МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ АГРАРНИЙ УНІВЕРСИТЕТ

Факультет менеджменту

Кафедра економічної кібернетики, комп'ютерних наук та інформаційних технологій

БАЗИ ДАНИХ

методичні рекомендації для практичних занять та самостійної роботи здобувачів першого (бакалаврського) рівня вищої освіти ОПП «Комп'ютерні науки» спеціальності F3(122) «Комп'ютерні науки» денної форми здобуття вищої освіти

МИКОЛАЇВ
2025

УДК 004.65
Б17

Друкується за рішенням науково-методичної комісії факультету менеджменту Миколаївського національного аграрного університету від 27 березня 2025 року, протокол № 7.

Укладачі:

С. І. Тищенко	к.п.н., доцент, доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету
О. Ю. Пархоменко	к.ф.-м.н., доцент, доцент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету
О. О. Жебко	асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету
В. С. Петренко	магістр MBA, викладач Університету прикладних наук Вайєнштефан-Тріздорф, Німеччина
А. М. Коломієць	асистент кафедри економічної кібернетики, комп'ютерних наук та інформаційних технологій Миколаївського національного аграрного університету

Рецензенти:

О. С. Садовий	- канд. техн. наук, доцент, завідувач кафедри агроінженерії Миколаївського національного аграрного університету
Ю. В. Грицук	- канд. техн. наук, доцент кафедри загальної інженерної підготовки Донбаської національної академії будівництва і архітектури

Б17 **Бази** даних : методичні рекомендації для практичних занять та самостійної роботи здобувачів першого (бакалаврського) рівня вищої освіти ОПП «Комп'ютерні науки» спеціальності F3(122) «Комп'ютерні науки» денної форми здобуття вищої освіти / уклад. О. О. Жебко. Миколаїв : МНАУ, 2025. 125 с.

УДК 004.65

ЗМІСТ

Передмова

Курс дисципліни «Бази даних» має важливе значення в теоретичній підготовці майбутніх фахівців і є обов'язковою компонентою підготовки здобувачів першого (бакалаврського) рівня вищої освіти ОПП «Комп'ютерні науки» спеціальності 122 «Комп'ютерні науки».

Мета дисципліни: сформувати у здобувачів необхідний обсяг теоретичних і практичних знань про термінологічний апарат та теоретичні концепції баз даних, принципи організації і побудови баз даних, систем управління базами даних, методів проектування і засобів використання систем управління базами даних як складових елементів комп'ютерних систем.

Основними завданнями, що мають бути вирішенні у процесі викладення дисципліни, є надання здобувачам вищої освіти:

- навичок проектування баз даних, визначення вимог, створення схем, розробку моделей даних та забезпечення відповідності цілям і вимогам бізнесу;

- здатності оптимізації запитів, індексації даних, забезпеченню цілісності та безпеки даних, а також веденню адміністрування баз даних;

- вмінь інтегрувати бази даних у програмні застосунки, розроблюючи зв'язки між застосунками та базами даних, створюючи API та інтерфейси для взаємодії з даними;

- знань про сучасні тенденції в галузі баз даних, такі як розподілені системи, обробка великих обсягів даних (Big Data), хмарні бази даних тощо.

Предмет дисципліни: основи сучасної теорії баз даних: введення в реляційні бази даних, нормалізація, реляційні системи управління базами даних SQL

Структура навчального курсу дозволяє здобувачам застосовувати мову структурованих запитів SQL до бази даних, специфікацій та проектувати логічні та фізичні моделі баз даних, запити до них та використовувати різноманітні СУБД.

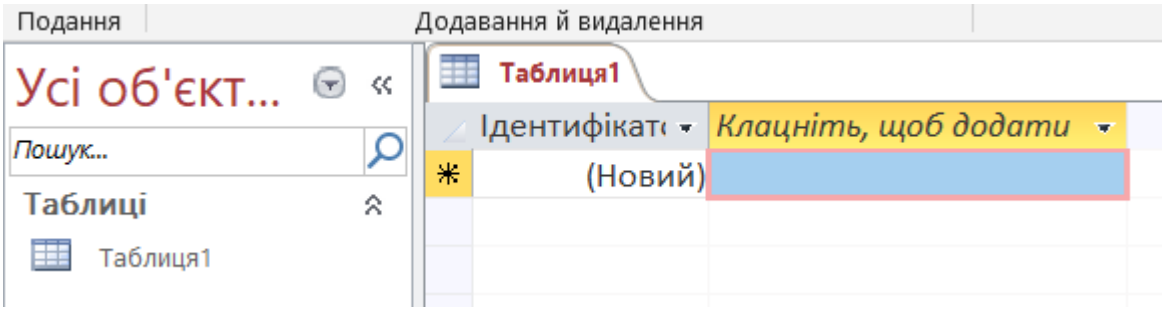
Практична робота № 1

MS Access. Створення таблиць бази даних та перегляд схеми даних

Створити базу даних «Відділ кадрів», що складається з трьох таблиць із взаємозалежними даними, формами, які використовуються для наочної роботи з даними (введення, редагування тощо).

Завдання 1. Самостійно створити нову базу даних з назвою «Відділ кадрів». Після створення порожньої бази даних необхідно створити об'єкти цієї бази даних.

Завдання 2. Створення таблиці «Співробітники». Під час створення нової бази даних MS Access відразу пропонує роботу у режимі Таблиці. Правою кнопкою миші виділіть Таблицю1 та оберіть режим Конструктор



(рис. 1.1).

Рис. 1.1 - Вибір режиму Конструктор із контекстного меню

2. Правою кнопкою миші виділіть Таблицю1 і оберіть режим Конструктор (рис.1.2).

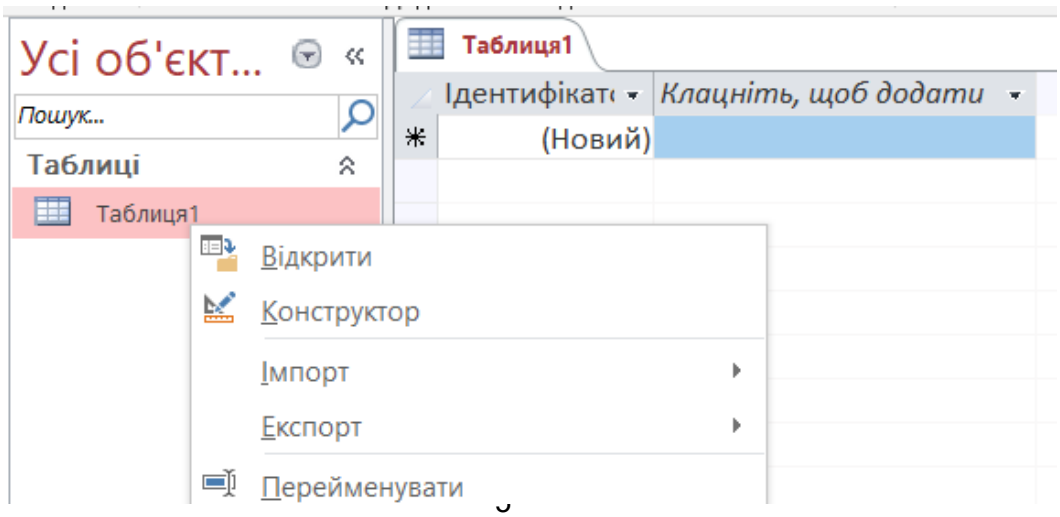
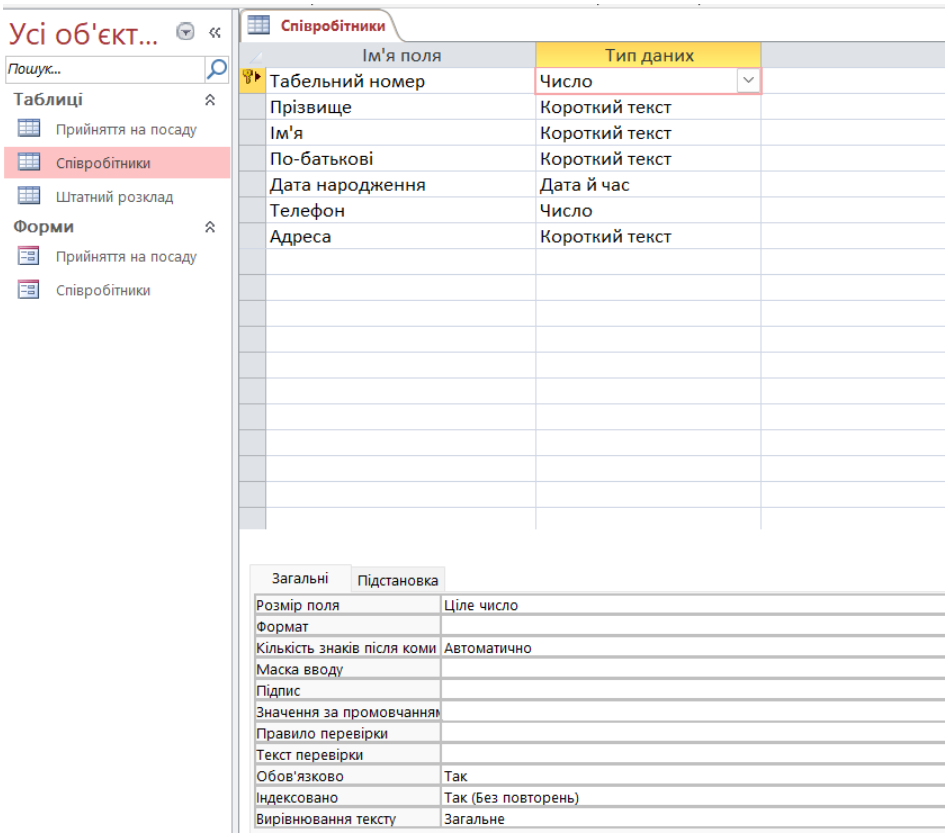


Рис. 1.2 - Вибір режиму Конструктор з контекстного меню

3. З'явиться вікно конструктора. У першому рядку введіть ім'я поля

«Табельний номер». У сусідній клітині з'явиться тип даних (за умовчанням Текстовий). У спадному меню виберіть тип Числовий (рис.1.3). Перейдіть до бланку *Властивості поля* в нижній частині вікна та задайте значення *Розмір поля: ціле*. Діючи аналогічно, задайте назви, вкажіть тип і



властивості даних для решти полів.

Рис. 1.3 - Створення структури таблиці у Конструкторі

Таблиця 1.1

Дані таблиці «Співробітники»

Ім’я поля	Тип даних	Властивості поля
Табельний номер (Ключове поле)	Числовий	Розмір поля: Ціле
Прізвище:	Текстовий	Розмір поля: 20
Ім’я	Текстовий	Розмір поля: 20
По-батькові	Текстовий	Розмір поля: 20

Дата народження	Дата/час	Формат поля: Короткий формат дати
Телефон	Числовий	Маска введення: 200-00-00
Адреса	Текстовий	Розмір поля: 50 Значення за замовчуванням: м. Київ

4. Після введення опису всіх полів таблиці вкажіть ключове поле, для чого, клацнувши область виділення рядка із записом поля Табельний номер, натисніть кнопку «Ключове поле» на панелі інструментів Конструктор або клацніть правою клавішею миші область виділення рядка із записом поля Табельний номер і в контекстному меню виберіть «Ключове поле». Після цього в області виділення поля «Табельний номер» з'явиться знак ключового поля - ключ.

5. Перейдіть до Режим таблиці за допомогою кнопки Режим і введіть наступні дані в таблицю «Співробітники» (табл. 1.2).

Таблиця 1.2

Дані для введення в таблицю «Співробітники»

Табл. номер	Прізвище	Ім’я	По-батькові	Дата народження	Телефон	Адреса
1	2	3	4	5	6	7
1	Шевченко	Микита	Іванович	01.01.1989	278-90-65	м. Львів
2	Мельник	Назар	Сергійович	04.11.1983	267-54-23	м. Вінниця
3	Горбенко	Роман	Станіславович	15.08.1979	245-87-65	м. Стрий

4	Антонов	Михайло	Степанович	25.03.1984	291-23-61	м. Одеса
5	Черемшина	Василь	Миколайович	15.04.1984	291-23-61	м. Полтава
6	Кожушко	Тетяна	Андріївна	18.01.1987	287-54-96	м. Миколаїв

6. Закрийте таблицю та збережіть під назвою «Співробітники».

Завдання 3. Створення таблиці «Штатний розклад»

1.Виберіть на другій вкладці стрічки *Створення* пункт конструктор таблиць та введіть дані подання у таблиці 1.3.

Таблиця 1.3

Дані таблиці «Штатний розклад»

Ім’я поля	Тип даних	Властивості поля
Код посади (Ключове поле)	Числовий	Розмір поля: Ціле
Назва посади	Текстовий	Розмір поля: 30
Оклад	Грошовий	

2.Після введення всіх полів таблиці вкажіть ключове поле.

3.Перейдіть до Режим таблиці та введіть дані з таблиці 1.4.

Таблиця 1.4

Дані для введення в таблицю «Штатний розклад»

Код посади	Назва посади	Оклад
5	Бригадир	5000.00
10	Конструктор	6000.00
15	Зам. начальника	12000.00
20	Інженер-технолог	10000.00
25	Начальник відділу	20000.00

4.Закрийте таблицю та збережіть під назвою «Штатний розклад».

Завдання 4. Створення таблиці «Прийняття на посаду»

Створюється за аналогією з таблицею «Штатний розклад». *Ключове поле для даної таблиці не потрібно!*

Таблиця 1.5

Дані таблиці «Прийняття на посаду»

Ім’я поля	Тип даних	Властивості поля
Табельний номер	Числовий	Розмір поля: Ціле
Код посади	Числовий	Розмір поля: Ціле
Дата наказу	Дата/час	Розмір поля: Короткий формат дати

Таблиця 1.6

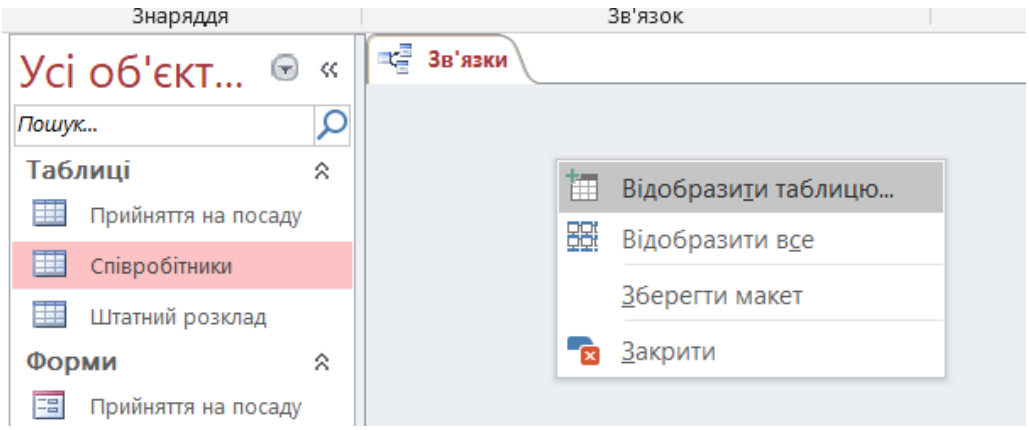
Дані для введення в таблицю «Прийняття на посаду»

Табельний номер	Код посади	Дата наказу
1	5	05.11.2009
2	10	23.12.2008
3	20	08.10.2009
4	10	15.07.2007
5	10	04.01.2010
6	20	15.12.2009

Завдання 5. Створення схеми даних

1. Щоб переглянути міжтабличні зв'язки, виберіть *Схема даних* на вкладці *Робота з базами* даних у групі *Відносини*(Отношения). Відкриється вікно «Схема даних».

2. У вікні правою кнопкою миші, що відкрилося, викликаємо



контекстне меню і вибираємо Відобразити таблицю (рис. 1.4)

Рис. 1.4 - Контекстне меню у пункті Схема даних

3. Додаємо всі створені раніше таблиці та розташуємо їх у порядку: Співробітники - Прийняття на посаду – Штатне розпис.

4. Збільшимо вікна таблиць, щоб було видно всі поля.

5. Перетягніть мишею поле Табельний номер із таблиці Співробітники на аналогічне поле в таблиці Прийняття на посаду. З'явиться діалогове вікно Зв'язку, представлене на рисунку 1.5.

— Увімкнемо значок *Забезпечення цілісності даних*. Це неможливо буде зробити, якщо типи обох полів задані не однаково.

— Увімкнемо значок *Каскадне оновлення* пов'язаних полів. Це призведе до того, що при зміні табельного номера у таблиці Співробітники автоматично зміниться відповідний номер у таблиці Прийняття на посаду.

— Увімкнемо значок Каскадне видалення пов'язаних полів. Це призведе до того, що при видаленні запису з табельним номером у таблиці Співробітники будуть видалені всі записи з таблиці Прийняття на посаду, на якій стояли відповідні номери груп.

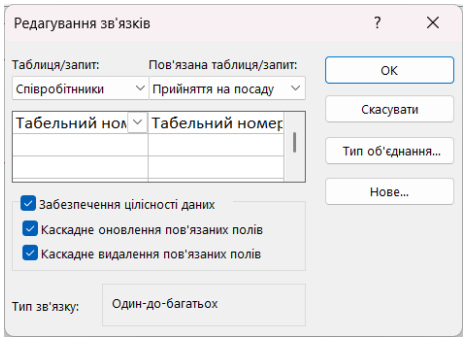


Рис. 1.5 - Діалогове вікно «Зв'язки»

— Натисніть кнопку Створити. З'явиться зв'язок «один-багатьом». Схема даних представлена рисунку 1.6.

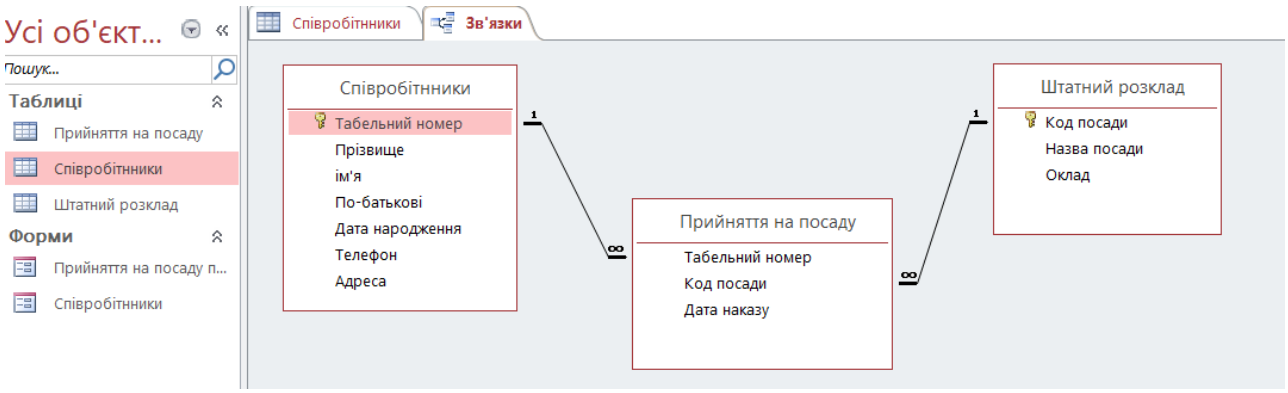


Рис. 1.6 - Вікно схеми даних

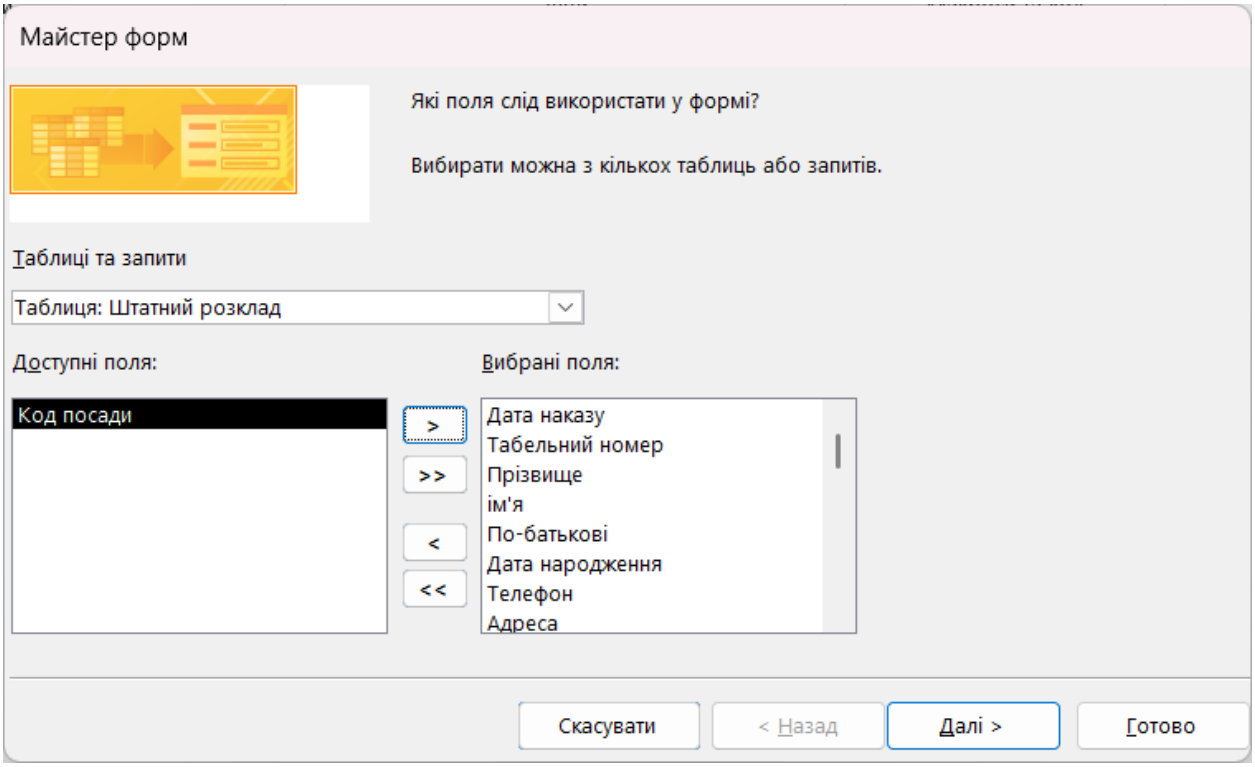
6. Далі перетягнемо поле Код посади з таблиці Прийняття на посаду на аналогічне поле в таблиці Штатне розклад. І проведемо аналогічні дії.

7. Закрийте схему даних, ствердно відповівши на питання про збереження схеми даних.

Завдання 6. Створення підлеглої форми за допомогою Майстра форм

1. Для створення форми за допомогою Майстра форм виберіть пункт Майстер форм на вкладці Створення у групі Форми.

2. У вікні Майстра форм виберіть: Таблиця Прийняття на посаду – поле Дата наказу Таблиця Співробітники – всі поля Таблиця Штатний



розклад – поля Назва посади та Оклад (рис. 1.7.)

Рис. 1.7 - Створення форми за допомогою Майстра форм

3. Натисніть кнопку Далі та вибираємо Підлеглу форму (рис. 1.8). При цьому дані таблиці Прийняття на посаду є одиночною формою.

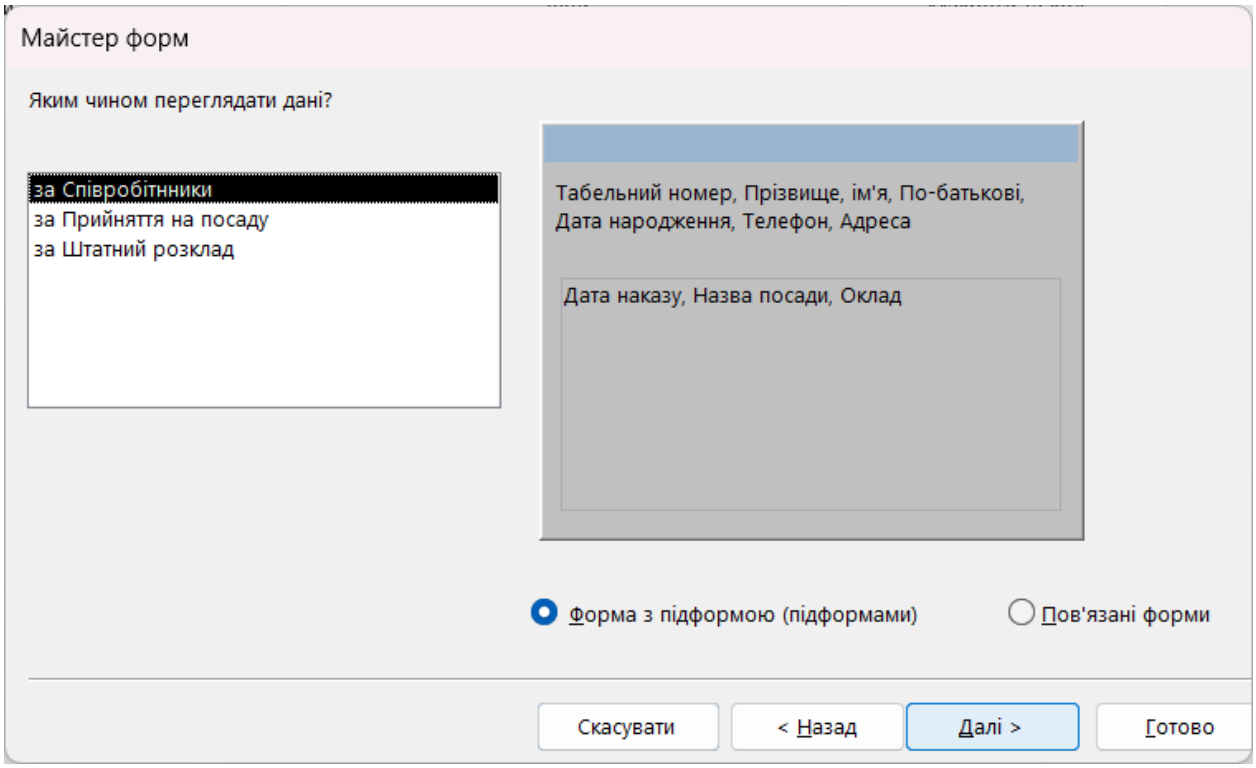


Рис.1.8 - Вибір виду форми

4. Натискаємо кнопку Далі та вибираємо зовнішній вигляд форми, в даному випадку краще буде Стрічковий.

5. Переходимо на наступний етап побудови форми. Задаємо ім'я головної та підлеглої форм. Натискаємо Змінити макет форми.

6. Форма відкрилася як конструктора. За бажанням змінюємо вигляд форми. Закриваємо та зберігаємо.

