

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**Український державний університет науки і технологій**

Кафедра «Комп’ютерні інформаційні технології»

**Курсовий проект**

**з дисципліни «Бази даних»**

Виконав:

Студент гр. ПЗ2011

Кулик С.В.

Прийняв:

Іванов О.П.

Дніпро

2024

**Міністерство освіти і науки України**

**Український державний університет науки і технологій**

Факультет: Комп’ютерних технологій і систем

Кафедра: Комп’ютерні інформаційні технології

Спеціальність: 121. Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ  
Зав. кафедри «КІТ»  
доцент Вадим ГОРЯЧКІН  
« 28» грудня 2023р.

**ЗАВДАННЯ**до курсового проекту по дисципліні **«Бази даних»**

студента \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Кулика Сергія Вадимовича\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Тема проекту: Розробка реляційної бази даних, проектування структури сховища даних. Проектування журналу успішності студентів.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_
2. Дата видачі завдання \_28 вересня 2023 р.\_\_\_\_\_\_\_\_\_\_\_\_\_
3. Перелік питань до розробки:  
   Вибір предметної області БД, інформаційно-логічне і концептуальне проектування бази даних у рамках EER моделі.
4. Постановка завдання, вибір предметної області, специфікація вимог до управління даними, визначення основних типів запитів.
   1. Розробка інформаційно-логічної і концептуальної EER моделі БД.
   2. Визначення типів сутностей, атрибутів, характеристик зв’язків, встановлення можливих і первинних ключів.
   3. Аналіз вимог до БД, виконання процедур спеціалізації/генерації, категоризації типів сутностей EER моделі БД.
5. Розробку логічного проекту бази даних.
   1. Розробка попередньої логічної схеми БД на основі EER моделі.
   2. Розподіл не ключових атрибутів, нормалізація БД.
   3. Вилучення зайвих, складних (m : n) і рекурсивних зв’язків, а також зв’язків що мають атрибути.
   4. Забезпечення цілісності БД.
   5. Аналіз запитів, утворення карт транзакцій.
   6. Заключна EER модель БД.
6. Розробка фізичного проекту бази даних.
   1. Розробка структур таблиць.
   2. Розробка схеми БД з урахуванням можливостей СУБД.
   3. Розробка запитів і форм.
   4. Контрольні випробування із керування БД, реалізація запитів (відповідність картам транзакцій).
7. Проектування логічної структури реляційних аналітичних систем накопичення і оброблення даних – ROLAP.
   1. Вибір предметної області, дослідження вимог до сховища даних складання попереднього набору аналітичних запитів.
   2. Розробка моделі сховища – вибір мір, вимірів, таблиць фактів.
   3. Графічні моделі нормалізованих (сніжинка) і ненормалізованих (зірка) схем сховища, які ураховують набори аналітичних запитів.
   4. Графічне представлення двох аналітичних запитів, а також реалізація запитів засобами OLAP для СУБД MS SQL Server.
8. Висновок по проекту – відмінності та особливості розробки інформаційних систем баз даних і сховищ даних.

Термін виконання курсового проекту **21 грудня 2023 р**.

Керівник курсового проекту\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_\_Іванов О.П.\_\_\_/

Завдання прийняв до виконання \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/\_\_\_Кулик С.В.\_\_\_/

**Календарний план**

|  |  |  |  |
| --- | --- | --- | --- |
| № | Етапи курсового проекту | Строк виконання | Примітка |
| 1 | Отримання завдання | 28.09.2023 |  |
| 2 | Аналіз методів вирішення поставленої задачі | 14.10.2023 |  |
| 3 | Розробка концептуальної моделі | 21.10.2023 |  |
| 4 | Логічне проектування бази даних | 28.10.2023 |  |
|  | Фізичне проектування бази даних |  |  |
|  | Нормалізація функціональних відносин в базі даних |  |  |
| 6 | Розробка клієнтського додатку | 14.11.2023 |  |
|  | Розробка інтерфейсу користувача | 21.11.2023 |  |
| 7 | Розробка сховища даних | 28.11.2023 |  |
| 8 | Розробка запитів до сховища даних | 07.11.2023 |  |
| 9 | Оформлення пояснювальної записки | 14.12.2023 |  |
| 10 | Здача пояснювальної записки | 21.12.2023 |  |
| 11 | Захист курсового проекту | 21.12.2023 |  |

**Реферат**

Курсовий проект: 36 стр., 7 рис., 7 джерел літератури, 3 додатки.

Об’єкт дослідження – база даних студентів.

Ціль роботи - розробка реляційної бази даних, проектування структури сховища даних.

Метод дослідження – Розробка бази даних на основі реляційної моделі представлення даних.

Було проведено аналіз вибраної галузі, ідентифіковано ключові моделі та зв'язки. На основі цього дослідження була створена концептуальна модель бази даних для освітньої галузі, а також розроблена реляційна модель бази даних з використанням сервера MS SQL Server. Крім того, був створений клієнт бази даних на мові C# з використанням Windows Forms.

БАЗИ ДАНИХ, РЕЛЯЦІЙНА МОДЕЛЬ, БАЗА ДАНИХ СТУДЕНТІВ, MICROSOFT SQL SERVER, WINDOWS FORMS.

**Зміст**

Розробка системи управління БД

[Опис предметної області 6](#_Toc154668621)

[Обґрунтування вибору СУБД 7](#_Toc154668621)

[Розробка концептуальної моделі бази даних 8](#_Toc154668621)

[Визначення сутностей та їх елементів 9](#_Toc154668621)

[Побудова реляційной моделі БД 11](#_Toc154668621)

[Нормалізація бази даних 1](#_Toc154668621)2

[Схема бази даних 1](#_Toc154668621)4

[Розробка проекту БД 1](#_Toc154668621)6

[Сховище даних з архутектурою «зірка» 18](#_Toc154668622)

[Сховище даних з архутектурою «сніжинка» 20](#_Toc154668623)

[**Висновки** 2](#_Toc154668624)1

[**Додаток 1** 2](#_Toc154668624)2

[**Додаток 2** 26](#_Toc154668624)

[**Додаток 3** 34](#_Toc154668625)

**Опис предметної області**

Предметною областю даного проекту є система управління інформацією про студентів, що дозволяє зберігати, обробляти та переглядати дані щодо студентів.

## Постановка завдання

Розробити об'єктно-орієнтовану програму для управління інформацією бази даних студентів:

* ***Можливість відбору даних за вказаними критеріями:***

Реалізувати функціонал для відбору студентів даних відповідно до заданих критеріїв і т.д.

* ***Пошук необхідної інформації в базі даних:***

Створити можливість здійснення пошуку в базі даних для виявлення конкретної інформації про студентів.

* ***Захист бази даних від несанкціонованого доступу:***

Реалізувати механізми захисту бази даних від неправомірних змін та забезпечити реєстрацію користувачів для контролю доступу.

* ***Можливість додавання потрібної інформації в базу даних:***

Розробити функціонал для додавання нових даних про студентів в базу даних, таких як оцінки, встигаємість.

* ***Інтерфейс користувача з перевіркою правильності введених даних:***

Створити зручний інтерфейс користувача з механізмами перевірки введених даних для уникнення помилок та забезпечення консистентності інформації в базі даних.

При цьому слід враховувати ефективність взаємодії програми з базою даних, забезпечуючи безпеку обробки і збереження інформації.

**Обґрунтування вибору СУБД**

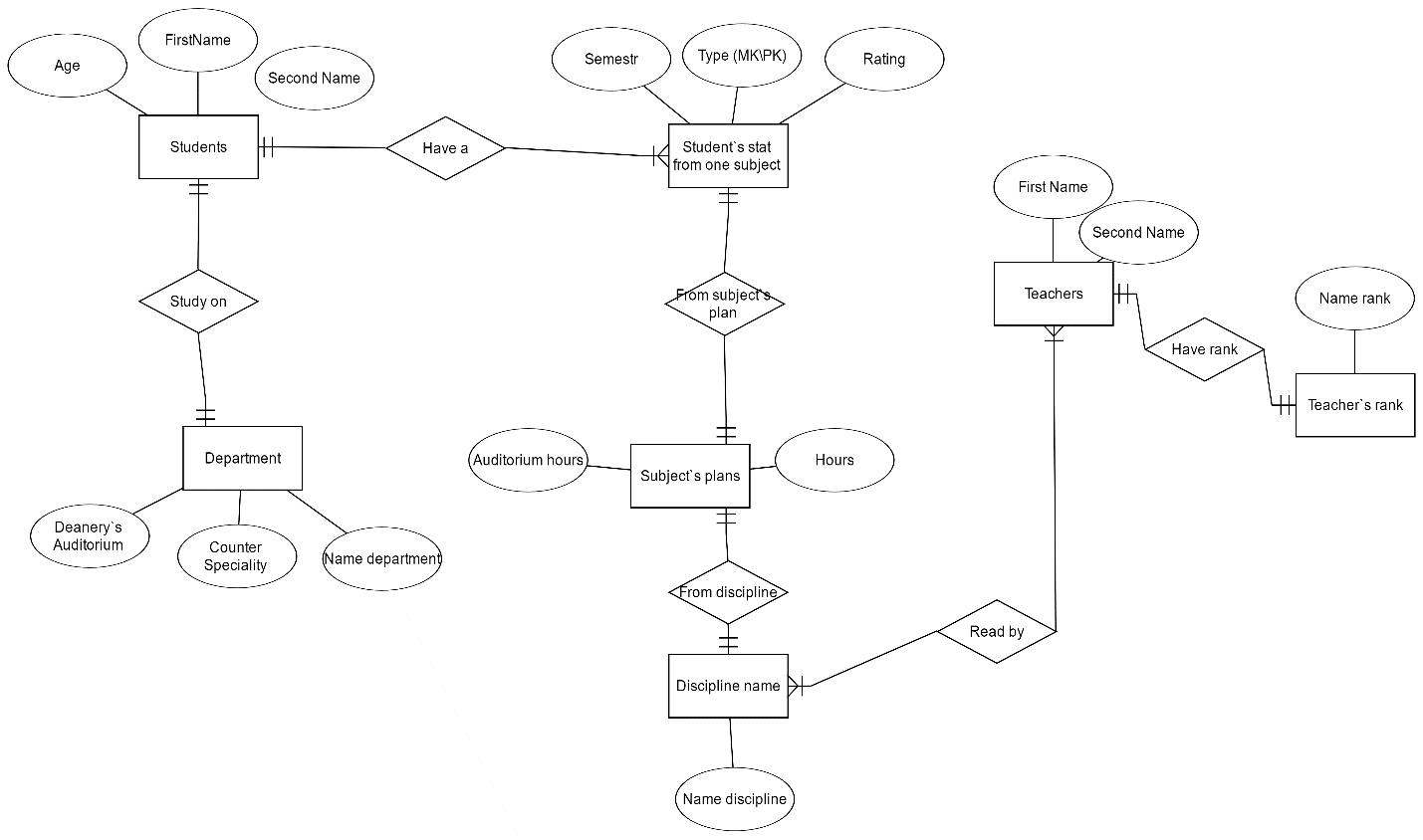
Вибір Системи Управління Базами Даних (СУБД) є стратегічним кроком, який впливає на ефективність та надійність роботи з інформацією в проекті. Ось чому обрання Microsoft SQL Server має свої переваги:

