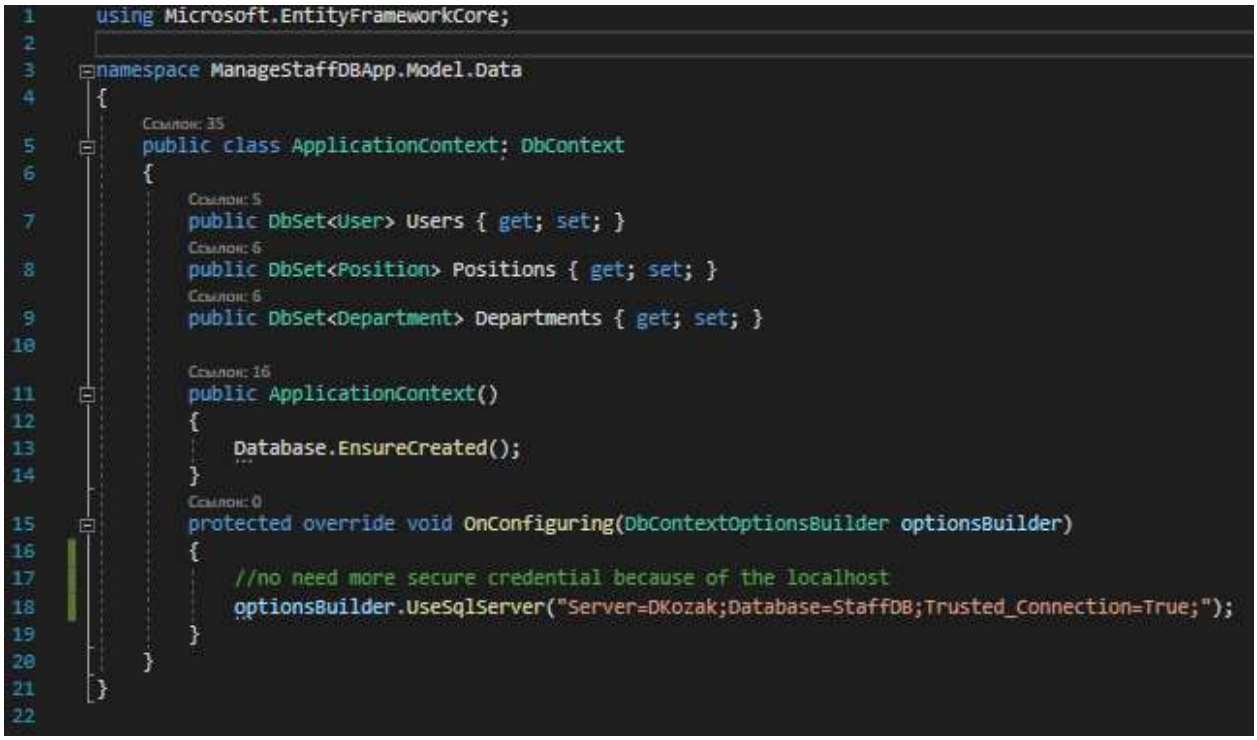


Діаграма прикладів комбінованих запитів SQL

Формування нашої бази даних відбувається на етапі першого запуску програми за допомогою класу «Initial» типу доступу «public» та типу класу «partial», що є нащадком класу «Migrations» розширення фреймворку .NET Core («ApplicationContextModelSnapshot.cs» - згенеровано автоматично):



Підключення до бази даних залежить від класу «ApplicationContext.cs»:



```
using Microsoft.EntityFrameworkCore;

namespace ManageStaffDBApp.Model.Data
{
    Ссылок: 35
    public class ApplicationContext: DbContext
    {
        Ссылок: 5
        public DbSet<User> Users { get; set; }
        Ссылок: 6
        public DbSet<Position> Positions { get; set; }
        Ссылок: 6
        public DbSet<Department> Departments { get; set; }

        Ссылок: 16
        public ApplicationContext()
        {
            Database.EnsureCreated();
        }
        Ссылок: 0
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            //no need more secure credential because of the localhost
            optionsBuilder.UseSqlServer("Server=DKozak;Database=StaffDB;Trusted_Connection=True;");
        }
    }
}
```

Клас «ApplicationContext.cs»

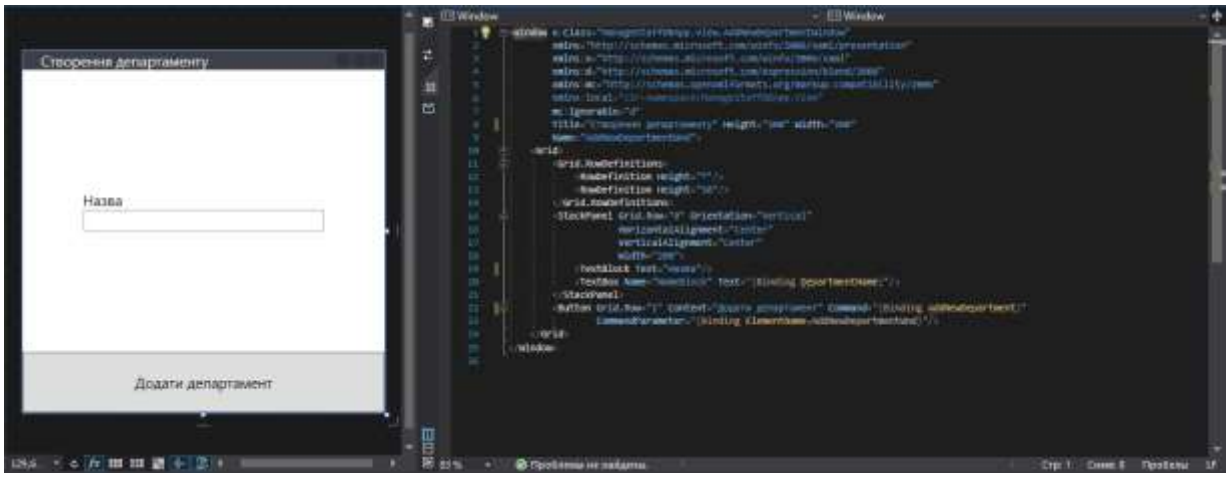
Відповідно до кожного наступного запуску програми будь-яким користувачем, всі дані, що містяться в базі даних будуть автоматично під вантажені в нашу CRM-систему і доступні користувачам для подальшої взаємодії.