Практична робота № 2

MS Access. Створення БД «Телефонний довідник співробітників»

Створити базу даних «Телефонний довідник співробітників» (самостійну складову БД «Відділу кадрів»), що складається з двох таблиць із взаємозалежними даними, запитами, а також формами, що використовуються для наочної роботи з даними (введення, редагування тощо).

Завдання 1. Самостійно створити нову базу даних під назвою «Телефонний довідник співробітників». Після створення порожньої бази даних необхідно створити об'єкти цієї бази даних.

Завдання 2. Створення таблиць 1. Під час створення нової бази даних MS Access одразу пропонує роботу у режимі Таблиці. 2. Правою кнопкою миші виділіть *Таблицю1* та оберіть режим Конструктор та введіть дані з таблиці 2.1.

Таблиця 2.1

Дані таблиці «Довідник»

Ім’я поля	Тип даних	Властивість поля
Прізвище	Текстовий	Розмір поля: 10
Ім’я	Текстовий	Розмір поля: 10
По-батькові	Текстовий	Розмір поля: 10
Адреса	Текстовий	Розмір поля: 10 Значення за замовчуванням: м. Київ
Домашній телефон (Ключове поле)	Числовий	Маска вводу: 200-00-00
Мобільний телефон	Текстовий	Маска вводу: 38-000-00

Щоб Microsoft Access міг зв'язати дані з різних таблиць, кожна таблиця повинна містити поле або набір полів, які будуть задавати індивідуальне значення кожного запису в таблиці. Таке поле чи набір полів називають основним ключем. Для поля *Домашній телефон* поставимо ключ і задаємо Маску введення 200-00-00 (рис. 2.1).

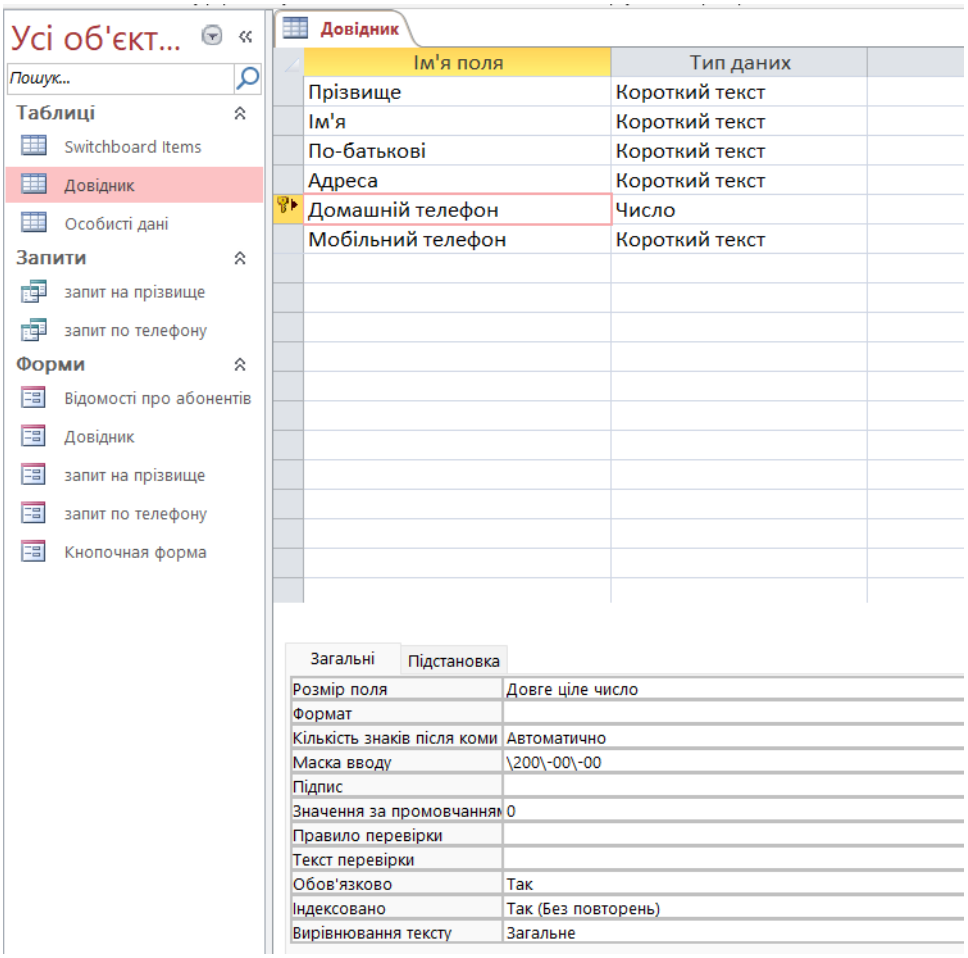


Рис. 2.1 - Створення таблиці «Довідник»

Далі закрийте конструктор та збережіть таблицю під назвою *Довідник*. Так само створимо другу взаємопов'язану таблицю. У ній зберігатимуться особисті дані абонентів. Задаймо для неї поля з таблиці 2.2. Для поля Домашній телефон знову поставимо ключ і задаємо ту саму маску вводу. Потім збережемо таблицю під назвою *Особисті дані*.

Таблиця 2.2

Дані таблиці «Особисті дані»

Ім’я поля	Тип даних	Властивості поля
1	2	3
Домашній телефон (Ключове поле)	Числовий	Маска вводу: 200-00-00
Номер паспорту	Числовий	Розмір поля: Довге ціле
Дата народження	Дата/час	Формат поля: короткий формат дати

Місце народження	Текстовий	Розмір поля: 10 Значення за замовчуванням: м. Київ
Освіта	Текстовий	Розмір поля: 10
Місце роботи	Текстовий	Розмір поля: 10
Посада	Текстовий	Розмір поля: 10

Після розподілу даних по таблицях та визначення ключових полів необхідно вибрати схему для зв'язку даних у різних таблицях. Для цього необхідно визначити зв'язки між таблицями. Для цього виберемо пункт меню *Робота з базами даних – Схема даних*. Додамо наші таблиці *Довідник* та *Особисті дані*. З'єднаємо ці таблиці по полю *Домашній телефон*. Для цього перетягніть, утримуючи ліву кнопку миші, поле *Домашній телефон* із однієї таблиці в іншу. У запиті зв'язку відзначимо пункти меню: *забезпечення цілісності даних, каскадне оновлення пов'язаних полів і каскадне видалення пов'язаних полів.*

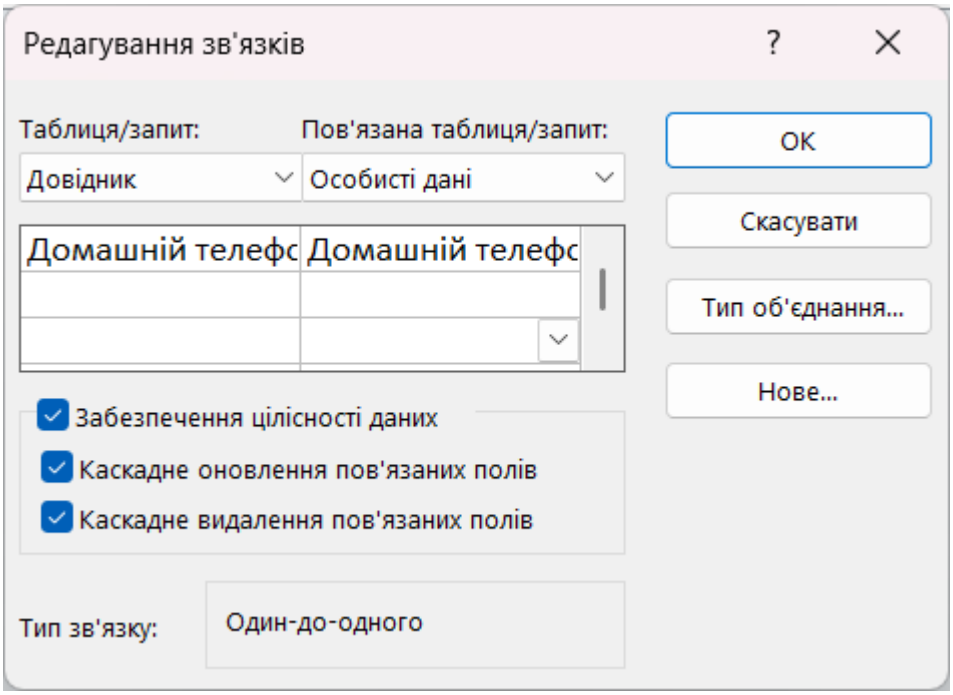


Рис. 2.2 - Створення зв'язку між таблицями

Після цього натискаємо кнопку *Створити* і у нас має з'явитися зв'язок між таблицями «*один до одного*».

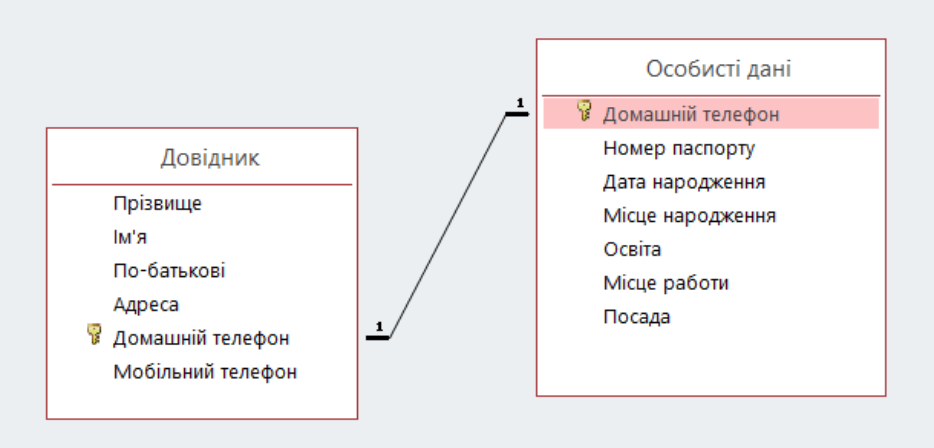
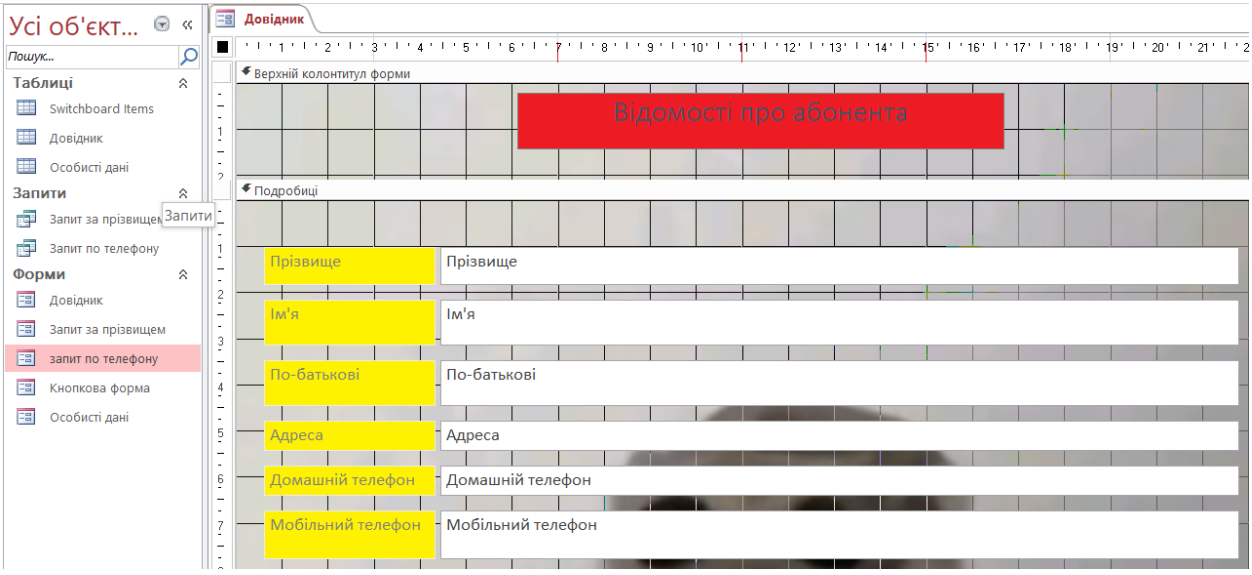


Рис. 2.3 - Схема даних

Завдання 3. Створення автоматичної форми «*Відомості про абонентів*» На вкладці *Створення* виділяємо таблицю *Довідник* і натисніть кнопку *Форма*. З'явиться автоматична форма, яка включає всі поля таблиці *Довідник*. Наступним кроком є розміщення елементів керування в логічному порядку. Перейдіть до режиму конструктора. Спочатку змінимо розміри форми, розсунувши границі до “розумних меж”. На панелі *Елементи керування*:

· У заголовку форми змініть назву на *Відомості про абонента*;

· після цього за допомогою піктограм *Панелі інструментів* змініть



колір напису, розмір шрифту та виділіть його жирним шрифтом;

Рис. 2.4 - Вікно редагування форми

· клацніть на панелі *Елементи керування* ***Прямокутник*** і розмістить в області даних на формі, задайте потрібні розміри. Клацніть кнопку *Сторінка властивостей на* вкладці *Конструктор* у групі *Сервіс*, у меню перейдіть на вкладку *Макет* і задайте властивість *Оформлення – припідняте*. За необхідності можна змінити колір фону;

· помістіть текстові вікна (у яких користувач вводитиме текстові дані) у цей прямокутник;

· змініть колір та шрифт цих текстових вікон.

Завдання 4. Створення кнопок на формі «Відомості про абонентів» Створимо кнопки на формі для переміщення за записами. Створимо кнопку *Наступний запис* за допомогою Майстра:

· відкриємо форму в режимі Конструктор;

· на панелі *Елементи управління* виділимо об'єкт *Кнопка* та перетягнемо його на форму;

· у меню виберемо категорію *Переходи за записами* і дію *Наступний запис*, натиснемо кнопку *Далі*;

· у цьому вікні ставимо прапорець у меню *Текст* і пишемо *Наступний запис* (цей напис буде відображатися на кнопці), і тиснемо кнопку *Готово*;

· за допомогою піктограм на панелі інструментів можна змінити колір та розмір напису на кнопці.

Аналогічно створюються кнопки *Попередній запис* та *Додати запис* (при цьому тільки використовуються інші категорії та дії). Їх можна розмістити у будь-якому місці форми. Також потрібно створити кнопку пошуку в полі запису за допомогою стандартних засобів Access: для цього вибираємо дію *Знайти запис* із категорії *Переходи по записам*.

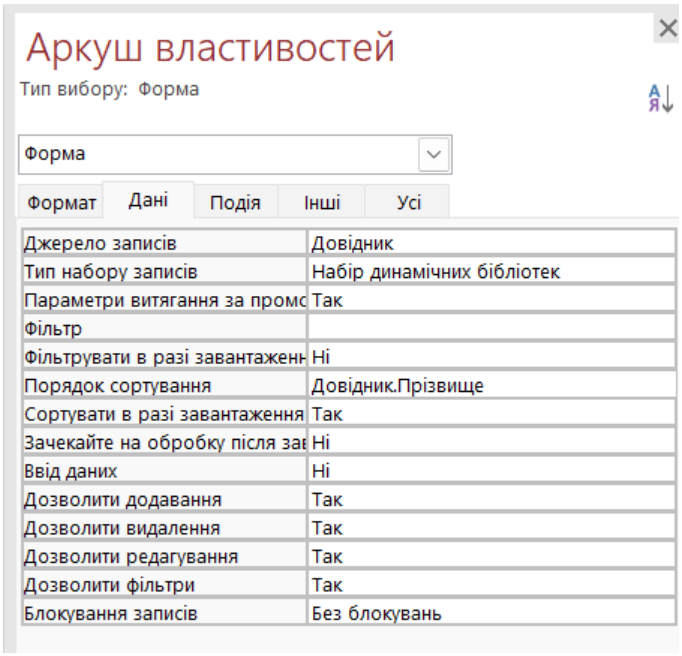
Ще зробимо кнопку для закриття форми, вибравши дію *Закриття форми* із категорії *Робота з формою*.

Тепер потрібно створити кнопку для оновлення даних у формі. Під час роботи з базою даних у мережі створення подібної кнопки дозволяє переглянути останню версію існуючих записів. Щоб відобразити всі оновлені

записи, включаючи нові, можна перезапитувати записи. Щоб створити цю кнопку, виберіть пункт *Оновити дані форми* з категорії *Робота з формою*.

Завдання 5. Встановлення властивостей форми «Відомості про абонентів».

Клацніть кнопку *Сторінка властивостей* на вкладці *Конструктор* у групі *Сервіс* (у заголовку вікна, що з'явиться, має відображатися напис *Форма*). По-перше, встановимо порядок фільтрації для записів таблиці відомостей. Для цього перейдемо на вкладку *Дані*, у пункті *Застосування фільтрів* поставимо *Так*, а потім у пункті *Порядок сортування* напишемо ***Довідник.Прізвище***. Тепер при відкритті цієї форми записи будуть



сортуватись у порядку зростання прізвищ.

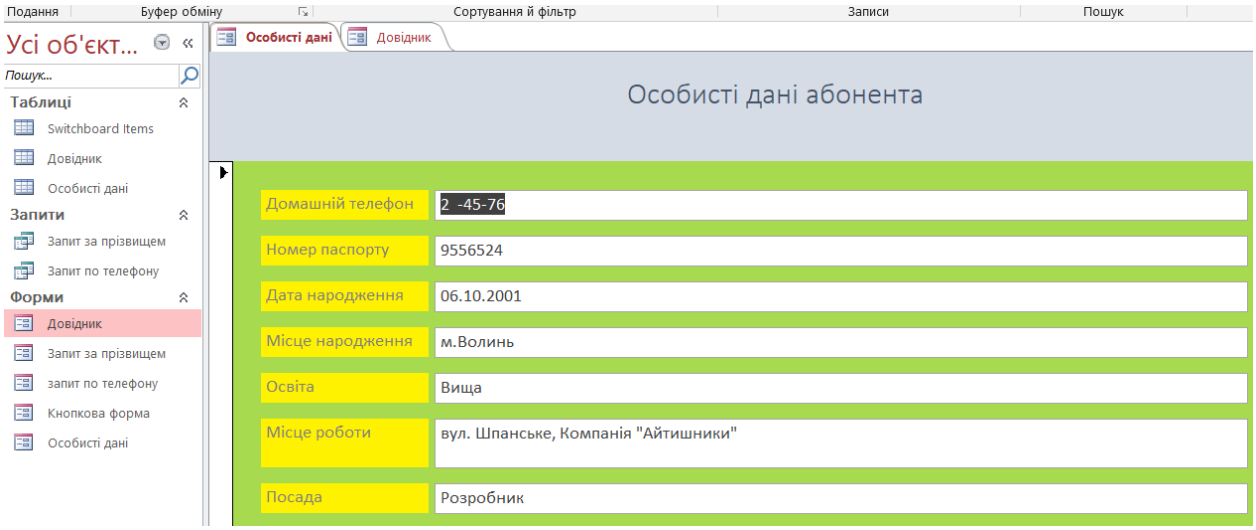
Рис. 2.5 - Встановлення властивостей форми

По-друге, зробимо цю форму спливаючою (випливаюча форма завжди розташовується над іншими вікнами Microsoft Access). Для цього у властивостях у пункті *Інші* введіть значення *Так* у комірці *спливаюче вікно*. По-третє, у пункті *Макет* у комірці *Тип границі* виберіть *Тонка*, якщо потрібно заборонити зміну розмірів форми; у протилежному випадку перейдіть до наступного кроку. Якщо встановлено значення *Тонка*, можна переміщати спливаючу форму, але не можна змінювати її розміри. По-

четверте, приберемо смуги прокручування та кнопки розмірів вікна. Для цього в пункті *Макет* в комірку *смуги прокручування* поставимо *Відсутні* і в комірку *кнопки розмірів вікна* введемо значення *Відсутні*. Ця форма буде ще модернізована надалі. А поки що закриємо її та збережемо під ім'ям *Відомості про абонентів*.

Завдання 6. Створення автоматичної форми «Особисті дані»

Так само можна зробити форму *Особисті дані*, використовуючи дані з таблиці *Особисті дані*. Але в цьому випадку не потрібно робити клавіші переходу по записах, оскільки ця форма використовуватиметься лише для введення (виведення) інформації про відповідного абонента *форми Відомості про абонентів*. Ці дві форми будуть пов'язані ключовим полем



Домашній телефон.

Рис. 2.6 - Форма "Особисті дані"

На формі ми розмістимо всі поля з таблиці *Особисті дані*, а з кнопок нам знадобиться тільки кнопка закриття форми. У формі *Особисті дані* задаємо такі ж властивості, як і у формі *Відомості про абонентів*, крім *порядку фільтрації та сортування*!

Завдання 7. Створення кнопки, яка зв'язує форми «*Відомості про абонентів*» та «*Особисті дані*»

Для того, щоб полегшити заповнення форми *Особисті дані*, ми створимо кнопку на формі *Відомості про абонентів*, при натисканні на яку

спливає форма *Особисті дані*, в яку і вводяться відомості про відповідного абонента. Зробимо наступним чином:

· відкриємо форму *Відомості про абонентів* у режимі Конструктор;

· на панелі *Елементи керування* виділимо об'єкт *Кнопка* та перетягнемо його на форму;

· у меню вибираємо категорію *Робота з формою – Відкриття форми* натискаємо кнопку Далі;

· вибираємо форму *Особисті дані* та натискаємо Далі;

· ставимо прапорець в пункт *Відкрити форму для відібраних записів* і натискаємо Далі;

· вибираємо з форми *Відомості про абонентів* та з форми *Особисті дані* поле *Домашній телефон* натискаємо кнопку < - > , а потім на кнопку Далі;

· ставимо прапорець у пункт *текст*, вводимо напис *Особисті дані* та тиснемо Готово.

Тепер при натисканні на цю кнопку з'являтиметься форма *Особисті дані* для відповідного абонента.

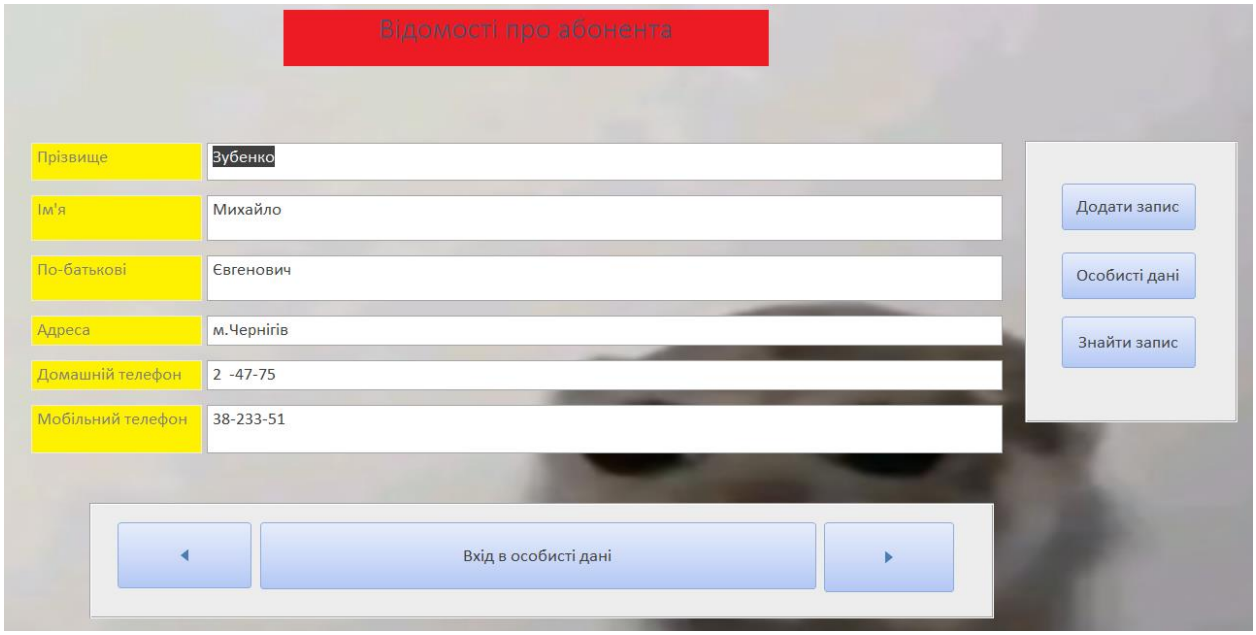


Рис. 2.7 - Форма "Дані про абонента"

Завдання 8. Створення запитів

Ми будемо використовувати запити для виведення інформації про відповідного абонента. Створимо запити за допомогою *Майстра запитів*:

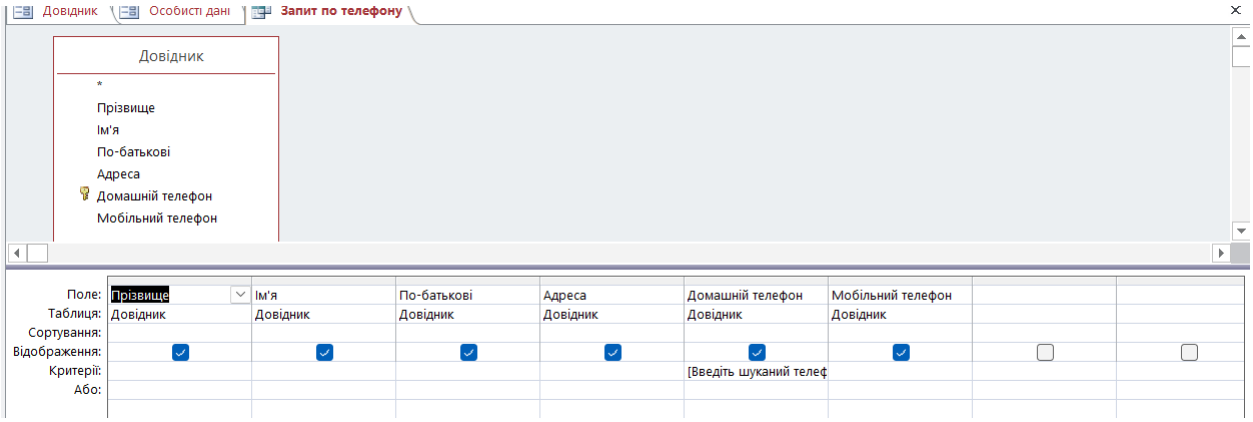
· перейдемо на вкладку *Створення* групи *Запити* виберемо *Майстер запитів*;

· натискаємо кнопку *Майстер запитів*, вибираємо *Простий запит* та натискаємо ОК;

· як джерело запиту виберемо таблицю *Довідник* і переміщуємо всі доступні поля, потім натискаємо кнопку Далі;

· задаємо ім'я *Запит по телефону*, ставимо прапорець у комірці *Зміна структури запиту* та натискаємо кнопку Готово;

· у режимі конструктора в полі Телефон в комірку *Умову відбору* записуємо *[Введіть шуканий телефон]*. Тепер щоразу з відкриттям запиту



з'являтиметься вікно, в якому запитуватиметься шуканий телефон.