1. **Розширені можливості:** Microsoft SQL Server пропонує різноманітні функції для управління даними, що дозволяє розробникам та адміністраторам баз даних працювати на високому рівні функціональності.
2. **Інтеграція з продуктами Microsoft:** SQL Server ідеально поєднується з іншими продуктами компанії Microsoft, такими як .NET Framework, Azure, Excel, що полегшує роботу в екосистемі Microsoft.
3. **Зручний інтерфейс користувача:** SQL Server Management Studio (SSMS) забезпечує зручний інтерфейс для адміністрування та роботи з базами даних. Простий інтерфейс допомагає в розробці, оптимізації та моніторингу даних.
4. **Підтримка транзакцій та безпека:** SQL Server надає ефективні механізми для керування транзакціями та забезпечує безпеку даних за допомогою ролевої аутентифікації та шифрування.
5. **Масштабованість і продуктивність:** SQL Server може масштабуватися від малих до великих проектів та має інструменти для оптимізації продуктивності.

Вибір Microsoft SQL Server дозволяє не лише отримати потужний інструмент для роботи з базами даних, але й інтегровану екосистему, що спрощує розробку та підтримку проектів і забезпечує високий рівень ефективності та надійності.

**Розробка концептуальної моделі бази даних**

Сутнісно-реляційна діаграма - це інструмент у вигляді графічної схеми, який використовується для моделювання зв'язків між об'єктами або сутностями у системі чи програмі. Вона відображає структуру бази даних та взаємозв'язки між різними сутностями, ґрунтуючись на підході до проектування баз даних, який базується на сутностях та їх взаємозв'язках.



Мал. 1. ER-діаграма бази даних моделі предметної області

**Визначення сутностей та їх елементів**

|  |  |
| --- | --- |
| **Сутність** | **Опис** |
| Students | Містить головну інформацію про студентів.  C:\Users\Storchak\Desktop\Снимок.JPG |
| Student`s stat | Містить інформацію про успішність та заборгованість студентів по кожному предмету.  C:\Users\Storchak\Desktop\Снимок.JPG |
| Teachers | Містить головну інформацію про викладачів  C:\Users\Storchak\Desktop\Снимок.JPG |
| Teacher\_ranks | Містить інформацію про ступінь викладачів  C:\Users\Storchak\Desktop\Снимок.JPG |

|  |  |
| --- | --- |
| Disciplines | Містить головну інформацію про навчальні предмети  C:\Users\Storchak\Desktop\Снимок.JPG |
| TeachersSubject | Містить інформацію про предмети які читають викладачі.  C:\Users\Storchak\Desktop\Снимок.JPG |
| TypeSubject | Містить інформацію про тип предмета у рейтингу студентів.  C:\Users\Storchak\Desktop\Снимок.JPG |
| Subject\_plan | Містить інформацію план навчальних предметів.  C:\Users\Storchak\Desktop\Снимок.JPG |
| Departments | Містить інформацію про факультети вищих навчальних закладів.  C:\Users\Storchak\Desktop\Снимок.JPG |
| Speciality | Містить інформацію про спеціальності що є на факультетах.  C:\Users\Storchak\Desktop\Снимок.JPG |

**Побудова реляційной моделі БД**

Реляційний метод побудови баз даних ґрунтується на певній філософії, яка спрощує структуру даних та полегшує їх використання. Його головні принципи включають подання даних у вигляді таблиць і використання математичних концепцій алгебри відносин для обробки даних.

В реляційній моделі дані представлені у вигляді таблиць, де кожен елемент відображає окремий запис. Основні властивості таких таблиць включають однорідність стовпців, унікальні назви стовпців і можливість зміни порядку записів.

Відносини в реляційній моделі подані у вигляді таблиць, де рядки відповідають окремим записам, а стовпці - атрибутам. Ключеві поля, такі як первинний ключ, грають важливу роль у забезпеченні унікальності записів.

Один з ключових аспектів реляційної моделі - це можливість встановлення відносин між таблицями за допомогою ключів. Це забезпечує зв'язаність інформації та робить модель гнучкою для подальших розширень.

Хоча реляційний підхід дозволяє зручний доступ до даних, він має свої недоліки, такі як обмежена швидкість доступу, складність проектування та можливість зайвого розширення бази даних через нормалізацію. Проте, цей підхід залишається одним з найпоширеніших завдяки своїй простоті та математичній основі.

**Нормалізація бази даних**

Нормалізація бази даних - це процес структурування даних з метою зменшення повторення і підвищення ефективності обробки та оновлення інформації. Цей процес дозволяє вибрати найоптимальніший набір таблиць для зберігання даних, спрощує взаємодію з базою та забезпечує надійність та однорідність даних.

Важливо, щоб атрибути кожної таблиці були раціонально організовані, уникнення зайвого чи непотрібного зберігання інформації. Головна мета полягає у тому, щоб кожна таблиця відповідала вимогам нормалізації відносин та забезпечувала ефективну обробку даних.

Існують п'ять рівнів нормальних форм (1НФ, 2НФ, 3НФ, BCNF, 4НФ), але для більшості випадків найчастіше використовуються лише три: перша, друга і третя нормальні форми. Кожен рівень нормалізації має свої критерії, які визначаються взаємозв'язками між атрибутами таблиць.

**Перша нормальна форма**

Перша нормальна форма (1НФ) в реляційній базі даних означає, що всі атрибути кожного відношення мають прості (атомарні) значення. Це означає, що значення в кожному атрибуті не може бути списком або множиною складних значень. Забезпечення атомарності гарантує відсутність повторюваних груп у відношенні, що робить його нормалізованим.

У першій нормальній формі кожна схема відношення має бути атомарною, тобто значення не можуть бути деталізовані. Це важливо для забезпечення чіткості та однозначності представлення даних у реляційній базі.

Додатково, умова, що всі схеми бази даних повинні відповідати першій нормальній формі, забезпечує консистентність та ефективну організацію даних. Для досягнення 1НФ можуть використовуватися методи нормалізації, такі як унікалізація та розщеплення відносин

**Друга нормальна форма**

Друга нормальна форма (2НФ) застосовується до відносин, де первинний ключ складається з композитних атрибутів, тобто з двох або більше атрибутів. Важливо, щоб кожна компонента ключа функціонально визначала всі інші атрибути відносини. Якщо відносина має первинний ключ, що складається з одного атрибута, вона вже вважається в другій нормальній формі.

**Третя нормальна форма**

Третя нормальна форма (3НФ) вимагає, щоб кожен атрибут, що не є частиною первинного ключа, функціонально залежав лише від самого ключа, а не від інших атрибутів, які також не є ключами. Іншими словами, кожен неключовий атрибут повинен мати залежність від ключа, а не від інших неключових атрибутів.

Основна мета 2НФ та 3НФ - уникнути аномалій та створити логічну структуру бази даних за допомогою поділу на окремі таблиці та використання унікальних ключів. Це сприяє підтримці консистентності та оптимізації бази даних для більш ефективного управління інформацією.

**Четверта нормальна форма**

Четверта нормальна форма (4НФ) використовується для уникнення многозначних залежностей, які можуть виникнути в відносинах бази даних. Це досягається шляхом розбиття таблиць таким чином, щоб зберігати неконтекстуальні (многозначні) атрибути в окремих таблицях.

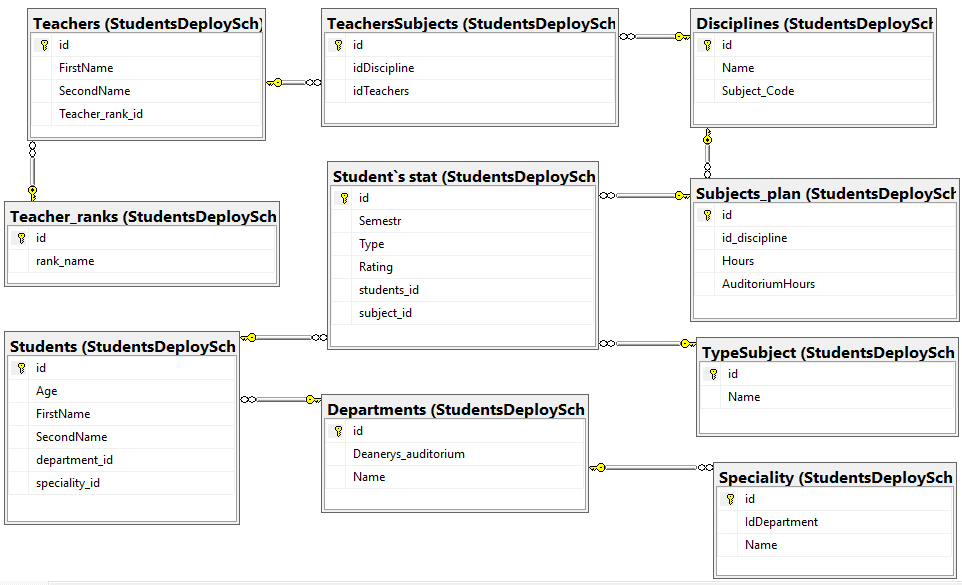
**П’ята нормальна форма**

П'ята нормальна форма (5НФ) вимагає розбиття відносин для уникнення ситуацій, коли існують залежності між неключовими атрибутами, що можуть призвести до аномалій та неконсистентності даних.

Ураховуючи 4НФ та 5НФ, база даних досягає високого рівня структурованості та уніфікованості, що забезпечує ефективне управління інформацією та уникнення аномалій в даних. Ці нормальні форми доповнюють попередні, сприяючи оптимізації та консистентності бази даних.

**Схема бази даних**

Структура бази даних, розроблена для проекту, спрямована на ефективне зберігання та керування інформацією про студентів та пов'язані з ними дані. Основні компоненти цієї структури бази даних відповідають високому рівню нормалізації. Це дозволяє уникнути дублювання даних та забезпечує консистентність інформації.



Мал. 2. База даних нормалізовна до 3 форми.

