

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

КОМАР ЮРІЙ ОЛЕКСАНДРОВИЧ

Допускається до захисту:

завідувач кафедри
інформаційних технологій,
д-р техн. наук, доцент

_____ Тетяна НЕСКОРОДСЬКА

«_____» _____ 2022р.

**РОЗРОБКА БАЗИ ДАНИЗ ДЛЯ ОБЛІКУ ТА МОНІТОРИНГУ ЦІН НА
ПРОДУКТИ**

Спеціальність 122 «Комп'ютерні науки»

Кваліфікаційна (бакалаврська) робота

Керівник:

Тетяна Мартянова, старший викладач
кафедри інформаційних технологій,
старший викладач

Оцінка: _____ / _____ / _____

(бали за шкалою ЄКТС / за національною шкалою)

Голова ЕК: _____

(підпис)

АНОТАЦІЯ

Комар Ю.О. Розробка бази даних для обліку та моніторингу цін на продукти. Спеціальність 122 «Комп'ютерні науки», освітня програма «Сучасні інформаційні технології та програмування». Донецький національний університет імені Василя Стуса, Вінниця, 2022.

Кваліфікаційна (бакалаврська) робота присвячена поетапному проектуванню бази даних з врахуванням усіх особливостей. Що дозволяє полегшити облік цифрової інформації, що пов'язана з продуктами у певній організації.

Користувач має змогу переглянути облік та ціну на продукти в різних місцях та організаціях. Наявний опис усіх деталей про продукти, місце постачання, де зберігаються та яка організація займається розповсюдженням.

Ключові слова: база даних, комерція, C#, моніторинг цін, продукти, PostgreSQL.

ABSTRACT

Komar Y.O. Development of a database for accounting and monitoring of product prices. Specialty 122 "Computer Science", educational program "Modern Information Technology and Programming". Vasyl Stus Donetsk National University, Vinnytsia, 2022.

Qualification (bachelor's) work is devoted to the stage-by-stage design of the database and all the features. This makes it easier to keep track of digital information related to products in a particular organization. The user has the opportunity to view the accounting and price of products in different places and organizations. There is a description of all the details of the products, where the supplies come from, where they are stored and which organization is involved in distribution.

Keywords: database, commerce, C #, price monitoring, products, PostgreSQL.

ЗМІСТ

ВСТУП	4
РОЗДІЛ 1. ПОСТАНОВКА ЗАДАЧІ, АКТУАЛЬНІСТЬ РОБОТИ, ОГЛЯД ІСНУЮЧИХ РІШЕНЬ	6
1.1 Постановка задачі	6
1.2 Основні поняття бази даних та аналіз предметної галузі.....	7
1.3 Огляд існуючих баз та рішень для обліку продуктів.....	9
Висновок до розділу 1	10
РОЗДІЛ 2. АНАЛІЗ ТА ВИБІР ІНСТРУМЕНТІВ ДЛЯ ПРОЕКТУВАННЯ БАЗИ ДАНИХ.....	11
2.1 Типи бази даних та вибір підходящої для вирішення задачі.....	11
2.2 Опис використаної система управління(СУБД)	18
2.3 Мова програмування C# та інструменти для роботи з базою даних	21
2.4 Додаткові інструменти для вирішення задачі.....	26
РОЗДІЛ 3. ПОКРОКОВИЙ ПРОЦЕС СТВОРЕННЯ БАЗИ ДАНИХ.....	36
3.1 Аналіз предметної області та створення схеми бази даних	36
3.2 Проектування таблиць, полів та зв'язків	37
3.3 Створення пустої бази даних та розгортання на сторонньому сервері...	44
3.4 Створення та накат міграцій	47
3.5 Заповнення таблиць даними	51
Висновок до розділу 3	54
ВИСНОВОК.....	55
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ	56

ВСТУП

На сьогоднішній день інформаційна індустрія набуває грандіозних обертів. Кількість цифрової інформації кожен день збільшується в геометричній прогресії. Вона тісно пов'язана з розвитком комп'ютерних технологій.

У 2022 році неможливо уявити життя без ефективного управління інформацією. Сприйняття реального світу можна порівняти з послідовністю різних, хоча іноді і взаємозалежних, явищ. Ці явища називають даними. Кожен вид інформації, або дані потрібно зберігати у певному місці в інформаційному просторі.

Дані – це сукупність окремої невеликої одиниці інформації. Вони можуть використовуватися у різних формах, таких як текст, цифри, носії інформації, байти тощо[1]. Вони можуть зберігатися на аркушах паперу, або в електронній пам'яті. Для цілей зберігання великої кількості інформації були створені бази даних.

Робота полягає у створенні бази даних для обліку та моніторингу цін на продукти. Кожен магазин, або навіть мережа магазинів має, хоч і невеличку, але базу даних. Часи коли інформація зберігалася на папері уже давно позаду, тому попит на зберігання даних у інформаційному просторі збільшується з кожним роком. Навіть у більшості невеликих організаціях наявний сайт за допомогою якого є можливість прослідкувати наявність товару. А коли мова йдеться про продукти, то це дійсно великі обсяги інформації.

У даній роботі розглянуто створення бази даних для обліку та моніторингу цін на продукти на основі реляційної моделі даних. Вона заснована на принципах теорії множин: об'єкти представляють собою елементи деяких множин і підмножин, а сама база даних є сукупністю двовимірних пов'язаних між собою таблиць.

Мета дипломного проектування:

- огляд предметної області та її особливостей;

- ознайомлення та використання найбільш підходящого інструментарію для вирішення задачі;
- створення безпосередньо бази даних та опис її наповнення.

Завдання дослідження:

Спроекувати та розробити базу даних для обліку та моніторингу цін на продукти.

Об'єкт дослідження:

База даних заповнена всією потрібною інформацією для роботи з предметною галуззю. Заповнені таблиці та спроектовані зв'язки.

Предмет дослідження:

Процес проектування бази даних, написання відповідного коду на обраній мові програмування для роботи з базою. У широкому сенсі вирішення даної задачі охоплює проектування, програмування, алгоритмізацію і аналіз даних.

Структура роботи:

Дана робота складається зі змісту, вступу, 3-х розділів. Перший розділ містить у собі три підрозділи, другий розділ містить чотири підрозділи, третій містить п'ять підрозділів. Робота містить 48 сторінок, 39 рисунків, список літератури з 20 джерел

РОЗДІЛ 1

ПОСТАНОВКА ЗАДАЧІ, АКТУАЛЬНІСТЬ РОБОТИ, ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Даний розділ присвячений постановці задач, також опису предметної області та огляд існуючих рішень.

1.1 Постановка задачі

Необхідно спроектувати базу даних для обліку та моніторингу цін на продукти. Повинні бути створені та заповнені відповідні таблиці. У базі даних необхідно побудувати зв'язки між таблицями та пояснити їх відношення. База має працювати таким чином, щоб була можливість отримати будь яку інформацію по певному продукту(пошук по ID, назві, кількості, тощо...). Відповідні запити мають бути оброблені.

У роботі використовується система управління базами даних – це комплекс програмних і мовних засобів, необхідних для створення баз даних.

Вимоги до роботи бази даних:

- a) модифікація даної структури;
- b) отримання певної потрібної інформації;
- c) оновлення застарілих даних.

У базі має бути зрозуміла інформація, якою надалі зможе розпоряджатись користувач. Кожна таблиця та поля максимально детально описані та зрозумілі.

Вимоги до опису об'єкту(продукту):

- a) об'єкт має свій унікальний первинний ключ;
- b) у продукту є зрозуміле ім'я;
- c) додатковий опис;
- d) ціна;
- e) регіон постачання.

База даних може бути використана у магазині, або в іншій організації для обліку продуктів.

1.2 Основні поняття бази даних та аналіз предметної галузі

Кожна людина майже щоденно відвідує магазини, склади, тощо... Для збільшення попиту та своєї репутації закладу потрібно постійно підтримувати наявність товарів на полицях. Кількість товару, його ціну, місце постачання, усю перелічену інформацію потрібно зберігати в певному вигляді в інформаційному просторі, так як великі об'єми інформації на паперових носіях уже давно позаду[2].

Окрім зберігання, дані постійно потребують оновлення та зміни. Для вирішення подібних задач були винайдені бази даних. З цього випливає, що подібний метод ведення обліку має свої переваги:

- а) компактність;
- б) швидкість;
- с) невисока трудозатратність;
- д) актуальність;
- е) централізоване управління даними.

Основні поняття про бази даних. База даних – це організоване зібрання даних, завдяки якому можна легко отримати до них доступ та керувати ними. У базі даних є можливість впорядковувати дані у таблиці, рядки, стовпці та індексувати їх, щоб полегшити пошук відповідної інформації. Обробники баз даних створюють базу таким чином, що лише один набір програмного забезпечення надає доступ до даних для всіх користувачів. Основною метою бази даних - є працювати з великою кількістю інформації шляхом зберігання, вилучення та управління даними. Кожна база даних має таблиці з набором полів та зв'язки між цими таблицями.

Для повноцінної роботи з базою використовується СУБД – система управління базами даних. Системи управління бувають двох видів, а саме: для одного користувача, або розраховані на багато користувачів.

Для одного користувача – це система, до якої тільки один користувач може отримати доступ, для багатьох користувачів – відповідною. Дані у системі бувають інтегрованими і подільними.

Під поняттям інтегрованості даних, мається на увазі можливість представити базу даних, як поєднання декількох окремих файлів даних, що повністю, або частково виключають надмірність зберігання інформації. Під поняттям подільності даних, розуміється можливість використання окремих елементів, які зберігаються у базі декількома різними користувачами.

Також наявний рівень програмного забезпечення, який знаходиться між фізичною базою і користувачами системи. Це є система управління. Усі запити, які користувач зробив до бази, оброблюються у системі управління. Основна задача систему управління – це надати користувачеві можливість працювати з базою, не поглиблюючись в деталі.

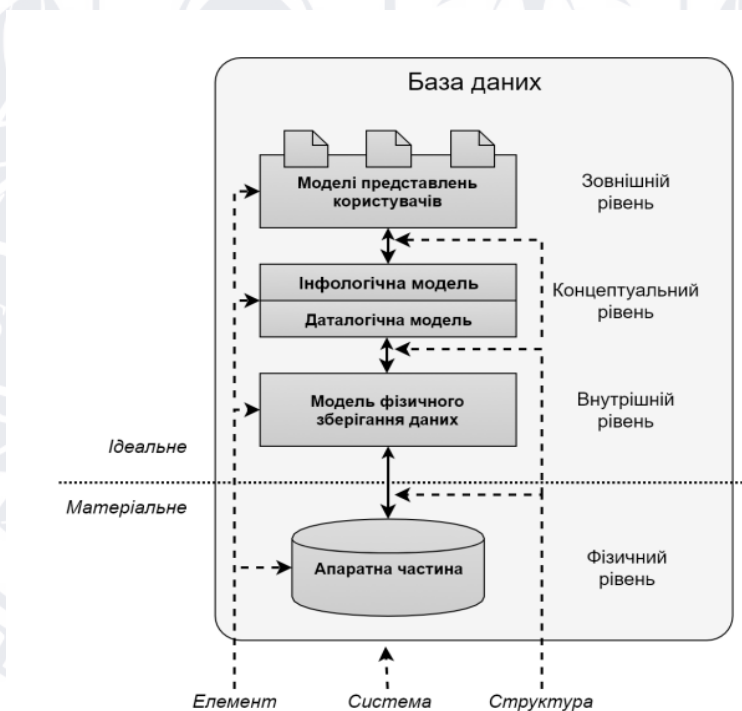


Рисунок 1.1 – Схема трьох рівнів архітектури[3]

На фоні вище викладеного матеріалу, потрібно зазначити про три рівня архітектури системи. Внутрішній, зовнішній та концептуальний. Внутрішній рівень найближчий до фізичного сховища інформації. По іншому, має зв'язок із різними способами зберігання інформації на різних фізичних пристроях. Зовнішній рівень найближчий до користувачів, концептуальний є проміжний рівень між першим та другим.

1.3 Огляд існуючих баз та рішень для обліку продуктів

На усьому інформаційному просторі є безліч різних баз даних та обліків певної продукції. Нажаль більшість із них знаходиться у закритому доступі, або на платній основі. Для огляду готового рішення, було обрано десктопний додаток «ABM Retail»[4]. Цей додаток був створений професійною командою розробників, саме для обліку та моніторингу наявності товарів, або їх ціни.

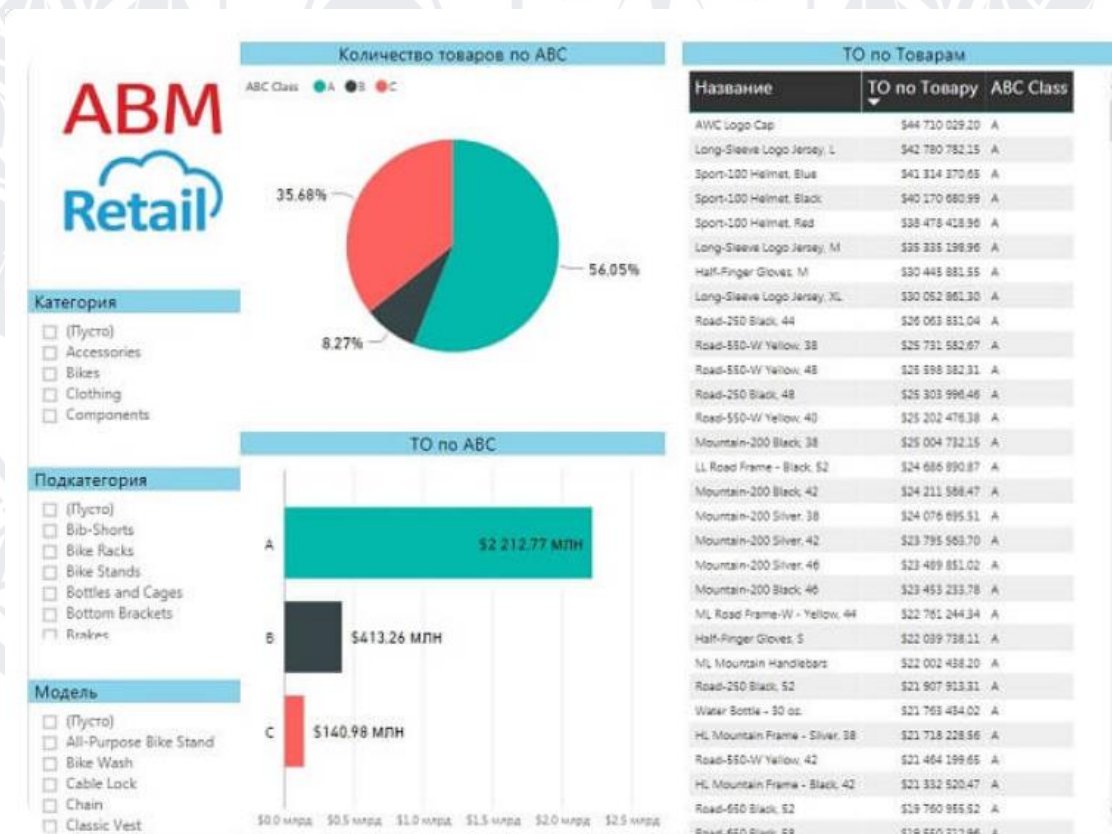


Рисунок 1.2 – Огляд готового рішення «ABM Retail»

Слід зазначити, що додаток працює з дуже великою кількістю даних, та має дійсно велику базу. Також у додатку реалізована функція підрахунку статистики. Цей додаток, а саме база даних допомагає магазинам та різним підприємствам значно облегшити облік та автоматизацію інформації.

Нажаль доступу до бази даних додатка немає, але з інших ресурсів є можливість зрозуміти, що база схожа за специфікацією до поставленої задачі.