Рис. 2.8 - Створення запиту на вибірку (параметричного запиту)

Абсолютно аналогічно створюється *запит на прізвище*. Тільки в режимі конструктора в полі *Прізвище* в комірку *Умову відбору* записуємо *[Введіть шукане прізвище]*.

Завдання 9. Створення форм «*Пошук по телефону*» та «*Пошук за прізвищем*»

На основі створених запитів створимо форми *Пошук за телефоном* та *Пошук за прізвищем*. Створюються вони за аналогією з формою *Відомості про абонентів*. Оформлення їх залежить від фантазії творця, але в жодному

разі не варто переборщувати з квітами та картинками. Форму знову ж таки зробимо спливаючою з тонкою границею та приберемо смуги прокручування та кнопки зміни розмірів вікна. І зробимо кнопку закриття форми.

Для форми *Пошук на прізвище* зробимо ще кнопку *наступний запис*, щоб переглянути всі знайдені відомості з шуканим прізвищем. Для виконання поставленого завдання створимо кнопку, вибравши дію *Наступний запис* із категорії *Переходи за записами*.

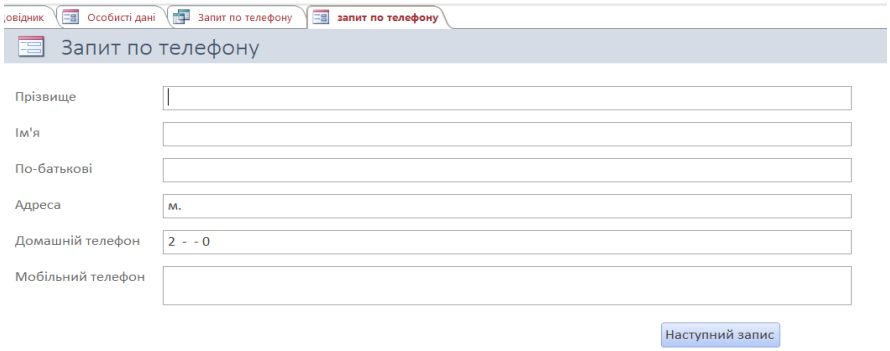


Рис.2.9 - Форма «Пошук на прізвище»

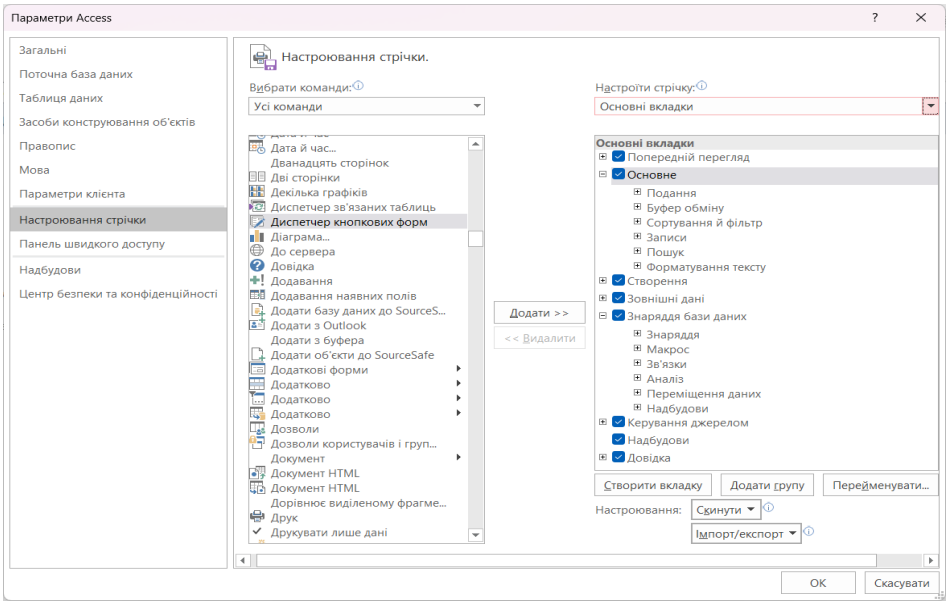
Завдання 10. Введення даних

У таблицях *Довідник* та *Особисті дані* введіть інформацію щодо 10 співробітників підприємства.

Завдання 11. Створення головної форми кнопки за допомогою диспетчера

MS Access дає можливість за допомогою диспетчера кнопкових форм створювати кнопкову форму стандартного вигляду. Створимо головну форму кнопки, яка буде використовуватися як панель управління БД «Телефонний довідник співробітника».

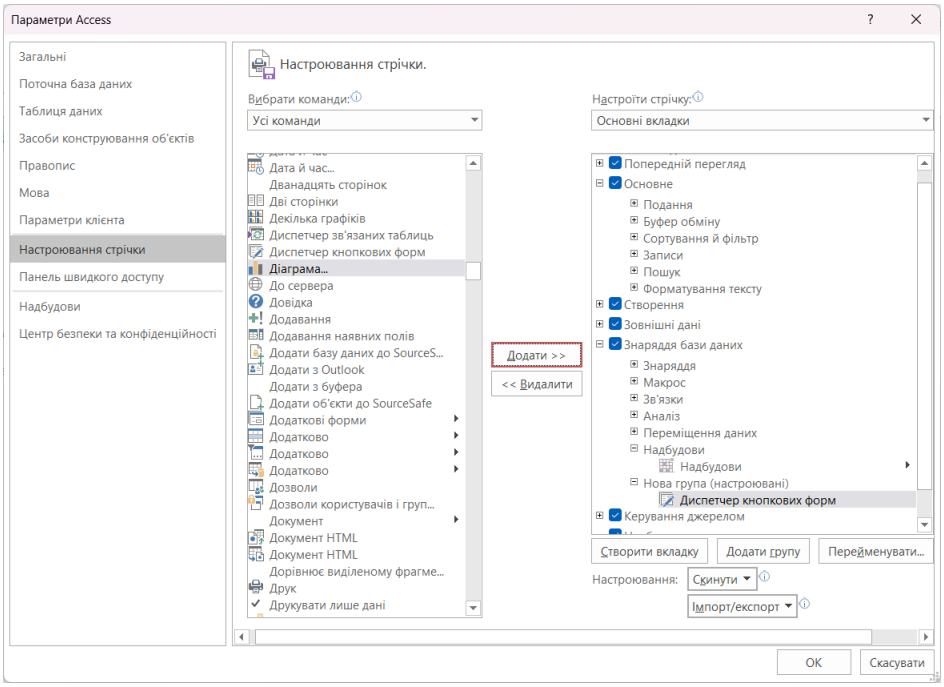
Перед початком роботи перевірте наявність кнопки *Диспетчер кнопкових форм* на вкладці *Знаряддя бази даних*. Якщо кнопка відсутня, зробіть такі дії: на вкладці *Файл* вибираємо *Параметри*. Відкриється вікно



Параметри Access (рис. 2.10).

Рис. 2.10 - Робота з параметрами

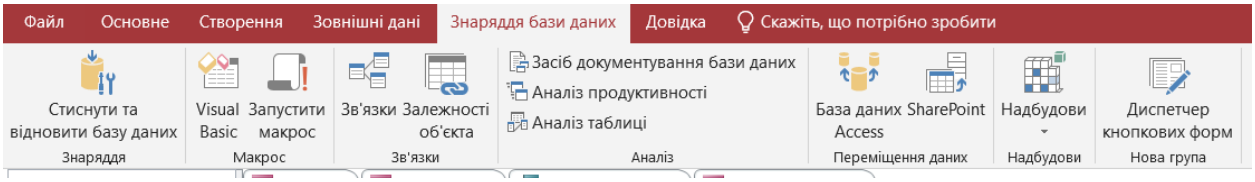
Далі вибираємо *Налаштування стрічки*. У списку праворуч вибираємо *Робота з базами даних* та натискаємо на кнопку *Створити групу*. У лівому списку вибираємо *Диспетчер кнопкових форм* та натискаємо кнопку *Додати*.



В результаті вікно діалогу має змінитись так, як показано на рис. 2.11.

Рис. 2.11 - Робота з параметрами (продовження)

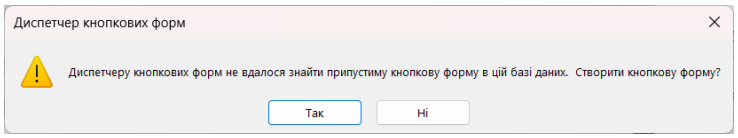
Тепер на вкладці **Робота з базами даних** у групі *Нова група* натисніть



кнопку *Диспетчер кнопкових форм* (рис. 2.12).

Рис. 2.12 - Вкладка Робота з базами даних, група Нова група

У вікні Диспетчер кнопкових форм підтвердимо створення кнопкової



форми (рис. 2.13).

Рис. 2.13 - Вікно підтвердження створення кнопкової форми програми

З'явиться вікно з рядком *Головна кнопкова форма* (рис.2.14), у якому формується список кнопкових форм різних рівнів.

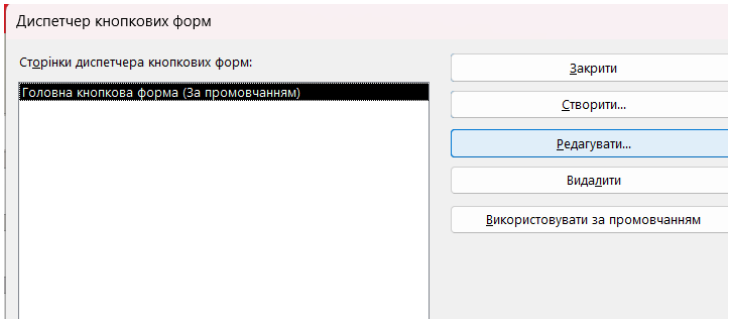


Рис. 2.14 - Вікно списку кнопкових форм програми

Створимо три кнопкові форми для комплексу задач телефонного довідника.

Для створення кнопкової форми у вікні диспетчера кнопкових форм натисніть кнопку *Створити*. У вікні *Створення* (рис. 2.15) у полі *Ім'я сторінки кнопкової форми* введемо ім'я першої кнопкової форми «Кнопкова форма Перегляд та редагування записів».

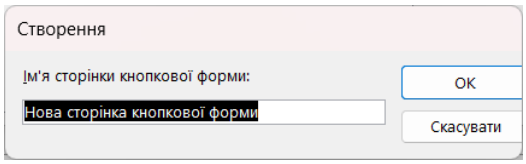


Рис. 2.15 - Вікно створення порожньої форми кнопки

Після натискання кнопки ОК у вікні диспетчера кнопкових форм з'явиться рядок «Кнопкова форма Перегляд і редагування записів».

Аналогічними діями створимо ще дві кнопкові форми: *Кнопкова форма Пошук за прізвищем* та *Кнопкова форма Пошук за телефоном* (рис. 2.16).

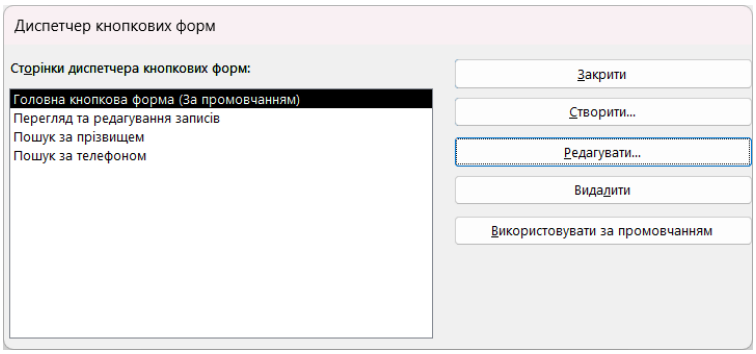


Рис. 2.16 - Список кнопкових форм БД «Телефонний довідник співробітника»

Будь-яку з підготовлених кнопкових форм можна зробити стартовою, яка відкриватиметься за замовчуванням під час відкриття бази даних. Для

цього слід вибрати її ім'я у вікні диспетчера кнопкових форм та натиснути кнопку *За замовчуванням*. Ми залишимо головну кнопкову форму стартової кнопковою формою. Ця кнопкова форма буде знаходитись на верхньому рівні ієрархії взаємопов'язаних кнопкових форм відповідно до структури створюваної програми.

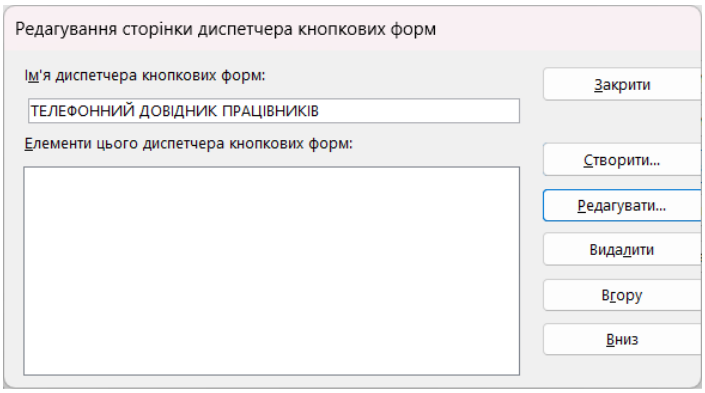
Елементами кнопкових форм є кнопки з підписами. Підпис задається користувачем і повинен наскільки можна коротко і точно називати дії, які будуть виконуватися при натисканні кнопки. Для формування елементів головної кнопкової форми у вікні *Диспетчер кнопкових форм* виділимо відповідний рядок і натиснемо кнопку *Змінити*.

У вікні діалогу *Зміна сторінки кнопкової форми* (рис. 2.17) головну форму кнопки перейменуємо в *ТЕЛЕФОННИЙ ДОВІДНИК СПІВРОБІТНИКІВ*, ввівши це ім'я в поле *Назва кнопкової форми*.

Елементи, створювані в кнопковій формі, можуть бути поділені на два основні типи:

· Елементи, призначені для організації виклику інших кнопкових форм;

· Елементи, що забезпечують відкриття форми, звіту, запуск макросу програми, вихід із додатка із закриттям бази даних або переходу в режим



продовження розробка кнопкової форми диспетчером кнопкових форм.

Рис. 2.17. Вікно редагування імені кнопкової форми та створення її кнопок

Для формування в головній формі кнопкової *БД Телефонний довідник працівників* кнопки виклику підлеглої кнопкової форми натиснемо кнопку

Створити.

У вікні *Зміна елемента кнопкової форми* в рядку *Команда* виберемо *Перехід до кнопкової форми* (рис. 2.18).

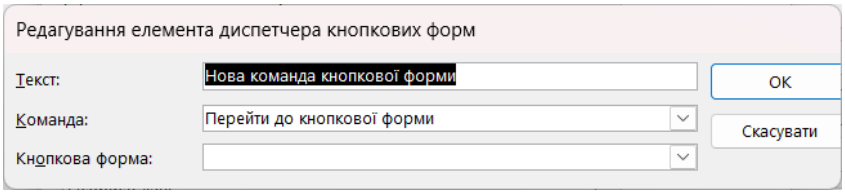


Рис. 2.18 – Вікно створення елемента кнопкової форми

У рядку *Кнопкова форма* вибирається форма, перехід до якої повинна забезпечувати кнопка поточної форми, що створюється. Відкриємо список створених для програми кнопкових форм і оберемо елемент *Кнопкова форма Пошук за телефоном*. У рядку *Текст* введемо підпис цієї кнопки: «Пошук співробітника по телефону» (рис. 2.19).

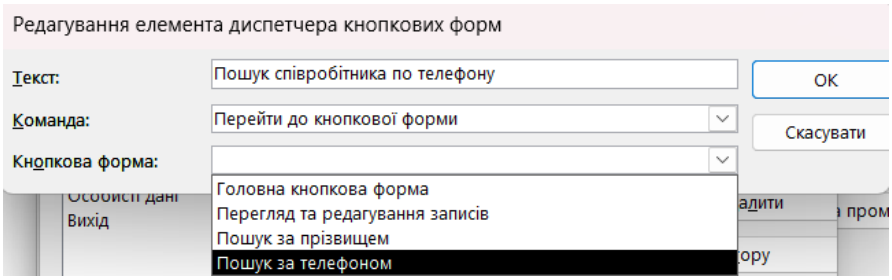


Рис. 2.19 - Вибір підлеглої кнопкової форми

Для завершення формування елемента натисніть кнопку ОК. Елемент з'явиться у списку *Елементи цієї кнопкової форми*.

Аналогічно створимо кнопки виклику інших підлеглих форм і задамо для них підписи «*Пошук співробітника за прізвищем*» та «*Перегляд та редагування записів довідника*». Щоб змінити або видалити будь-яку зі створених кнопок, потрібно вибрати ім'я цієї кнопки у списку *Елементи цієї кнопкової форми* та натиснути кнопку *Змінити* або *Видалити*. При необхідності змінити порядок кнопок у списку потрібно вибрати елемент та натиснути кнопку *Вгору* або *Вниз*.

Для переходу до редагування створених форм кнопок створимо в головній формі кнопку *Зміна кнопкової форми*, вибравши для неї команду *Конструктор програми*. Надалі це дозволить будь-якої миті викликати

диспетчер кнопкових форм і з його допомогою внести необхідні зміни.

Для завершення роботи з програмою створимо кнопку *Вийти з довідника*, вибравши для нього команду *Вийти з програми*, і надамо їй це ж ім'я.

Сторінка головної кнопкової форми *БД Телефонний довідник співробітників* після створення всіх елементів представлена на рис. 2.20.

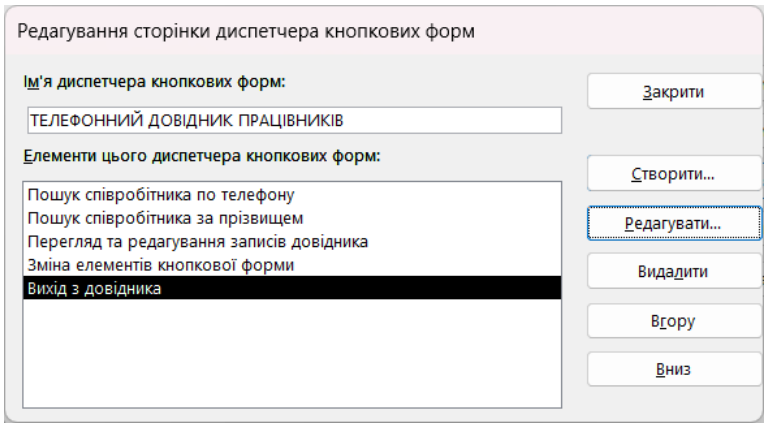


Рис. 2.20 - Сторінка кнопкової форми БД Телефонний довідник співробітників

Закінчивши створення сторінки кнопкової форми, натисніть кнопку *Закрити*. Це дозволить повернутися до списку всіх кнопкових форм програми. У підлеглих формах кнопок створимо кнопку для повернення до головної кнопкової форми, заповнивши поля у вікні *Зміна елемента кнопкової форми*, як показано на рис. 2.21.

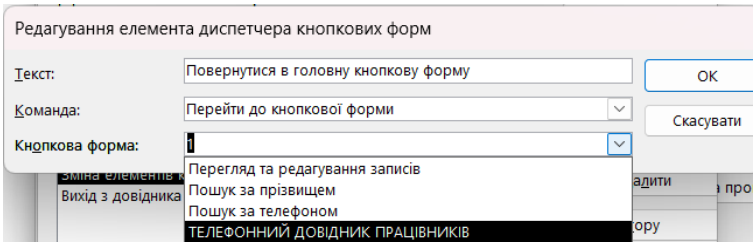


Рис. 2.21 – Створення кнопки для повернення в головну форму

Таким чином, можуть бути встановлені всі необхідні зв'язки між формами кнопок, представленими в списку вікна диспетчера кнопкових форм. Практично на ці зв'язки та кількість рівнів не накладаються обмеження і можуть бути створені будь-які зручні користувачеві переходи за

кнопковими формами.

Для формування кнопок, що забезпечують виконання задач користувача, необхідно, щоб вони були пов'язані з одним із об'єктів програми. Диспетчер кнопкових форм забезпечує зв'язок із формами, звітами, макросами та модулями.

Для зв'язку створюваної кнопки з потрібним об'єктом треба вибрати в рядку *Команда* (рис. 2.22) одну з команд:

• Відкрити форму для додавання

• Відкрити форму для зміни

• Відкрити звіт

• Виконати макрос

• Виконати програму

У третьому рядку вікна *Зміна елемента кнопкової форми*, яка буде відповідати команді, вибраній у другому рядку, виберемо конкретний об'єкт.

Наприклад, якщо вибрати в рядку *Команда* команду - *Відкрити звіт*, з'явиться третій рядок - *Звіт*, де можна буде вибрати ім'я звіту, що відкривається. Якщо вибрати в рядку *Команда*- команду *Відкрити форму для зміни*, з'явиться рядок *Форма*.

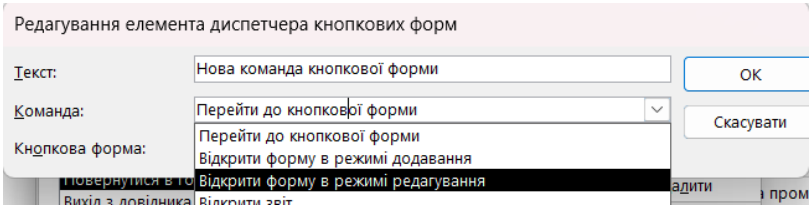


Рис. 2.22 - Вибір команди для елемента кнопкової форми

Створимо в кнопковій формі *Пошук співробітника за телефоном* кнопку для роботи з формою *Пошук за телефоном*, через яку здійснюється пошук даних співробітників.

Для кнопкової форми *Перегляд та редагування записів співробітників* кнопку для роботи з формою *Відомості про абонентів*. Аналогічним чином можуть бути створені всі необхідні кнопкові форми кнопки для виклику форм, звітів, макросів і програм користувача. Головна кнопкова форма *БД*

Телефонний довідник працівників представлений на рис. 2.23.

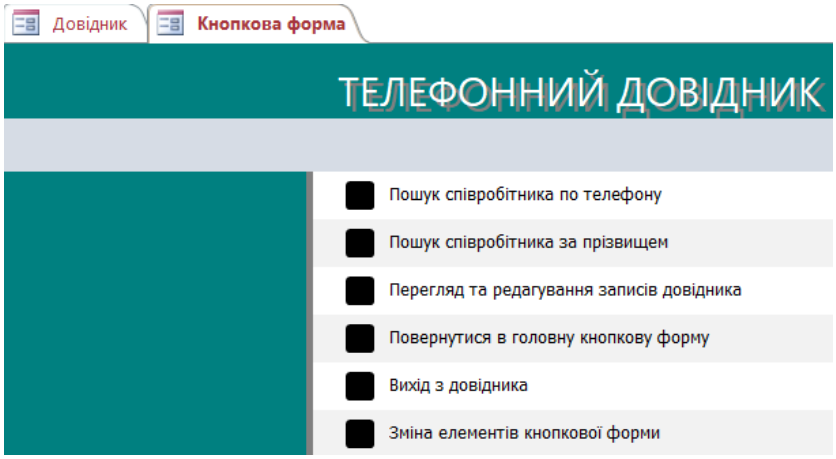


Рис. 2.23 - Головна форма кнопки БД Телефонний довідник співробітників

Після завершення роботи з диспетчером кнопкових форм у списку форм бази даних з'являється форма з ім'ям *Кнопкова форма*.

Головна кнопкова форма може запускатися під час відкриття бази даних.

Для цього натисніть кнопку *Файл*, а потім виберіть пункт *Параметри*. У діалоговому вікні Параметри Access клацніть *Поточна база даних*. У *формі перегляду* виберіть *Головну форму кнопки.*

Ця команда доступна при відкритій базі даних. Параметри запуску визначають вигляд вікна програми під час його відкриття. Ці параметри дозволяють змінити заголовок вікна програми, вибрати власне меню, контекстне меню, яке використовується за замовчанням у формах та звітах, панелі інструментів, а також вказати форму, що відкривається у базі даних за умовчанням.

За допомогою параметрів запуску можна заборонити користувачеві роботу у вікні бази даних, і він зможе виконувати лише ті роботи, які передбачені в кнопкових формах програми.

Диспетчер кнопкових форм Access створює таблицю *Елементи кнопкової форми*, що містить опис кнопок форми та виконуваних ними дій.

Спроба змінити форму кнопки в режимі конструктора форми може призвести до того, що програма перестане працювати. Однак вставка в

форму кнопки малюнків не призводить до таких наслідків.

Практична роботи № 3

MS Access. Створення БД «Ринки збуту»

Створити базу даних «Ринки збуту», що складається з трьох таблиць із взаємозалежними даними, запитами, формами, що використовуються для наочної роботи з даними (введення, редагування тощо), а також зі звітами.

Завдання 1. Самостійно створити нову базу даних під назвою «Ринки збуту». Після створення порожньої бази даних необхідно створити об'єкти цієї бази даних.

Завдання 2. Створення таблиць.

1. Створіть базу даних Ринки збуту, що складається з 3 таблиць. У режимі Конструктор створіть три таблиці та введіть дані з таблиць 3.1 – 3.3:

Таблиця 3.1

Дані таблиці «Релігія»

Ім'я поля	Тип даних
Код релігії (Ключове поле)	Лічильник або Числовий
Релігія	Текстовий

Таблиця 3.2

Дані таблиці «Державний устрій»

Ім'я поля	Тип даних
Код державного устрою (Ключове поле)	Лічильник або Числовий
Державний устрій	Текстовий

Таблиця 3.3

Дані таблиці «Країни Європи»

Ім'я поля	Тип даних
Код країни (Ключове поле)	Лічильник або Числовий
Країна	Текстовий

Столиця	Текстовий
Площа	Числовий
Населення	Числовий
Грошова одиниця	Текстовий
Код релігії	Числовий
Код державного устрою	Числовий

Завдання 3. Створіть зв'язок між таблицями: таблиці *Релігія та Країни Європи* зв'яжіть по полю *Код релігії*, а таблиці *Державний устрій* та *Країни Європи* по полю *Код державного устрою*. При цьому увімкніть режими забезпечення цілісності даних, каскадне оновлення та каскадне видалення.

Завдання 4. Створення форм. Створіть дві одиночні форми для заповнення таблиці *Релігія* та *Державний устрій*. Заповніть їх відомостями, наведеними нижче.

Державний устрій: Республіка, Князівство, Монархія, Герцогство.