3.3. Розробка функцій інтерфейсу користувача

Так як патерн Model-View-ViewModel (MVVM) - це потужна архітектура для побудови користувацьких інтерфейсів у додатках, що забезпечує чіткий розподіл завдань, які полегшують обслуговування, тестування та незалежну розробку компонентів, ієрархічна побудова структури проекту була поділена на розділи, про які ми згадували вище – розділ (тека) «View», «Model», «ViewModel», відповідно за користувацький інтерфейс відповідає розділ «View».

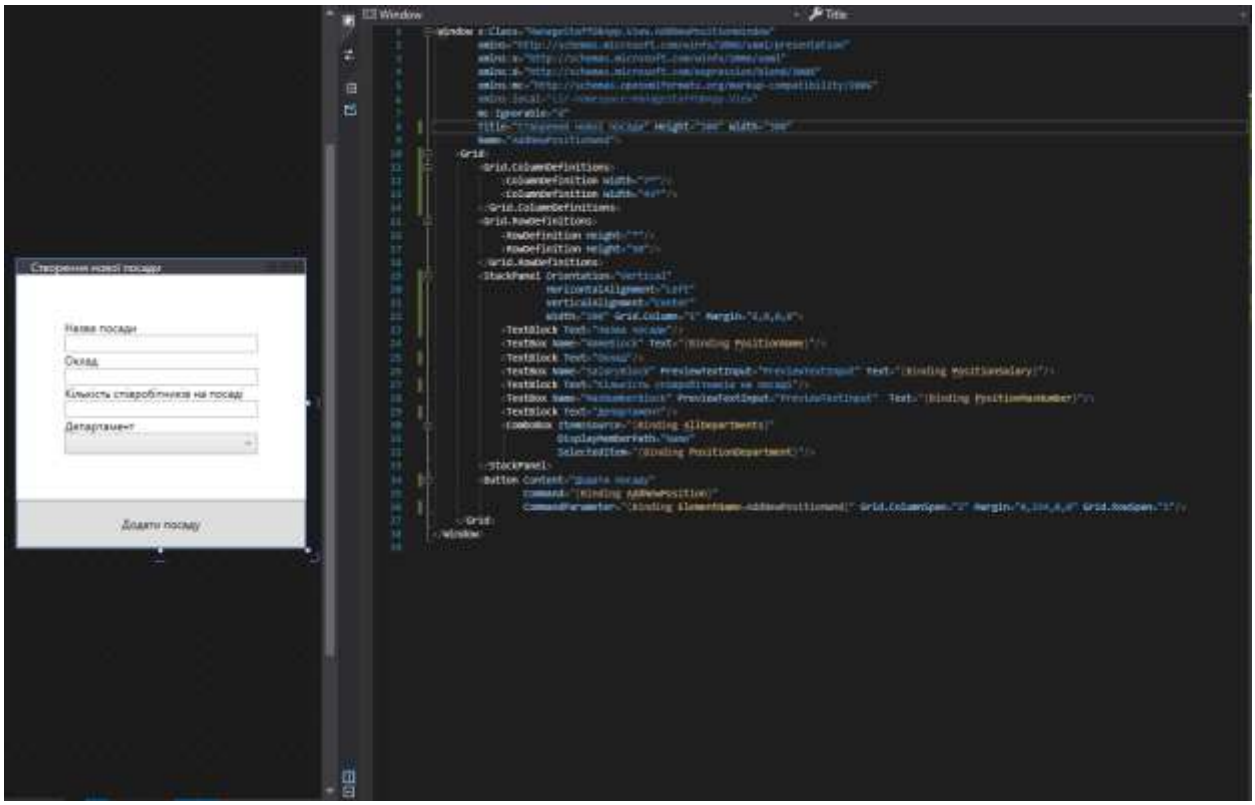
Цей розділ містить в собі наступні сторінки для взаємодії:

- Сторінка «AddNewDepartmentWindow.xaml», або вікно «Додавання нового департаменту» та друга частина – програмний код вікна