Висновок до розділу 1

У цьому розділі була сформульована постановка задачі, розглянуто предметну галузь, а також визначено актуальність даної роботи. Наявний короткий опис основних понять про бази даних та їх використання. Проведено огляд існуючого програмного забезпечення для обліку та моніторингу цін на продукти. З наведеним прикладом робота є зрозумілою та наявний простір для вирішення поставленої задачі різними способами.

РОЗДІЛ 2

АНАЛІЗ ТА ВИБІР ІНСТРУМЕНТІВ ДЛЯ ПРОЕКТУВАННЯ БАЗИ ДАНИХ

Даний розділ присвячений інструментам та технологіям, які використовуються для вирішення поставленої задачі. На сьогоднішній день існує велика кількість інструментів для проектування баз даних. Огляд інструментів їх переваги та недоліки.

2.1 Типи бази даних та вибір підходящої для вирішення задачі

Існує багато типів баз даних і кожен з них призначений для певних цілей. Якийсь тип баз підходить для великих підприємств та обліку великої кількості інформації. Інші підходять для обліку «паперових» даних та документації. Деякі з них є відкритими, а інші є власністю певних компаній, або фірм. Також варто зазначити, що деякі з них є масштабні, оптимізовані, високодоступні. Натомість деякі важко підтримувати в великих масштабах.

Вибираючи певний тип бази, потрібно розуміти, що може запропонувати конкретна база даних. Специфіка кожної бази даних може відрізнятися, але загалом існує лише кілька типів у рамках яких можливо досягти тих самих цілей. Нижче перелічено усі найпопулярніші та сучасні типи бази даних.

Реляційні бази даних

Одна з основних моделей бази — реляційна модель. Робота з базами починається саме з цієї моделі. Реляційна модель є найпопулярнішою, та має доволі багато інформації для полегшення проектування. Такі бази дозволяють зберігати дані в реляційних таблицях з певними стовпцями певного типу.

ID	Name	Phone	Birthday
1	Andrew	222-31-31	1976
2	Bohdan	264-05-06	1990
3	Ivan	237-01-02	1989

Рисунок 2.1 – Таблиця як основний елемент реляційної бази даних[5]

Переваги даної моделі:

- підтримка SQL;
- простота;
- зрозумілість для користувачів;
- зручність реалізації на персональному комп'ютері;
- підтримка індексування та розділення.

Недоліки даної моделі:

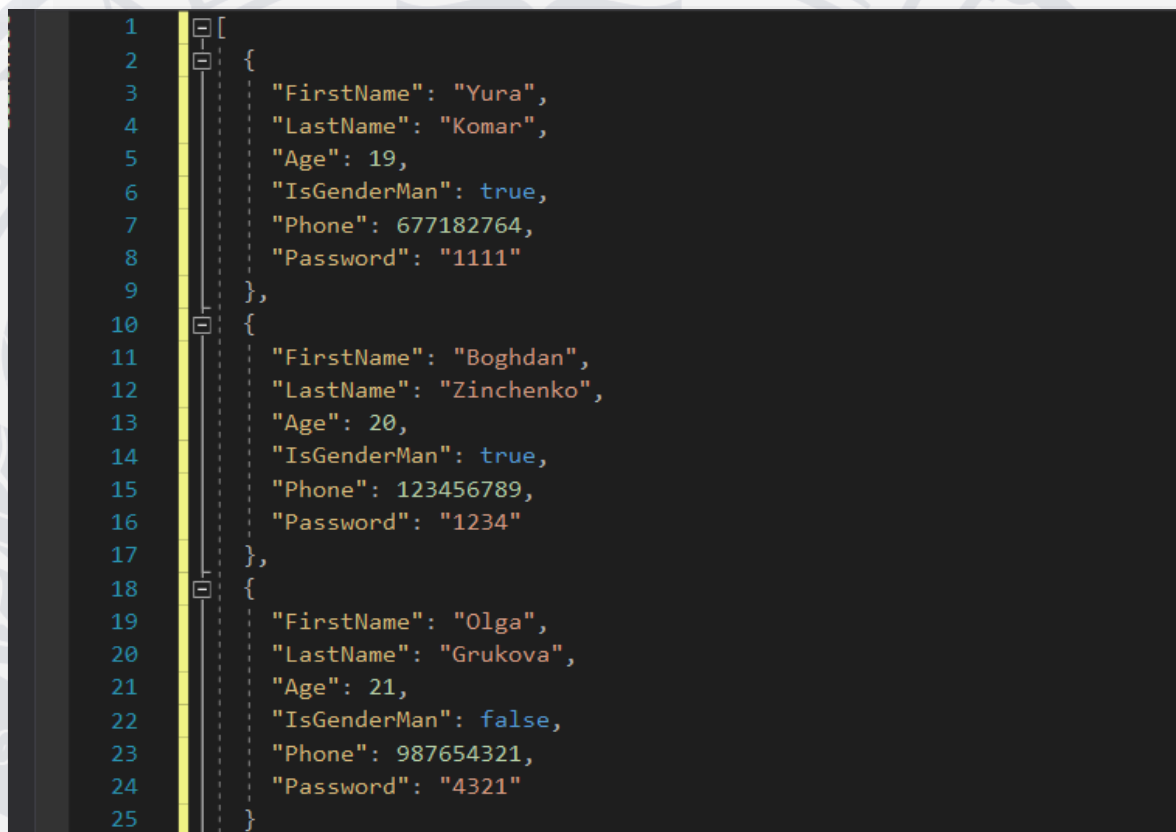
- погана підтримка неструктурованих даних та складних типів;
- погана оптимізація обробки подій;
- розширення бази у великих масштабах є трудозатратне у наслідок чого - дороге;
- складний опис мережових та ієрархічних зв'язків[6].

Приклади: MySQL, PostgreSQL, Oracle, MS Access.

Документо-орієнтовані бази даних

Якщо немає потреби об'єднувати кілька таблиць, щоб отримати певні дані, є можливість використовувати документо-орієнтовані бази. Вони дозволяють зберігати дані у вигляді JSON файлів. У цьому форматі можливо створити складне значення для будь-якого ключа і відразу включити всю структуру даних в один запис.

JSON – це текстовий формат обміну даними. JSON має текстовий формат, може бути прочитаним користувачем. Даний формат дає змогу описувати об’єкти та інші структури даних. Також використання JSON доволі просте і добре підходить для реалізації бази даних у простих програмах[7].



```
1  [
2  {
3      "FirstName": "Yura",
4      "LastName": "Komar",
5      "Age": 19,
6      "IsGenderMan": true,
7      "Phone": 677182764,
8      "Password": "1111"
9  },
10 {
11     "FirstName": "Boghdan",
12     "LastName": "Zinchenko",
13     "Age": 20,
14     "IsGenderMan": true,
15     "Phone": 123456789,
16     "Password": "1234"
17 },
18 {
19     "FirstName": "Olga",
20     "LastName": "Grukova",
21     "Age": 21,
22     "IsGenderMan": false,
23     "Phone": 987654321,
24     "Password": "4321"
25 }
```

Рисунок 2.2 – Приклад збереження даних у JSON файлі

Переваги:

- модель є безкоштовною;
- немає необхідності постійно записувати поля у кожному записі;
- хороша підтримка складних типів.

Недоліки:

- погана підтримка транзакцій;
- слабка аналітична підтримка;

- як і у випадку з реляційними базами, розширення у великих масштабах є трудозатратне у наслідок чого – дороге.

Приклад: MongoDB.

Бази даних в пам'яті

Бази даних цього типу можуть забезпечити відповідь у режимі реального часу для вибірки та змінення конкретних записів. Більшість цих записів зберігаються в RAM(Random Acces Memory – більше відома, як оперативна пам'ять). У даній моделі є можливість зберігати дані не тільки в оперативній пам'яті, а також можливість постійно зберігати інформацію на жорстких дисках(твердотільних накопичувачах). Більшість баз даної моделі працюють із записами типу «ключ – значення», тому значення можна зберігати у форматі, які є орієнтовані на документи. Наявна особливість того, що деякі бази даних працюють також зі стовпцями і дозволяють вторинне індексування тієї самої таблиці. Важливого значення набуває той факт, що використання оперативної пам'яті дозволяє швидко обробляти дані, але такий вид зберігання даних не є стабільним та безпечним.

```
box.execute("CREATE TABLE tester (s1 INT PRIMARY KEY, s2 VARCHAR(10))");

function string_function()
  local random_number
  local random_string
  random_string = ""
  for x = 1,10,1 do
    random_number = math.random(65, 90)
    random_string = random_string .. string.char(random_number)
  end
  return random_string
end;

function main_function()
  local string_value, t, sql_statement
  for i = 1,1000000,1 do
    string_value = string_function()
    sql_statement = "INSERT INTO tester VALUES (" .. i .. ", '" .. string_value .. "')"
    box.execute(sql_statement)
  end
end;

start_time = os.clock();
main_function();
end_time = os.clock();
'insert done in ' .. end_time - start_time .. ' seconds';
```

Рисунок 2.3 – Приклад написання програми для зберігання інформації в RAM

Переваги:

- швидке проектування бази даних такого типу;
- швидка обробка даних та швидке їх зчитування.

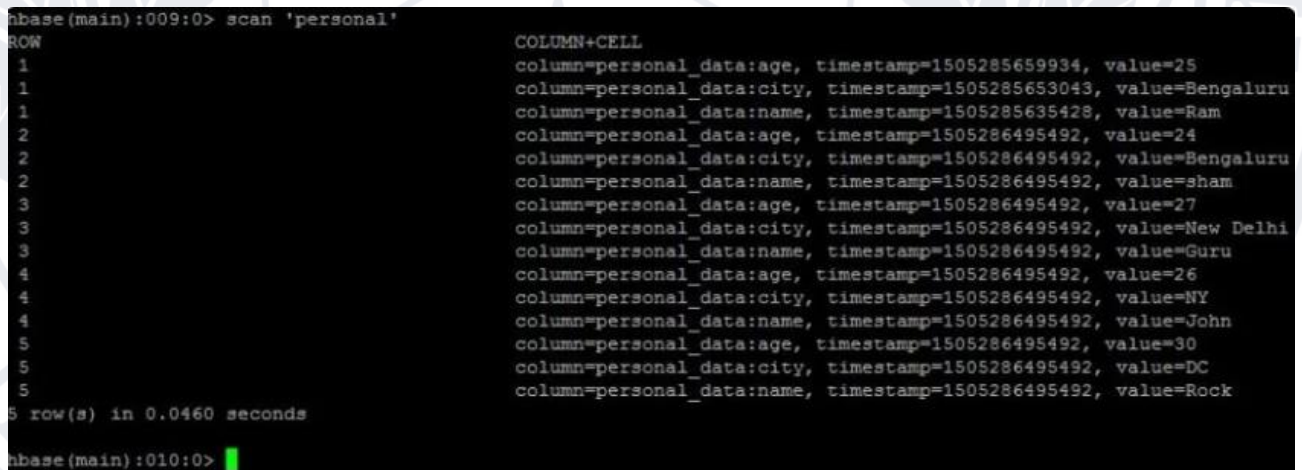
Недоліки:

- низька надійність;
- важко використовувати(тільки для певних цілей).

Приклади: Redis, Apache Ignite.

Бази даних з широкими стовпцями

Даний тип баз зберігають дані на жорсткому диску, або твердотільному диску у вигляді записів «ключ-значення». Ця модель розроблена якраз для цілей керування інформації у великих об'ємах. Зазвичай через такі бази даних проходять тисячі терабайт інформації на тисячах спільних серверів у розподіленій системі стабільної архітектури. Ця архітектура була створена для двох випадків використання: швидкого доступу до ключів і швидкого запису.



```

hbase(main):009:0> scan 'personal'
ROW                                COLUMN+CELL
1      column=personal_data:age, timestamp=1505285659934, value=25
1      column=personal_data:city, timestamp=1505285653043, value=Bengaluru
1      column=personal_data:name, timestamp=1505285635428, value=Ram
2      column=personal_data:age, timestamp=1505286495492, value=24
2      column=personal_data:city, timestamp=1505286495492, value=Bengaluru
2      column=personal_data:name, timestamp=1505286495492, value=sham
3      column=personal_data:age, timestamp=1505286495492, value=27
3      column=personal_data:city, timestamp=1505286495492, value=New Delhi
3      column=personal_data:name, timestamp=1505286495492, value=Guru
4      column=personal_data:age, timestamp=1505286495492, value=26
4      column=personal_data:city, timestamp=1505286495492, value=NY
4      column=personal_data:name, timestamp=1505286495492, value=John
5      column=personal_data:age, timestamp=1505286495492, value=30
5      column=personal_data:city, timestamp=1505286495492, value=DC
5      column=personal_data:name, timestamp=1505286495492, value=Rock
5 row(s) in 0.0460 seconds
hbase(main):010:0>
  
```

Рисунок 2.4 – Приклад роботи бази даних з широкими стовпцями

Переваги:

- швидкий вхід рядок за рядком;
- швидке зчитування клавіш;
- хороша масштабованість, для цього ці бази були створені;

- висока доступність.

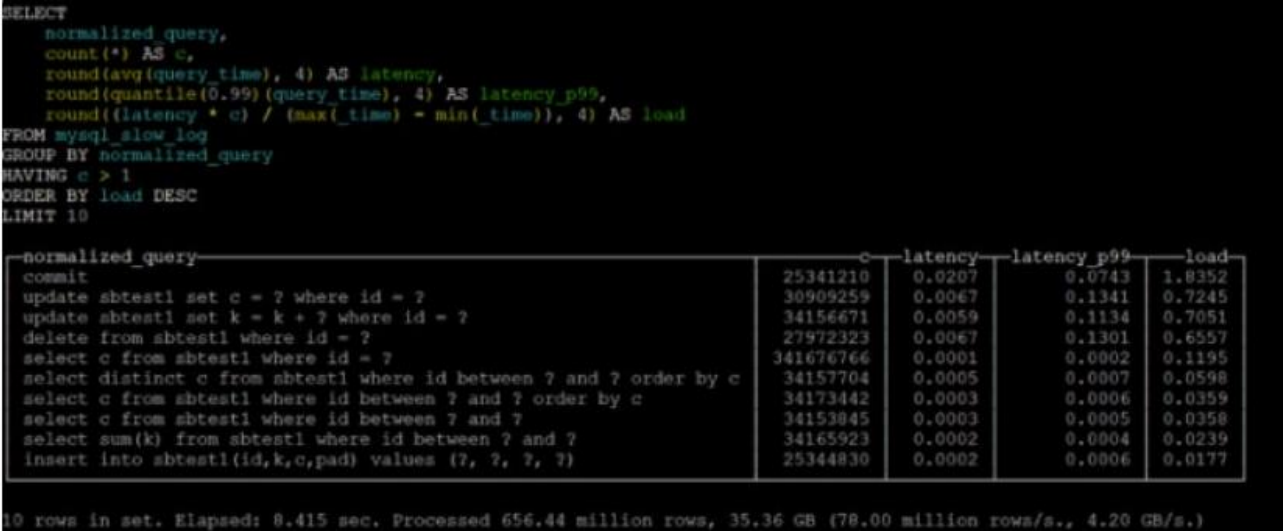
Недоліки:

- формат «ключ-значення»;
- немає аналітичної підтримки та вибірки.

Приклади: Cassandra, HBase.