Релігія: Атеїзм, Християнство (православні), Християнство (католики), Християнство (протестанти).

У режимі *Конструктора* розробіть форму заповнення таблиці *Країни Європи*, при цьому поля *Код релігії* та *Код державного устрою* повинні являти собою поля зі списком, щоб при заповненні таблиці ви могли користуватися вже введеними даними з таблиць *Релігія* та *Державний устрій*. Після створення цих полів необхідно відредагувати їх властивості – у категорії *Дані* у властивості *Дані* встановити значення *Код релігії* та *Код державного устрою* відповідно.

Заповніть таблицю *Країни Європи* відомостями, наведеними у таблиці 3.4.

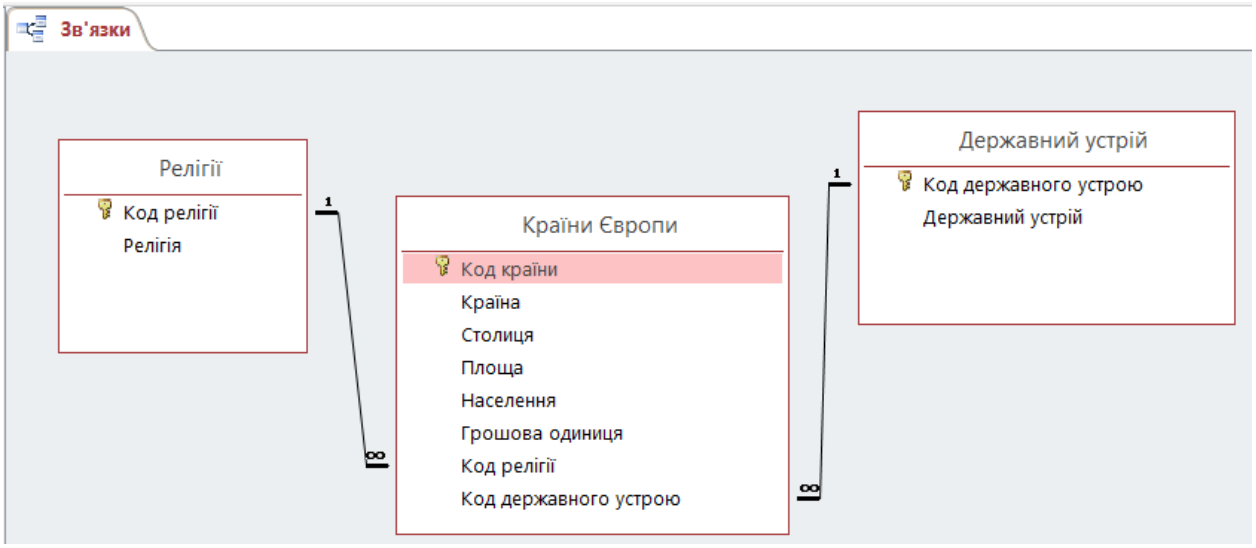
Таблиці 3.4

Дані для вводу

Країна	Площа ($км^2$)	Столиця	Кількість жителів	Основна релігія	Грошова од. до вступу в ЄС	Державний устрій
Австрія	83 857	Відень	7 557 000	Католики	Шилінг	Республіка
Албанія	28 748	Тирана	3 149 000	Атеїзм	Лек	Республіка
Андорра	468	Андорра-ла-Велья	51 400	Католики	Франк	Князівство
Бельгія	30 518	Брюссель	9 865 000	Католики	Франк	Монархія
Болгарія	110 994	Софія	8 978 000	Православні	Лев	Республіка
Данія	43 092	Копенгаген	5 130 000	Протестанти	Крона	Монархія
Ірландія	70 285	Дублін	3 553 000	Католики	Фунт	Республіка
Фінляндія	338 145	Гельсінкі	4 952 050	Католики	Франк	Республіка
Греція	131 957	Афіни	10 055 000	Православні	Драхма	Республіка
Нідерланди	41 863	Амстердам	14 741 000	Католики	Флорін	Монархія
Норвегія	323 878	Осло	4 202 000	Протестанти	Крона	Монархія
Польща	312 683	Варшава	37 864 000	Католики	Злота	Республіка
Португалія	92 389	Лісабон	10 349 000	Католики	Ескудо	Республіка
Румунія	237 500	Бухарест	23 014 000	Православні	Лей	Республіка
Сан-Марино	61	Сан-Марино	22 830	Католики	Ліра	Республіка
Швейцарія	41 293	Берн	6 626 000	Католики	Франк	Республіка
Ісландія	103 000	Рейк'явік	248 000	Протестанти	Крона	Республіка
Італія	301 277	Рим	57 401 000	Католики	Ліра	Республіка
Ліхтенштейн	160	Вадуц	27 840	Католики	Франк	Монархія
Люксембург	2 586	Люксембург	372 000	Католики	Франк	Герцогство
Мальта	316	Валлетта	347 000	Католики	Фунт	Республіка
Монако	2	Монако	28 000	Католики	Франк	Князівство
Іспанія	504 783	Мадрид	38 996 000	Католики	Песета	Монархія
Великобританія	244 110	Лондон	57 006 000	Протестанти	Фунт	Монархія
Швеція	449 964	Стокгольм	8 415 000	Протестанти	Крона	Монархія
Угорщина	93 031	Будапешт	10 591 000	Католики	Форинт	Республіка

Німеччина	357 042	Берлін	77 370 000	Протестанти	Марка	Республіка

Завдання 5. Створення запитів. Створіть запит у режимі Конструктора. При цьому додайте у запит усі три таблиці. Переконайтеся, що між доданими



таблицями встановлено зв'язок, показаний на рисунку 3.1.

Рис. 3.1 – Створення запиту

Тепер потрібно вибрати поля для запиту. З таблиці *Країни Європи* візьмемо поля: ***Країна, Столиця, Площа, Населення, Грошові одиниці.*** З таблиці *Державний устрій* поле ***Державний устрій***, а з таблиці *Релігія* поле ***Релігія***. Налаштуйте запит на різні умови вибірки. Зазвичай при цьому використовують один запит, змінюючи ці умови. Але для перевірки виконаної роботи створіть декілька запитів (на кожну умову по запиту):

· запит, що виводить країни з Православ'ям;

· запит, що не виводить країни з Православ'ям;

· запит, що виводить країни з населенням більше 10000000 і менше 200000000;

· запит, що виводить країни з населенням більше 3000000 і площею менше 30000 кв. км.;

· запит, що запитує: країни з якою грошовою одиницею вивести на екран (запит з параметром);

· запит, що виводить країни з грошовою одиницею, що містить літеру **к.**

Для того, щоб вивести в запиті лише країни з певною релігією, необхідно в полі запиту *Релігія* в рядок *Умова відбору* ввести цю релігію. Аналогічно будь-якого поля. Для того, щоб заборонити країнам з певною релігією виводитися в запиті, необхідно в цьому полі та вказаному рядку ввести задану релігію, але перед нею поставити оператор ***Not.***

Щоб вивести країни з населенням більше 1000000, достатньо в полі *Населення* в рядок *Умова відбору* ввести >1000000. До речі, для умов передбачено два рядки, тому для одного поля можна вводити дві умови.

Для того щоб знаходити в полі не конкретне значення, а лише його фрагмент, використовують оператор ***Like***. Його ставлять попереду заданого фрагмента, а *до або після* фрагмента можна використовувати зірочки маски. Наприклад, для поля *Країна* можна ввести таку умову відбору: ***Like «*р*»***. Результатом виконання такого запиту будуть всі записи, що відповідають країні, що містить у своїй назві літеру **р**. Збережіть базу даних.

Практична робота № 4

Створення БД «Матеріали»

Створити базу даних «Матеріали», що складається з трьох таблиць із взаємозалежними даними, запитами, формами, що використовуються для наочної роботи з даними (введення, редагування тощо), а також звітів.

Хід роботи

Завдання 1. Самостійно створити нову базу даних під назвою «Матеріали». Після створення порожньої бази даних потрібно створити об'єкти цієї бази даних.

Завдання 2. Створення таблиць: 1. У режимі Конструктор створіть три таблиці та введіть дані з таблиць 3.1 – 3.3.

Таблиця 3.1

Дані таблиці «Номенклатура матеріалів»

Ім'я поля	Тип даних	Властивості поля
Код матеріалу (Ключове поле)	Лічильник	
Матеріал	Текстовий	Розмір поля: 20

Таблиця 3.2

Дані таблиці «Постачальники»

Ім'я поля	Тип даних	Властивості поля
Код постачальника (**Ключове поле**)	Лічильник	
Постачальник	Текстовий	Розмір поля: 20

Таблиця 3.3

Дані таблиці «Постачання матеріалів»

Ім'я поля	Тип даних	Властивості поля
Код постачальника	Числовий	
Код матеріалу	Числовий	
Дата поставки	Дата/час	Формат поля:

		Короткий формат дати Підпис: *Дата поставки*
Одиниця виміру	Текстовий	Розмір поля: 10 Підпис: од. .
Кількість	Числовий	
Ціна	Грошовий	Формат поля: Грошовий

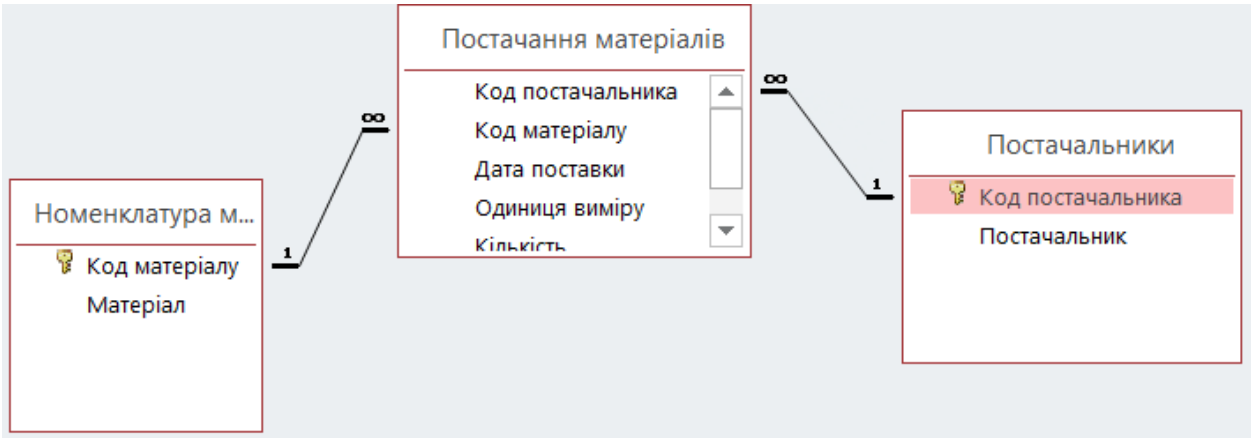
* *ключове поле в таблиці відсутнє*

Завдання 3. Створення схеми даних

Виберіть пункт меню ***Робота з базами даних*** - ***Схема даних***. У вікні *Додавання таблиці*, що з'явилося, вибрати таблицю *Номенклатура матеріалів* і подвійним клацанням лівої кнопки миші або кнопкою *Додати* додати таблицю у вікно *Схема даних*, що знаходиться поки за вікном *Додавання таблиці*. Також додати інші таблиці *Постачання матеріалів* і *Постачальники*. Натиснути кнопку *Закрити*.

У вікні *схема даних* будуть всі три таблиці з полями, ключові поля будуть виділені жирним шрифтом. Лівою копкою миші захопити поле *Код матеріалу* з таблиці *Номенклатура матеріалів* перетягнути його на поле *Код матеріалу* таблиці *Постачання матеріалів* та відпустити копку миші. У вікні, що з'явиться, поставити галочку в поле прапорця **Забезпечення цілісності даних** і поставити галочки в полях прапорців *каскадне оновлення пов'язаних полів і каскадне видалення в'язаних полів*. Дані дії будуть виконуватися автоматично і це забезпечуватиме цілісність і правильність даних у БД. Натиснути кнопку *Створити*. Від таблиці *Номенклатура матеріалів* до таблиці *Постачання матеріалів* з’явилася лінія зі значками 1 і ∞, що означає зв'язок *один-до-одного*. Тобто одному коду матеріалу в таблиці *Номенклатура матеріалів* буде відповідати кілька записів з таким самим кодом матеріалу в таблиці *Постачання матеріалів*.

Також надаємо з полем *Код постачальника* з таблиці *Постачальники*



та перетягуємо його на поле *Код постачальника* з таблиці *Постачання матеріалів*. Зв'язок аналогічний як розглянули вище.

Рис. 3.1. Схема даних

Закриваємо вікно *Схема даних*, на питання збереження відповідаємо так.

Завдання 4. Створення форм для занесення даних у таблиці

Дані можна вводити та використовуючи таблиці, відкривши їх подвійним клацанням лівої кнопки миші, проте в цьому випадку незручно буде заповнювати таблицю Постачання матеріалів, оскільки в перші поля потрібно буде вводити коди матеріалу та постачальника. Для зручності введення даних у таблиці створіть наступні форми ПОСТАЧАЛЬНИКИ, МАТЕРІАЛИ та ДАНІ ПРО ПОСТАВКУ МАТЕРІАЛІВ. Для цього запустіть ***Майстра форм***. Виберете таблицю *Номенклатура матеріалів* з полів даної таблиці оберемо поле *Матеріал* і тиснемо на кнопку «>«, натискаємо ***Далі***, оберіть *Стрічковий*, ***Далі***, введіть найменування форми МАТЕРІАЛИ в поле і натисніть кнопку ***Готово***. Введіть назви матеріалів. Закрийте форму. Внесіть у форму дані таблиці 3.4.

Таблиця 3.4

Дані для введення у форму Матеріал

№п/п	Найменування матеріалу
1	Процесори

2	Системи охолодження
3	Материнські плати
4	Модулі пам'яті
5	Відеокарти
6	Жорсткі диски
7	Корпуси
8	Блоки Живлення
9	Приводи
10	Звукові карти

Також створіть форму ПОСТАЧАЛЬНИКИ, оберіть таблицю *Постачальник* – поле *Постачальник*. Введіть назви постачальників. Закрийте форму. Внесемо у форму дані таблиці 3.5.

Таблиця 3.5

Дані для введення в форму Постачальники

№п/п	Найменування матеріалу
1	Діджітек
2	Іліон Технолоджі
3	USN Computers
4	ІнформПортал
5	Dom Nouta

Для створення форми ДАНІ ПРО ПОСТАВКУ МАТЕРІАЛІВ оберіть Майстер форм, оберіть таблицю *Постачання матеріалів* та оберіть поля *Дата поставки, Кількість, Ціна, Одиниця виміру.* Далі, виберіть зовнішній вигляд форми *в один стовпець* і натисніть кнопку *Далі*, наберіть ім'я форми *ДАНІ ПРО ПОСТАВКУ МАТЕРІАЛІВ* і перейдіть в режим *Змінити макет форми*.

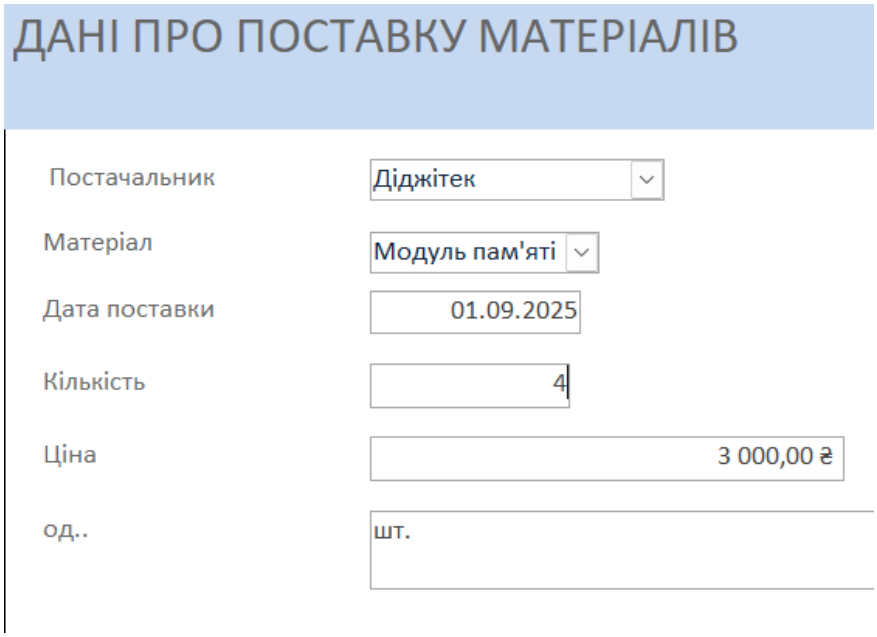
Розсуньте область даних зачепивши мишею нижню межу і почніть рух поля та напису відповідно вниз розташувавши їх при цьому в тому порядку як зручніше вводити дані. Пересувати написи і поля можна виділивши їх і

зачепивши мишею, коли курсор набуде вигляду долоні. Регулюємо розмір полів, виділивши поле і зачепивши відповідну межу поля, коли курсор миші набуде вигляду двох стрілок.

На звільнене верхнє місце ставимо Поле зі списком (), у вікні, що залишилося, залишаємо Об'єкт *«поле зі списком»* отримає значення з іншої таблиці або запиту, далі оберіть таблицю *Постачальники*, далі вибираємо поле *Постачальник*, за кнопкою «>», Далі, вибираємо зберегти в полі, і в правому полі зі списком із списку вибираємо *Код постачальника*, далі введіть назву підпису *Постачальник*, натисніть кнопку *Готово*.

Також нижче постачальника ставимо поле зі списком для введення матеріалу, при цьому відмінність у тому, що вибираємо таблицю *Номенклатура матеріалів* та поле *Матеріал*, а зберігаємо в полі *Код матеріалу*, підпис відповідно *Матеріал*.

Закриваємо конструктор і питання збереження змін відповідаємо ***Так***.



Заповніть форму довільними даними.

Рис. 3.2. Вид форми Дані про поставку матеріалів

Завдання 5. Створення запитів. Для подальшого формування звітів необхідно створити два запити на вибірку це *Надходження матеріалів від постачальників за видами* та *Надходження матеріалів за датами за певний*

період. Спочатку сформуйте запит *Надходження матеріалів від постачальників за видами.* За допомогою *Майстра запитів* створіть простий запит із полями з наступних таблиць:

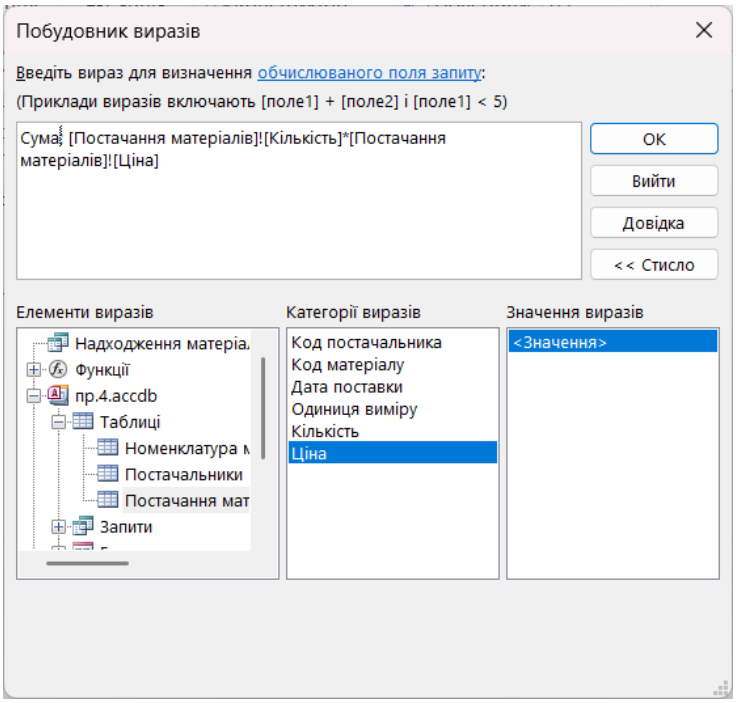
- *Постачальники – поле Постачальник;*

– Номенклатура матеріалів – поле Матеріал;

– Постачання матеріалів – поля Дата поставки, Одиниця виміру, Кількість, Ціни.

Далі оберіть *докладний звіт*, потім надайте ім'я запиту *Надходження матеріалів від постачальників за видами* і нижче оберіть режим *Змінити макет запиту*. У сьомому полі запиту у верхньому рядку помістіть курсор і клацнувши правою кнопкою миші з меню, що з'явилося, виберете ***Побудувати...*** У нижньому лівому вікні (*Елементи виразів*) натисніть на плюс, де таблиці відкрийте їх і оберіть таблицю *Постачання матеріалів*. З полів, що з'явилися, в середньому нижньому вікні подвійним клацанням виберете поле Кількість, після натисніть кнопку на клавіатурі зі знаком *«*»* і далі виберете поле *Ціна*. Натисніть кнопку **ОК**.

Поставте курсор назад у верхній рядок сьомого поля запиту та замість слова *Вираз1* наберіть *Сума*. Закрийте запит, збережіть зміни, у вікні, введіть ім'я запиту *Надходження матеріалів від постачальників за видами* і



натисніть ОК.

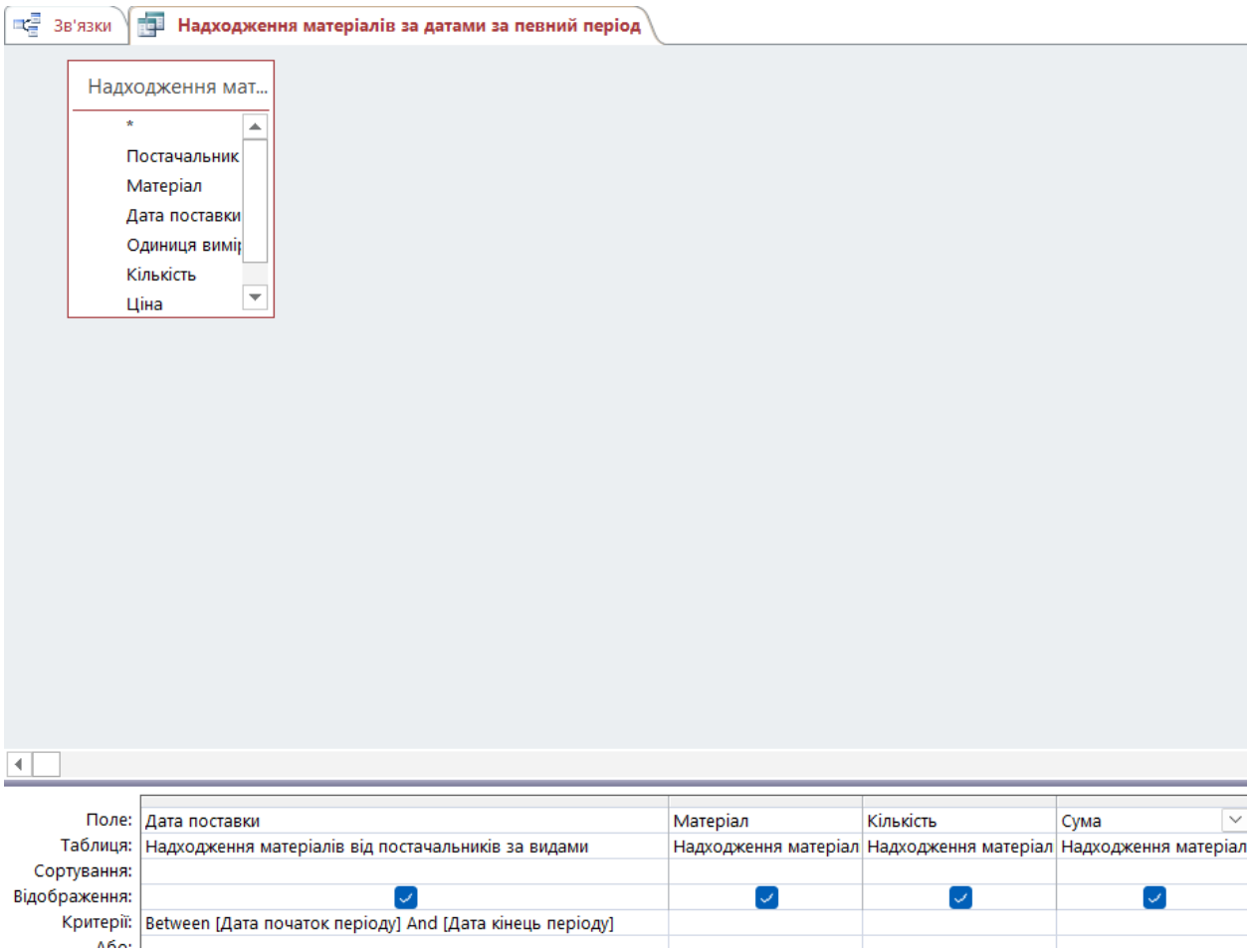
Рис. 3.3. Вікно генератору виразів

Далі створюємо запит *Надходження матеріалів за датами за певний період*.

За допомогою *Майстра запитів* створіть простий запит із запиту *Надходження матеріалів від постачальників за видами* та оберіть наступні поля: *Дата постачання, Постачальник, Матеріал, Кількість та Сума*.

Далі виберете докладний звіт, потім надайте ім'я запиту *Надходження матеріалів за датами за певний період* і нижче оберіть режим *Змінити макет запиту*.

У рядку *умова відбору* першому полі (де стоїть *Дата поставки*) ставимо курсор і пишемо ***Between [Дата початок періоду] And [Дата кінець періоду]***. Закриваємо конструктор, зберігаємо зміни. Тепер перед виконанням запиту у користувача буде спочатку запрошено *Дата початок*



періоду, а потім *Дата кінець періоду* і так буде заданий період між якими датами відібрати записи.

Рис. 3.4. Тип запиту Надходження матеріалів за датами за певний період

Завдання 6. Формування звітів. Необхідно сформувати три звіти - ***Надходження матеріалів від постачальників за видами, Надходження матеріалів за видами від постачальників та Надходження матеріалів за датами за певний період.***

Для формування звіту *Надходження матеріалів від постачальників за видами*. Відкрийте *Майстер звітів* виберіть зі списку *Запит: Надходження матеріалів від постачальників за видами*. Далі виберіть всі поля із запиту. Натискаємо кнопку *Далі*. Вигляд представлення даних оберіть *Постачальники*, це перший рівень групування. Далі виберіть *Матеріал*, це другий рівень групування. Далі натисніть кнопку *Підсумки…* та поставте галочку на перетині рядка *Сума* та стовпця *Sum* та натисніть *ОК*. Далі оберіть *Блок* та орієнтацію паперу *альбомний*. Далі введіть ім'я звіту *Надходження матеріалів від постачальників за видами* та виберіть режим Зміна макету звіту.