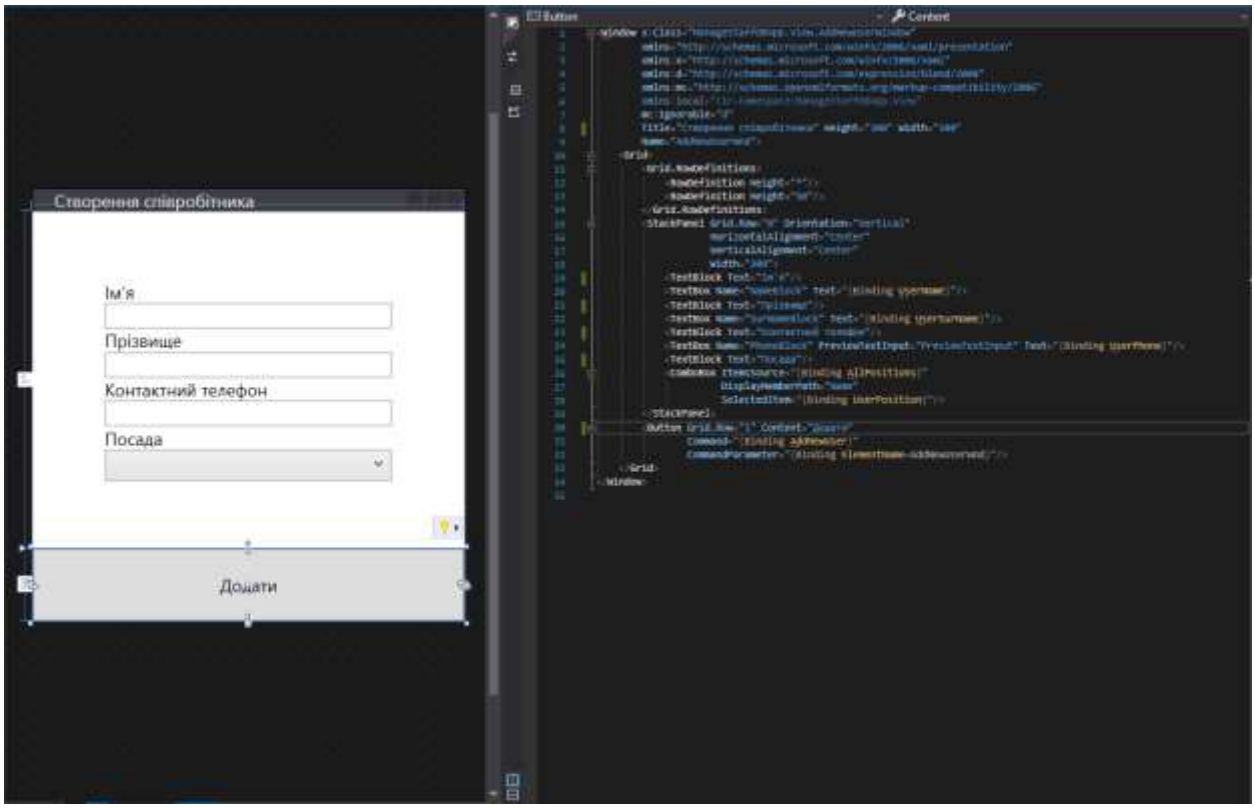
Сторінка «AddNewDepartmentWindow.xaml»

- Сторінка «AddNewPositionWindow.xaml», або «Створення нової посади в департаменті»



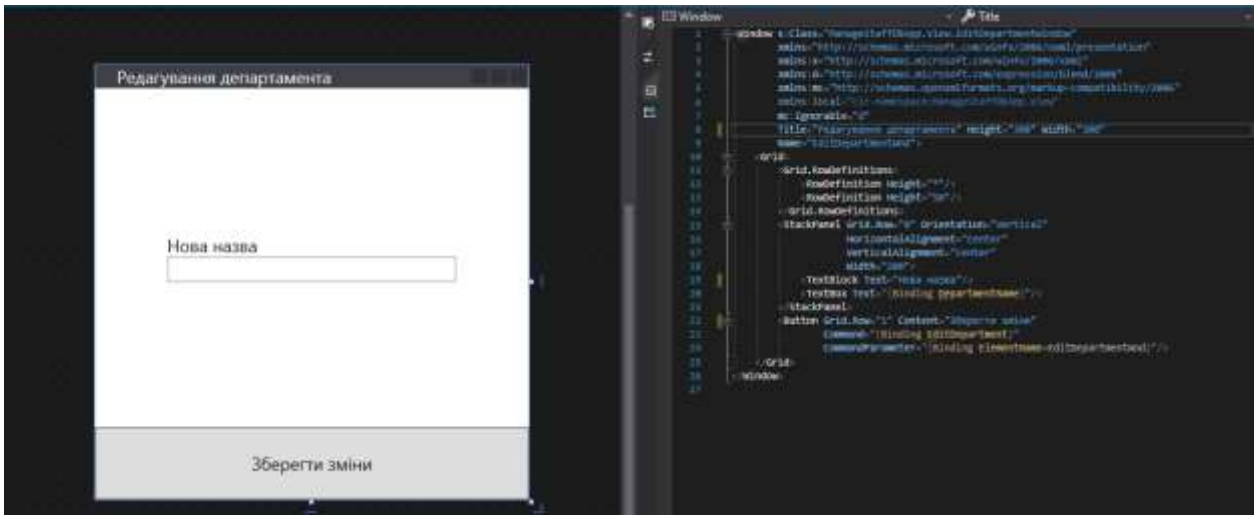
Вікно «AddNewPositionWindow.xaml»

- Сторінка «AddNewUserWindow.xaml», або «Створення співробітника»



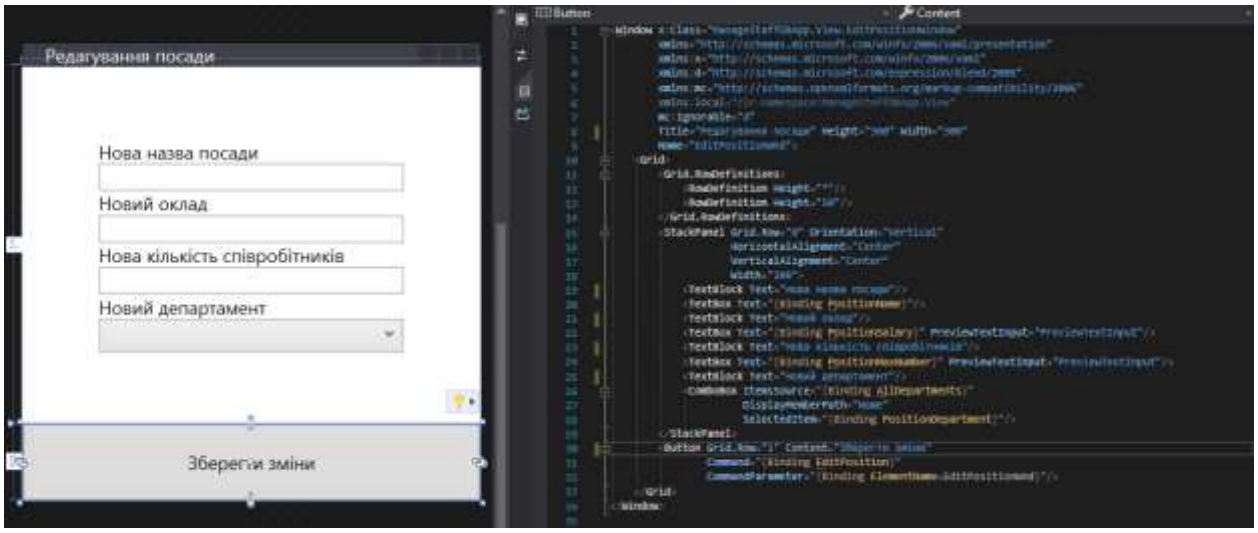
Вікно «AddNewUserWindow.xaml»