Стовпчасті бази даних

Стовпчасті бази даних використовуються в тих випадках, коли дані потрібно отримати не за допомогою ключів, а за допомогою конкретних стовпців. У такому випадку краще не використовувати построчкові вставки, а використовувати пакетний запис. Такий тип запису дозволяє стовпчастим базам підготувати дані для швидкого читання по стовпцях.



```
SELECT
  normalized_query,
  count(*) AS c,
  round(avg(query_time), 4) AS latency,
  round(quantile(0.99)(query_time), 4) AS latency_p99,
  round((latency * c) / (max(_time) - min(_time)), 4) AS load
FROM mysql_slow_log
GROUP BY normalized_query
HAVING c > 1
ORDER BY load DESC
LIMIT 10
```

normalized_query	c	latency	latency_p99	load
commit	25341210	0.0207	0.0743	1.8352
update sbtest1 set c = ? where id = ?	30909259	0.0067	0.1341	0.7245
update sbtest1 set k = k + ? where id = ?	34156671	0.0059	0.1134	0.7051
delete from sbtest1 where id = ?	27972323	0.0067	0.1301	0.6557
select c from sbtest1 where id = ?	341676766	0.0001	0.0002	0.1195
select distinct c from sbtest1 where id between ? and ? order by c	34157704	0.0005	0.0007	0.0598
select c from sbtest1 where id between ? and ? order by c	34173442	0.0003	0.0006	0.0359
select c from sbtest1 where id between ? and ?	34153045	0.0003	0.0005	0.0358
select sum(k) from sbtest1 where id between ? and ?	34165923	0.0002	0.0004	0.0239
insert into sbtest1(id,k,c,pad) values (?, ?, ?, ?)	25344830	0.0002	0.0006	0.0177

10 rows in set. Elapsed: 8.415 sec. Processed 656.44 million rows, 35.36 GB (78.00 million rows/s., 4.20 GB/s.)

Рисунок 2.5 – Приклад використання стовпчастої бази даних

Переваги:

- швидке читання колонки за колонкою;
- наявна хороша аналітична підтримка;
- великий потенціал для масштабованості.

Недоліки:

- підходить тільки для пакетних вставок.

Приклади: Vertical, Clickhouse.

Графічні бази даних

Графічні бази даних використовують в графічному представленні систему графів. Це дуже специфічні бази, які використовуються тільки в вузьких спеціалізаціях, але натомість вони існують і їх також використовують.

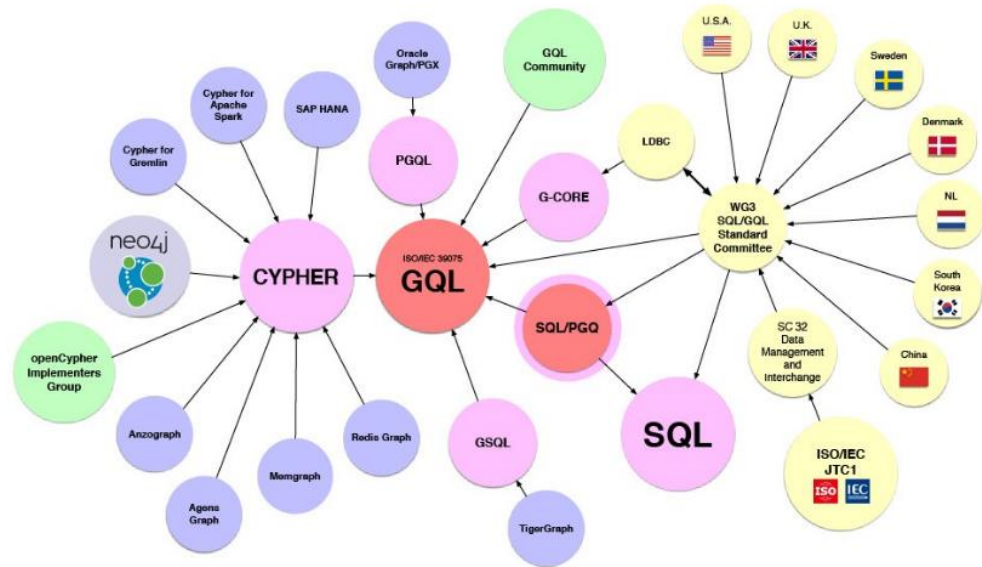


Рисунок 2.6 – Приклад бази даних у Neo4j[8]

Переваги:

- структура даних у графовому представленні;
- керовані відношення між сутностями;
- гнучкі конструкції.

Недоліки:

- спеціальна мова запитів;
- важко спроектувати масштабну базу.

Приклади: Neo4j.

Важливо правильно вибрати тип бази за призначенням, тому що це збереже певну кількість часу і якщо говорити про готовий продукт на ринку, то збереже кошти на проектуванні.

Розглянувши майже усі сучасні моделі баз даних, проаналізувавши переваги та недоліки кожної з них, можна зробити висновок, що універсальним інструментом у цьому випадку буде реляційна модель бази даних. Вона має велику кількість переваг та незначні недоліки. Також для вирішення задачі можливо використовувати документо-орієнтований тип, але реляційна модель більш поширена та вже буде готова для використання у різних підприємствах.

2.2 Опис використаної система управління(СУБД)

Так як база даних буде створюватись на основі реляційної моделі, вибір системи управління обмежений декількома продуктами, а саме: MySQL, PostgreSQL, Oracle, MS Access.

Система управління дозволяє користувачам розробляти персоналізовані бази даних для задоволення своїх потреб у інформаційному обліку. Більшість систем управління дозволяють користувачам виконувати різні операції(створення бази даних, зберігання даних, оновлення або отримання даних за допомогою SQL запитів).

Опираючись на вище розглянуті операції система управління бажано має задовільняти наступні функції:

- Нормалізація даних – оскільки через великі об'єми різної інформації ризик дублювання тільки зростає, так як кілька користувачів діляться нею одночасно. Нормалізація даних значно зменшує ризик виникнення даної проблеми і мінімізує збитки після виявлення конфлікту. З цього можна зробити висновок, що нормалізація також впливає на швидкодію роботи системи управління;
- Визначені правила для користувача – визначені та обмежені правила для користувача значно зменшують ризик пошкодження бази та даних, які вона містила;
- Протоколи безпеки – деякі системи управління мають функцію шифрування даних, що значно підвищує рівень захищеності бази;

- Структурування даних – система управління повинна упорядковувати інформацію у базі даних у чіткій ієрархічній послідовності, також таку можливість має і користувач. Це означає, що всі об'єкти, записи та таблиці мають бути впорядковані. За допомогою цього до записів можна отримати доступ легко та швидко.

Варто зазначити, що система управління має бути об'єкто-орієнтованою. Це тип рішення для управління даними, де сутності представлені в об'єктах і зберігаються в пам'яті. Вона забезпечує єдине середовище програмування і сумісний з різними мовами програмування, включаючи Java, C#, тощо...

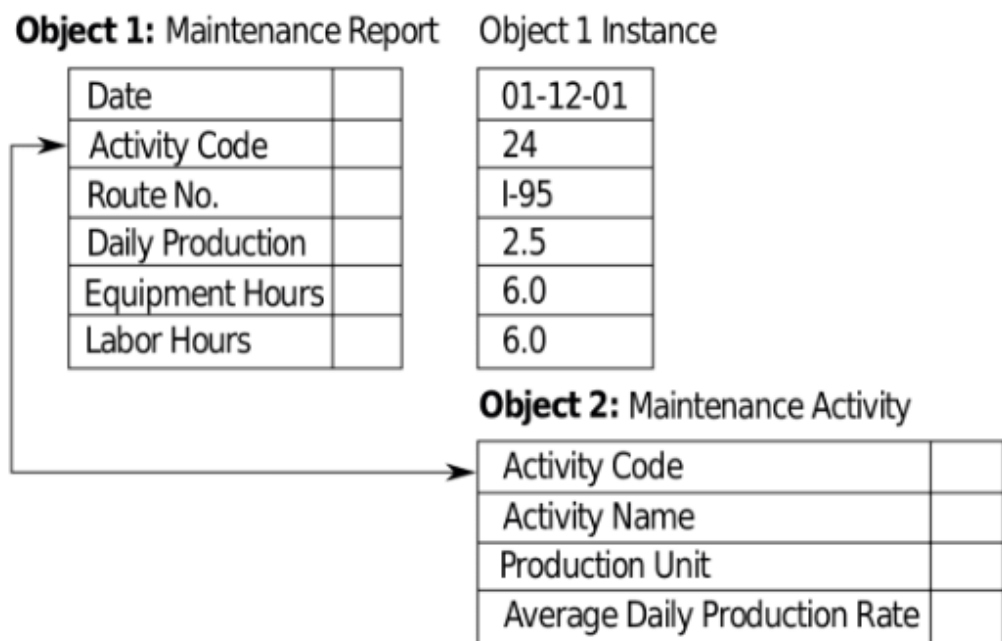


Рисунок 2.7 – Приклад об'єктно-орієнтованої бази даних[9]

Зважаючи на те, що програмний код для взаємодії з базою даних буде написаний на певній об'єктно-орієнтованій мові програмування, вибір звужується до двох найпопулярніших представників у цій сфері: MySQL, PostgreSQL.

MySQL і PostgreSQL є основними джерелами баз даних та найпопулярнішими. Дві системи мають бази з відкритим вихідним кодом, що значно полегшує роботи з ними. На даних системах існує незлічена кількість великих комерційних проєктів і їх актуальність зростає з кожним роком. Все ж таки обидві системи мають ряд характеристик, які відрізняють їх одна від одної.

PostgreSQL – це повністю об’єктно-реляційна база даних, в той час коли MySQL є виключено реляційною. З даної інформації можливо зробити висновок, що PostgreSQL дозволяє працювати з наступними особливостями: можливість роботи з більш складними типами даних, дозвіл об’єктам успадковувати властивості, що є достатньо важливо.

Програми кардинально відрізняються у виділенні пам’яті. PostgreSQL створює новий системний процес з виділенням пам’яті для кожного клієнтського з’єднання, яке було встановлене. Тому він вимагає великої кількості пам’яті в системах з великою кількістю клієнтських з’єднань. В той час як MySQL підтримує тільки один процес та потік на з’єднання. В цьому випадку MySQL більше підходить для невеликих додатків, а не для великих комерційних корпоративних проєктів.

Що стосується безпеки, індексації, підтримки та документації, загалом кожна з систем має схожий функціонал і можливості. PostgreSQL і MySQL можуть наділяти користувачів, або групу користувачів певними правами.

PostgreSQL має такі функції, як розбиття на розділи таблиць та перевантаження функцій, і, як правило, краще справляється з одночасністю. SQL, з іншого боку, краще для простих навантажень і, як правило, використовує більше пам’яті.

Варто зазначити про підтримку спільноти кожної з систем, тому як вона має важливе значення для вдосконалення будь-якої системи. PostgreSQL має дуже активну та велику спільноту, яка постійно впливає на оновлення та вдосконалення програмного забезпечення. Звідси випливає, що PostgreSQL завжди слідує теперішньому вектору розвитку баз даних та займає місце

сучасної, багатфункціональної системи. Також завдяки такій обширній спільноті пошук потрібної інформації не є часо та трудозатратним. Вирішення багатьох популярних проблем є у вільному доступі, що значно полегшує роботу з системою управління.

MySQL теж має перелічені переваги, але в відчутно меншій кількості. Оновлення версій, усунення багів то додавання нового функціоналу займає більше часу ти виходить не часто.

Опираючись на всі вище викладені факти, вибір в якості системи управління для бази даних був зроблений у користь PostgreSQL.

2.3 Мова програмування C# та інструменти для роботи з базою даних

Для реалізації необхідних функцій було обрано мову програмування C#. Дана мова програмування містить всі необхідні технології та компоненти для комфортної роботи з базами даних.

C# є мовою компонентного програмування, і в цьому одна з головних переваг мови, спрямована на можливість повторного використання створених компонентів. C# створювалась паралельно з каркасом .Net і повною мірою враховує всі його можливості. C# є повністю об'єктно-орієнтованою мовою, де навіть типи, вбудовані в мову, представлені класами. C# є потужною об'єктною мовою з можливостями спадкування й універсалізації. C# є спадкоємцем мов C/C++, зберігаючи кращі риси цих популярних мов програмування. Завдяки каркасу .Net, що стали надбудовою над операційною системою, програмісти C# одержують ті ж переваги роботи з віртуальною машиною, що й програмісти Java[10].

Корпорацією Microsoft запропоновано новаторський компонентно-орієнтований підхід до програмування, який є розвитком об'єктно-орієнтованого спрямування. Відповідно до цього підходу, інтеграція об'єктів проводиться на основі інтерфейсів, що представляють ці об'єкти як незалежні компоненти. Такий підхід суттєво полегшує написання та взаємодія програмних компонент у

середовищі проектування та реалізації. Стандартизується зберігання і повторне використання компонент програмного проекту в умовах розподіленої мережевої середовища обчислень, де різні комп'ютери і користувачі обмінюються інформацією, наприклад, взаємодіючи в рамках дослідницького або бізнес-проекту. Істотною перевагою слід вважати і можливість практичної реалізації принципу "будь-яка сутність є об'єктом" в програмному середовищі. Говорячи про .NET як про технологічну платформу, слід відзначити той факт, що вона забезпечує одночасну підтримку проектування та реалізацію програмного забезпечення з використанням різних мов програмування. При цьому підтримуються десятки мов програмування, починаючи від найперших (зокрема, COBOL і FORTRAN) і закінчуючи сучасними (наприклад, C #).

.Net є одною з найбільш широко використовуваних технологій для створення спеціальних програмних додатків. Це є високопродуктивним сучасним хмарним фреймворком для створення корпоративних додатків, який при цьому має відкрий вихідний код. Важливо зазначити, що .Net має сучасні мовні конструкції(LINQ), які значно полегшують роботу розробника. Також технологія є мультиплатформенною. Це означає, що розробники можуть повторно використовувати навички та код на будь якій платформі у знайомому собі середовищі. Додатки написані на .Net мають відмінну продуктивність на відмінну наприклад від Java. Додатки мають кращий час відгуку та використовують менше обчислювальної потужності. Платформу офіційно підтримується компанією Microsoft, компанія суворо відноситься до питань безпеки розроблених додатків на своїй платформі. Тому швидко випускає виправлення багів та оновлення. Спільнота розробників на платформі .Net нараховує понад 5 000 000 розробників, що дає можливість використовувати велику екосистему.

Entity Framework

Однією з важливих використаних технологій, яка потребує опису є – Entity Framework. Так як код для взаємодії з базою даних може бути доволі громіздким, фреймворк допомагає його значно зменшити, тим самим полегшивши роботу.

Entity Framework – це фреймворк, який був створений для допомоги в роботі з базами даних. Він використовується у мові програмування C#, а саме у надбудові, яка була розглянута вище - .Net. Фреймворк дозволяє працювати з базою за допомогою сутностей, а не таблиць. Код з використанням даного фреймворку пишеться набагато швидше і легше[11].

Коли розробник працює з базами даних. Він повинен виконати велику кількість певних дій, для вирішення задачі(забезпечити підключення, підготовка різних SQL параметрів, відправка запитів та транзакції). Ці всі дії Entity Framework робить автоматично, при цьому не урізаючи функціонал. Розробнику так само доступні функції використання:

- зовнішніх та внутрішніх ключів;
- зв'язки (один-до-багатьох, багато-до-багатьох);
- параметризація запитів;
- Збережені процедури.

Майже усі таблиці бази даних є визначеними у фреймворку, як моделі класів сутностей. Це називається конвенціями і вони визначаються в класі контексту даних як набори.

Під час довгої розробки може виникнути ситуація, коли клас моделі змінився і потрібно видалити, або змінити базу для підтримки відповідності моделі. Однак в результаті виконання даного алгоритму дані теж видаляються. Щоб зберегти дані під час змінення моделі у даному фреймворку наявна функція міграцій. Це дозволяє послідовно застосовувати зміни схеми до бази для синхронізації з моделлю.

Але слід зазначити, що Entity Framework є додатковим рівнем між додатком та базою даних, що безпосередньо може позначитись на

продуктивності. Це не впливово для невеликих проектів, але якщо проект є сильно навантаженим, тоді скоріше за все розробники будуть використовувати чистий .Net.

