**МIНIСТЕРСТВО  ОСВIТИ І НАУКИ  УКРАЇНИ**

**НАЦІОНАЛЬНИЙ   ТЕХНІЧНИЙ УНІВЕРСИТЕТ   УКРАЇНИ**

**“КИЇВСЬКИЙ  ПОЛІТЕХНІЧНИЙ  ІНСТИТУТ**

**ІМЕНІ ІГОРЯ СІКОРСЬКОГО”**

**Факультет прикладної математики**

**Кафедра програмного забезпечення комп’ютерних систем**

**Лабораторна робота № 7**

**з дисципліни “ Основи програмування ”**

**тема “Бази даних. Основи SQL”**

|  |  |  |
| --- | --- | --- |
| **Виконав**  **студент I курсу**  **групи КП-82**  **Мельничук Олексій Геннадійович**  **варіант №15** |  | **Перевірив**  **“\_\_\_\_” “\_\_\_\_\_\_\_\_\_\_\_\_” 20\_\_\_ р.**  **викладач**  **Гадиняк Руслан Анатолійович**  **(прізвище, ім’я, по батькові)** |

**Київ 2019**

**Мета роботи**

Навчитись працювати з реляційними базами даних.

Реалізувати модуль для доступу до SQLite бази даних.

**Постановка завдання**

Визначення та позначки:

SQLite клієнт - програма для керування SQLite базами даних.

Кроки виконання:

Скопіювати у проект даного завдання ваше CUI рішення лабораторної роботи №6.

Встановити будь-який графічний (або консольний) SQLite клієнт.

Сховище даних SQLite:

Створити за допомогою SQLite клієнта файл бази даних data.sqlite у директорії data/sqlite (UPD-1):

Створити у базі даних дві таблиці, що відповідатимуть за основний та додатковий тип даних. Назвати таблиці аналогічно назвам файлів з даними (наприклад, students, courses).

(UPD-2) Для цього можна також імпортувати CSV файли з даними і відредагувати типи і обмеження атрибутів створених таблиць.

Кожна таблиця має мати відповідні своєму типу даних типізовані стовпці.

Назви таблиць і їх стовпців мають задовольняти правила іменування змінних мови С (і бути в snake case).

Кожна таблиця обов'язково має мати стовпець id, як автоінкрементний цілочисельний ключ.

Заповнити кожну таблицю відповідними записами з даними (можна взяти з CSV файлів).

Створити модуль sqlite\_storage з класом SqliteStorage, що публічно наслідується від абстрактного класу Storage.

Реалізувати SqliteStorage з використанням Qt SQL класів:

Додати у файл налаштувань проекту (.pro) QT += sql як перший рядок .

Підключити у модулі sqlite\_storage заголовочний файл #include <QtSql>

(UPD-2) Ідентифікатори нових записів (для автоінкрементних атрибутів таблиць) генеруються базою даних. При додаванні нового запису в БД ідентифікатор не задається. Його можна отримати після виконання цього запиту через метод lastInsertId() (див. Додаток А).

(UPD-2) Особливістю даного сховища буде те, що всі зміни сутностей в його методах одразу змінюватимуть базу даних.

При створенні об'єкта сховища замінити його тип на SqliteStorage, а шлях до файлу на ../lab8/data/sqlite (UPD-1).

**Тексти коду програм**

|  |
| --- |
| **main.cpp** |
| #include "mainwindow.h"  #include <QApplication>  int main(int argc, char \*argv[])  {  QApplication a(argc, argv);  MainWindow w;  w.show();  return a.exec();  } |

|  |
| --- |
| **storage.h** |
| #pragma once  #include <iostream>  #include <fstream>  #include <vector>  #include <sstream>  #include "optional.h"  #include "dino.h"  #include "period.h"  using namespace std;  class Storage  {  public:  virtual bool load(string path) = 0;  virtual bool save() = 0;  // dinosaurs  virtual vector<Dino> getAllDinos(void) = 0;  virtual optional<Dino> getDinoById(int Dino\_id) = 0;  virtual bool updateDino(const Dino &dvno) = 0;  virtual bool removeDino(int Dino\_id) = 0;  virtual int insertDino(const Dino &dvno) = 0;  // periods  virtual vector<Per> getAllPers(void) = 0;  virtual optional<Per> getPerById(int Per\_id) = 0;  virtual bool updatePer(const Per &pr) = 0;  virtual bool removePer(int Per\_id) = 0;  virtual int insertPer(const Per &pr) = 0;  }; |