- Сторінка «EditDepartmentWindow.xaml», редагування назви департамента



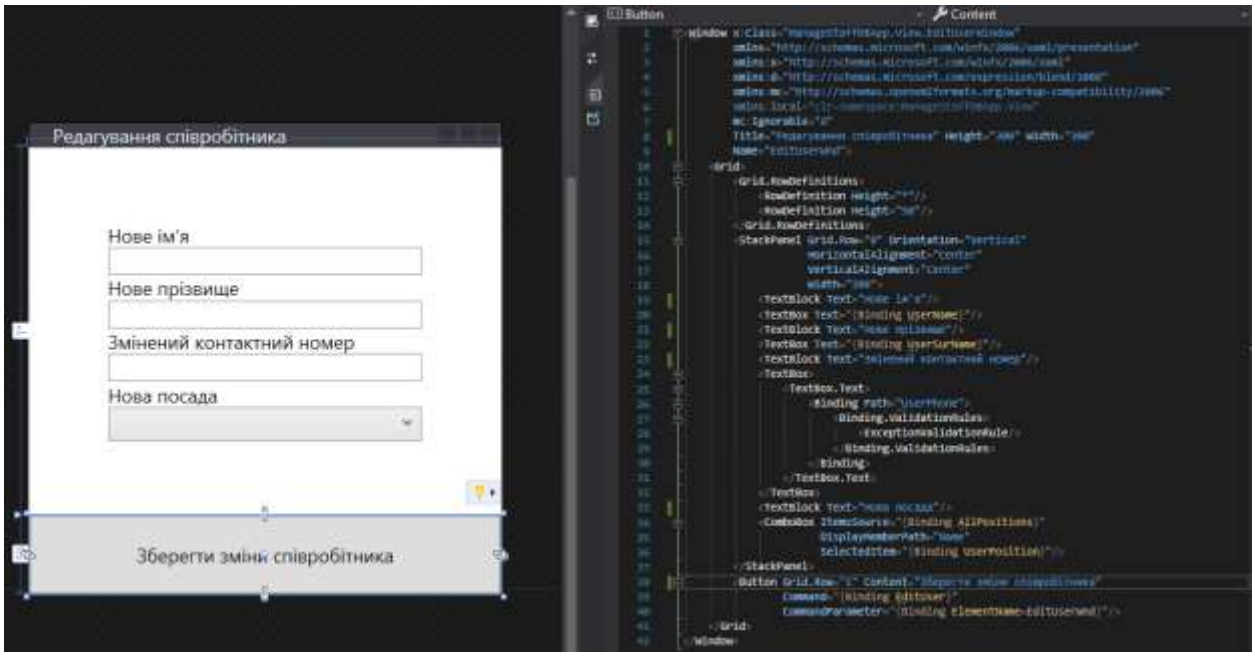
«EditDepartmentWindow.xaml»

- Сторінка «EditPositionWindow.xaml», редагування посади



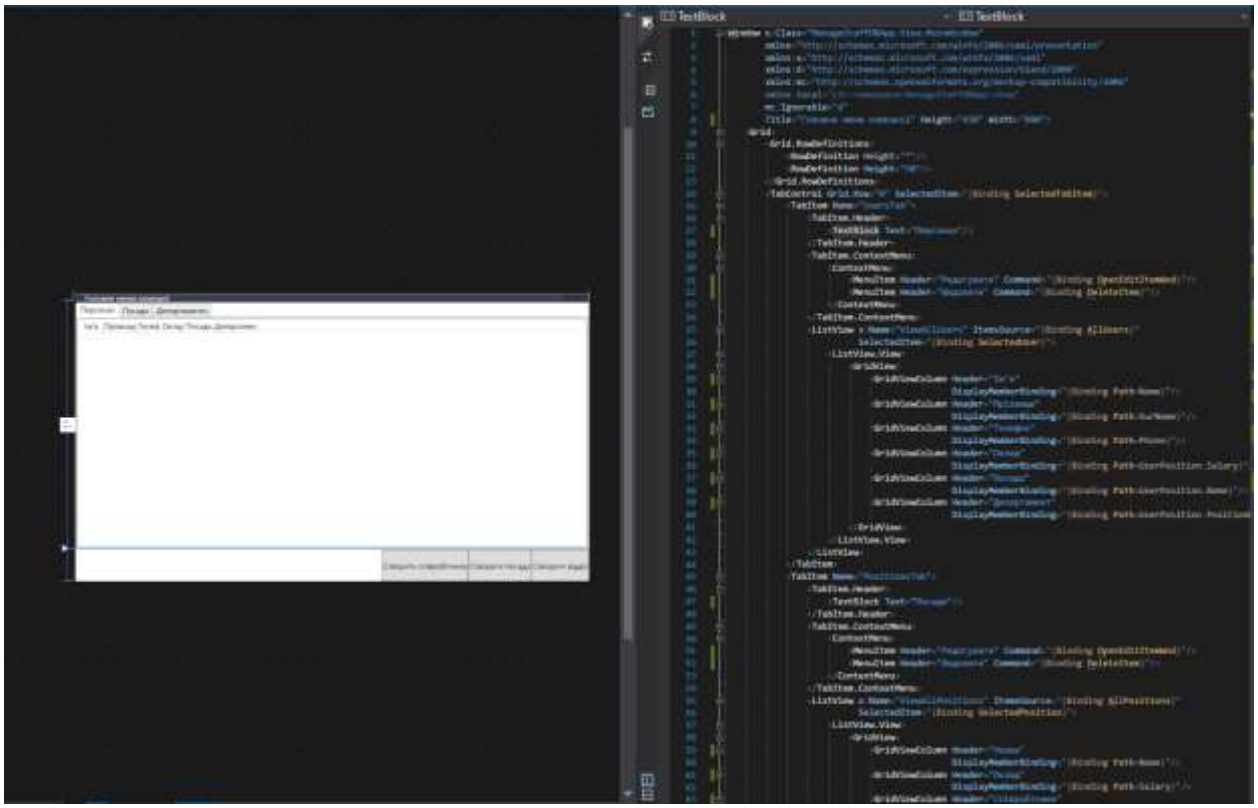
«EditPositionWindow.xaml»