У нашій базі даних всі таблиці вже відповідають першій нормальній формі, оскільки всі атрибути в них є простими і не містять повторюваних груп.

Щодо другої нормальної форми, вона застосовується, коли кожен атрибут, який не є ключем, повністю залежить від цілого первинного ключа, а не лише від частини його. У нашій базі даних цей підхід врахований, і всі неключові атрибути повністю залежать від всього первинного ключа відповідної таблиці.

Щодо третьої нормальної форми, вона передбачає відсутність транзитивних залежностей відносно первинного ключа. Після аналізу таблиць ми переконалися, що такі транзитивні залежності відсутні, що підтверджує, що всі таблиці бази даних перебувають у третій нормальній формі.

**Розробка проекту БД**

Під час фізичного проектування бази даних, яке базується на розробленій логічній моделі, створюється конкретна фізична модель, що враховує особливості певної системи управління базами даних (СУБД). Основна мета цього етапу полягає в уточненні та оптимізації структури даних для досягнення максимальної продуктивності та ефективної взаємодії з обраною СУБД.

Початковий етап фізичного проектування включає створення основних таблиць, що відображають структуру даних, визначену на логічному рівні. Потім встановлюються зовнішні ключі для визначення зв'язків між різними таблицями, що відображають взаємозв'язки між об'єктами домену.

Логічна модель даних базується на вибраній моделі організації даних, такій як реляційна або мережева. З іншого боку, фізична модель деталізує, як саме дані будуть зберігатися та оброблятися в конкретній СУБД.

Необхідність фізичного проектування випливає з вимог обраної СУБД. На цьому етапі приймаються рішення стосовно створення реляційних таблиць, встановлення обмежень, методів зберігання даних та розробки засобів захисту інформації.

Важливо відзначити, що між логічним та фізичним проектуванням існує тісний зворотний зв'язок. Зміни, внесені на етапі фізичного проектування для оптимізації продуктивності, можуть вплинути на структуру логічної моделі даних.

Microsoft SQL Server, обрана для використання в проекті, є комерційною СУБД, яка підтримує мову запитів Transact-SQL та здатна працювати з базами даних будь-якого розміру, що підтверджується її успішністю на ринку протягом багатьох років.

*Скрипт розгортання бази даних наведено в додатку 1.*

**Транзакції та забезпечення цілісності**

**Транзакції**

Транзакція в базі даних (БД) - це сукупність операцій, які виконуються як єдине ціле. Ці операції вважаються атомарними, тобто вони виконуються або повністю, або не виконуються взагалі. Головна мета використання транзакцій - забезпечення консистентності та цілісності даних в БД.

Основні властивості транзакцій включають такі концепції:

***Атомарність (Atomicity):*** Атомарність в контексті транзакцій визначає, що операції виконуються як одне ціле, і якщо одна операція не вдалася, то всі інші операції також скасовуються. У такому випадку база даних повертається до свого початкового стану, забезпечуючи цілісність та консистентність даних*.*

***Консистентність (Consistency):*** Консистентність в контексті транзакцій означає, що транзакція переводить базу даних з одного стабільного стану в інший, забезпечуючи цілісність даних. Якщо транзакція порушує будь-які правила цілісності, вона автоматично скасовується, і база даних залишається незмінною.

***Ізоляція (Isolation):*** Ізоляція в контексті транзакцій означає, що транзакції виконуються паралельно, і їх взаємний вплив має бути мінімізованим. Це гарантує унікальність результату транзакції для кожного користувача, незалежно від одночасно проведених транзакцій.

***Тривалість (Durability):*** Gісля успішного завершення транзакції всі її зміни залишаються в базі даних і будуть відновлені навіть у випадку відмови системи. Це забезпечує надійність та коректність взаємодії з базою даних, що є критичним для правильності та надійності операцій з даними.

**Забезпечення цілісності бази даних**

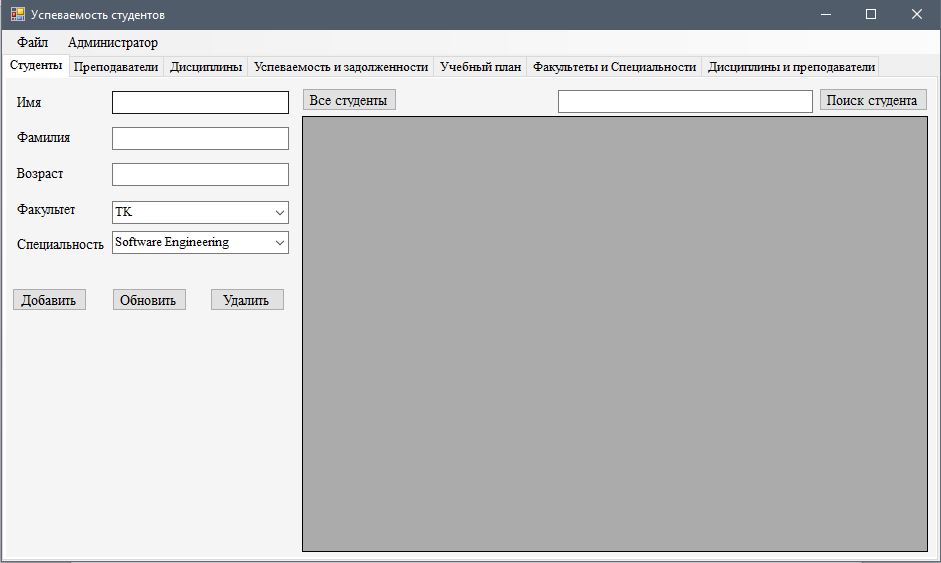
Цілісність даних у базі даних визначається її здатністю забезпечувати коректність та валідність даних на будь-якому етапі їхнього зберігання та обробки. Обмеження грають ключову роль у забезпеченні цієї цілісності, встановлюючи умови для вставки або модифікації даних в базі. Наприклад:

* **NOT** **NULL** вимагає, щоб значення у певному стовпці не були порожніми, що особливо важливо для обов'язкових атрибутів.
* **UNIQUE** гарантує унікальність значень у стовпці чи групі стовпців, щоб уникнути дублювання даних.
* **PRIMARY** **KEY** визначає унікальний ідентифікатор для кожного запису, що допомагає ідентифікувати та встановлювати відносини між записами.
* **FOREIGN** **KEY** вказує на зв'язок між таблицями, забезпечуючи цілісність даних та забороняючи вставку некоректних значень.
* **CHECK** встановлює умови для значень у стовпці, дозволяючи перевірити їхню відповідність логічним умовам чи обмеженням.

Використання обмежень є важливою практикою для забезпечення консистентності та цілісності даних, мінімізації помилок та некоректних операцій у базі даних.

**Розробка форм та запитів до бази даних**

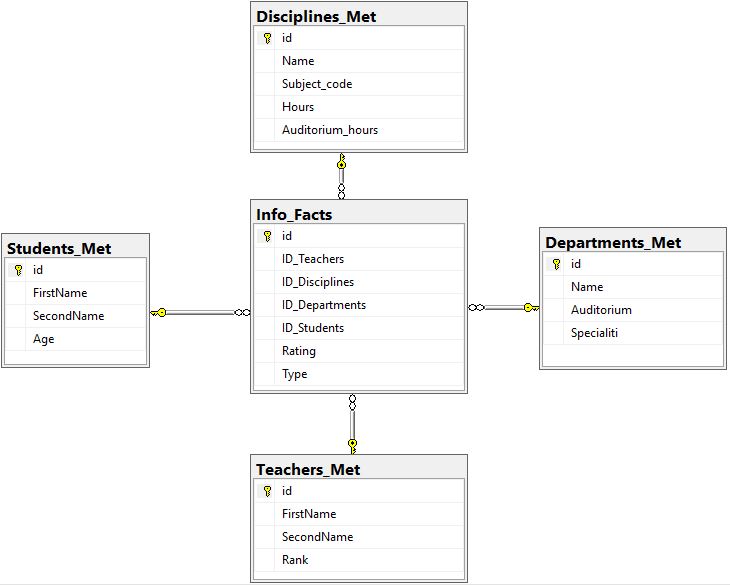
Гарно спроектований графічний інтерфейс користувача (GUI) є ключовим елементом для успішності будь-якого програмного забезпечення або системи. Він не лише робить використання програми зручним та ефективним, але й привертає увагу та створює позитивне враження у користувачів. Гарний GUI спрощує процес навчання користувачів, допомагає їм виконувати завдання швидше та з меншою кількістю помилок. При розробці форм та запитів до бази даних, зрозумілі форми та інтуїтивно зрозумілі елементи керування полегшують введення та отримання даних.



## **Сховище даних з архітектурою «зірка»**

Схема «зірки» - спеціальна організація реляційних таблиць, зручна для зберігання багатовимірних показників.

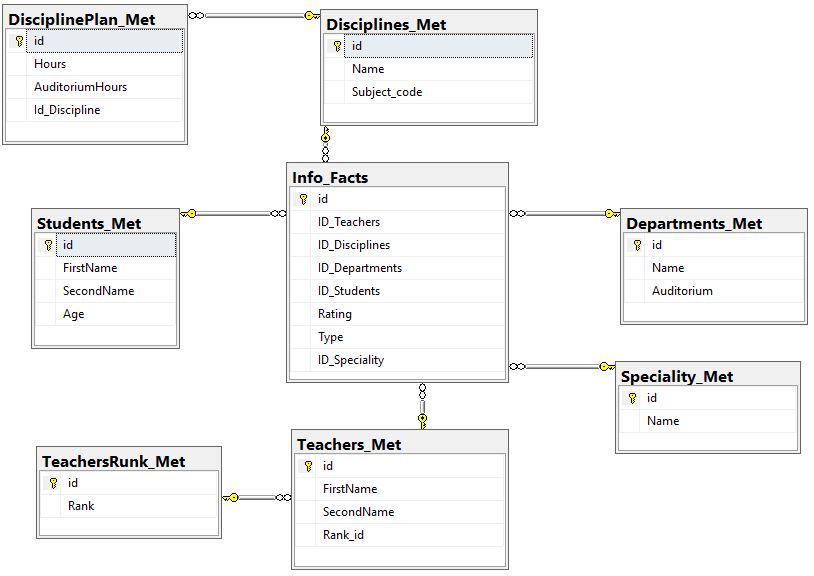
Модель даних складається з двох типів таблиць: однієї таблиці фактів - центр «зірки» - і декількох таблиць вимірів.