В області *примітки* видаліть верхній вираз. Нижче замість *Sum* напишіть *Разом за матеріалами*.

Поле *Од.* зменшуємо у довжині, а збільшуємо поле *Дата поставки*, те

Надходження матеріалів від постачальника за видами

Постачальник	Матеріал	Дата поставки	од..	Кількість	Ціна	Сума
Dom Nouta	Жорсткі диски	05.09.2025 шт.		5	500,00 ₴	4500
	Sum					2500
Sum						2500
USN Computers	Материнські плати	03.09.2025 шт.		2	4 300,00 ₴	4500
	Sum					8600
Sum						8600
Діджітек	Модуль пам'яті	01.09.2025 шт.		4	3 000,00 ₴	4500
	Sum					12000
Sum						12000
Іліон Технолоджі	Системи охолодження	02.09.2025 шт.		3	2 000,00 ₴	4500
	Sum					6000
Sum						6000
ІнформПортал	Відеокарти	04.09.2025 шт.		1	1 000,00 ₴	4500
	Sum					1000
Sum						1000
Всього						30100

саме зробивши і з відповідними полями в області даних. Для полей *Дата поставки* та інших у властивостях вибираємо вирівнювання по центру. Ну і так далі робимо зміни, щоб поля добре читалися, не наповзали один на одного і поміщалися всі дані, що виводяться. Закриваємо звіт та зберігаємо зміни.

Рис. 3.5. Вид звіту ***Надходження матеріалів від постачальників за видами***

Також створюємо звіт Надходження матеріалів за видами від постачальників тільки вибираємо не всі поля із запиту, а лише поля *Матеріал, Постачальник, Кількість, Ціна та Сума* та групування першого рівня за матеріалами та другого за *постачальниками*.

Створіть звіт *Надходження матеріалів за датами за певний період*. Для цього запустіть *Майстер форм*, виберіть *Запит: Надходження матеріалів за датами за певний період*. Виберіть усі поля, далі оберіть вид представлення *Постачання матеріалів* натисніть кнопку Далі. Виберіть рівень групування за *датою постачання*, для цього натисніть на це поле двічі лівою кнопкою миші. Натисніть кнопку *Групування* та виберіть інтервал угруповання *щодня*, натисніть ОК. Далі оберіть *Сортування* по полю *Дата поставки*. Натисніть кнопку *Підсумки* та поставте галочку на перетині *рядка Сума та стовпця Sum*. Далі виберіть макет *Блок*, орієнтація *альбомна*. Введіть ім'я звіту *Надходження матеріалів за датами за певний період* та перейдіть до режиму *Змінити макет звіту*. У режимі конструктора налаштуйте звіт.

Практична робота № 5

Створення БД «Прокат автомобілів»

Створити базу даних «Прокат автомобілів», що складається з трьох таблиць із взаємозалежними даними, запитами, формами, які використовуються для наочної роботи з даними (введення, редагування тощо), і навіть звітів.

Хід роботи

Завдання 1. Самостійно створити нову базу даних під назвою «Прокат автомобілів». Після створення порожньої бази даних необхідно створити об'єкти цієї бази даних.

Завдання 2. Створення таблиць Структура таблиці *"Автомобілі*": **Код автомобіля**, модель, колір, номерний знак, страхова вартість автомобіля, вартість одного дня прокату.

Структура таблиці "*Клієнти*": **Код клієнта**, прізвище, ім'я, по батькові, паспорт.

Структура таблиці *Прокат*: **Код прокату**, код клієнта, код автомобіля, дата початку прокату, кількість днів прокату.

Визначити самостійно типи полів у таблицях. Ключові поля таблиць виділені жирним шрифтом. Ключові поля визначити типом *Лічильник*. Узгодити вибрані типи полів і типи зв'язків, що передбачаються, між таблицями з викладачем (на власний розсуд).

Створити в режимі Конструктор таблиці заданої структури. При створенні структури таблиць врахувати, що ім'я поля може не співпадати з

підписом поля (імена доцільно вибирати коротшими). Це полегшить надалі роботу з базою даних.

Завдання 3. Створення зв'язків. Встановити відносини між таблицями.

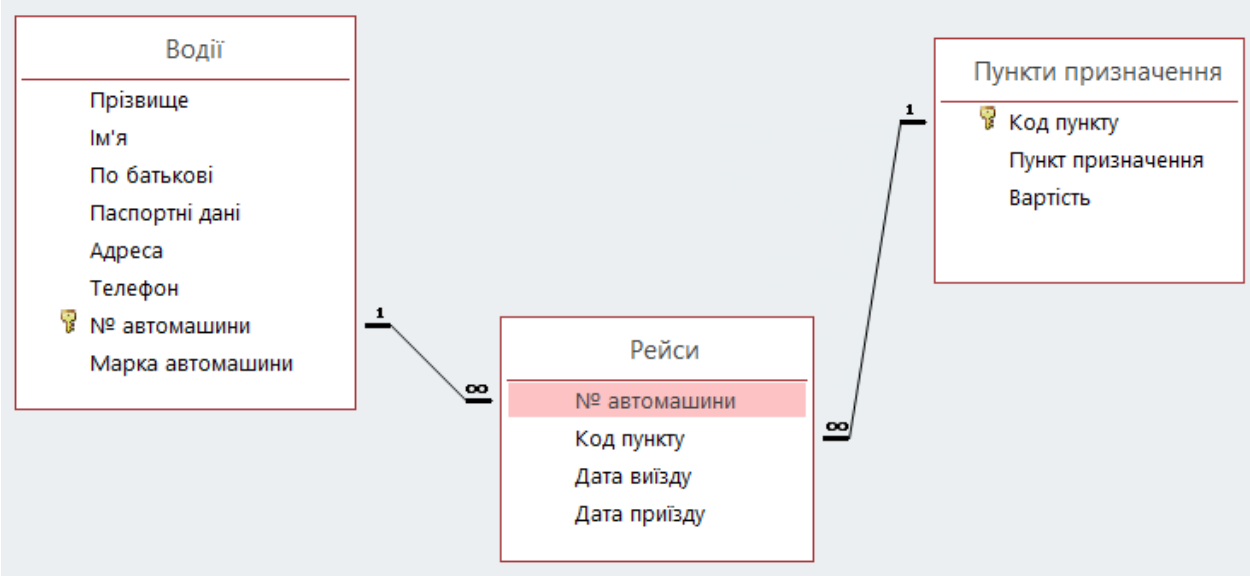


Рис. 5.1 - Схема даних бази даних «Прокат автомобілів»

Заповнити таблиці даними (щонайменше 10 записів у кожній таблиці).

Завдання 4. Створення форм:

- 1 проста форма з урахуванням однієї таблиці;

- 1 форма із підлеглою формою.

Завдання 5. Створення запитів:

- 4 запити на вибірку автомобіля за кольором;

- 3 запити з параметрами (довільно);

- запит з полями, що обчислюються (Вартість прокату автомобіля визначається: Вартість одного дня прокату * Кількість днів прокату. Фірма щорічно страхує автомобілі, що видаються клієнтам. Страхові внески дорівнюють 10 відсоткам від страхової вартості автомобіля.);

Завдання 6. Створення звітів:

- автозвіт з урахуванням будь-якої базової таблиці;

- автозвіт на основі будь-якого запиту.

Практична робота № 6

SQLite. Загальні відомості. Типи даних. Створення БД. Створення таблиці. Резервне копіювання та відновлення. Імпорт-експорт даних.

1. Загальні теоретичні відомості.

SQLite — це програмна бібліотека, що реалізує самодостатню, не потребуючу сервера, транзакційну систему управління реляційними базами даних SQL. SQLite - це одна з найбільш швидко зростаючих СУБД. Вихідний код SQLite є у вільному доступі. Це база даних, яка не потребує конфігурування, ще означає, що вам не потрібно її налаштовувати під вашу операційну систему. SQLite-движок не є окремим процесом, як і інші бази даних, ви можете поєднати його статично або динамічно відповідно до ваших вимог з вашим додатком. SQLite отримує прямий доступ безпосередньо до файлів даних.

Чому SQLite?

• SQLite не вимагає окремого серверного процесу або систему для роботи (serverless).

• SQLite поставляється з нульової конфігурації, який означає, що не потрібні установки або адміністрування.

• Вся база даних SQLite зберігається у одному крос платформному файлі.

• SQLite - дуже мала неважка, менше, ніж 400KiB у повній комплектації або менше, ніж 250KiB з виключенням деяких функції.

• SQLite є самодостатньою, що означає, що немає зовнішніх залежностей.

• SQLite транзакції повністю задовольняють вимогам ACID, дозволяючи безпечний доступ з декількох процесів або потоків.

• SQLite підтримує більшість функцій мови запитів, знайдені в SQL92 (SQL2) стандарті.

• SQLite написано в ANSI-C та забезпечує простий і легкий у використанні API.

• SQLite доступна на UNIX (Linux, Mac OS X, Android, iOS) і Windows (Win32, WinCE, WinRT).

2. Інсталяція та запуск SQLite.

Крок 1: перейдіть на сторінку завантаження SQLite http://www.sqlite.org/download.html в розділ Windows.

Крок 2: Завантажити sqlite-shell-win32*.zip і sqlite-dll-win32-*.zip архіви.

Крок 3: Створити папку C:\>sqlite і розпакувати завантажені архіви у цю папку (мають бути sqlite3.def, sqlite3.dll та sqlite3.exe файли).

Крок 4: Додавати C:\>sqlite у змінній оточення PATH

Крок 5: Перейти до командного рядка і виконати команду sqlite3.

При виконанні роботи в комп'ютерному класі:

- Завантажити з інтернету (або скопіювати в локальній мережі – уточнити у викладача де саме) файли sqlite3.def, sqlite3.dll та sqlite3.exe у папку c:\temp\sqlite (або у будь-яку іншу, де є права на запис даних).
- Запустити командний рядок Windows->run (win+r) ->cmd

```
cd c:\temp\sqlite
```

- Перейти до c:\sqlite (виконати команду c:\sqlite)
- Запустити sqlite (виконати команду sqlite3)

```
c:\temp\sqlite>sqlite3
SQLite version 3.21.0 2017-10-24 18:55:49
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite>
```

Далі введемо команду .help для отримання відомостей щодо переліку управляючих команд та .exit для завершення першого сеансу роботи із sqlite.

3. Типи даних

Тип даних SQLite є атрибутом, який визначає тип даних будь-якого об'єкта. Кожен стовпець, змінна і вираз має пов'язані тип даних у SQLite. Ви

повинні використовувати ці типи даних під час створення таблиці. Але при цьому слід розуміти, що SQLite використовує більш загальну динамічну систему типів. У SQLite тип даних значення асоціюється з самим значенням, не з її контейнером.

Класи для збереження даних в SQLite

Кожне значення, що зберігається в БД SQLite з має один із наступних класів.

- NULL - це значення NULL.
- INTEGER - значення є цілим, зберігаються в 1, 2, 3, 4, 6 або 8 байтів в залежності від величини значення.
- REAL – значення числа з плаваючою точкою, збережені як IEEE 8-байтове число з рухомою точкою.
- TEXT - це текстовий рядок, збережений використовуючи кодову сторінку БД (UTF8, UTF-16BE або UTF-16LE)
- BLOB значення blob, які зберігаються точно, як були уведені.

SQLite клас зберігання даних є трохи більш загальним, ніж тип даних. Клас зберігання ціле число, наприклад, включає в себе 6 різних цілих типів даних різної довжини.

SQLite підтримує концепцію спорідненості типів для колонок. Будь яка колонка може зберігати будь який тип даних, але клас, якому надається перевага має назву спорідненого. Кожна колонка таблиці в SQLite3 БД віднесена до одного із наступних споріднених типів: TEXT, NUMERIC, INTEGER, REAL, NONE.

SQLite не має окремого булевого класу зберігання. Замість цього, логічні значення зберігаються як цілі числа 0 (невірно) і 1 (вірно).

4. Створення бази даних.

Для створення БД за допомогою SQLite не потрібно превілегій або додаткових команд. Все, що потрібно зробити – при запуску sqlite3 вказати ім’я БД. Якщо БД існує, робота буде продовжена в контексті вказаної БД, в

іншому випадку вона буде автоматично створена. В командному рядку (див.п.2) виконаємо команду sqlite3 myFirstDB.db

```
c:\temp\sqlite>sqlite3 myFirstDB.db
```

Ця команда створить файл myFirstDB.db в поточному каталозі. Цей файл буде використовуватися як поточна база даних SQLite. Якщо ви помітили, після успішного створення бази даних, sqlite3 виводить

```
c:\temp\sqlite>sqlite3 myFirstDB.db
SQLite version 3.21.0 2017-10-24 18:55:49
Enter ".help" for usage hints.
sqlite> .databases
main: c:\temp\sqlite\myFirstDB.db
sqlite>
```

запрошення sqlite > для вводу наступної команди.

Після створення бази даних, ви можете перевірити у списку баз даних, використовуючи команду .databases.

5. Створимо таблицю students, та додамо до неї декілька рядків

```
sqlite> create table students(id int, name text, age int);
sqlite> insert into students values(1, 'Ivanov', 20);
sqlite> insert into students values(2, 'Petrov', 19);
sqlite> insert into students values(3, 'Sidorov', 21);
```

```
sqlite> .mode column
sqlite> .headers on
sqlite> select * from students;
id          name        age
----------  ----------  ----------
1           Ivanov      20
2           Petrov      19
3           Sidorov     21
```

Перевіримо вміст таблиці, попередньо виконавши налаштування для відображення імен колонок виводу

6. Для відображення схеми поточної БД можемо виконати команду .schema

Також уявлення про перелік об’єктів БД можна отримати звернувшись до системної таблиці sqlite_master.

```
sqlite> select * from sqlite_master;
type        name        tbl_name    rootpage    sql
----------  ----------  ----------  ----------  ---------------------------------------------
table       students    students    2           CREATE TABLE students(id int, name text, age int)
```

7. За допомогою команди. dump можна виконати резервну копію БД у вигляді скрипта на створення об’єктів БД та заповнення їх даними.

```
sqlite> .output myDump.sql
sqlite> .dump
sqlite> .output stdout
```

Наведемо вміст створеного файлу myDump.sql

```
PRAGMA foreign_keys=OFF;
BEGIN TRANSACTION;
CREATE TABLE students(id int, name text, age int);
INSERT INTO students VALUES(1,'Ivanov',20);
INSERT INTO students VALUES(2,'Petrov',19);
INSERT INTO students VALUES(3,'Sidorov',21);
COMMIT;
```

Дамп БД також може бути створений з командного рядку за допомогою sqlite3

```
c:\temp\sqlite>sqlite3 myFirstDB.db .dump > myDump.sql
```

```
c:\temp\sqlite>sqlite3 myFirstDB.db < myDump.sql
```

8. Маючи резервну копію можемо у разі необхідності відновити БД. Видалимо файл myFirstDB.db та виконаємо команду

Далі завантажимо БД myFirstDB та пересвідчимося у наявності таблиці students та даних у ній

```
c:\temp\sqlite>sqlite3 myFirstDB.db
SQLite version 3.21.0 2017-10-24 18:55:49
Enter ".help" for usage hints.
sqlite> .tables
students
sqlite> select * from students;
id          name        age
----------  ----------  ----------
1           Ivanov      20
2           Petrov      19
3           Sidorov     21
```

9. Виконаємо експорт даних таблиці students до зовнішнього файлу stud.csv

Видалимо дані з таблиці students та пересвідчимося, що таблиця порожня

```
sqlite> .import stud.csv students
sqlite> select * from students;
1|Ivanov|20
2|Petrov|19
3|Sidorov|21
sqlite>
```

Завантажимо дані з файлу до таблиці students та перевіримо її вміст

Практична робота № 7

SQLite. Зміна таблиці. Обмеження цілісності, первинний, зовнішній ключі та перевірка. Індекси

1. Запустимо sqlite3 із вказанням поточної БД studentsList.db та створимо таблицю students із полями id та name. (у разі необхідності

```
c:\temp\sqlite>sqlite3 studentsList.db
SQLite version 3.21.0 2017-10-24 18:55:49
Enter ".help" for usage hints.
sqlite>
```

```
sqlite> create table students(id int, name text);
sqlite> .schema students
CREATE TABLE students(id int, name text);
sqlite>
```

виконати попередньо п.2 пр.3)

2. Змінимо ім’я таблиці. Виконаємо команду.

Додамо до таблиці поле age.

!УВАГА. За допомогою команди ALTER TABLE може бути змінене ім’я таблиці а також додані додаткові стовпці. Ніякі інші операції зміни

```
sqlite> .schema stud
CREATE TABLE IF NOT EXISTS "stud"(id int, name text);
sqlite> alter table stud add column age int;
sqlite> .schema stud
CREATE TABLE IF NOT EXISTS "stud"(id int, name text, age int);
sqlite> alter table students rename to stud;
sqlite> .tables
stud
```

командою ALTER TABLE в SQLite не підтримується.

3. Видалимо створену таблицю drop table name, та створимо її заново із

```
sqlite> create table students(id INTEGER PRIMARY KEY, name TEXT, age INTEGER);
sqlite> .schema students
CREATE TABLE students(id INTEGER PRIMARY KEY, name TEXT, age INTEGER);
sqlite>
```

вказанням обмеження цілісності первинний ключ по полю id.

Спробуємо додати до таблиці 2 рядки з однаковим значенням ключа

4. Створимо таблицю заново із заданням автоінкрементного значення

```
sqlite> insert into students(id, name, age) values(1, 'Vasya', 20);
sqlite> insert into students(id, name, age) values(1, 'Petya', 21);
Error: UNIQUE constraint failed: students.id
sqlite>
sqlite> drop table students;
sqlite> create table students(id INTEGER PRIMARY KEY AUTOINCREMENT, name TEXT, age INTEGER);
```

первинного ключа

Та додамо до неї 2-х студентів

```
sqlite> insert into students(name, age) values('Petya', 21);
sqlite> insert into students(name, age) values('Vasya', 20);
sqlite> .headers on
sqlite> .mode column
sqlite> select * from students;
id          name        age
----------  ----------  ----------
1           Petya       21
2           Vasya       20
sqlite>
```

Переглянемо результат

5. Створимо таблицю групи (id, номер групи, староста групи)

Та додамо до неї декілька рядків даних

Перевіримо результат

```
sqlite> select * from groups;
id          number      starosta
----------  ----------  ----------
1           201         Ivanov
2           202         Petrova
sqlite>
```

```
ary key autoincrement, number text, starosta text);
starosta) values(201, 'Ivanov');
starosta) values(202, 'Petrova');
```

```
sqlite> .schema groups
CREATE TABLE groups(id integer primary key autoincrement, number text, starosta text);
```

6. Бувають ситуації, коли первинний ключ складається з декількох полів. Наприклад сутність «паспорт» може мати складений первинних ключ із окремих полів «серія» та «номер». В даному випадку синтаксис створення таблиці матиме наступний вигляд.

```
CREATE TABLE passport(seria TEXT, number TEXT, who_gave TEXT, PRIMARY KEY(seria, numb
er));
```

Перевіримо роботу, додамо 3 рядки, 2 без порушення унікальності, 1 із

```
sqlite> insert into passport values('EO', '123432', 'Mykolaiv');
sqlite> insert into passport values('EK', '123432', 'Odesa');
sqlite> insert into passport values('EK', '123432', 'Kiev');
Error: UNIQUE constraint failed: passport.seria, passport.number
sqlite>
```

порушенням

7. Далі додамо до таблиці students поле group_id із обмеженням цілісності зовнішний ключ

Виконаємо зміну поля group_id для students

Як можна переконатись із результатів запитів до таблиць students та

```
sqlite> alter table students add column group_id references groups(id);
sqlite> update students set group_id = 3;
sqlite> select * from students;
id          name        age         group_id
----------  ----------  ----------  ----------
1           Petya       21          3
2           Vasya       20          3
```

```
sqlite> select * from groups;
id          number      starosta
----------  ----------  ----------
1           201         Ivanov
2           202         Petrova
```

```
sqlite> pragma foreign_keys;
foreign_keys
------------
0
```

groups маємо порушення посилальної цілісності, оскільки групи із кодом 3 не існує. Це стало можливим через теж, що за замосвенням sqlite налаштований таким чином, що не виконує перевірку на порушення області значень

зовнішнього ключа.

Змінимо це

При повторному виконанні запиту на зміну номера групи матимемо

```
sqlite> pragma foreign_keys = on;
sqlite> pragma foreign_keys;
foreign_keys
------------
1
```

помилку

```
sqlite> update students set group_id = 3;
Error: FOREIGN KEY constraint failed
```

```
sqlite> update students set group_id = 1 where id = 1;
sqlite> update students set group_id = 2 where id = 2;
```

Змінимо номера груп для усунення порушення посилальної цілісності

Та переглянемо результат

```
sqlite> select * from students;
id          name        age         group_id
----------  ----------  ----------  ----------
1           Petya       21          1
2           Vasya       20          2
```

8. Обмеження цілісності зовнішній ключ бажано має бути визначене на етапі створення таблиці. Розберемо на прикладі, як це робиться. Для цього

```
sqlite> drop table students;
```

```
sqlite> CREATE TABLE students(
   ...> id INTEGER PRIMARY KEY AUTOINCREMENT,
   ...> name TEXT,
   ...> group_id INTEGER,
   ...> FOREIGN KEY(group_id) REFERENCES groups(id)
   ...> );
```

```
sqlite> .schema students
CREATE TABLE students(
id INTEGER PRIMARY KEY AUTOINCREMENT,
name TEXT,
group_id INTEGER,
FOREIGN KEY(group_id) REFERENCES groups(id)
);
```

видалимо таблицю students і створимо її заново відразу із зовнішнім ключем.

9. Виконуючи повторне створення таблиці students не було додане поле age. Додамо його із перевіркою на >0 і <150.

```
sqlite> .schema students
CREATE TABLE students(
id INTEGER PRIMARY KEY AUTOINCREMENT,
name TEXT,
group_id INTEGER, age INTEGER CHECK(age>0 and age<150),
FOREIGN KEY(group_id) REFERENCES groups(id)
);
```

```
ge>0 and age<150);
```

Перевіримо роботу обмеження цілісності

```
sqlite> select * from students;
id          name        group_id    age
----------  ----------  ----------  ----------
1           Vasya       1           20
sqlite>
```

До таблиці Students створимо 2 індекси для полів age та name – без та з

```
sqlite> insert into students(name, age, group_id)
sqlite> create index stud_age on students(age);
sqlite> create unique index stud_name on students(name);
   ...> values('Petya', 220, 1);
Error: CHECK constraint failed: students
sqlite> insert into students(name, age, group_id)
   ...> values('Klya', -2, 2);
Error: CHECK constraint failed: students
sqlite>
```

контролем унікальності відповідно.

Переглянемо перелік індексів таблиці

```
sqlite> .indices students
stud_age    stud_name
```

```
sqlite> select * from sqlite_master where type='index';
type        name        tbl_name    rootpage    sql
----------  ----------  ----------  ----------  ------------------------------------
index       stud_age    students    5           CREATE INDEX stud_age on students(age)
index       stud_name   students    6           CREATE UNIQUE INDEX stud_name on stude
sqlite>
```

Переглянути індекси також можна скориставшись системною таблицею sqlite_master

Перевіримо роботу унікального індексу

10. Дозволимо додавання студентів із однаковим іменем, але у різні групи. Для цього видалимо індекс stud_name

```
sqlite> insert into students(name, age, group_id)
   ...> values('Petya', 19, 2);
sqlite> insert into students(name, age, group_id)
   ...> values('Petya', 20, 1);
Error: UNIQUE constraint failed: students.name
sqlite> drop index stud_name;
```

```
sqlite> select * from sqlite_master where type='index';
type        name        tbl_name    rootpage    sql
----------  ----------  ----------  ----------  ------------------------------------
index       stud_age    students    5           CREATE INDEX stud_age on students(age)
sqlite>
```

Та створимо його заново, але цього разу він складатиметься з 2-х полів – name та group_id

Перевіримо його роботу

```
sqlite> select * from students;
id          name        group_id    age
----------  ----------  ----------  ----------
1           Vasya       1           20
2           Petya       2           19
```

```
ts(group_id, name);
```

```
sqlite> insert into students(name, age, group_id)
   ...> values('Petya', 20, 2);
Error: UNIQUE constraint failed: students.group_id, students.name
index       stud_name   students    6           CREATE UNIQUE INDEX stud_name on stude
```

```
sqlite> insert into students(name, age, group_id)
   ...> values('Petya', 20, 1);
sqlite> select * from students;
id          name        group_id    age
----------  ----------  ----------  ----------
1           Vasya       1           20
2           Petya       2           19
3           Petya       1           20
sqlite>
```

Отже, Petya не додається до групи із кодом 2, в якій від уже є, але може бути доданий до іншої групи.

Завдання на самостійну роботу

В окремій БД створити 2 зв’язані таблиці із вказанням обмеженнями цілісності первинний ключ (в кожній таблиці), зовнішній ключ(в одній із таблиць), хоча б однієї перевірки та 2-х індексів (один із яких унікальний).

Практична робота № 8

SQLite. Виконання запитів на вибірку даних

```
c:\temp\sqlite>sqlite3 food.db < foods.sql
```

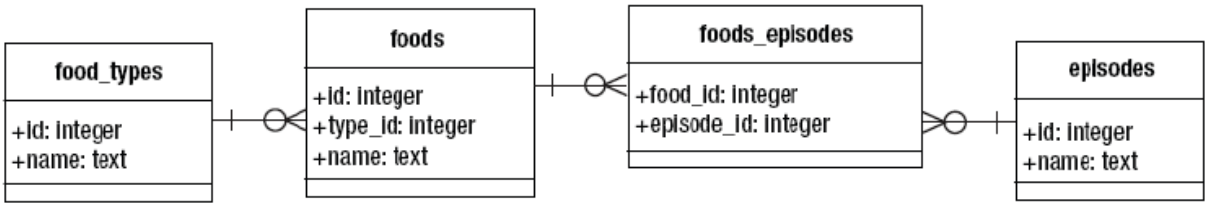
1. Завантажимо тестову БД Seinfeld із дампу foods.sql.

Далі запустимо sqlite3 із вказанням поточної БД food.db та переглянемо перелік таблиць БД

```
c:\temp\sqlite>sqlite3 food.db
SQLite version 3.21.0 2017-10-24 18:55:49
Enter ".help" for usage hints.
sqlite> .tables
episodes        food_types      foods           foods_episodes
```

Як бачимо, БД складається з 4-х таблиць. Наведемо даталогічну модель БД та її загальний огляд.

База даних, що використовується містить імена страв з кожного епізоду серіалц Seinfeld. (If you've ever watched Seinfeld, you can't help but notice a slight preoccupation with food). Глядач цього серіалу не може не відчути занепокоєність їжею ☺. Більш ніж ста вісімдесяти епізодів згадується більш



ніж чотириста різних страв. На рисунку показано схему бази даних.