- «EditUserWindow.xaml», редагування співробітника



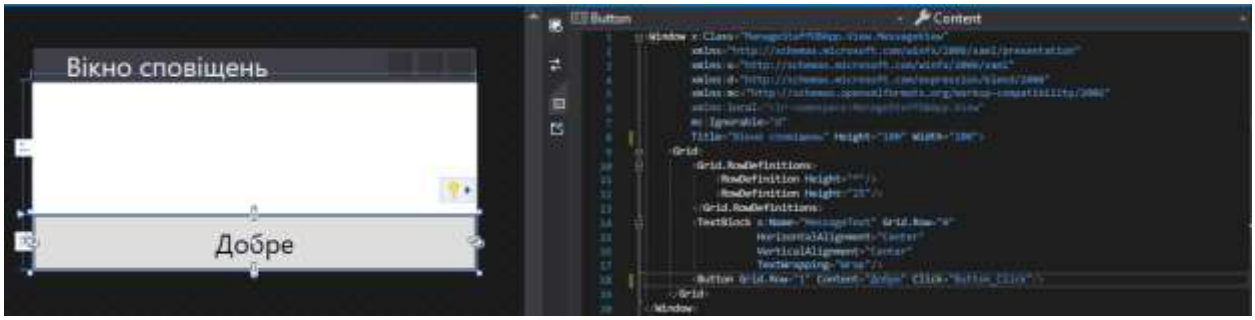
«EditUserWindow.xaml»

- Сторінка «MainWindow.xaml», або головне меню програми



«MainWindow.xaml»

- Сторінка «MessageView.xaml», або вікно сповіщень



«MessageView.xaml»

Для прикладу розглянемо наведене зображення «MainWindow.xaml», що складається з двох частин – лівої і правої:

- Ліва частина зображення – те саме «view», що може містити певні дані як з нашої БД (бази даних), так і з нашої програмної частини, тобто «розробники» повідомлення, прописані вручну, аби сповістити користувача про певну подію.
- Права частина – саме програмний код, або, як прийнято, називати серед спільноти розробників – «розмітка» - тобто файл, що містить в собі код розміщення тих чи інших елементів для взаємодії з даними з боку користувача.

Як ми можемо спостерігати на зображенні нижче, ми маємо три основні кнопки для взаємодії з головною сторінкою нашої CRM, які «спілкуються» з нашим класом «DataManageVM.cs» в розділі «ViewModel» за допомогою підходу «прив'язки», або «Binding».

«Binding» - сленг програмістської спільноти, що пішов від англійського слова «», тобто – прив'язати.

Фрагмент коду: «<Button Content="Створити співробітника" Command="{Binding OpenAddNewUserWnd}"/>» містить в собі прив'язку до команди «OpenAddNewUserWnd», що реалізувана в нашій «ViewModel», яка, насамперед, в собі містить визов методу «OpenAddUserWindowMethod()», в якому вже і відбувається виклик нового вікна.

```
<StackPanel Grid.Row="1" Orientation="Horizontal"
            HorizontalAlignment="Right">
    <Button Content="Створити співробітника" Command="{Binding OpenAddNewUserWnd}"/>
    <Button Content="Створити посаду" Command="{Binding OpenAddNewPositionWnd}"/>
    <Button Content="Створити відділ" Command="{Binding OpenAddNewDepartmentWnd}"/>
</StackPanel>
```

Фрагмент файлу розмітки сторінки «MainWindow.xaml»

```
public RelayCommand OpenAddNewUserWnd
{
    get
    {
        return openAddNewUserWnd ?? new RelayCommand(obj =>
        {
            OpenAddUserWindowMethod();
        }
        );
    }
}
```

Команда «OpenAddNewUserWnd» в розділі «ViewModel»

```
private void OpenAddUserWindowMethod()
{
    AddNewUserWindow newUserWindow = new AddNewUserWindow();
    SetCenterPositionAndOpen(newUserWindow);
}
```

Метод «OpenAddUserWindowMethod()»

3.4 Висновки до розділу

C# - це сучасна універсальна мова програмування, яку можна використовувати для виконання широкого спектру завдань і цілей, що охоплюють різні професії. C# в основному використовується на платформі Windows .NET, хоча її можна застосовувати і на платформах з відкритим кодом.

Ця надзвичайно універсальна мова програмування є об'єктно-орієнтованою мовою програмування (ООП) і є порівняно новою, але надійною та популярною.

