РЕФЕРАТ

Пояснювальна записка: ___ с., ___ рис., ___ табл., __ дод., ___ джерел.

Об'єкт розробки: інтегроване середовище розробки для СУБД MySQL.

Мета кваліфікаційної роботи: надати графічний клієнт для створення та адміністрування СУБД MySQL.

У вступі розглядається аналіз та сучасний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, наведено обґрунтування актуальності теми та уточнюється постановка завдання.

У першому розділі проведено аналіз предметної галузі, визначено актуальність завдання та призначення розробки, розроблена постановка завдання, задані вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі виконано аналіз існуючих рішень, обрано вибір платформи для розробки, виконано проектування і розробка програми, наведено опис алгоритму і структури функціонування програми, визначені вхідні і вихідні дані, наведені характеристики складу параметрів технічних засобів, описаний виклик та завантаження програми, описана робота програми.

В економічному розділі визначено трудомісткість розробленої інформаційної системи, проведений підрахунок вартості роботи по створенню програми та розраховано час на його створення.

Практичне значення роботи полягає в відкритості розробленого проєкту, який дозволяє спостерігати за всіма виконуваними запитами, що для новачків особливо корисно, так як сприяє візуальному запам’ятовуванню, а наглядність структури інтерфейсу сприяє мотивації вчитись та працювати в напрямку створення баз даних.

Актуальність теми кваліфікаційної роботи визначається тим, що наразі існують програми, створені для професіоналів, які мають певний стаж роботи з СУБД MySQL. Вони нагромаджені багатим функціоналом, але більшість з них використовуються в окремих випадках, тому в них легко загубитись новачку, який лише знайомиться з базами даних, що може відбити бажання вчитись. В даній кваліфікаційній роботі розроблено програмний модуль, який призначений для полегшення розуміння СУБД та її швидкому створенню і налаштуванню.

Список ключових слів: ПРОГРАМНИЙ ПРОДУКТ, МОДУЛЬ, ФОРМА, КОМПОНЕНТ, ІНФОРМАЦІЯ, ФАЙЛ, ЗАПИТ, БАЗА ДАНИХ, MYSQL.

ABSTRACT

Explanatory note: ___ pp., ___ fig., ___ table, __ appendix, ___ sources.

Development object: integrated development environment for MySQL database.

The purpose of the qualification work: to provide a graphical client for the creation and administration of MySQL databases.

The introduction considers the analysis and the current state of the problem, specifies the purpose of the qualification work and the field of its application, provides a justification for the relevance of the topic and clarifies the problem.

In the first section the analysis of the subject area is carried out, the urgency of the task and purpose of development is defined, the statement of the task is developed, requirements to software realization, technologies and software are set.

The second section analyzes existing solutions, selects a platform for development, performs design and development of the program, describes the algorithm and structure of the program, defines input and output data, provides characteristics of the parameters of hardware, describes the call and download of the program, describes the program .

The economic section determines the complexity of the developed information system, calculates the cost of work to create a program and calculates the time for its creation.

The practical significance of the work lies in the openness of the developed project, which allows you to monitor all requests, which is especially useful for beginners, as it promotes visual memorization, and clarity of the interface structure motivates to learn and work towards creating databases.

The relevance of the topic of qualification work is determined by the fact that currently there are programs designed for professionals who have some experience with DBMS MySQL. They are rich in functionality, but most of them are used in some cases, so they are easy to get lost in a beginner who is just getting acquainted with databases, which may discourage the desire to learn. In this qualification work, a software module has been developed, which is designed to facilitate the understanding of the database and its rapid creation and configuration.

List of keywords: SOFTWARE PRODUCT, MODULE, FORM, COMPONENT, INFORMATION, FILE, REQUEST, DATABASE, MYSQL.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних;

ОС – операційна система;

ПК – персональний комп’ютер;

СУБД, СКБД – Database Management System, DBMS, систе́ма управління ба́зами да́них

IDE – Integrated development environment, Інтегроване середовище розробки (ICP);

ODBC – Open DataBase Connectivity, відкритий прикладний програмний інтерфейс доступу до баз даних.

ЗМІСТ

ВСТУП

Багато компаній створюють різні багатофункціональні програми для полегшення управління, розробки та адміністрування баз даних.

Більшість реляційних баз даних складаються з двох окремих компонентів: «back-end», де зберігаються дані і «front-end» - призначений для користувача інтерфейс для взаємодії з даними. Цей тип конструкції паралелізує дворівневу модель програмування, яка відділяє шар даних від призначеного для користувача інтерфейсу і дозволяє користувачам значно облегшити роботу зі створення та використання баз даних. Ця модель відкриває нові можливості для розробників, які створюють свої додатки для взаємодії з різними базами даних.

Мета кваліфікаційної роботи полягає в наданні користувачам баз даних, які використовують СУБД MySQL, зручного графічного середовища для інтуїтивного розуміння структури, швидкого створення та налаштування бази даних.

Відкритість проєкту дозволяє спостерігати за всіма виконуваними запитами, що для новачків особливо корисно, так як сприяє візуальному запам’ятовуванню, а наглядність структури інтерфейсу сприяє мотивації вчитись та працювати в напрямку створення баз даних.

Наразі існують програми, створені для професіоналів, що мають певний стаж роботи з СУБД MySQL. Вони нагромаджені багатим функціоналом, але більшість з них використовуються в окремих випадках, тому в них легко загубитись новачку, який лише знайомиться з базами даних, що може відбити бажання вчитись.

В даній кваліфікаційній роботі бакалавра засобами середовища Qt Creator 4.11, розроблено програмний модуль, який призначений для полегшення розуміння СУБД та її швидкому створенню і налаштуванню.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Загальні відомості з предметної галузі

1.1.1. Загальні поняття створення баз даних

Систе́ма управління ба́зами да́них (СУБД, СКБД англ. Database Management System, DBMS) - набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних. Надає можливості створення, збереження, оновлення та пошуку інформації в базах даних з контролем доступу до даних.

Основні характеристики СКБД:

– контроль за надлишковістю даних;
– несуперечливість даних;
– підтримка цілісності бази даних (коректність та несуперечливість);
– цілісність описується за допомогою обмежень;
– незалежність прикладних програм від даних;
– спільне використання даних;
– підвищений рівень безпеки.

Можливості СКБД:

1. Дозволяється створювати БД (здійснюється за допомогою мови визначення даних DDL (Data Definition Language)).

2. Дозволяється додавання, оновлення, видалення та читання інформації з БД (за допомогою мови маніпулювання даними DML, яку часто називають мовою запитів).

3. Можна надавати контрольований доступ до БД за допомогою:

– системи забезпечення захисту, яка запобігає несанкціонованому доступу до БД;

– системи керування паралельною роботою прикладних програм, яка контролює процеси спільного доступу до БД;

– система відновлення - дозволяє відновлювати БД до

попереднього несуперечливого стану, що був порушений в результаті збою апаратного або програмного забезпечення

Архітектура СКБД: існує трирівнева система організації СКБД ANSI-SPARC, при якій існує незалежний рівень для ізоляції програми від особливостей представлення даних на нижчому рівні.

Рівні:

1. Зовнішній - представлення БД з точки зору користувача.
2. Концептуальний - узагальнене представлення БД, описує які дані зберігаються в БД і зв'язки між ними. Підтримує зовнішні представлення, підтримується внутрішнім рівнем.
3. Внутрішній - фізичне представлення БД в комп'ютері.

Логічна незалежність - це повна захищеність зовнішніх моделей від змін, що вносяться в концептуальну модель.

Фізична незалежність - це захищеність концептуальної моделі від змін, які вносяться у внутрішню модель. [6]

Адміністрування баз даних це функція управління та підтримки програмного забезпечення систем управління базами даних (СУБД). Основні програмні засоби СУБД, такі як Oracle, IBM DB2 і Microsoft SQL Server, потребують постійного управління. Таким чином, корпорації, які використовують програмне забезпечення СУБД, часто наймають спеціалістів з інформаційних технологій, які називаються адміністраторами баз даних.

Обов'язки адміністрування баз даних:

– встановлення, налаштування та оновлення серверного програмного забезпечення баз даних та супутніх продуктів;

– оцінка функцій бази даних та продуктів, пов'язаних із базою даних;

– створення та підтримка політики та процедури резервного копіювання та відновлення;

– піклування про розробку та впровадження бази даних;

– впровадження та підтримка безпеки бази даних (створення та підтримка користувачів та ролі, призначення привілеїв);

– налаштування бази даних та моніторинг продуктивності;

– налаштування програм та моніторинг продуктивності;

– налаштування та підтримка документації та стандартів;

– планування зростання та змін (планування потужності);

– робота як частина команди та надання підтримки 24x7, коли це необхідно;

– виконання загальних технічних усунень несправностей та визначення мінусів;

– відновлення бази даних.

Є три типи адміністраторів баз даних:

1. Системні адміністратори баз даних (також звані фізичні адміністратори баз даних, операційні адміністратори баз даних або адміністратори підтримки баз даних): зосереджуються на фізичних аспектах адміністрування баз даних, таких як встановлення СУБД, конфігурація, виправлення, покращення, резервне копіювання, відновлення, оновлення, оптимізація продуктивності, обслуговування та аварійне відновлення .

2. Адміністратори розробки баз даних: зосереджуються на логічних аспектах і аспектах управління базами даних, таких як розробка та підтримка моделі даних, генерація DDL (мови визначення даних), написання та налаштування SQL, кодування збережених процедур, співпраця з розробниками, що допомагає з вибором найбільш відповідних можливостей СУБД / функціональностей та інших передвиробничих заходів.

3. Адміністратори програмного забезпечення баз даних: зазвичай зустрічаються в організаціях, які придбали програмне забезпечення третьої сторони, такі як ERP (планування ресурсів підприємства) та CRM (системи управління взаємовідносинами з клієнтами). Прикладами таких прикладних програм є Oracle Applications, Siebel і PeopleSoft (обидві тепер є частиною Oracle Corp.) і SAP. Адміністратори програмного забезпечення баз даних поширюють огорожу між СУБД і прикладним програмним забезпеченням та несуть відповідальність за те, щоб програма була повністю оптимізована для

бази даних і навпаки. Зазвичай вони керують усіма компонентами програми, які взаємодіють з базою даних і виконують такі дії, як встановлення та виправлення програм, оновлення програм, клонування баз даних, побудова та виконання процедур очищення даних, керування процесами завантаження даних, тощо.

Хоча люди зазвичай спеціалізуються на одному типі адміністрування баз даних, у невеликих організаціях не рідко можна знайти окрему особу або групу, яка виконує більше одного типу адміністрування баз даних.

Ступінь автоматизації управління базою даних диктує навички та кадровий склад, необхідний для управління базами даних. Робота з базами даних складна, повторювана, трудомісткою і вимагає значної підготовки. Оскільки бази даних містять цінні та критично важливі дані, компанії зазвичай шукають кандидатів з багаторічним досвідом. Адміністрування баз даних часто вимагає, щоб адміністратори баз даних працювали в неробочий час (наприклад, для планового простою, у випадку відключення, пов'язаного з базою даних, або якщо продуктивність значно знижена).

Одною з ключових необхідних, що часто забувається, навичок при виборі адміністратора баз даних є відновлення бази даних (частина аварійного відновлення). Це не випадок «якщо», але випадок «коли» база даних зазнає невдачі, починаючи від простого збою до повної катастрофічної невдачі. Невдачею можуть бути пошкодження даних, помилка носія або помилки, викликані користувачем. У будь-якій ситуації адміністратор баз даних повинен мати навички відновлення бази даних до певного моменту часу, щоб запобігти втраті даних.

1.1.2. Засоби адміністрування баз даних

Часто програмне забезпечення СУБД постачається з певними інструментами, які допомагають адміністраторам баз даних керувати СУБД. Такі інструменти називаються рідними інструментами. Наприклад, Microsoft

SQL Server поставляється з SQL Server Management Studio, а Oracle має такі інструменти, як SQL * Plus і Oracle Enterprise Manager / Grid Control. Крім того, треті сторони, такі як BMC, Quest Software, Embarcadero Technologies і SQL Maestro Group, пропонують графічні інструменти для моніторингу СУБД і допомагають адміністраторам баз даних легше виконувати певні функції всередині бази даних.

Інший вид програмного забезпечення для баз даних існує для керування наданням нових баз даних й управління існуючими базами даних та їх відповідними ресурсами. Процес створення нової бази даних може складатися з сотень або тисяч унікальних кроків від задоволення передумов до налаштування резервних копій, де кожен крок повинен бути успішним до того, як почнеться наступний. Очікується, що людина не зможе завершити цю процедуру точно таким же чином раз за разом - саме тоді, коли існує кілька баз даних. З ростом кількості адміністраторів баз даних без автоматизації кількість унікальних конфігурацій часто стає дорогою / важкою для підтримки. Всі ці складні процедури можуть бути змодельовані кращими адміністраторами баз даних у програмне забезпечення автоматизації баз даних та можуть бути виконані стандартними адміністраторами баз даних. Програмне забезпечення було створено спеціально для підвищення надійності та відтворюваності цих процедур, таких як Stratavia «s Data Palette і GridApp системи Clarity. [5]

1.1.3. Інтегроване середовище розробки

Інтегроване середовище розробки (ICP, англ. Integrated development environment або англ. IDE) - комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм. Більшість сучасних середовищ розробки мають можливість автодоповнення коду.

Деякі середовища розробки містять компілятор, інтерпретатор або ж

обидва (наприклад NetBeans та Eclipse), інші не містять жодного з них (SharpDevelop та Lazarus). Деякі інтегровані середовища розробки містять систему керування версіями або інструменти для полегшення розробки графічного інтерфейсу користувача (GUI) (XCode, Embarcadero Delphi). Багато сучасних ІСР містять інспектор класів, інспектор об'єктів, схему ієрархії класів для полегшення об'єктно-орієнтованої розробки програмного забезпечення.

Інтегровані середовища розробки створені для того, щоб максимізувати продуктивність програміста, надавши йому пов'язані інструменти розробки зі схожими інтерфейсами як одну програму, в якій відбуватиметься весь процес розробки й яка надає необхідні функції для модифікації, компілювання, розгортання та налагодження програмного забезпечення. Протилежним до цього є підхід до розробки ПЗ, під час якого використовуються окремі інструменти, так як vi, GCC або make.

Одним із завдань ІСР є зменшення часу, необхідного на конфігурацію різноманітних інструментів розробки, натомість пропонуючи той самий набір, як єдине ціле. Це може збільшити продуктивність розробника, у випадку, коли навчання тому, як працює інтегроване середовище розробки є швидшим, ніж освоєння всіх інструментів зокрема. Крім того, більша інтеграція між вбудованими інструментами потенційно може сприяти додатковому збільшенню продуктивності. Наприклад, синтаксичний аналіз коду може відбуватися безпосередньо під час його редагування, тим самим виявляючи помилки ще до трансляції коду.

Деякі інтегровані середовища розробки призначені для використання певної мови програмування (або декількох споріднених мов), надаючи набір можливостей, які більш підходять до парадигми програмування відповідної мови. Такими ІСР є, наприклад PhpStorm, Xcode, Xojo та Delphi.

З іншої сторони, існує чимало більш універсальних ІСР, які є багатомовними, наприклад Eclipse, ActiveState Komodo, IntelliJ IDEA, MyEclipse, Oracle JDeveloper, NetBeans, Codenvy and Microsoft Visual Studio.

Переважна більшість нинішніх ІСР мають графічний інтерфейс

користувача. До появи систем, які підтримують вікна, таких як Microsoft Windows та X Window System (X11), широко використовувалися консольні інтегровані середовища розробки, такі як Turbo Pascal. Особливою їх прикметою було широке використання функціональних клавіш та поєднань клавіш для запуску команд або макросів, які часто використовувалися.

При візуальному програмуванні без використання інтегрованого середовища розробки неможливо обійтися взагалі. Візуальні ІСР дозволяють користувачам створювати ПЗ шляхом будування діаграм з програмних блоків. Потім ці діаграми компілюються або інтерпретуються. Часто, базою для таких діаграм є Unified Modeling Language.

1.1.4. Застосування програмного інтерфейсу доступу до баз даних

ODBC (англ. Open DataBase Connectivity) - це відкритий прикладний програмний інтерфейс доступу до баз даних [7], розроблений консорціумом X/Open.

На початку 1990 років існувало декілька постачальників баз даних, кожен з яких мав власний інтерфейс. Якщо застосунку було необхідно підключатися до кількох джерел даних, для взаємодії з кожною з баз даних був необхідний нестандартний код. Для вирішення цієї проблеми Microsoft та ряд інших компаній створили стандартний інтерфейс для отримання і відправки даних джерелам даних різних типів. Цей інтерфейс отримав назву open database connectivity (відкритий зв'язок з базами даних).

За допомогою ODBC програмісти могли розробляти застосунки для використання одного інтерфейсу доступу до даних, не турбуючись про тонкощі взаємодії з кількома джерелами.

MFC удосконалила ODBC для розробників застосунків. Дійсний інтерфейс ODBC є звичайним функціональним API. Замість створення простої оболонки функціонального API, розробники MFC створили набір абстрактних класів, що представляють логічні сутності в базі даних.

При застосуванні ODBC потрібно пам'ятати, що ця технологія доступу до даних не розрахована на роботу з великою кількістю клієнтів. Якщо необхідно забезпечити одночасну роботу з БД багатьох активних клієнтів, то слід використовувати SQL API або спеціальний інтерфейс для взаємодії з конкретною БД.

1.1.5. Огляд існуючих рішень для роботи з БД

При аналізі предметної області були розглянуті існуючи аналоги - це середовища для роботи з базами даних dbForge for MySQL та MySQL Workbench.

dbForge Studio for MySQL - універсальне рішення для розробки, адміністрування та управління базами даних MySQL і MariaDB. Даний продукт дозволяє створювати і виконувати запити, розробляти і налагоджувати процедури і функції, а також автоматизувати управління об'єктами баз даних MySQL за допомогою зручного для користувача інтерфейсу. dbForge Studio також містить інструменти для порівняння, синхронізації, створення резервних копій баз даних за графіком, а також для аналізу та створення звітів за даними таблиць MySQL.

Середовище dbForge for MySQL має такі функції:

– створення бази даних з обраними параметрами;
– відлагуджування запити до СУБД;
– робота з декількома підключеннями одночасно;
– редагувати зв’язки таблиць у вигляді діаграм;
– сворення таблиць, тригерів, процедур та функцій;
– редагування прав користувачів;
– редагування наповненя таблиць;
– створення звітів та аналіз даних;
– швидку підтримку користувача [13].

Програма призначена для розробників та адміністраторів.

Розповсюджується на платній основі, але є тестовий режим до 30 днів безкоштовного користування.

Офіційним аналогом є Workbench, що поставляються разом з СУБД MySQL.

MySQL Workbench - інструмент для візуального проектування баз даних, що інтегрує проектування, моделювання, створення й експлуатацію БД в єдине безкоштовне оточення для системи баз даних MySQL. Є наступником DBDesigner 4 з FabForce.

MySQL Workbench пропонується в двох редакціях:

– Community Edition - поширюється під вільною ліцензією GNU GPL;

– Standard Edition - доступна за щорічною передплатою. Ця версія включає в себе додаткові функції, які підвищують продуктивність розробників та адміністраторів БД.[4]

MySQL Workbench має інтерфейсну реалізацію всіх можливостей СУБД MySQL та оновлюється одночасно з виходом нової версії СУБД MySQL. Має такі ключові можливості:

– гнучке налаштування СУБД та всієї структури;

– сумісність зі старими версіями СУБД MySQL;

– повна інтеграція з СУБД MySQL;

– офіційна підтримка зі сторони розробників СУБД MySQL 14].

Дані аналоги реалізовують майже повний функціонал СУБД MySQL, мають підтримку від розробників, але більше пристосовані для адміністрування та великих проектів. Обидва аналоги мають як безкоштовну версію, так і платну. Безкоштовні версії обмежені функціоналом, що зменшує простір для розробника.

1.2. Призначення розробки та галузь застосування

Інтегровані середовища розробки створені для того, щоб максимізувати продуктивність програміста, надавши йому пов'язані інструменти розробки зі схожими інтерфейсами як одну програму, в якій відбуватиметься весь процес розробки й яка надає необхідні функції для модифікації, компілювання, розгортання та налагодження програмного забезпечення. Протилежним до цього є підхід до розробки ПЗ, під час якого використовуються окремі інструменти, так як vi, GCC або make.

Одним із завдань ICP є зменшення часу, необхідного на конфігурацію різноманітних інструментів розробки, натомість пропонуючи той самий набір, як єдине ціле. Це може збільшити продуктивність розробника, у випадку, коли навчання тому, як працює інтегроване середовище розробки є швидшим, ніж освоєння всіх інструментів зокрема. Крім того, більша інтеграція між вбудованими інструментами потенційно може сприяти додатковому збільшенню продуктивності. Наприклад, синтаксичний аналіз коду може відбуватися безпосередньо під час його редагування, тим самим виявляючи помилки ще до трансляції коду.

Темою кваліфікаційної роботи бакалавра є «Розробка інтегрованого середовище розробки для СУБД MySQL засобами середовища Qt Creator 4.11», яка передбачає розробку програмного модуля, який призначений для полегшення розуміння СУБД та створення зручного середовища для її швидкого створення і налаштування.

Розробка програми даної роботи призначена для налагодження продуктивнішої розробки баз даних та легшого розуміння структури для новачків.

1.3. Підстава для розробки

Підставою для розробки кваліфікаційної роботи бакалавра на тему «Розробка інтегрованого середовище розробки для СУБД MySQL засобами Qt Creator 4.11» є наказ по Національному технічному університету «Дніпровська політехніка» від __.__. 2021р. № ____-_.

1.4. Постановка завдання

Темою кваліфікаційної роботи бакалавра є «Розробка інтегрованого середовище розробки для СУБД MySQL засобами Qt Creator 4.11», яка передбачає розробку програмного модуля, який призначений для полегшення розуміння СУБД та створення зручного середовища для її швидкого створення і налаштування.

Мета кваліфікаційної роботи полягає в наданні користувачам баз даних, які використовують СУБД MySQL, зручного графічного середовища для інтуїтивного розуміння структури, швидкого створення та налаштування бази даних.

Програма повинна підключатися через «ODBC» драйвер до локального або віддаленого серверу СУБД MySQL. Налаштування останнього вдалого підключення необхідно зберігати в файлі, якщо користувач цього бажає.

При обриві зв’язку з сервером, програма повинна надати можливість повторно ввести дані та підключитися до серверу.

Дані, що вводяться для підключення до серверу СУБД:

– ім’я джерела;
– iP-адреса серверу;
– порт;
– ім’я користувача;
– пароль.

Налаштування середовища також зберігаються у файлі «settings.txt».

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Розроблений додаток повинен забезпечувати наступне функціонування:

1. Графічне середовище повинно надавати структуру серверу у вигляді дерева з всіма наявними базами даних та можливість перемикатися між ними. Зміна активної бази даних повинна змінювати відповідно таблиці, до яких будуть надходити запити користувача.
2. При обранні певної таблиці надавати можливість показувати її структуру у окремому вікні.
3. До таблиць виведених з дерева структури створити введення фільтру та сортування по стовпцям.
4. Окремим вікном викликати створення та редагування таблиць, тригерів та процедур.
5. Надати можливість зв’язатися з розробником та надіслати Bug report через інтерфейс програми.

1.5.2. Вимоги до інформаційної безпеки

Під час розробки програмного додатку необхідно забезпечити наступні вимоги до інформаційної безпеки:

1. Налаштування останнього вдалого підключення додатку до БД необхідно зберігати в файлі, якщо користувач цього бажає.
2. При обриві зв’язку з сервером, програма повинна надати можливість повторно ввести дані та підключитися до серверу.
3. Налаштування середовища повинно зберігатися у файлі.

1.5.3. Вимоги до складу та параметрів технічних засобів

Рекомендовані вимоги до апаратного забезпечення сервера:

- 64-розрядний процесор (x64) 1 ГГц;
- оперативна пам’ять 2 Гб;
- 1 Гб вільного дискового простору.

1.5.4. Вимоги до інформаційної та програмної сумісності

Створений додаток повинен працювати на будь-якому персональному комп'ютері зі встановленою операційною системою Windows 7/8/10 і не конфліктувати з іншими, вже встановленими на персональному комп'ютері програмами та мати встановленим версію MySQL 8.0 зі встановленими MySQL connector for Python та ODBC x64.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Метою кваліфікаційної роботи бакалавра «Розробка інтегрованого середовище розробки для СУБД MySQL засобами середовища Qt Creator 4.11», є розробка програмного модуля, який призначений для полегшення розуміння СУБД та створення зручного середовища для її швидкого створення і налаштування.

Розробка програми даної роботи призначена для налагодження продуктивнішої розробки баз даних та легшого розуміння структури для новачків

В ході виконання роботи розроблений програмний додаток, що дозволяє створювати бази даних в СУБД MySQL.

Інтерфейс додатку виконаний у сучасному дизайні та забезпечує реалізацію всіх найвикористовуваніших функцій СУБД.

Основні функції СУБД – створення зручного інтерфейсу для вирішення основних задач використання БД:

– створення та редагування бази даних, її структури;
– вибір, пошук інформації за певними критеріями;
– захист цілісності даних.

Для нереалізованих функцій передбачений інтегрований командний рядок, котрий дасть змогу користувачу надсилати свої запити до бази даних.

2.2. Опис застосованих математичних методів

Математичні методи та операції для виконання даної теми бакалаврської роботи не передбачені.

2.3. Опис використаних технологій та мов програмування

2.3.1. Обґрунтування вибору мови програмування

Для реалізації програмного продукту обрано мову C++ в інтегрованому середовищі Qt Creator. Підставою для обрання C++ стала швидкість та гнучка робота з пам’ятю. Середвище Qt Creator пропонує нативну підтримку бібліотеки Qt та кросплатформену компіляцію.

2.3.1.1. Опис середовища розробки Qt Creator

Qt Creaetor надає розробнику не тільки зручний набір бібліотек класів, а й певну модель розробки додатків, певний каркас та структуру. Дотримання принципів і правил «гарного стилю програмування на C++/Qt» істотно знижує частоту таких важко відловлюваних помилок в додатках, як виток пам'яті (memory leaks), необроблені виключення, незакриті файли або незвільнені дескриптори ресурсних об'єктів, чим нерідко страждають програми, написані «на голому» C++ без використання бібліотеки Qt.

Важливою перевагою Qt є добре продуманий, логічний і стрункий набір класів, що надає програмісту дуже високий рівень абстракції. Завдяки цьому, програмістам, які використовують Qt, доводиться писати значно менше коду, ніж це має місце при використанні, наприклад, бібліотеки класів MFC. Сам же код виглядає більш струнким і простішим, логічнішим і зрозумілішим, ніж аналогічний за функціональністю код MFC. Його легше підтримувати, розвивати та розширювати.

Крім того, навіть якщо програмісту в даний момент не потрібна кросплатформність для його конкретного додатка (наприклад, планується версія тільки для Windows), ніхто не може знати, що знадобиться завтра. Плани можуть змінитися, і може виявитися необхідним, і вигідним випустити версію для іншої операційної системи або іншої апаратної платформи. У разі використання Qt для цього знадобиться всього лише перекомпіляція вихідного

коду. У разі ж використання, наприклад, MFC або «рідних» системних API знадобиться багато важкої роботи по портуванню, адаптації та налагодженню, а то і переписуванню з нуля існуючого вихідного коду для іншої ОС або апаратної платформи[15].

2.3.1.2. Опис мови програмування C++

C++ не лише займає «призові» місця в рейтингах популярності мов програмування (с самого початку його існування), а також була зразком для створення таких мов програмування як Java та C#.

Підтримує такі парадигми програмування, як процедурне програмування, об'єктно-орієнтоване програмування, узагальнене програмування. Мова має багату стандартну бібліотеку, яка включає в себе контейнери і алгоритми, введення-виведення, регулярні вирази, підтримку багатопоточності і інші можливості. C++ поєднує властивості як високорівневих, так і низькорівневих мов. У порівнянні з його попередником - мовою C, найбільшу увагу приділено підтримці об'єктно-орієнтованого і узагальненого програмування.

У книзі «Дизайн і еволюція C++» Бйорн Страуструп описує принципи, яких він дотримувався при проєктуванні C++. Ці принципи пояснюють, чому C++ саме такий, яким він є. Деякі з них:

– отримати універсальну мову зі статичними типами даних, ефективністю і переносимістю мови C;

– безпосередньо і всебічно підтримувати безліч стилів програмування, в тому числі процедурне програмування, абстракцію даних, об'єктно-орієнтоване програмування та узагальнене програмування;

– дати програмісту свободу вибору, навіть якщо це дасть йому можливість обирати неправильно;

– максимально зберегти сумісність з C, тим самим роблячи можливим легкий перехід від програмування на C;

– уникнути різночитань між C і C++, тобто будь-яка конструкція,

допустима в обох мовах, повинна в кожному з них означати одне і теж, та викновути однакові дії;

– уникати особливостей, які залежать від платформи або не є універсальними;

– «Не платити за те, що не використовується» – модулі, що використовуються не повинні призводити до зниження продуктивності програми.

Нові стандарти виходять для C++ кожні три роки. Вони привносять мові стабільності та додатковий функціонал. Наразі останнім стандартом є C++17, в проєкті використовується стандарт C++11 [16].

2.3.2. Опис СУБД MySQL

Було обрано СУБД MySQL, так як вона поширюється безкоштовно, створена для малих та середніх додатків і має конектори з C++.