|  |
| --- |
| **csv\_storage.h** |
| #pragma once  #include <vector>  #include <string>  #include "dino.h"  #include "period.h"  #include "csv.h"  #include "optional.h"  #include "storage.h"  using std::string;  using std::vector;  class CsvStorage : public Storage  {  const string dir\_name\_;  vector<Dino> dinos\_;  vector<Per> pers\_;  static Dino rowToDino(const CsvRow &row);  static CsvRow DinoToRow(const Dino &dino);  static Per rowToPer(const CsvRow &row);  static CsvRow PerToRow(const Per &pr);  int getNewDinoId();  int getNewPerId();  public:  CsvStorage(const string & dir\_name) : dir\_name\_(dir\_name) { }  bool load(string path);  bool save();  // Dinos  vector<Dino> getAllDinos();  optional<Dino> getDinoById(int Dino\_id);  bool updateDino(const Dino &dvno);  bool removeDino(int Dino\_id);  int insertDino(const Dino &dvno);  vector<Per> getAllPers();  optional<Per> getPerById(int Per\_id);  bool updatePer(const Per &pr);  bool removePer(int Per\_id);  int insertPer(const Per &pr);  }; |

|  |
| --- |
| **csv\_storage.cpp** |
| #include "dino.cpp"  #include "period.cpp"  #include "csv\_storage.h"  #include <fstream>  #include <iostream>  using namespace std;  string fillStringFromFile(string fromw)  {  string csv;  string line;  ifstream myfile(fromw);  string::iterator it = line.end();  if (myfile.is\_open())  {  while (getline(myfile, line))  {  line+="\n";  csv += line;  }  myfile.close();  }  else  {  cerr << "File was not found: " << fromw << endl;  abort();  }  it = csv.end();  it--;  \*it = '\0';  return csv;  }  void writeToFile(string from, string tofile)  {  ofstream myfile(tofile);  if(myfile.is\_open())  {  myfile << from;  myfile.close();  }  else  {  cout << "Cannot open file\n"  << endl;  abort();  }  }  bool CsvStorage::load(string path)  {  string dinos\_filename = path + "/data.csv";  string dinos\_csv = fillStringFromFile(dinos\_filename);  CsvTable dinos\_table = Csv::createTableFromString(dinos\_csv);  for(CsvRow & row : dinos\_table)  {  Dino newone;  newone.id = stoi(row[0]);  newone.name = row[1];  newone.per = row[2];  newone.fam = row[3];  newone.disc = stoi(row[4]);  newone.mass = stod(row[5]);  this->dinos\_.push\_back(newone);  }  string pers\_filename = path + "/data2.csv";  string pers\_csv = fillStringFromFile(pers\_filename);  CsvTable pers\_table = Csv::createTableFromString(pers\_csv);  for(CsvRow & row2 : pers\_table)  {  Per newone;  newone.id = stoi(row2[0]);  newone.name = row2[1];  newone.time = stod(row2[2]);  this->pers\_.push\_back(newone);  }  return true;  }  bool CsvStorage::save()  {  string dinos\_filename = this->dir\_name\_ + "/data.csv";  CsvTable newt;  for(Dino & a : this->dinos\_)  {  CsvRow row;  row.push\_back(to\_string(a.id));  row.push\_back(a.name);  row.push\_back(a.per);  row.push\_back(a.fam);  row.push\_back(to\_string(a.disc));  row.push\_back(to\_string(a.mass));  newt.push\_back(row);  }  string csv\_txt = Csv::createStringFromTable(newt);    writeToFile(csv\_txt, dinos\_filename);  string pers\_filename = this->dir\_name\_ + "/data2.csv";  CsvTable newt2;  for(Per & p : this->pers\_)  {  CsvRow row;  row.push\_back(to\_string(p.id));  row.push\_back(p.name);  row.push\_back(to\_string(p.time));  newt2.push\_back(row);  }  string csv\_txt2 = Csv::createStringFromTable(newt2);    writeToFile(csv\_txt2, pers\_filename);  return true;  }  vector<Dino> CsvStorage::getAllDinos()  {  return this->dinos\_;  }  optional<Dino> CsvStorage::getDinoById(int Dino\_id)  {  for(Dino & c : this->dinos\_)  {  if(c.id == Dino\_id)  {  return c;  }  }  return nullopt;  }  bool CsvStorage::removeDino(int Dino\_id)  {  int index = -1;  for(int i = 0; i < this->dinos\_.size(); i++)  {  if(this->dinos\_[i].id == Dino\_id)  {  index = i;  break;  }  }  if(index >= 0)  {  this->dinos\_.erase(this->dinos\_.begin() + index);  return true;  }  return false;  }  int CsvStorage::getNewDinoId()  {  int max\_id = 0;  for(Dino & d : this->dinos\_)  {  if(d.id > max\_id)  {  max\_id = d.id;  }  }  int new\_id = max\_id + 1;  return new\_id;  }  int CsvStorage::insertDino(const Dino &dvno)  {  int new\_id = this->getNewDinoId();  Dino copy = dvno;  copy.id = new\_id;  this->dinos\_.push\_back(copy);  return new\_id;  }  bool CsvStorage::updateDino(const Dino &dvno)  {  int Dino\_id = dvno.id;  for(Dino & c : this->dinos\_)  {  if(c.id == Dino\_id)  {  c.disc = dvno.disc;  c.fam = dvno.fam;  c.mass = dvno.mass;  c.name = dvno.name;  c.per = dvno.per;  return true;  }  }  return false;  }  vector<Per> CsvStorage::getAllPers()  {  return this->pers\_;  }  optional<Per> CsvStorage::getPerById(int id)  {  for(Per & p : this->pers\_)  {  if(p.id == id)  {  return p;  }  }  return nullopt;  }  bool CsvStorage::removePer(int id)  {  int index = -1;  for(int i = 0; i < this->pers\_.size(); i++)  {  if(this->pers\_[i].id == id)  {  index = i;  break;  }  }  if(index >= 0)  {  this->pers\_.erase(this->pers\_.begin() + index);  return true;  }  return false;  }  int CsvStorage::getNewPerId()  {  int max\_id = 0;  for(Per & p : this->pers\_)  {  if(p.id > max\_id)  {  max\_id = p.id;  }  }  int new\_id = max\_id + 1;  return new\_id;  }  int CsvStorage::insertPer(const Per &pr)  {  int new\_id = this->getNewPerId();  Per copy = pr;  copy.id = new\_id;  this->pers\_.push\_back(copy);  return new\_id;  }  bool CsvStorage::updatePer(const Per &pr)  {  int Per\_id = pr.id;  for(Per & p : this->pers\_)  {  if(p.id == Per\_id)  {  p.name = pr.name;  p.time = pr.time;  return true;  }  }  return false;  } |