Як і інші універсальні мови програмування, C# можна використовувати для створення різноманітних програм і додатків: мобільних додатків, десктопних додатків, хмарних сервісів, веб-сайтів, корпоративного програмного забезпечення та ігор. Багато-багато ігор. Незважаючи на те, що мова C# є надзвичайно універсальною, є три сфери, в яких вона найчастіше використовується:

- Програми для Windows
- Ігрова індустрія комп’ютерних ігор і не тільки
- Для розробки веб-сайтів

C# надає як початківцям, так і досвідченим програмістам безліч різноманітних переваг.

- ✓ Мабуть, найбільшою перевагою є те, скільки часу ви можете заощадити, використовуючи C# замість іншої мови програмування. Оскільки мова C# має статичну типізацію і легко читається, користувачі можуть розраховувати на те, що їм доведеться витрачати менше часу на пошук крихітних помилок, які порушують роботу програми.
- ✓ C# також наголошує на простоті та ефективності, тому програмісти можуть витрачати менше часу на написання складних блоків коду, які багаторазово використовуються в проекті. Додайте до цього обширний банк пам'яті, і ви отримаєте мову, що ефективно використовує час, яка може легко скоротити робочі години та допомогти вам вкластися у стислі терміни.
- ✓ C# є повністю об'єктно-орієнтованою, що є рідкісною характеристикою для мови програмування. Багато з найпоширеніших мов в тій чи іншій мірі включають об'єктну орієнтацію, але дуже мало хто з них досягнув такого масштабу, як C#, не втративши при цьому прихильності людей. Об'єктно-орієнтоване програмування (ООП) має багато різних переваг, таких як ефективність та гнучкість.

Microsoft .NET - це платформа преміум-класу, яка зарекомендувала себе як одне з найефективніших і найнадійніших рішень для створення надійних, безпечних і масштабованих веб та Десктопні додатків. Широке використання більшості компаній свідчить про те, що Microsoft .NET є популярним вибором для розробки великомасштабних додатків.

Платформа пропонує безліч переваг як для розробників, так і для кінцевих користувачів. Кінцеві користувачі можуть розраховувати на повнофункціональні, багатофункціональні додатки з інтуїтивно зрозумілим інтерфейсом, що дозволяє їм легко та ефективно виконувати свої завдання.

З іншого боку, розробники та дизайнери отримують гнучкість та динамічні функції, необхідні для створення веб-сайтів та додатків з простотою, швидкістю та ефективністю.

.NET розроблений таким чином, щоб бути платформо незалежним, тобто один і той же код може працювати на широкому спектрі операційних систем, таких як Windows, macOS, Linux, і навіть на мобільних платформах, таких як Android і iOS. Це означає, що розробники можуть створити єдину версію свого додатку, яка може працювати на різних платформах, без необхідності писати окремий код для кожної платформи.

Загалом, незалежність .NET від платформи надає багато переваг як для розробників, так і для користувачів, що робить його ідеальним вибором для створення крос-платформних додатків, які відповідають вимогам сучасного цифрового ландшафту, що швидко розвивається та постійно змінюється.

Служби розробки Microsoft .NET мають повний набір елементів керування інтерфейсом користувача, які забезпечують інтуїтивно зрозумілу та чуйну взаємодію з користувачем. Ці елементи включають кнопки, текстові поля, списки, що розкриваються, календарі та багато іншого. Платформа розроблена таким чином, щоб полегшити розробникам створення цікавих і функціональних користувацьких інтерфейсів.

Microsoft .NET - це комплексна програмна платформа, яка спрощує процес розробки, полегшуючи та підвищуючи ефективність створення та підтримки

додатків. Завдяки її потужним інструментам розробники можуть долати перешкоди та розробляти додатки будь-якою мовою програмування, не обмежуючись мовними рамками.

Це призводить до підвищення продуктивності та скорочення часу розробки, дозволяючи розробникам зосередитися на створенні інноваційних рішень для задоволення потреб вашого бізнесу.

Таким чином, Microsoft .NET пропонує комплексне рішення для потреб вашого бізнесу, забезпечуючи спрощену розробку, безшовну інтеграцію, масштабованість та вбудовані функції безпеки. Незалежно від того, чи ви розробляєте новий додаток, чи модернізуєте існуючий, .NET є найкращим вибором для підприємств, які потребують надійної продуктивності та інноваційних рішень.

В наш час неможливо собі уявити використання .NET без одного з підходів в роботі з базами даних, таких як «Code First», «Database First», «Model First», саме тому пропоную коротко підсумувати за що відповідає кожен з підходів, і окремо розглянути обраний мною підхід «Code First».

Використання підходу «Database First» полягає в тому, аби оперувати даними в вашій програмі із вже існуючої БД з колись створеними таблицями, що містять в собі дані, відповідно до вимог тих чи інших користувачів, технічних особливостей системи чи потреб компанії загалом і зазвичай використовується великими компаніями, що мать не одну базу даних із великим обсягом даних всередині.

Використання підходу «Model First» більш підходить для компаній, яким необхідно надати БД певних особливих можливостей на основі вимог бізнесу, який вже створив певну початкову модель бази даних і хоче зберегти її структура, а не лишати створення БД но розсуд розробників, які можуть мати однакове уявлення про зберігання даних, однак інакше уявлення і розуміння структури бази.

Тепер пропоную розглянути переваги та недоліки використання «Code First».