Рис. 6.1 – Схема бази даних

Foods є головною таблицею. Кожен запис відповідає окремому блюду, чиє ім'я реєструється в полі name. Поле Type_id посилається на таблицю food_types, яка містить класифікацію продуктів (тобто, фаст-фуд, напої і так далі). Нарешті, таблиця food_episodes поєднує продукти з великою кількістю епізодів.

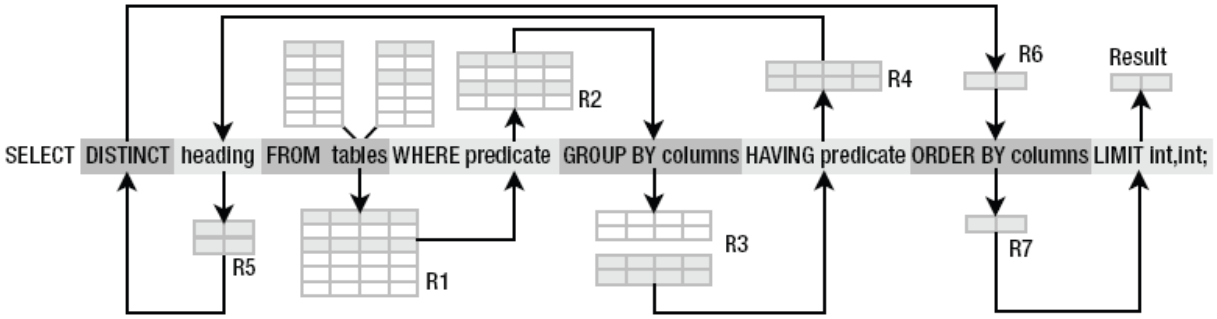
2. Перед розглядом конкретних прикладів щодо вибірки даних із БД, наведемо загальний огляд команди select.

Оператор select включає реляційні операції за допомогою серії фраз (речення - clause). Кожна фраза відповідає своїй реляційної операції. У

```
select [distinct] heading
from tables
where predicate
group by columns
having predicate
order by columns
limit count,offset;
```

SQLite майже всі фрази не обов'язкові. Користувач SQLite може використовувати тільки ті, які операції він потребує. Найбільша загальна форма select в SQLite може бути представлена як:

Одна із інтерпретацій оператора select - це уявити його як конвеєр, який обробляє відношення. На конвеєрі є необов'язкові процеси, виконання



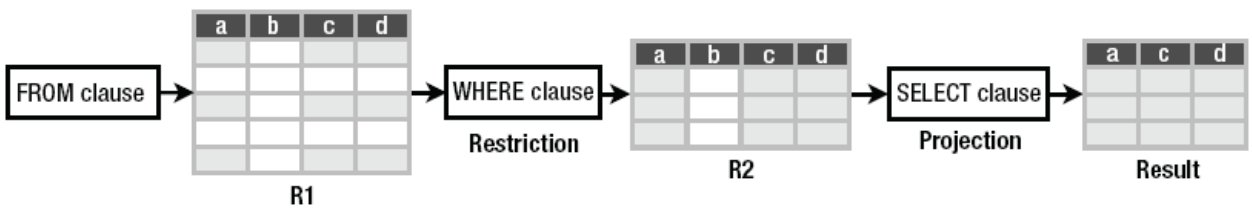
яких можна пропускати. Незалежно від того, використовуються чи не використовуються конкретні операції (процеси), конвеєр завжди працює однаково. На малюнку нижче можна подивитися порядок виконання. Виконання оператора select починається з фрази from, яка приймає одне або більше відношень і з'єднує їх в одне складне відношення і, потім, передає ланцюжку послідовних операцій.

Рис. 6.2 – Послідовність операцій

Фраза from містить список декількох таблиць, представлень і підзапитів (представлених у вигляді змінної tables), розділених комами. Більш ніж одна таблиця (представлення або підзапит) буде з'єднуватися в одне відношення, яке на тому ж рисунку представляються іменем R1. Різні елементи поєднуються в одне відношення операцією join.

Фраза where відбирає необхідні записи з R1. За ключовим словом where слідує предикат (логічне вираз), який визначає критерій відбору записів з R1, які повинні бути включені в наступне відношення. Обрані записи утворюють нове відношення R2.

Як показує наступний рисунок, фраза select в SQLite з'єднує всі дані, розглянуті у фразі from, відбирає записи (обмежує) їх у фразі where і відбирає



поля (проектує) у фразі select.

Рис. 6.3 – Проектування запиту

3. Далі перейдемо до розгляду простих прикладів.

a. Вибрати список типів страв

```
sqlite> select * from food_types;
id          name
----------  ----------
1           Bakery
2           Cereal
3           Chicken/Fo
4           Condiments
5           Dairy
6           Dip
7           Drinks
8           Fruit
9           Junkfood
10          Meat
11          Rice/Pasta
12          Sandwiches
13          Seafood
14          Soup
15          Vegetables
sqlite>
```

```
sqlite> select name from food_types;
name
----------
Bakery
Cereal
Chicken/Fo
Condiments
Dairy
Dip
Drinks
Fruit
Junkfood
Meat
```

b. Вибрати назви типів страв

c. Вибрати назви серій серіалу

d. Кількість рядків у результаті попереднього запиту досить велика, виберемо лише 10 серій із загального списку

```
sqlite> select name from episodes;
sqlite> select name from episodes LIMIT 5;
name
------------------
Good News Bad News
Male Unbonding
The Stake Out
The Robbery
The Stock Tip
sqlite>
```

```
sqlite> select * from foods limit 5;
id          type_id     name
----------  ----------  ----------
1           1           Bagels
2           1           Bagels, ra
3           1           Bavarian C
4           1           Bear Claws
5           1           Black and
sqlite>
```

e. Виберемо 5 страв

Для зручності виводу можна змінити ширину стовбців

4. Фільтрування даних

```
sqlite> select * from episodes where season = 2;
id          season      name
----------  ----------  ----------------
5           2           The Ex-Girlfriend
6           2           The Pony Remark
7           2           The Busboy
8           2           The Baby Shower
9           2           The Jacket
10          2           The Chinese Resta
11          2           The Phone Message
12          2           The Apartment
3    1    Bavarian Cream Pie
4    1    Bear Claws
5    1    Black and White cookies
```

a. Список назв серій другого сезону серіалу

b. Список страв, що відноситься до 2-го та 3-го типу

Або

c. Страви, назва яких пов’язана із морквою

```
sqlite> select * from foods where name like '%carrot%';
id          type_id     name
----------  ----------  ----------
8           1           Carrot Cake
363         14          Carrot Soup
387         15          Carrots
sqlite>
49          2           Cheerios
sqlite> select * from episodes where season = 3 and name like '%boyfriend%';
id   seaso  name
---  -----  ----------
32   3      The Boyfriend 1
33   3      The Boyfriend 2
```

d. Серії 3-го сезону про бойфренда

5. Групування та функції агрегації

```
sqlite> .width auto
sqlite> select count(*) from episodes;
count(*)
----------
181
```

a. Загальна кількість серій у серіалі

b. Кількість серій 2-го сезону

c. Кількість сезонів

d. Кількість серій по сезонах

```
sqlite> select count(*) from episodes where season = 2;
count(*)
sqlite> select count(distinct season) from episodes;
count(distinct season)
sqlite> select season, count(id)
   ...> from episodes
   ...> group by season;
season      count
----------  -----
            2
1           4
2           13
3           22
4           24
5           22
6           24
7           24
8           22
9           24
sqlite>
```

e. Сезони, в яких більше 20 серій

```
sqlite> select season
   ...> from episodes
   ...> group by season
   ...> having count(id) > 20;
season
----------
3
4
5
6
7
8
9
sqlite>
```

Впорядкування виводу

```
sqlite> select name from food_types order by name;
name
----------
Bakery
Cereal
Chicken/Fo
Condiments
Dairy
Dip
Drinks
Fruit
```

f. Список типів блюд, впорядкований за назвою

g. Список страв 1-го типу, відсортований за назвою в зворотному порядку

h. Список серій, відсортований за сезоном за зростанням, та за назвою серії за спаданням

i. Список сезонів серіалу, впорядкований за кількістю серій за

```
sqlite> select season, count(id)
   ...> from episodes
   ...> group by season
   ...> order by 2 desc;
season      count(id)
----------  ----------
4           24
6           24
7           24
9           24
3           22
5           22
8           22
2           13
1           4
            2
sqlite>
```

```
id = 1 order by name desc;

-----
 season, name desc;

--------------------
l:The Seinfeld Story
lews
```

спаданням

6. Запити до декількох таблиць

```
sqlite> select f.name, t.name
   ...> from foods f inner join food_types t
   ...> on f.type_id = t.id
   ...> limit 10;
name        name
----------  ----------
Bagels      Bakery
Bagels, ra  Bakery
Bavarian C  Bakery
Bear Claws  Bakery
Black and   Bakery
Bread (wit  Bakery
Butterfing  Bakery
Carrot Cak  Bakery
Chips Ahoy  Bakery
Chocolate   Bakery
sqlite>
```

a. Список страв із типом страви

b. Салати із категорії овочі

c. Перші 5 епізодів за кількістю згадувань про їжу

```
sqlite> select e.name, count(f.food_id)
   ...> from episodes e inner join foods_episodes f
   ...> on e.id = f.episode_id
   ...> group by e.name
   ...> order by 2 desc
   ...> limit 5;
name                  count(f.food_id)
--------------------  ---------------
The Soup              23
The Fatigues          14
The Bubble Boy        12
The Finale 1          10
The Dinner Party      9
sqlite>
   ...> from foods f inner join food_types t
   ...> on f.type_id = t.id
sqlite> select e.season, count(distinct f.food_id)  getables'
   ...> from episodes e inner join foods_episodes f
   ...> on e.id = f.episode_id
   ...> group by e.season
   ...> order by 2 desc
   ...> limit 3;
season                count(distinct f.food_id)
--------------------  ------------------------
9                     82
5                     73
6                     69
sqlite>
```

d. Перші 3 сезони за кількістю згадувань різних страв

e. У скількох серіях кожного із сезонів згадується про морозиво

f. У яких серіях 5-го сезону згадується випічка.

Завдання для самостійної роботи

a. Вибрати типи страв, що починаються з літери «D»

b. Вибрати страви із кодом категорії від 3 до 5.

```
sqlite> select e.season, count(distinct e.id)
   ...> from episodes e inner join foods_episodes fe
sqlite> select distinct e.*
   ...> from episodes e inner join foods_episodes fe
   ...> on e.id = fe.episode_id
   ...> inner join foods f on fe.food_id = f.id
   ...> inner join food_types t on f.type_id = t.id
   ...> where t.name = 'Bakery' and e.season = 5;
id                  season      name
------------------  ----------  ----------------------
67                  5           The Sniffing Accountant
82                  5           The Raincoats 2
76                  5           The Dinner Party
66                  5           The Glasses
78                  5           The Pie
75                  5           The Stall
sqlite>
```

c. Вибрати серії 9 сезону, в назві яких зустрічається «fin»

d. Типи страв із кількістю страв, що входять до них

e. Перші 3 типи страв за кількістю страв

f. Страва, що згадується у найбільшій кількості епізодів.

g. Скільки різних типів страв згадується в кожному із сезонів.

Практична робота № 9

SQLite. Представлення та тригери.

1. Крім таблиць та індексів в SQLite є ще 2 типи логічних об’єктів – це представлення та тригери. Почнемо із огляду представлень.

Представлення (view) є віртуальними таблицями. Їх ще називають похідними таблицями, в зв'язку з тим, що їх вміст є результатом виконання запитів до інших таблиць. Насправді, хоча представлення схожі на справжні таблиці, вони ними не є. Вміст справжніх таблиць є справжніми дані, в той

час, як вміст представлення динамічно генерується під час звернення до нього.

```
create view name as select-stmt;
```

Синтаксис для створення представлення наступний:

Назва представлення задається текстом name, а визначається оператором select - stmt. В результаті представлення буде виглядати, як таблиця з назвою name. Уявіть, що існує запит, який потрібно часто виконувати. Представлення - це засіб, який дозволяє уникати постійного набрання такого запиту. Інший приклад – якщо запит є частиною деякої кількості складних запитів.

2. Створимо представлення, що буде відображати сезони, назви

```
c:\temp\sqlite>sqlite3 foods.db
SQLite version 3.21.0 2017-10-24 18:55:49
Enter ".help" for usage hints.
sqlite> .tables
episodes        food_types      foods           foods_episodes
```

епізодів, страв, що в них згадуються та типів блюд, до яких вони відносяться.

```
sqlite> CREATE VIEW episode_food as
   ...> select e.season, e.name ename, f.name fname, t.name tname
   ...> from episodes e inner join foods_episodes fe on e.id = fe.episode_id
   ...> inner join foods f on fe.food_id = f.id
   ...> inner join food_types t on f.type_id = t.id;
sqlite>
```

Перевіримо результат, виберемо перші 10 рядків із створеного

```
sqlite> select * from episode_food limit 10;
season      ename       fname       tname
----------  ----------  ----------  ----------
9           The Strike  Bagels      Bakery
9           The Strike  Bagels, ra  Bakery
8           The Muffin  Bagels, ra  Bakery
7           The Soup N  Bavarian C  Bakery
9           The Strong  Bear Claws  Bakery
5           The Sniffi  Bear Claws  Bakery
5           The Rainco  Bear Claws  Bakery
5           The Dinner  Black and   Bakery
6           The Unders  Black and   Bakery
9           The Apolog  Bread (wit  Bakery
sqlite>
```

представлення

Виконаємо ще декілька запитів із використанням створеного представлення. Виберемо які типи страв зустрічаються у другому сезоні серіалу

Або назву серії, в якій згадується найбільше страв

Завдання для самостійної роботи

a. Створити представлення для запитів 8.1 – 8.3

b. Переробити запити 8.4 – 8.7 наступним чином: створити представлення що включає всі необхідні для запиту поля і таблиці (із

```
sqlite> select distinct tname
   ...> from episode_food
   ...> where season = 2;
tname
sqlite> select ename
   ...> from episode_food
   ...> group by ename
   ...> order by count(fname) desc
   ...> limit 1;
ename
----------
The Soup
sqlite>
Seafood
Vegetables
sqlite>
```

зв’язками). Наступний запит із group by спрямувати до створеного представлення (як в останньому запиті п.2 даної практичної роботи)

SQLite Triggers

3. SQLite Triggers - функції зворотного виклику бази даних, які виконуються / викликаються автоматично при виникненні певної події бази даних.

- Тригер SQLite може створений для запуску кожного разу, коли спрацьовує команда DELETE, INSERT або UPDATE до конкретнії таблиці бази даних або коли відбувається UPDATE на одному або декількох вказаних стовпцях таблиці.

• В даний час SQLite підтримує тільки тригери FOR EACH ROW, а не FOR EACH STATEMENT. Отже, чітке визначення FOR EACH ROW є необов'язковим.

• Ключові слова BEFORE або AFTER визначають, коли буде виконуватись тригер, відносно вставки, модифікації або видалення відповідної рядка.

• Тригери автоматично видаляються, коли видаляється таблиця, з якою вони пов'язані

• Таблиця, яку потрібно змінити, повинна існувати в тій самій базі даних, що і таблиця або представлення, до якої прикріплений тригер, звернення до таблиці має виглядати, як tablename, а не database.tablename.

• Спеціальна функція SQL RAISE() може бути використана в тілі тригерів для ініціювання exception-у.

Загальний синтаксис створення тригеру має наступний вигляд:

```
CREATE  TRIGGER trigger_name [BEFORE|AFTER] event_name
ON table_name
BEGIN
 -- Trigger logic goes here....
END;
```

Event_name може приймати значення операцій БД INSERT, DELETE та UPDATE до таблиці table_name. Ви можете за бажанням вказати FOR EACH ROW після імені таблиці. Далі наводиться синтаксис для створення тригеру

```
CREATE  TRIGGER trigger_name [BEFORE|AFTER] UPDATE OF column_name
ON table_name
BEGIN
 -- Trigger logic goes here....
END;
```

операції оновлення на одному або більше заданих стовпцях таблиці.

4. Створимо окрему таблицю для фіксації додавання нових серій до

```
sqlite> create trigger episode_add BEFORE INSERT
   ...> on episodes
   ...> begin
   ...> insert into episodes_ins_log(id, added)
   ...> values(NEW.id, datetime());                    foods_episodes
   ...> end;                                  ed text);
```

серіалу та тригер, що буде її автоматично заповнювати при виконанні команди INSERT у таблицю episodes.

Додамо новий рядок до таблички episodes та перевіримо роботу тригера

5. Заборонимо додавання нової серії, якщо в сезоні вже більше 20 серій. Створимо тригер.

```
sqlite> create trigger episode_to_season_add BEFORE INSERT
   ...> on episodes
   ...> begin
   ...> SELECT CASE
   ...> WHEN (SELECT COUNT(id) FROM episodes WHERE season=NEW.season) > 20
   ...> THEN RAISE(ABORT, 'Quantity of sesies in one season should be less then 20 !')
   ...> END;
   ...> end;
```

```
sqlite> select * from episodes_ins_log;
```

```
sqlite> select season , count(id) from episodes group by season;
season      count(id)
----------  ----------
            2
1           4
2           13
3           22
4           24
5           22
6           24
7           24
8           22
9           24
20          1
sqlite>
```

Перевіримо поточну кількість серій по сезонах

Додамо серію до першого сезону

Та переконаємось, що вона є у табличці

Тепер спробуємо додати серію до 3-го сезону, де вже 22 серії

```
sqlite> select * from episodes where season = 1 and name like '%triggers%';
id          season      name
----------  ----------  ----------------------------------------
182         1           One more series about triggers, not food :)
sqlite> insert into episodes(id, season, name)
   ...> select max(id)+1, 3, 'One more series about triggers, not food :)'
   ...> from episodes;
Error: Quantity of sesies in one season should be less then 20 !
sqlite>
```

```
sqlite> insert into episodes(id, season, name)
   ...> select max(id)+1, 1, 'One more series about triggers, not food :)'
   ...> from episodes;
```

Виконавши запит на вибірку серії про тригери із 3-го сезону, переконуємось у тому, що рядок не додався.

6. Переглянути перелік тригерів у БД та у конкретній табличці можна використовуючи системну табличку sqlite_master

```
sqlite> select * from episodes where season = 3 and name like '%triggers%';
sqlite>
```

```
sqlite> select * from sqlite_master
   ...> where tbl_name = 'episodes' and type='trigger';
type        name         tbl_name    rootpage    sql

----------  -----------  ----------  ----------  ----------------------------------------------
-------------------------------------------
trigger     episode_add  episodes    0           CREATE TRIGGER episode_add BEFORE INSERT
on episodes
begin
insert into episodes_ins_log(id, added)
values(NEW.id, datetime());
end
trigger     episode_to_  episodes    0           CREATE TRIGGER episode_to_season_add BEFORE INSERT
on episodes
begin
SELECT CASE
WHEN (SELECT COUNT(id) FROM episodes WHERE season
sqlite>
```

7. При необхідності тригер може бути видалений за допомогою команди drop.

```
sqlite> drop trigger episode_add;
```

Завдання для самостійного виконання

a. Створити таблицю-журнал тригер, на видалення даних із таблиці foods (реєструвати назву блюда, що видаляється). Додати та потім видалити рядок із таблиці foods для перевірки роботи тригера.

b. Створити тригер, що при видаленні серії перевіряє, щоб це була не остання серія у сезоні та якщо так, відміняє операцію.

c. Створити тригер, що забороняє зміну поля сезон таблички епізодів.

d. Створити тригер, що забороняє додавати більше 12 різних сезонів (при чому номер сезону може біти >12, наприклад 1, 2 та 20 сезони – всього їх 3, отже вимога не порушується)

е. Створити 3 тригера (по одному додавання, зміна та видалення) до одної з табличок із свого варіанта індивідуального завдання. Зміст тригера задати за власним бажанням.

Практична робота № 10

SQLite. Графічний інтерфейс

1. Майже будь-яка сучасна СКБД має графічний інтерфейс менеджера баз даних. Менеджер баз даних часто допомагають розробникам і прискорюють процес розробки баз даних. Окрім командного рядку (sqlite.exe), який по праву можна назвати менеджером баз даних SQLite, існують інші менеджери, які можуть працювати з базами даних під управлінням SQLite3. Їх загальний недолік – всі вони від сторонніх розробників. Переваги: при використанні графічних менеджерів ми пишемо менше коду. Далі наведемо загальний огляд трьох менеджерів баз даних SQLite3:

SQLiteStudio - розповсюджується безкоштовно, може бути завантажено з офіційного сайту SQLiteStudio. Менеджер БД SQLiteStudio є крос платформним, на сторінці завантажень можна обрати відповідну версію. Цей менеджер БД встановлюється шляхом розпакування архіву у будь-яку папку на вашому комп'ютері.

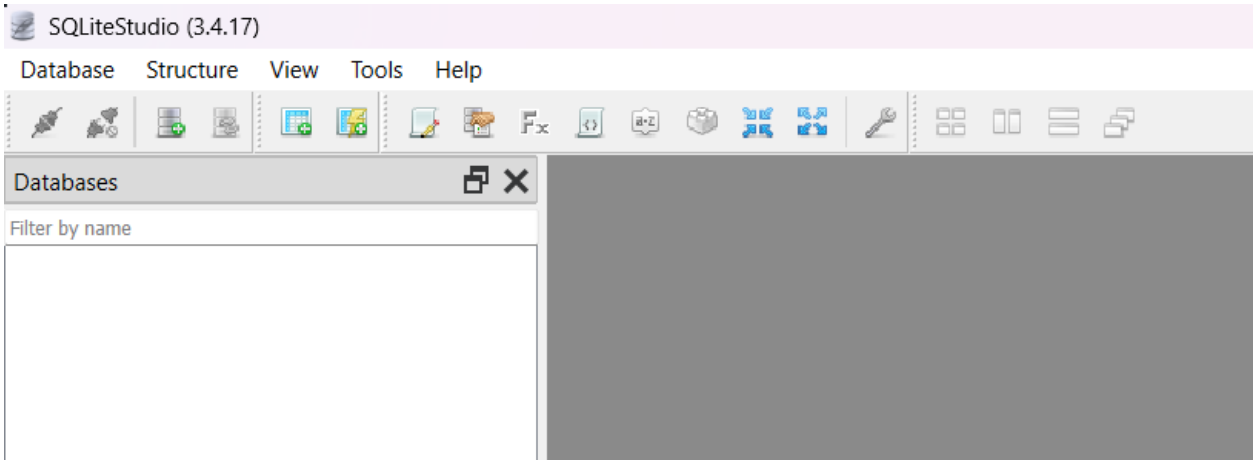
SQLite Manager – плагін, що представляє собою зручний менеджер баз даних SQLite3. Функціонал менш багатий, ніж у попередньої програми, але його у більшості випадків цілком достатньо..Плагін крос платформний, працює на будь-якій ОС, де є Firefox. Може бути завантажений на офіційній сторінці.

DBeaver – потужний та безкоштовний засіб для проектувальника БД, підтримує синтаксис багатьх СКБД, у тому числі і SQLite. Перераховувати можливості даного менеджера аз даних не має сенсу, оскільки він вимагає окремого вивчення. Менеджер баз даних DBeaver написан на Java, а значить

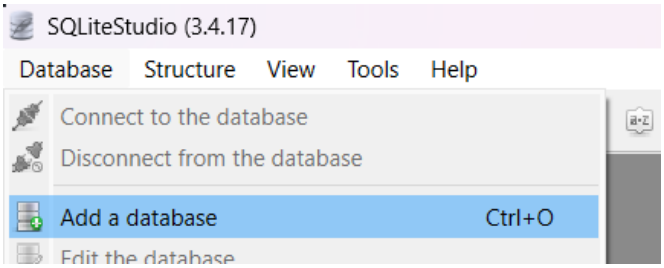
він крос платформний і для його роботи необхідно встановити JRE. Завантажити DBeaver мажна з офіційного сайту.

Насправді для бібліотеки SQLite3 існує набагато більша кількість менеджерів: як безкоштовних, так и платних. Але для досвідчених користувачів в 95 випадків із 100 зручніше та швидше користуватись консоллю.

2. Далі розглянемо роботу із SQLiteStudio. Після першого запуску головне вікно виглядає наступним чином



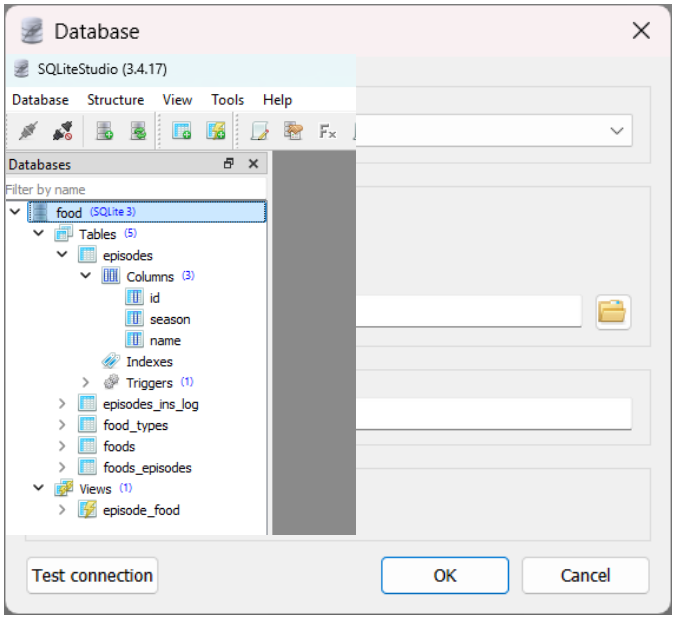
Для створення нової БД, або підключення існуючої (тут принцип такий самий, як і в консолі – при відсутності БД вона автоматично створюється)



виконуємо «База даних» -> «Додати базу даних»

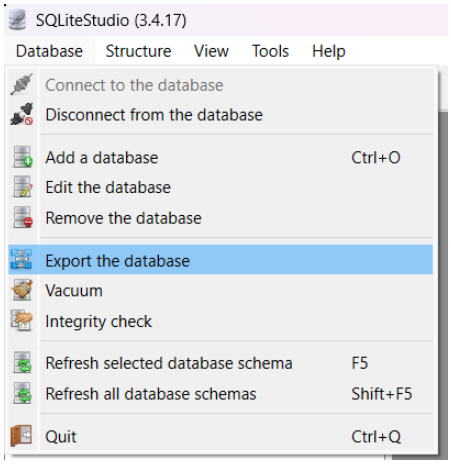
Далі обираємо або набираємо шлях до файлу БД, ім’я БД у списку баз даних SQLite Studio та натискаємо ок.