MySQL - вільна реляційна система управління базами даних. Розробка та підтримка сайта MySQL здійснює корпорація Oracle. Гнучкість СУБД MySQL забезпечується підтримкою великої кількості типів таблиць: користувачі можуть обирати як таблиці типу MyISAM, що підтримують повнотекстовий пошук, так і таблиці InnoDB, що підтримують транзакції на рівні окремих записів. Більш того, СУБД MySQL поставляється із спеціальним типом таблиць EXAMPLE, що демонструє принципи створення нових типів таблиць. Завдяки відкритій архітектурі і GPL-ліцензуванню, в СУБД MySQL постійно з'являються нові типи таблиць.

MySQL має API та конектори для мов Delphi, C, C ++, Ейфель, Java, lisp, Perl, PHP, Python, Ruby, Smalltalk, Компонентний Паскаль і Tcl, бібліотеки для мов платформи .NET, а також забезпечує підтримку для ODBC за допомогою ODBC-драйвера MyODBC.

MySQL є реляційною базою даних, яка широко використовується. Дані та зв'язки між даними організовані за допомогою таблиць. Кожен стовпець у

таблиці має ім'я і тип. Кожен рядок представляє окремий запис або елемент даних в таблиці, який містить значення для кожного з стовпців.

MySQL заснований на моделі клієнт-сервер. Ядром MySQL є сервер MySQL, який обробляє всі інструкції бази даних (або команди). Сервер MySQL доступний у вигляді окремої програми для використання в мережевому середовищі клієнт-сервер і у вигляді бібліотеки, яка може бути вбудована (або пов'язана) в окремі додатки. MySQL працює разом з декількома службовими програмами, які підтримують адміністрування баз даних MySQL. Команди відправляються MySQL через клієнт MySQL, який встановлений на комп'ютері. Він також дозволяє зберігати дані і отримувати до них доступ через кілька механізмів зберігання, включаючи InnoDB, CSV і NDB.

В додатку «Електронний підручник з інформатики» обрано механізм збереження InnoDB. Це одна з обраних підсистем низького рівня в СУБД MySQL, входить в усі стандартні збірки для різних операційних систем. Основною відмінністю InnoDB від інших підсистем низького рівня MySQL є наявність механізму транзакцій і зовнішніх ключів.

MySQL також здатна реплікувати дані і таблиці секціонування для підвищення продуктивності і довговічності. З метою безпеки MySQL використовує систему прав доступу і зашифрованих паролів, яка забезпечує перевірку на основі хоста. Клієнти MySQL можуть підключатися до MySQL, використовуючи кілька протоколів, включаючи сокети TCP/IP на будь-якій платформі.

MyODBC є драйвер ODBC (2.50) рівня 0 (з деякими можливостями рівнів 1 і 2) для під'єднання сумісного з ODBC додатком до MySQL. MyODBC працює на всіх системах Microsoft Windows і на більшості платформ Unix[17].

2.4. Опис структури системи та алгоритмів її функціонування

2.4.1. Опис структури програми

Програма складається з 9 форм, переходи між ними відбуваються по діаграмі, наведеній на рис. 2.1.

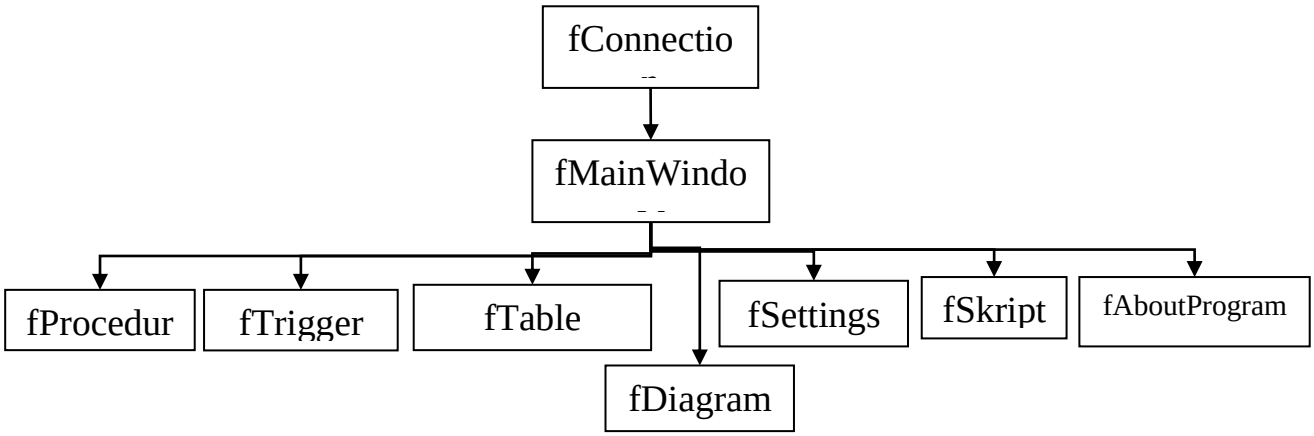

Рис. 2.1. Схема взаємодії складових частин програми

Використовуються стандартні візуальні компоненти бібліотеки Qt.

В програмі використовуються наступні компоненти (рис. 2.2 - 2.6):

– QTreeView - відображає певну модель даних у вигляді дерева. Використовується для відображення структури бази даних та файлової системи при роботі зі скриптами;

– QPlainTextEdit - призначений для вводу та виводу великих обсягів тексту. Використується у вікнах для вводу коду запиту та виведенню великих обсягів інформації;

– QLineEdit - поле для вводу розміром в один рядок. Використовується для вводу імен та інших коротких за обсягом даних. QMenu - контекстне меню. Використовується для створення контекстного меню на різних компонентах;

– QPushButton - кнопка. Використовується для запуску певних процесів програми;

– QInputDialog - діалогове вікно для введення даних. Використовується для введення ім’я при створені бази даних та скрипту;

– QMessageBox - діалогове вікно для показу повідомлень. Використовується для показу помилок та іншої інформації;

– QFileDialog - діалогові вікна для роботи з файлами. Використовується для збереження та відкриття файлів;

– QComboBox - випадаючий список. Використовується для обрання певного пункт зі списку;

– QCheckBox - прапорець. Використовується для управління параметрами.

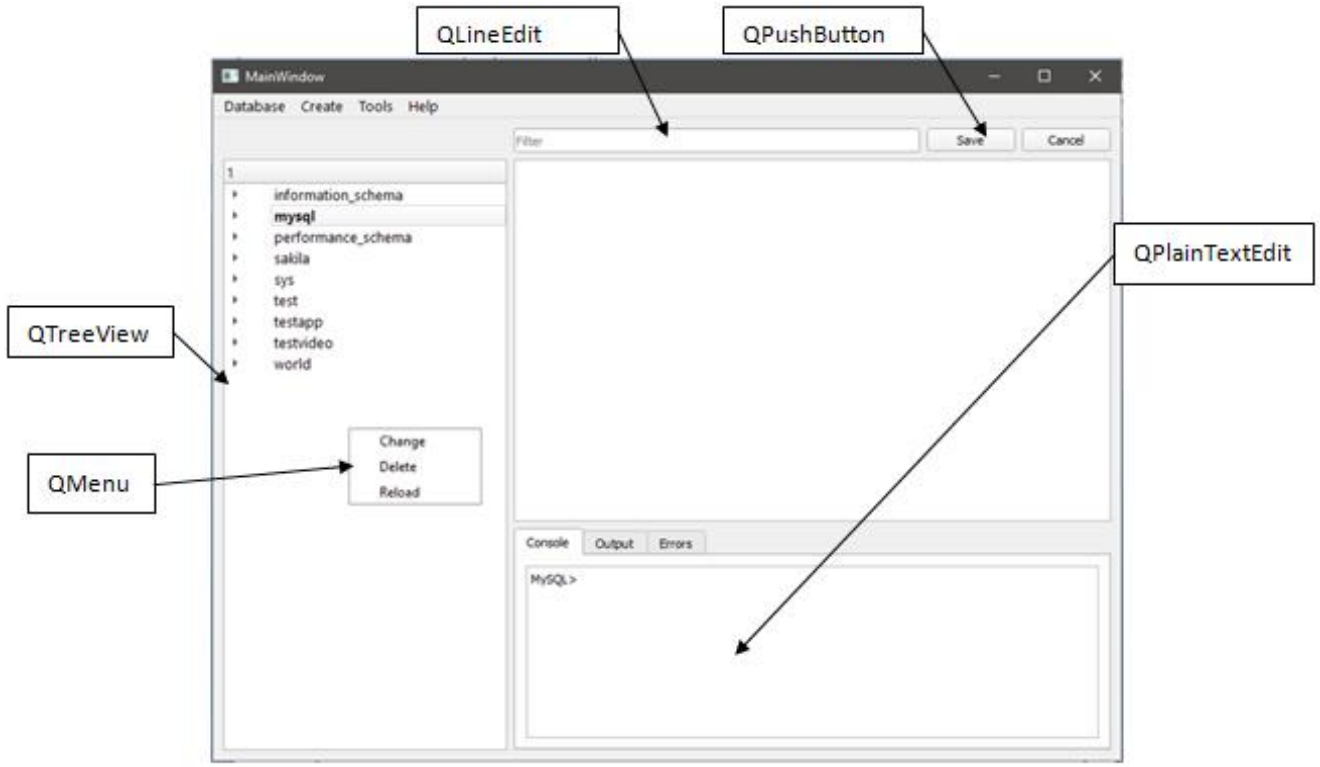

Рис. 2.2. Компоненти

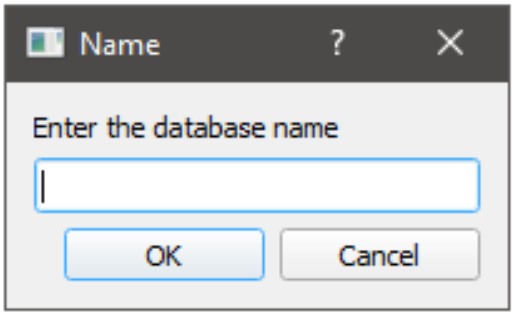

Рис. 2.3. Компонент QInputDialog

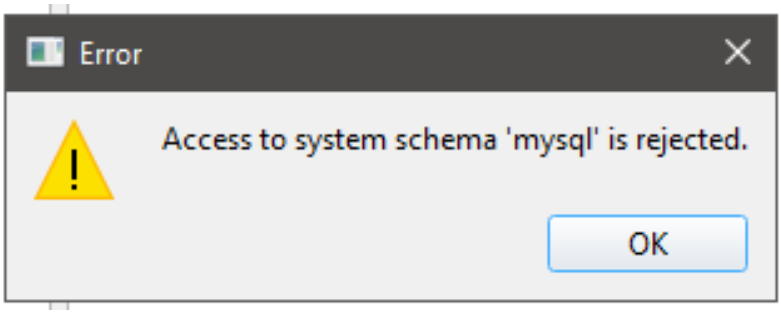

Рис. 2.4. Компонент QMessageBox

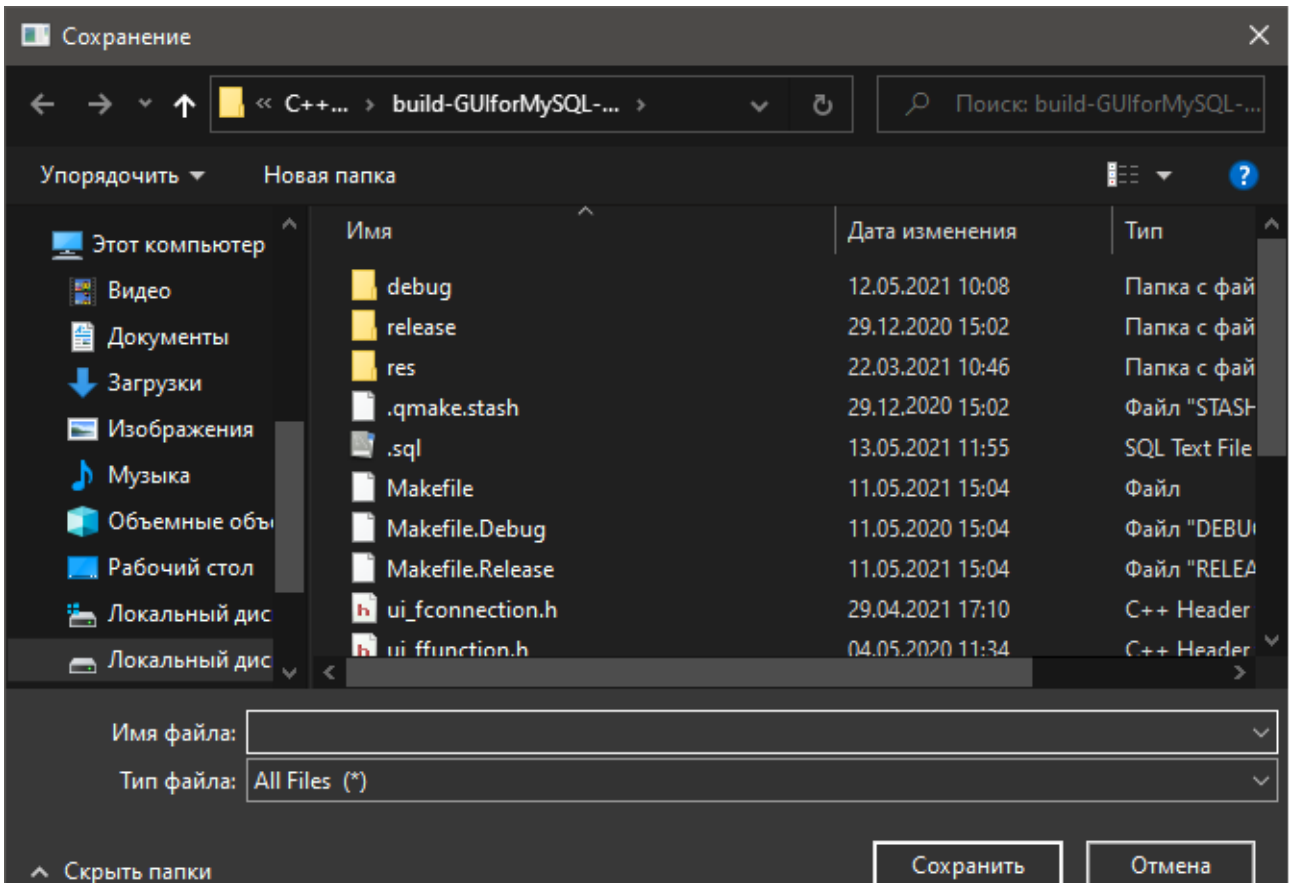

Рис. 2.5. Компонент QFileDialog

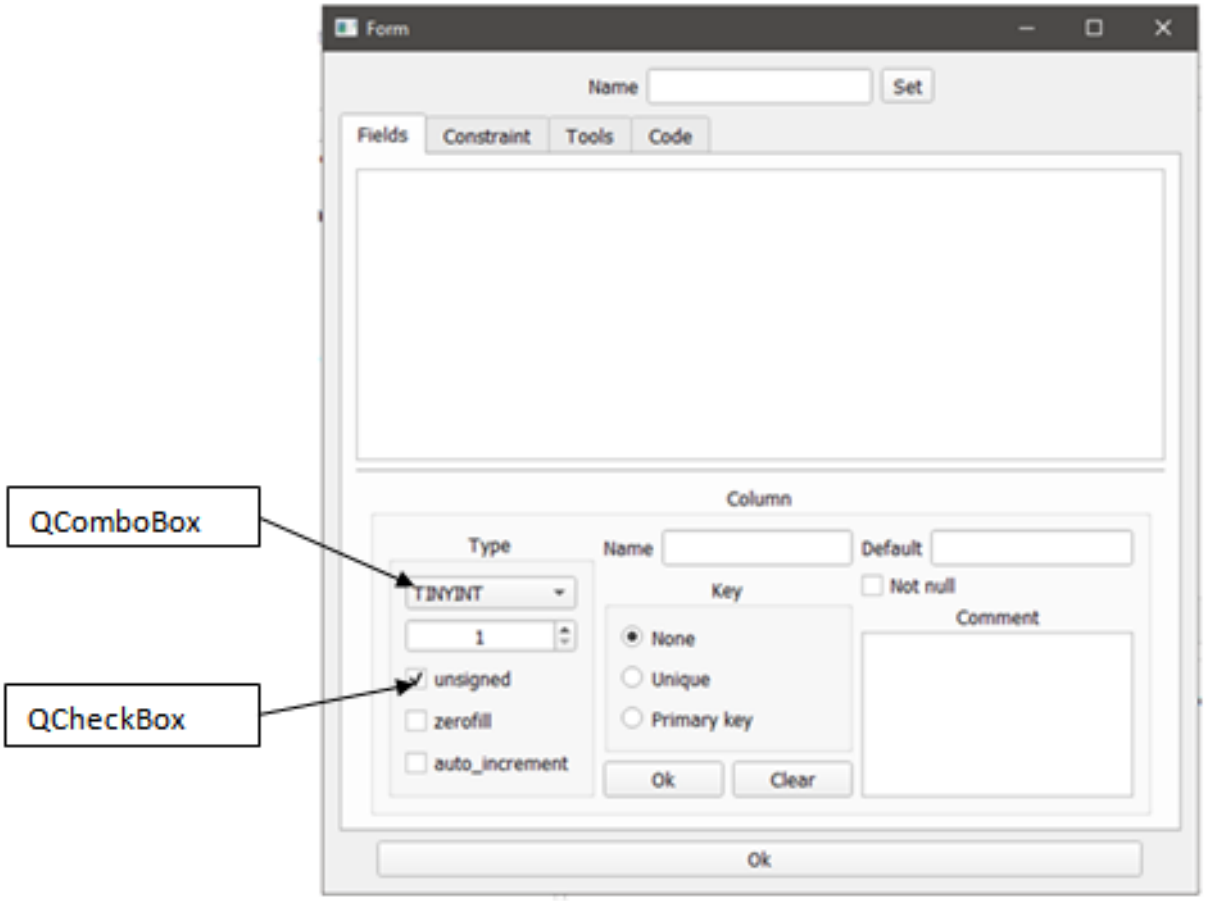

Рис. 2.6. Компоненти

Для реалізації таких функцій як: командний рядок і автодоповнювач, створено власні візуальні компоненти. Вони успадковані від базових в Qt. До них додано новий функціонал, а непотрібний був видалений.

Форма fConnection призначена для підключення до джерела даних. Вона містить поля для введення та методи підключення. Форма представлена на рис. 2.7 та містить наступні компоненти:

– LineEdit призначенні для вводу інформації о підключені;

– wConnect призначеня для підтвердження введеної інформації і здійснення підключення до джерела даних;

– wTools призначений для швидкого переходу до адміністративної панелі операційної системи;

– wSave призначений для зберігання введених даних;

– wHelp призначений для виводу допомоги.

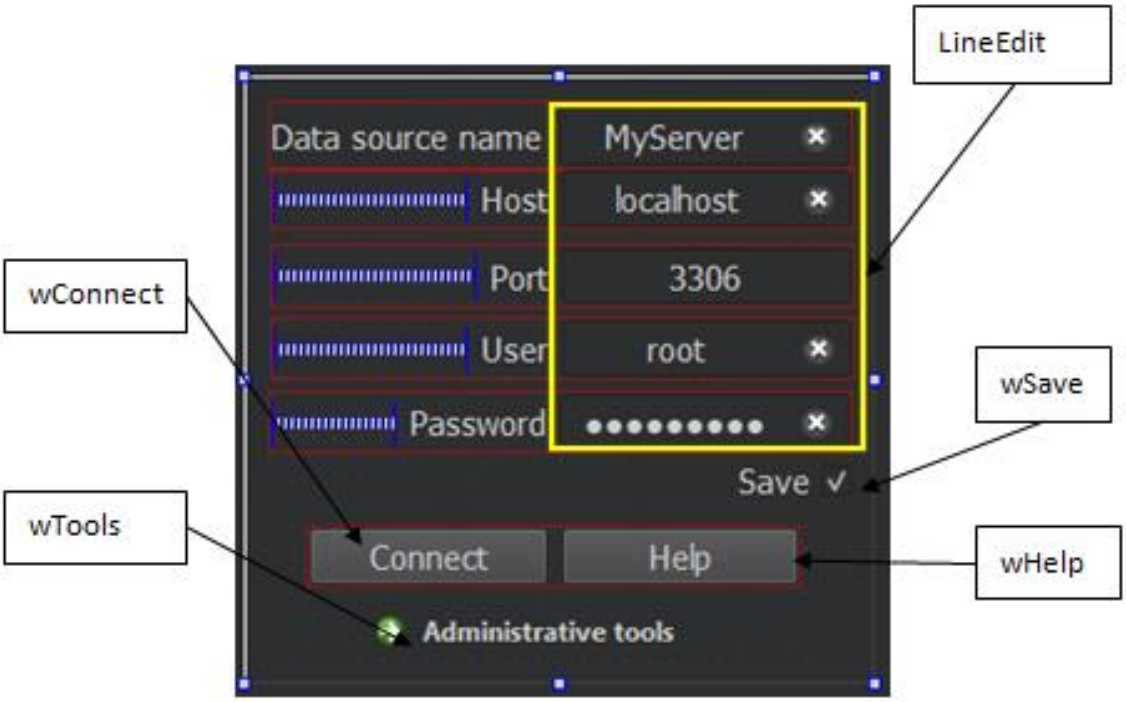

Рис. 2.7. Форма fConnection

Форма fDiagram призначена для виведення діаграми. На ній розміщено поле для виведення зображення діаграми та є можливість зберегти її. Форма представлена на рис. 2.8 та містить наступні компоненти:.

– wSave призначений для зберігання діаграми;
– wArea призначений для виводу діаграми;

Рисунок 2.8 – Форма fDiagram

Форма fProcedure призначена для створення та редагування процедур. На ній розміщено поле для виведення зображення діаграми та є можливість зберегти її. Форма представлена на рис. 2.9 та містить наступні компоненти:

– wHelp призначений для виводу допомоги;
– wDelete призначений для видалення параметру;
– wParameters призначений для введення параметрів процедури;
– wName призначений для введення ім’я процедури;
– wComment призначений для введення коментарю до процедури;
– wEdit призначений для введення коду;
– wExecute призначений для виконування процедури.

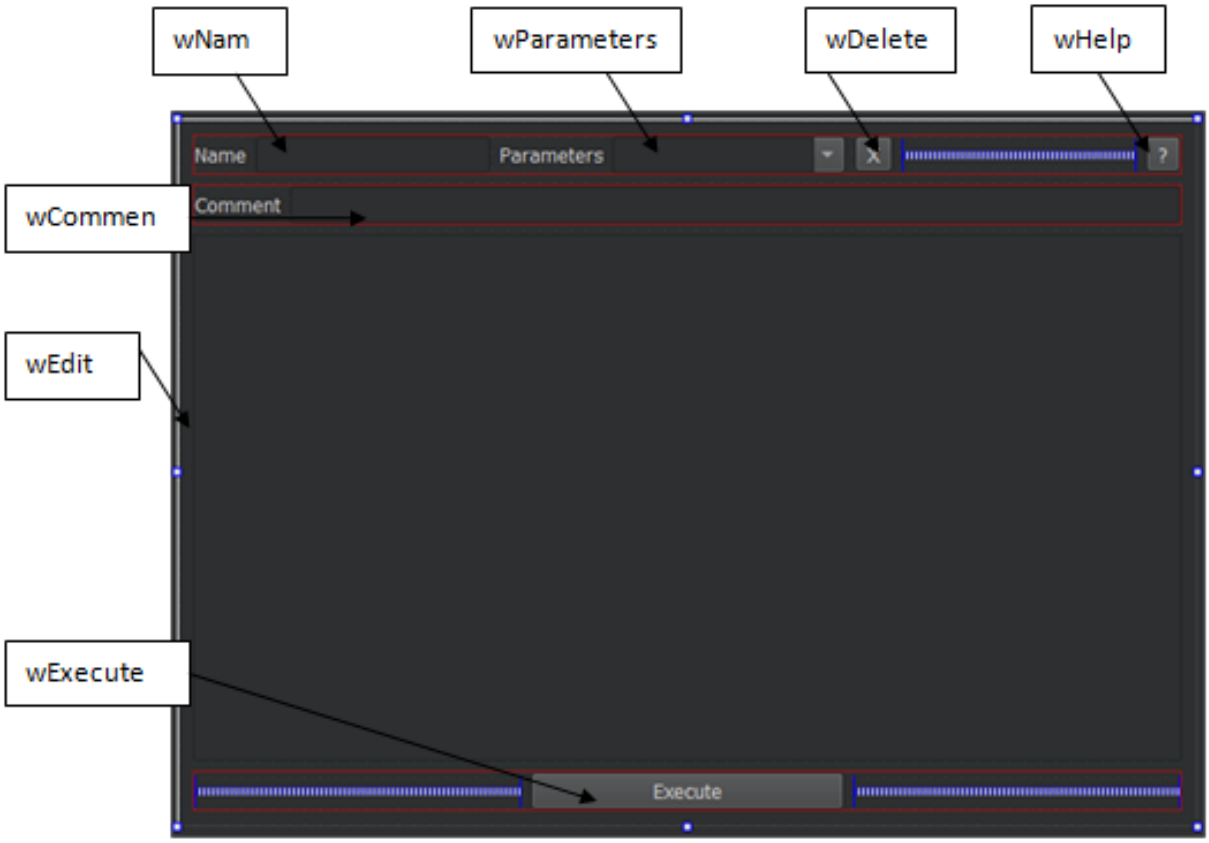

Рис. 2.9. Форма fProcedure

Форма fMainwindow головна форма проєкту, на ній здійснюється перегляд структури та виклики інших форм. Форма представлена на рис.2.10 та містить наступні компоненти:

– wFilter призначений для вводу фільтру до таблиць;

– wSave призначений для збереження змін в таблиці;

– wCancel призначений для відміни останніх змін в таблиці;

– wMenu призначений для створення головного меню програми;

– wButton призначені для дублювання меню Create;

– wTree призначений для виведення структури джерела даних;

– wTable призначений для виведення таблиці з бази даних;

– wTab призначений для розміщення в ньому командного рядка, виводу та помилок.