LINQ

Перш за все цікаво, навіщо був розроблений LINQ. Основна причина була у тому, як додаток бачив реляційні дані. Якщо це стосується звичайного додатку та звичайної бази даних все добре працює і на звичайному .NET. Однак, якщо до бази регулярно додають таблиці, вона збільшується та разом з цим збільшується і пов'язані дані, які зберігались у цих самих таблицях. Після такого зростання додатку уже буде складно робити різні операції з даними, тому що він розцінює їх по іншому. Але все ж програмісти хочуть працювати в одному концептуальному порядку без необхідності робити складні операції об'єднання для кожного запиту, який був пов'язаним. Тому програміст і реалізують різні шари відображення, для створення єдиної сутності, яка представляє собою об'єкт. У LINQ існує зіставлення між схемами даних в об'єкті, а також реляційним домено, що полегшує його використання.

LINQ (Language Integrated Query) – є єдиним синтаксисом запитів у C# і .Net для того, щоб отримати дані з різних джерел і форматів. Даний синтаксис є інтегрованим у C# та допомагає встановити відповідність між мовою програмування і базою даних, а також забезпечує єдиний інтерфейс запиту для різних типів і джерело[12].

SQL є - структурованою мовою запитів, яка в свою чергу використовується для збереження та отримання інформації з баз даних. У схожому вигляді LINQ – це також синтаксис, який уже вбудований в C# . LINQ має змогу отримати дані з різни джерел, а саме: колекцій, DataSet, Xml docs, MS SQL server та інші різні бази даних.

Результатом запиту LINQ – є об'єкт. Завдяки цьому розробник може використовувати об'єктно-орієнтований підхід до результатів запиту та не брати до уваги перетворення різних форматів у об'єкти.

NUGET

NuGet - ключовий інструмент для будь-якої сучасної платформи розробки - це механізм, за допомогою якого розробники можуть створювати, передавати один одному і використовувати корисний код. Часто такий код розподілений по "пакетах", що включає скомпільований код (у вигляді бібліотек DLL) і інший вміст.

Для .NET механізмом спільного використання коду, підтримуваним Майкрософт, є NuGet, який визначає, як створюються, розміщуються і використовуються пакети для .NET.

Отже технологія NuGet є окремим ZIP-файлом з розширенням .nupkg, який містить скомпільований код (DLL), інші файли пов'язані з цим кодом, що включає такі відомості, як номер версії пакета. Розробники, у яких є код, до якого потрібно надати загальний доступ, створюють пакети і публікують їх на закритих або відкритих ресурсах. Користувачі отримують ці пакети з відповідних ресурсів, додають їх у свої проекти, а потім викликають функції пакета в коді свого проекту. При цьому NuGet сам обробляє всі проміжні дані.

Оскільки NuGet підтримує закриті ресурси поряд з відкритим вузлом nuget.org, за допомогою пакетів NuGet, можна ділитися кодом, використовуваним в рамках організації або робочої групи[13]. Іншими словами, пакет NuGet є спільно використаною одиницею коду, проте не вимагає і не має на увазі будь-якого певного способу надання загального доступу.

Бібліотеки .NET Standard є специфічними для платформи в тому сенсі, що вони надають всі функціональні можливості базової платформи (без штучних платформ або пересічний платформ). Вони працюють на всіх підтримуваних платформах.

.NET Standard надає набір бібліотек. Реалізації .NET повинні підтримувати кожну бібліотеку повністю або не підтримувати взагалі. Таким чином, кожна програма підтримує набір стандартних .NET бібліотек. Наслідком з цього є те, що кожна бібліотека класів .NET Standard підтримується на платформах, які підтримують її залежності.

Бібліотеки .NET Standard не пропонують всі функції платформи .NET Framework, але вони надають набагато більше API, ніж переносяться бібліотеки класів. Згодом будуть додані додаткові API.

Типи .NET являють собою основу для створення елементів управління. .NET також містить методи, призначені для виконання таких завдань:

- уявлення базових типів даних і винятків;
- інкапсуляція структур даних;
- операції введення-виведення;
- доступ до даних про завантажених типах;
- виклик перевірок безпеки .NET;
- доступ до даних, надання графічного інтерфейсу користувача на стороні клієнта і керованого сервером графічного призначеного для користувача інтерфейсу на стороні клієнта.

.NET пропонує розширений набір інтерфейсів, а також абстрактних і конкретних класів. Можна використовувати існуючі конкретні класи. Крім того, у багатьох випадках на їх основі можна створювати власні класи. Щоб скористатися наявними можливостями інтерфейсу, можна або створити клас, який реалізує інтерфейс, або створити похідний клас на основі одного з класів .NET, що реалізує інтерфейс.

2.4 Додаткові інструменти для вирішення задачі

Існує велика кількість програмного забезпечення для написання коду. В кожному з середовищ можливо створювати програми, або працювати з базами даних

Для проектування бази даних було використано ряд інструментів, які описані нижче. Серед них інтегровані середовища розробки (Visual Studio, Rider). Docker – програмне забезпечення для автоматизації розгортання та керування програмами в середовищах та контейнеризатор додатків. Data grip для роботи з таблицями у базі даних. Сайт «Heroku» для запуску бази даних на сервері Amazon.

Інтегроване середовище розробки Visual Studio – саме «правильне» середовище розробки. З Visual Studio багато хто починає знайомитися з мовою програмування і не розлучаються з нею протягом всієї кар'єри програміста.

Project Rider – середовище від JetBrains для роботи з платформою .NET. Випущена недавно, але вже має багато шанувальників.

Eclipse – одне з найпопулярніших багатомовних середовищ розробки. Орієнтоване переважно на розробку Java-додатків, але корисне і для кодів на C#.

Code Blocks – середовище розробки, відоме простотою і зручністю в налаштуванні і використанні.

Отже, порівнявши переваги та недоліки існуючих середовищ розробки, було вирішено розробити програму на базі IDE – Visual Studio. Також вибір був оснований на попередньому досвіді.

Visual Studio

Інтегроване середовище розробки Visual Studio – це стартовий майданчик для написання, налагодження і складання коду, а також подальшої публікації додатків. Інтегроване середовище розробки (IDE) являє собою багатофункціональну програму, яку можна використовувати для різних аспектів розробки програмного забезпечення. Крім стандартного редактора і налагоджувача, які існують в більшості середовищ IDE, Visual Studio включає в себе компілятори, засоби автозавершення коду, графічні конструктори і багато інших функцій для спрощення процесу розробки.

Visual Studio має декілька важливих та корисних інструментів:

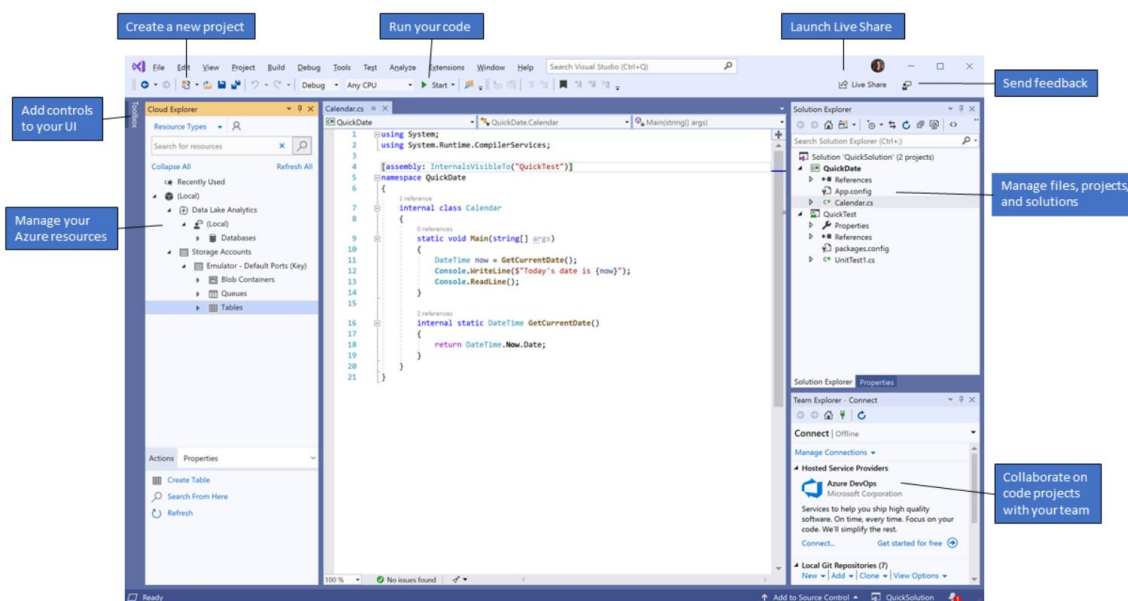


Рисунок 2.8 – Інтерфейс середовища Visual Studio[14]

На рисунку показано інтерфейс середовища Visual Studio з відкритим проектом і декількома вікнами основних інструментів. Які є досить використовуваними:

- оглядач рішень (вгорі праворуч) дозволяє переглядати файли коду, пересуватися по ньому і управляти ними. Оглядач рішень дозволяє впорядкувати код шляхом об'єднання файлів в рішення і проекти;
- у вікні редактора (центр), де швидше за все, користувач буде проводити більшу частину часу, відображається вміст файлу. Тут можна редагувати код, або розробляти призначений для користувача інтерфейс, наприклад вікно з кнопками або текстові поля;
- Team Explorer (правий нижній кут) дозволяє відслідковувати робочі елементи і використовувати код спільно з іншими користувачами за допомогою технологій управління версіями, таких як Git і система управління версіями Team Foundation.

JetBrains Rider

Також при безпосередньо роботі з базою, було використане програмне забезпечення від компанії «JetBrains», інтегроване середовище розробки Rider.

Rider – це повноцінне крос-платформне середовище розробки, яке працює з широким спектром проектів .NET Framework та .NET Core. Він підтримує більшість мов, що використовується у розробці на даних платформах. Rider має декілька особливостей, про які варто зазначити[15].

Глобальний пошук у продукті від «JetBrains» працює набагато краще, ніж Visual Studio. У першому випадку він працює швидко без нарікань і представляє результати набагато простіше і зрозуміліше. У Visual Studio він працює достатньо повільно. У той час як Rider відображає результати пошуку чітко і зі скороченими іменами файлів, Visual Studio відображає результати пошуку як інструмент командного рядка.

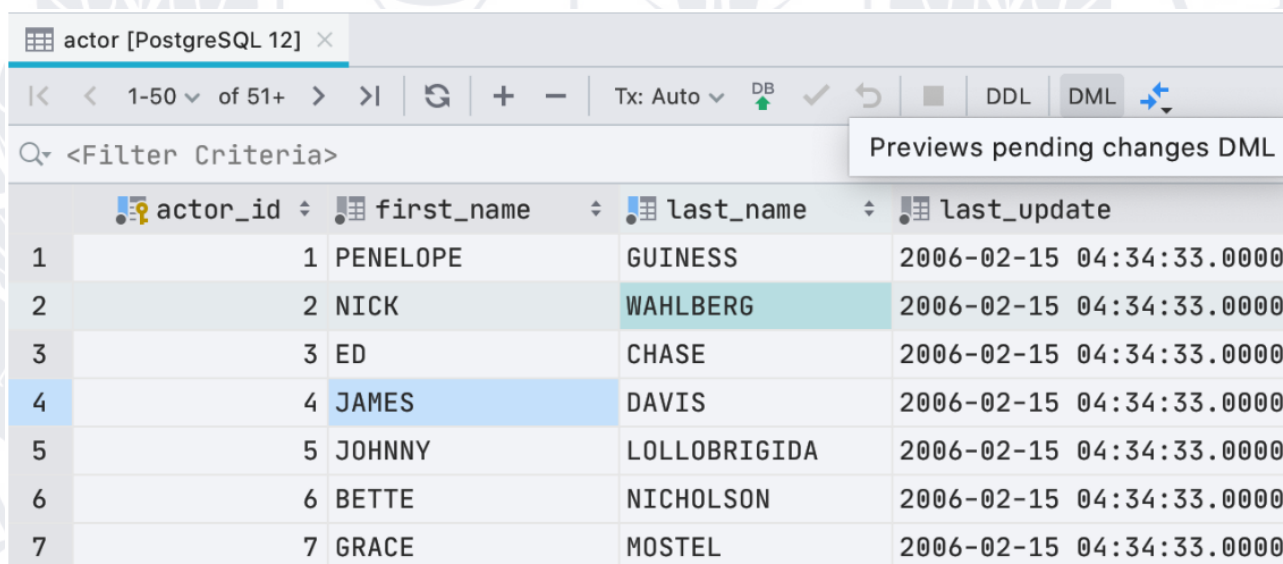
Також відмінність є у аналізатора коду. Він має вирішальне значення, особливо у великомасштабних проектах. Статичний аналіз коду значно зменшує витрачений час на пошук помилок. У кожного середовища розробки є свої правила аналізу коду. У Visual Studio – приблизно 600 правил, в той час як у Rider – понад 1300. Що дозволяє передбачити помилки у багатьох варіаціях та сценаріях.

Visual Studio IDE був одним з перших інструментів, призначених для розробки .NET. Ось чому він підтримує застарілий код .NET і розширення. І хоча Visual Studio спочатку призначалася для розробки на платформах Microsoft, в даний час вона дозволяє користувачам писати програми на інших мовах програмування, таких як Python. Райдер призначений в основному для розробки .NET і, таким чином, не підходить для створення додатків на інших мовах. В цілому, Visual Studio IDE відмінно підходить для компаній, які працюють зі застарілим кодом .NET або хочуть IDE, який підтримує кілька мов програмування.

Data Grip

Data Grip – це середовище керування базами даних для розробників, призначене для запитів, створення та керування базами. Data grip надає інструменти для роботи з об'єктами бази даних. Цей сервіс стане в нагоді, якщо користувач створює або змінює таблиці, додає або змінює поля. Такі зміни супроводжуються генерацією відповідного скрипта.

Також Data Grip виконує необхідні функції для виконання запитів, або для частини запиту, для якого в свою чергу необхідно дістати необхідний код і запустити його. Велика кількість інструментів доступні у вікні редактора таблиць(змінення даних, текстовий пошук. Сусідні вікна використовується для порівняння результатів.



	actor_id	first_name	last_name	last_update
1	1	PENELOPE	GUINNESS	2006-02-15 04:34:33.0000
2	2	NICK	WAHLBERG	2006-02-15 04:34:33.0000
3	3	ED	CHASE	2006-02-15 04:34:33.0000
4	4	JAMES	DAVIS	2006-02-15 04:34:33.0000
5	5	JOHNNY	LOLLOBRIGIDA	2006-02-15 04:34:33.0000
6	6	BETTE	NICHOLSON	2006-02-15 04:34:33.0000
7	7	GRACE	MOSTEL	2006-02-15 04:34:33.0000

Рисунок 2.9 – Приклад використання редактору таблиць у додатку Data Grip[16]

Data Grip має важливі та використовувані інструменти. Автоматичний редактор даних для файлів. Якщо вставляти дані в таблицю з буфера обміну, автоматично створюється необхідна кількість нових рядків.

Редактор SQL – показує усі помилки в запиті та в оброблені сценарію. Автоматично допомога в перейменуванні – якщо перейменувати щось не за допомогою вбудованого рефакторингу, а змінити саме ім'я в програмному коді, то додатком буде запропоноване перейменування усіх використаних видів.

З перелічених переваг, можна зробити висновок, що JetBrains Data Grip – є потужним інструментом для роботи розробника з базами даних. Також цей додаток є використовуваним у багатьох великих компаніях та на різних проектах, тому навички володіння даною програмою є безперечним плюсом.

Docker

Docker – є основою для створення, запуску та управління контейнерами на серверах, або у хмарині з відкритим вихідним кодом. Це надає можливість розробникам упаковувати свої програми в контейнери – стандартизовані компоненти, що поєднують вихідний код програми з бібліотеками операційної системи. Також дана програма поєднує контейнери з залежностями, необхідними для запуску даного в будь-якому середовищі. Це технологія стає все більш популярною, оскільки багато організації використовують хмарну розробку та хмарні сховища.