3. Підключившись до БД (подвійний клік на БД у списку або кнопка



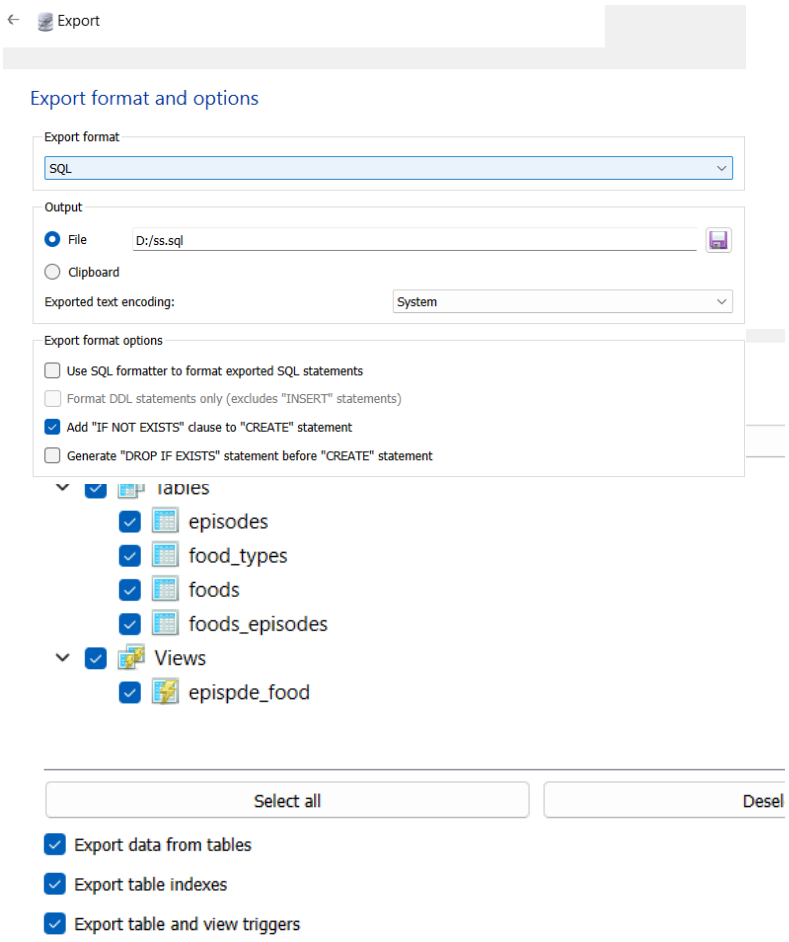
«Підключитися до бази даних» на панелі інструментів можемо переглянути список табличок та представлень БД

Зробимо експорт БД в sql формат. Для цього оберемо «База даних» => «Експортувати базу даних»



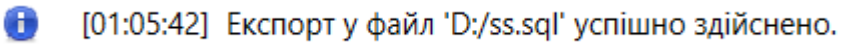
Відмітимо таблички та представлення, що необхідно експортувати

Оберемо необхідний формат даних та шлях до файлу, у який буде



виконано експорт БД

Та натискаємо finish. Після завершення отримаємо повідомлення у вікно «статус»



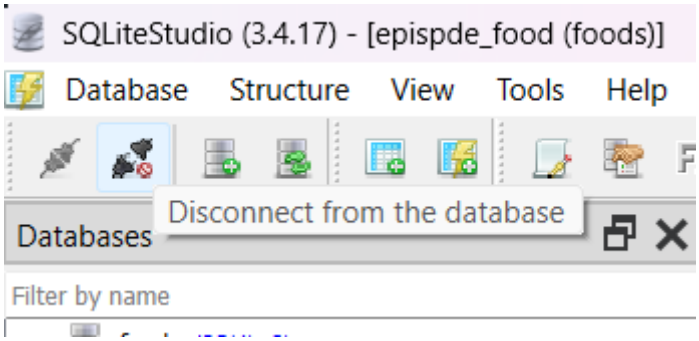
Натиснувши праву кнопку миші на табличці, і обравши в контекстному меню «імпортувати дані у таблицю» або «експортувати таблицю» можна виконати відповідну операцію для окремої таблиці БД.

При деталізації вкладених об’єктів таблички (або представлення) у дереві метаданих «Бази даних» можемо переглянути окремі стовбці таблички, їх типи даних, а також індекси та тригери, що були створені для цієї таблиці.

Користуючись контекстним меню можна додати новий стовбець, видалити існуючий, чи змінити його ім’я та тип даних. Те ж саме має відношення до індексів та тригерів.

Зверніть увагу, що оскільки обмеження синтаксису команди alter table мови SQLite не дозволяє змінювати або видаляти колонку таблички, в цьому випадку буде згенеровано послідовність команд. Спочатку дані таблички будуть збережені в тимчасовій таблиці, потім початкова таблиця буде знищена і створена із новою структурою. Після цього дані будуть перенесені назад. Для запобігання порушення посилальної цілісності перед початком буде відключений режим контролю по зовнішнім ключам, а після завершення знову відновлено.

4. Користуючись графічним інтерфейсом, створимо нову БД. За бажанням можемо від’єднатися від foods.db



Та додамо нову БД students_list.db

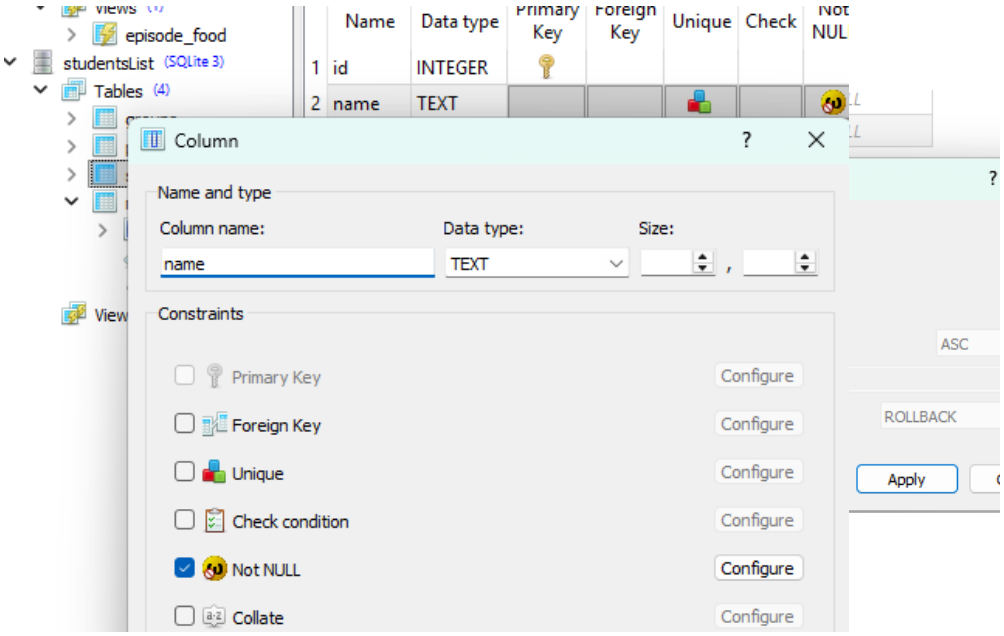
5. Створимо таблицю групи

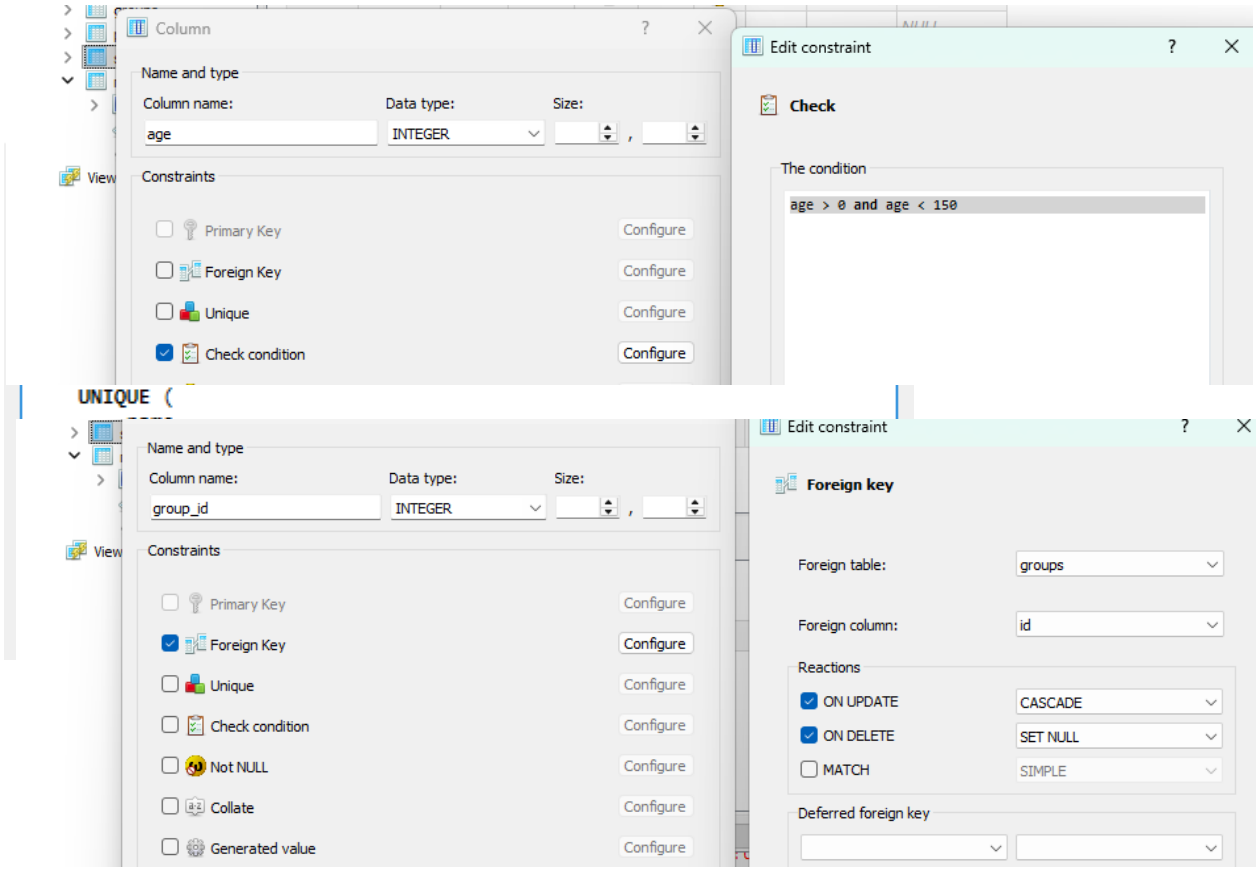
Після вказання необхідних полів та типів даних на панелі інструментів натискаємо «підтвердити зміну структури». Та виконуємо спеціально згенерований фрагмент коду. Перейдемо на вкладку «дані» та додамо до таблички декілька рядків. Після введення даних натиснемо кнопку «підтвердити». Зверніть увагу, що автоінкрементні та стандартні поля

	id	number	stud_quantity
1	1	103	0
2	2	102	0
3	3	101	0

заповнилися автоматично.

6. За аналогією створимо табличку студент(код, ім’я, вік, група)

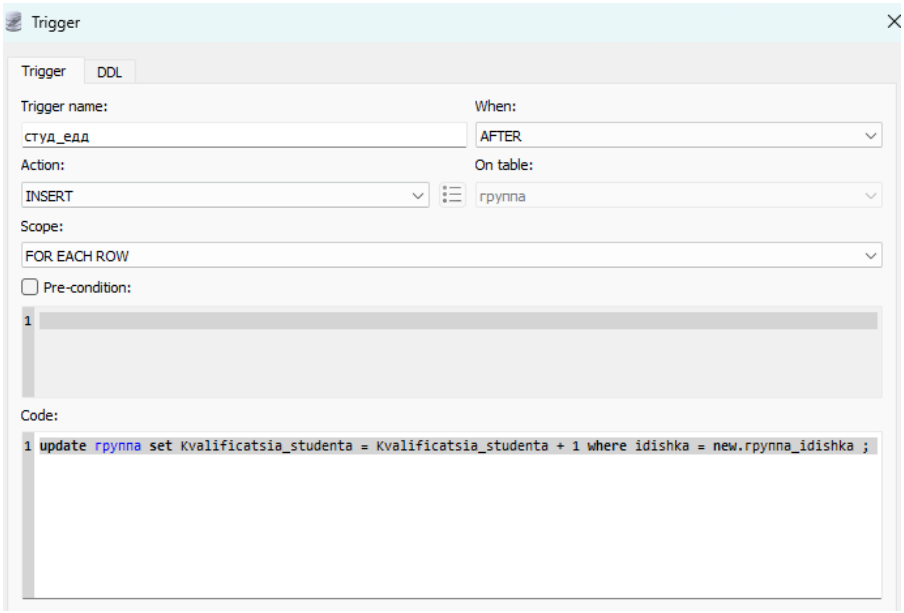


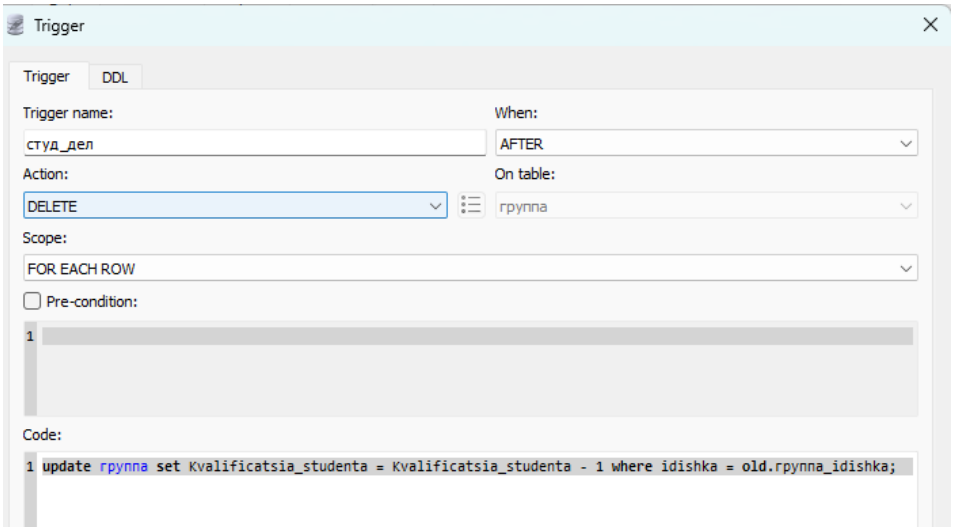


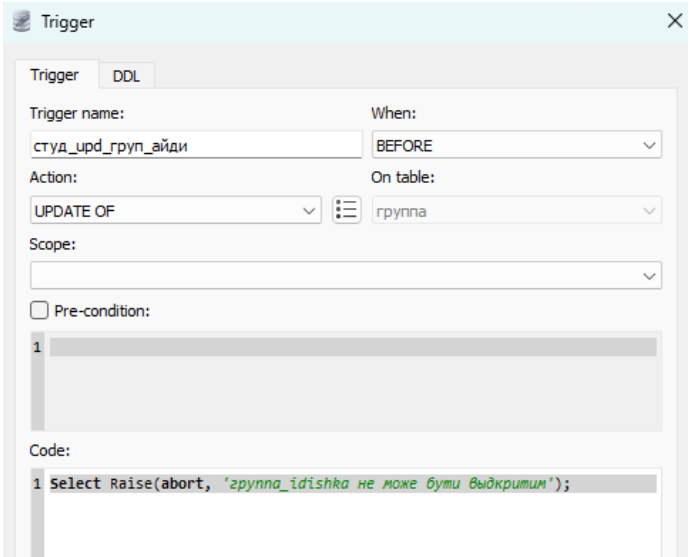
7. Виконуємо згенерований скрипт

Перед заповненням таблиці студентів даними, створимо тригери для підтримки в актуальному стані поля «кількість студентів» таблиці «студенти». При додаванні студента його потрібно збільшити для відповідної

групи, при видаленні зменшити. Для спрощення задачі додатково створимо тригер, що заборонить зміну поля «група» таблиці «студенти».


Trigger
Trigger
DDL
Trigger name:
студ_едд
When:
AFTER
Action:
INSERT
On table:
группа
Scope:
FOR EACH ROW
Pre-condition:
1
Code:
1 update группа set Kvalificatsia_studenta = Kvalificatsia_studenta + 1 where idishka = new.группа_idishka ;


Trigger
Trigger
DDL
Trigger name:
студ_дел
When:
AFTER
Action:
DELETE
On table:
группа
Scope:
FOR EACH ROW
Pre-condition:
1
Code:
1 update группа set Kvalificatsia_studenta = Kvalificatsia_studenta - 1 where idishka = old.группа_idishka;


Trigger
Trigger
DDL
Trigger name:
студ_upd_груп_айди
When:
BEFORE
Action:
UPDATE OF
On table:
группа
Scope:
Pre-condition:
1
Code:
1 Select Raise(abort, 'группа_idishka не може бути выдкритим');

7. Додамо до таблиці студенти декілька рядків та перевіримо роботу

	id	name	age	group_id
1	1	Петров	17	1
2	NULL	Пристарелов	200	1
3	NULL	Сидоров	18	2
4	NULL	Іванов	18	1

тригерів та правила.

При підтвердженні отримуємо порушення правила

Видаляємо помилковий рядок та повторюємо операцію. Переходимо до перегляду вмісту таблички «групи»

[15:01:52] Error while committing new row: CHECK constraint failed: student

	id	number	stud_quantity
1	1	103	2
2	2	102	1
3	3	101	0

Як бачимо, вміст поля «кількість студентів» був змінений автоматично при додаванні студентів у групу.

Спробуємо змінити номер групи у студента та переконуємось, що наш

	id	name	age	group_id
1	1	Петров	17	1
2	2	Сидоров	18	2
3	3	Іванов	18	3

[15:06:04] An error occurred while committing the data: group_id cannot be updated !

тригер не дозволяє цього зробити

Видаляємо студента та перевіряємо зміну кількості студентів у групі

8. Створимо представлення для відображення списку студентів із номером групи, в якій він навчається.

На вкладці дані переглянемо результат

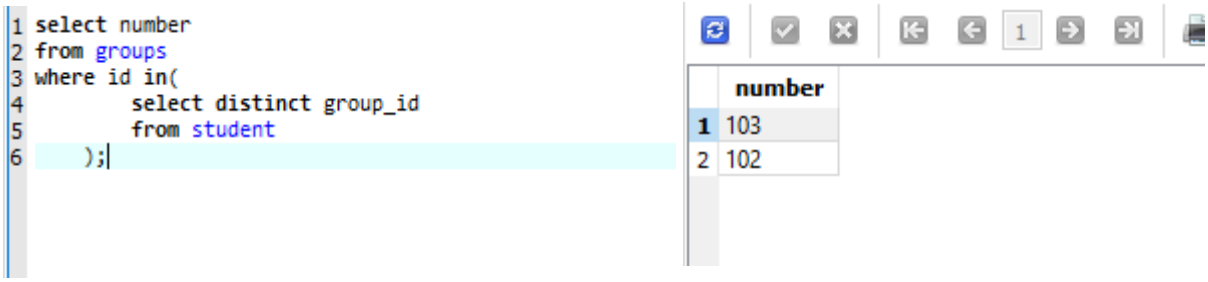
	id	name	age	group_id
1	1	Петров	17	1

	name	number
1	Сидоров	102
2	Петров	103

```
2 from student s inner join groups g
3 on s.group_id = g.id
4 order by g.number, s.name;
```

9. Напишемо довільний SQL-запит. Для цього відкриємо редактор

Та наберемо наступний запит, що має повернути список номерів



непорожніх груп (в який навчаються студенти).

Завдання для самостійної роботи

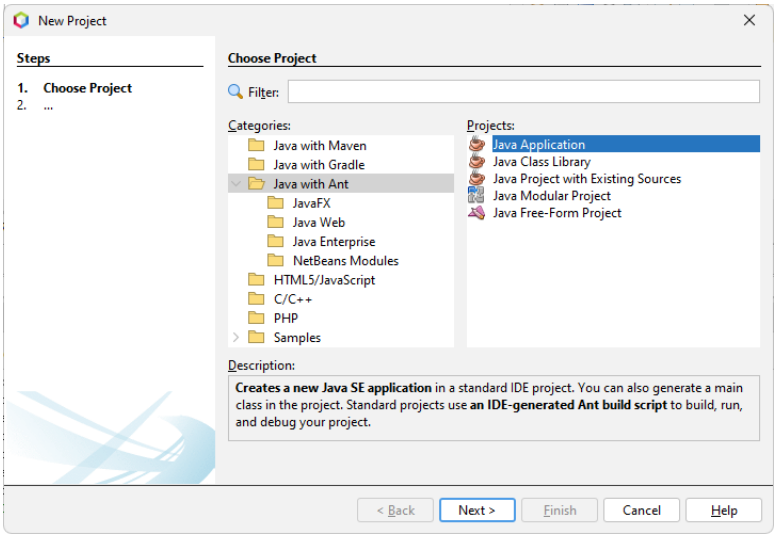
Користуючись графічним додати до БД, що була виконана у п.12 ПР № 5, всі таблиці з індивідуального завдання згідно вашого варіанту.

Практична робота № 11

Написання desktop-застосунку на мові Java та C++

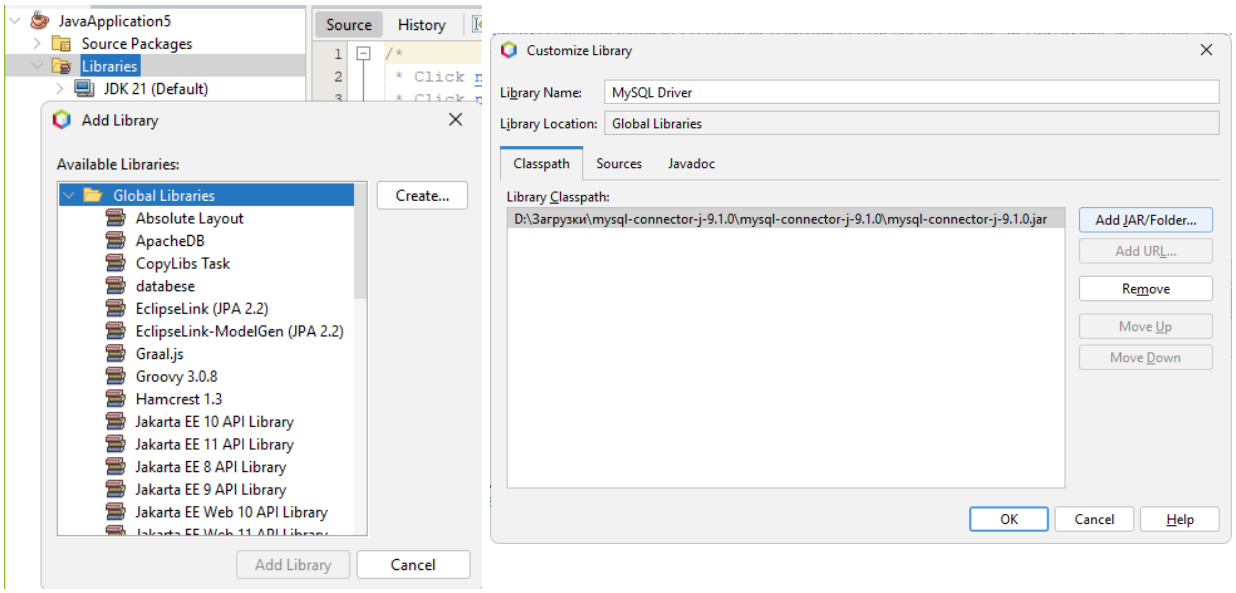
1. Розробка додатку на Java. Робота з базою даних

Для зберігання даних, як правило, використовуються бази даних. У додатках на Java ми можемо використовувати різні системи управління базами даних. У середовище NetBeans вбудовано JavaDB (Apache Derby), а також є можливість підключення інших, ми будемо користуватися MySQL. Для того щоб підключитись, спочатку запустимо сервер бази даних(*Практична робота № 10.* ***Встановлення MySQL Server***). Потім з

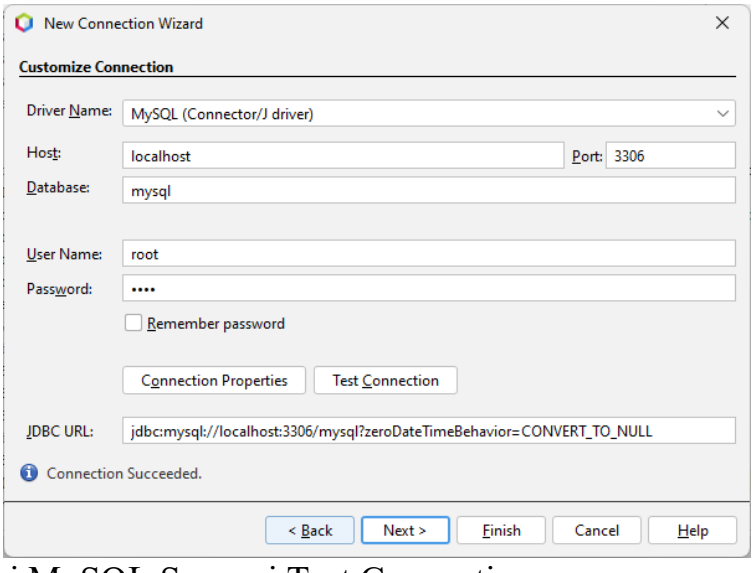


Moodle завантажимо файл ***mysql-connector-j-9.1.0***. Створимо новий проект:

Додамо до створеного проекту бібліотеку MySQL Driver та завантажимо у середовище розробки файл ***mysql-connector-j-9.1.0*** (ПКМ по *Libraries - Add Library - Create*. Назву бібліотеки задаймо самостійно. *Add jar* і файл конектору.

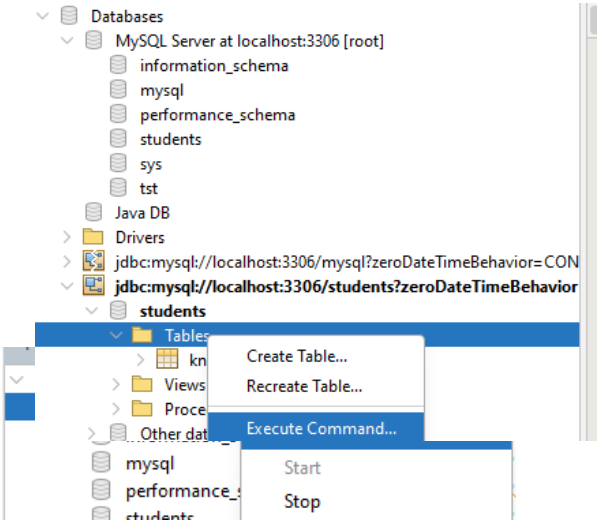


У навігаційному меню зліва перейдемо на вкладку «Services» - «Databases» та натиснемо «New Connection». Як драйвер оберемо MySQL (Connector/ J driver) та натиснемо Add та додамо знов файл ***mysql-connector-j-9.1.0***. Далі з’явиться вікно, де введемо тільки пароль, який вказували при встановленні MySQL Server і Test Connection.