*Мал. 3. Сховище з архітектурою “Зірка”*

## **Сховище даних з архутектурою «сніжинка»**

Схема сніжинки отримала свою назву за форму, в якій відображається логічна схема таблиць у багатовимірній базі даних. Ця схема подібна до схеми зірки, оскільки також включає централізовану таблицю фактів, яка з'єднується з таблицями вимірювань. Однак схема сніжинки відрізняється тим, що в ній таблиці вимірювань нормалізовані разом з рядом інших пов'язаних вимірювальних таблиць. У схемі сніжинки структура складніша, оскільки вона має більш високий рівень нормалізації таблиць вимірів. Такий підхід створює ефект «сніжинки», який зачіпає тільки таблиці вимірювань, а не таблиці фактів, як у схемі зірки.



*Мал. 4. Сховище з архітектурою “Cніжинка”*

*Код форм та результати роботи програми наведено у додатку 3.*

**Висновки**

У курсовій роботі було розроблено та реалізовано базу даних для студентів проекту з використанням Microsoft SQL Server. Вибрано реляційний підхід для моделювання структури даних, проведено нормалізацію для забезпечення цілісності та уникнення аномалій. Для контролю цілісності даних були використані обмеження (constraint), такі як унікальність, непорожність та перевірка умов.

Велика увага була приділена розробці зручного графічного інтерфейсу користувача, що сприяє легкій взаємодії з системою. Для ефективної взаємодії з базою даних була використана мова запитів Transact-SQL. Фізична модель бази даних відповідає вимогам Microsoft SQL Server і забезпечує оптимальні структури зберігання даних та методи доступу.

Загальна мета полягала в створенні ефективної, цілісної та безпечної бази даних для задоволення інформаційних потреб користувачів в галузі освітнього проекту. Отримана система дозволяє ефективно обробляти дані, забезпечуючи надійність та зручність використання.

**ЛІТЕРАТУРА**

1. Роберт, В. Програмування Баз Даних Microsoft SQL Server 2005 [Текст] / В. Роберт. - ООО.: "І.Д.Вільямс", 2007. – 832 с.
2. Іцик, Бєн-Ган. Microsoft SQL Server 2012. Основи T-SQL [Текст] / Б. Іцик. - [Пер. з англ. М. А. Райтман]. – Москва: Ексмо, 2015. – 400 с.
3. Андрусевич, І. С. Основи роботи з базами даних та SQL [Текст] / І. С. Андрусевич. - Видавництво "Нова книга", 2019. - 320 с.
4. Грофф, Майкл. SQL для простих смертных [Текст] / Майкл Грофф, Пол Вайнберг. - [Пер. з англ. А. Осовської] - Київ: Добра книга, 2017. - 416 с.
5. Фейлд, Кріс. Використання Microsoft SQL Server 2016. [Текст] / Кріс Фейлд, Шон Харінгтон. - [пер. з англ. А. П. Грінберга]. - СПб: Пітер, 2017. - 416 с.
6. Джонсон, Деніел. Програмування віконних додатків на C# для Microsoft Windows [Текст] / Деніел Джонсон. - [Пер. з англ. О. В. Моренець]. - Київ: Видавничий дім "Гельветика", 2018. - 464 с.
7. Краббе, Альфред. Програмування на C# для професійних програмістів [Текст] / Альфред Краббе. - [Пер. з англ. Д. С. Шерстюк]. - Київ: Видавництво "БІНОМ", 2019. - 560 с.

**ДОДАТКИ****Додаток 1**

Скрипт створення бази даних.

-- Переключення на базу даних master

USE master;

GO

-- Створення бази даних StudentsDeploy, якщо вона не існує

IF NOT EXISTS (SELECT \* FROM sys.databases WHERE NAME = 'StudentsDeploy')

BEGIN

CREATE DATABASE StudentsDeploy

CONTAINMENT = NONE

ON PRIMARY

(

NAME = N'StudentsDeploy',

FILENAME = N'C:\DB\StudentsDeployment.mdf',

SIZE = 8192KB,

MAXSIZE = UNLIMITED,

FILEGROWTH = 15%

),

FILEGROUP StorageFileGroup

(

NAME = N'StudentsDeploySG',

FILENAME = N'C:\DB\StudentsDeploymentSG.ndf',

SIZE = 8192KB,

MAXSIZE = UNLIMITED,

FILEGROWTH = 15%

)

LOG ON

(

NAME = N'StudentsDeploy\_log',

FILENAME = N'C:\DB\StudentsDeploymentLog.txt',

SIZE = 8192KB,

MAXSIZE = 512MB,

FILEGROWTH = 8192KB

);

END

GO

-- Переключення на базу даних StudentsDeploy

USE StudentsDeploy;

GO

-- Створення схеми StudentsDeploySch, якщо вона не існує

IF NOT EXISTS (SELECT \* FROM INFORMATION\_SCHEMA.SCHEMATA WHERE SCHEMA\_NAME = 'StudentsDeploySch')

BEGIN

EXEC sp\_executesql N'CREATE SCHEMA StudentsDeploySch';

END

GO

-- Видалення таблиць, якщо вони існують

IF OBJECT\_ID(N'StudentsDeploySch.[Student`s stat]', N'U') IS NOT NULL

DROP TABLE StudentsDeploySch.[Student`s stat];

IF OBJECT\_ID(N'StudentsDeploySch.Students', N'U') IS NOT NULL

DROP TABLE StudentsDeploySch.Students;

IF OBJECT\_ID(N'StudentsDeploySch.Departments', N'U') IS NOT NULL

DROP TABLE StudentsDeploySch.Departments;

IF OBJECT\_ID(N'StudentsDeploySch.Subjects\_plan', N'U') IS NOT NULL

DROP TABLE StudentsDeploySch.Subjects\_plan;

IF OBJECT\_ID(N'StudentsDeploySch.TeachersSubjects', N'U') IS NOT NULL

DROP TABLE StudentsDeploySch.TeachersSubjects;

IF OBJECT\_ID(N'StudentsDeploySch.Teachers', N'U') IS NOT NULL

DROP TABLE StudentsDeploySch.Teachers;

IF OBJECT\_ID(N'StudentsDeploySch.Teacher\_ranks', N'U') IS NOT NULL

DROP TABLE StudentsDeploySch.Teacher\_ranks;

IF OBJECT\_ID(N'StudentsDeploySch.Disciplines', N'U') IS NOT NULL

DROP TABLE StudentsDeploySch.Disciplines;

IF OBJECT\_ID(N'StudentsDeploySch.TypeSubject', N'U') IS NOT NULL

DROP TABLE StudentsDeploySch.TypeSubject;

GO

-- Створення таблиці Departments

CREATE TABLE StudentsDeploySch.Departments (

id INT IDENTITY(1, 1) NOT NULL UNIQUE,

Deanerys\_auditorium INT NOT NULL,

Counter\_speciality INT NOT NULL CHECK (Counter\_speciality >= 0) DEFAULT (5),

Name NVARCHAR(50) NOT NULL DEFAULT ('TK'),

CONSTRAINT PK\_Departments PRIMARY KEY CLUSTERED (id ASC) WITH (IGNORE\_DUP\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY];

GO

-- Створення таблиці Disciplines

CREATE TABLE StudentsDeploySch.Disciplines (

id INT IDENTITY(1, 1) NOT NULL UNIQUE,

Name NVARCHAR(30) NOT NULL,

CONSTRAINT PK\_Disciplines PRIMARY KEY CLUSTERED (id ASC) WITH (IGNORE\_DUP\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY];

GO

-- Створення таблиці Students

CREATE TABLE StudentsDeploySch.Students (

id INT IDENTITY(1, 1) NOT NULL UNIQUE,

Age INT NOT NULL CHECK (Age >= 0),

FirstName NVARCHAR(50) NOT NULL,

SecondName NVARCHAR(50) NOT NULL,

department\_id INT NOT NULL,

CONSTRAINT PK\_Students PRIMARY KEY CLUSTERED (id ASC) WITH (IGNORE\_DUP\_KEY = OFF) ON [PRIMARY],

CONSTRAINT [FK\_Students\_Departments] FOREIGN KEY([department\_id]) REFERENCES StudentsDeploySch.Departments ([id])

) ON [PRIMARY];

GO

-- Включення перевірки обмежень для таблиці Students

ALTER TABLE [StudentsDeploySch].[Students] CHECK CONSTRAINT [FK\_Students\_Departments];

GO

-- Створення таблиці Subjects\_plan

CREATE TABLE StudentsDeploySch.Subjects\_plan (

id INT IDENTITY(1, 1) NOT NULL UNIQUE,

id\_discipline INT NOT NULL,

Hours INT NOT NULL CHECK (Hours >= 0),

AuditoriumHours INT NOT NULL CHECK (AuditoriumHours >= 0),

CONSTRAINT PK\_Subjects PRIMARY KEY CLUSTERED (id ASC) WITH (IGNORE\_DUP\_KEY = OFF) ON [PRIMARY],

CONSTRAINT [FK\_Subjects\_plan\_Disciplines] FOREIGN KEY ([id\_discipline]) REFERENCES StudentsDeploySch.Disciplines ([id])

) ON [PRIMARY];

GO

-- Включення перевірки обмежень для таблиці Subjects\_plan

ALTER TABLE [StudentsDeploySch].[Subjects\_plan] CHECK CONSTRAINT [FK\_Subjects\_plan\_Disciplines];

GO

-- Створення таблиці Student`s stat

CREATE TABLE StudentsDeploySch.[Student`s stat] (

id INT IDENTITY(1, 1) NOT NULL UNIQUE,

Semestr INT NOT NULL CHECK (Semestr >= 0),

Type INT NOT NULL,

Rating FLOAT NOT NULL,

students\_id INT NOT NULL,

subject\_id INT NOT NULL,

CONSTRAINT [PK\_Student`s stat] PRIMARY KEY CLUSTERED (id ASC) WITH (IGNORE\_DUP\_KEY = OFF) ON [PRIMARY],

CONSTRAINT CK\_Students\_type CHECK (Type = 1 OR Type = 2),

CONSTRAINT [FK\_Student`s stat\_Students] FOREIGN KEY ([students\_id]) REFERENCES StudentsDeploySch.Students ([id]),

CONSTRAINT [FK\_Student`s stat\_Subjects] FOREIGN KEY ([subject\_id]) REFERENCES StudentsDeploySch.Subjects\_plan ([id])

);

GO

-- Включення перевірки обмежень для таблиці Student`s stat

ALTER TABLE StudentsDeploySch.[Student`s stat] CHECK CONSTRAINT [FK\_Student`s stat\_Students];

ALTER TABLE StudentsDeploySch.[Student`s stat] CHECK CONSTRAINT [FK\_Student`s stat\_Subjects];