Платформа значно полегшує та робить безпечним створення даних контейнерів. За допомогою Docker розробник може керувати усіма функціями, використовуючи прості команди.

Варто зазначити, що контейнери стають можливими завдяки віртуалізації, яка вбудована в ядро Linux. Це дозволяє кільком компонентам застосунку ділитися ресурсами одного екземпляра. В результаті контейнера технологія пропонує всі функціональні можливості та переваги віртуальних машин. Менша вага контейнерів, більш ефективно використані ресурси, підвищення продуктивності розробників.

Docker постійно оновлюється та приносить все нові функції та оновлення для своїх користувачів, а саме:

- зменшення розміру об'єктів – дає можливість створювати додаток, який може продовжувати працювати, поки одна з його частин знімається для оновлення;

- автоматичне створення контейнерів – Docker автоматично створює контейнери на основі коду програми;
- версія контейнера – це дозволяє відстежувати версії контейнера, є можливість повернутись до попередньої версії, також відстеження правок користувачів у відповідній версії;
- спільні бібліотеки контейнерів – розробники можуть отримати доступ до реєстру з відкритим кодом, що містить безліч контейнерів, які були внесені користувачем.

На сьогоднішній день контейнеризація також працює з платформою Windows. І більшість хмарних провайдерів пропонують конкретні послуги, щоб допомогти розробникам створювати, відправляти і запускати контейнери Docker[17].

У головному меню Docker перелічені всі запуснені контейнери та програми.

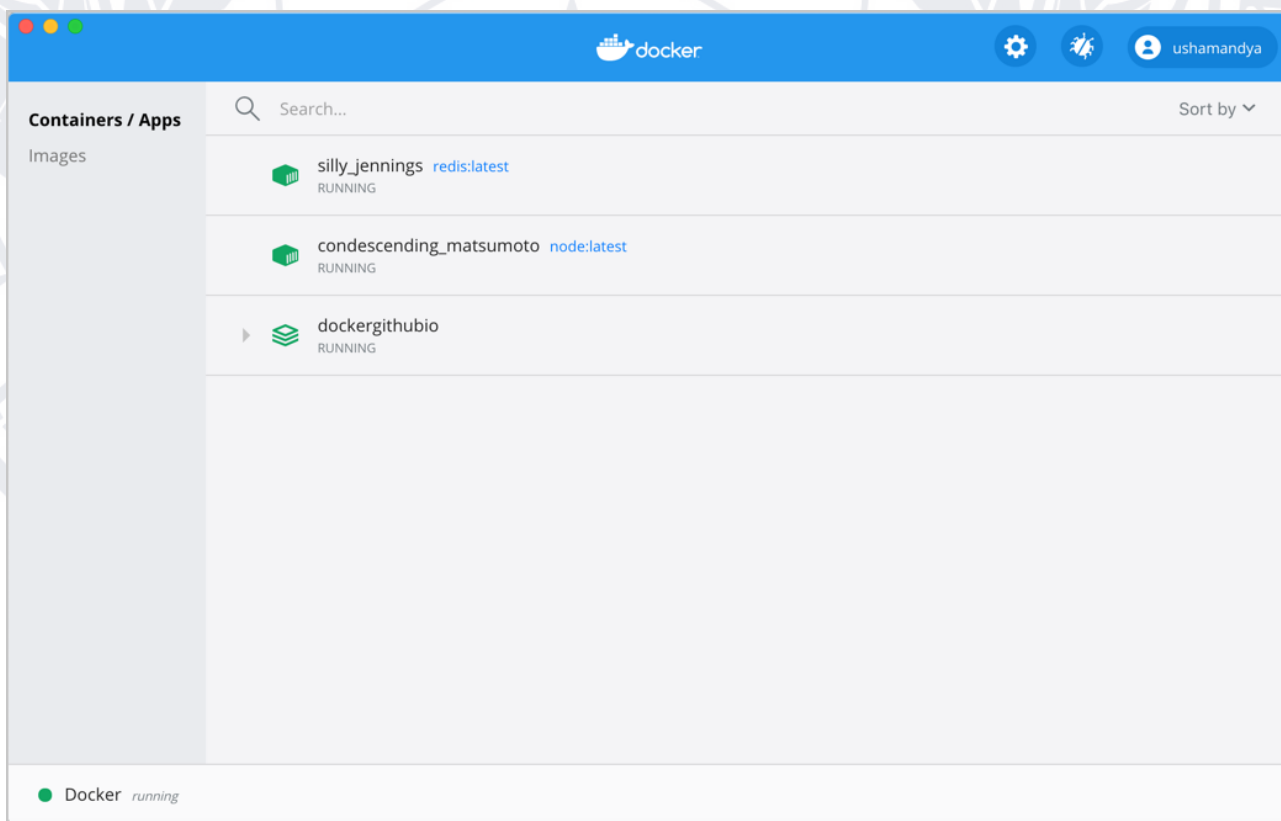


Рисунок 2.10 – Перелік заpusнених контейнерів у Docker

На інформаційній панелі у програмі можливо вибрати контейнер, які потрібно використати. У розділі перелічені усі контейнери, що працюють у наявній програм. Є можливість запуску, зупинення, або видалення програми. У верхній частині рисунку можна помітити поле для пошуку. Воно використовується для пошуку журналів та застосунків для певних подій. Знайдений код пізніше можна використати у будь якому редакторі коду, або інтегрованому середовищі розробки.

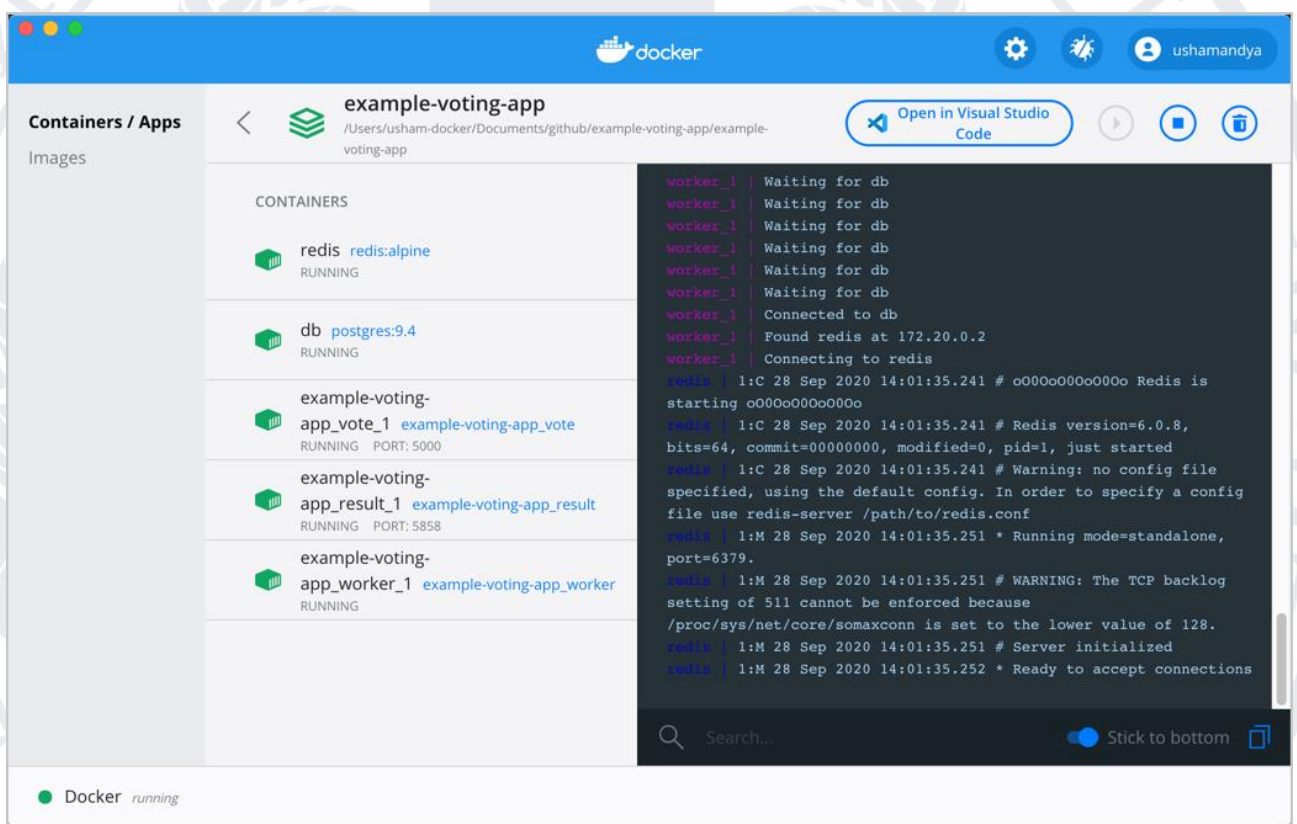


Рисунок 2.11 – Приклад пошуку журналів у додатку Docker

Docker – це революційна технологія, яка спрощує ізоляцію та забезпечує незалежність навколишнього середовища. Однак в його нинішньому вигляді, є можливість використання даного додатку тільки у сферах розробки та тестування. Програмне забезпечення широко застосовується у великих компаніях на різних проектах та є необхідною навичкою для розробника.

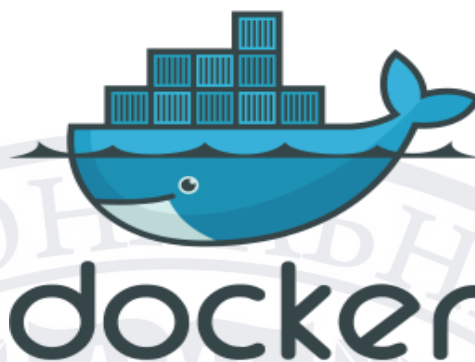


Рисунок 2.12 – Фірмовий логотип додатку Docker[18]

Heroku

Heroku – це хмарна платформа в основі якої лежить контейнеризація як і в додатку Docker. Розробники використовують Heroku для розгортання, керування та масштабування сучасних програм. Платформа дуже приємно оформлена та добре виконує усі свої функції. Heroku надає розробнику свободу зосередитись на основному продукті, не відволікаючись на підтримку серверів, обладнання та інфраструктуру. Сервіс надає послуги, інструменти, робочі процеси та підтримку – все це призначене для підвищення продуктивності розробників[19].

Heroku має потужний та великий набір функцій для розробника, а саме:

- Runtime – програми працюють всередині контейнерів у керованому середовищі виконання та забезпечує оброблення усіх функцій;
- Heroku PostgreSQL – сервіс базується на системі управління базами даних Postgre, що в свою чергу є простим налаштуванням, безперервним захистом даних;
- Додаткові компоненти – Heroku також працює з різними сервісами та системами управління(MongoDB, MySQL, New Relic...);
- Відкат коду та даних – сервіс дозволяє безпечно працювати з кодом та даними, якщо стався збій є можливість відкату до попередньої версії та відновлення даних, також це стосується бази данх;

- Розумні контейнери – усі програми працюють у розумних контейнерах dynos, у цих контейнерах система та мовні стеки повністю контролюються та оновлюються сервісом;
- Інтеграція GitHub – це означає, що кожен запит обертає одноразовий додаток для огляду і тестування. І будь який репозиторій можна налаштувати для автоматичного розгортання у кожній доступній гілці на GitHub[17];
- Зменшення розміру програм – сервіс добре масштабує усі проекти, що в свою чергу дозволяє комфортно працювати як і з малими додатками так і з корпоративною електронною комерцією.

Висновок до розділу 2

У другому розділі були розглянуті усі інструменти для вирішення задачі. Спершу розглянуті типи бази даних, та вибір підходящої для обліку та моніторингу цін на продукти, також описаний кожен тип, його недоліки та переваги. Наявна аргументація вибору системи управління базами даних та порівняння зі схожим програмним забезпеченням. Вибір був зроблений у користь PostgreSQL. Також наявний аналіз мови програмування, яка була використана для виконання поставленої задачі, огляд альтернативних рішень, переваги та недоліки. Описані використані технології та бібліотеки на базі обраної мови програмування, а саме: LINQ, NUGET, Entity Framework. У другому розділі були розглянуті додаткові інструменти, які були використані. До них входять: інтегровані середовища розробки(Visual Studio, Rider), програмне забезпечення для зручності роботи з таблицями у базі даних(Data Grip), інструменти розгортання програм та розміщення бази даних на певному сервері(Docker, Heroku). В описі усіх використаних інструментів розглянута проблематика актуальності та доцільності використання.

РОЗДІЛ 3

ПОКРОКОВИЙ ПРОЦЕС СТВОРЕННЯ БАЗИ ДАНИХ

3.1 Аналіз предметної області та створення схеми бази даних

Створення бази даних починається з аналізу предметної області, кількості потрібних таблиць та полів. Ціллю бази даних є облік та моніторинг цін на продукти.

Кожна організація, або магазин, які розповсюджують продукти мають відповідно постачальника, який надає ці самі продукти. Роль постачальника у базі даних виконує країна. Зазвичай у кожного магазину наявний склад на якому є можливість переглянути кількість продуктів. Усі операції з продуктами мають документаційну основу, щоб засвідчити, що дана операція була проведена. У базі даних цю роль виконують накладні, маючи у таблиці усі необхідні поля.

Проектування кожної бази, будь-то комерційний проект для обліку великої кількості даних та співробітників, або локальна база невеликої спільноти починається зі створення схеми. Схеми бази даних відіграють важливу роль у процесі проектування. Вони дають графічне та зрозуміле представлення самої структури. Завдяки схемі на початку проектування є можливість виявити велику кількість помилок(неправильно побудовані зв'язки, невідповідність полів, невірні типи полів, тощо...) та виправити їх вчасно, що в свою чергу збереже багато часу та грошей.

Дана база не є виключенням та також має структуровану схему.

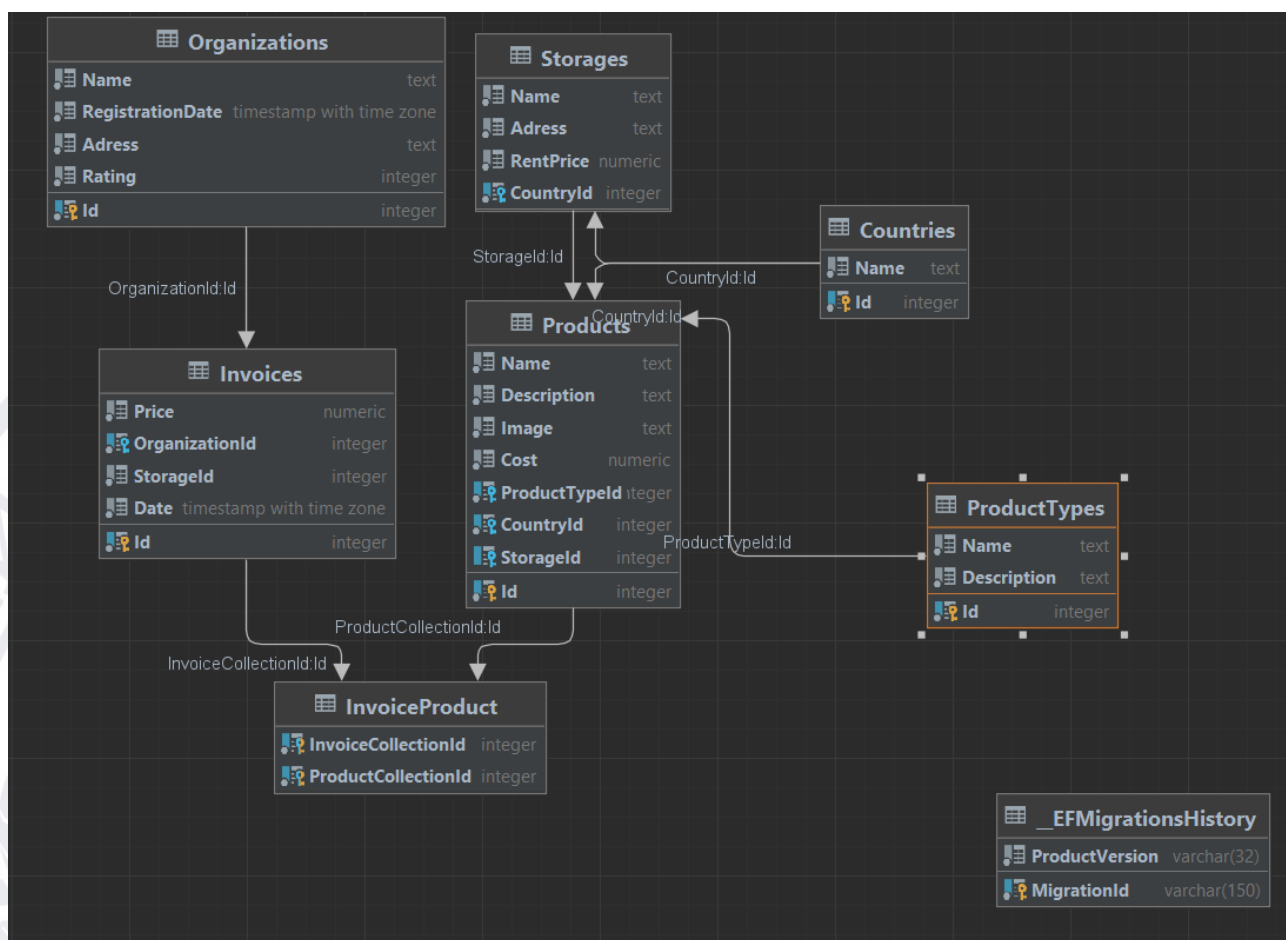


Рисунок 3.1 – Схема бази даних для обліку та моніторингу цін на продукти

Відповідно на рисунку 3.1 можна спостерігати, створену схему з заповненими таблицями та спроектованими зв'язками. Були створені основні таблиці: «Products», «Countries», «ProductTypes», «Invoices», «Organisation», «Storage».