|  |
| --- |
| **add\_d.h** |
| #ifndef ADD\_D\_H  #define ADD\_D\_H  #include <QDialog>  #include <QIntValidator>  #include <QDoubleValidator>  #include <QRegExpValidator>  #include "dino.h"  namespace Ui {  class add\_d;  }  class add\_d : public QDialog  {  Q\_OBJECT  public:  explicit add\_d(QWidget \*parent = 0);  ~add\_d();  Dino getDino();  \  private slots:  void on\_buttonBox\_accepted();  void on\_buttonBox\_rejected();  private:  Ui::add\_d \*ui;  Dino dino;  QIntValidator int\_val;  QDoubleValidator double\_val;  QRegExpValidator regx\_val;  };  #endif // ADD\_D\_H |

|  |
| --- |
| **add\_d.cpp** |
| #include "add\_d.h"  #include "ui\_add\_d.h"  #include <QDebug>  #include <QMessageBox>  add\_d::add\_d(QWidget \*parent) :  QDialog(parent),  ui(new Ui::add\_d)  {  ui->setupUi(this);  double\_val.setNotation(QDoubleValidator::StandardNotation);  QLocale locale(QLocale::C);  double\_val.setLocale(locale);  ui->name->setValidator(&regx\_val);  ui->fam->setValidator(&regx\_val);  ui->per->setValidator(&regx\_val);  ui->disc->setValidator(&int\_val);  ui->mass->setValidator(&double\_val);  }  add\_d::~add\_d()  {  delete ui;  }  Dino add\_d::getDino()  {  return this->dino;  }  void add\_d::on\_buttonBox\_accepted()  {  if(ui->name->text().isEmpty() || ui->per->text().isEmpty() || ui->fam->text().isEmpty() || ui->disc->text().isEmpty() || ui->mass->text().isEmpty())  {  QMessageBox::warning(  this,  "WARNING",  "Some fields are empty",  QMessageBox::Ok);  return;  }  Dino d;  d.name = ui->name->text().toStdString();  d.per = ui->per->text().toStdString();  d.fam = ui->fam->text().toStdString();  d.disc = ui->disc->text().toInt();  d.mass = ui->mass->text().toDouble();  this->dino = d;  this->accept();  }  void add\_d::on\_buttonBox\_rejected()  {  this->reject();  } |

|  |
| --- |
| **add\_p.h** |
| #ifndef ADD\_P\_H  #define ADD\_P\_H  #include <QDialog>  #include <QIntValidator>  #include <QDoubleValidator>  #include <QRegExpValidator>  #include "period.h"  namespace Ui {  class add\_p;  }  class add\_p : public QDialog  {  Q\_OBJECT  public:  explicit add\_p(QWidget \*parent = 0);  ~add\_p();  Per getPer();  private slots:  void on\_buttonBox\_accepted();  void on\_buttonBox\_rejected();  private:  Ui::add\_p \*ui;  Per per;  QIntValidator int\_val;  QDoubleValidator double\_val;  QRegExpValidator regx\_val;  };  #endif // ADD\_P\_H |