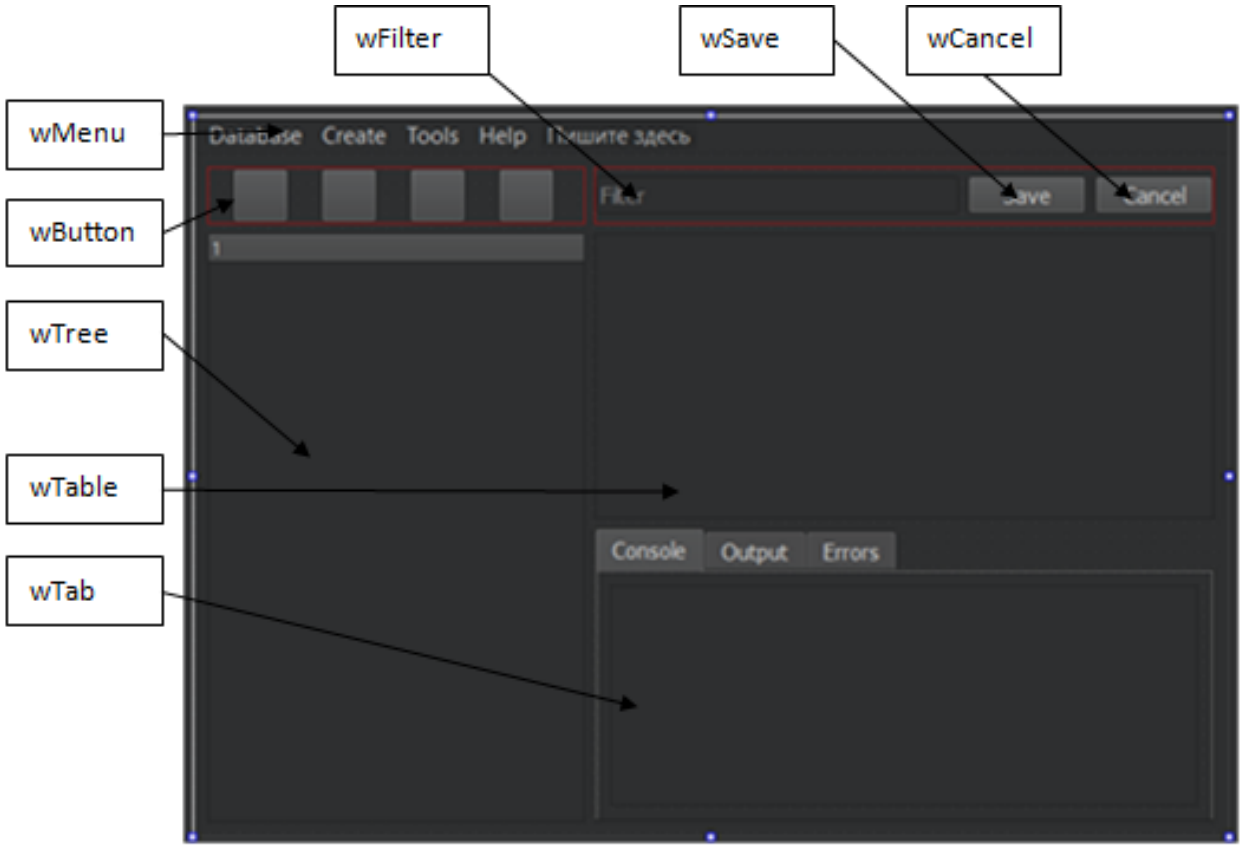

Рис. 2.10. Форма fMainwindow

Форма fScripts призначена для роботи зі скриптами. Форма представлена на рис. 2.11 та містить наступні компоненти:

– wFolder призначений обрання директорії зі скриптами;

– wNew призначений для додавання нового скрипту чи директорії;

– wDelete призначений для видалення скрипту чи директорії;
– wEdit призначений для редагування тексту скрипту;
– wExecute призначений для виконання скрипту;
– wTree призначений для виведення структури директорії;
– wSave призначений для збереження змін в тексті скрипту;

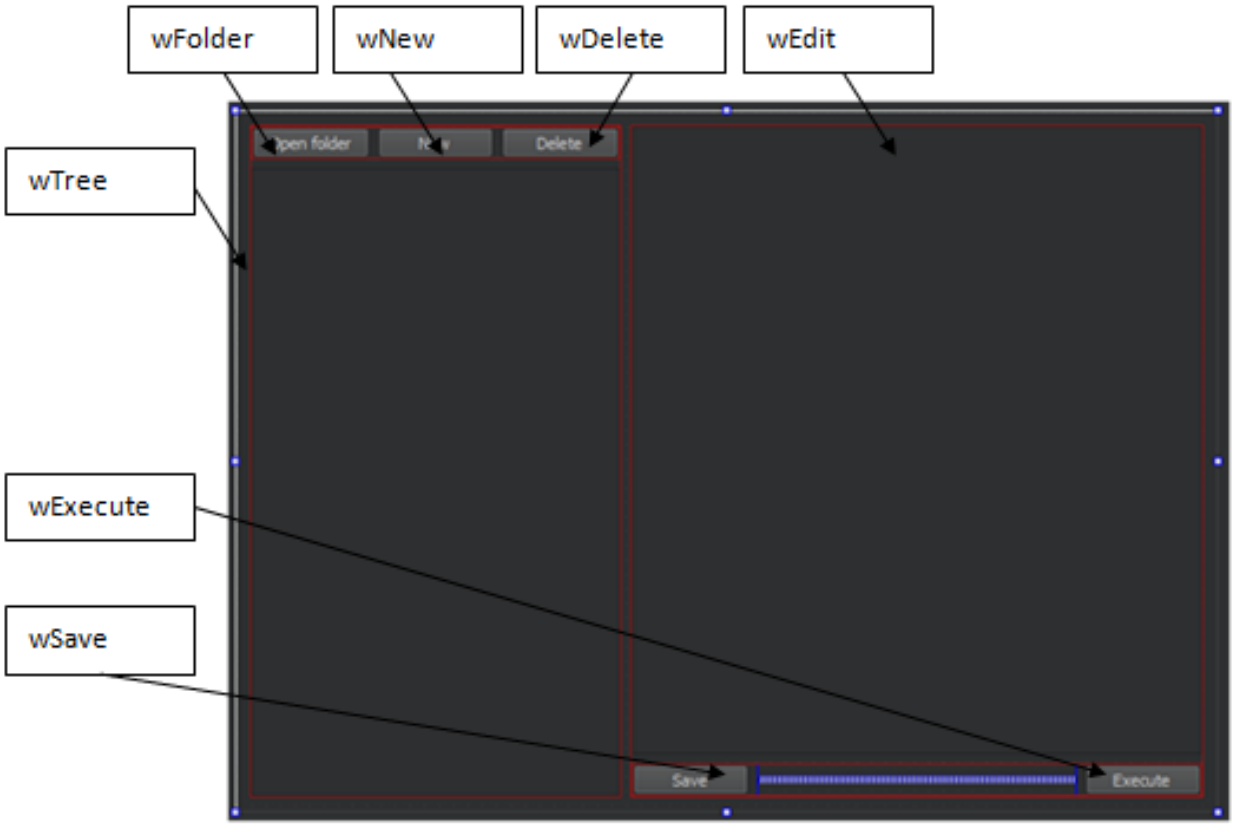

Рис. 2.11. Форма fScripts

Форма fTable призначена для створення таблиць. Всі компоненти розбиті на чотири вкладки . Форма представлена на рис. 2.12 – 2.15 та містить наступні компоненти:

– wName призначений для введення ім’я таблиці;
– wSet призначений для встановлення введеного ім’я;
– wTable призначений для показу структури таблиці;
– wColumnName призначений для введення ім’я стовпцю;
– wType призначений для обрання типу поля;

– wSettings призначений для визначення властивостей поля;
– wKey призначений для визначення типу ключа для поля;
– wDefault призначений для введення значення за замовчуванням;
– wNotnull призначений для встановлення властивості Not Null;
– wComment призначений для додавання коментарю;
– wOK призначений для додавання стовпця в таблицю;
– wClear призначений для очищення полів;
– wConstraint призначений для відображення обмежень;
– wCheck призначений для введеня коду обмеження Check;
– wAddCheck призначений для додавання обмеження Check;
– wForeign призначений для визначення обмеження зовнішнього ключа;
– wAddKey призначений для додавання зовнішнього ключа;
– wClone призначений для клонування таблиць;
– wEngine призначений для змінення системи зберігання даних;
– wCode призначений для відображення коду таблиці;

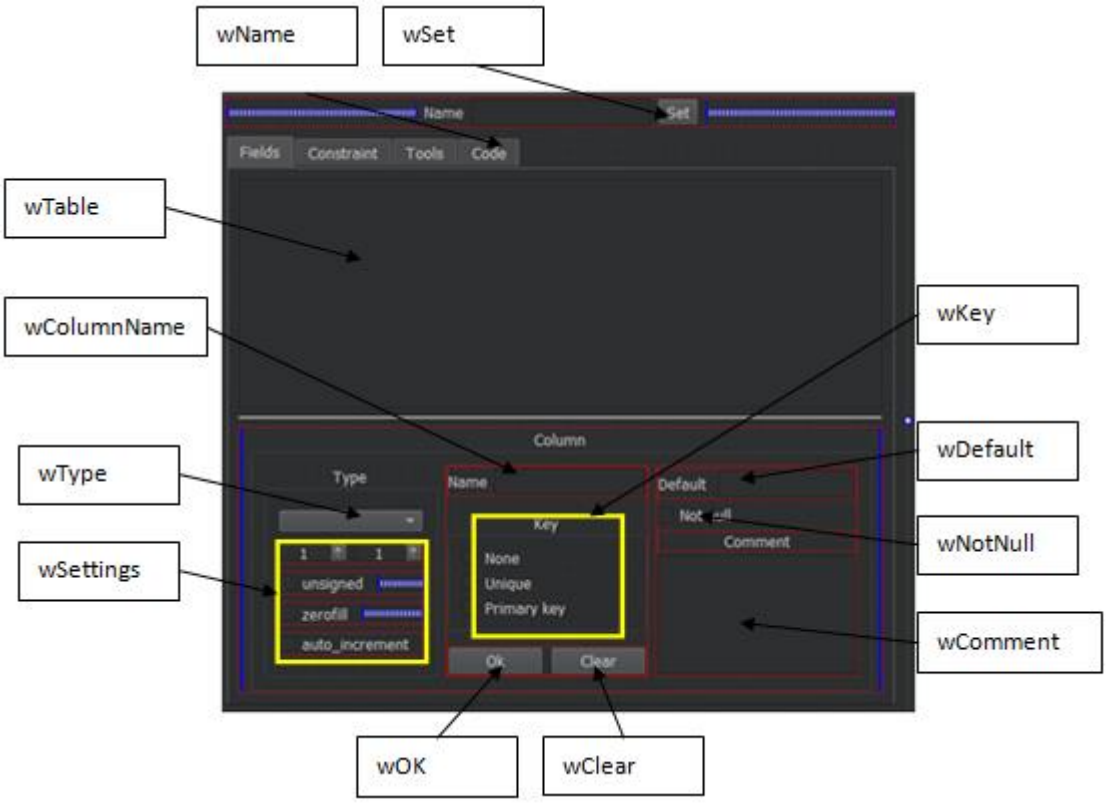

Рис. 2.12. Форма fTable, перша вкладка

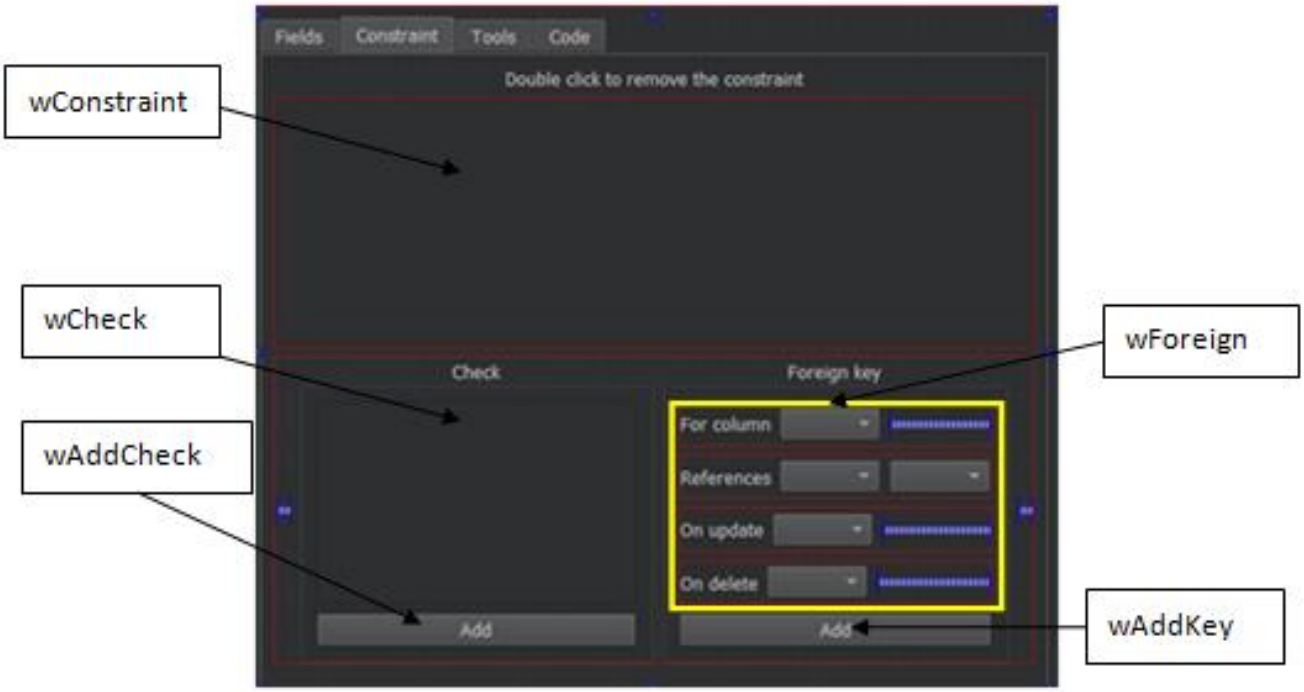

Рис. 2.13. Форма fTable, друга вкладка

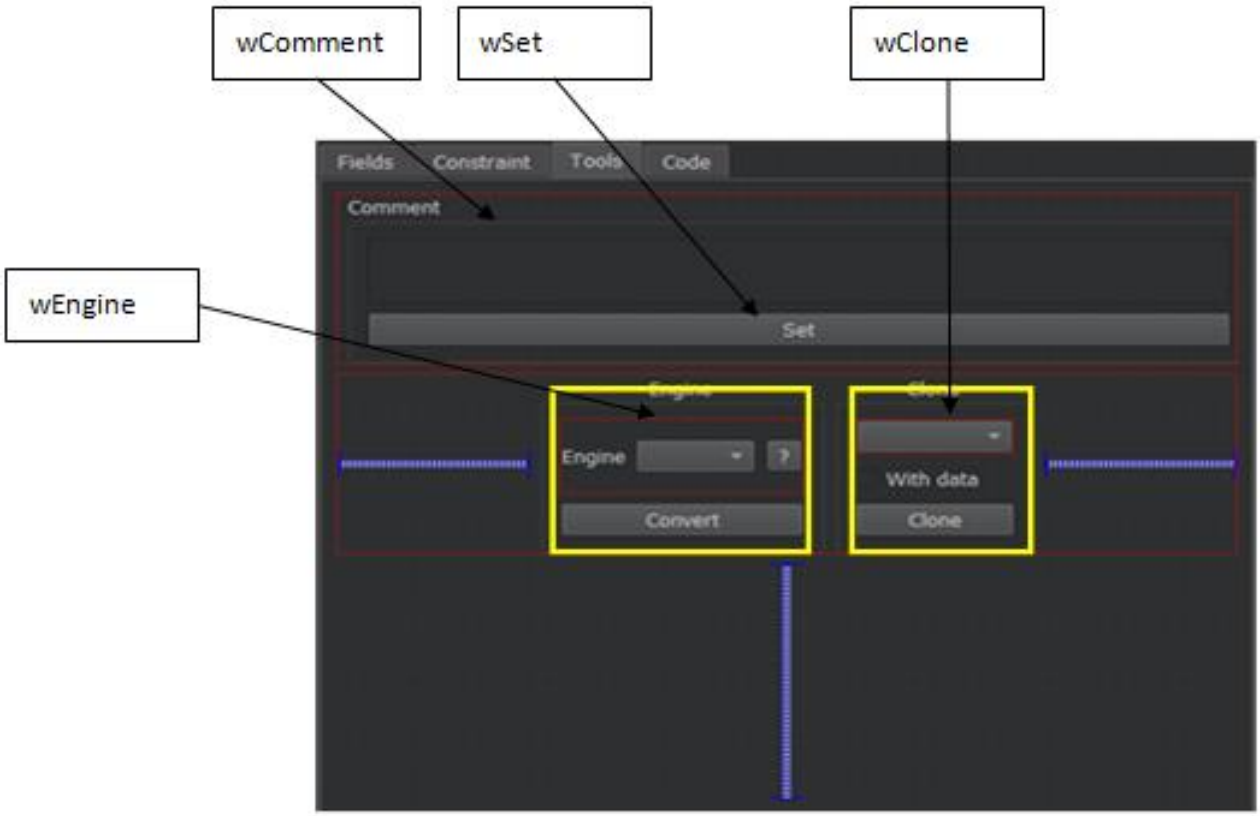

Рис. 2.14. Форма fTable, третя вкладка

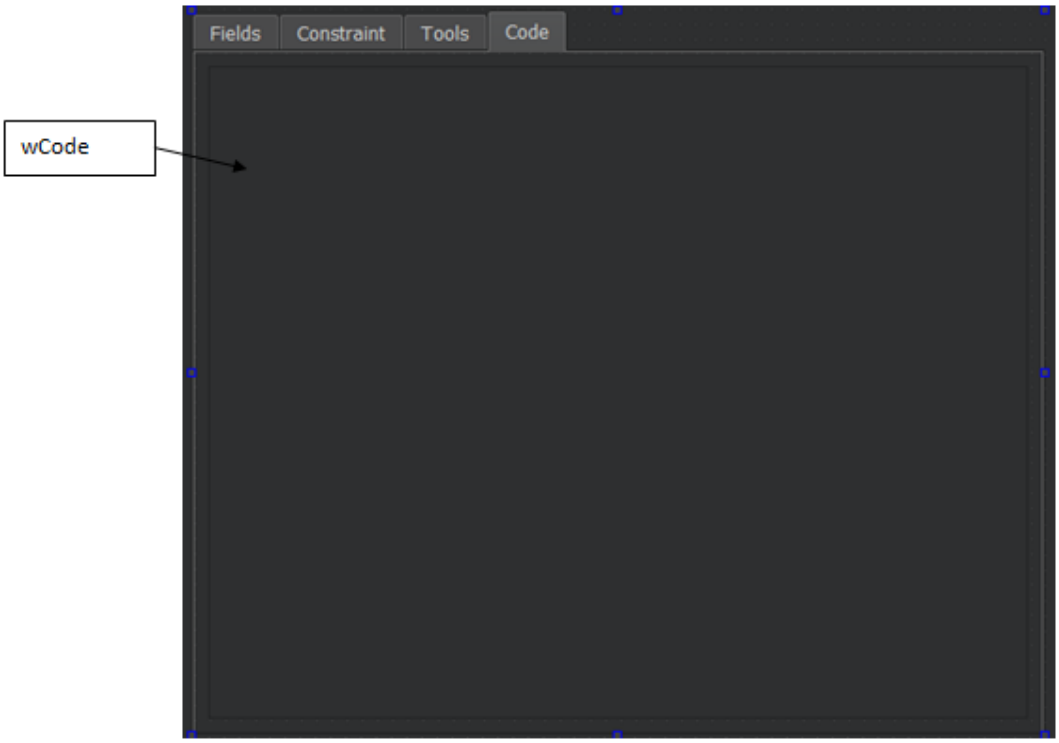

Рис. 2.15. Форма fTable, четверта вкладка

Форма fScripts призначена для роботи з тригерами. Форма представлена на рис. 2.16 та містить наступні компоненти:.

– wName призначений для встановлення ім’я тригеру;
– wTime призначений для обрання часу виконання;
– wEvent призначений для обрання події;
– wTable призначений для обрання таблиці;
– wCode призначений для введення коду тригеру;
– wExecute призначений для виконання тригеру;

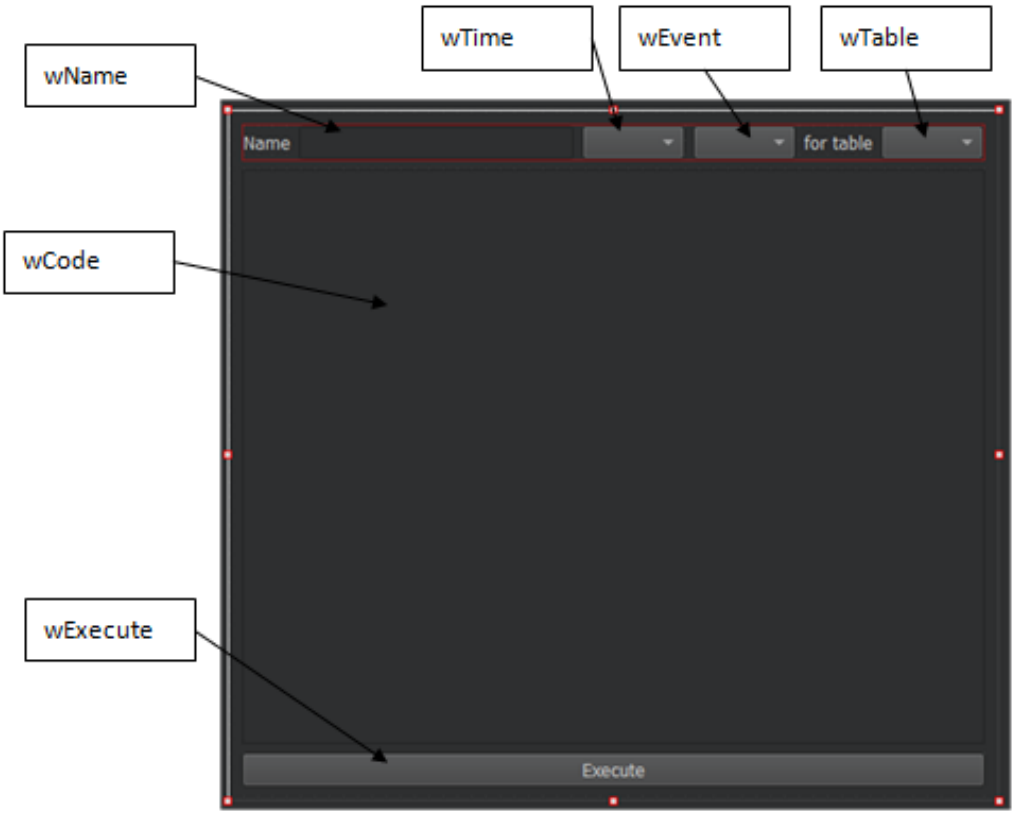

Рис. 2.16. Форма fTrigger

2.4.2. Опис класів програми

Управління базою даних здійснюється через спеціальний клас CDatabaseControl, що створюється в головній формі та встановлюється на інші. Даний клас працює з MySQL, він приймає запити з інших класів та оброблює їх. Виведення повідомлень здійснюється через головну форму діалоговим вікном.

Клас AutoHelper відповідальний за автодоповнювач коду. Він створюється в кожній формі окремо. Всі слова зчитуються з файлу, а специфічні слова, такі як таблиці та їх поля, встановлюються під час роботи програми. Для імітації командного рядка створено клас Console, успадкований від QPlainTextEdit. Перевизначено методи подій та додано методи посилання запитів і їх обробки.

Всі класи обмінюються даними за допомогою методу «сигнали і слоти». При необхідності обмінятися даними, клас посилає сигнал з відповідною інформацією, на цей сигнал відповідають методи інших класів.

При відкриті програми головна форма відразу створює всі інші форми та посилає їм сигнал, що містить клас CDatabaseControl, для встановлення класу управління базою даних.

Клас CDatabaseControl для роботи з базою даних містить такі методи:

– tablesList() - повертає список таблиць в базі даних;
– proceduresList() - повертає список процедур в базі даних;
– triggerList() - повертає список тригерів в базі даних;
– execQuery() - виконує запит та повертає модель запиту;
– execQueryModel() - виконує запит та повертає таблицю;
– getCurrentDatabase() - повертає ім’я поточної бази даних;
– connectToDatabase() - здійснює підключення до MySQL;
– changeWorkDatabase - метод призначений для змінення активної бази даних.

Автодоповнювач створюється в кожній формі окремо. Спеціальні слова встановлюються через головну форму. Автодоповнювач представлений класом AutoHelper, він успадкований від візуального компоненту QListWidget. Виклик автодоповнювача відбувається коли в полі для вводу коду знаходиться схоже слово з тими, що є в наявності у класу.

Клас AutoHelper має такі методи:

– setWordTip() - матод зчитує слова з файлу;
– setAddTip() - метод додає додаткові слова до бібліотеки;
– choosedItem() - повертає обране слово;
– showTips() - метод показує список запропонованих слів;
– setFocusOnTips() - переводить фокус на список.

Програма використовує такі дані, як:

– файли ресурсів;
– текстові файли;

– файли з розширенням «.sql».

Файли ресурсів містять іконки та картинки. Вони використовуються в програмі для поліпшення користувацького інтерфейсу, роблячи його наглядніщим та інтуїтивно зрозумілим. Файли ресурсів зберігаються в директорії з програмою.

Текстові файли містять інформацію о користувацьких налаштуваннях. З цих файлів здійснюється зчитування користувацьких налаштувань при запуску програми. Коли користувач змінює налаштування, всі зміни записуються до файлу.

Файли з розширенням «.sql» містять користувацькі скрипти написані на мові SQL. Дані файли запускаються не лише в дипломному проекті, а і засобами самого MySQL. Можуть бути згенеровані як самою програмою, так і іншим чином. Цим файлам не обов’яково знаходитись поряд с програмою, для них реалізовано перегляд файлової системи.

2.4.3. Опис основних алгоритмів програми

Ключові методи роботи з MySQL зосереджені в класі CDatabaseControl. Даний клас виступає в ролі посередника між користувачем і MySQL. Ключовимим методами класу є:

– execQuery() - даний метод спочатку посилає сигнал, що йому надійшов запит, на цей сигнал, що містить текст запиту, відповідає метод головної форми, що записує запит в журнал. Після відправки сигналу, метод відправляє запит до MySQL через об’єкт класу QSqlQuery, якщо запит вдалий повертає True, якщо ні, то посилає сигнал з текстом помилки та повертає False;

– lastQuery() - даний метод повертає модель даних останнього запиту. Використовується коли потрібно пройтись по таблиці результату запиту та отримати деякі дані;

– execQueryModel() - даний метод виконує ту саму функцію, що й метод execQuery(), але відразу повертає результуючу модель даних запиту;

– connectToDatabase() - даний метод використовується для підключення до СУБД MySQL через ODBC драйвер.

Клас головної форми fMainWindow також має ключові методи:

– showError() - даний метод виводить діалогове вікно помилки;

– showMessage() - виводить різні повідомлення в діалоговому вікні.

У вікнах, де виконується введення коду, велике значення має автодоповнювач. Він надає змогу значно полегшити та прискорити написання коду. Ключовим алгоритмом в роботі автодоповнювача є алгоритм пошуку схожих слів, що продемонстровано у лістингу 2.1. При введені коду відокремлюється останнє слово та надсилається до автодоповнювача. Автодоповнювач починає шукати слова в бібліотеці, що починаються схожим чином, і додає їх до списку. Якщо останній символ крапка, то вона видаляється зі слова та йде пошук таблиці з таким ім’ям, якщо таблиця знайдена, то до списку також додаються всі поля таблиці. В кінці алгоритму йде перевірка, якщо список виявився порожнім, то він не буде показуватися користувачу.

Лістинг 2.1. Додавання слів в автодоповнювач коду

```
void AutoHelper::showTips(QString text)
{
   this->clear();
   text = text.toUpper();
   if(!text.isEmpty() && !text.isNull())
   {
      for (QString var : this->_words) {
         if(var.indexOf(text) == 0)
            QListWidgetItem * item = new QListWidgetItem(var, this);
      }
      text = text.toLower();
      for (QString var : _additional.keys()) {
         if(var.indexOf(text) == 0)
```

```
            QListWidgetItem * item = new QListWidgetItem(var, this);
        }
        if(text[text.size() - 1] == '.')
        {
            text = text.mid(0, text.size() - 1);
            if(_additional.keys().indexOf(text) >= 0)
            {
                for(QString var : _additional.value(text))
                    QListWidgetItem * item = new QListWidgetItem(var, this);
            }
        }
        if(this->count() == 0)
            hide();
        else
            show();
    }
    else
        hide();}
```

При надсиланням автодоповнювачем слова, яке потрібно замінити, активізується метод changeCurrentWord(), представлено на лістингу 2.2, в класі Console, який реалізує імітування командного рядку. Даний метод розбиває останній блок тексту на рядки та перевіряє наявність крапки в останньому рядку, якщо крапки немає, то останній рядок видаляється і на його місце вставляється нове слово, якщо крапка є, то нове слово приєднується до останнього рядку. Далі виділяється блок тексту під текстовим курсором та видаляється, а на його місце встановлюється новий блок з заміненим словом.

Лістинг 2.2. Змінення слова

```
void Console::changeCurrentWord(QListWidgetItem * item)
```

```
{
   QString word = item->text();
   QStringList textList = textCursor().block().text().mid(promt.size()).split(" ");
   if(textList.last().back() == ".")
      textList.last() += word;
   else
      textList.last() = word;
   QTextCursor cursor(textCursor().block());
   cursor.select(QTextCursor::BlockUnderCursor);
   cursor.removeSelectedText();
   cursor.insertText(promt);
   cursor.insertText(textList.join(" ") + " ");
   moveCursor(toPlainText().length());}
```

У лістингу 2.3 показано важливий алгоритм для виконання користувацьких скриптів. Коли користувач обрав скрипт для виконання та натиснув на кнопку «Виконати», алгоритм визначає шлях до скрипту та поміщає його в єдиний рядок. Рядок розбивається на список рядків, розділених знаком крапка з комою та пропускаються порожні рядки. Далі кожен рядок в списку починає надсилатись на виконання до MySQL, якщо виникла помилка, то всі зміни відміняються, і користувач бачe діалогове вікно з помилкою.

Лістинг 2.3. Виконання скрипту

```
void fSkripts::on_wExecute_clicked()
{
   QFile file(editPath);
   if (!file.open(QIODevice::ReadOnly | QIODevice::Text))

   {
      QMessageBox::warning(this, "Error", "File not opened");
```

```
        return;
    }
    QTextStream in(&file);
    QString sql = in.readAll();
    QStringList sqlStatements = sql.split(';', QString::SkipEmptyParts);
    foreach(const QString& statement, sqlStatements)
        if (statement.trimmed() != "")
            dbcontrol->execQuery(statement);
QMessageBox::information(this, "Done", "Successful");}
```

На формі fTable відбувається створення таблиць. Важливим методом є посилання коректного запиту на додавання чи змінення поля в таблиці. Метод перевіряє, чи була введено ім’я поля. Потім перевіряє, чи є вже таке поле в таблиці, якщо так, то створює рядок з модифікатором Add, якщо ні, то Modify. Далі до рядку додається, зібране з форми, визначення поля і виконується запит. Якщо було змінено ключ, то окремо від запиту поля надсилається запит на встановлення полю відповідного ключа. Код методу представлено на лістингу 2.4.

Лістинг 2.4. Створення поля в таблиці

```
void fTable::on_wAdd_clicked()
{
    QString columnName = ui->wColumnName->text();
    if(!columnName.isEmpty())
    {
      QString command;
      if(dbcontrol->columnList(this->currentTable).contains(columnName.toLower()))
      {
         command = "ALTER TABLE " + this->currentTable + " MODIFY ";
      }
```

```
    else
    {
        command = "ALTER TABLE " + this->currentTable + " ADD ";
    }
    command += collectCommand();
    dbcontrol->execQuery(command);

    if(ui->wKeyNone->isChecked())
    {
        ;;;
    }
    else if(ui->wKeyUnique->isChecked())
    {
        if(!isHave(ui->wAdvancedEdit->toPlainText().split("\n"), columnName,
"UNIQUE"))
        {
            dbcontrol->execQuery("ALTER TABLE " + this->currentTable + " ADD
UNIQUE (" + columnName + ")");
        }
    }
    else if(ui->wKeyPrimary->isChecked())
    {
        if(!isHave(ui->wAdvancedEdit->toPlainText().split("\n"), columnName,
"PRIMARY"))
        {
            dbcontrol->execQuery("ALTER TABLE " + this->currentTable + " ADD
PRIMARY KEY (" + columnName + ")");
        }
    }
  }
```

```
   else
   {
      qDebug() << "empty";
   }
   showTable();}
```

Коли користувач хоче змінити позицію поля в таблиці, активується функція columnUp() чи columnDown() відповідно до напрямку. Вони є схожими за алгоритмом, різниця лише в напрямі, код функції представлено у лістингу 2.5. Спочатку зчитується ім’я поля з таблиці, далі перевіряється, чи не стоїть поле вже на кінцевій позиції, береться наступне поле після поточного та за допомогою ключового слова after змінюється позиція в таблиці.