Серед переваг можна позначити:

- ✓ Ви можете створити базу даних і необхідні таблиці з бізнес-сутностей
- ✓ Рекомендується для невеликих додатків, які не передбачають велику обробку даних
- ✓ Ви можете вказати колекції для швидкого завантаження та реалізації даних
- ✓ Надає повний доступ до коду, і ви можете легко вносити зміни в код

Не дивлячись на те, що даний підхід користується популярністю серед розробників, що працюють в малому та середньому бізнесі, цей підхід також містить і недоліки:

- Потрібно писати код, пов'язаний зі створенням бази даних
- Якщо після створення бази даних відбуваються якісь зміни в ній, це потрібно робити в класі бізнес-суб'єкта коду та запускати додаток для оновлення бази даних або за допомогою консолі менеджера пакетів
- Керувати базою даних через код складно, тому не рекомендується в додатках з великими обсягами даних, де потрібно обробляти велику кількість даних і є складна логіка побудови або підтримки даних
- Будь-які ручні зміни будуть втрачені, якщо ви оновите код з програми

Зважаючи на те, що наш проект не мав чіткого технічного завдання від бізнесу та структури, однак ми мали за мету створити зручний та зрозумілий інтерфейс для роботи з нормалізованими даними в базі, даний підхід повністю задовольнив всі потреби, враховуючи побудову нормалізованої БД 2 типу нормалізації.

Наостанок хотів би розглянути поняття нормалізації, які має переваги і чи доцільно застосовувати її взагалі.

Нормалізація бази даних має багато переваг. Ось деякі з основних переваг:

- ✓ За допомогою нормалізації можна усунути надмірність бази даних або дублювання даних.

- ✓ За допомогою нормалізації можна мінімізувати нульові значення.
- ✓ Це призводить до більш компактної бази даних (через меншу надлишковість даних/нулів).
- ✓ Мінімізація/уникнення проблем з модифікацією даних.
- ✓ Це спрощує написання запитів
- ✓ Структура бази даних є більш чіткою та зрозумілою.
- ✓ Базу даних можна розширювати, не впливаючи на існуючі дані.
- ✓ Пошук, сортування та індексування можуть бути швидшими, оскільки таблиця невелика і на сторінці даних можна розмістити більше рядків.

Де нормалізація - це метод оптимізації бази даних, за допомогою якого ми додаємо надлишкові дані до однієї або декількох таблиць. Це може допомогти нам уникнути дорогих з'єднань у реляційній базі даних.

По суті, процес перетворення нормалізованої схеми на ненормалізовану називається де нормалізацією, і розробники використовують його для налаштування продуктивності систем для підтримки критично важливих за часом операцій.

Тепер ми бачимо, що поняття де нормалізації та нормалізації - це технології, які використовуються в базах даних і є різними термінами. Нормалізація - це метод мінімізації винятків вставки, видалення та оновлення даних шляхом усунення надлишкових даних. Де нормалізація - зворотний процес нормалізації, який додає надлишковість до даних для покращення продуктивності та цілісності даних для конкретної програми.

На моїй практиці, застосування де нормалізації даних в базі допомогло скоротити виконання асинхронних запитів для зчитування даних в користувацький інтерфейс з години до лічених хвилин, завдяки з’єднанню декількох таблиць в одну, саме тому ці дві технології – нормалізація та де нормалізація – не є певним партерному створення бази даних, а виступають інструментами для налаштування середовища зберігання даних під особливі потреби кожного.

ВИСНОВКИ

Підводячи результати дипломної магістерської роботи ми чітко можемо відповісти на запитання: «Що ж таке нормалізація даних?».

Нормалізація - це фундаментальна концепція у світі проектування та управління базами даних. Вона забезпечує цілісність даних, зменшує надмірність і підвищує узгодженість даних, що полегшує обслуговування баз даних та ефективні запити до них.

Розуміючи різні нормальні форми і коли їх застосовувати, ви можете створювати добре структуровані бази даних, які підтримують інформаційні потреби вашої організації, уникаючи при цьому аномалій і неефективності даних.

Наступне питання, не менш важливе за попереднє: «Як працює нормалізація даних?».

Організація даних у базі даних здійснюється шляхом нормалізації. Це означає створення таблиць і зв'язування цих таблиць між собою відповідно до принципів, призначених для захисту даних і підвищення адаптивності бази даних шляхом усунення дублювання і непослідовної залежності.

Дисковий простір марнується через надлишкові дані, що призводить до проблем з обслуговуванням. Якщо дані, які вже існують у кількох місцях, потрібно змінити, вони повинні оновлюватися скрізь однаково. Якщо інформація зберігається виключно в таблиці "Клієнти", а не деінде в базі даних, змінити адресу клієнта значно простіше.

Якщо користувачеві цілком алогічно шукати адресу конкретного клієнта в базі даних "Клієнти", то для працівника, який телефонує від імені цього клієнта, це може здатися неправильним. Заробітна плата працівника повинна бути перенесена в таблицю Працівники, оскільки вона пов'язана з працівником або залежить від нього. Доступ до даних може бути ускладнений внаслідок непослідовних залежностей, оскільки шлях до них може бути неповним або пошкодженим.

Ключі в SQL

Перш ніж перейти до різних форм нормалізації даних, вам потрібно спочатку зрозуміти концепцію ключів в SQL. Ключем може бути один стовпець або комбінація стовпців, які однозначно ідентифікують рядки (або кортежі) в таблиці. Він також допомагає виявити повторювану інформацію і встановити зв'язки між різними таблицями.

Навіщо нормалізувати дані?

Нормалізація даних має вирішальне значення для баз даних і компаній через її вплив на ефективну комунікацію, прийняття рішень і загальну зручність використання інформації. Безладні та ненормалізовані дані створюють проблеми, які перешкоджають розумінню їх людиною та машиною.

Крім того, неорганізовані дані в базі даних негативно впливають на функціональність функцій і можливостей. Наприклад, пошук "Google Chrome" може давати не ті самі результати, що й пошук "Chrome" через неузгодженість у представленні даних. Це підриває точність і ефективність процесів пошуку і запитів, призводячи до менш точних результатів і неповних в’язків.