|  |
| --- |
| **add\_p.cpp** |
| #include "add\_p.h"  #include "ui\_add\_p.h"  #include <QDebug>  #include <QMessageBox>  add\_p::add\_p(QWidget \*parent) :  QDialog(parent),  ui(new Ui::add\_p)  {  ui->setupUi(this);  double\_val.setNotation(QDoubleValidator::StandardNotation);  QLocale locale(QLocale::C);  double\_val.setLocale(locale);  ui->name->setValidator(&regx\_val);  ui->time->setValidator(&double\_val);  }  add\_p::~add\_p()  {  delete ui;  }  Per add\_p::getPer()  {  return this->per;  }  void add\_p::on\_buttonBox\_accepted()  {  if(ui->name->text().isEmpty() || ui->time->text().isEmpty())  {  QMessageBox::warning(  this,  "WARNING",  "Some fields are empty",  QMessageBox::Ok);  return;  }  Per p;  p.name = ui->name->text().toStdString();  p.time = ui->time->text().toDouble();  this->per = p;  this->accept();  }  void add\_p::on\_buttonBox\_rejected()  {  this->reject();  } |

|  |
| --- |
| **csv.cpp** |
| #include "csv.h"  CsvTable Csv::createTableFromString(const string & csv\_str)  {  int bracount = 0;  string buf;  const char \* p = csv\_str.c\_str();  CsvRow row;  CsvTable table;  while (1)  {  if (bracount == 0)  {  if (\*p == '\"')  {  bracount = 1;  }  else  {  if (\*p == '\n' || \*p == '\0')  {  row.push\_back(buf);  table.push\_back(row);  row.clear();  buf.clear();  }  else if(\*p == ',')  {  row.push\_back(buf);  buf.clear();  }  else  {  buf += \*p;  }  }  }  else  {  if (\*p == '\"')  {  if (\*(p + 1) == '\"')  {  buf += '\"';  p += 1;  }  else  {  row.push\_back(buf);  buf.clear();  if (\*(p + 1) == ',')  {  p += 1;  bracount = 0;  }  else if(\*(p + 1) == '\n' || \*(p + 1) == '\0')  {  table.push\_back(row);  row.clear();  p += 1;  bracount = 0;  }  else  {  std::cerr << "CSV format error: invalid char" << std::endl;  abort();  }  }  }  else  {  buf += \*p;  }  }  if (\*p == '\0')  {  break;  }  p += 1;  }  return table;  }  string Csv::createStringFromTable(const CsvTable &csv\_table)  {  bool escape = false;  string buf;  string::size\_type n = 0;  std::string bracount;  CsvRow row;  for (int i = 0; i < csv\_table.size(); i++)  {  row = csv\_table.at(i);  for (int j = 0; j < row.size(); j++)  {  n = 0;  bracount.clear();  std::string value = row.at(j);  if((value.find(',') != string::npos || value.find('\n') != string::npos) && value.find('\"') == string::npos)  {  buf += "\"";  buf += value;  buf += "\"";  }  else if(value.find('\"') != string::npos)  {  buf += "\"";  bracount = value;  while(n != string::npos)  {  if(n != 0)  {  bracount.insert(n, "\"");  n += 2;  }  n = value.find('\"', n);  }  buf += bracount;  buf += "\"";  }  else  {  buf += value;  }    if (j != csv\_table[i].size() - 1)  {  buf += ",";  }  }  if (i != csv\_table.size() - 1)  {  buf += "\n";  }  }  return buf;  } |

|  |
| --- |
| **optional.h** |
| #pragma once  #include <string>  using namespace std;  struct Dino  {  int id;  string name;  string per;  string fam;  int disc;  double mass;  }; |

|  |
| --- |
| **mainwindow.h** |
| #ifndef MAINWINDOW\_H  #define MAINWINDOW\_H  #include <QMainWindow>  #include "storage.h"  #include "csv\_storage.h"  #include <QMainWindow>  #include <QtCore>  #include <QDebug>  #include <QFileDialog>  #include <QListWidgetItem>  #include <QMessageBox>  #include <QFileInfo>  #include "add\_d.h"  #include "add\_p.h"  #include "edit\_d.h"  #include "edit\_p.h"  #include "ui\_mainwindow.h"  namespace Ui {class MainWindow;  }  class MainWindow : public QMainWindow  {  Q\_OBJECT  public:  explicit MainWindow(QWidget \*parent = 0);  ~MainWindow();  private slots:  void onNew();  void onOpen();  void on\_add\_clicked();  void on\_edit\_clicked();  void on\_remove\_clicked();  void on\_add2\_clicked();  void on\_edit2\_clicked();  void on\_remove2\_clicked();  void on\_listWidget2\_itemSelectionChanged();  void on\_listWidget\_itemSelectionChanged();  private:  Ui::MainWindow \*ui;  void loadfrom(QString & path);  void fillWListDino();  void fillWListPer();  bool fileWrite(QString & path, QString & data);  bool defaultFile(QString & path);  Storage \* storage\_; // <-- STORAGE  };  #endif // MAINWINDOW\_H |