Лістинг 2.5. Функція переміщення поля в таблиці

```
void fTable::columnUp()
{
   if(ui->wTable->currentIndex().row() > 0){
      if(dbcontrol->execQuery("show create table " + currentTable)){
         QString columnName = ui->wTable->model()->index(ui->wTable-
>currentIndex().row(), 0).data().toString();
         QStringList definition;
         while (dbcontrol->lastQuery()->next()) {
            definition = dbcontrol->lastQuery()->value(1).toString().split("\n");
         }
         QString columnDefinition;
         for (QString str : definition) {
            if(str.indexOf("`"+ columnName +"`") < 4 && str.indexOf("`"+
columnName +"`") >= 0){
               str.remove(str.size() - 1, 1);
               columnDefinition = str;
```

```
            break;
        }
    }
    if (ui->wTable->currentIndex().row() == 1) {
        dbcontrol->execQuery("ALTER TABLE " + currentTable + " MODIFY
COLUMN "
                             + columnDefinition + " FIRST");
    }
    else {
        dbcontrol->execQuery("ALTER TABLE " + currentTable + " MODIFY
COLUMN "
                             + columnDefinition + " AFTER " + ui->wTable->model()-
>index(ui->wTable->currentIndex().row() - 2, 0).data().toString());
    }
    showTable();}}
```

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними для роботи програми є дані, що вводяться для підключення до серверу СУБД:

– ім’я джерела;
– iP-адреса серверу;
– порт;
– ім’я користувача;
– пароль.

Також до вхідних даних відносять скрипти, таблиці, зв’язки та інші команди, що вводить користувач під час проектування бази даних.

Програма використовує такі дані, як:

– файли ресурсів;

– текстові файли;

– файли з розширенням «.sql».

Файли ресурсів містять іконки та картинки. Вони використовуються в програмі для поліпшення користувацького інтерфейсу, роблячи його наглядніщим та інтуїтивно зрозумілим. Файли ресурсів зберігаються в директорії з програмою.

Текстові файли містять інформацію о користувацьких налаштуваннях. З цих файлів здійснюється зчитування користувацьких налаштувань при запуску програми. Коли користувач змінює налаштування, всі зміни записуються до файлу.

Файли з розширенням «.sql» містять користувацькі скрипти написані на мові SQL. Дані файли запускаються не лише в роботі, а і засобами самого MySQL. Можуть бути згенеровані як самою програмою, так і іншим чином. Цим файлам не обов’яково знаходитись поряд с програмою, для них реалізовано перегляд файлової системи.

2.6. Опис роботи розробленої системи

2.6.1. Використані технічні засоби

Розробка програмного забезпечення велася на ПК з наступними характеристиками:

– 64-розрядний процесор (x64) 1 ГГц;

– оперативна пам’ять 2 Гб;

– 1 Гб вільного дискового простору

– встановлена версія MySQL 8.0 зі встановленими MySQL connector for Python та ODBC x64.

2.6.2. Використані програмні засоби

Для реалізації програмного продукту обрано мову C++ в інтегрованому середовищі Qt Creator. Підставою для обрання C++ стала швидкість та гнучка робота з пам’яттю. Середовище Qt Creator пропонує нативну підтримку бібліотеки Qt та кросплатформену компіляцію.

2.6.3. Виклик та завантаження програми

Для забезпечення повноцінної роботи додатку на ПК необхідна встановлена версія MySQL 8.0 зі встановленими MySQL connector for Python та ODBC x64.

При першому відкриті програми з’явиться вікно, яке представлено на рис. 2.17, де потрібно буде ввести дані для підключення програми до MySQL через ODBC драйвер. Необхідно ввести такі дані як: джерело даних, адрес серверу, порт, ім’я користувача та пароль. Щоб не вводити постійно дані при роботі з одним сервером, є можливість поставити галочку напроти опції Save, таким чином, при наступних запусках програми буде проходити підключення до останнього успішного підключення.

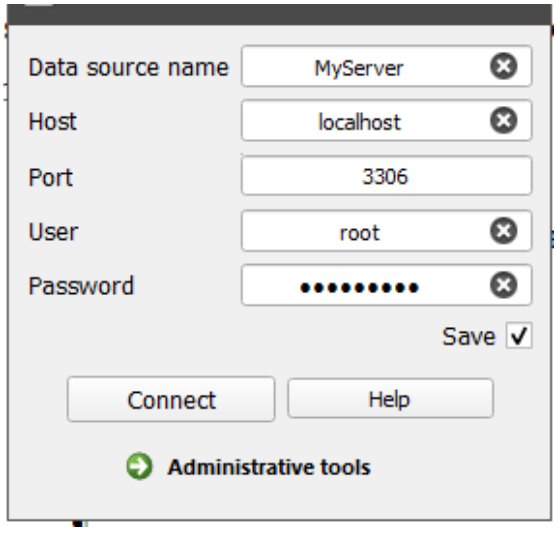

Рис. 2.17. Вікно підключення

Кнопка Help виводить текстову допомогу по створенню джерела даних, показано на рис. 2.18. Швидкий перехід до панелі адміністрування операційної системи відбувається по кнопці Administrative tools. Коли джерело налаштовано, а всі дані введено, потрібно натиснути на Connect.

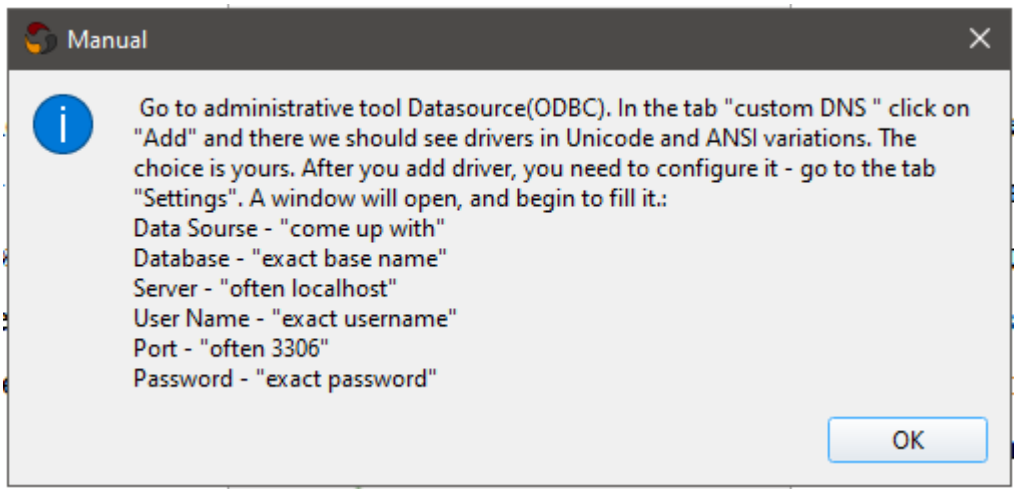

Рис. 2.18. Вікно допомоги

2.6.4. Опис інтерфейсу користувача

Після підключення відкривається головне вікно програми, яке наведено на рис. 2.19. В лівій частині вікна розташоване дерево наповнення, в якому відображено всі доступні бази, з переліком таблиць, тригерів та процедур, які за нею закріплені. При подвійному кліку відбувається зміна активної бази даних, яка позначається жирним шрифтом. При подвійному натисканні на таблицю в правій частині форми відображається вміст таблиці з можливістю її редагування.

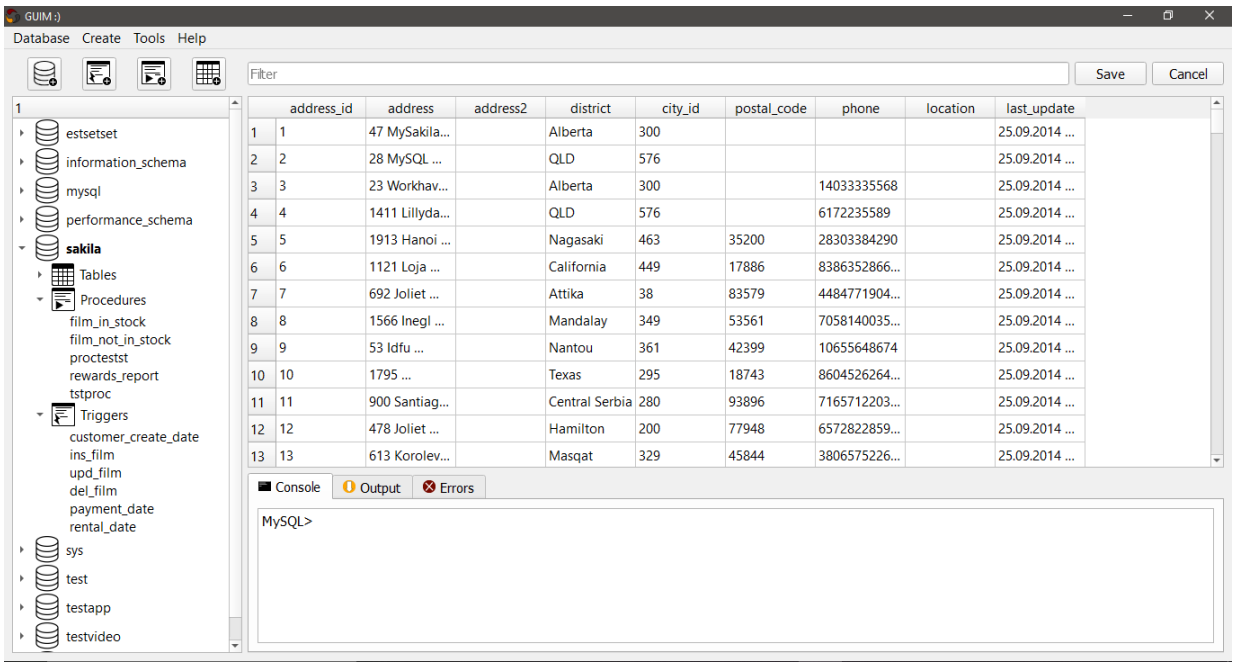

Рис. 2.18. Головне вікно програми

Для відображення даних за певними ознаками передбачена фільтрація даних. Поле для задання фільтру розташовується над таблицею. Під таблицею розміщується область, яка має три вкладки. Перша вкладинка Console, (рис. 2.19) являє собою командний рядок. В цьому полі вводяться користувацькі запити та відправляються до СУБД MySQL. Просте натискання клавіші Enter призводить до відправки запиту на виконання, а з модифікатором Shift створюється новий рядок в командному рядку для зручного написання багатострокових запитів.

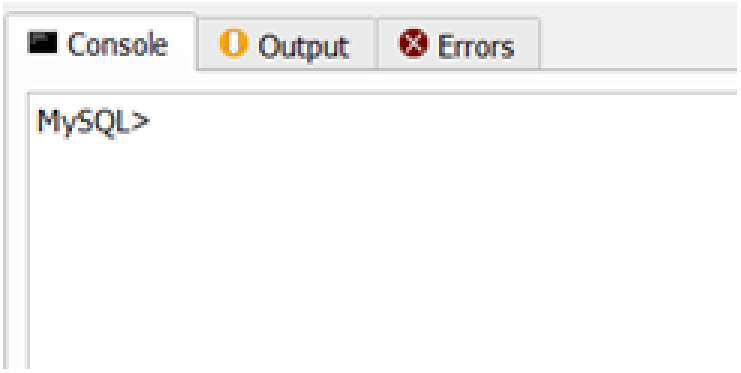

Рис. 2.19. Вкладинка Console

На вкладці Output, яка наведена на рис. 2.20, виводяться всі запити, створені програмою чи користувачем.

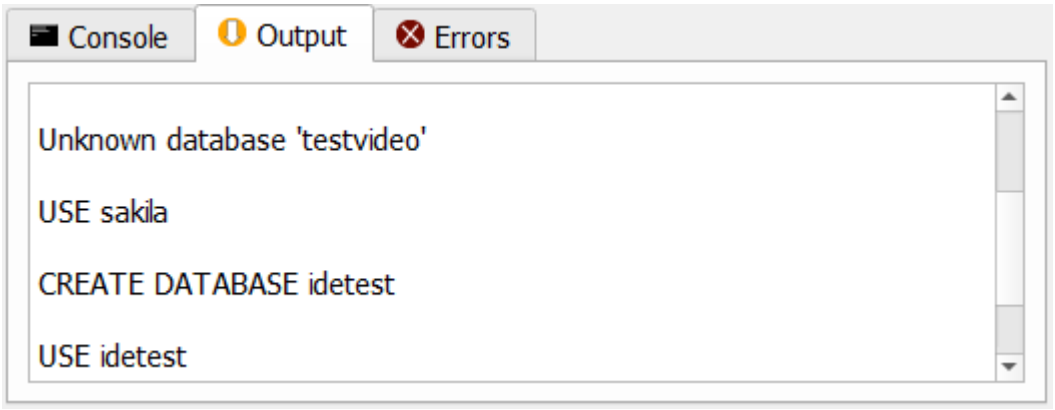

Рис. 2.20. Вкладка Output

На вкладці Error, яка наведена на рис. 2.21, виводяться всі помилки, що виникли під час роботи програми.

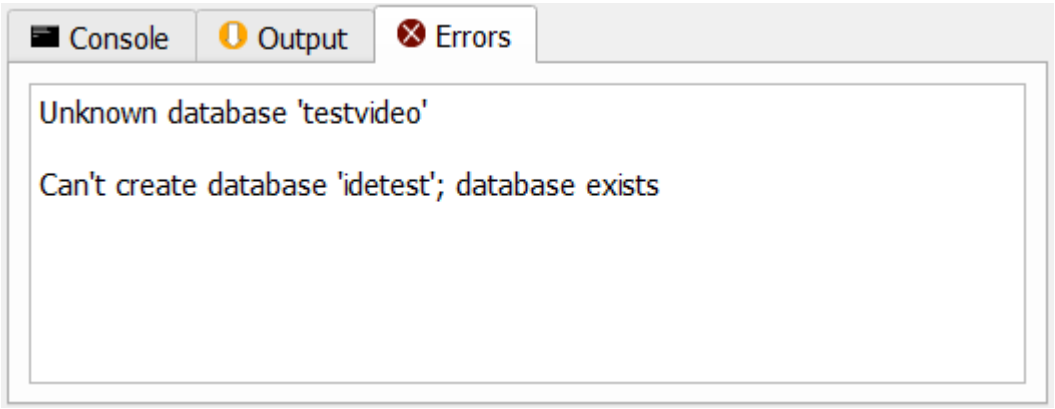

Рис. 2.21. Вкладка Error

На рис. 2.22 показано підпункти меню Database: Connect, Export, Import, Settings та Exit.

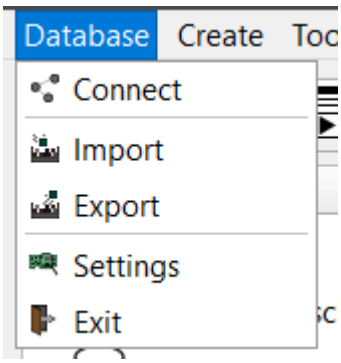

Рис. 2.22. Пункти в підменю Database

Перший підпункт Connect відкриває вікно підключення (рис. 2.17), що дозволяє підключитися до іншого джерела даних. Програма має можливість експортувати та імпортувати бази. Для цього потрібно обрати підпункт Export для експорту, або Import для імпорту. З’явиться ділогове вікно, як представлено на рис. 2.23, щоб обрати місце збереження sql файлу при експорті, або обрання файлу при імпорті.

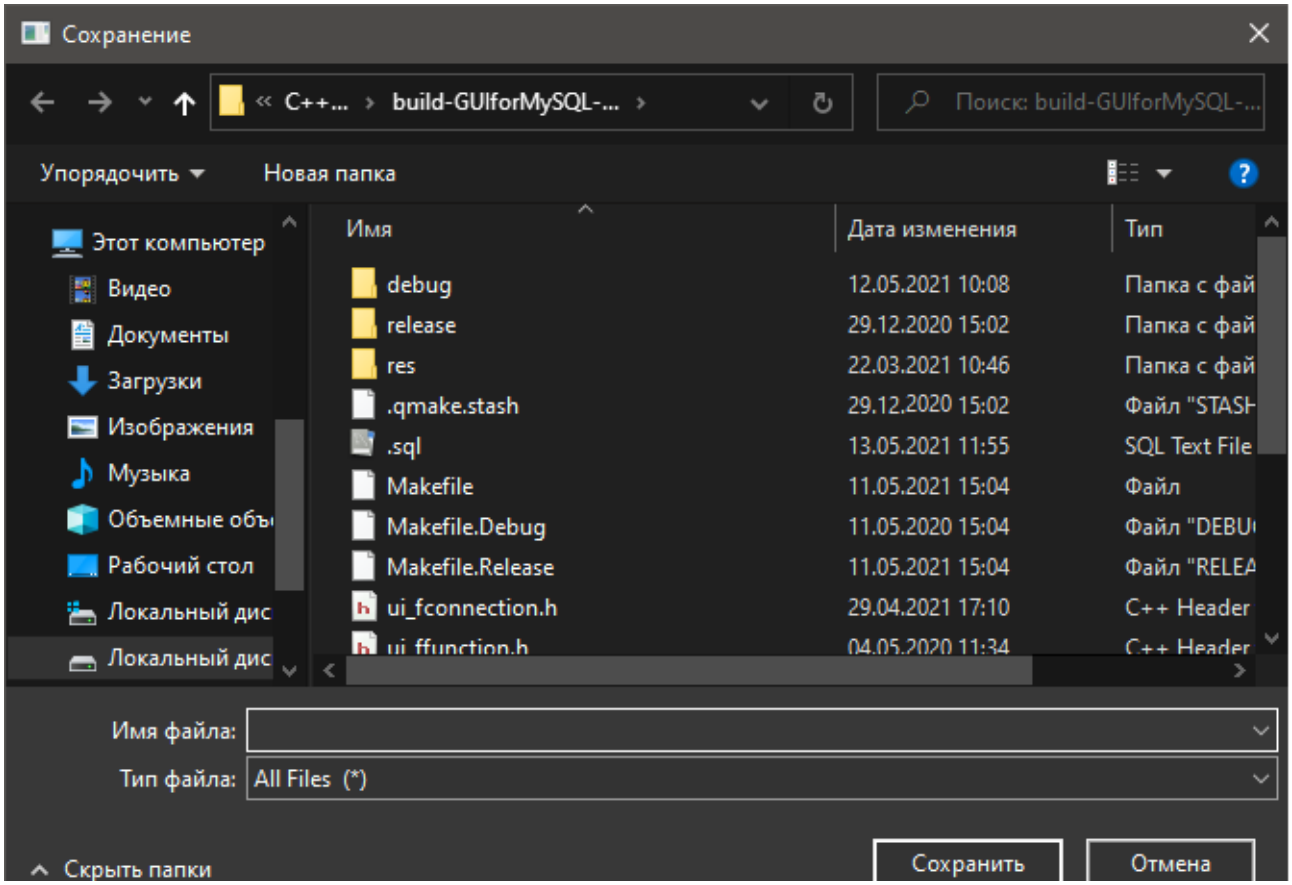

Рис. 2.23.Діалогове вікно обрання місця збереження

Меню Create, що показане на рис. 2.24, містить такі підпункти, як: Create database, Create table, Create trigger, Create procedure. Дані підпункти продубльовані над деревом наповнення на головному вікні (рис. 2.24).

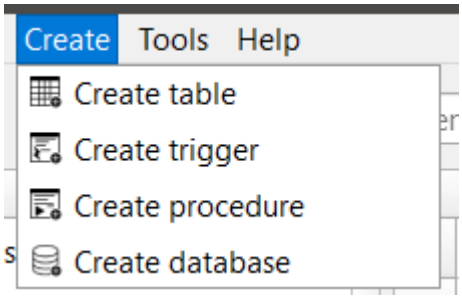

Рис. 2.24. Пункти в підменю Create

Пункт Create database відповідає за створення нової, порожньої бази даних. На рис. 2.25 показано вікно введення ім’я, після чого буде створена нова база даних.

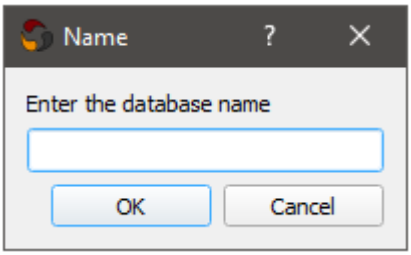

Рис. 2.25. Діалогове вікно введення імені

Якщо база даних з таким ім’ям вже існує, виведеться відповідна помилка. На рис. 2.26. зображено вікно, що з’являтиметься при виникненні помилок на будь-якому етапі роботи програми.

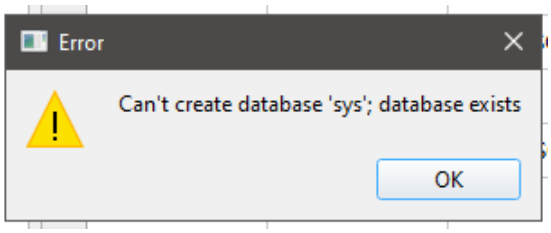

Рис. 2.26. Діалогове вікно помилки

Для створення таблиці потрібно обрати підпункт Create table. З’явиться вікно створення таблиці, що показано на рис. 2.27.

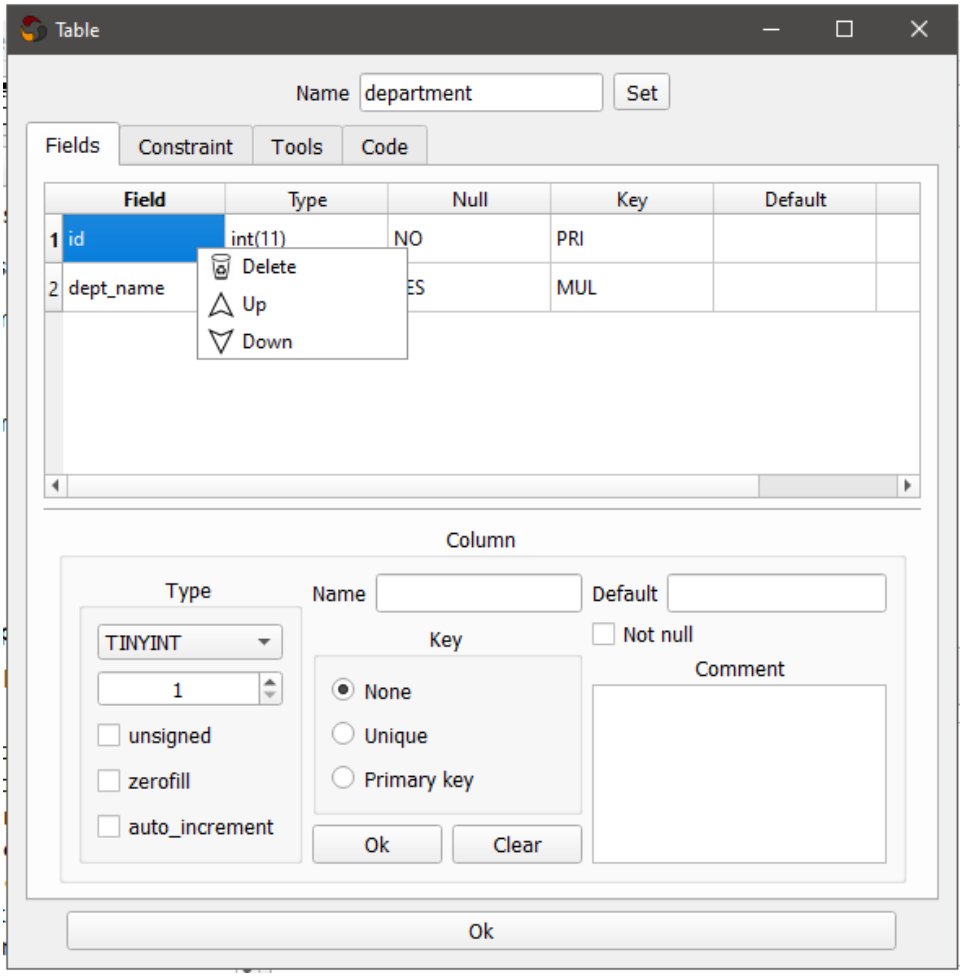

Рис. 2.27. Вікно створення таблиці

Вікно має чотири вкладки: Fields, Constraint, Tools та Code. Спочатку потрібно ввести ім’я таблиці, в полі над вкладками, та натиснути на кнопку Set. Створиться таблиця з відповідним ім’ям і полем за замовчуванням. На вкладці Fields створюються та редагуються поля для таблиць. Щоб створити нове поле, потрібно заповнити всі поля в Column та натиснути Add (рис. 2.28). Кнопка Clear відповідає за очищення полів в Column. Для редагування поля, потрібно натиснути на відповідне поле в таблиці, після чого властивості поля будуть відображені в Column. Якщо клікнути правою кнопкою миші по таблиці, то відкриється контекстне меню з наступними пунктами: Delete, Up, Down (див.

рис. 2.29). Перший пункт видаляє активне поле, Up переміщує поле на одну позицію вгору, а Down вниз.

На рис. 2.28 зображено вкладку Constraint, на ній до таблиці додаються обмеження зовнішнього ключа та правила Check. Зверху, в списку, відображаються всі обмеження. При подвійному кліку обмеження видаляється. Для додавання нового обмеження Check потрібно ввести в поле, зліва під списком, умову та натиснути Add. Щоб додати обмеження зовнішнього ключа, потрібно з випадаючих списків обрати поле, таблицю та її поле, дії при додаванні та видалені, і натиснути Add. Нове обмеження з’явиться додасться до таблиці і з’явиться в списку.

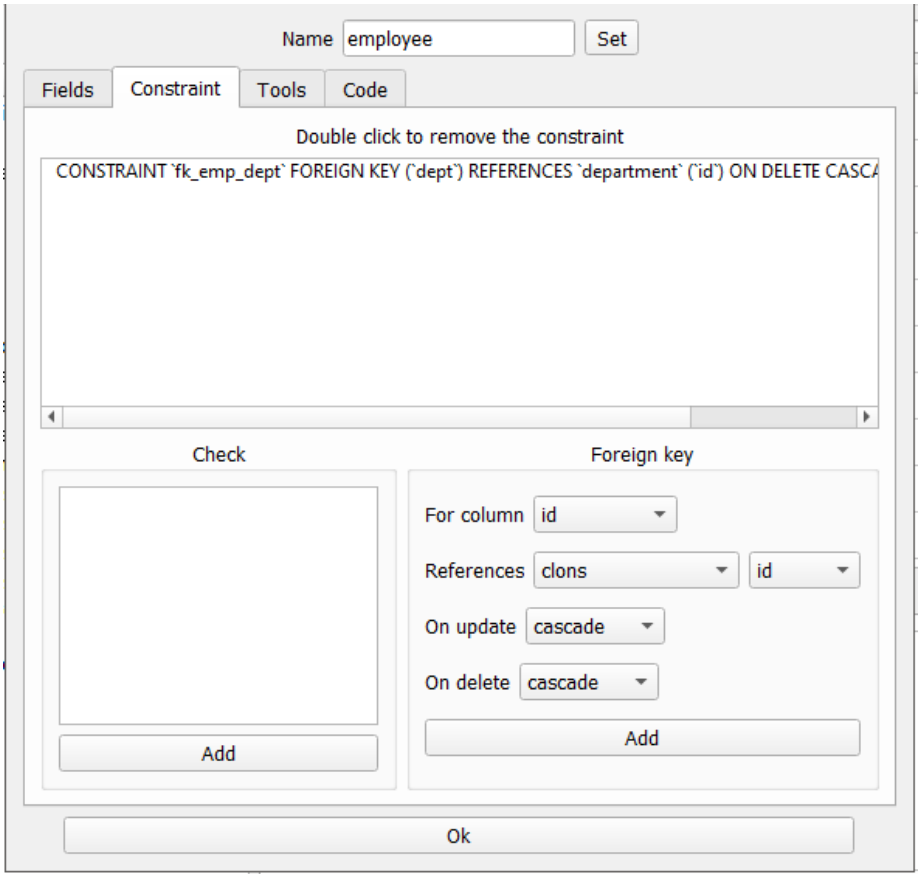

Рис. 2.28. Вікно додавання обмежень

На вкладці Tools, форма наведена на рис. 2.29, можливо встановити коментар до таблиці, змінити систему зберігання даних та клонувати структуру таблиці. Зверху розташовується поле для вводу коментаря до таблиці, для

підтвердження змін кнопка Set. Поле Engine відповідає за встановлення системи зберігання даних. Для перегляду доступних систем зберігання даних натисніть на «?». Щоб змінити поточну систему, потрібно з випадаючого списку обрати потрібну систему та натиснути Convert. Проєкт дає можливість переносити структуру інших таблиць в поточну. Для цього в полі Clone з випадючого списку оберіть таблицю для клонування та натисніть Clone. Якщо встановити галочка у напроти With data, то при клонуванні також перенесуться всі дані.