3.2 Проектування таблиць, полів та зв'язків

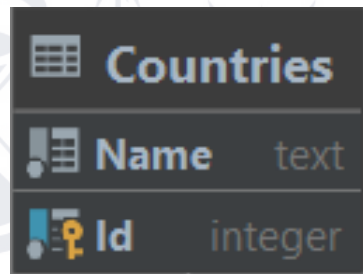
Таблиці є важливим об'єктом у базі, саме вони зберігають усі відомості і дані. Кожна таблиця має назву відповідною її об'єкту, або предметній області. Інші об'єкти бази даних доволі залежні від структури таблиць, тому розробку бази даних завжди слід почитати зі створення таблиць і лише після цього можна переходити до створення інших об'єктів.

Кожна таблиця містить у собі певну кількість полів, яка характеризує

певний об'єкт. Наявність полів є обов'язковою. Кількість атрибутів залежить від того на скільки детального опису потребує об'єкт. Також є декілька правил нормалізації таблиць, а саме:

- кожна таблиця не містить в собі повторюваних полів;
- кожна таблиця має унікальний ідентифікатор (первинний ключ);
- різного роду зміна інформації не повинна впливати на інформацію в інших полях.

Опираючись на вище описані пункти, були створені відповідні таблиці для обліку та моніторингу цін на продукти.



Countries	
Name	text
Id	integer

Рисунок 3.2 – Таблиця країни - постачальника продуктів

На рисунку 3.2 зображено таблицю країни, яка постачає продукти до певної організації. Таблиця має лише два поля, але також є можливість розширити кількість полів для більш детального опису. У об'єкта наявний первинний ключ та назва країни. Поля мають свій тип, який зазначено правіше від назви. Поле «Name» є текстовим полем, що в свою чергу означає, що поле представляє собою строкове значення. У мові C# - це тип даних «String». Наступне поле «Id» - є внутрішнім ключем, яке самостійно генерується. Поле має цілочисельний тип даних «Integer». Таблиця створена згідно з правилами нормалізації.

Name	text
RegistrationDate	timestamp with time zone
Adress	text
Rating	integer
Id	integer

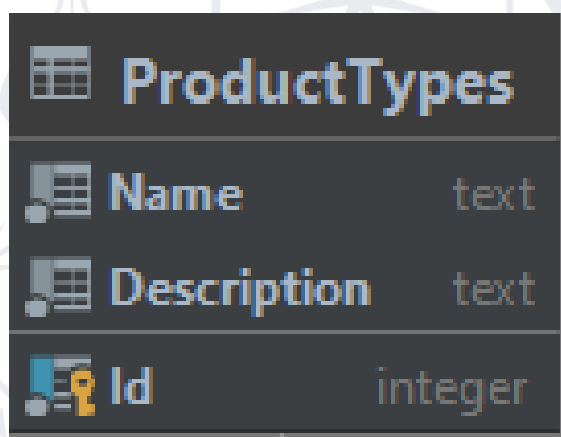
Рисунок 3.3 – Таблиця організації, яка отримує і надалі розповсюджує продукти

На рисунку 3.3 зображено таблиця організації, яка приймає та займається розповсюдженням продуктів. Таблиця має дещо більше полів ніж таблиця країн. Відповідно має первинний ключ, що є обов’язковим, назву організації, дату реєстрації, адресу та рейтинг. Дата реєстрація та рейтинг є додатковими полями, які більше детально характеризують об’єкт організації. Поля «Name» і «Adress» є текстовим полем, що в свою чергу означає, що поле представляє собою строкове значення (String). Поле «Rating» - цілочисельним полем для оцінки організації. «RegistrationDate» - є особливим атрибутом, який має вигляд дати: день – місяць – рік.

Name	text
Adress	text
RentPrice	numeric
CountryId	integer
Id	integer

Рисунок 3.4 – Таблиця опису об’єкту «Склад»

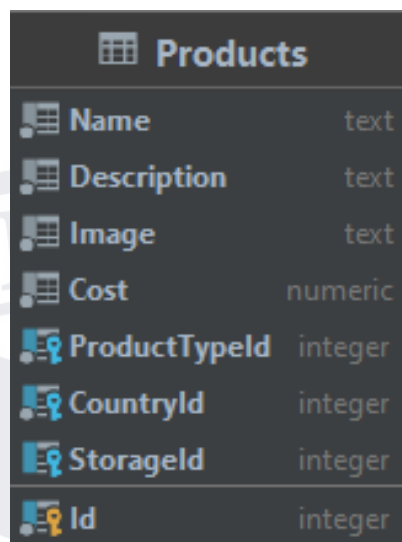
Таблиця склад описує місце, де зберігаються продукти певної організації, також на склад прибувають продукти з країни-постачальника. Тому особистих атрибутів у таблиці тільки два. Поля «Name» та «Adress» є строковими полями, та зазначаються типом даних «String». Таблиця має одне нове поле з типом даних, який раніше не був зазначений. «RentPrice» - ціна оренди складського приміщення. Так як ціна може бути не цілою, використання цілочисельного типу даних не є можливим. Тому було використано тип даних «Decimal», число з плаваючою комою. Як і в попередніх таблицях, склад має свій унікальний ідентифікатор. На відмінну від попередніх, дана таблиця має один зовнішній ключ, що є унікальним ідентифікатором для іншого об'єкту, в даному випадку це країна-постачальник.



ProductTypes	
Name	text
Description	text
Id	integer

Рисунок 3.5 Таблиця для визначення типу продукту

Продукти є різних типів і для того, щоб розрізнити та відформатувати дані була створена таблиця для визначення типу продукту. Вона містить у собі три поля, одне з яких є унікальним внутрішнім ключем. Поля «Name» та «Discription» вказують назву та опис об'єкту. Ці два поля визначається строковим представленням, типу «String».

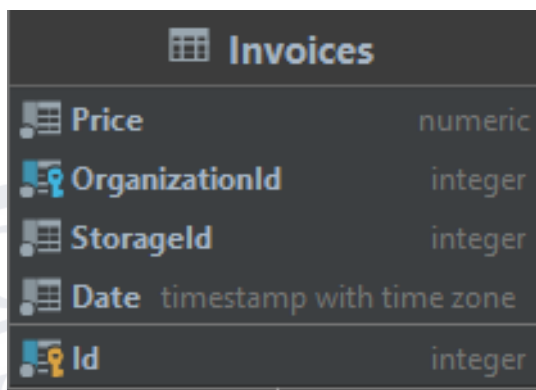


Products	
Name	text
Description	text
Image	text
Cost	numeric
ProductTypeId	integer
CountryId	integer
StorageId	integer
Id	integer

Рисунок 3.6 – Таблиця об'єкту продукти

Таблиця «Products» є одною з основних таблиць, яка має достатню кількість полів з різними типами даних. Важливою особливістю є те, що окрім унікального ідентифікатора у таблиці наявні декілька зовнішніх ключів для зв'язку таблиць між собою.

На рисунку 3.6 три поля у таблиці явно виділені символом блакитного ключа, що характеризує, що дані атрибути є зовнішніми ключами інших об'єктів. «ProductTypeId», «CountryId» та «StorageId» - зовнішні ключі, які є цілочисельними типами. «Name» - ім'я продукту. Поле «Description» дає можливість надати опис певному об'єкту, або зробити замітку. «Image» - поле для більш детального опису об'єкту, шляхом додавання зображення. В даному випадку поле має тип «String», це пов'язано з тим, що у мові програмування C# є можливість вказати посилання на зображення у вигляді строкового представлення, в результаті чого можна надалі його використовувати для різних цілей. Як і в попередній таблиці наявний тип даних «Decimal», який використовується для позначення ціни продукту, що є однією з основних цілей проектування бази даних.



Invoices	
Price	numeric
OrganizationId	integer
StorageId	integer
Date	timestamp with time zone
Id	integer

Рисунок 3.7 – Таблиця накладних

Як зазначалось попередньо для документованого підтвердження про отримання, або відправку продуктів у базі даних використовується накладні, відповідно таблиця «Invoices». Вона має унікальний ідентифікатор, та дату створення, яка описується уже знайомим способом та типом даних. Атрибут, який свідчить про ціну угоди також визначається типом даних «Decimal». Інші поля є зовнішніми ключами, які використовуються для поєднання таблиць з різними відношеннями.

Проектування зв'язків

Правильне проектування зв'язків у базі даних відіграє ключову роль, так як неправильно погодження призведе до непрацездатності продукту. Зв'язок між таблицями встановлюється завдяки двом загальним полям у них.

Важливим є наявність того факту, що є декілька типів логічних зв'язків між таблицями:

- один-до-одного – кожному запису з однієї таблиці відповідає один запис у іншій;
- один-до-багатьох – кожному запису з однієї таблиці відповідає кілька записів у іншій;
- багато-до-одного – безліч записів з однієї таблиці відповідає один запис у іншій таблиці;
- багато-до-багатьох – безліч записів з однієї таблиці відповідає безліч

записів з іншої.

Проаналізувавши рисунок 3.1 можна зробити висновок, щодо проектування зв'язків.

«Organisation» → «Invoices» - має тип зв'язку один-до-багатьох. Так як одна організація проводила не одну операцій з продуктами, тому має багато накладних, які свідчать про це.

«ProductType» → «Product» - тип зв'язку один-до-багатьох. Один тип продукту, може містити в собі велику кількість різних продуктів, які в свою чергу є об'єктами в таблиці.

«Country» → «Product» - тип зв'язку один-до-багатьох. Тип зв'язку між двома таблицями означає, що одна країна має можливість постачати безліч продуктів.

«Country» → «Storage» - тип зв'язку один-до-багатьох. Також таблиця країн має не один зв'язок даного типу. Як і у випадку з продуктами країна може постачає безліч продуктів до багатьох складів.

«Invoices» ↔ «Products» - тип зв'язку багато-до-багатьох. Цей тип зв'язку формується іншим способом на відмінну від один-до-багатьох. Фактично це є два відношення один-до-багатьох з додаванням третьої таблиці. Ключ нової таблиці складається з двох полів зовнішніх ключів інших таблиць. У даному випадку є багато накладних і багато продуктів, які пов'язані з ними.

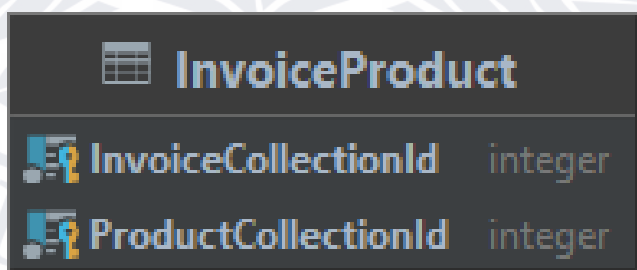


Рисунок 3.7 – Додавання нової таблиці для зв'язку типу «багато-до-багатьох»

Таблиця «InvoiceProduct» - це додаткова таблиця, яка була створена з ціллю з'єднання двох інших таблиць. Вона містить усього два поля, які є

зовнішніми ключами таблиць «Invoice» та «Product».

«Product» ↔ «Storage» - тип логічного зв'язку багато-до-багатьох. Схожим способом поєднані дані дві таблиці. Так як безліч продуктів може міститись на багатьох складах. Додаткова таблиця має два поля, які є зовнішніми ключами зазначених вище таблиць.

3.3 Створення пустої бази даних та розгортання на сторонньому сервері

Для початку, щоб повноцінно користуватись базою даних її потрібно створити та розгорнути на певному сервері. Створення пустої бази не є трудозатратною задачею. Натомість деплой, або розгортання на певному сервері – задача більш високої складності. Метою розгортання бази є її тестування на сервері, а також перевірка бази на працездатність. Так як ціллю даного проектування було не лише локальна обробка даних, розгортання є обов'язковим.

Для вирішення даної задачі, був обраний додаток Docker. На офіційному сайті було зроблено реєстрацію, та приступлено до створення контейнера для програмного коду. Але вже на цьому етапі виникли певні проблеми пов'язані з технічними характеристиками процесору. Також були налаштовані усі параметри, які впливають на цей процес, в тому числі зроблені і в BIOS. Взагалі це проблема є поширеною у багатьох розробників. Так як від самого початку Docker був розроблений для користувачів Linux, а вже пізніше вийшов додаток на Windows. У другому випадку платформа Windows має досить вузький та малий потенціал для контейнеризації

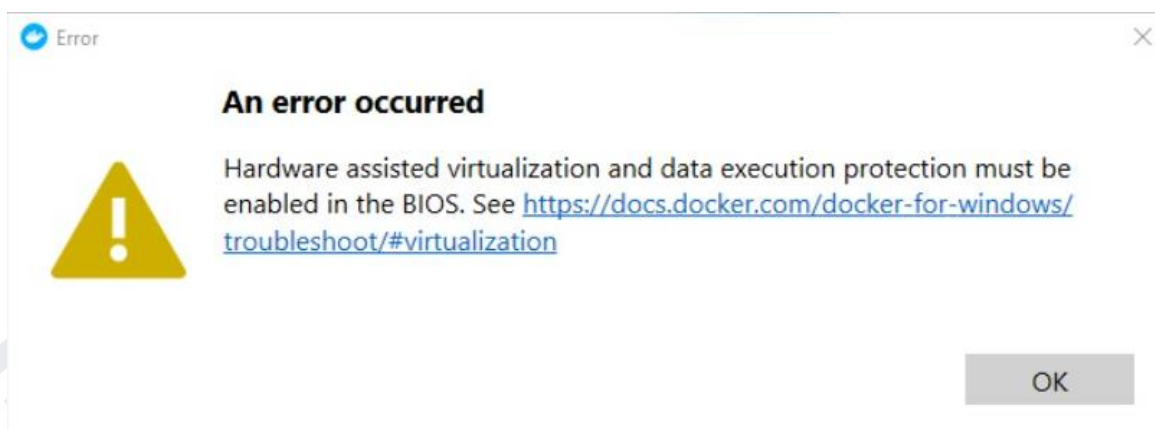


Рисунок 3.8 – Проблема віртуалізації у додатку Docker

У зв'язку з проблемою, яка була вказана вище, вибір був змінений у користь другого сервісу для розгортання бази даних. Для цього був використаний Heroku на безоплатній основі, про який йшлося у другому розділі.