|  |
| --- |
| **mainwindow.cpp** |
| #include "mainwindow.h"  #include "ui\_mainwindow.h"  #include "dino.h"  #include "period.h"  #include <QDebug>  #include <QFileDialog>  #include <QMessageBox>  MainWindow::MainWindow(QWidget \*parent) :  QMainWindow(parent),  ui(new Ui::MainWindow)  {  ui->setupUi(this);  connect(ui->actionOption1, &QAction::triggered, this, &MainWindow::onNew);  connect(ui->actionOption2, &QAction::triggered, this, &MainWindow::onOpen);  connect(ui->actionExit, &QAction::triggered, this, &MainWindow::close);  }  void MainWindow::onNew()  {  QFileDialog dialog(this);  dialog.setFileMode(QFileDialog::Directory);  QString current\_dir = QDir::currentPath();  QString default\_name = "new\_CsvStorage";  QString folder\_path = dialog.getSaveFileName(  this,  "Select New Storage Folder",  current\_dir + "/" + default\_name,  "Folders");  if(!folder\_path.isEmpty())  {  QDir().mkdir(folder\_path);  bool success = this->defaultFile(folder\_path);  if(!success)  {  ui->listWidget->clear();  ui->listWidget2->clear();  ui->add->setEnabled(false);  ui->add2->setEnabled(false);  }  }  }  MainWindow::~MainWindow()  {  delete this->storage\_;  delete ui;  }  void MainWindow::onOpen()  {  qDebug() << "OPEN";  QString file\_path = QFileDialog::getExistingDirectory(  this, // parent  "Dialog Caption", // caption  "", // directory to start with  QFileDialog::ShowDirsOnly | QFileDialog::DontResolveSymlinks); // file name filter  qDebug() << file\_path;  if(!file\_path.isEmpty())  {  this->loadfrom(file\_path);  }  }  void MainWindow::loadfrom(QString & path)  {  std::string filenamestd;  CsvStorage \* csv\_storage;  // QFileInfo check\_file(path);  // if (check\_file.exists() && check\_file.isFile())  // {  // }  // else  // {  // QDir().mkdir(path);  // bool success = this->defaultFile(path);  // if(!success)  // {  // ui->listWidget->clear();  // ui->listWidget2->clear();  // ui->add->setEnabled(false);  // ui->add2->setEnabled(false);  // }  // return;  // }  filenamestd = path.toUtf8().constData();  csv\_storage = new CsvStorage(filenamestd);  delete this->storage\_;  storage\_ = csv\_storage;  if(storage\_->load(filenamestd) == false)  {  QMessageBox::warning(  this,  "WARNING",  "Cannot open file!!!",  QMessageBox::Ok);  return;  }  ui->listWidget->clear();  this->fillWListDino();  ui->add->setEnabled(true);  ui->listWidget2->clear();  this->fillWListPer();  ui->add2->setEnabled(true);  }  void MainWindow::fillWListDino()  {  QVector<Dino> dinos = QVector<Dino>::fromStdVector(storage\_->getAllDinos());  QVariant var;  for(Dino & dino : dinos)  {  string stext = dino.name;  QString text = QString::fromStdString(stext);  QListWidgetItem \* new\_item = new QListWidgetItem(text);  var = QVariant::fromValue(dino);  new\_item->setData(Qt::UserRole, var);  ui->listWidget->addItem(new\_item);  }  }  void MainWindow::fillWListPer()  {  QVector<Per> pers = QVector<Per>::fromStdVector(storage\_->getAllPers());  QVariant var;  for(Per & per : pers)  {  string stext = per.name;  QString text = QString::fromStdString(stext);  QListWidgetItem \* new\_item = new QListWidgetItem(text);  var = QVariant::fromValue(per);  new\_item->setData(Qt::UserRole, var);  ui->listWidget2->addItem(new\_item);  }  }  bool MainWindow::fileWrite(QString &path, QString &data)  {  QFile file(path);  if(!file.open(QFile::WriteOnly | QFile::Text))  {  QMessageBox::warning(  this,  "WARNING",  "Cannot create storage!!!",  QMessageBox::Ok);  return false;  }  QTextStream out(&file);  out << data;  file.close();  return true;  }  bool MainWindow::defaultFile(QString &path)  {  QString filename = path + "/data.csv";  QString data = "1,Dino,Period,Family,999,999.0";  bool dinow = this->fileWrite(filename, data);  QString filename2 = path + "/data2.csv";  QString data2 = "1,Period,999.0";  bool perw = this->fileWrite(filename2, data2);  if(dinow == true && perw == true)  {  this->loadfrom(path);  return true;  }  else return false;  }  void MainWindow::on\_add\_clicked()  {  add\_d window(this);  if(window.exec())  {  QVariant var;  Dino dino = window.getDino();  dino.id = this->storage\_->insertDino(dino);  QListWidgetItem \* item = new QListWidgetItem;  var.setValue(dino);  const QString name = QString::fromUtf8(dino.name.c\_str());  item->setText(name);  item->setData(Qt::UserRole, var);  ui->listWidget->addItem(item);  }  this->storage\_->save();  }  void MainWindow::on\_edit\_clicked()  {  QList<QListWidgetItem \*> items = ui->listWidget->selectedItems();  Dino dino;  QVariant var;  foreach (QListWidgetItem \* item, items)  {  dino = item->data(Qt::UserRole).value<Dino>();  }  edit\_d window(dino, this);    if(window.exec())  {  dino = window.getDino();  this->storage\_->updateDino(dino);  foreach (QListWidgetItem \* item, items)  {  var.setValue(dino);  const QString name = QString::fromUtf8(dino.name.c\_str());  item->setText(name);  item->setData(Qt::UserRole, var);  }  }  this->storage\_->save();  }  void MainWindow::on\_remove\_clicked()  {  QMessageBox::StandardButton remove;  remove = QMessageBox::question(  this,  "Remove",  "Are you sure?",  QMessageBox::Yes|QMessageBox::No);  if (remove == QMessageBox::Yes)  {  QList<QListWidgetItem \*> items = ui->listWidget->selectedItems();  Dino dino;  foreach(QListWidgetItem \* item, items)  {  dino = item->data(Qt::UserRole).value<Dino>();  this->storage\_->removeDino(dino.id);  delete ui->listWidget->takeItem(ui->listWidget->row(item));  }  }  this->storage\_->save();  }  void MainWindow::on\_add2\_clicked()  {  add\_p window(this);  if(window.exec())  {  QVariant var;  Per per = window.getPer();  per.id = this->storage\_->insertPer(per);  QListWidgetItem \* item = new QListWidgetItem;  var.setValue(per);  const QString name = QString::fromUtf8(per.name.c\_str());  item->setText(name);  item->setData(Qt::UserRole, var);  ui->listWidget2->addItem(item);  }  this->storage\_->save();  }  void MainWindow::on\_edit2\_clicked()  {  QList<QListWidgetItem \*> items = ui->listWidget2->selectedItems();  Per per;  QVariant variant;  foreach (QListWidgetItem \* item, items)  {  per = item->data(Qt::UserRole).value<Per>();  }  edit\_p window(per, this);    if(window.exec())  {  per = window.getPer();  this->storage\_->updatePer(per);    foreach (QListWidgetItem \* item, items)  {  variant.setValue(per);  const QString name = QString::fromUtf8(per.name.c\_str());  item->setText(name);  item->setData(Qt::UserRole, variant);  }  }  this->storage\_->save();  }  void MainWindow::on\_remove2\_clicked()  {  QMessageBox::StandardButton remove;  remove = QMessageBox::question(  this,  "Remove",  "Are you sure?",  QMessageBox::Yes|QMessageBox::No);  if (remove == QMessageBox::Yes)  {  QList<QListWidgetItem \*> items = ui->listWidget2->selectedItems();  Per per;  foreach(QListWidgetItem \* item, items)  {  per = item->data(Qt::UserRole).value<Per>();  this->storage\_->removePer(per.id);  delete ui->listWidget2->takeItem(ui->listWidget2->row(item));  }  }  this->storage\_->save();  }  void MainWindow::on\_listWidget2\_itemSelectionChanged()  {  QList<QListWidgetItem \*> items = ui->listWidget2->selectedItems();  if(items.size() != 1)  {  ui->lselected\_2->setText("");  ui->lplname->setText("");  ui->lpltime->setText("");  ui->lpname->setText("");  ui->lptime->setText("");  if(items.size() == 0)  {  ui->edit2->setEnabled(false);  ui->remove2->setEnabled(false);  }  else  {  ui->edit2->setEnabled(false);  ui->remove2->setEnabled(true);  }  }  else  {  ui->edit2->setEnabled(true);  ui->remove2->setEnabled(true);  foreach (QListWidgetItem \* item, items)  {  Per per = item->data(Qt::UserRole).value<Per>();  ui->lselected\_2->setText("Selected period:");  ui->lplname->setText("Name:");  ui->lpltime->setText("Length:");  ui->lpname->setText(QString::fromUtf8(per.name.c\_str()));  ui->lptime->setText(QString::number(per.time));  }  }  }  void MainWindow::on\_listWidget\_itemSelectionChanged()  {  QList<QListWidgetItem \*> items = ui->listWidget->selectedItems();  if(items.size() != 1)  {  ui->lselected->setText("");  ui->ldlname->setText("");  ui->ldlper->setText("");  ui->ldlfam->setText("");  ui->ldldisc->setText("");  ui->ldlmass->setText("");  ui->ldname->setText("");  ui->ldper->setText("");  ui->ldfam->setText("");  ui->lddisc->setText("");  ui->ldmass->setText("");  if(items.size() == 0)  {  ui->edit->setEnabled(false);  ui->remove->setEnabled(false);  }  else  {  ui->edit->setEnabled(false);  ui->remove->setEnabled(true);  }  }  else  {  ui->edit->setEnabled(true);  ui->remove->setEnabled(true);  foreach (QListWidgetItem \* item, items)  {  Dino dino = item->data(Qt::UserRole).value<Dino>();  ui->lselected->setText("Selected dino:");  ui->ldlname->setText("Name:");  ui->ldlper->setText("Period:");  ui->ldlfam->setText("Family:");  ui->ldldisc->setText("Discovered:");  ui->ldlmass->setText("Mass");  ui->ldname->setText(QString::fromUtf8(dino.name.c\_str()));  ui->ldper-> setText(QString::fromUtf8(dino.per.c\_str()));  ui->ldfam-> setText(QString::fromUtf8(dino.fam.c\_str()));  ui->lddisc->setText(QString::number(dino.disc));  ui->ldmass->setText(QString::number(dino.mass));  }  }  } |