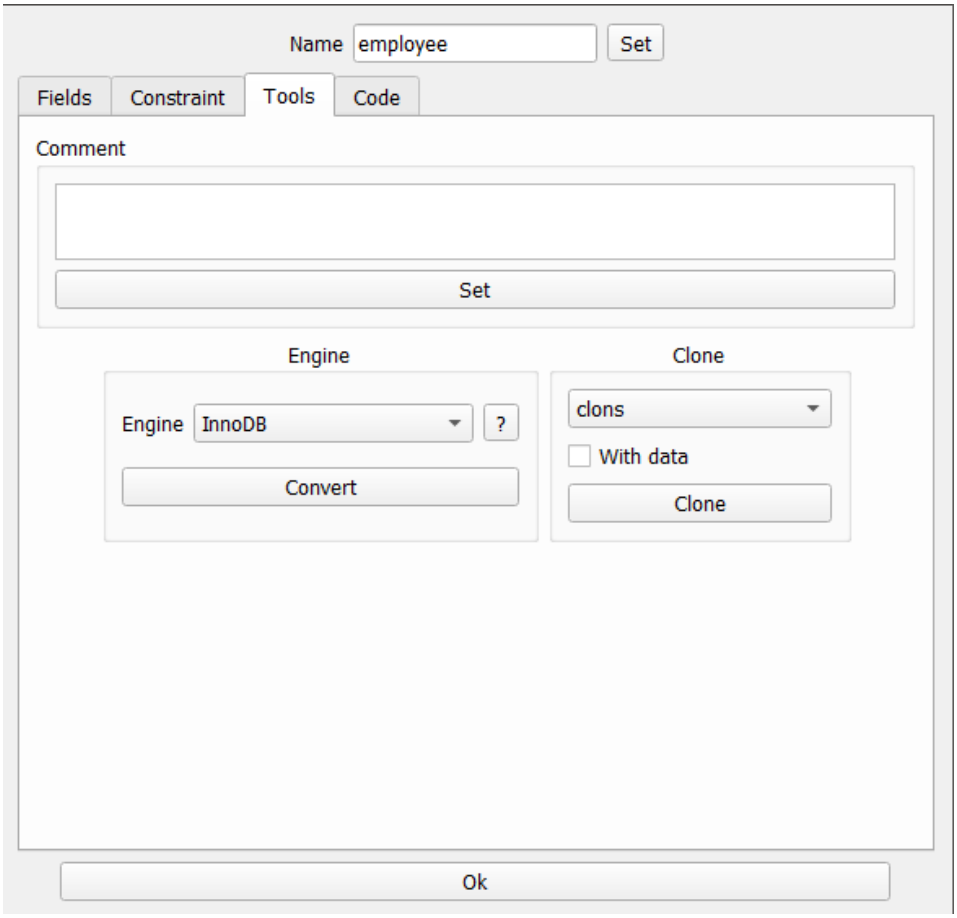

Рис. 2.29. Розділ інструментів

На рис. 2.30 показано останню вкладку, на ній відображається код таблиці.

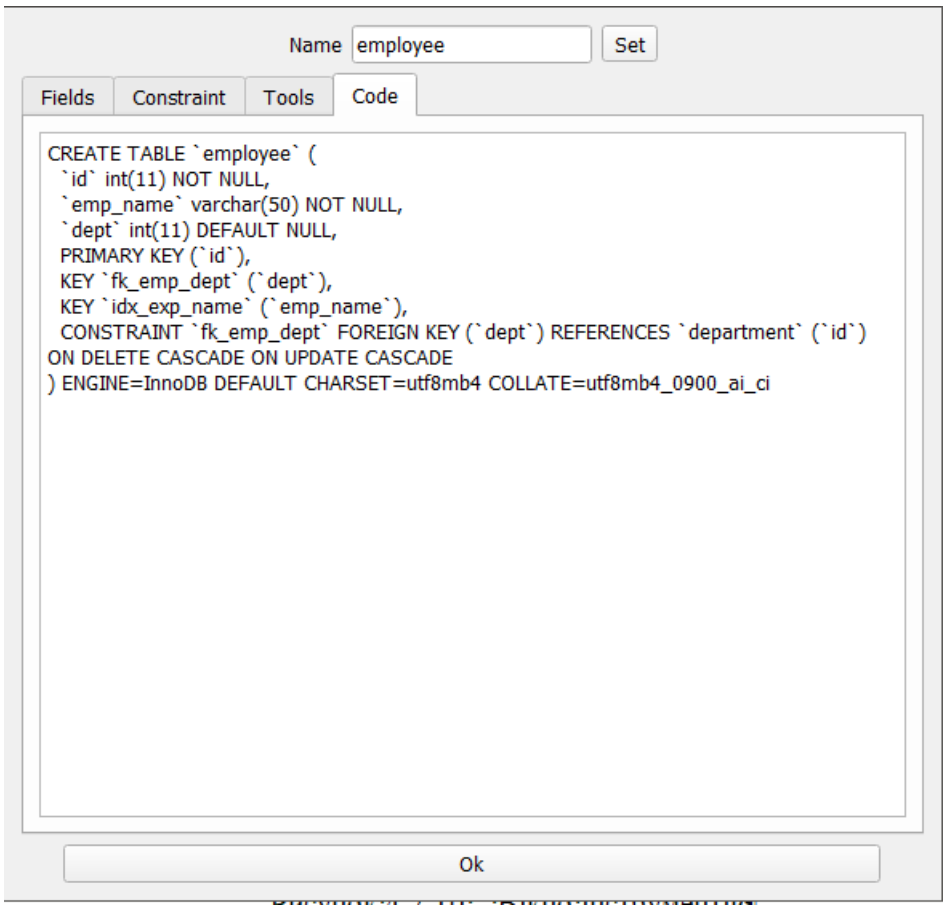

Рис. 2.30. Розділ перегляду коду

Для створення тригеру потрібно обрати пункт Create trigger в підменю Create (рис. 2.31). На рис. 2.32 показано вікно створення тригеру. Спочатку, вверху вікна, потрібно ввести ім’я тригеру, обрати час виконання, подію, а також таблицю для якої створюється тригер. Далі введіть код тригеру, під час вводу працюватиме автодоповнювач коду, та натисніть Execute.

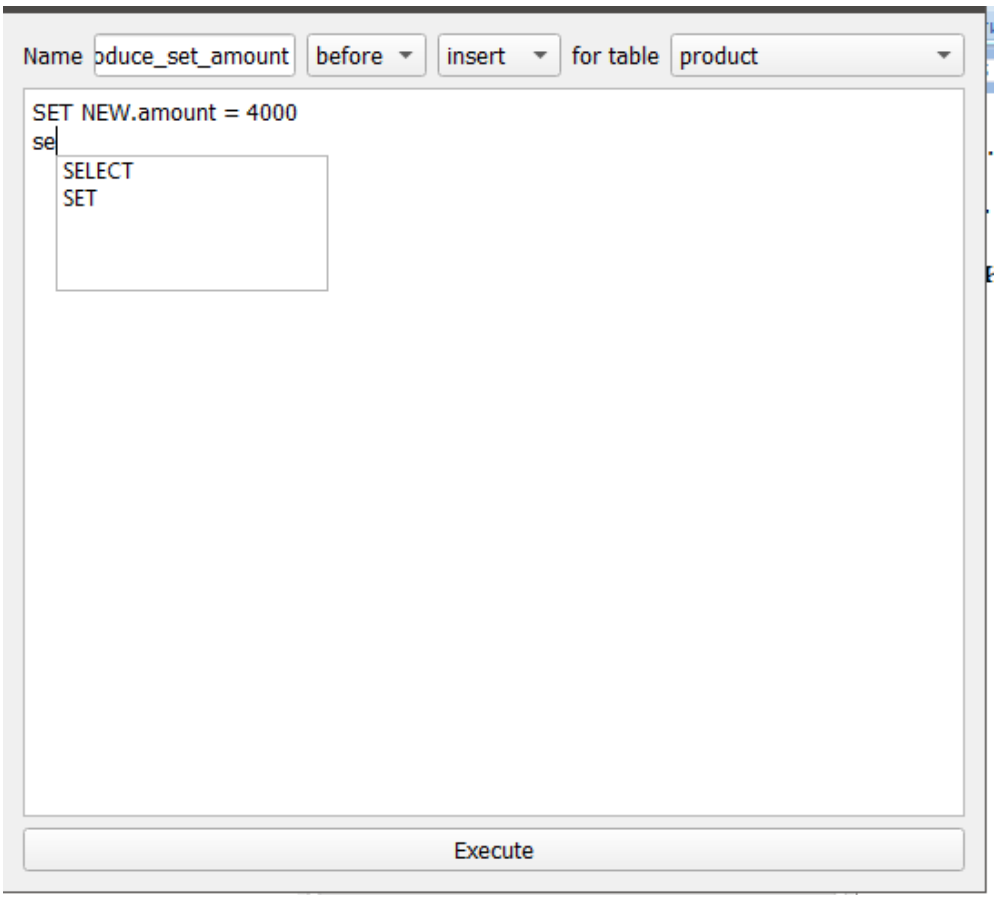

Рис. 2.31. Вікно створення тригеру

Пункт Create procedure в підменю Create відповідає за створення процедур. Вікно створення процедури показано на рис. 2.32. В поле Name вводиться ім’я процедури. У випадючий список Parametrs вводяться параметри для процедури, для видалення параметру натисніть на кнопку «X». Під полем для ім’я розташовується поле для введення коментарю. В полі для введення коду процедури працюватиме автодоповнювач. Для додавання процедури натисніть Execute.

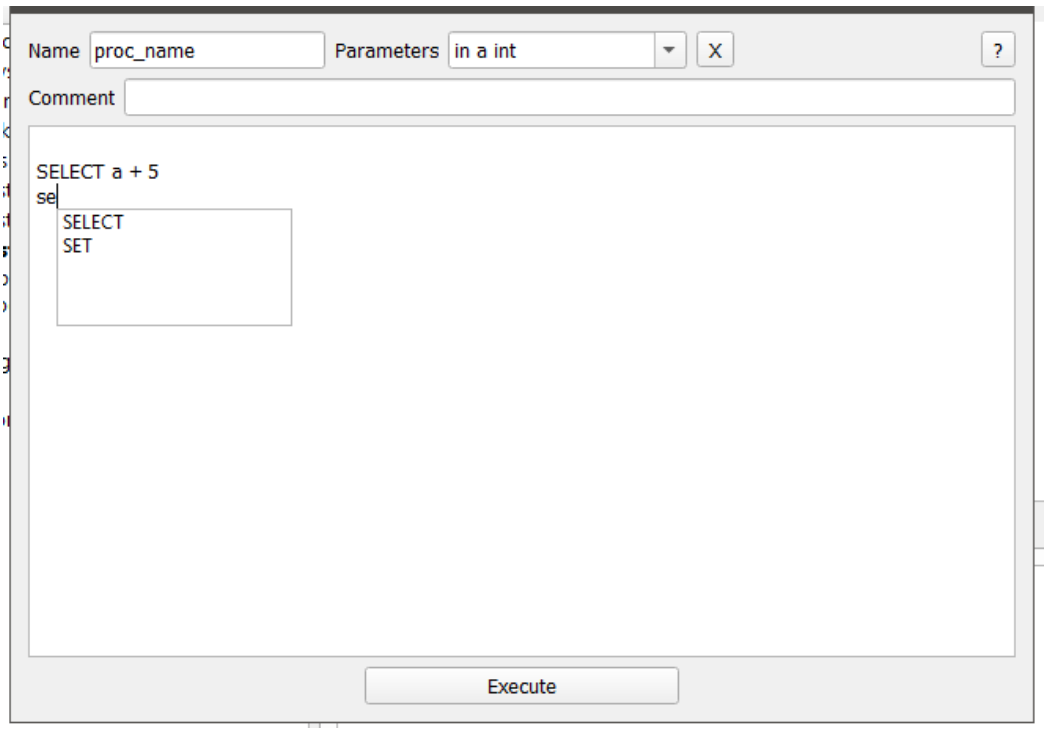

Рис. 2.32. Створення процедури

Для переходу у вікно роботи зі скриптами потрібно обрати пункт меню Skripts, як показано на рис. 2.33.

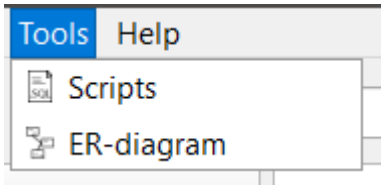

Рис. 2.33. Пункти в підменю Tools

На рис. 2.34 зображено вікно роботи зі скриптами, в даному вікні спочатку потрібно обрати кореневий каталог, де знаходяться скрипти, по кнопці Open Folder.

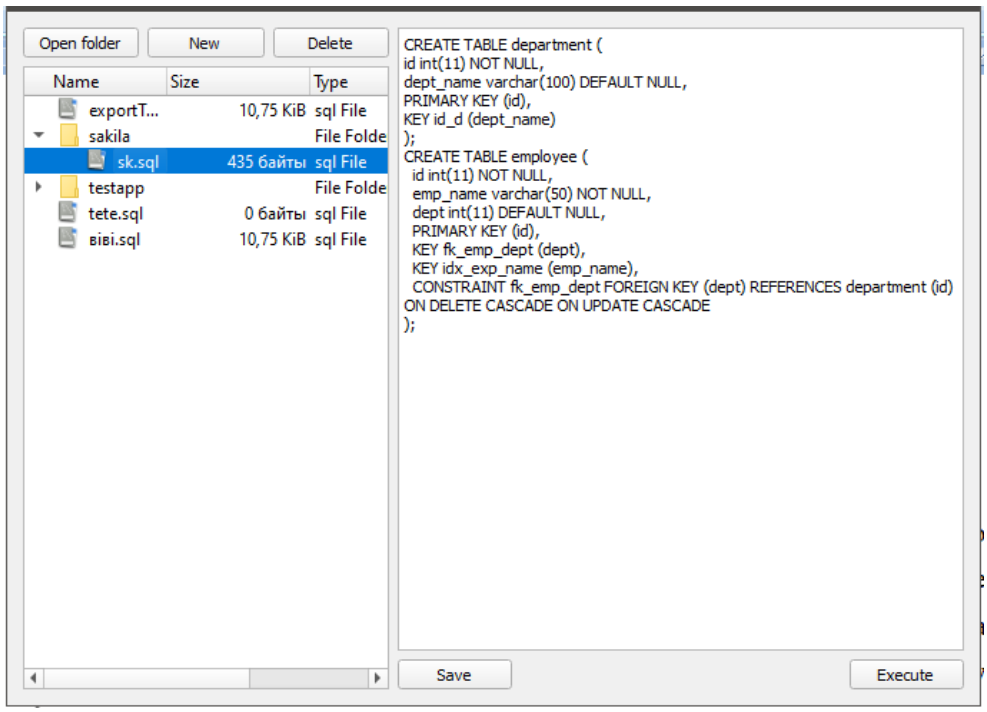

Рис. 2.34. Вікно роботи зі скриптами

Зліва відображається файлова система для навігації. При натисканні на файл з розширенням sql текст файлу буде поміщений у полі з права. В даному полі можливо змінювати текст файлу. Кнопка Save зберігає зміни до файлу. Кнопка Delete видаляє файл чи директорію після підтвердження дії, показано на рис. 2.35.

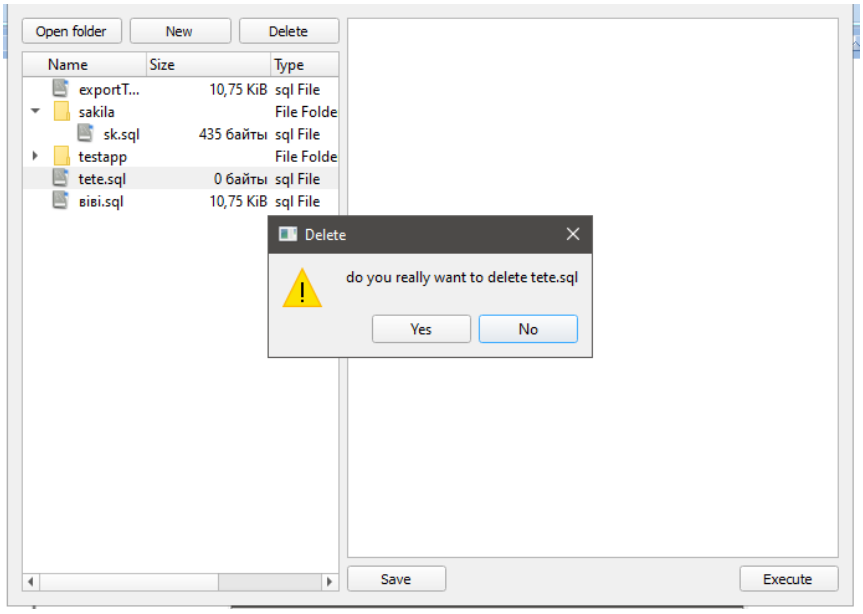

Рис. 2.35. Видалення файлу

Кнопка New створює файл чи директорію в поточному каталозі, вікно створення показано на рис. 2.36. Щоб запустити скрипт на виконання, потрібно натиснути на Execute. На рис. 2.37 показано вікно успішного виконання скрипту.

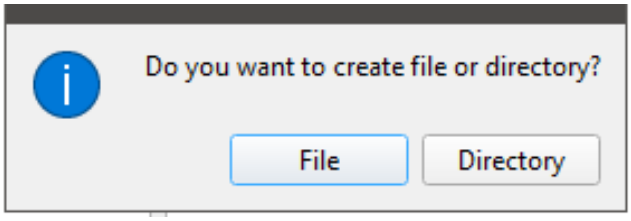

Рис. 2.36. Діалогове вікно вибору створюваного файлу

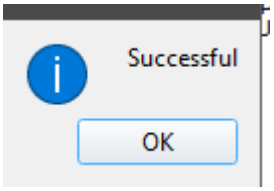

Рис. 2.37. Діалогове вікно успішного виконання скрипту

2.6.5. Тестування програми

При тестуванні програмного проєкту було обрано метод білого ящика. Тестування методом білого ящика - метод тестування програмного забезпечення, який передбачає, що реалізація системи відома тестувальнику. Вибираються вхідні значення, ґрунтуючись на знанні коду, який буде їх обробляти. Точно так само ми знаємо, яким повинен бути результат цієї обробки. Знання всіх особливостей продукту і його реалізації - обов'язкові для цієї техніки. Тестування білого ящика - поглиблення в внутрішній устрій системи, за межі її інтерфейсів.

Даний метод обрано, так як автор виконував роль тестувальника, тобто йому була відома реалізація системи, що є обов’язковим для даного методу. Таким чином, конкретний вид тестування покриває як умога більше різних випадків роботи, порівнюються фактично отримані дані з очікуваними та

тестування може проводитись на різних етапах виробництва, навіть без існуючого інтерфейсу.

Спочатку було виявлені всі недоліки при створені таблиць. Одним з таких недоліків була неможливість створювати більше одного обмеження для поля. Наприклад, поле не могло бути зовнішнім ключем для двох таблиць або містити більше одного обмеження Check. Для усунення даної проблеми, перероблено інтерфейс створення таблиць. Тепер у вікні створення таблиці є окрема вкладка для додавання та видалення обмежень, що не прив’язана до конкретного поля, а працює на рівні з таблицею. На рис. 2.38 показано новий інтерфейс з усуненими проблемами.

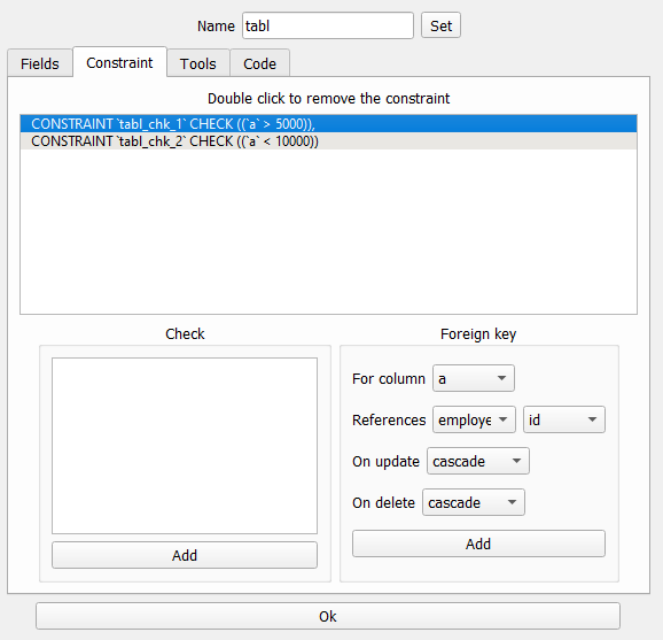

Рис. 2.38. Інтерфейс роботи з обмеженнями

При обранні типу поля, користувачу забороняється обирати ті властивості, що не відносяться до обраного типу поля. Наприклад, при обранні типу Char, користувач не може обрати такі властивості як: Unsigned, Zerofill, Auto_increment.

На рис. 2.39 – 2.40 показано різні властивості, що змінюються в залежності від обраного типу.

Рис. 2.39. Властивості типу Char

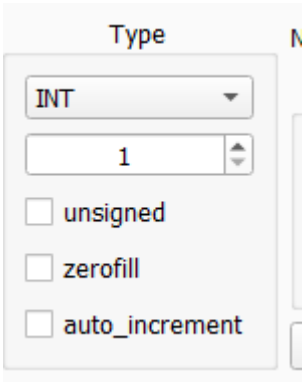

Рис. 2.40. Властивості типу Int

Також використано по максимуму компонентів QComboBox для уникнення некоректних даних.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Вхідні дані:

- передбачуване число операторів - 1500;
- коефіцієнт складності програми - 2;
- коефіцієнт корекції програми в ході її розробки - 0,08;
- годинна заробітна плата програміста, грн./год - 30.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_u + t_a + t_n + t_{отл} + t_д, \text{ людино-годин,} \quad (3.1)$$

де to – витрати праці на підготовку й опис поставленої задачі (50),

ти – витрати праці на дослідження алгоритму рішення задачі,

та – витрати праці на розробку блок-схеми алгоритму,

тп – витрати праці на програмування по готовій блок-схемі,

тотл – витрати праці на налагодження програми на ЕОМ,

тд – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \times C \times (1+ p) \text{ , людино-годин,} \quad (3.2)$$

де q - передбачуване число операторів,

C - коефіцієнт складності програми;

p - коефіцієнт кореляції програми в ході її розробки.

$$Q = 1500 \cdot 2 \cdot (1 + 0{,}08) = 3240 \text{ людино-годин.} \quad (3.3)$$

Витрати праці на вивчення опису задачі ti визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{QB}{(75...85)K}, \text{ людино-годин,} \quad (3.4)$$

де B – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі; B=1.2 … 1.5,

k – коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності.

Витрати праці на розробку алгоритму рішення задачі:

$$t_u = \frac{3240 \cdot 1{,}3}{80 \cdot 1{,}2} = 44, \text{ людино-годин.} \quad (3.5)$$

Витрати на складання програми по готовій блок-схемі:

$$t_a = \frac{Q}{(20...25)K} \quad \text{людино-годин.} \quad (3.6)$$

$$t_a = \frac{3240}{22 \cdot 1{,}2} = 123 \text{ людино-годин.} \quad (3.7)$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)K} \quad \text{людино-годин.} \tag{3.8}$$

$$t_n = \frac{3240}{23 \cdot 1{,}2} = 117 \text{ людино-годин.} \tag{3.9}$$

Витрати праці на налагодження програми на ЕОМ:

$$t_{отл} = \frac{Q}{(4...5)K} \quad \text{людино-годин.} \tag{3.10}$$

$$t_{отл} = \frac{3240}{5 \cdot 1{,}2} = 540 \text{ людино-годин.} \tag{3.11}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до}, \text{ людино-годин,} \tag{3.12}$$

де tдр - трудомісткість підготовки матеріалів і рукопису.

$$t_{др} = \frac{Q}{(15...20)K}, \text{ людино-годин.} \tag{3.13}$$

$$t_{др} = \frac{3240}{18 \cdot 1{,}2} = 150 \text{ людино-годин,} \tag{3.14}$$

де tдо - трудомісткість редагування, печатки й оформлення документації.

$$t_{до} = 0{,}75 \cdot t_{др}, \text{ людино-годин.} \tag{3.15}$$

$$t_{до} = 150 + 73 = 223 \text{ людино-годин.} \tag{3.16}$$

Отримуємо трудомісткість розробки програмного забезпечення:

$$t = 50 + 44 + 123 + 117 + 540 + 223 = 1097 \text{ людино-годин.} \tag{3.17}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ Кпо включають витрати на заробітну плату виконавця програми Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн.} \quad (3.18)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн,} \quad (3.19)$$

де t - загальна трудомісткість, людино-годин,

Спр - середня годинна заробітна плата програміста, грн/година.

$$З_{ЗП} = 1097 \cdot 30 = 32910 \text{ грн.} \quad (3.20)$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ:

$$З_{МВ} = t_{отл} \times C_{М}, \text{ грн,} \quad (3.21)$$

де tотл - трудомісткість налагодження програми на ЕОМ, год.,

Смч - вартість машино-години ЕОМ, грн/год.

Визначені в такий спосіб витрати на створення програмного забезпечення є частиною одноразових капітальних витрат на створення АСУП.

$$З_{МВ} = 540 \times 5 = 2700 \text{ грн.}, \quad (3.22)$$

$$Кпо = 2700 + 32910 = 35610 \text{ грн.} \quad (3.23)$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{ міс.,} \tag{3.24}$$

де Bk - число виконавців,

Fp - місячний фонд робочого часу (при 40 годинному робочому тижні Fp=176 годин).

$$T = \frac{1097}{1 \cdot 176} = 6.23 \text{ міс.} \tag{3.25}$$

Визначено трудомісткість розробленої інформаційної системи (1097 люд-год), проведений підрахунок вартості роботи по створенню програми (39610 грн.) та розраховано час на його створення (6,23 міс).

ВИСНОВКИ

Метою кваліфікаційної роботи бакалавра «Розробка інтегрованого середовище розробки для СУБД MySQL засобами середовища Qt Creator 4.11», є розробка програмного модуля, який призначений для полегшення розуміння СУБД та створення зручного середовища для її швидкого створення і налаштування.

Розробка програми даної роботи призначена для налагодження продуктивнішої розробки баз даних та легшого розуміння структури для новачків

В ході виконання роботи розроблений програмний додаток, що дозволяє створювати бази даних в СУБД MySQL.

Інтерфейс додатку виконаний у сучасному дизайні та забезпечує реалізацію всіх найвикористовуваних функцій СУБД.

Основні функції СУБД – створення зручного інтерфейсу для вирішення основних задач використання БД:

– створення та редагування бази даних, її структури;
– вибір, пошук інформації за певними критеріями;
– захист цілісності даних.

Для нереалізованих функцій передбачений інтегрований командний рядок, котрий дасть змогу користувачу надсилати свої запити до бази даних.

У майбутньому проєкт буде розширено, буде доданий новий функціонал і поліпшиться існуючий. Також, буде додано підтримку користувацьких додатків, що розширюватимуть функціонал за бажанням користувача.

Для реалізації програмного продукту обрано мову C++ в інтегрованому середовищі Qt Creator. Підставою для обрання C++ стала швидкість та гнучка робота з пам’ятю. Середвище Qt Creator пропонує нативну підтримку бібліотеки Qt та кросплатформену компіляцію.

В економічному розділі визначено трудомісткість розробленої інформаційної системи (1097 люд-год), проведений підрахунок вартості роботи по створенню програми (39610 грн.) та розраховано час на його створення (6,23 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання : (ГОСТ 7.1-2003, IDT) : ДСТУ ГОСТ 7.1:2006. – Чинний з 2007–07–01. – К. : Держспоживстандарт України, 2007. – 47 с. – (Система стандартів з інформації, бібліотечної та видавничої справи) (Національний стандарт України).

2. Бусигін Б.С., Коротенко Г.М., Коротенко Л.М. Прикладна інформатика. Підручник для студентів комп'ютерних спеціальностей. – Дніпропетровськ: Видавництво НГУ, 2004. – 559 с. URL: http://www.programmer.dp.ua/book-ua-k01.php. дата звернення: 15.03.2019.

3. Бусыгин Б.С., Дивизинюк М.М., Коротенко Г.М., Коротенко Л.М. Введение в современную информатику. Учебник. – Севастополь: Издательство СНУЯЭиП, 2005. – 644 с. / URL: http://www.programmer.dp.ua/book-ru-k02.php. дата звернення: 23.05.2021.

4. Вікіпедія MySQL Workbench Features (англійською) URL: https://www.mysql.com/products/workbench/features.html Процитовано 2010-02-25. дата звернення: 23.05.2021.

5. Вікіпедія Адміністрування_баз_даних URL: https://uk.wikipedia.org/ wiki/ Адміністрування_баз_даних_та_автоматизація дата звернення: 23.05.2021.

6. Вікіпедія Управління_базами_даних URL: https://uk.wikipedia.org/ wiki/Система_ управління_базами_даних дата звернення: 23.05.2021.

7. Вікіпедія ODBC URL: https://uk.wikipedia.org/wiki/ODBC дата звернення: 23.05.2021.

8. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення : ДСТУ 3008-95. – Чинний від 1996–01–01. – К. : Держстандарт України, 1996. – 39 с.

9. ДСТУ 2394-94 Інформація та документація. Комплектування фонду, бібліографічний опис, аналіз документів. Терміни та визначення. – Чинний від 01.01.1995. - Київ : Держстандарт України, 1994. – 88 с.

10. Методичні вказівки з виконання економічного розділу в дипломних проектах студентів спеціальності “Комп’ютерні системи ” / Укладачі О.Г. Вагонова, Нікітіна О.Б. Н.Н. Романюк – Дніпропетровськ: Національний гірничий університет. – 2013. – 23с.

11. Методичні рекомендації до виконання кваліфікаційних робіт бакалаврів напряму підготовки 122 «Комп’ютерні науки» галузі знань 12 Інформаційні технології/, Л.М. Коротенко , О.С. Шевцова; Нац. гірн. ун-т. – Д : ДВНЗ НГУ, 2019. – 65 с.

12. Методичні рекомендації щодо написання, оформлення та представлення учнівських науково-дослідницьких робіт учнів – членів Малої академії наук України / Г.Г. Півняк, Л.М. Коротенко, І.М. Удовик, Є.М. Головня – Д.: ДВНЗ «Національний гірничий університет», 2017. – 24 с.

13. Офіційний сайт середовища розробки Visual Studio Code URL: https://code.visualstudio.com/docs

14. Офіційний сайт програми «dbForge Studio» URL: https://www.devart.com/ru/dbforge/mysql/studio/. дата звернення: 23.05.2021.

15. Офіційний сайт програми «MySQL Workbench» URL: https://www.mysql.com/products/workbench/. дата звернення: 23.05.2021.

16. Офіційний сайт бібліотеки «Qt» URL:: https://doc.qt.io/. дата звернення: 23.05.2021.

17. Офіційний сайт мови програмування URL: C++: https://isocpp.org/. дата звернення: 23.05.2021.

18. Офіційний сайт СУБД «MySQL» URL:: https://dev.mysql.com/doc/. дата звернення: 23.05.2021.

19. Скорочення слів в українській мові у бібліографічному описі. Загальні правила та вимоги : ДСТУ 3582-97. – Чинний від 1998–07–01. – К. : Держстандарт України, 1998. – 24 с. – (Державний стандарт України).

20. SQL. URL: https://ru.wikipedia.org/wiki/SQL. Дата звернення: 15.01.2020.

21. Qt. URL: https://uk.wikipedia.org/wiki/Qt. Дата звернення: 16.01.2021.

ДОДАТОК А

КОД ПРОГРАМИ

AUTOHELPER_cpp

```
#ifndef AUTOHELPER_H
#define AUTOHELPER_H

#include <QListWidget>

class AutoHelper : public QListWidget
{
   Q_OBJECT
private:
   QStringList _words;
   QMap<QString, QStringList> _additional;

public:
   AutoHelper(QWidget *parent = nullptr);
   ~AutoHelper();

   void setWordTip();
   void setAddTip(QMap<QString, QStringList>);

signals:
   void choosedItem(QListWidgetItem *);

public slots:
   void showTips(QString text);
   void setFocusOnTips();

   // QWidget interface
protected:
   void keyPressEvent(QKeyEvent *event);
};

#endif // AUTOHELPER_H
#ifndef CDATABASECONTROL_H
#define CDATABASECONTROL_H
```

Object.cpp

```
#include <QObject>
#include <QSqlDatabase>
#include <QSqlQueryModel>
#include <QSqlQuery>
#include <QSqlError>
#include <QSqlTableModel>
#include <QSqlRecord>
#include <QSqlQueryModel>

class CDatabaseControl : public QObject
{
   Q_OBJECT

public:
   explicit CDatabaseControl(QObject *parent = nullptr);
   ~CDatabaseControl();

   bool connectToDatabase(QString driver, QString dbName, QString hostname,
                  int port, QString username, QString password);
   QStringList databasesList();
```

```
    QMap<QString, QString> tablesList(QString dbName);
    QStringList procedureList(QString dbName);
    QStringList columnList(QString tableName);
    QStringList triggersList(QString dbName);
    QString workDatabaseName();
    QString lastErrorText(QSqlQuery query);
    bool changeWorkDatabase(QString dbName);
    QString getCurrentDatabaseName();
    QSqlTableModel * getTable(QString dbName, QString tableName);
    QSqlTableModel *getLastTable();
    QSqlQueryModel * lastQueryModel();
    QSqlQuery * lastQuery();
    QMap<int, QStringList> getTypeOfDefinition();
    void changeEngine(QString tableName, QString engine);

    QSqlQueryModel *execTemporaryQuery( QString query);

private:
    QSqlDatabase _database;
    QString _currentDataBaseName;
    QSqlQueryModel * _queryModel;
    QSqlQuery * _query;
    QSqlQuery _workQuery;
    QSqlError queryError;
    QSqlTableModel * _tableModel;
    QMap<int, QStringList> _typeOfDefenition;

public slots:
    bool execQueryModel(QString query);
    bool execQuery(QString query);

signals:
    void connectionBreak();
    void error(QString);
    void databaseChanged(QString dbName);
    void output(QString);
    void showQueryModel(QSqlQueryModel *);
    void querySucces(QString);
    void successfully(QString);

};

#endif // CDATABASECONTROL_H
#ifndef CONSOLE_H
#define CONSOLE_H
```

DIALOGS**.cpp**

```
#include <QWidget>
#include <QPlainTextEdit>
#include <QListWidgetItem>

class Console : public QPlainTextEdit
{
    Q_OBJECT
public:
    explicit Console(QWidget *parent = 0);
    ~Console();

    void setPromt(QString promt);
    void insertPromt();
    //void printOutput(QString);
    void changeCurrentWord(QListWidgetItem * item);
```

```
public slots:
    void printOutput(QString);

private:
    QString promt;
    int amountBlock;
    bool lock;
    void moveCursor(int);
    void queryEnter();
    void scrollDown();

    // QWidget interface
protected:
    void keyPressEvent(QKeyEvent *event);

    // QWidget interface
protected:
    void mousePressEvent(QMouseEvent *event);

    // QWidget interface
protected:
    void mouseDoubleClickEvent(QMouseEvent *event);

signals:
    void queryEntered(QString query);
    void tabPress();
    void focusOut();

    // QWidget interface
protected:
    void contextMenuEvent(QContextMenuEvent *event);

    // QWidget interface
protected:
    void mouseReleaseEvent(QMouseEvent *event);

    // QWidget interface
protected:
    void focusOutEvent(QFocusEvent *event);
};

#endif // CONSOLE_H
#ifndef DIALOGS_H
#define DIALOGS_H
```

QDialog.cpp

```
#include <QDialog>
#include <QString>

class Dialogs
{
    static int question(QString text)
    {
        return 1;
    }
};

#endif // DIALOGS_H
#ifndef FCONNECTION_H
#define FCONNECTION_H
```

FFUNCTION.cpp

```
#include <QWidget>

namespace Ui {
class fConnection;
}

class fConnection : public QWidget
{
   Q_OBJECT

public:
   explicit fConnection(QWidget *parent = nullptr);
   ~fConnection();

private slots:
   //Call administrative tools from control panel
   void on_clbToAdministrativeTools_clicked();

private:
   Ui::fConnection *ui;
};

#endif // FCONNECTION_H
#ifndef FFUNCTION_H
#define FFUNCTION_H
```

QSignalLabel.cpp

```
#include "QSignalLabel.h"

#include <QWidget>
#include <QLabel>

namespace Ui {
class fFunction;
}

class fFunction : public QWidget
{
   Q_OBJECT

public:
   explicit fFunction(QWidget *parent = nullptr);
   ~fFunction();

private:
   Ui::fFunction *ui;

   QSignalLabel * lbl;


   // QWidget interface
protected:
   void resizeEvent(QResizeEvent *event);
};

#endif // FFUNCTION_H
#ifndef FMAINWINDOW_H
#define FMAINWINDOW_H
```

FPROCEDURE.cpp

```
#include <QMainWindow>
```

```
#include <QTreeWidget>
#include <QMenu>

#include "fconnection.h"
#include "ffunction.h"
#include "fprocedure.h"
#include "fsettings.h"
#include "ftable.h"
#include "ftrigger.h"
#include "cdatabasecontrol.h"
#include "autohelper.h"
#include "fskripts.h"

QT_BEGIN_NAMESPACE
namespace Ui { class fMainWindow; }
QT_END_NAMESPACE

class fMainWindow : public QMainWindow
{
    Q_OBJECT

public:
    fMainWindow(QWidget *parent = nullptr);
    ~fMainWindow();

private slots:
    void on_twContent_itemDoubleClicked(QTreeWidgetItem *item, int column);
    void showError(QString);
    void reloadTree();
    void toOutput(QString);
    void toError(QString);
    void showTableInTableView(QAbstractItemModel *);
    void callFormCreateTable();
    void callFormFProcedure();
    void callFormFTrigger();
    void setModel(QString query);
    void showMessage(QString str);
    void hideHelper();
    void createDataBase();
    void exportDB();

    void callContextMenuForTreeWidget(const QPoint &point);

    //slots for context menu
    void contextChange();
    void contextDelete();

    void on_pteConsole_textChanged();

private:
    //Windows
    fConnection * fconnection;
    fFunction * ffunction;
    fProcedure * fprocedure;
    fSettings * fsettings;
    fSkripts * fskripts;
    fTable * ftable;
    fTrigger * ftrigger;
    AutoHelper * helper;

    QMenu * contentContextMenu;
```

```
    //Database control
    CDatabaseControl * dbControl;

    Ui::fMainWindow *ui;

signals:
    void showFormCreateTable(QString dbName, QString tableName);
    void showFormFProcedure(QString);
    void showFormFTrigger(QString);
    void installDBControl(CDatabaseControl * db);
    void showTips(QString);
    void installHelper(QMap<QString, QStringList> map);


    // QWidget interface
protected:
    void moveEvent(QMoveEvent *event);
    void resizeEvent(QResizeEvent *event);
    void closeEvent(QCloseEvent *event);
    void hideEvent(QHideEvent *event);
};
#endif // FMAINWINDOW_H
#ifndef FPROCEDURE_H
#define FPROCEDURE_H
```

Autohelper.cpp

```
#include <QWidget>
#include "tabFilter.h"
#include "autohelper.h"
#include "cdatabasecontrol.h"

namespace Ui {
class fProcedure;
}

class fProcedure : public QWidget
{
    Q_OBJECT

public:
    explicit fProcedure(QWidget *parent = nullptr);
    ~fProcedure();

signals:
    void showTips(QString);
    void showMessage(QString);

private slots:
    void changeCurrentWord(QListWidgetItem * item);

    void on_wEdit_textChanged();

    void on_wDelete_clicked();

    void on_wSave_clicked();

public slots:
    void setHelper(QMap<QString, QStringList> map);
    void showForm(QString procName);
    void setDBControl(CDatabaseControl * dbcontrol);

private:
```

```
    Ui::fProcedure *ui;
    tabFilter * filter;
    AutoHelper * helper;
    QString procName;
    CDatabaseControl * dbcontrol;

    void reload();
    QString collect();

    // QWidget interface
protected:
    void moveEvent(QMoveEvent *event);
    void resizeEvent(QResizeEvent *event);
    void closeEvent(QCloseEvent *event);
    void hideEvent(QHideEvent *event);
};

#endif // FPROCEDURE_H
#ifndef FSETTINGS_H
#define FSETTINGS_H
```

FSKRIPTS.cpp

```
#include <QWidget>
#include <QTranslator>

namespace Ui {
class fSettings;
}

class fSettings : public QWidget
{
    Q_OBJECT

public:
    explicit fSettings(QWidget *parent = nullptr);
    ~fSettings();

private slots:
    void on_wLang_currentTextChanged(const QString &arg1);

private:
    Ui::fSettings *ui;

    QTranslator translator;

    // QWidget interface
protected:
    void changeEvent(QEvent * pe);
};

#endif // FSETTINGS_H
#ifndef FSKRIPTS_H
#define FSKRIPTS_H

#include <QFileSystemModel>
#include <QWidget>
#include "cdatabasecontrol.h"

namespace Ui {
class fSkripts;
}
```

```
class fSkripts : public QWidget
{
   Q_OBJECT

public:
   explicit fSkripts(QWidget *parent = nullptr);
   ~fSkripts();

public slots:
   void showForm();

private slots:
   void on_wFolder_clicked();

   void on_wTree_clicked(const QModelIndex &index);

   void on_wDelete_clicked();

   void on_wNewFile_clicked();

   void on_wSave_clicked();

   void on_wExecute_clicked();

public slots:
   void setDBControl(CDatabaseControl * dbcontrol);

private:
   Ui::fSkripts *ui;
   QFileSystemModel * dirmodel;
   CDatabaseControl * dbcontrol;

   QString editPath;
};

#endif // FSKRIPTS_H
#ifndef FTABLE_H
#define FTABLE_H
```

QSIGNALLABEL.cpp

```
#include <QMenu>
#include <QWidget>
#include <QListWidgetItem>
#include <QDebug>
#include <QTableView>
#include "cdatabasecontrol.h"

namespace Ui {
class fTable;
}

class fTable : public QWidget
{
   Q_OBJECT

private:
   void toDefaultView();
   void toDefaultFields();
   void toDefaultConstraint();
   void toDefaultTools();
   void toDefaultCode();
```

```
        void setShowedWidgets(int modificator);
        void installShowedWidgets(bool unsign, bool firstParamtrs, bool secondParametrs, bool zerofill, bool
increment);
        void setReferences(QStringList definition, QString columnName);
        //bool checkDublicateKey(QString columnName, QString key);
        QString isCommentColumn(QStringList definition, QString columnName);
        QString isType(QStringList definition, QString columnName);
        int isParametr1(QStringList definition, QString columnName);
        int isParametr2(QStringList definition, QString columnName);
        QString isDefault(QStringList definition, QString columnName);
        int isKey(QStringList definition, QString columnName);
        bool isReferences(QStringList definition, QString columnName);
        QString isOnDelete(QStringList definition, QString columnName);
        QString isOnUpdate(QStringList definition, QString columnName);
        bool isHave(QStringList definition, QString columnName, QString parametr);
        QString collectCommand();
        int checkKey(QStringList definition, QString columnName);
        void showConstraint();

    public:
        explicit fTable(QWidget *parent = nullptr);
        ~fTable();

    public slots:
        void showForm(QString dbName, QString tableName);
        void setDBControl(CDatabaseControl * dbcontrol);

    private slots:
        void on_wType_currentTextChanged(const QString &arg1);
        void toDefaultColumn();

        void columnUp();
        void columnDown();
        void deleteColumn();

        void showTable();

        void on_wTable_customContextMenuRequested(const QPoint &pos);

        void on_wClear_clicked();

        void on_wTable_clicked(const QModelIndex &index);

        void on_wReferencesTable_currentTextChanged(const QString &arg1);

        void on_wAdd_clicked();

        void on_wTabs_currentChanged(int index);

        void on_wConstrainfList_itemDoubleClicked(QListWidgetItem *item);

        void on_pushButton_clicked();

        void on_pushButton_2_clicked();

        void on_wHelpEngines_clicked();

        void on_wConvertEngine_clicked();

        void on_wSetTableName_clicked();

        void on_pushButton_3_clicked();
```

```
    void on_wCreateTemporary_clicked();

    void on_wClone_clicked();

private:
    Ui::fTable *ui;
    CDatabaseControl * dbcontrol;
    QString currentTable;
    QString forDatabase;
    QTableView * showEngines;

    QMenu *menu;
};

#endif // FTABLE_H
#ifndef FTRIGGER_H
#define FTRIGGER_H

#include <QWidget>
#include "tabFilter.h"
#include "autohelper.h"
#include "cdatabasecontrol.h"

namespace Ui {
class fTrigger;
}

class fTrigger : public QWidget
{
    Q_OBJECT

public:
    explicit fTrigger(QWidget *parent = nullptr);
    ~fTrigger();

signals:
    void showTips(QString);

private slots:
    void changeCurrentWord(QListWidgetItem * item);

    void on_wEdit_textChanged();

    void on_wOk_clicked();

public slots:
    void setHelper(QMap<QString, QStringList> map);
    void showForm(QString trigName);
    void setDBControl(CDatabaseControl * dbcontrol);

private:
    Ui::fTrigger *ui;
    tabFilter * filter;
    AutoHelper * helper;
    QString trigName;
    CDatabaseControl * dbcontrol;

    void reload();
    QString collect();

    // QWidget interface
```

```
protected:
   void moveEvent(QMoveEvent *event);
   void resizeEvent(QResizeEvent *event);
   void closeEvent(QCloseEvent *event);
   void hideEvent(QHideEvent *event);
};

#endif // FTRIGGER_H
#ifndef QSIGNALLABEL_H
#define QSIGNALLABEL_H

#include <QLabel>
#include <QMouseEvent>

class QSignalLabel : public QLabel
{
   Q_OBJECT
public:
   QSignalLabel(QWidget *parent = 0) : QLabel(parent){};
signals:
   void clicked();
protected:
   void mouseReleaseEvent(QMouseEvent *e)
   {
      if(e->button() == Qt::LeftButton)
      {
         emit clicked();
      }
   }
};

#endif // QSIGNALLABEL_H
#ifndef CTEST_H
#define CTEST_H
```

Console.cpp

```
#include <QObject>

class tabFilter : public QObject
{
   Q_OBJECT
public:
   explicit tabFilter(QObject *parent = nullptr);

signals:
   void tabPress();

public slots:

   // QObject interface
public:
   bool eventFilter(QObject *watched, QEvent *event);
};

#endif // CTEST_H
#include "autohelper.h"
#include <QDebug>
#include <QKeyEvent>
#include <QTableView>

AutoHelper::AutoHelper(QWidget *parent)
   : QListWidget(parent)
```

```
{
    setWindowFlag(Qt::FramelessWindowHint);
    setWindowFlag(Qt::WindowStaysOnTopHint);
}

AutoHelper::~AutoHelper()
{

}

void AutoHelper::setWordTip()
{
    QStringList stringList;
    QFile textFile(":/new/helper/autohelper/words.txt");
    if(textFile.open(QFile::OpenModeFlag::ReadOnly))
    {
        QTextStream textStream(&textFile);
        while (true)
        {
            QString line = textStream.readLine();
            if (line.isNull())
            {
                break;
            }
            else
            {
                stringList.append(line);
            }
        }
        this->_words = stringList;
    }
    else
    {
        qDebug() << "not open";
    }
}

void AutoHelper::setAddTip(QMap<QString, QStringList> map)
{
    if(map.size() >= 1)
    {
        this->_additional = map;
    }
}

void AutoHelper::showTips(QString text)
{
    this->clear();
    text = text.toUpper();
    if(!text.isEmpty() && !text.isNull())
    {
        for (QString var : this->_words) {
            if(var.indexOf(text) == 0)
            {
                QListWidgetItem * item = new QListWidgetItem(var, this);
            }
        }
        text = text.toLower();
        for (QString var : _additional.keys()) {
            if(var.indexOf(text) == 0)
            {
                QListWidgetItem * item = new QListWidgetItem(var, this);
```

```
            }
        }
        if(text[text.size() - 1] == '.')
        {
            text = text.mid(0, text.size() - 1);
            if(_additional.keys().indexOf(text) >= 0)
            {
                for(QString var : _additional.value(text))
                {
                    QListWidgetItem * item = new QListWidgetItem(var, this);
                }
            }
        }
        if(this->count() == 0)
        {
            hide();
        }
        else
        {
            show();
        }
    }
    else
    {
        hide();
    }

}

void AutoHelper::setFocusOnTips()
{
    setCurrentRow(0);
    this->activateWindow();
}

void AutoHelper::keyPressEvent(QKeyEvent *event)
{
    if(event->key() ==  Qt::Key_Return)
    {
        emit choosedItem(this->currentItem());
    }

    QListWidget::keyPressEvent(event);
}
#include <QDebug>

#include "cdatabasecontrol.h"

CDatabaseControl::CDatabaseControl(QObject *parent) : QObject(parent)
{
    _currentDataBaseName = "mysql";
    _queryModel = new QSqlQueryModel;
    _query = new QSqlQuery;
    _tableModel = new QSqlTableModel;
    _typeOfDefenition.insert(1, {"TINYINT", "INT", "SMALLINT", "MEDIUMINT", "BIGINT"});
    _typeOfDefenition.insert(2, {"CHAR", "VARCHAR"});
    _typeOfDefenition.insert(3, {"DECIMAL", "FLOAT", "DOUBLE"});
    _typeOfDefenition.insert(4, {"DATE", "TIME", "DATETIME", "TIMESTAMP", "YEAR", "TINYBLOB",
                                 "BLOB", "MEDIUMBLOB", "LARGEBLOB", "BOOL", "TINYTEXT",
                                 "TEXT", "MEDIUMTEXT", "LARGETEXT"});
}
```

```
CDatabaseControl::~CDatabaseControl()
{

}

bool CDatabaseControl::connectToDatabase(QString driver, QString dbName, QString hostname,
                                         int port, QString username, QString password)
{
   _database = QSqlDatabase::addDatabase(driver);
   _database.setDatabaseName(dbName);
   _database.setHostName(hostname);
   _database.setPort(port);
   _database.setUserName(username);
   _database.setPassword(password);

   return _database.open();
}

QStringList CDatabaseControl::databasesList()
{
   QStringList databaseList;
   QSqlQuery query;
   if(query.exec("show databases"))
   {
      while (query.next())
      {
         databaseList << query.value(0).toString();
      }

      return databaseList;
   }
   emit error(lastErrorText(query));
   return databaseList;
}

QMap<QString, QString> CDatabaseControl::tablesList(QString dbName)
{
   QMap<QString, QString> tablesList;
   QSqlQuery query;
   if(query.exec("use " + dbName))
   {
      if(query.exec("show table status"))
      {
         while (query.next()) {
            tablesList.insert(query.value(0).toString(), query.value(17).toString());
         }

         query.exec("use " + _currentDataBaseName);
         return tablesList;
      }
   }
   emit error(lastErrorText(query));
   query.exec("use " + _currentDataBaseName);
   return tablesList;
}

QStringList CDatabaseControl::procedureList(QString dbName)
{
   QStringList procedureList;
   if(_workQuery.exec("use " + dbName))
   {
      if(_workQuery.exec("show procedure status where Db = Database() and type = 'PROCEDURE'"))
```

```
        {
            while (_workQuery.next())
            {
                procedureList << _workQuery.value(1).toString();
            }

            _workQuery.exec("use " + _currentDataBaseName);
            return procedureList;
        }
        _workQuery.exec("use " + _currentDataBaseName);
    }
    emit error(lastErrorText(_workQuery));
    return procedureList;
}

QStringList CDatabaseControl::columnList(QString tableName)
{
    QStringList tableList;
    if(_workQuery.exec("describe " + tableName))
    {
        while (_workQuery.next())
        {
            tableList << _workQuery.value(0).toString();
        }

        return tableList;
    }
    emit error(lastErrorText(_workQuery));
    return tableList;
}

QStringList CDatabaseControl::triggersList(QString dbName)
{
    QStringList triggerList;
    QSqlQuery query;
    if(query.exec("use " + dbName))
    {
        if(query.exec("show triggers"))
        {
            while(query.next())
            {
                triggerList << query.value(0).toString();
            }

            query.exec("use " + _currentDataBaseName);
            return triggerList;
        }
    }
    emit error(lastErrorText(query));
    query.exec("use " + _currentDataBaseName);
    return triggerList;
}

QString CDatabaseControl::workDatabaseName()
{
    QSqlQuery query;
    if(query.exec("select Database()"))
    {
        while(query.next())
        {
            return query.value(0).toString();
        }
```

```
    }
    emit error(lastErrorText(query));
    return "";
}

QString CDatabaseControl::lastErrorText(QSqlQuery query)
{
    queryError = query.lastError();
    emit output(queryError.databaseText().remove(0, 42));
    return queryError.databaseText().remove(0, 42);
}

bool CDatabaseControl::changeWorkDatabase(QString dbName)
{
    QSqlQuery query;
    if(query.exec("USE " + dbName))
    {
        _currentDataBaseName = dbName;
        emit databaseChanged(dbName);
        emit output("USE " + dbName);
        return true;
    }
    emit error(lastErrorText(query));
    return false;
}

QString CDatabaseControl::getCurrentDatabaseName()
{
    return _currentDataBaseName;
}

QSqlTableModel * CDatabaseControl::getTable(QString dbName, QString tableName)
{
    QSqlQuery query;
    if(query.exec("USE " + dbName))
    {
        _tableModel->setTable(tableName);
        if(_tableModel->select())
        {
            query.exec("USE " + this->_currentDataBaseName);
            emit output("select * from " + tableName + "; " + "(" + dbName +")");
            return _tableModel;
        }
    }
    emit error("Table not found");
    return nullptr;
}

QSqlTableModel *CDatabaseControl::getLastTable()
{
    return _tableModel;
}

bool CDatabaseControl::execQueryModel(QString query)
{
    emit output(query);
    if(_workQuery.exec(query))
    {
        this->_queryModel->setQuery(query);
        if(_queryModel->rowCount() > 0)
        {
            emit showQueryModel(_queryModel);
```

```
        }
        emit querySucces("Query succes");
        return true;
    }
    emit querySucces(lastErrorText(_workQuery));
    return false;
}

bool CDatabaseControl::execQuery(QString query)
{
    emit output(query);
    if(_query->exec(query))
    {
        return true;
    }
    emit error(lastErrorText(*_query));
    return false;
}

QSqlQueryModel *CDatabaseControl::lastQueryModel()
{
    return _queryModel;
}

QSqlQuery *CDatabaseControl::lastQuery()
{
    return _query;
}

QMap<int, QStringList> CDatabaseControl::getTypeOfDefinition()
{
    return this->_typeOfDefenition;
}

void CDatabaseControl::changeEngine(QString tableName, QString engine)
{
    if(_workQuery.exec("ALTER TABLE " + tableName + " ENGINE=" + engine))
    {
        emit output("ALTER TABLE " + tableName + " ENGINE=" + engine);
        emit successfully("Engine changes to " + engine);
        return;
    }
    emit error(lastErrorText(_workQuery));
}

QSqlQueryModel *CDatabaseControl::execTemporaryQuery(QString query)
{
    if(_workQuery.exec(query))
    {
        emit output(query);
        QSqlQueryModel * model = new QSqlQueryModel;
        model->setQuery(query);
        return model;
    }
    emit error(lastErrorText(_workQuery));
    return nullptr;
}
```