GO

-- Створення таблиці Teacher\_ranks

CREATE TABLE StudentsDeploySch.Teacher\_ranks (

id INT IDENTITY(1, 1) NOT NULL UNIQUE,

rank\_name NVARCHAR(50) NOT NULL,

CONSTRAINT PK\_Teacher\_ranks PRIMARY KEY CLUSTERED (id ASC) WITH (IGNORE\_DUP\_KEY = OFF) ON [StorageFileGroup]

) ON [StorageFileGroup];

GO

-- Створення таблиці Teachers

CREATE TABLE StudentsDeploySch.Teachers (

id INT IDENTITY(1, 1) NOT NULL UNIQUE,

FirstName NVARCHAR(30) NOT NULL,

SecondName NVARCHAR(30) NOT NULL,

Teacher\_rank\_id INT NOT NULL,

CONSTRAINT PK\_Teachers PRIMARY KEY CLUSTERED (id ASC) WITH (IGNORE\_DUP\_KEY = OFF) ON [StorageFileGroup],

CONSTRAINT [FK\_Teachers\_Teacher\_ranks] FOREIGN KEY ([Teacher\_rank\_id]) REFERENCES StudentsDeploySch.[Teacher\_ranks] ([id])

) ON [StorageFileGroup];

GO

-- Включення перевірки обмежень для таблиці Teachers

ALTER TABLE StudentsDeploySch.[Teachers] CHECK CONSTRAINT [FK\_Teachers\_Teacher\_ranks];

GO

-- Створення таблиці TeachersSubjects

CREATE TABLE StudentsDeploySch.TeachersSubjects (

id INT IDENTITY(1, 1) NOT NULL UNIQUE,

idDiscipline INT NOT NULL,

idTeachers INT NOT NULL,

CONSTRAINT PK\_TeachersSubjects PRIMARY KEY CLUSTERED (id ASC) WITH (IGNORE\_DUP\_KEY = OFF) ON [StorageFileGroup],

CONSTRAINT [FK\_TeachersSubjects\_Disciplines] FOREIGN KEY ([idDiscipline]) REFERENCES StudentsDeploySch.[Disciplines] ([id]),

CONSTRAINT [FK\_TeachersSubjects\_Teachers] FOREIGN KEY ([idTeachers]) REFERENCES StudentsDeploySch.[Teachers] ([id])

) ON [StorageFileGroup];

GO

-- Включення перевірки обмежень для таблиці TeachersSubjects

ALTER TABLE StudentsDeploySch.[TeachersSubjects] CHECK CONSTRAINT [FK\_TeachersSubjects\_Disciplines];

ALTER TABLE StudentsDeploySch.[TeachersSubjects] CHECK CONSTRAINT [FK\_TeachersSubjects\_Teachers];

GO

-- Видалення таблиці TypeSubject, якщо вона існує

IF OBJECT\_ID(N'StudentsDeploySch.TypeSubject', N'U') IS NOT NULL

DROP TABLE StudentsDeploySch.TypeSubject;

GO

-- Створення таблиці TypeSubject

CREATE TABLE StudentsDeploySch.TypeSubject (

id INT IDENTITY(1, 1) NOT NULL UNIQUE,

Name NVARCHAR(30) NOT NULL,

CONSTRAINT PK\_TypeSubject PRIMARY KEY CLUSTERED (id ASC) WITH (IGNORE\_DUP\_KEY = OFF) ON [PRIMARY]

) ON [PRIMARY];

GO

-- Вставка значень до таблиці TypeSubject

INSERT INTO StudentsDeploySch.TypeSubject (Name) VALUES ('MK'), ('PK');

-- Додавання колонки TypeI до таблиці Student`s stat

ALTER TABLE StudentsDeploySch.[Student`s stat]

ADD TypeI INT NULL;

GO

-- Оновлення значень колонки TypeI

UPDATE StudentsDeploySch.[Student`s stat]

SET TypeI = CASE WHEN Type = 1 THEN 1 WHEN Type = 2 THEN 2 END;

-- Видалення обмеження CHECK для колонки Type в таблиці Student`s stat

ALTER TABLE StudentsDeploySch.[Student`s stat]

DROP CONSTRAINT CK\_Students\_type;

-- Видалення колонки Type з таблиці Student`s stat

ALTER TABLE StudentsDeploySch.[Student`s stat]

DROP COLUMN Type;

-- Перейменування колонки TypeI в Type в таблиці Student`s stat

EXEC sp\_rename 'StudentsDeploySch.[Student`s stat].TypeI', 'Type', 'COLUMN';

-- Зміна типу колонки Type в таблиці Student`s stat

ALTER TABLE StudentsDeploySch.[Student`s stat]

ALTER COLUMN Type INT NOT NULL;

-- Додавання зовнішнього ключа до колонки Type в таблиці Student`s stat

ALTER TABLE StudentsDeploySch.[Student`s stat]

ADD CONSTRAINT FK\_Type FOREIGN KEY (Type) REFERENCES StudentsDeploySch.TypeSubject (id);

-- Включення перевірки обмежень для зовнішнього ключа Type в таблиці Student`s stat

ALTER TABLE StudentsDeploySch.[Student`s stat] CHECK CONSTRAINT FK\_Type;

GO

-- Оновлення значень колонки AuditoriumHours в таблиці Subjects\_plan

UPDATE StudentsDeploySch.Subjects\_plan

SET AuditoriumHours = Hours / 2

WHERE id\_discipline % 2 = 1;

-- Видалення записів в таблиці Student`s stat за певним умовою

DELETE

FROM StudentsDeploySch.[Student`s stat]

WHERE students\_id = 2;

-- Видалення записів в таблиці Students за певним умовою

DELETE

FROM StudentsDeploySch.Students

WHERE id = 2;

-- Вставка значень до таблиці Departments

INSERT INTO StudentsDeploySch.Departments (Name, Counter\_speciality, Deanerys\_auditorium)

VALUES

('TK', 4, 435),

('UPP', 1, 420),

('Mechaniks', 3, 246),

('PGS', 10, 315),

('Teach', 4, 5413),

('MT', 3, 123),

('Lokomotives', 2, 299),

('PM', 1, 518),

('GPM', 3, 216),

('Economics', 2, 222);

-- Вставка значень до таблиці Disciplines

INSERT INTO StudentsDeploySch.Disciplines (Name)

VALUES

('Algebra'),

('Geometria'),

('Phiziks'),

('Informatica'),

('Programming'),

('Buildings'),

('Electricity'),

('Databases'),

('Linear algebra'),

('OOP'),

('Economics');

-- Вставка значень до таблиці Teacher\_ranks

INSERT INTO StudentsDeploySch.Teacher\_ranks (rank\_name)

VALUES

('Aspirant'),

('Teacher'),

('Senior Teacher'),

('Docent'),

('Proffesor');

-- Вставка значень до таблиці Teachers

INSERT INTO StudentsDeploySch.Teachers (SecondName, FirstName, Teacher\_rank\_id)

VALUES

('Ivanov', 'Ivan', 5),

('Dmitriev', 'Vadim', 4),

('Sergeev', 'Nikolay', 4),

('Alexandrova', 'Anna', 2),

('Parksenov', 'Fedor', 3),

('Maksimova', 'Irina', 1),

('Nikolaenko', 'Pavel', 2),

('Andreev', 'Alexey', 1),

('Viy', 'Alexandr', 3),

('Danilov', 'Sergey', 5);

-- Вставка значень до таблиці TeachersSubjects

INSERT INTO StudentsDeploySch.TeachersSubjects (idTeachers, idDiscipli

**Додаток 2**

Процедури для роботи з базою даних що використовуються у програмі.

/\*Процедура для добавления студентов\*/

IF OBJECT\_ID('InsertStudent', 'P') IS NOT NULL

DROP PROCEDURE InsertStudent;

GO

CREATE PROCEDURE InsertStudent

@age AS int,

@FirstName AS nvarchar(50),

@SecondName AS nvarchar(50),

@departmentid AS int,

@specialityid AS int

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO StudentsDeploySch.Students(Age, FirstName, SecondName, department\_id, speciality\_id)

VALUES(@age, @FirstName, @SecondName, @departmentid, @specialityid)

END;

GO

IF OBJECT\_ID('SelectStudent', 'P') IS NOT NULL

DROP PROCEDURE SelectStudent;

GO

CREATE PROCEDURE SelectStudent

AS

BEGIN

SET NOCOUNT ON;

SELECT st.id, st.FirstName AS Имя, st.SecondName AS Фаммилия, st.Age AS Возраст, dp.Name AS Факультет, sp.Name AS Специальность

FROM StudentsDeploySch.Students AS st

JOIN StudentsDeploySch.Departments AS dp ON st.department\_id = dp.id

JOIN StudentsDeploySch.Speciality AS sp ON st.speciality\_id = sp.id

END;

GO

IF OBJECT\_ID('SearchStudentByName', 'P') IS NOT NULL

DROP PROCEDURE SearchStudentByName;

GO

CREATE PROCEDURE SearchStudentByName

@SecondName AS nvarchar(50)

AS

BEGIN

SET NOCOUNT ON;

SELECT st.id AS ID, st.FirstName AS Имя, st.SecondName AS Фаммилия, st.Age AS Возраст, dp.Name AS Факультет, sp.Name AS Специальность

FROM StudentsDeploySch.Students AS st

JOIN StudentsDeploySch.Departments AS dp ON st.department\_id = dp.id

JOIN StudentsDeploySch.Speciality AS sp ON st.speciality\_id = sp.id

WHERE st.SecondName LIKE @SecondName+'%'

END;

GO

/\*Процедура для обновления студентов\*/

IF OBJECT\_ID('UpdateStudent', 'P') IS NOT NULL

DROP PROCEDURE UpdateStudent;

GO

CREATE PROCEDURE UpdateStudent

@id AS int,

@age AS int,

@FirstName AS nvarchar(50),

@SecondName AS nvarchar(50),

@departmentid AS int,

@specialityid AS int

AS

BEGIN

UPDATE StudentsDeploySch.Students

SET

age = @age,

FirstName = @FirstName,

SecondName = @SecondName,

department\_id = @departmentid,

speciality\_id = @specialityid

WHERE id = @id

END;

GO

/\*Процедура для удаления студентов\*/

IF OBJECT\_ID('DeleteStudent', 'P') IS NOT NULL

DROP PROCEDURE DeleteStudent;

GO

CREATE PROCEDURE DeleteStudent

@id AS int,

@FirstName AS nvarchar(50),

@SecondName AS nvarchar(50)

AS

BEGIN

DELETE StudentsDeploySch.Students

WHERE id = @id AND FirstName = @FirstName AND SecondName = @SecondName

END;