|  |
| --- |
| **dino.h** |
| #pragma once  #include <string>  using namespace std;  struct Dino  {  int id;  string name;  string per;  string fam;  int disc;  double mass;  }; |

|  |
| --- |
| **period.h** |
| #pragma once  #include <string>  using namespace std;  struct Per  {  int id;  string name;  double time;  }; |

|  |
| --- |
| **dino.cpp** |
| #include "dino.h"  #include <QDebug>  Dino::Dino() { }  Dino::Dino(int id, string name, string per, string fam, int disc, double mass)  {  this->id = id;  this->name = name;  this->per = per;  this->fam = fam;  this->disc = disc;  this->mass = mass;  } |

|  |
| --- |
| **period.cpp** |
| #include "period.h"  #include <QDebug>  Per::Per() { }  Per::Per(int id, string name, double time)  {  this->id = id;  this->name = name;  this->time = time;  } |

|  |
| --- |
| **add\_d.ui** |
| #include "dino.h"  #include <QDebug>  Dino::Dino() { }  Dino::Dino(int id, string name, string per, string fam, int disc, double mass)  {  this->id = id;  this->name = name;  this->per = per;  this->fam = fam;  this->disc = disc;  this->mass = mass;  } |

|  |
| --- |
| **add\_p.ui** |
| #include "dino.h"  #include <QDebug>  Dino::Dino() { }  Dino::Dino(int id, string name, string per, string fam, int disc, double mass)  {  this->id = id;  this->name = name;  this->per = per;  this->fam = fam;  this->disc = disc;  this->mass = mass;  } |

|  |
| --- |
| **data.csv** |
| 1,Plotosaurus,Cretaceous,Mosasauridae,1941,2400.000000  2,Coelophysis,Triassic,Coelophysidae,1887,20.000000  3,Excison,Modern,Destroid,2012,75.000000  5,Diplodocus,Jurassic,Diplodocidae,1877,13000.000000  6,KJSawka,Modern,Destroid,2014,75.000000  7,Kek,Sus,Lel,228,1337.000000 |

|  |
| --- |
| **data2.csv** |
| 6,Stone,30000.000000  9,Iron,2500.000000  42,Golden,100.000000  43,Platinum,67854.000000 |

**Робота коду**

Програма будується у QTCreator.

Є опція відкрити і заповнити сховище з CSV файлу.

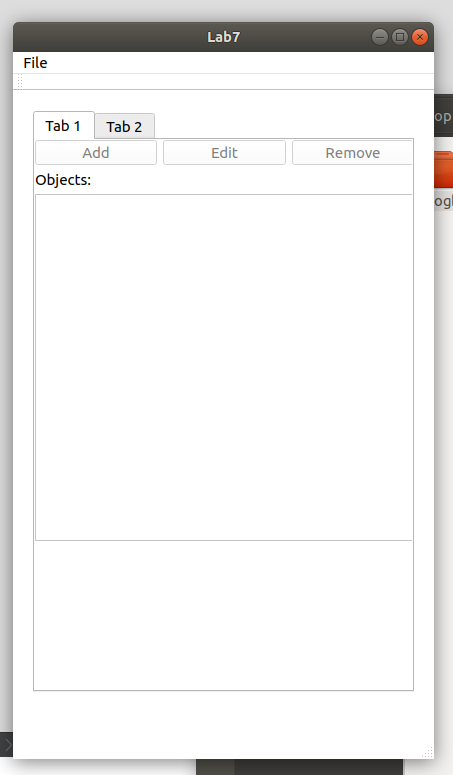
Є опція створити нове CSV сховище, при цьому створються два файли.