Fmainwindow.cpp

```
#include "console.h"
#include <QDebug>
```

```
#include <QApplication>
#include <QTextBlock>
#include <QClipboard>
#include <QScrollBar>

Console::Console(QWidget *parent) :
    QPlainTextEdit(parent)
{
    this->promt = "MySQL> ";
    this->amountBlock = 1;
    textCursor().insertText(promt);
    this->lock = false;
}

Console::~Console()
{
    qDebug() << "Destroy console";
}

void Console::setPromt(QString promt)
{
    this->promt = promt;
}

void Console::insertPromt()
{
    textCursor().insertBlock();
    textCursor().insertText(promt);
    scrollDown();
}

void Console::moveCursor(int n)
{
    QTextCursor cursor(textCursor());
    cursor.setPosition(n);
    setTextCursor(cursor);
}

void Console::printOutput(QString str)
{
    this->lock = false;
    textCursor().insertBlock();
    textCursor().insertText(str);
    textCursor().insertBlock();

    insertPromt();
}

void Console::changeCurrentWord(QListWidgetItem * item)
{
    QString word = item->text();
    QStringList textList = textCursor().block().text().mid(promt.size()).split(" ");
    if(textList.last().back() == ".")
    {
        textList.last() += word;
    }
    else
    {
        textList.last() = word;
    }

    QTextCursor cursor(textCursor().block());
```

```
    cursor.select(QTextCursor::BlockUnderCursor);
    cursor.removeSelectedText();
    cursor.insertText(promt);
    cursor.insertText(textList.join(" ") + " ");

    moveCursor(toPlainText().length());
}

void Console::queryEnter()
{
    this->lock = true;
    //QStringList query = toPlainText().split(promt);
    //emit queryEntered(query.last());
    QStringList query;
    QTextBlock block(textCursor().block());
    while(amountBlock > 1)
    {
        query.insert(0, block.text().right(block.text().size() - promt.size()));
        block = block.previous();
        amountBlock--;
    }
    query.insert(0, block.text().right(block.text().size() - promt.size()));
    emit queryEntered(query.join("\n"));
}

void Console::keyPressEvent(QKeyEvent * event)
{
    if(this->lock)
    {
        return;
    }

    if(textCursor().position() <
            toPlainText().length() - toPlainText().split(promt).last().length() + promt.length())
    {
        moveCursor(toPlainText().length());
    }

    if(event->key() >= 0x20 && event->key() <= 0x7e &&
            (event->modifiers() == Qt::NoModifier || event->modifiers() == Qt::ShiftModifier))
    {
        QPlainTextEdit::keyPressEvent(event);
    }
//    if(event->key() == Qt::Key_Backspace && textCursor().positionInBlock() > promt.length())
//    {
//        QPlainTextEdit::keyPressEvent(event);
//    }
    if(event->key() == Qt::Key_Backspace)
    {
        if(textCursor().positionInBlock() > promt.length())
        {
            QPlainTextEdit::keyPressEvent(event);
        }
        else
        {
            if(amountBlock > 1)
            {
                QTextCursor cur(textCursor().block());
                cur.select(QTextCursor::BlockUnderCursor);
                cur.removeSelectedText();
                amountBlock--;
            }
```

```
        }
    }
    if(event->key() == Qt::Key_Return && event->modifiers() == Qt::NoModifier)
    {
        queryEnter();
    }
    if(event->key() == Qt::Key_Return && event->modifiers() == Qt::ShiftModifier)
    {
        amountBlock++;
        textCursor().insertBlock();
        textCursor().insertText(promt);
        scrollDown();
    }
    if(event->key() == Qt::Key_C && event->modifiers() == Qt::ControlModifier)
    {
        QApplication::clipboard()->setText(textCursor().selectedText());
    }
    if(event->key() == Qt::Key_V && event->modifiers() == Qt::ControlModifier)
    {
        textCursor().insertText(QApplication::clipboard()->text());
    }
    if(event->key() == Qt::Key_Left && event->modifiers() == Qt::NoModifier
            && textCursor().positionInBlock() > promt.length())
    {
        moveCursor(textCursor().position() - 1);
    }
    if(event->key() == Qt::Key_Right && event->modifiers() == Qt::NoModifier
            && textCursor().position() < toPlainText().length())
    {
        moveCursor(textCursor().position() + 1);
    }
    if(event->key() == Qt::Key_Tab)
    {
        emit tabPress();
    }
}

void Console::mousePressEvent(QMouseEvent * event)
{
    if(event->button() == Qt::MouseButton::LeftButton)
    {
        if(!textCursor().selectedText().isEmpty())
        {
            QTextCursor cursor(textCursor());
            cursor.clearSelection();
            setTextCursor(cursor);
        }
        QPlainTextEdit::mousePressEvent(event);
        if(textCursor().positionInBlock() < promt.length())
        {
            moveCursor(7);
        }
    }
}

void Console::mouseDoubleClickEvent(QMouseEvent *){}

void Console::contextMenuEvent(QContextMenuEvent *){}

void Console::mouseReleaseEvent(QMouseEvent *)
{
    if(textCursor().positionInBlock() < promt.length())
```

```
    {
        QTextCursor cursor(textCursor());
        cursor.clearSelection();
        setTextCursor(cursor);
        moveCursor(7);
    }
}

void Console::focusOutEvent(QFocusEvent *event)
{
    emit focusOut();
    QPlainTextEdit::focusOutEvent(event);
}

void Console::scrollDown()
{
    QScrollBar *vbar = verticalScrollBar();
    vbar->setValue(vbar->maximum());
}
#include "fconnection.h"
#include "ui_fconnection.h"

fConnection::fConnection(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::fConnection)
{
    ui->setupUi(this);

    this->setWindowIcon(QIcon(":/new/fMainwindow/mainwindow/main_icon3.png"));
    this->setWindowTitle("Connection");
}

fConnection::~fConnection()
{
    delete ui;
}

void fConnection::on_clbToAdministrativeTools_clicked()
{
    system("control /name Microsoft.AdministrativeTools");
}
#include "ffunction.h"
#include "ui_ffunction.h"

#include <QDebug>

fFunction::fFunction(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::fFunction)
{
    ui->setupUi(this);

    this->setWindowIcon(QIcon(":/new/fMainwindow/mainwindow/main_icon3.png"));
    this->setWindowTitle("Function");

    lbl = new QSignalLabel(this);
    lbl->setStyleSheet("background-color: rgb(0, 0, 0, 150); color: rgb(255, 255, 255, 255);");
    lbl->setGeometry(0, 0, this->geometry().width(), this->geometry().height());
    lbl->setText("Hello, it`s help for you BITCH");
    lbl->setAlignment(Qt::AlignmentFlag::AlignCenter);
    lbl->hide();
```

```
    connect(lbl, &QSignalLabel::clicked, lbl, &QSignalLabel::hide);
    connect(ui->wHelp, &QToolButton::clicked, lbl, &QSignalLabel::show);
}

fFunction::~fFunction()
{
    delete ui;
}

void fFunction::resizeEvent(QResizeEvent *event)
{
    QWidget::resizeEvent(event);
    lbl->setGeometry(0, 0, this->geometry().width(), this->geometry().height());
}
```

Connection.cpp

```
#include <QDebug>
#include <QString>
#include <QMessageBox>
#include <QTextBlock>
#include <QInputDialog>
#include <QFileDialog>
#include <QProcess>

#include "fmainwindow.h"
#include "ui_fmainwindow.h"

fMainWindow::fMainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::fMainWindow)
{
    ui->setupUi(this);

    //Create a windows
    fconnection = new fConnection;
    ffunction = new fFunction;
    fprocedure = new fProcedure;
    fsettings = new fSettings;
    ftable = new fTable;
    ftrigger = new fTrigger;
    helper = new AutoHelper;
    fskripts = new fSkripts;

    ui->actionScripts->setIcon(QIcon(":/new/fMainwindow/mainwindow/skripts.png"));
    ui->actionDatabase->setIcon(QIcon(":/new/fMainwindow/mainwindow/create_database.png"));
    ui->actionTrigger->setIcon(QIcon(":/new/fMainwindow/mainwindow/create_trigger.png"));
    ui->actionProcedure->setIcon(QIcon(":/new/fMainwindow/mainwindow/create_procedure.png"));
    ui->actionTable->setIcon(QIcon(":/new/fMainwindow/mainwindow/create_table.png"));
    ui->actionConnection->setIcon(QIcon(":/new/fMainwindow/mainwindow/connection.png"));
    ui->actionImport->setIcon(QIcon(":/new/fMainwindow/mainwindow/import.png"));
    ui->actionExport->setIcon(QIcon(":/new/fMainwindow/mainwindow/export.png"));
    ui->actionSettings->setIcon(QIcon(":/new/fMainwindow/mainwindow/settings.png"));
    ui->actionExit->setIcon(QIcon(":/new/fMainwindow/mainwindow/exit.png"));

    ui->tabWidget->setTabIcon(0, QIcon(":/new/fMainwindow/mainwindow/console.png"));
    ui->tabWidget->setTabIcon(1, QIcon(":/new/fMainwindow/mainwindow/Output.png"));
    ui->tabWidget->setTabIcon(2, QIcon(":/new/fMainwindow/mainwindow/errors.png"));

    this->setWindowIcon(QIcon(":/new/fMainwindow/mainwindow/main_icon3.png"));
    this->setWindowTitle("GUIM :)");

    ui->wToolDatabase->setIcon(QIcon(":/new/fMainwindow/mainwindow/create_database.png"));
```

```
ui->wToolTable->setIcon(QIcon(":/new/fMainwindow/mainwindow/create_trigger.png"));
ui->wToolProcedure->setIcon(QIcon(":/new/fMainwindow/mainwindow/create_procedure.png"));
ui->wToolTrigger->setIcon(QIcon(":/new/fMainwindow/mainwindow/create_table.png"));

helper->setWordTip();
connect(ui->pteConsole, &Console::tabPress, helper, &AutoHelper::setFocusOnTips);
connect(helper, &AutoHelper::choosedItem, ui->pteConsole, &Console::changeCurrentWord);
connect(helper, &AutoHelper::itemClicked,  ui->pteConsole, &Console::changeCurrentWord);
connect(this, &fMainWindow::showTips, helper, &AutoHelper::showTips);
connect(ui->pteConsole, &Console::focusOut, this, &fMainWindow::hideHelper);

connect(this, &fMainWindow::installHelper, ftrigger, &fTrigger::setHelper);
connect(this, &fMainWindow::installHelper, fprocedure, &fProcedure::setHelper);


contentContextMenu = new QMenu;
contentContextMenu->addAction("Change");
contentContextMenu->addAction("Delete");
contentContextMenu->addAction("Reload");
connect(contentContextMenu->actions()[0], &QAction::triggered, this, &fMainWindow::contextChange);
connect(contentContextMenu->actions()[2], &QAction::triggered, this, &fMainWindow::reloadTree);
connect(contentContextMenu->actions()[1], &QAction::triggered, this, &fMainWindow::contextDelete);

//Create database control
dbControl = new CDatabaseControl;

if(!dbControl->connectToDatabase("QODBC", "MyServer", "localhost", 3306, "root", "123456789"))
{
   qDebug() << "Fuck this shit i`m out";
}
else
{
   //Create database control
   delete dbControl;
   dbControl = new CDatabaseControl;
}

//Actions connection to the call windows
connect(ui->actionConnection, &QAction::triggered, fconnection, &QWidget::show);
connect(ui->actionFunction, &QAction::triggered, ffunction, &QWidget::show);
connect(ui->actionProcedure, &QAction::triggered, fprocedure, &QWidget::show);
connect(ui->actionSettings, &QAction::triggered, fsettings, &QWidget::show);
connect(ui->actionTable, &QAction::triggered, this, &fMainWindow::callFormCreateTable);
connect(ui->actionProcedure, &QAction::triggered, this, &fMainWindow::callFormFProcedure);
connect(ui->actionTrigger, &QAction::triggered, this, &fMainWindow::callFormFTrigger);
connect(ui->actionScripts, &QAction::triggered, fskripts, &fSkripts::showForm);
connect(ui->actionDatabase, &QAction::triggered, this, &fMainWindow::createDataBase);
connect(ui->actionExport, &QAction::triggered, this, &fMainWindow::exportDB);

connect(dbControl, &CDatabaseControl::error, this, &fMainWindow::showError);
connect(dbControl, &CDatabaseControl::error, this, &fMainWindow::toError);
connect(dbControl, &CDatabaseControl::successfully, this, &fMainWindow::showMessage);
connect(fprocedure, &fProcedure::showMessage, this, &fMainWindow::showMessage);
connect(dbControl, &CDatabaseControl::error, this, &fMainWindow::toOutput);
connect(dbControl, &CDatabaseControl::output, this, &fMainWindow::toOutput);

connect(ui->pteConsole, &Console::queryEntered, this, &fMainWindow::setModel);
connect(dbControl, &CDatabaseControl::querySucces, ui->pteConsole, &Console::printOutput);

connect(this, &fMainWindow::showFormCreateTable, ftable, &fTable::showForm);
connect(this, &fMainWindow::showFormFProcedure, fprocedure, &fProcedure::showForm);
connect(this, &fMainWindow::showFormFTrigger, ftrigger, &fTrigger::showForm);
```

```
    connect(this, &fMainWindow::installDBControl, ftable, &fTable::setDBControl);
    connect(this, &fMainWindow::installDBControl, fprocedure, &fProcedure::setDBControl);
    connect(this, &fMainWindow::installDBControl, ftrigger, &fTrigger::setDBControl);
    connect(this, &fMainWindow::installDBControl, fskripts, &fSkripts::setDBControl);

    connect(ui->twContent, &QTreeWidget::customContextMenuRequested, this,
&fMainWindow::callContextMenuForTreeWidget);
    ui->twContent->setContextMenuPolicy(Qt::CustomContextMenu);

    ui->actionFunction->setVisible(false);
    reloadTree();
    emit installDBControl(dbControl);
}

fMainWindow::~fMainWindow()
{
    delete ui;
}

void fMainWindow::reloadTree()
{
    //Clear TreeWidget on the mainWindow
    ui->twContent->clear();

    for (auto dbName : dbControl->databasesList()) {
        QTreeWidgetItem * top = new QTreeWidgetItem(ui->twContent, QStringList(QString(dbName)));
        top->setData(0, 9, 1);
        top->setIcon(0, QIcon(":/new/fMainwindow/mainwindow/database.png"));

        if(dbName == dbControl->getCurrentDatabaseName())
        {
            QFont font = top->font(0);
            font.setBold(true);
            top->setFont(0, font);
            ui->twContent->setCurrentItem(top);
        }

        //Add tables
        QTreeWidgetItem * topTables = new QTreeWidgetItem(top, QStringList("Tables"));
        topTables->setIcon(0, QIcon(":/new/fMainwindow/mainwindow/table.png"));
        QMap<QString, QString> tables = dbControl->tablesList(dbName);
        for (auto str : tables.keys()) {
            QTreeWidgetItem * child = new QTreeWidgetItem(topTables,  QStringList(QString(str)));
            //child->setIcon(0, QIcon(":/new/fMainwindow/mainwindow/table.png"));
            child->setToolTip(0, QString(tables.value(str)));
            child->setData(0, 9, 2);
        }

        //Add procedures
        QTreeWidgetItem * topProcedure = new QTreeWidgetItem(top, QStringList("Procedures"));
        topProcedure->setIcon(0, QIcon(":/new/fMainwindow/mainwindow/procedure.png"));
        for (auto str : dbControl->procedureList(dbName)) {
            QTreeWidgetItem * child = new QTreeWidgetItem(topProcedure,  QStringList(QString(str)));
            child->setData(0, 9, 3);
        }

        //Add triggers
        QTreeWidgetItem * topTrigger = new QTreeWidgetItem(top, QStringList("Triggers"));
        topTrigger->setIcon(0, QIcon(":/new/fMainwindow/mainwindow/trigger.png"));
        for (auto str : dbControl->triggersList(dbName)) {
            QTreeWidgetItem * child = new QTreeWidgetItem(topTrigger,  QStringList(QString(str)));
            child->setData(0, 9, 4);
```

```
        }
    }
    QMap<QString, QStringList> map;
    for(QString table : dbControl->tablesList(dbControl->getCurrentDatabaseName()).keys())
    {
        map.insert(table, dbControl->columnList(table));
    }
    helper->setAddTip(map);
    emit installHelper(map);
}

void fMainWindow::toOutput(QString command)
{
    if(!command.isEmpty())
    {
        ui->pteOutput->insertPlainText(command + "\n\n");
    }
}

void fMainWindow::toError(QString error)
{
    if(!error.isEmpty())
    {
        ui->wErrors->insertPlainText(error + "\n\n");
    }
}

void fMainWindow::showTableInTableView(QAbstractItemModel * model)
{
    ui->tvShowTable->setModel(model);
}

void fMainWindow::callFormCreateTable()
{
    emit showFormCreateTable(dbControl->getCurrentDatabaseName(), "");
}

void fMainWindow::callFormFProcedure()
{
    emit showFormFProcedure("");
}

void fMainWindow::callFormFTrigger()
{
    emit showFormFTrigger("");
}

void fMainWindow::setModel(QString query)
{
    if(dbControl->execQueryModel(query))
    {
        ui->tvShowTable->setModel(dbControl->lastQueryModel());
    }
}

void fMainWindow::showMessage(QString str)
{
    QMessageBox message;
    message.setText(str);
    message.setIcon(QMessageBox::Icon::NoIcon);
    message.setWindowTitle("Message");
    message.exec();
```

```
}

void fMainWindow::hideHelper()
{
    if(this->focusWidget() != ui->pteConsole)
    {
        helper->hide();
    }
}

void fMainWindow::createDataBase()
{
    QString name = QInputDialog::getText(this, "Name", "Enter the database name");
    if(!name.isEmpty())
    {
        dbControl->execQuery("CREATE DATABASE " + name);
    }
}

void fMainWindow::exportDB()
{
    QString path = QFileDialog::getSaveFileName() + ".sql";
    qDebug() << path;

    if(!path.isEmpty())
    {
        //QString commandFake = "\"D:/Program Files/MySQL/MySQL Server 8.0/bin/mysqldump.exe\" -uroot -
p123456789 testapp > D:\\skripts\\sd.sql";
        QString command = "\"D:/Program Files/MySQL/MySQL Server 8.0/bin/mysqldump.exe\" -uroot -
p123456789 " + dbControl->getCurrentDatabaseName() + " > " + path;
        qDebug() << command;
        //QProcess::execute(commandFake);
        system(qPrintable(command));
    }
}

void fMainWindow::callContextMenuForTreeWidget(const QPoint &point)
{
    if(ui->twContent->currentItem()->data(0, 1).toInt() > 0)
        contentContextMenu->exec(ui->twContent->viewport()->mapToGlobal(point));
}

void fMainWindow::contextChange()
{
    switch(ui->twContent->currentItem()->data(0, 1).toInt())
    {
    case 1:{
        qDebug() << "1" << ui->twContent->currentItem()->text(0);
        break;
    }
    case 2:{
        qDebug() << "2" << ui->twContent->currentItem()->text(0);
        showFormCreateTable(dbControl->getCurrentDatabaseName(), ui->twContent->currentItem()->text(0));
        break;
    }
    case 3:{
        qDebug() << "3" <<ui->twContent->currentItem()->text(0);
        break;
    }
    case 4:{
        qDebug() << "4" <<ui->twContent->currentItem()->text(0);
        break;
```

```
    }
    }
}

void fMainWindow::contextDelete()
{
    switch(ui->twContent->currentItem()->data(0, 1).toInt())
    {
    case 1:{
        qDebug() << "1" << ui->twContent->currentItem()->text(0);
        dbControl->execQuery("DROP DATABASE " + ui->twContent->currentItem()->text(0));
        break;
    }
    case 2:{
        qDebug() << "2" << ui->twContent->currentItem()->text(0);
        showFormCreateTable(dbControl->getCurrentDatabaseName(), ui->twContent->currentItem()->text(0));
        break;
    }
    case 3:{
        qDebug() << "3" <<ui->twContent->currentItem()->text(0);
        break;
    }
    case 4:{
        qDebug() << "4" <<ui->twContent->currentItem()->text(0);
        break;
    }
    }
}

void fMainWindow::on_twContent_itemDoubleClicked(QTreeWidgetItem *item, int column)
{
    qDebug() << item->data(0, 9);
    switch(item->data(0, 9).toInt())
    {
    case 1:{
        dbControl->changeWorkDatabase(item->text(column));
        reloadTree();
        break;
    }
    case 2:{
        showTableInTableView(dbControl->getTable(item->parent()->parent()->text(0), item->text(column)));
        break;
    }
    case 3:{
        emit showFormFProcedure(item->text(column));
        break;
    }
    case 4:{
        emit showFormFTrigger(item->text(column));
        break;
    }
    }

}

void fMainWindow::showError(QString text)
{
    QMessageBox message;
    message.setText(text);
    message.setIcon(QMessageBox::Icon::Warning);
    message.setWindowTitle("Error");
    message.exec();
```

```
}

void fMainWindow::on_pteConsole_textChanged()
{
   QRect rc = ui->pteConsole->cursorRect();
   helper->setGeometry(ui->pteConsole->viewport()->mapToGlobal(QPoint(rc.x(), rc.y())).x(),
                       ui->pteConsole->viewport()->mapToGlobal(QPoint(rc.x(), rc.y())).y() + rc.height(),
                       150, 75);
   emit showTips(ui->pteConsole->textCursor().block().text().split(" ").last());
   ui->pteConsole->activateWindow();
}

void fMainWindow::moveEvent(QMoveEvent *event)
{
   helper->hide();
   QMainWindow::moveEvent(event);
}

void fMainWindow::resizeEvent(QResizeEvent *event)
{
   helper->hide();
   QMainWindow::resizeEvent(event);
}

void fMainWindow::closeEvent(QCloseEvent *event)
{
   helper->hide();
   QMainWindow::closeEvent(event);
}

void fMainWindow::hideEvent(QHideEvent *event)
{
   helper->hide();
   QMainWindow::hideEvent(event);
}
#include "fprocedure.h"
#include "ui_fprocedure.h"

#include <QTextBlock>
#include <QDebug>

fProcedure::fProcedure(QWidget *parent) :
   QWidget(parent),
   ui(new Ui::fProcedure)
{
   ui->setupUi(this);

   this->setWindowIcon(QIcon(":/new/fMainwindow/mainwindow/main_icon3.png"));
   this->setWindowTitle("Procedure");

   filter = new tabFilter();
   ui->wEdit->installEventFilter(filter);

   this->helper = new AutoHelper;
   helper->setWordTip();
   connect(filter, &tabFilter::tabPress, helper, &AutoHelper::setFocusOnTips);
   connect(helper, &AutoHelper::choosedItem, this, &fProcedure::changeCurrentWord);
   connect(helper, &AutoHelper::itemClicked,  this, &fProcedure::changeCurrentWord);
   connect(this, &fProcedure::showTips, helper, &AutoHelper::showTips);
}

fProcedure::~fProcedure()
```

```
{
   delete ui;
}

void fProcedure::changeCurrentWord(QListWidgetItem * item)
{
   QStringList text = ui->wEdit->toPlainText().split(" ");
   if(text.last().back() == '.')
   {
      ui->wEdit->setPlainText(text.join(" ") +  item->text() + " ");
   }
   else
   {
      if(text.isEmpty())
      {
         ui->wEdit->setPlainText(item->text() + " ");
      }
      else
      {
         text.pop_back();
         text.push_back(item->text());
         ui->wEdit->setPlainText(text.join(" ") +  " ");
      }
   }
   QTextCursor cur(ui->wEdit->textCursor());
   cur.setPosition(ui->wEdit->toPlainText().length());
   ui->wEdit->setTextCursor(cur);
}

void fProcedure::on_wEdit_textChanged()
{
   QRect rc = ui->wEdit->cursorRect();
   helper->setGeometry(ui->wEdit->viewport()->mapToGlobal(QPoint(rc.x(), rc.y())).x(),
                       ui->wEdit->viewport()->mapToGlobal(QPoint(rc.x(), rc.y())).y() + rc.height(),
                       150, 75);
   emit showTips(ui->wEdit->textCursor().block().text().split(" ").last());
   ui->wEdit->activateWindow();
}

void fProcedure::setHelper(QMap<QString, QStringList> map)
{
   helper->setAddTip(map);
}

void fProcedure::showForm(QString procName)
{
   if(procName.isEmpty())
   {
      this->procName = "";
   }
   else
   {
      this->procName = procName;
      reload();
   }
   show();
}

void fProcedure::setDBControl(CDatabaseControl * dbcontrol)
{
   this->dbcontrol = dbcontrol;
}
```

```
void fProcedure::reload()
{
   if(!procName.isEmpty() && dbcontrol->execQuery("show create procedure " + procName))
   {
      QString procDefinition;
      while(dbcontrol->lastQuery()->next())
      {
         procDefinition = dbcontrol->lastQuery()->value(2).toString();
      }
      procDefinition = procDefinition.mid(procDefinition.indexOf("PROCEDURE"));
      ui->wParametrs->clear();
      QString param = procDefinition.mid(procDefinition.indexOf("(") + 1, procDefinition.indexOf(")") -
procDefinition.indexOf("(") - 1);
      if(param.size() > 3)
      {
         if(param.indexOf(",") > 0)
         {
            QStringList list = param.split(",");
            ui->wParametrs->insertItems(0, list);
         }
         else
         {
            ui->wParametrs->insertItem(0, param);
         }
      }
      ui->wName->clear();
      ui->wName->setText(procName);
      ui->wEdit->clear();
      if(procDefinition.contains("COMMENT '"))
      {
         ui->wComment->clear();
         ui->wComment->setText(procDefinition.mid(procDefinition.indexOf("COMMENT '") +
9).split("'")[0]);
         ui->wEdit->setPlainText(procDefinition.mid(procDefinition.indexOf("COMMENT '") +
9).split("'")[1]);
      }
      else
      {
         ui->wEdit->setPlainText(procDefinition.mid(procDefinition.indexOf(")") + 1));
      }

   }
}

QString fProcedure::collect()
{
   QString query = "CREATE PROCEDURE ";
   query += ui->wName->text();
   query += " (";
   for(int i = 0; i < ui->wParametrs->count(); i++)
   {
      query += ui->wParametrs->itemText(i);
   }
   query += ")\n";
   query += "COMMENT '" + ui->wComment->text() + "'\n";
   query += ui->wEdit->toPlainText();
   return query;
}

void fProcedure::moveEvent(QMoveEvent *event)
{
```

```
    helper->hide();
    QWidget::moveEvent(event);
}

void fProcedure::resizeEvent(QResizeEvent *event)
{
    helper->hide();
    QWidget::resizeEvent(event);
}

void fProcedure::closeEvent(QCloseEvent *event)
{
    helper->hide();
    QWidget::closeEvent(event);
}

void fProcedure::hideEvent(QHideEvent *event)
{
    helper->hide();
    QWidget::hideEvent(event);
}

void fProcedure::on_wDelete_clicked()
{
    ui->wParametrs->removeItem(ui->wParametrs->currentIndex());
}

void fProcedure::on_wSave_clicked()
{
    if(procName.isEmpty())
    {
        if(dbcontrol->execQuery(collect()))
        {
            procName = ui->wName->text();
            reload();
        }
    }
    else
    {
        if(dbcontrol->execQuery("show create procedure " + procName))
        {
            QString oldDefinition;
            while(dbcontrol->lastQuery()->next())
            {
                oldDefinition = dbcontrol->lastQuery()->value(2).toString();
            }

            if(dbcontrol->execQuery("DROP PROCEDURE " + procName))
            {
                if(dbcontrol->execQuery(collect()))
                {
                    procName = ui->wName->text();
                    reload();
                }
                else
                {
                    dbcontrol->execQuery(oldDefinition);
                }
            }
        }
    }
}
```

Fsettings.cpp

```
#include "fsettings.h"
#include "ui_fsettings.h"
#include <QDebug>

fSettings::fSettings(QWidget *parent) :
   QWidget(parent),
   ui(new Ui::fSettings)
{
   ui->setupUi(this);

   this->setWindowIcon(QIcon(":/new/fMainwindow/mainwindow/main_icon3.png"));
   this->setWindowTitle("Settings");

   ui->wLang->insertItem(0, "English");
   ui->wLang->insertItem(0, "Українська");
}

fSettings::~fSettings()
{
   delete ui;
}

void fSettings::on_wLang_currentTextChanged(const QString &arg1)
{
   qDebug() << "lol";
   if(arg1 == "English")
   {
      qDebug() << translator.load("QLang_eng", ".");
      qApp->installTranslator(&translator);
   }
   else
   {
      qDebug() << translator.load("QLang_ukr", ".");
      qApp->installTranslator(&translator);
   }
}

void fSettings::changeEvent(QEvent * pe)
{
   qDebug() << "lol2";
   if(pe->type() == QEvent::LanguageChange)
   {
      ui->retranslateUi(this);
   }
   QWidget::changeEvent(pe);
}
```