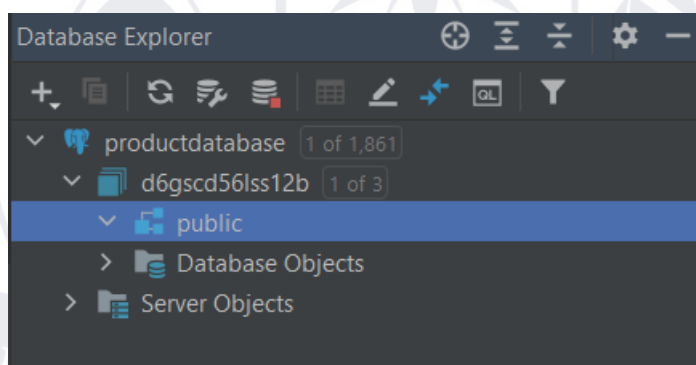


Рисунок 3.9 – Створення пустої бази даних

Робота почалась, як уже вище зазначалось зі створення пустої бази даних. Надалі базу потрібно було розмістити на сервері. На офіційному сайті Heroku була зроблена реєстрація, та запит на вільний сервер для розгортання. На рисунку 3.9 зазначено створення бази у вже згаданому додатку, який був використаний для наглядності та зручності у роботі з базами даних DataGrip. База не має ніякого наповнення, так як комірка «public» розгорнута і не має таблиця, або полів.

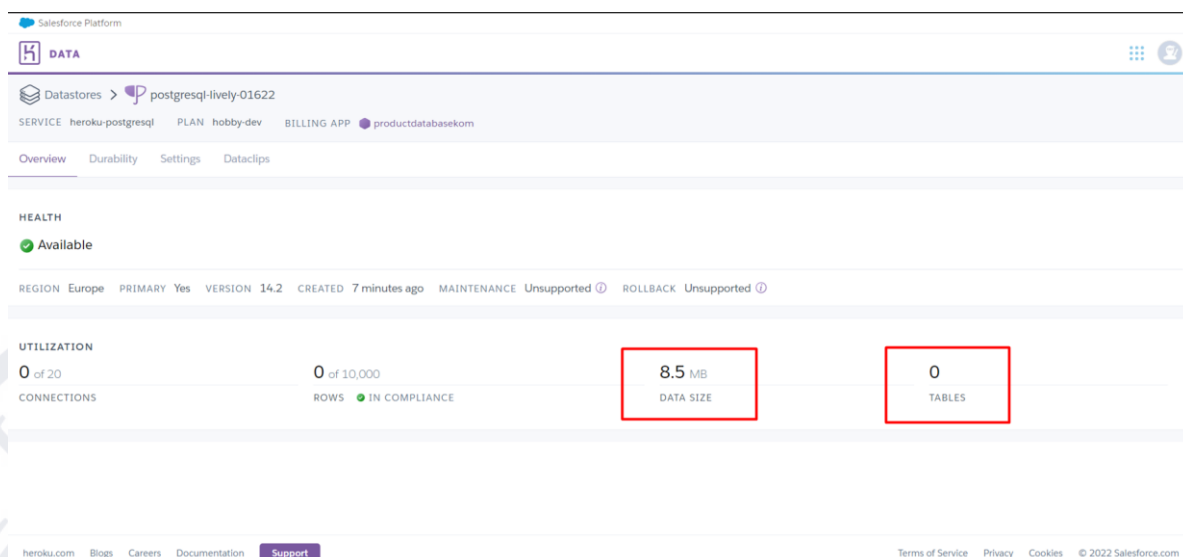


Рисунок 3.10 – Створення пустої бази Heroku

На рисунку 3.10 зображено створення пустої бази вже на фоні сервісу Heroku. Червоними квадратами на рисунку виділені значення, які вказують, що база уже має певний розмір, а саме 8,5 мегабайт та містить у собі 0 таблиць. Також вище можливо переглянути, що база використовує систему управління PostgreSQL, вказано додаток за допомогою якого вона створювалась, регіон на якому розташований сервер та скільки днів тому була створена база.

Наступним етапом у проектуванні бази даних - це підключення до віддаленої бази, яка була розгорнута на Heroku. Для цього, отримавши необхідні дані, а саме: логін, пароль, ім'я домену, хост, ID, можна підключати бази.

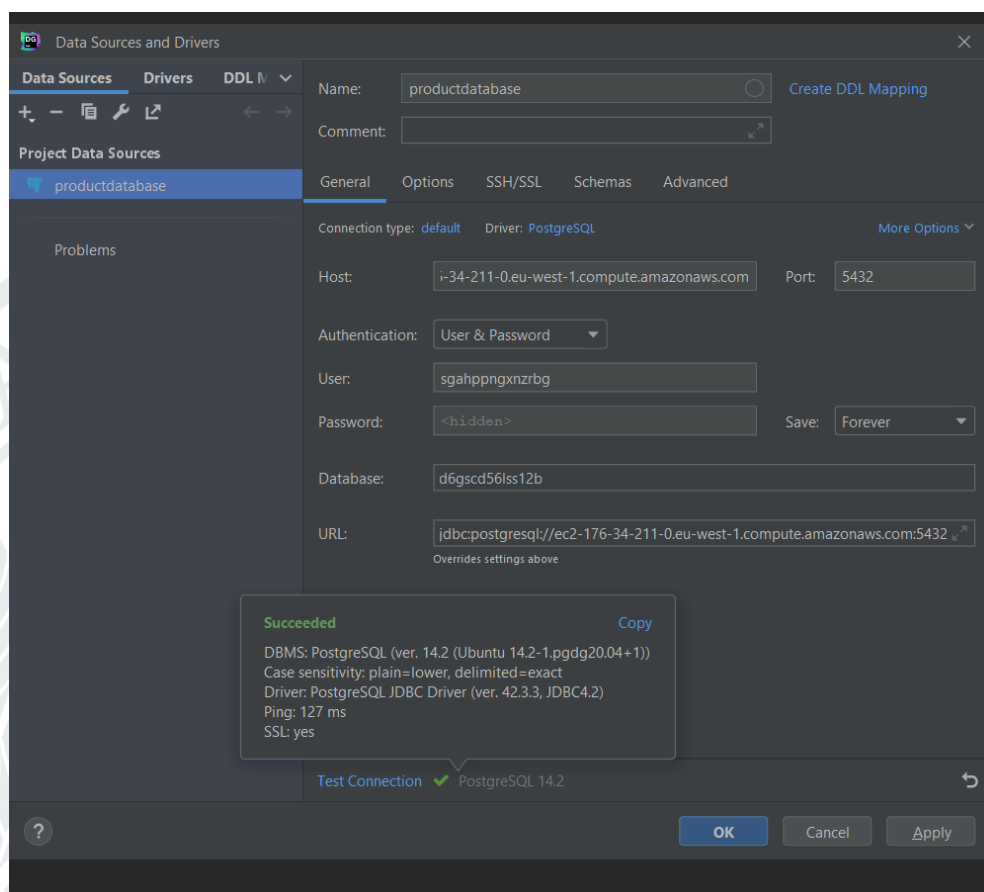


Рисунок 3.11 – Підключення до віддаленої бази

На рисунку 3.11 зображено підключення до віддаленої бази Негоки з усіма відповідними параметрами. Варто зазначити, що база розміщена на одному з європейських серверів Amazon. На рисунку зазначені усі параметри, які були перераховані вище та співпадають з ними.

3.4 Створення та накат міграцій

Зміни бази даних з однієї версії на другу, як правило завжди супроводжуються міграціями. Це потрібно для адаптації даних, щоб їх потім можна було використовувати в новій схемі, або архітектурі. Наприклад видалення поля, або стовпця може спричинити зміни в табілці, тому міграції є обов'язковим інструментом для використання. Це є легке рішення для розробників, так як займає набагато менше часу ніж кожену зміну поля прописувати програмним кодом.

Для початку міграцій у базі, були створені класи та деяка програмна логіка для створення таблиць та полів.

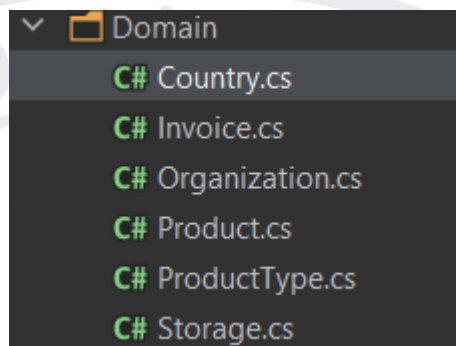


Рисунок 3.12 – Створення класів на мові C# для роботи з базою даних

На рисунку зображено створення усіх класів – таблиць, які були зазначені у схемі бази даних.

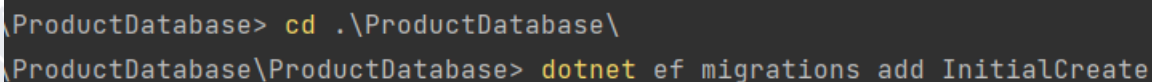
```
public class Invoice
{
    public int Id { get; set; }
    public decimal Price { get; set; }
    public int OrganizationId { get; set; }
    [JsonIgnore]
    public Organization Organization { get; set; }
    public int StorageId { get; set; }
    public ICollection<Product> ProductCollection { get; set; }
    public DateTime Date { get; set; }
}
```

Рисунок 3.13 – Програмний код класу накладних

На рисунку 3.13 зображено програмний код для таблиці накладних. Кожне поле створене з певними типами даних, які були розглянуті на етапі проектування схеми бази даних. Можна побачити поля які були створені, унікальний ідентифікатор таблиці, ціни та зовнішні ключі інших таблиць з якими пов'язані накладні. У даному випадку зв'язок таблиць «багато-до-багатьох». Процедура створення таблиць була виконана схожим способом для кожної

описаної таблиці вище. На рисунку фіолетовим зазначені поля, які є зовнішнім ключем інших таблиць.

Наступним кроком після створення класів та опис відповідних полів для функціонування бази даних – є створення та застосування міграцій. Завдяки зручним та швидким інструментам у платформі .Net операції з міграціями є доволі легкими та не потребують багато часу. Натомість потребують правильного проектування архітектури та змін у базі, так як цілком можливий сценарій, коли попередні дані будуть видалені, а поточні не будуть застосовані.



```
ProductDatabase> cd .\ProductDatabase\  
ProductDatabase> dotnet ef migrations add InitialCreate
```

Рисунок 3.14 – Команда для створення міграцій та подальшого їх застосування



```
Build started...  
Build succeeded.  
The Entity Framework tools version '6.0.3' is older than that of the runtime '7.0.0-preview.2.22153.1'. Update the tools for the latest features and bug fixes. See https://aka.ms/AAC1f  
for more information.  
Info: Microsoft.EntityFrameworkCore.Infrastructure[10403]  
Entity Framework Core 7.0.0-preview.2.22153.1 initialized 'ApplicationContext' using provider 'Npgsql.EntityFrameworkCore.PostgreSQL:7.0.0-preview.2+b1d0ef42bdc96adf002f761edc848  
9b46d6d54b5' with options: None  
Done. To undo this action, use 'ef migrations remove'
```

Рисунок 3.15 – Процес створення міграцій та супутня інформація

Після того як міграції було правильно створено та отримано повідомлення про успіх, можливо продовжити роботу з ними. Застосування міграції є ключовим етапом у всій подальшій роботі з базою даних[20]. Також як зазначалось вище після фундаментальних змін у базі, вона потребує оновлення.

```

Build started...
Build succeeded.
The Entity Framework tools version '6.0.3' is older than that of the runtime '7.0.0-preview.2.22153.1'. Update the tools for the latest features and bug fixes. See https://aka.ms/AaC1f
for more information.
Info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 7.0.0-preview.2.22153.1 initialized 'ApplicationContext' using provider 'Npgsql.EntityFrameworkCore.PostgreSQL:7.0.0-preview.2+b1d0ef42bdc96adf002f761edc848
9b46d6d54b5' with options: None
Info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (160ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT EXISTS (SELECT 1 FROM pg_catalog.pg_class c JOIN pg_catalog.pg_namespace n ON n.oid=c.relnamespace WHERE c.relname='__EFMigrationsHistory');
Info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (76ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE "__EFMigrationsHistory" (
        "MigrationId" character varying(150) NOT NULL,
        "ProductVersion" character varying(32) NOT NULL,
        CONSTRAINT "PK___EFMigrationsHistory" PRIMARY KEY ("MigrationId")
      );
Info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (51ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT EXISTS (SELECT 1 FROM pg_catalog.pg_class c JOIN pg_catalog.pg_namespace n ON n.oid=c.relnamespace WHERE c.relname='__EFMigrationsHistory');
Info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (52ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      SELECT "MigrationId", "ProductVersion"
      FROM "__EFMigrationsHistory"
      ORDER BY "MigrationId";
Info: Microsoft.EntityFrameworkCore.Migrations[20402]
      Applying migration '20220411190106_InitialCreate'.
Applying migration '20220411190106_InitialCreate'.
Info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (63ms) [Parameters=[], CommandType='Text', CommandTimeout='30']

```

Рисунок 3.16 – Процес застосування міграцій(1)

```

Info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (63ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE "Countries" (
        "Id" integer GENERATED BY DEFAULT AS IDENTITY,
        "Name" text NOT NULL,
        CONSTRAINT "PK_Countries" PRIMARY KEY ("Id")
      );
Info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (59ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE "Organizations" (
        "Id" integer GENERATED BY DEFAULT AS IDENTITY,
        "Name" text NOT NULL,
        "RegistrationDate" timestamp with time zone NOT NULL,
        "Address" text NOT NULL,
        "Rating" integer NOT NULL,
        CONSTRAINT "PK_Organizations" PRIMARY KEY ("Id")
      );
Info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (66ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE "ProductTypes" (
        "Id" integer GENERATED BY DEFAULT AS IDENTITY,
        "Name" text NOT NULL,
        "Description" text NOT NULL,
        CONSTRAINT "PK_ProductTypes" PRIMARY KEY ("Id")
      );
Info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (59ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE "Storages" (
        "Id" integer GENERATED BY DEFAULT AS IDENTITY,

```

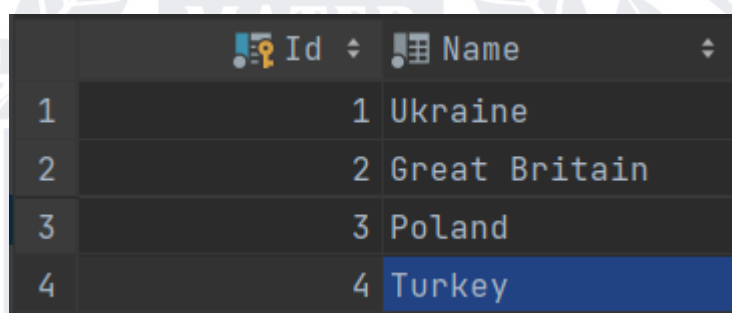
Рисунок 3.17 – Процес застосування міграцій (2)

На даних двох рисунках зображено процес застосування міграцій. У графі «info» зображено усю інформацію зі створенням таблиць та кожного поля відповідної таблиці. Також наявна інформація за допомогою якої технології цей процес відбувається(EntityFrameworkCore). Наявна затримка до сервера на якому застосовуються міграції.

3.5 Заповнення таблиць даними

Останнім етап для повноцінного створення бази даних – є заповнення кожної таблиці відповідними даними. Для досягнення даних цілей був використаний інструмент «DataGrip», який був описаний вище.