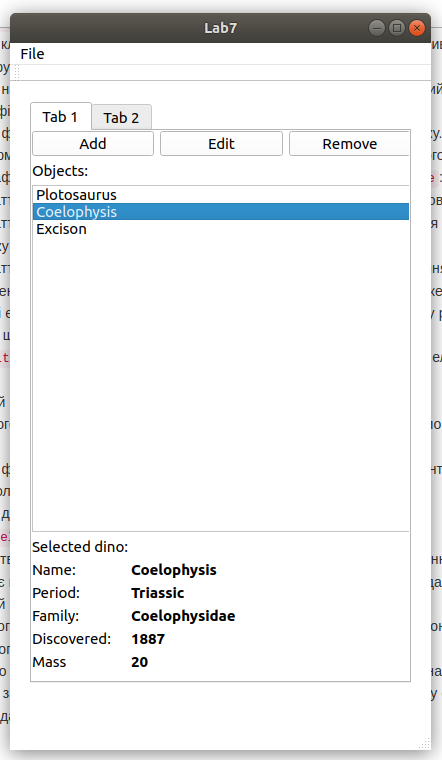
При завантаженні сховища заповнюються два списки.

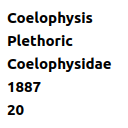
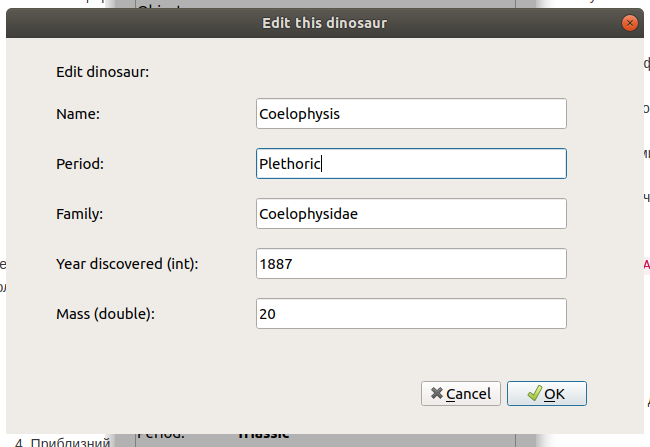
Є можливість видаляти, редагувати та створювати нові сутності.

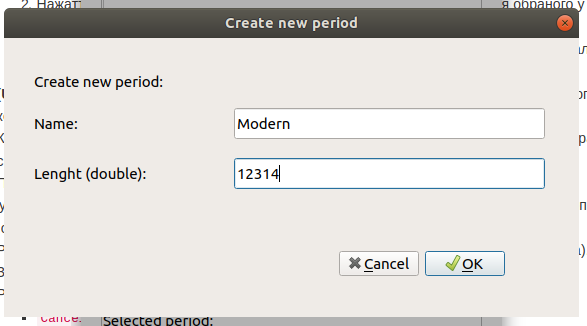
При цьому сховище CSV записуж зміни.

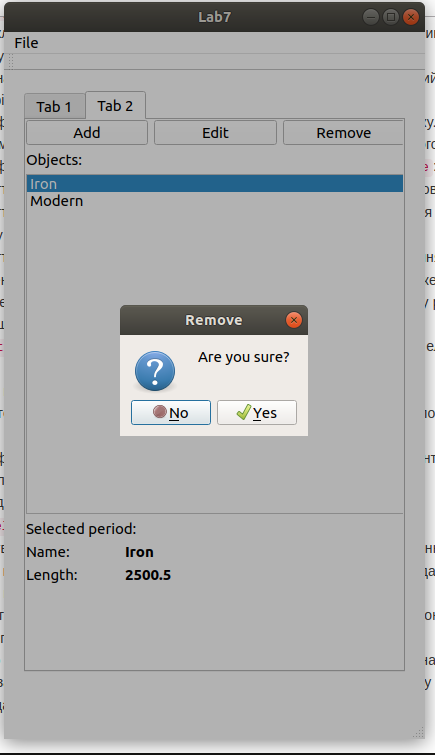
**Приклади результатів**

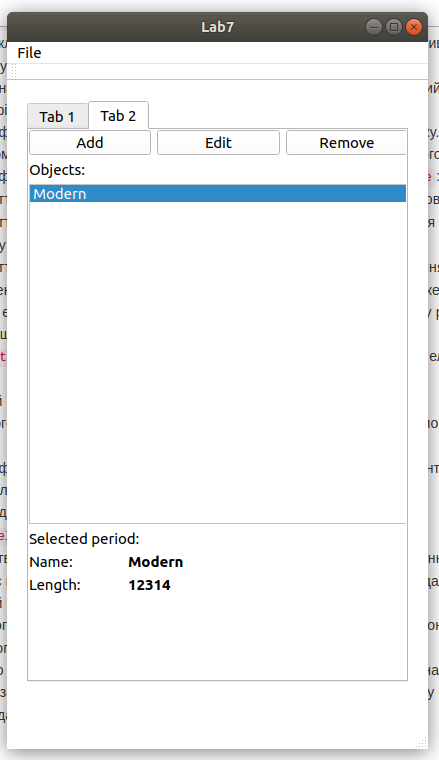
****

****

****

****

****

****

**Висновки**

Лабораторна допомогла опанувати навичками роботи з с sqlite.

Також був ознайомлений із поняттям бази даних, що таке реляційна база даних та для чого вони використовуються.