GO

/\*Процедура для проверки существования студента\*/

IF OBJECT\_ID('ExistStudent', 'P') IS NOT NULL

DROP PROCEDURE ExistStudent;

GO

CREATE PROCEDURE ExistStudent

@id AS int

AS

BEGIN

SELECT COUNT(st.id)

FROM StudentsDeploySch.Students AS st

WHERE st.id = @id

END;

GO

/\*Процедура для добавления преподавателей\*/

IF OBJECT\_ID('InsertTeachers', 'P') IS NOT NULL

DROP PROCEDURE InsertTeachers;

GO

CREATE PROCEDURE InsertTeachers

@FirstName AS nvarchar(30),

@SecondName AS nvarchar(30),

@Teachers\_runk\_id AS int

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO StudentsDeploySch.Teachers(FirstName, SecondName, Teacher\_rank\_id)

VALUES(@FirstName, @SecondName, @Teachers\_runk\_id)

END;

GO

/\*Процедура для вывода всех преподавателей в GridView\*/

IF OBJECT\_ID('SelectTeachers', 'P') IS NOT NULL

DROP PROCEDURE SelectTeachers;

GO

CREATE PROCEDURE SelectTeachers

AS

BEGIN

SET NOCOUNT ON;

SELECT teach.id, teach.FirstName AS Имя, teach.SecondName AS Фаммилия, runk.rank\_name AS Степень

FROM StudentsDeploySch.Teachers AS teach

JOIN StudentsDeploySch.Teacher\_ranks AS runk ON Teacher\_rank\_id = runk.id

END;

GO

/\*Процедура для поиска преподавателей по фамилии\*/

IF OBJECT\_ID('SearchTeachersByName', 'P') IS NOT NULL

DROP PROCEDURE SearchTeachersByName;

GO

CREATE PROCEDURE SearchTeachersByName

@SecondName AS nvarchar(50)

AS

BEGIN

SET NOCOUNT ON;

SELECT teach.id, teach.FirstName AS Имя, teach.SecondName AS Фаммилия, runk.rank\_name AS Степень

FROM StudentsDeploySch.Teachers AS teach

JOIN StudentsDeploySch.Teacher\_ranks AS runk ON Teacher\_rank\_id = runk.id

WHERE teach.SecondName LIKE @SecondName+'%'

END;

GO

/\*Процедура обновления данных о преподавателях\*/

IF OBJECT\_ID('UpdateTeachers', 'P') IS NOT NULL

DROP PROCEDURE UpdateTeachers;

GO

CREATE PROCEDURE UpdateTeachers

@id AS int,

@FirstName AS nvarchar(50),

@SecondName AS nvarchar(50),

@Teachers\_runk\_id AS int

AS

BEGIN

UPDATE StudentsDeploySch.Teachers

SET

FirstName = @FirstName,

SecondName = @SecondName,

Teacher\_rank\_id = @Teachers\_runk\_id

WHERE id = @id

END;

GO

/\*Процедура удаления данных о преподавателях\*/

IF OBJECT\_ID('DeleteTeachers', 'P') IS NOT NULL

DROP PROCEDURE DeleteTeachers;

GO

CREATE PROCEDURE DeleteTeachers

@id AS int,

@FirstName AS nvarchar(50),

@SecondName AS nvarchar(50)

AS

BEGIN

DELETE StudentsDeploySch.Teachers

WHERE id = @id AND FirstName = @FirstName AND SecondName = @SecondName

END;

GO

/\*Проверка на существование преподавателя\*/

IF OBJECT\_ID('ExistTeachers', 'P') IS NOT NULL

DROP PROCEDURE ExistTeachers;

GO

CREATE PROCEDURE ExistTeachers

@Teachers\_id AS int

AS

BEGIN

SELECT COUNT(id)

FROM StudentsDeploySch.Teachers

WHERE id = @Teachers\_id

END;

GO

IF OBJECT\_ID('InsertDisciplines', 'P') IS NOT NULL

DROP PROCEDURE InsertDisciplines;

GO

CREATE PROCEDURE InsertDisciplines

@Name AS nvarchar(30),

@Subject\_code AS smallint

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO StudentsDeploySch.Disciplines(Name, Subject\_Code)

VALUES(@Name, @Subject\_code)

END;

GO

/\*Процедура для вывода всех дисциплин в GridView\*/

IF OBJECT\_ID('SelectDiscipline', 'P') IS NOT NULL

DROP PROCEDURE SelectDiscipline;

GO

CREATE PROCEDURE SelectDiscipline

AS

BEGIN

SET NOCOUNT ON;

SELECT dis.id, dis.Name, dis.Subject\_Code

FROM StudentsDeploySch.Disciplines AS dis

END;

GO

/\*Процедура для поиска дисциплин\*/

IF OBJECT\_ID('SearchDiscipline', 'P') IS NOT NULL

DROP PROCEDURE SearchDiscipline;

GO

CREATE PROCEDURE SearchDiscipline

@Name AS nvarchar(50)

AS

BEGIN

SET NOCOUNT ON;

SELECT dis.id, dis.Name, dis.Subject\_Code

FROM StudentsDeploySch.Disciplines AS dis

WHERE dis.Name LIKE @Name+'%'

END;

GO

/\*Процедура обновления дисциплин\*/

IF OBJECT\_ID('UpdateDiscipline', 'P') IS NOT NULL

DROP PROCEDURE UpdateDiscipline;

GO

CREATE PROCEDURE UpdateDiscipline

@id AS int,

@Name AS nvarchar(30),

@Subject\_code AS smallint

AS

BEGIN

UPDATE StudentsDeploySch.Disciplines

SET

Name = @Name,

Subject\_Code = @Subject\_code

WHERE id = @id

END;

GO

/\*Процедура удаления дисциплин\*/

IF OBJECT\_ID('DeleteDiscipline', 'P') IS NOT NULL

DROP PROCEDURE DeleteDiscipline;

GO

CREATE PROCEDURE DeleteDiscipline

@id AS int,

@Name AS nvarchar(80)

AS

BEGIN

DELETE StudentsDeploySch.Disciplines

WHERE id = @id AND Name = @Name

END;

GO

/\*----------СТАТИСТИКА---------------\*/

/\*Процедура для добавления оценок\*/

IF OBJECT\_ID('InsertStut', 'P') IS NOT NULL

DROP PROCEDURE InsertStut;

GO

CREATE PROCEDURE InsertStut

@Students\_id AS int,

@Subject\_id as int,

@Type as int,

@Semestr as int,

@Rating as float

AS

BEGIN

INSERT INTO StudentsDeploySch.[Student`s stat](Semestr, Type, Rating, students\_id, subject\_id)

VALUES(@Semestr, @Type, @Rating, @Students\_id, @Subject\_id)

END;

GO

/\*Процедура для добавления учебного плана для выбранного студента\*/

IF OBJECT\_ID('InsertStudentsPlan', 'P') IS NOT NULL

DROP PROCEDURE InsertStudentsPlan;

GO

CREATE PROCEDURE InsertStudentsPlan

@StudId AS int

AS

BEGIN

SET NOCOUNT ON;

INSERT INTO StudentsDeploySch.[Student`s stat](Semestr, Type, Rating, students\_id, subject\_id)

VALUES (1,1,0,@StudId,1),(2,2,0,@StudId,2),(1,1,0,@StudId,3),(2,2,0,@StudId,4),(1,1,0,@StudId,5),

(2,2,0,@StudId,6),(1,1,0,@StudId,7),(2,2,0,@StudId,8),(1,1,0,@StudId,9),(2,2,0,@StudId,10),

(1,1,0,@StudId,11),(2,2,0,@StudId,12),(1,1,0,@StudId,13),(2,2,0,@StudId,15)

END;

GO

/\*Процедура для вывода всей статистики всех студентов в GridView\*/

IF OBJECT\_ID('SelectALLStudentStut', 'P') IS NOT NULL

DROP PROCEDURE SelectALLStudentStut;

GO

CREATE PROCEDURE SelectALLStudentStut

AS

BEGIN

SET NOCOUNT ON;

SELECT Stut.id AS ID, St.FirstName AS Имя, St.SecondName AS Фамилия, Dep.Name AS Факультет, Sp.Name AS Специальность,

Dis.Name AS Дисциплина, stut.Semestr AS Семестр, Ty.Name AS Тип, Stut.Rating AS Рейтинг, St.id AS StudentsId

FROM StudentsDeploySch.[Student`s stat] AS Stut

JOIN StudentsDeploySch.Students AS St ON stut.students\_id = St.id

JOIN StudentsDeploySch.Disciplines AS Dis ON stut.subject\_id = Dis.id

JOIN StudentsDeploySch.TypeSubject AS Ty ON Stut.Type = Ty.id

JOIN StudentsDeploySch.Departments AS Dep ON St.department\_id = Dep.id

JOIN StudentsDeploySch.Speciality AS Sp ON St.speciality\_id = Sp.id

END;

GO

IF OBJECT\_ID('SelectStudentStut', 'P') IS NOT NULL

DROP PROCEDURE SelectStudentStut;

GO

CREATE PROCEDURE SelectStudentStut

@id AS int,

@FirstName AS nvarchar(50),

@SecondName AS nvarchar(50)

AS

BEGIN

SET NOCOUNT ON;

SELECT Stut.id AS ID, St.FirstName AS Имя, St.SecondName AS Фамилия, Dep.Name AS Факультет, Sp.Name AS Специальность,

Dis.Name AS Дисциплина, stut.Semestr AS Семестр, Ty.Name AS Тип, Stut.Rating AS Рейтинг, St.id AS StudentsId

FROM StudentsDeploySch.Students AS St

LEFT JOIN StudentsDeploySch.[Student`s stat] AS Stut ON stut.students\_id = St.id

JOIN StudentsDeploySch.Disciplines AS Dis ON stut.subject\_id = Dis.id

JOIN StudentsDeploySch.TypeSubject AS Ty ON Stut.Type = Ty.id

JOIN StudentsDeploySch.Departments AS Dep ON St.department\_id = Dep.id

JOIN StudentsDeploySch.Speciality AS Sp ON St.speciality\_id = Sp.id

WHERE stut.students\_id = @id AND St.FirstName = @FirstName AND St.SecondName = @SecondName

END;

GO

/\*Процедура для вывода задолженностей выбранного студента в GridView\*/

IF OBJECT\_ID('SelectStudentArrears', 'P') IS NOT NULL

DROP PROCEDURE SelectStudentArrears;

GO

CREATE PROCEDURE SelectStudentArrears

@id AS int,

@FirstName AS nvarchar(50),

@SecondName AS nvarchar(50)