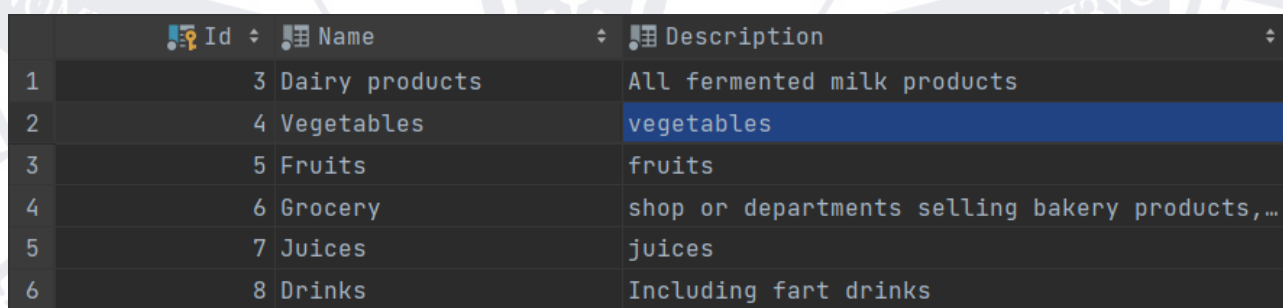
Заповнення таблиць даними починається з таблиці «Country», яка є постачальником певного продукту.



	Id	Name
1	1	Ukraine
2	2	Great Britain
3	3	Poland
4	4	Turkey

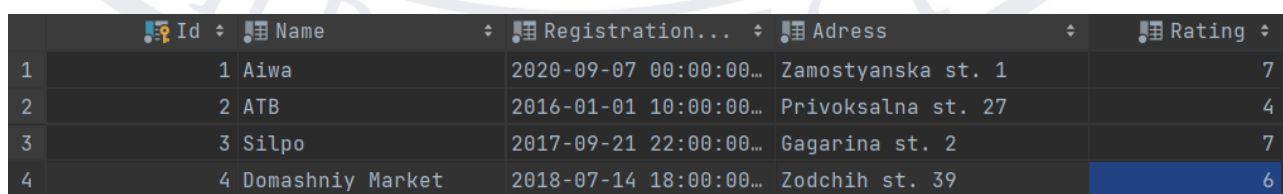
Рисунок 3.18 – Заповнення таблиці «Country» даними

Наступною була таблиця «ProductTypes», яка описує тип продукту та зв'язана з головною таблицею «Products».



	Id	Name	Description
1	3	Dairy products	All fermented milk products
2	4	Vegetables	vegetables
3	5	Fruits	fruits
4	6	Grocery	shop or departments selling bakery products,...
5	7	Juices	juices
6	8	Drinks	Including fart drinks

Рисунок 3.19 – Заповнення таблиці «ProductTypes» даними



	Id	Name	Registration...	Adress	Rating
1	1	Aiwa	2020-09-07 00:00:00...	Zamostyanska st. 1	7
2	2	ATB	2016-01-01 10:00:00...	Privoksalna st. 27	4
3	3	Silpo	2017-09-21 22:00:00...	Gagarina st. 2	7
4	4	Domashniy Market	2018-07-14 18:00:00...	Zodchih st. 39	6

Рисунок 3.20 – Заповнення таблиці «Organisation» даними

На рисунку 3.20 зображено заповнення таблиці даними, відповідно організація має назву, адресу, дату реєстрації, адресу та рейтинг.

	Id	Name	Adress	RentPrice	CountryId
1	1	Storage Big	Soborna st.	3000	1
2	2	Storage small aiwa	Striletska st.	1500	4
3	3	Storage of fruits ...	Zodchih st.	1800	2
4	4	Storage of vegetab...	Soborna st.	1800	4

Рисунок 3.21 – Заповнення таблиці «Storage» даними

На рисунку 3.21 зображено заповнення даними таблиці «Storage», яка є складом певної організації.

	Id	Name	Description	Image	Cost	ProductTypeId	CountryId	StorageId
1	3	Milk	Pasteurized milk	210.img	2	3	1	1
2	4	Cheese	cream cheese	211.img	3	3	1	1
3	5	Eggplant	vegetables	159.img	0.5	4	2	4
4	6	Tomato	vegetables	150.img	0.4	4	2	4
5	7	Apple	fruits	101.img	0.4	5	3	3
6	8	Mango	fruits	102.img	0.6	5	3	3
7	9	Bread	grocery	55.img	0.3	6	1	4
8	10	Coca-cola	drinks	01.img	1	8	4	2

Рисунок 3.22 – Заповнення таблиці «Products» даними

Таблиця «Products» заповнена даними про ті чи інші продукти. Вони мають короткий опис, ціну, та зовнішні ключі, які пов'язані з іншими таблицями.

У полях також реалізована перевірка на правильність вводу даних, або перевірка чи поля пусті.

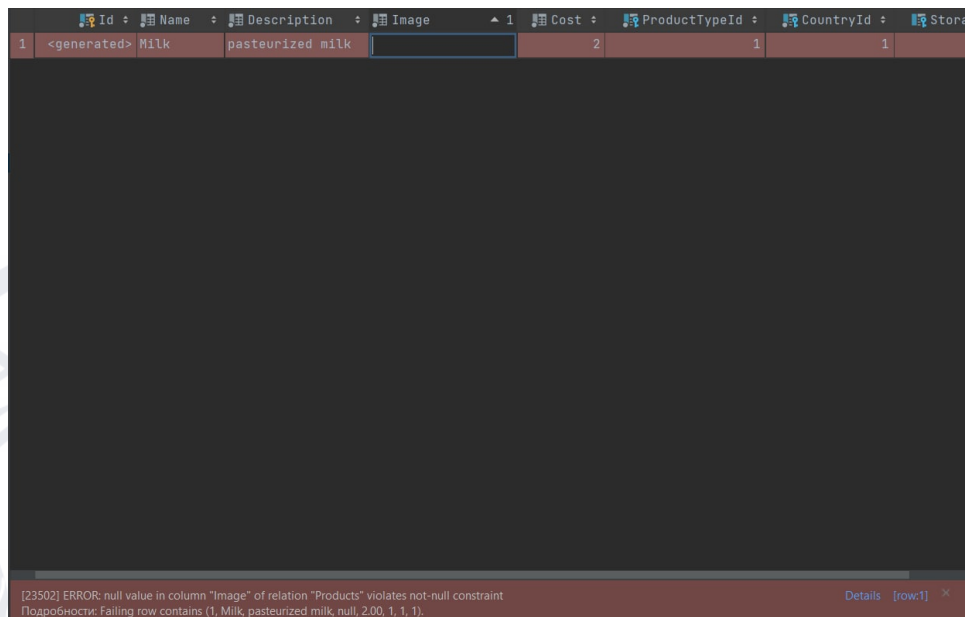


Рисунок 3.23 – Перевірка на порожнє поле, запис неможливо додати до бази

Як уже зазначалось вище, важливим полем є додавання дати у таблиці «invoices».

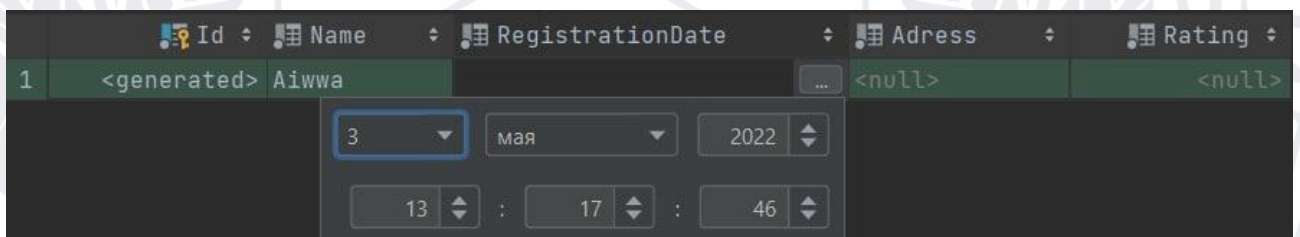


Рисунок 3.24 – Приклад заповнення поля «RegistrationDate»

На рисунку 3.24 зображено заповнення поля дати, де можна вказати число, місяць, рік та час створення накладної.

У підсумку, щоб зрозуміти чи всі дані були збережені та передані на сервер, на сайті «Негоки» потрібно переглянути інформацію про поточну базу даних.

HEALTH			
Available			
REGION	Europe	PRIMARY	Yes
VERSION	14.2	CREATED	22 days ago
MAINTENANCE	Unsupported	ROLLBACK	Unsupported
UTILIZATION			
1 of 20	1 of 10,000	9.2 MB	8
CONNECTION	ROW	DATA SIZE	TABLES
	IN COMPLIANCE		

Рисунок 3.25 –Аналіз інформації про базу даних.

На даному рисунку помітно, що поля «tables» набуло значення 8. Також збільшився розмір самої бази даних та поле «health» свідчить про працездатність системи. З чого можливо зробити висновок, що всі дані були успішно збережені, зв'язки встановлені та передані на сервер, де інші користувачі можуть повноцінно користуватись базою.

Висновок до розділу 3

У даному розділі було розглянуто поетапне створення схеми бази даних, її зв'язків, структур та таблиць. Кожен із кроків був описаний та супроводжувався рисунками.

ВИСНОВОК

У рамках даної роботи була розглянута актуальність проектування бази даних та реалізація задачі за допомогою різних інструментів .

Була проаналізована предметна область, та визначення шляхів проектування. Також було розглянуте готове рішення бази даних, а саме: додаток «IBM Retail» для моніторингу та обліку цін на продукти.

Надалі були обрані основні інструменти та технології для розробки. JetBrains Rider зі своїми перевагами – є обраним середовищем розробки. Проектування здійснювалось на мові програмування C# та на платформі .Net. Наявний опис технологій, такий як: NuGet та JSON, LINQ, Entity Framework з переліком їх переваг. Також були розглянуті різні моделі бази даних та їх системи управління і обрано найоптимальнішу.

У використаних інструментах також зрівнювались дві основні системи управління базами даних, їх переваги та недоліки. Відповідно було обрано оптимальну та на якій є досвід управління базами.

Робота має повну схему бази даних з усіма переліченими таблицями, їх зв'язками та поясненням до кожного поля. Опис поля, виходячи з того, які саме дані будуть передаватись.

У роботі представлені рисунки та результати роботи бази даних. Кожен крок створення супроводжувався відповідним описом та рисунками. Описана її працездатність та підтвердження, що все система працює оптимально.

Таким чином, всі поставлені задачі були виконані у повному об'ємі та система функціонує відповідно до задумки.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Що таке дані [Електронний ресурс] – Режим доступу до ресурсу: <https://futurenow.com.ua/shho-take-bazy-danyh-yih-pryznachennya-ta-vydy/>
2. Основні поняття про база даних [Електронний ресурс] – Режим доступу до ресурсу: <https://bondarenko.dn.ua/osnovni-ponyattya-relyatsijnih-bd-normalizatsiya-zv-yazok-ta-klyuchi/#:~:text=%D0%9F%D0%B5%D1%80%D0%B2%D0%B8%D0%BD%D0%BD%D0%B8%D0%B9%20%D0%BA%D0%BB%D1%8E%D1%87%20%D0%B2%D0%B8%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D0%BE%D0%B2%D1%83%D1%94%D1%82%D1%8C%D1%81%D1%8F%20%D0%B4%D0%BB%D1%8F%20%D0%B7%D0%B2,%D0%B2%D0%B8%D0%B7%D0%BD%D0%B0%D1%87%D0%B0%D1%94%20%D1%81%D0%BF%D0%BE%D1%81%D1%96%D0%B1%20%D0%BE%D0%B1%27%D1%94%D0%B4%D0%BD%D0%B0%D0%BD%D0%BD%D1%8F%20%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%8C.>
3. Схема архітектура бази даних [Електронний ресурс] – Режим доступу до ресурсу: <https://studfile.net/preview/9038964/page:2/>
4. Приклад готового рішення [Електронний ресурс] – Режим доступу до ресурсу: <https://abmcloud.com/avtomatizatsiya-torgivli-produktovih-magaziniv-amigos-u-hmarnij-sistemi-abm-retail/>
5. Таблиця як основний елемент реляційної бази даних [Електронний ресурс] – Режим доступу до ресурсу: https://elib.lntu.edu.ua/sites/default/files/elib_upload/%D0%95%D0%9D%D0%9F_%D0%A1%D0%B0%D0%B2%D0%B0%D1%80%D0%B8%D0%BD_%D0%9B%D0%B5%D0%BF%D0%BA%D0%B8%D0%B9/teoretic/lec4.html
6. Інформація про таблиці [Електронний ресурс] – Режим доступу до ресурсу: <https://support.microsoft.com/uk-ua/office/%D0%B7%D0%B0%D0%B3%D0%B0%D0%BB%D1%8C%D0%BD%D1%96->

%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BE%D1%81%D1%82%D1%96-%D0%BF%D1%80%D0%BE-
%D1%82%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%96-78ff21ea-2f76-4fb0-8af6-
c318d1ee0ea7#:~:text=%D0%A2%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%96%20%E2%80%93%D1%86%D0%B5%20%D0%B2%D0%B0%D0%B6%D0%BB%D0%B8%D0%B2%D1%96%20%D0%BE%D0%B1%27%D1%94%D0%BA%D1%82%D0%B8,%D0%B7%D0%B1%D0%B5%D1%80%D1%96%D0%B3%D0%B0%D1%8E%D1%82%D1%8C%D1%81%D1%8F%20%D0%B2%D1%81%D1%96%20%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BE%D1%81%D1%82%D1%96%20%D0%B0%D0%B1%D0%BE%20%D0%B4%D0%B0%D0%BD%D1%96.

7. Загальні відомості про JSON [Електронний ресурс] / ukr.com – Режим доступу до ресурсу: <https://ukr.kagutech.com/3910481-json-format-description-example>

8. Приклад бази даних у графовому представленні [Електронний ресурс] – Режим доступу до ресурсу: <https://protey.net/threads/grafovaja-baza-dannyx-neo4j.223/>

9. Приклад об'єкто-орієнтованої бази даних [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%9E%D0%B1%27%D1%94%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D1%96%D1%94%D0%BD%D1%82%D0%BE%D0%B2%D0%B0%D0%BD%D0%B0_%D0%B1%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85

10. Основи відомості про C# та .Net [Електронний ресурс] / znannya.org – Режим доступу до ресурсу: <http://www.znannya.org/?view=csharp-dotNET>

11. Entity Framework [Електронний ресурс] – Режим доступу до ресурсу: <https://www.entityframeworktutorial.net/what-is-entityframework.aspx>

12. LINQ [Електронний ресурс] – Режим доступу до ресурсу: <https://www.tutorialsteacher.com/linq/what-is-linq>
13. Опис технології NugGet [Електронний ресурс] / habr.com – Режим доступу до ресурсу: <https://habr.com/ru/post/274283/>
14. Загальні відомості про Visual Studio [Електронний ресурс] / docs.microsoft.com – Режим доступу до ресурсу: <https://docs.microsoft.com/ru-ru/visualstudio/get-started/visual-studio-ide?view=vs-2019>
15. Загальні відомості про Rider [Електронний ресурс] – Режим доступу до ресурсу: <https://www.jetbrains.com/ru-ru/rider/>
16. Приклад таблиці у DataGrip [Електронний ресурс] – Режим доступу до ресурсу: <https://www.jetbrains.com/ru-ru/datagrip/>
17. Загальні відомості про додаток Docker [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/get-started/overview/>
18. Логотип додатку Docker [Електронний ресурс] – Режим доступу до ресурсу: <https://docs.docker.com/get-started/overview/>
19. Загальні відомості про сервіс Heroku [Електронний ресурс] – Режим доступу до ресурсу: <https://www.heroku.com/about>
20. Накат міграцій [Електронний ресурс] – Режим доступу до ресурсу: <https://www.red-gate.com/simple-talk/databases/sql-server/database-administration-sql-server/database-deployment-cribsheet/>