Отже, нормалізація робить пошук конкретних термінів або об'єктів більш ефективним і точним. Зміцнюються зв'язки між пов'язаними елементами даних, що дозволяє покращити пошук та аналіз інформації.

Крім того, ця практика також має ширші наслідки для функціональності та продуктивності бази даних. Нормалізовані дані легше сортувати, фільтрувати та аналізувати, що полегшує дослідження даних і розпізнавання закономірностей. Завдяки зменшенню кількості стовпців і покращеній організації користувачі можуть переглядати більше записів на одній сторінці, що покращує візуалізацію, розуміння і розпізнавання шаблонів.

З точки зору користувацького досвіду, це спрощує процеси, дозволяючи користувачам легко отримувати доступ до інформації та змінювати її, забезпечуючи при цьому узгодженість, точність і відсутність дублікатів або надлишків. Таким чином, кілька користувачів можуть одночасно працювати з однією і тією ж базою даних.

Після того, як ви впровадите нормалізацію даних у своїй організації, ви побачите численні покращення, і це значно спростить ваші щоденні завдання. Ось основні переваги, які вона принесе вам:

Легше сортувати дані - з нормалізованими даними легко працювати, що полегшує роботу ваших агентів.

Більше місця для зберігання - коли ви маєте справу з тер байтами і педа байтами, нормалізація даних може значно оптимізувати простір для зберігання.

Легше працювати з інструментами аналізу даних - нормалізовану базу даних можна легко підключити до інструментів обробки даних для їх візуалізації та аналізу. Без стандартизації ці рішення не мають точної інформації для роботи і можуть видавати некоректні результати.

Основні висновки

Завдяки нормалізації даних ви гарантуєте, що вся ваша інформація буде чистою, впорядкованою та легкодоступною. У підсумку, наявність надійних вхідних даних для роботи закладає фундамент для ефективної роботи всього іншого. І, звичайно, це стосується ITAM.

Нормалізація не лише покращує узгодженість і точність даних, але й покращує можливість пошуку, звітність і зручність роботи користувачів. У контексті управління активами вона відіграє життєвою важливу роль у підтримці надійної та всеосяжної бази даних активів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Нормалізація баз даних: повний посібник - видання 2021 року. 250 стр.

2. Нормалізація баз даних Друге видання м'яка обкладинка - 2021 р. 19-50 стр.

3. Нормалізація баз даних Мілан Нікіч – 2019 р. 30-34 стр.

4. Практикум з SQL від Френка Соломона, Прашанта Джаярама, Ауні Аль Сакка. 150 стр.

5. Пояснення щодо нормалізації бази даних. – 2018 р. 300 стр.

6. Нормалізація баз даних Повний посібник - видання 2019 р. 136-150 стр.

7. CLR через C# (Довідник розробника) 4-е видання - 2022 р. 64 стр.

8. "Murach's SQL Server 2019 для розробників" Брайан Сиверсон и Джоэл Мурах – 2019 р. 57 стр.

9. Системи баз даних: Практичний підхід до проектування, впровадження та управління 6-е видання Хомас Конноллі , Керолін Бегг – 2022 р. 89-94 стр.

10. Проектування та реалізація баз даних Едуард Скьоре – 2018 р. 24-70 стр.

11. Великі дані: Принципи та найкращі практики масштабованих систем даних у реальному часі 1-е видання Натан Марз, Джеймс Уоррен – 2015 р. 90-106 стр.

12. Вступ до систем баз даних Крістофер Дейт – 2016 р. 50-80 стр.

13. Інформаційні технології в науці, виробництві та підприємництві : збірник наукових праць молодих вчених, аспірантів, магістрів кафедри комп’ютерних наук та технологій / за заг. наук. ред. В. Ю. Щербаня. – Київ : ТОВ "Фастбінд Україна", 2023. 7-69 стр.

14. Дейт К. Введення в системи баз даних.- 2017 р. 98- 140 стр.

15. Бази даних MySQL, Надія Балик, Віктор Мандзюк – 2018 р. 390 стр.

16. Системи баз даних та знань. Книга 1. Організація баз даних та знань, Берко А.Ю., Верес О.М., Пасічник В.В. – 2019 р. 34-60 стр.

17. Head First патерни проєктування, Ерік Фрімен, Елізабет Робсон, Кеті Сьєрра, Берт Бейтс – 2022 р. 200-220 стр.

18. Мова програмування C# 7.0 та платформи .NET та .NET Core, Філіп Джепікс, Ендрю Троєлсен (англ.) – 2019 р. 277 стр.

19. Чистий Код, Роберт Мартін (укр.) – 2017 р. 130-132 стр.

20. C# in Depth: тонкощі програмування, Джон Скіт (англ.) – 2019 р. 100-120 стр.

21. Domain-Driven Design: Tackling Complexity in the Heart of Software, Ерік Еванс (укр.) – 2018 р. 40 стр.

22. Книга C# 4.0: повний посібник, Герберт Шилдт (англ.) – 2023 р.

23. CLR за допомогою C#. Програмування на платформі Microsoft .NET Framework 4.5 мовою C# , Джеффрі Ріхтер – 2022 р. 283 стр.

24. Джез Хамбл, Девід Фарлі "Безперервна доставка" – 2023р. 1-4 стр.

25. Роберт Мартін «Чистий код: створення, аналіз і рефакторинг» - 2022р 5-10 стр.

26. Томас Х. Кормен "Вступ до алгоритмів" – 2023р. 89-95 стр.

27. Корі Альтхофф "Програміст-самоучка: Вичерпний посібник з професійного програмування" – 2023 р. 45-50 стр.

28. Гарольд Абельсон, Джеральд Джей Сассман "Структура та інтерпретація комп'ютерних програм" – 2022 р. 28-37 стр.

29. Стів Макконнелл "Швидкий розвиток" – 2022 р. 45-39 стр.