AS

BEGIN

SET NOCOUNT ON;

SELECT Stut.id AS ID, St.FirstName AS Имя, St.SecondName AS Фамилия, Dep.Name AS Факультет, Sp.Name AS Специальность,

Dis.Name AS Дисциплина, stut.Semestr AS Семестр, Ty.Name AS Тип, Stut.Rating AS Рейтинг, St.id AS StudentsId

FROM StudentsDeploySch.Students AS St

JOIN StudentsDeploySch.[Student`s stat] AS Stut ON stut.students\_id = St.id

JOIN StudentsDeploySch.Disciplines AS Dis ON stut.subject\_id = Dis.id

JOIN StudentsDeploySch.TypeSubject AS Ty ON Stut.Type = Ty.id

JOIN StudentsDeploySch.Departments AS Dep ON St.department\_id = Dep.id

JOIN StudentsDeploySch.Speciality AS Sp ON St.speciality\_id = Sp.id

WHERE stut.students\_id = @id AND St.FirstName = @FirstName AND St.SecondName = @SecondName AND stut.Rating = 0

END;

GO

/\*Процедура для вывода задолженностей всех студентов в GridView\*/

IF OBJECT\_ID('SelectALLStudentArrears', 'P') IS NOT NULL

DROP PROCEDURE SelectALLStudentArrears;

GO

CREATE PROCEDURE SelectALLStudentArrears

AS

BEGIN

SET NOCOUNT ON;

SELECT Stut.id AS ID, St.FirstName AS Имя, St.SecondName AS Фамилия, Dep.Name AS Факультет, Sp.Name AS Специальность,

Dis.Name AS Дисциплина, stut.Semestr AS Семестр, Ty.Name AS Тип, Stut.Rating AS Рейтинг, St.id AS StudentsId

FROM StudentsDeploySch.[Student`s stat] AS Stut

JOIN StudentsDeploySch.Students AS St ON stut.students\_id = St.id

JOIN StudentsDeploySch.Disciplines AS Dis ON stut.subject\_id = Dis.id

JOIN StudentsDeploySch.TypeSubject AS Ty ON Stut.Type = Ty.id

JOIN StudentsDeploySch.Departments AS Dep ON St.department\_id = Dep.id

JOIN StudentsDeploySch.Speciality AS Sp ON St.speciality\_id = Sp.id AND stut.Rating = 0

END;

GO

/\*Процедура обновления успеваемости\*/

IF OBJECT\_ID('UpdateStut', 'P') IS NOT NULL

DROP PROCEDURE UpdateStut;

GO

CREATE PROCEDURE UpdateStut

@id AS int,

@Students\_id AS int,

@Subject\_id as int,

@Type as int,

@Semestr as int,

@Rating as float

AS

BEGIN

SET NOCOUNT ON;

UPDATE StudentsDeploySch.[Student`s stat]

SET

Semestr = @Semestr,

Type = @Type,

Rating = @Rating,

students\_id = @Students\_id,

subject\_id = @Subject\_id

WHERE id = @id

END;

GO

IF OBJECT\_ID('ExistStudentStut', 'P') IS NOT NULL

DROP PROCEDURE ExistStudentStut;

GO

CREATE PROCEDURE ExistStudentStut

@id AS int

AS

BEGIN

SELECT COUNT(stut.id)

FROM StudentsDeploySch.[Student`s stat] AS stut

WHERE stut.students\_id = @id

END;

GO

/\*Процедура удаления учебного плана студента\*/

IF OBJECT\_ID('DeleteStudentsPlan', 'P') IS NOT NULL

DROP PROCEDURE DeleteStudentsPlan;

GO

CREATE PROCEDURE DeleteStudentsPlan

@student\_id AS int

AS

BEGIN

SET NOCOUNT ON;

DELETE StudentsDeploySch.[Student`s stat]

WHERE students\_id = @student\_id

END;

GO

/\*Процедура удаления оценки студента\*/

IF OBJECT\_ID('DeleteStudentsMark', 'P') IS NOT NULL

DROP PROCEDURE DeleteStudentsMark;

GO

CREATE PROCEDURE DeleteStudentsMark

@id AS int

AS

BEGIN

DELETE StudentsDeploySch.[Student`s stat]

WHERE id = @id

END;

GO

/\*-----------ПЛАН ДИСЦИПЛИН---------\*/

/\*Добавление плана\*/

IF OBJECT\_ID('InsertSubPlan', 'P') IS NOT NULL

DROP PROCEDURE InsertSubPlan;

GO

CREATE PROCEDURE InsertSubPlan

@Discipline\_id AS int,

@Hours as int,

@AuditHours as int

AS

BEGIN

INSERT INTO StudentsDeploySch.Subjects\_plan(id\_discipline, Hours, AuditoriumHours)

VALUES(@Discipline\_id, @Hours, @AuditHours)

END;

GO

/\*Проверка на существование дисциплины\*/

IF OBJECT\_ID('ExistDiscipline', 'P') IS NOT NULL

DROP PROCEDURE ExistDiscipline;

GO

CREATE PROCEDURE ExistDiscipline

@Discipline\_id AS int

AS

BEGIN

SELECT COUNT(dis.id)

FROM StudentsDeploySch.Disciplines AS dis

WHERE dis.id = @Discipline\_id

END;

GO

IF OBJECT\_ID('SearchDisciplinePlan', 'P') IS NOT NULL

DROP PROCEDURE SearchDisciplinePlan;

GO

CREATE PROCEDURE SearchDisciplinePlan

@Name AS nvarchar(50)

AS

BEGIN

SET NOCOUNT ON;

SELECT splan.id, dis.Name AS Дисциплина, dis.Subject\_Code AS Код, splan.Hours AS Часы, splan.AuditoriumHours AS [Аудиторные часы]

FROM StudentsDeploySch.Disciplines AS dis

JOIN StudentsDeploySch.Subjects\_plan AS splan ON dis.id = splan.id\_discipline

WHERE dis.Name LIKE @Name+'%'

END;

GO

IF OBJECT\_ID('SelectDisciplinePlan', 'P') IS NOT NULL

DROP PROCEDURE SelectDisciplinePlan;

GO

CREATE PROCEDURE SelectDisciplinePlan

AS

BEGIN

SELECT splan.id, dis.Name AS Дисциплина, dis.Subject\_Code AS Код, splan.Hours AS Часы, splan.AuditoriumHours AS [Аудиторные часы]

FROM StudentsDeploySch.Disciplines AS dis

JOIN StudentsDeploySch.Subjects\_plan AS splan ON dis.id = splan.id\_discipline

END;

GO

/\*Процедура для обновления плана дисциплин\*/

IF OBJECT\_ID('UpdateDisciplinePlan', 'P') IS NOT NULL

DROP PROCEDURE UpdateDisciplinePlan;

GO

CREATE PROCEDURE UpdateDisciplinePlan

@Plan\_id AS int,

@Hours as int,

@AuditHours as int

AS

BEGIN

SET NOCOUNT ON;

UPDATE StudentsDeploySch.Subjects\_plan

SET

Hours = @Hours,

AuditoriumHours = @AuditHours

WHERE id = @Plan\_id

END;

GO

/\*Процедура для удаления плана дисциплин\*/

IF OBJECT\_ID('DeleteSubjectPlan', 'P') IS NOT NULL

DROP PROCEDURE DeleteSubjectPlan;

GO

CREATE PROCEDURE DeleteSubjectPlan

@id AS int

AS

BEGIN

DELETE StudentsDeploySch.Subjects\_plan

WHERE id = @id

END;

GO

/\*-----ФАКУЛЬТЕТЫ И СПЕЦИАЛЬНОСТИ-----\*/

/\*Добавление факультета\*/

IF OBJECT\_ID('InsertDepartment', 'P') IS NOT NULL

DROP PROCEDURE InsertDepartment;

GO

CREATE PROCEDURE InsertDepartment

@Auditorium AS int,

@Name AS nvarchar(50)

AS

BEGIN

INSERT INTO StudentsDeploySch.Departments(Deanerys\_auditorium, Name)

VALUES(@Auditorium, @Name)

END;

GO

/\*Процедура для выборки факультетов\*/

IF OBJECT\_ID('SelectDepartments', 'P') IS NOT NULL

DROP PROCEDURE SelectDepartments;

GO

CREATE PROCEDURE SelectDepartments

AS

BEGIN

SELECT dep.id, dep.Name AS Факультет, dep.Deanerys\_auditorium AS Аудитория

FROM StudentsDeploySch.Departments AS dep

END;

GO

/\*Процедура для поиска факультета\*/

IF OBJECT\_ID('SearchDepartments', 'P') IS NOT NULL

DROP PROCEDURE SearchDepartments;

GO

CREATE PROCEDURE SearchDepartments

@Name AS nvarchar(50)

AS

BEGIN

SET NOCOUNT ON;

SELECT dep.id, dep.Name AS Факультет, dep.Deanerys\_auditorium AS Аудитория

FROM StudentsDeploySch.Departments AS dep

WHERE dep.Name LIKE @Name+'%'

END;

GO

IF OBJECT\_ID('UpdateDeparments', 'P') IS NOT NULL

DROP PROCEDURE UpdateDeparments;

GO

CREATE PROCEDURE UpdateDeparments

@id AS int,

@Name as nvarchar(50),

@Auditorium as int

AS

BEGIN

SET NOCOUNT ON;

UPDATE StudentsDeploySch.Departments

SET

Name = @Name,

Deanerys\_auditorium = @Auditorium

WHERE id = @id

END;

GO

/\*Проверка на существование факультета\*/

IF OBJECT\_ID('ExistDepartment', 'P') IS NOT NULL

DROP PROCEDURE ExistDepartment;

GO

CREATE PROCEDURE ExistDepartment

@Department\_id AS int

AS

BEGIN

SELECT COUNT(id)

FROM StudentsDeploySch.Departments

WHERE id = @Department\_id

END;

GO

/\*Процедура удаления факультетов\*/

IF OBJECT\_ID('DeleteDepartments', 'P') IS NOT NULL

DROP PROCEDURE DeleteDepartments;

GO

CREATE PROCEDURE DeleteDepartments

@id AS int

AS

BEGIN

DELETE StudentsDeploySch.Departments

WHERE id = @id

END;

GO

/\*Добавление специальности\*/

IF OBJECT\_ID('InsertSpeciality', 'P') IS NOT NULL

DROP PROCEDURE InsertSpeciality;

GO

CREATE PROCEDURE InsertSpeciality

@Id\_department AS int,

@Name AS nvarchar(100)