Fskripts.cpp

```
#include "fskripts.h"
#include "ui_fskripts.h"
#include <QFileDialog>
#include <QDebug>
#include <QMessageBox>
#include <QInputDialog>

fSkripts::fSkripts(QWidget *parent) :
   QWidget(parent),
   ui(new Ui::fSkripts)
{
```

```
    ui->setupUi(this);

    this->setWindowIcon(QIcon(":/new/fMainwindow/mainwindow/main_icon3.png"));
    this->setWindowTitle("Skripts");

    dirmodel = new QFileSystemModel(this);
    dirmodel->setRootPath("/home");
    ui->wTree->setModel(dirmodel);
    editPath = "";
}

fSkripts::~fSkripts()
{
    delete ui;
}

void fSkripts::showForm()
{
    show();
}

void fSkripts::on_wFolder_clicked()
{
    QString path = QFileDialog::getExistingDirectory(this, "Open directory", "/home");
    ui->wTree->setRootIndex(dirmodel->index(path));
}

void fSkripts::on_wTree_clicked(const QModelIndex &index)
{
    if(dirmodel->type(index).split(" ")[0] == "sql")
    {
        QFile file(dirmodel->filePath(index));
        if(file.open(QFile::OpenModeFlag::ReadOnly))
        {
            ui->wEdit->clear();
            QTextStream textStream(&file);
            while (true)
            {
                QString line = textStream.readLine();
                if (line.isNull())
                {
                    break;
                }
                else
                {
                    ui->wEdit->insertPlainText(line + "\n");
                }
            }
            editPath = dirmodel->filePath(index);
            qDebug() << editPath;
        }
        else
        {
            //Вывод ошибки
        }

    }
}

void fSkripts::on_wDelete_clicked()
{
    QModelIndex index = ui->wTree->currentIndex();
```

```
        if(index.isValid())
        {
          int answer  = QMessageBox::warning(this, "Delete", "do you really want to delete " + dirmodel->fileName(index), "Yes", "No", QString(), 1, 1);
          if(answer == 0)
          {
            if(dirmodel->remove(index))
            {
              QMessageBox::information(this, "Successful", "File removed");
            }
            else
            {
              QMessageBox::information(this, "Error", "File not removed");
            }
          }
        }
      }

      void fSkripts::on_wNewFile_clicked()
      {
        int answer = QMessageBox::information(this, "Create", "Do you want to create file or directory?", "File", "Directory", QString(), 0, 2);
        switch (answer) {
        case 0: {
          QString name = QInputDialog::getText(this, "Name", "Enter the name");
          if(!name.isEmpty())
          {
            //создание файла
          }
          break;
        }
        case 1: {
          QString name = QInputDialog::getText(this, "Name", "Enter the name");
          if(!name.isEmpty())
          {
            dirmodel->mkdir(ui->wTree->currentIndex(), name);
          }
        }
        }
      }

      void fSkripts::on_wSave_clicked()
      {
        //Сделать сохранение
      }

      void fSkripts::on_wExecute_clicked()
      {
        qDebug() << editPath;
        QFile file(editPath);
        if (!file.open(QIODevice::ReadOnly | QIODevice::Text))

        {
          QMessageBox::warning(this, "Error", "File not opened");
          return;
        }

        QTextStream in(&file);
        QString sql = in.readAll();
        QStringList sqlStatements = sql.split(';', QString::SkipEmptyParts);

        foreach(const QString& statement, sqlStatements)
```

```
    {
        if (statement.trimmed() != "")
        {
            dbcontrol->execQuery(statement);
        }
    }
    QMessageBox::information(this, "Done", "Successful");
}

void fSkripts::setDBControl(CDatabaseControl *dbcontrol)
{
    this->dbcontrol = dbcontrol;
}
#include "ftable.h"
#include "ui_ftable.h"

fTable::fTable(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::fTable)
{
    ui->setupUi(this);

    this->setWindowIcon(QIcon(":/new/fMainwindow/mainwindow/main_icon3.png"));
    this->setWindowTitle("Table");

    ui->wTable->setContextMenuPolicy(Qt::CustomContextMenu);
    menu = new QMenu;
    menu->addAction(QIcon(":/new/fTable/res/trash_backet.png"), "Delete");
    menu->addAction(QIcon(":/new/fTable/res/White_Arrow_Up.png"), "Up");
    menu->addAction(QIcon(":/new/fTable/res/White_Arrow_Down.png"), "Down");

    connect(menu->actions()[0], &QAction::triggered, this, &fTable::deleteColumn);
    connect(menu->actions()[1], &QAction::triggered, this, &fTable::columnUp);
    connect(menu->actions()[2], &QAction::triggered, this, &fTable::columnDown);

    connect(ui->wClear, &QPushButton::clicked, this, &fTable::toDefaultColumn);
    ui->groupBox_8->hide();
    showEngines = new QTableView;
}

fTable::~fTable()
{
    delete ui;
}

void fTable::showForm(QString dbName, QString tableName)
{
    toDefaultView();
    this->forDatabase = dbName;
    if(tableName.isEmpty())
    {
        this->currentTable = "";
    }
    else
    {
        this->currentTable = tableName;
        ui->wTableName->setText(currentTable);
        showTable();
    }
    show();
}
```

```
void fTable::setDBControl(CDatabaseControl *dbcontrol)
{
   this->dbcontrol = dbcontrol;
}

void fTable::toDefaultView()
{
   toDefaultFields();
   toDefaultConstraint();
   toDefaultTools();
   toDefaultCode();
   ui->wEngine->clear();
   dbcontrol->execQuery("show engines");
   while(dbcontrol->lastQuery()->next())
   {
      ui->wEngine->insertItem(ui->wEngine->count(), dbcontrol->lastQuery()->value(0).toString());
   }
   ui->wTabs->setCurrentIndex(0);
}

void fTable::toDefaultFields()
{
   ui->wType->clear();
   for (QStringList lst : dbcontrol->getTypeOfDefinition().values()) {
      ui->wType->insertItems(0, lst);
   }
   ui->wTable->setModel(nullptr);
   ui->wTableName->clear();
   toDefaultColumn();
}

void fTable::toDefaultConstraint()
{
   ui->wConstrainfList->clear();
   ui->wCheck->clear();
   ui->wOnDelete->clear();
   ui->wOnUpdate->clear();
   ui->wOnDelete->insertItems(0, {"cascade", "set null", "restrict", "no action", "set default"});
   ui->wOnUpdate->insertItems(0, {"cascade", "set null", "restrict", "no action", "set default"});
   if(ui->wTable->model() != nullptr)
   {
      ui->wReferencesTable->clear();
      ui->wReferencesTable->insertItems(0,                    dbcontrol->tablesList(dbcontrol-
>getCurrentDatabaseName()).keys());
   }
}

void fTable::toDefaultTools()
{
   ui->wCommentTable->clear();
   ui->wCloneTable->clear();
   ui->wWithDataTemporary->setCheckState(Qt::Unchecked);
   ui->wWithDataClone->setCheckState(Qt::Unchecked);
   QStringList defList = ui->wAdvancedEdit->toPlainText().split("\n");
   if(defList[defList.size() - 1].contains("COMMENT", Qt::CaseInsensitive))
   {
      ui->wCommentTable->setPlainText(defList[defList.size() - 1].split("'")[1]);
   }
   if(!ui->wAdvancedEdit->toPlainText().isEmpty())
   {
      ui->wEngine->setCurrentText(defList[defList.size() - 1].split("ENGINE=")[1].split(" ")[0]);
   }
```

```
    ui->wCloneTable->insertItems(0, dbcontrol->tablesList(dbcontrol->getCurrentDatabaseName()).keys());
}

void fTable::toDefaultCode()
{
    ui->wAdvancedEdit->clear();
}

void fTable::setShowedWidgets(int modificator)
{
    switch (modificator) {
    case 1:{
        installShowedWidgets(true, true, false, true, true);
        break;
    }
    case 2:{
        installShowedWidgets(false, true, false, false, false);
        break;
    }
    case 3:{
        installShowedWidgets(true, true, true, true, true);
        break;
    }
    default:{
        installShowedWidgets(false, false, false, false, false);
    }
    }
}

void fTable::installShowedWidgets(bool unsign, bool firstParamtrs, bool secondParametrs, bool zerofill, bool
increment)
{
    ui->wUnsigned->setVisible(unsign);
    ui->wUnsigned->setChecked(false);
    ui->wTypeParametr1->setVisible(firstParamtrs);
    ui->wTypeParametr1->setValue(1);
    ui->wTypeParametr2->setVisible(secondParametrs);
    ui->wTypeParametr2->setValue(1);
    ui->wIncrement->setVisible(increment);
    ui->wIncrement->setChecked(false);
    ui->wZerofill->setVisible(zerofill);
    ui->wZerofill->setChecked(false);
}

void fTable::toDefaultColumn()
{
    ui->wKeyNone->setChecked(true);
    ui->wColumnName->clear();
    ui->wDefault->clear();
    ui->wNotNull->setChecked(false);
    ui->wCommentColumn->clear();
}

void fTable::columnUp()
{
    if(ui->wTable->currentIndex().row() > 0){
        if(dbcontrol->execQuery("show create table " + currentTable)){
            QString columnName = ui->wTable->model()->index(ui->wTable->currentIndex().row(),
0).data().toString();
            QStringList definition;
            while (dbcontrol->lastQuery()->next()) {
                definition = dbcontrol->lastQuery()->value(1).toString().split("\n");
```

```
                }
                QString columnDefinition;
                for (QString str : definition) {
                    if(str.indexOf("`"+ columnName +"`") < 4 && str.indexOf("`"+ columnName +"`") >= 0){
                        str.remove(str.size() - 1, 1);
                        columnDefinition = str;
                        break;
                    }
                }
                if (ui->wTable->currentIndex().row() == 1) {
                    dbcontrol->execQuery("ALTER TABLE " + currentTable + " MODIFY COLUMN "
                            + columnDefinition + " FIRST");
                }
                else {
                    dbcontrol->execQuery("ALTER TABLE " + currentTable + " MODIFY COLUMN "
                            + columnDefinition + " AFTER " + ui->wTable->model()->index(ui->wTable-
>currentIndex().row() - 2, 0).data().toString());
                }
                showTable();
            }
        }
    }

    void fTable::columnDown()
    {
        if(ui->wTable->currentIndex().row() < ui->wTable->model()->rowCount() - 1){
            if(dbcontrol->execQuery("show create table " + currentTable)){
                QString columnName = ui->wTable->model()->index(ui->wTable->currentIndex().row(),
0).data().toString();
                QStringList definition;
                while (dbcontrol->lastQuery()->next()) {
                    definition = dbcontrol->lastQuery()->value(1).toString().split("\n");
                }
                QString columnDefinition;
                for (QString str : definition) {
                    if(str.indexOf("`"+ columnName +"`") < 4 && str.indexOf("`"+ columnName +"`") >= 0){
                        str.remove(str.size() - 1, 1);
                        columnDefinition = str;
                        break;
                    }
                }

                dbcontrol->execQuery("ALTER TABLE " + currentTable + " MODIFY COLUMN "
                        + columnDefinition + " AFTER " + ui->wTable->model()->index(ui->wTable-
>currentIndex().row() + 1, 0).data().toString());
                showTable();
            }
        }
    }

    void fTable::deleteColumn()
    {
        QString columnName = ui->wTable->model()->index(ui->wTable->currentIndex().row(),
0).data().toString();
        dbcontrol->execQuery("ALTER TABLE " + currentTable + " DROP " + columnName);
        showTable();
    }

    void fTable::showTable()
    {
        if(!currentTable.isEmpty())
        {
```

```
        ui->wTableName->setText(currentTable);
        ui->wTable->setModel(dbcontrol->execTemporaryQuery("describe " + currentTable));
        if(dbcontrol->execQuery("show create table " + currentTable)){
            while (dbcontrol->lastQuery()->next()) {
                ui->wAdvancedEdit->setPlainText(dbcontrol->lastQuery()->value(1).toString());
            }
        }
        else
        {
            ui->wAdvancedEdit->setPlainText("Error");
        }
    }
}

void fTable::on_wType_currentTextChanged(const QString &arg1)
{
    int i = 1;
    for (QStringList lst : dbcontrol->getTypeOfDefinition().values()) {
        if(lst.contains(arg1))
        {
            setShowedWidgets(i);
            return;
        }
        ++i;
    }
}

void fTable::on_wTable_customContextMenuRequested(const QPoint &pos)
{
    if(ui->wTable->currentIndex().row() > -1)
    {
        menu->exec(ui->wTable->viewport()->mapToGlobal(pos));
    }
}

void fTable::on_wClear_clicked()
{
    toDefaultColumn();
}

void fTable::on_wTable_clicked(const QModelIndex &index)
{
    if(dbcontrol->execQuery("show create table " + currentTable)){
        while (dbcontrol->lastQuery()->next()) {

            QStringList definition = dbcontrol->lastQuery()->value(1).toString().split("\n");
            QString columnName = ui->wTable->model()->index(index.row(), 0).data().toString();

            toDefaultColumn();

            ui->wColumnName->setText(columnName);
            ui->wUnsigned->setChecked(isHave(definition, columnName, "unsigned"));
            ui->wIncrement->setChecked(isHave(definition, columnName, "auto_increment"));
            ui->wNotNull->setChecked(isHave(definition, columnName, "not null"));
            ui->wType->setCurrentText(isType(definition, columnName));
            if(ui->wTypeParametr1->isVisible()){
                ui->wTypeParametr1->setValue(isParametr1(definition, columnName));
            }
            if(ui->wTypeParametr2->isVisible()){
                ui->wTypeParametr2->setValue(isParametr2(definition, columnName));
            }
            ui->wDefault->setText(isDefault(definition, columnName));
```

```
                ui->wZerofill->setChecked(isHave(definition, columnName, "zerofill"));
                ui->wKeyUnique->setChecked(isHave(definition, columnName, "unique key"));
                ui->wKeyPrimary->setChecked(isHave(definition, columnName, "primary key"));
                ui->wCommentColumn->setPlainText(isCommentColumn(definition, columnName));
            }
        }
    }

    QString fTable::isCommentColumn(QStringList definition, QString columnName)
    {
        int i = 0;
        while(i < definition.size())
        {
            if(definition[i].contains("`" + columnName + "`", Qt::CaseInsensitive) &&
definition[i].contains("COMMENT", Qt::CaseInsensitive))
            {
                QString temp = "COMMENT '";
                return definition[i].mid(definition[i].indexOf(temp) + temp.size(),
                                         definition[i].lastIndexOf("'") - definition[i].indexOf(temp) - temp.size());
            }
            else
            {
              i++;
            }
        }
        return "";
    }

    QString fTable::isType(QStringList definition, QString columnName)
    {
        int i = 0;
        while(i < definition.size())
        {
            if(definition[i].indexOf("`" + columnName + "`", Qt::CaseInsensitive) >= 0 &&
                    definition[i].indexOf("`" + columnName + "`", Qt::CaseInsensitive) < 5)
            {
                QString type = definition[i].split(" ")[3].mid(0, definition[i].split(" ")[3].indexOf("("));
                return type.toUpper();
            }
            else
            {
              i++;
            }
        }
        return "ERROR";
    }

    int fTable::isParametr1(QStringList definition, QString columnName)
    {
        int i = 0;
        while(i < definition.size())
        {
            if(definition[i].indexOf("`" + columnName + "`", Qt::CaseInsensitive) >= 0 &&
                    definition[i].indexOf("`" + columnName + "`", Qt::CaseInsensitive) < 5)
            {
                if(definition[i].split(" ")[3].contains("(")){
                    if(definition[i].split(" ")[3].contains(",")){
                        return definition[i].split(" ")[3].mid(definition[i].split(" ")[3].indexOf("(") + 1,
                                definition[i].split(" ")[3].indexOf(",") - definition[i].split(" ")[3].indexOf("(") - 1).toInt();
                    }
                    else {
                        return definition[i].split(" ")[3].mid(definition[i].split(" ")[3].indexOf("(") + 1,
```

```
                        definition[i].split(" ")[3].indexOf(")") - definition[i].split(" ")[3].indexOf("(") - 1).toInt();
                }
              }
            }
            else
            {
              i++;
            }
          }
          return 1;
        }

        int fTable::isParametr2(QStringList definition, QString columnName)
        {
          int i = 0;
          while(i < definition.size())
          {
            if(definition[i].indexOf("`" + columnName + "`", Qt::CaseInsensitive) >= 0 &&
                definition[i].indexOf("`" + columnName + "`", Qt::CaseInsensitive) < 5)
            {
              if(definition[i].split(" ")[3].contains("(")){
                if(definition[i].split(" ")[3].contains(",")){
                  return definition[i].split(" ")[3].mid(definition[i].split(" ")[3].indexOf(",") + 1,
                      definition[i].split(" ")[3].indexOf(")") - definition[i].split(" ")[3].indexOf(",") - 1).toInt();
                }
                else {
                  return 1;
                }
              }
            }
            else
            {
              i++;
            }
          }
          return 1;
        }

        QString fTable::isDefault(QStringList definition, QString columnName)
        {
          int i = 0;
          qDebug() << "isDefault";
          while(i < definition.size())
          {
            if(definition[i].contains("`" + columnName + "`", Qt::CaseInsensitive) &&
                definition[i].contains("default", Qt::CaseInsensitive))
            {
              qDebug() << "Seeking " + definition[i];
              qDebug() << "Seek: " << definition[i].split(" ");
              return definition[i].split("  ")[definition[i].split("  ").indexOf("DEFAULT", Qt::CaseInsensitive) +
1].replace(QRegExp("[`,]"), "");
            }
            else
            {
              i++;
            }
          }
          return "";
        }

        bool fTable::isHave(QStringList definition, QString columnName, QString parametr)
        {
```

```
    int i = 0;
    while(i < definition.size())
    {
      if(definition[i].contains("`" + columnName + "`", Qt::CaseInsensitive) &&
definition[i].contains(parametr, Qt::CaseInsensitive))
      {
        return true;
      }
      else
      {
        i++;
      }
    }
    return false;
}

void fTable::on_wReferencesTable_currentTextChanged(const QString &arg1)
{
    ui->wReferencesColumn->clear();
    if(!arg1.isEmpty())
    {
        ui->wReferencesColumn->insertItems(0, dbcontrol->columnList(ui->wReferencesTable->currentText()));
    }
}

QString fTable::collectCommand()
{
    QString command;
    command = ui->wColumnName->text() + " ";
    command += ui->wType->currentText();
    if(ui->wTypeParametr1->isVisible())
    {
        command += "(" + QString::number(ui->wTypeParametr1->value());
    }
    if(ui->wTypeParametr2->isVisible())
    {
        command += ", " + QString::number(ui->wTypeParametr2->value());
    }
    command += ") ";
    if(ui->wUnsigned->isVisible() && ui->wUnsigned->isChecked())
    {
        command += "UNSIGNED ";
    }
    if(ui->wZerofill->isVisible() && ui->wZerofill->isChecked())
    {
        command += "ZEROFILL ";
    }
    if(ui->wIncrement->isVisible() && ui->wIncrement->isChecked())
    {
        command += "AUTO_INCREMENT ";
    }
    if(ui->wNotNull->isChecked())
    {
        command += "NOT NULL ";
    }
    if(!ui->wDefault->text().isEmpty())
    {
        command += "DEFAULT (" + ui->wDefault->text() + ") ";
    }
    if(!ui->wCommentColumn->toPlainText().isEmpty())
    {
        command += "COMMENT '" + ui->wCommentColumn->toPlainText() + "'";
```

```
    }
    return command;
}

void fTable::showConstraint()
{
    ui->wConstrainfList->clear();
    QStringList definition = ui->wAdvancedEdit->toPlainText().split("\n");
    for(int i = 0; i < definition.size(); i++)
    {
        if(definition[i].contains("CONSTRAINT", Qt::CaseSensitive))
        {
            ui->wConstrainfList->insertItem( ui->wConstrainfList->count(), definition[i]);
        }
    }
    if(ui->wTable->model() != nullptr)
    {
        ui->wColumn->clear();
        ui->wReferencesTable->clear();
        for(int i = 0; i < ui->wTable->model()->rowCount(); i++)
        {
            ui->wColumn->insertItem(i, ui->wTable->model()->index(i, 0).data().toString());
        }
        ui->wReferencesTable->insertItems(0, dbcontrol->tablesList(dbcontrol->getCurrentDatabaseName()).keys());
    }
}

void fTable::on_wAdd_clicked()
{
    QString columnName = ui->wColumnName->text();
    if(!columnName.isEmpty())
    {
      QString command;
      if(dbcontrol->columnList(this->currentTable).contains(columnName.toLower()))
      {
        command = "ALTER TABLE " + this->currentTable + " MODIFY ";
      }
      else
      {
        command = "ALTER TABLE " + this->currentTable + " ADD ";
      }
      command += collectCommand();
      dbcontrol->execQuery(command);

      if(ui->wKeyNone->isChecked())
      {
        ;;;
      }
      else if(ui->wKeyUnique->isChecked())
      {
        if(!isHave(ui->wAdvancedEdit->toPlainText().split("\n"), columnName, "UNIQUE"))
        {
          dbcontrol->execQuery("ALTER TABLE " + this->currentTable + " ADD UNIQUE (" + columnName + ")");
        }
      }
      else if(ui->wKeyPrimary->isChecked())
      {
        if(!isHave(ui->wAdvancedEdit->toPlainText().split("\n"), columnName, "PRIMARY"))
        {
          dbcontrol->execQuery("ALTER TABLE " + this->currentTable + " ADD PRIMARY KEY (" +
```

```
columnName + ")");
                }
            }
        }
        else
        {
            qDebug() << "empty";
        }
        showTable();
    }

    void fTable::on_wTabs_currentChanged(int index)
    {
        switch(index)
        {
        case 0:
        {
            toDefaultFields();
            showTable();
            break;
        }
        case 1:
        {
            toDefaultConstraint();
            showConstraint();
            break;
        }
        case 2:
        {
            toDefaultTools();
            break;
        }
        case 3:
        {
            break;
        }
        }
    }

    void fTable::on_wConstrainfList_itemDoubleClicked(QListWidgetItem *item)
    {
        QString constraintText = item->text();
        if(constraintText.contains("FOREIGN KEY", Qt::CaseSensitive))
        {
            dbcontrol->execQuery("ALTER TABLE " + this->currentTable + " DROP FOREIGN KEY " +
constraintText.split("`")[1]);
        }
        else
        {
            dbcontrol->execQuery("ALTER TABLE " + this->currentTable + " DROP CHECK " +
constraintText.split("`")[1]);
        }
        showTable();
        showConstraint();
    }

    void fTable::on_pushButton_clicked()
    {
        if(!ui->wCheck->toPlainText().isEmpty())
        {
            dbcontrol->execQuery("ALTER TABLE " + this->currentTable + " ADD CHECK (" + ui->wCheck-
>toPlainText() + ")");
```

```
            ui->wCheck->clear();
            showTable();
            showConstraint();
        }
    }

    void fTable::on_pushButton_2_clicked()
    {
        if(!ui->wColumn->currentText().isEmpty())
        {
            dbcontrol->execQuery("ALTER TABLE " + this->currentTable + " ADD FOREIGN KEY (" + ui->wColumn->currentText() +
                                 ") REFERENCES " + ui->wReferencesTable->currentText() + " (" + ui->wReferencesColumn->currentText() + ") " +
                                 "ON DELETE " + ui->wOnDelete->currentText() + " ON UPDATE " + ui->wOnUpdate->currentText());
            showTable();
            showConstraint();
        }
    }

    void fTable::on_wHelpEngines_clicked()
    {
        dbcontrol->execQueryModel("show engines");
        showEngines->setModel(dbcontrol->lastQueryModel());
        showEngines->resizeColumnsToContents();
        showEngines->resize(725, 310);
        showEngines->show();
    }

    void fTable::on_wConvertEngine_clicked()
    {
        dbcontrol->changeEngine(currentTable, ui->wEngine->currentText());
        showTable();
    }

    void fTable::on_wSetTableName_clicked()
    {
        if(currentTable.isEmpty())
        {
            if(dbcontrol->execQuery("create table " + ui->wTableName->text() + "(a int)"))
            {
                currentTable = ui->wTableName->text();
            }
        }
        else
        {
            if(!ui->wTableName->text().isEmpty())
            {
                if(dbcontrol->execQuery("RENAME TABLE " + currentTable + " TO " + ui->wTableName->text()))
                {
                    currentTable = ui->wTableName->text();
                }
            }
        }
        showTable();
    }

    void fTable::on_pushButton_3_clicked()
    {
        if(!ui->wCommentTable->toPlainText().isEmpty())
        {
```

```
            dbcontrol->execQuery("ALTER TABLE " + currentTable + " COMMENT='" + ui->wCommentTable->toPlainText() + "'");
            showTable();
        }
    }

    void fTable::on_wCreateTemporary_clicked()
    {

    }

    void fTable::on_wClone_clicked()
    {
        if(dbcontrol->execQuery("show create table " + ui->wCloneTable->currentText()))
        {
            QString createNewTable;
            while(dbcontrol->lastQuery()->next())
            {
                createNewTable = dbcontrol->lastQuery()->value(1).toString();
            }
            qDebug() << "get create table";
            QString oldName = currentTable + "888deleted";
            if(dbcontrol->execQuery("rename table " + currentTable + " to " + oldName))
            {
                qDebug() << "rename table";
                QStringList lst = createNewTable.split("`");
                lst[1] = currentTable;
                qDebug() << lst.join("`");
                if(dbcontrol->execQuery(lst.join("`")))
                {
                    qDebug() << "create table";
                    if(ui->wWithDataClone->isChecked())
                    {
                        dbcontrol->execQuery("insert into " + currentTable + " select * from " + oldName);
                    }
                    dbcontrol->execQuery("drop table " + oldName);
                }
            }

        }
    }
```

Ftrigger.cpp

```
#include "ftrigger.h"
#include "ui_ftrigger.h"

#include <QTextBlock>
#include <QDebug>

fTrigger::fTrigger(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::fTrigger)
{
    ui->setupUi(this);

    this->setWindowIcon(QIcon(":/new/fMainwindow/mainwindow/main_icon3.png"));
    this->setWindowTitle("Trigger");

    ui->wTime->insertItem(0, "before");
    ui->wTime->insertItem(0, "after");
    ui->wMode->insertItem(0, "insert");
```

```
    ui->wMode->insertItem(0, "update");
    ui->wMode->insertItem(0, "delete");

    filter = new tabFilter();
    ui->wEdit->installEventFilter(filter);

    this->helper = new AutoHelper;
    helper->setWordTip();
    connect(filter, &tabFilter::tabPress, helper, &AutoHelper::setFocusOnTips);
    connect(helper, &AutoHelper::choosedItem, this, &fTrigger::changeCurrentWord);
    connect(helper, &AutoHelper::itemClicked,  this, &fTrigger::changeCurrentWord);
    connect(this, &fTrigger::showTips, helper, &AutoHelper::showTips);
}

fTrigger::~fTrigger()
{
    delete ui;
}

void fTrigger::changeCurrentWord(QListWidgetItem * item)
{
    QStringList text = ui->wEdit->toPlainText().split(" ");
    if(text.last().back() == '.')
    {
        ui->wEdit->setPlainText(text.join(" ") +  item->text() + " ");
    }
    else
    {
        if(text.isEmpty())
        {
            ui->wEdit->setPlainText(item->text() + " ");
        }
        else
        {
            text.pop_back();
            text.push_back(item->text());
            ui->wEdit->setPlainText(text.join(" ") +  " ");
        }
    }
    QTextCursor cur(ui->wEdit->textCursor());
    cur.setPosition(ui->wEdit->toPlainText().length());
    ui->wEdit->setTextCursor(cur);
}

void fTrigger::on_wEdit_textChanged()
{
    QRect rc = ui->wEdit->cursorRect();
    helper->setGeometry(ui->wEdit->viewport()->mapToGlobal(QPoint(rc.x(), rc.y())).x(),
                        ui->wEdit->viewport()->mapToGlobal(QPoint(rc.x(), rc.y())).y() + rc.height(),
                        150, 75);
    emit showTips(ui->wEdit->textCursor().block().text().split(" ").last());
    ui->wEdit->activateWindow();
}

void fTrigger::setHelper(QMap<QString, QStringList> map)
{
    helper->setAddTip(map);
}

void fTrigger::showForm(QString trigName)
{
    ui->wTable->clear();
```

```
    ui->wTable->insertItems(0, dbcontrol->tablesList(dbcontrol->getCurrentDatabaseName()).keys());
    if(trigName.isEmpty())
    {
        this->trigName = "";
        reload();
    }
    else
    {
        this->trigName = trigName;
        reload();
        if(ui->wName->text().isEmpty())
        {
            this->trigName = "";
            reload();
        }
    }
    show();
}

void fTrigger::setDBControl(CDatabaseControl *dbcontrol)
{
    this->dbcontrol = dbcontrol;
}

void fTrigger::reload()
{
    if(trigName.isEmpty())
    {
        ui->wName->clear();
        ui->wEdit->clear();
    }
    else
    {
        if(dbcontrol->execQuery("show triggers"))
        {
            while (dbcontrol->lastQuery()->next()) {
                if(dbcontrol->lastQuery()->value(0).toString() == trigName)
                {
                    break;
                }
            }
            if(dbcontrol->lastQuery()->isValid())
            {
                ui->wName->setText(dbcontrol->lastQuery()->value(0).toString());
                ui->wMode->setCurrentText(dbcontrol->lastQuery()->value(1).toString().toLower());
                ui->wTable->setCurrentText(dbcontrol->lastQuery()->value(2).toString().toLower());
                ui->wEdit->setPlainText(dbcontrol->lastQuery()->value(3).toString());
                ui->wTime->setCurrentText(dbcontrol->lastQuery()->value(4).toString());
            }
        }
    }
}

QString fTrigger::collect()
{
    QString query = "CREATE TRIGGER ";

    query += ui->wName->text() + " ";
    query += ui->wTime->currentText() + " ";
    query += ui->wMode->currentText() + " ";
    query += "ON " + ui->wTable->currentText() + "\n";
    query += "FOR EACH ROW\n";
```

```
    query += ui->wEdit->toPlainText();

    return query;
}

void fTrigger::moveEvent(QMoveEvent *event)
{
    helper->hide();
    QWidget::moveEvent(event);
}

void fTrigger::resizeEvent(QResizeEvent *event)
{
    helper->hide();
    QWidget::resizeEvent(event);
}

void fTrigger::closeEvent(QCloseEvent *event)
{
    helper->hide();
    QWidget::closeEvent(event);
}

void fTrigger::hideEvent(QHideEvent *event)
{
    helper->hide();
    QWidget::hideEvent(event);
}

void fTrigger::on_wOk_clicked()
{
    if(trigName.isEmpty())
    {
        if(dbcontrol->execQuery(collect()))
        {
            trigName = ui->wName->text();
            reload();
        }
    }
    else
    {
        if(dbcontrol->execQuery("show triggers"))
        {
            while (dbcontrol->lastQuery()->next()) {
                if(dbcontrol->lastQuery()->value(0).toString() == trigName)
                {
                    break;
                }
            }
            if(dbcontrol->lastQuery()->isValid())
            {
                QString oldDefinition = "CREATE TRIGGER ";
                oldDefinition += dbcontrol->lastQuery()->value(0).toString() + " ";
                oldDefinition += dbcontrol->lastQuery()->value(4).toString() + " ";
                oldDefinition += dbcontrol->lastQuery()->value(1).toString().toLower() + " ";
                oldDefinition += "ON " + dbcontrol->lastQuery()->value(2).toString().toLower() + "\n";
                oldDefinition += "FOR EACH ROW\n";
                oldDefinition += dbcontrol->lastQuery()->value(3).toString();

                if(dbcontrol->execQuery("DROP TRIGGER " + trigName))
                {
                    if(dbcontrol->execQuery(collect()))
```

```
                {
                    trigName = ui->wName->text();
                    reload();
                }
                else
                {
                    dbcontrol->execQuery(oldDefinition);
                }
            }
        }
    }

    }
}
```

Fmainwindow.cpp

```
include "fmainwindow.h"

#include <QApplication>
#include <QStyleFactory>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);

    QApplication::setStyle(QStyleFactory::create("fusion"));

    fMainWindow w;
    w.show();

    return a.exec();
}
#include "tabFilter.h"
#include <QEvent>
#include <QKeyEvent>

tabFilter::tabFilter(QObject *parent) : QObject(parent)
{

}

bool tabFilter::eventFilter(QObject * watched, QEvent *event)
{
    if(event->type() == QEvent::KeyPress)
    {
        QKeyEvent * kEvent = static_cast<QKeyEvent*>(event);
        if(kEvent->key() == Qt::Key_Tab)
        {
            emit this->tabPress();
            return true;
        }
    }
    return false;
}
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_ doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом_.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і откомпільовану програму
Презентація	
Презентація_.ppt	Презентація кваліфікаційної роботи