AS

BEGIN

INSERT INTO StudentsDeploySch.Speciality(IdDepartment, Name)

VALUES(@Id\_department, @Name)

END;

GO

/\*Процедура для выборки специальностей\*/

IF OBJECT\_ID('SelectSpeciality', 'P') IS NOT NULL

DROP PROCEDURE SelectSpeciality;

GO

CREATE PROCEDURE SelectSpeciality

AS

BEGIN

SELECT sp.id, dep.Name AS Факультет, sp.Name AS Специальность, sp.IdDepartment, dep.Deanerys\_auditorium

FROM StudentsDeploySch.Speciality AS sp

JOIN StudentsDeploySch.Departments AS dep ON sp.IdDepartment = dep.id

END;

GO

/\*Процедура для поиска специальностей\*/

IF OBJECT\_ID('SearchSpeciality', 'P') IS NOT NULL

DROP PROCEDURE SearchSpeciality;

GO

CREATE PROCEDURE SearchSpeciality

@Name AS nvarchar(100)

AS

BEGIN

SET NOCOUNT ON;

SELECT sp.id, dep.Name AS Факультет, sp.Name AS Специальность, sp.IdDepartment, dep.Deanerys\_auditorium

FROM StudentsDeploySch.Speciality AS sp

JOIN StudentsDeploySch.Departments AS dep ON sp.IdDepartment = dep.id

WHERE sp.Name LIKE @Name+'%'

END;

GO

/\*Процедура обновления специальностей\*/

IF OBJECT\_ID('UpdateSpeciality', 'P') IS NOT NULL

DROP PROCEDURE UpdateSpeciality;

GO

CREATE PROCEDURE UpdateSpeciality

@id AS int,

@Id\_department AS int,

@Name AS nvarchar(100)

AS

BEGIN

SET NOCOUNT ON;

UPDATE StudentsDeploySch.Speciality

SET

IdDepartment = @Id\_department,

Name = @Name

WHERE id = @id

END;

GO

IF OBJECT\_ID('ExistSpeciality', 'P') IS NOT NULL

DROP PROCEDURE ExistSpeciality;

GO

CREATE PROCEDURE ExistSpeciality

@Speciality\_id AS int

AS

BEGIN

SELECT COUNT(id)

FROM StudentsDeploySch.Speciality

WHERE id = @Speciality\_id

END;

GO

/\*Процедура для удаления специальности\*/

IF OBJECT\_ID('DeleteSpeciality', 'P') IS NOT NULL

DROP PROCEDURE DeleteSpeciality;

GO

CREATE PROCEDURE DeleteSpeciality

@id AS int

AS

BEGIN

DELETE StudentsDeploySch.Speciality

WHERE id = @id

END;

GO

/\*Процедура для назначения дисциплин преподавателям\*/

IF OBJECT\_ID('InsertDisTeach', 'P') IS NOT NULL

DROP PROCEDURE InsertDisTeach;

GO

CREATE PROCEDURE InsertDisTeach

@TeachId AS int,

@DisId AS nvarchar(100)

AS

BEGIN

INSERT INTO StudentsDeploySch.TeachersSubjects(idTeachers, idDiscipline)

VALUES(@TeachId, @DisId)

END;

GO

/\*Процедура для вывода дисциплин, которые ведет выбранный преподаватель\*/

IF OBJECT\_ID('SelectTeachDis', 'P') IS NOT NULL

DROP PROCEDURE SelectTeachDis;

GO

CREATE PROCEDURE SelectTeachDis

@TeachId AS int

AS

BEGIN

SELECT tsub.id, tich.FirstName AS Имя, tich.SecondName AS Фамилия, dis.Name AS Дисциплина, tich.id

FROM StudentsDeploySch.TeachersSubjects AS tsub

JOIN StudentsDeploySch.Teachers AS tich ON tsub.idTeachers = tich.id

JOIN StudentsDeploySch.Disciplines AS dis ON tsub.idDiscipline = dis.id

WHERE tsub.idTeachers = @TeachId

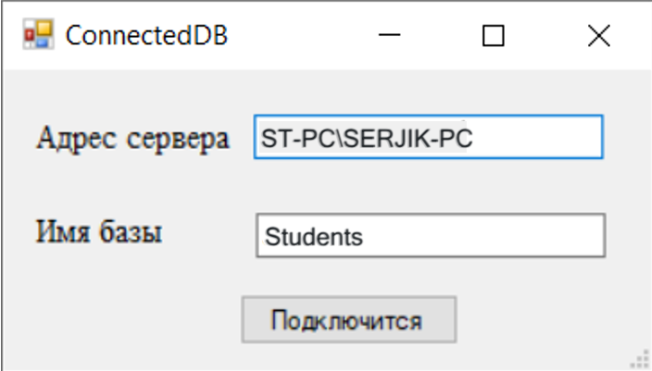
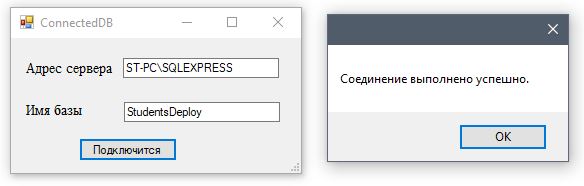
END;

GO

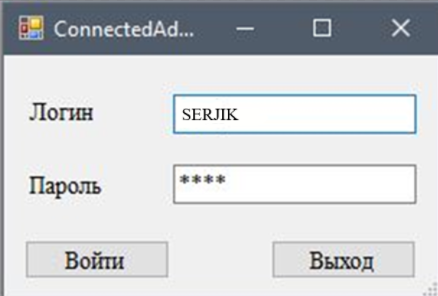
/\*процедура для вывборки дисциплин которые ведут все преподаватели\*/

IF OBJECT\_ID('SelectAllTeachDis', 'P') IS NOT NULL

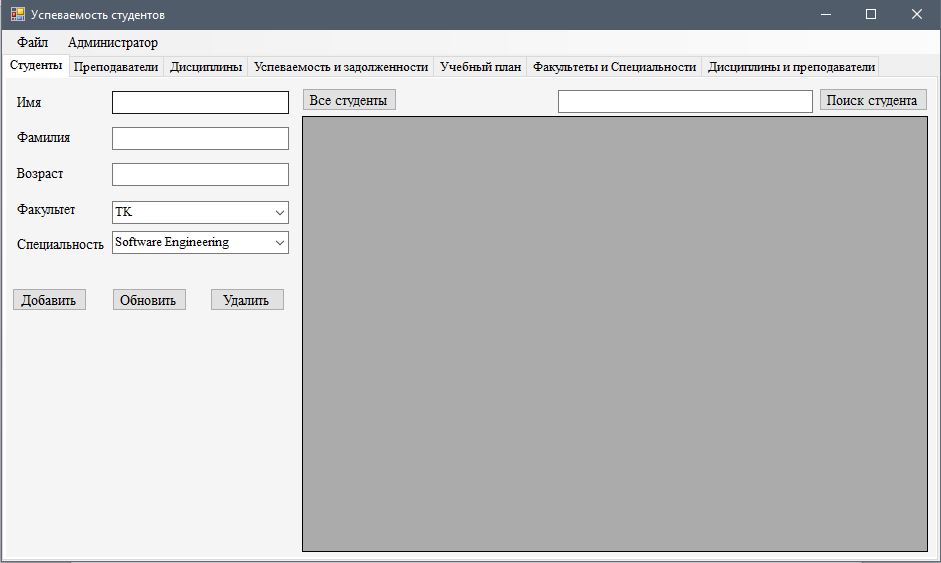
**Додаток 3**

Мал. 5. Підключення до бази даних.



Мал. 6. Вікно авторизації адміністратора.



Мал. 7. Головне вікно програми.

***Запити до схеми «Зірка» з результатами виконання.***

-- Вибірка інформації про студентів, дисципліни та їх оцінки

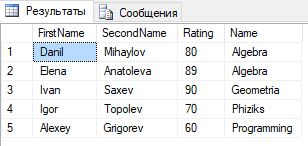
SELECT stud.FirstName AS 'Ім'я\_студента', stud.SecondName AS 'Прізвище\_студента', inf.Rating AS 'Оцінка', dis.Name AS 'Назва\_дисципліни'

FROM dbo.Info\_Facts AS inf

JOIN dbo.Disciplines\_Met AS dis ON inf.ID\_Disciplines = dis.id

JOIN dbo.Students\_Met AS stud ON stud.id = inf.ID\_Students

GO



-- Вибірка інформації про викладачів, дисципліни, які вони викладають

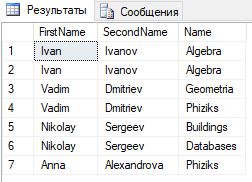
SELECT tich.FirstName AS 'Ім'я\_викладача', tich.SecondName AS 'Прізвище\_викладача', dis.Name AS 'Назва\_дисципліни'

FROM dbo.Info\_Facts AS inf

JOIN dbo.Teachers\_Met AS tich ON tich.id = inf.ID\_Teachers

JOIN dbo.Disciplines\_Met AS dis ON dis.id = inf.ID\_Disciplines

GO



***Запити до схеми «Сніжинка» з результатами виконання***

-- Вибірка імені, прізвища студента, назви відділення та спеціальності для студентів відділення 'TK'

SELECT st.FirstName AS 'Ім'я\_студента', st.SecondName AS 'Прізвище\_студента', dep.Name AS 'Назва\_відділення', sp.Name AS 'Назва\_спеціальності'

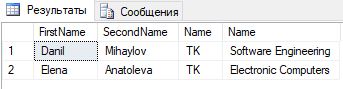
FROM dbo.Info\_Facts AS inf

JOIN dbo.Students\_Met AS st ON st.id = inf.ID\_Students

JOIN dbo.Departments\_Met AS dep ON dep.id = inf.ID\_Departments

JOIN dbo.Speciality\_Met AS sp ON sp.id = inf.ID\_Speciality

WHERE dep.Name = 'TK'



-- Вибірка імені, прізвища студента, назви дисципліни, оцінки та типу оцінки для студентів з оцінкою 0 і типом 'PK'

SELECT st.FirstName AS 'Ім'я\_студента', st.SecondName AS 'Прізвище\_студента', dis.Name AS 'Назва\_дисципліни', inf.Rating AS 'Оцінка', inf.Type AS 'Тип\_оцінки'

FROM dbo.Students\_Met AS st

JOIN dbo.Info\_Facts AS inf ON inf.ID\_Students = st.id

JOIN dbo.Disciplines\_Met AS dis ON inf.ID\_Disciplines = dis.id

WHERE inf.Rating = 0 AND inf.Type = 'PK'