У навігаційному меню зліва перейдемо на вкладку «Services» - «Databases» та натиснемо «Register MySQL Server» та знову введемо наш пароль. Після появи у «Databases» поля «MySQL Server at localhost» створемо нову базу даних, як на рисунку:

І у нас нижче з'явиться створена база даних. Натиснемо ПКМ та «connect». Після папки Drivers з'явиться *jbdc: mysql:/localhost:3306/Database_Name.*

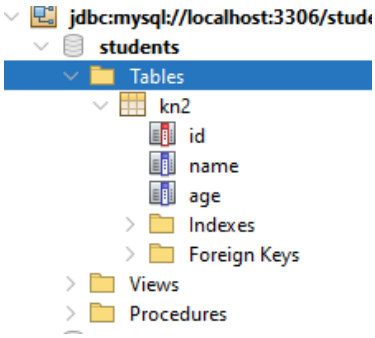


Створимо нову таблицю, як показано на рисунку нижче.

Примітка: при створенні бази з полем id, що має атрибут auto_increment, краще використовувати командний рядок ,ніж графічний інтерфейс, тому опція auto_increment у NetBeans виглядає наступним чином: *GENERATED ALWAYS AS IDENTITY* , тому краще замість «Create table», натиснути «Execute command» по назві нашої БД та по завершенню написанні SQL транзакції створення таблиці натиснути на . Приклад створення простої БД:

```
create table kn2 (
id INTEGER NOT NULL AUTO_INCREMENT,
name VARCHAR(255) NOT NULL,
age INTEGER,
PRIMARY KEY (id)
);
```

Оскільки, таблиця створена, перейдемо до наступного розділу



підключення БД до нашого коду додатку.

Підключення БД у Java коді

У JDBC є 3 основні інтерфейси:

Connection – відповідає за з'єднання з базою даних

Statement – відповідає за запит до бази даних

ResultSet – відповідає за результат запиту до бази даних

Зчитування даних

1. Додамо імпорти.

```
import java.sql.Connection;
import java.sql.Statement;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
```

2. Створимо підключення до бази даних:

```
Class.forName("com.mysql.cj.jdbc.Driver");
Connection                              conn                                    =
DriverManager.getConnection("jdbc:mysql://localhost:3306/students","ro
ot","1234");
```

3. Створюємо запит до бази даних. Виберемо всіх студентів із таблиці kn2. Треба додати такий рядок коду:

```
String SQL_SELECT = "SELECT * FROM kn2";
Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(SQL_SELECT);
```

У першому рядку ми створюємо об'єкт Statement, а у другому з його допомогою виконуємо запит до бази даних. Метод executeQuery() виконує запит до бази даних та повертає об'єкт типу ResultSet.

4. Виведемо на екран дані, що містяться у об'єкті ResultSet.

У ResultSet зберігається результат виконання запиту. Цей об'єкт чимось схожий на ітератор: він дозволяє встановлювати/міняти поточний рядок результату, а потім із цього поточного рядка можна отримати дані. Додаймо такий код:

```
while(rs.next()){
         int id = rs.getInt("id");
         System.out.println("Student with ID: "+ id);
         String name = rs.getString("name");
         System.out.println(" has name: "+ name);
         int age= rs.getInt("age");
         System.out.println(" and is" + age + " y.o.");
         }
        rs.close();
```

Метод next() змінює поточний рядок результату на наступний. Він повертає true, якщо такий рядок є, і false, якщо рядки закінчилися.

З поточного рядка об'єкта ResultSet можна отримати дані з колонок:

getRow() – повертає номер поточного рядка в об'єкті ResultSet

getInt(N) – поверне дані N-колонки поточного рядка як тип int

getString(N) – поверне дані N-колонки поточного рядка як тип String

5. Нижче наведено основні транзакції для додавання, оновлення та видалення запису з таблиці.

```
String SQL_INSERT = "INSERT INTO kn2 (name, age) VALUES ('?',?)";
String SQL_UPDATE="UPDATE kn2 SET name='?' WHERE id=?";
String SQL_DELETE="DELETE FROM kn2 WHERE id=?";
PreparedStatement ps = conn.prepareStatement(SQL_STATEMENT);
ps.executeUpdate();
```

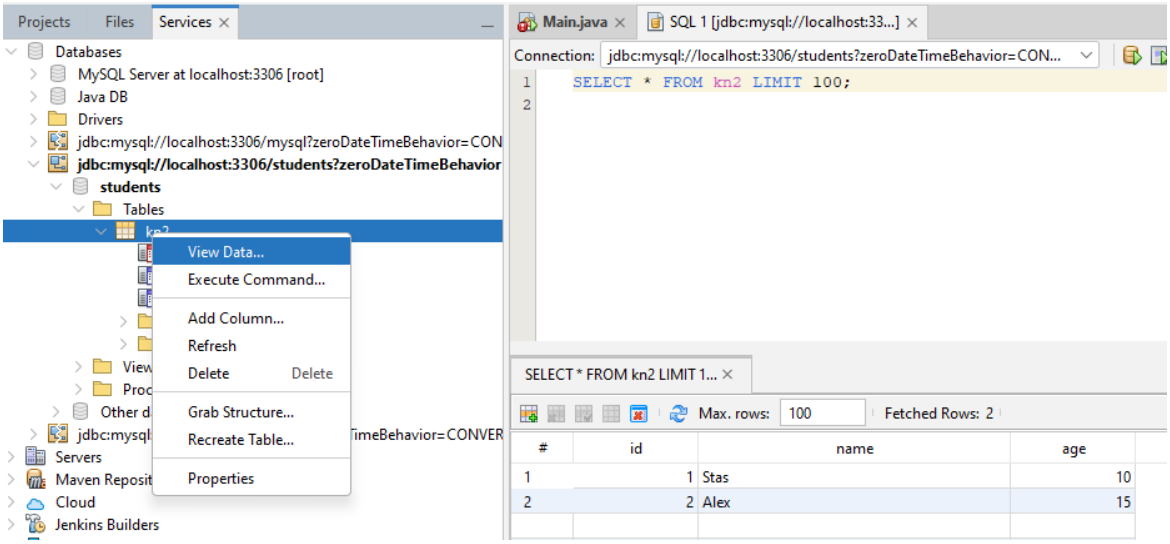
Об'єкт PreparedStatement дозволяє об'єднати декілька команд SQL в одну групу і передати їх на обробку до бази даних у пакетному режимі.

Для виконання запиту PreparedStatement має три методи:

boolean execute(): виконує будь-яку SQL-команду,

ResultSet executeQuery(): виконує команду SELECT, яка повертає дані у вигляді ResultSet,

int executeUpdate(): виконує такі SQL-команди, як INSERT, UPDATE, DELETE, CREATE та повертає кількість змінених рядків.



В самому NetBeans можна переглянути готову таблицю.

2. Розробка додатку на C++. Робота з базою даних

MyForm.h

```
#pragma once
#include <cliext/vector>
#include <cliext/vector>
namespace Проект1 {

	using namespace System;
	using namespace System::ComponentModel;
	using namespace System::Collections;
	using namespace System::Windows::Forms;
	using namespace System::Data;
	using namespace System::Drawing;
	using namespace System::Data::SQLite;
	using namespace System::Text;
	using namespace cliext;
	/// <summary>
	/// </summary>
	public ref class MyForm : public System::Windows::Forms::Form
	{
	public:
		DataTable^ table;
		DataColumn^ column;
		DataRow^ row;
		MyForm(void)
		{
			InitializeComponent();
			//
			//TODO:                    //
		}
	private: System::Windows::Forms::Button^  button3;
	private: System::Windows::Forms::Button^  button4;
	public:

	SQLiteConnection ^db;
	protected:
		/// <summary>
				/// </summary>
		~MyForm()
		{
			if (components)
			{
```

```
                        delete components;
                    }
                    db->Close();
                }
        private: System::Windows::Forms::Button^  button1;
        private: System::Windows::Forms::TextBox^  textBox1;
        private: System::Windows::Forms::OpenFileDialog^  openFileDialog1;
        private: System::Windows::Forms::Button^  button2;
        private: System::Windows::Forms::DataGridView^  dataGridView1;
        protected:

        private:
                /// <summary>
                /// </summary>
                System::ComponentModel::Container ^components;

#pragma region Windows Form Designer generated code
                /// <summary>
                /// </summary>
                void InitializeComponent(void)
                {
                        System::ComponentModel::ComponentResourceManager^  resources =
(gcnew System::ComponentModel::ComponentResourceManager(MyForm::typeid));
                    this->openFileDialog1 = (gcnew
System::Windows::Forms::OpenFileDialog());
                    this->button2 = (gcnew System::Windows::Forms::Button());
                    this->dataGridView1 = (gcnew
System::Windows::Forms::DataGridView());
                    this->button3 = (gcnew System::Windows::Forms::Button());
                    this->button4 = (gcnew System::Windows::Forms::Button());

        (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>dataGridView1))->BeginInit();
                    this->SuspendLayout();
                    //
                    // openFileDialog1
                    //
                    this->openFileDialog1->Filter = L"Файл БД(*.db)|*.db";
                    //
                    // button2
                    //
                    this->button2->Location = System::Drawing::Point(218, 12);
                    this->button2->Name = L"button2";
                    this->button2->Size = System::Drawing::Size(93, 23);
                    this->button2->TabIndex = 2;
                    this->button2->Text = L"Відкрити БД";
                    this->button2->UseVisualStyleBackColor = true;
                    this->button2->Click += gcnew System::EventHandler(this,
&MyForm::button2_Click);
                    //
                    // dataGridView1
                    //
                    this->dataGridView1->ColumnHeadersHeightSizeMode =
System::Windows::Forms::DataGridViewColumnHeadersHeightSizeMode::AutoSize;
                    this->dataGridView1->Location = System::Drawing::Point(47, 47);
                    this->dataGridView1->Name = L"dataGridView1";
                    this->dataGridView1->Size = System::Drawing::Size(438, 266);
                    this->dataGridView1->TabIndex = 3;
                    //
                    // button3
                    //
                    this->button3->Location = System::Drawing::Point(85, 319);
                    this->button3->Name = L"button3";
                    this->button3->Size = System::Drawing::Size(75, 23);
                    this->button3->TabIndex = 4;
                    this->button3->Text = L"Додати";
                    this->button3->UseVisualStyleBackColor = true;
                    this->button3->Click += gcnew System::EventHandler(this,
&MyForm::button3_Click);
                    //
```

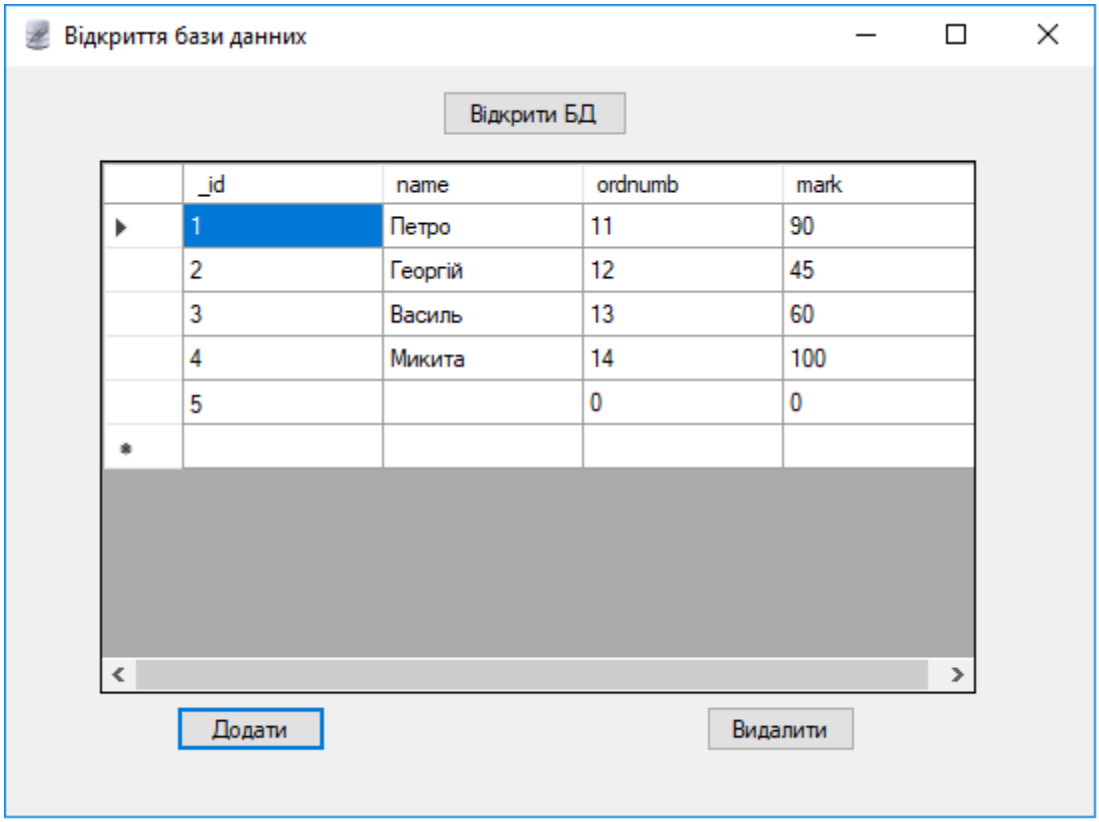
```
                    // button4
                    //
                    this->button4->Location = System::Drawing::Point(350, 319);
                    this->button4->Name = L"button4";
                    this->button4->Size = System::Drawing::Size(75, 23);
                    this->button4->TabIndex = 5;
                    this->button4->Text = L"Видалити";
                    this->button4->UseVisualStyleBackColor = true;
                    this->button4->Click += gcnew System::EventHandler(this,
&MyForm::button4_Click);
                    //
                    // MyForm
                    //
                    this->AutoScaleDimensions = System::Drawing::SizeF(6, 13);
                    this->AutoScaleMode =
System::Windows::Forms::AutoScaleMode::Font;
                    this->ClientSize = System::Drawing::Size(544, 374);
                    this->Controls->Add(this->button4);
                    this->Controls->Add(this->button3);
                    this->Controls->Add(this->dataGridView1);
                    this->Controls->Add(this->button2);
                    this->Icon = (cli::safe_cast<System::Drawing::Icon^>(resources-
>GetObject(L"$this.Icon")));
                    this->Name = L"MyForm";
                    this->Text = L"Відкриття бази данних";

       (cli::safe_cast<System::ComponentModel::ISupportInitialize^>(this-
>dataGridView1))->EndInit();
                    this->ResumeLayout(false);

             }
#pragma endregion

                     private: DataTable^ fillDataTable() {

                            try
                            {
                                   SQLiteCommand ^cmdSelect = db->CreateCommand();
                                                cmdSelect->CommandText = "SELECT *
FROM students;";
                                   SQLiteDataReader ^reader = cmdSelect-
>ExecuteReader();
                                   table = gcnew DataTable();
                                   vector<String^>^ nameColumns = gcnew
vector<String^>();
                                   for (int i = 0; i < reader->FieldCount; i++) {
                                          nameColumns->push_back(reader-
>GetName(i));
                                          column = gcnew DataColumn(nameColumns-
>at(i), String::typeid);
                                          table->Columns->Add(column);
                                   }
                                   while (reader->Read()) {

                                          row = table->NewRow();
                                          for (int i = 0; i < reader->FieldCount;
i++) {
                                                 row[nameColumns->at(i)] = reader-
>GetValue(i)->ToString();
                                                 reader->GetValue(i)->ToString();
                                          }
                                          table->Rows->Add(row);
                                   }
                            }
                            catch (Exception ^e)
                            {
                                   MessageBox::Show("Error Executing SQL: " + e-
>ToString(), "Exception While Displaying MyTable ...");
                            }
                            return table;
```

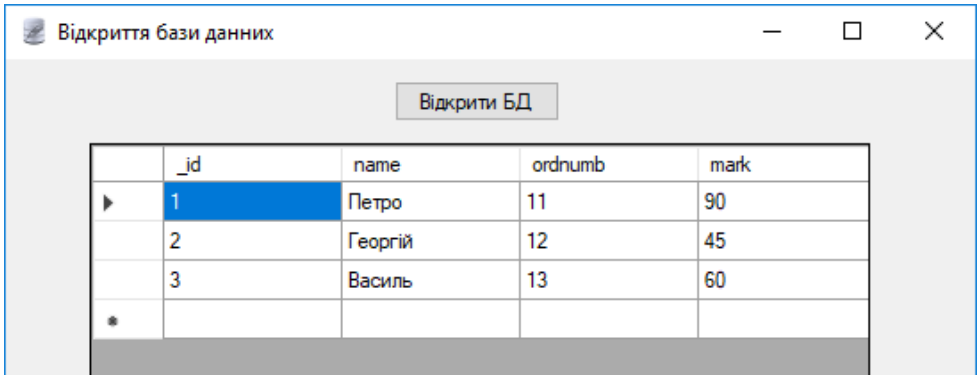
```
                        }
        private: System::Void button2_Click(System::Object^  sender,
System::EventArgs^  e) {
                if (openFileDialog1->ShowDialog() ==
System::Windows::Forms::DialogResult::OK) {
                        String^ fileName = openFileDialog1->FileName;

                        db = gcnew SQLiteConnection();
                        try
                        {
                                db->ConnectionString = "Data Source=\"" + fileName +
"\"";
                                db->Open();

                                DataTable ^table = fillDataTable();

                                dataGridView1->DataSource = table;
                        }
                        catch (Exception ^e)
                        {
                                MessageBox::Show("Error Working SQL: " + e->ToString(),
"Exception ...");
                        }
                }
        }
private: System::Void button3_Click(System::Object^  sender, System::EventArgs^  e)
{
    int index = dataGridView1->CurrentCell->RowIndex;
        String^ name = dataGridView1->Rows[index]->Cells["name"]->Value->ToString();
        String^ ordnumb = dataGridView1->Rows[index]->Cells["ordnumb"]->Value-
>ToString();
        String^ mark = dataGridView1->Rows[index]->Cells["mark"]->Value->ToString();

        try
        {
                SQLiteCommand ^cmdInsertValue = db->CreateCommand();

                cmdInsertValue->CommandText = "INSERT INTO students (name,ordnumb,
mark) VALUES(" + name + ", " + ordnumb + ", " + mark + ");";

                cmdInsertValue->ExecuteNonQuery();

                DataTable ^table = fillDataTable();

                dataGridView1->DataSource = table;
        }
        catch (Exception ^e)
        {
                MessageBox::Show("Error Executing SQL: " + e->ToString(), "Exception
...");
        }
}
private: System::Void button4_Click(System::Object^  sender, System::EventArgs^  e)
{
        int index = dataGridView1->CurrentCell->RowIndex;

        String^ _id = dataGridView1->Rows[index]->Cells["_id"]->Value->ToString();

        try
        {
                SQLiteCommand ^cmdInsertValue = db->CreateCommand();
                cmdInsertValue->CommandText = "DELETE FROM students WHERE _id = " +
_id + ";";
                cmdInsertValue->ExecuteNonQuery();

                DataTable ^table = fillDataTable();

                dataGridView1->DataSource = table;
        }
        catch (Exception ^e)
```

```
        {
                MessageBox::Show("Error Executing SQL: " + e->ToString(), "Exception
...");
        }
}
private: System::Void textBox1_TextChanged(System::Object^  sender,
System::EventArgs^  e) {
}
};
}
```

MyForm.h

```
#include "MyForm.h"

using namespace System;
using namespace System::Windows::Forms;
[STAThreadAttribute]
void Main(array<String^>^ args) {
        Application::EnableVisualStyles();
        Application::SetCompatibleTextRenderingDefault(false);
        Проект1::MyForm form;
        Application::Run(%form);
}
```





Індивідуальне завдання

Завдання для самостійної роботи. На мові програмування на власний вибір реалізувати форму перегляду та редагування (додавання, зміни та видалення) даних таблиці із БД згідно свого варіанту індивідуального завдання.

Практична робота № 12

Введення в MySQL. Встановлення сервера. Підключення до MySQL.

MySQL представляє систему управління реляційними базами даних (СУБД). На сьогоднішній день це одна з найпопулярніших систем керування базами даних.

Початковим розробником цієї СУБД була шведська компанія MySQL AB. У 1995 році вона випустила перший реліз MySQL. У 2008 році компанія MySQL AB була куплена компанією Sun Microsystems, а в 2010 році компанія Oracle поглинула Sun і тим самим придбала права на торгову марку MySQL. Тому MySQL на сьогоднішній день розвивається під егідою Oracle.

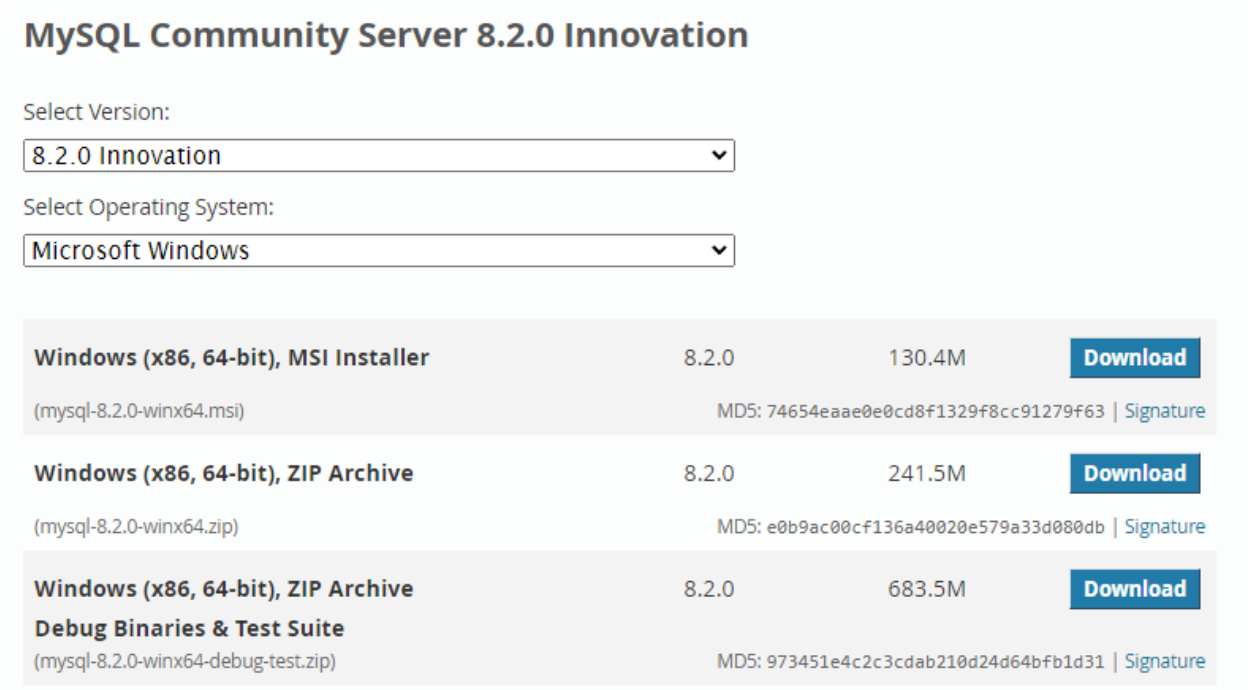
Поточною актуальною версією СУДБ є версія 8.2, яка вийшла у жовтень 2023 року, але для якої постійно виходять підверсії.

MySQL має кросплатформенність, є дистрибутиви під різні ОС, у тому числі найбільш популярні версії Linux, Windows, MacOS.

Офіційний сайт проекту: https://www.mysql.com/.

Встановлення MySQL

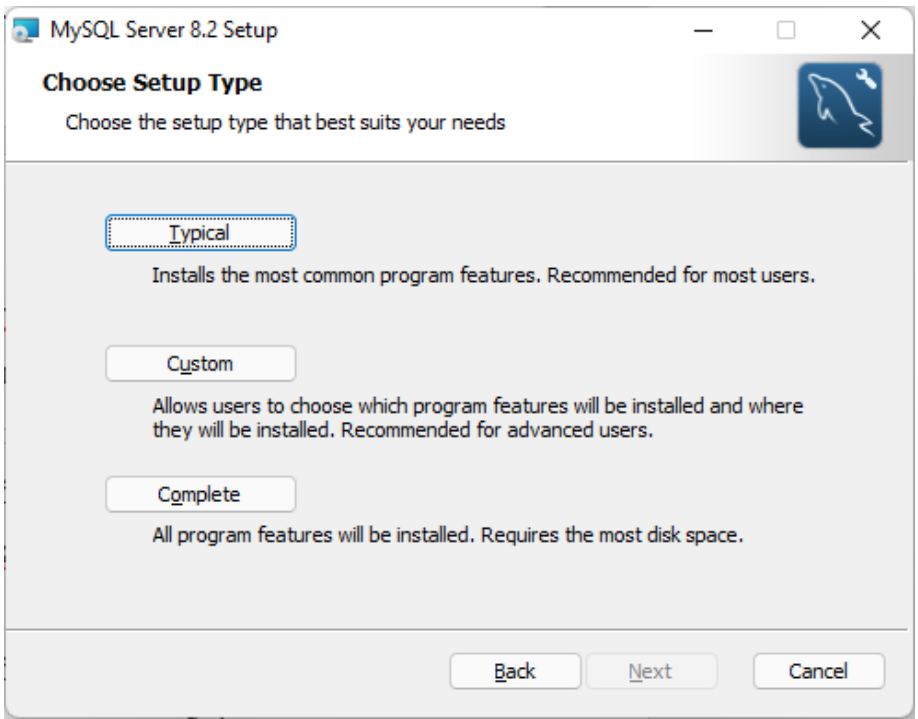
Для встановлення MySQL завантажимо дистрибутив за адресою



http://dev.mysql.com/downloads/mysql/ та виберемо потрібну версію.

Тут можна вибрати або ZIP Archive або повний пакет інсталятора MSI Installer. Після вибору версії натиснемо кнопку "Download", і нас перенаправить на сторінку завантаження дистрибутива. Далі може бути запропоновано увійти за допомогою облікового запису Oracle, але нижче можна натиснути на посилання "No thanks, just start my download" і почнеться завантаження без реєстрації.

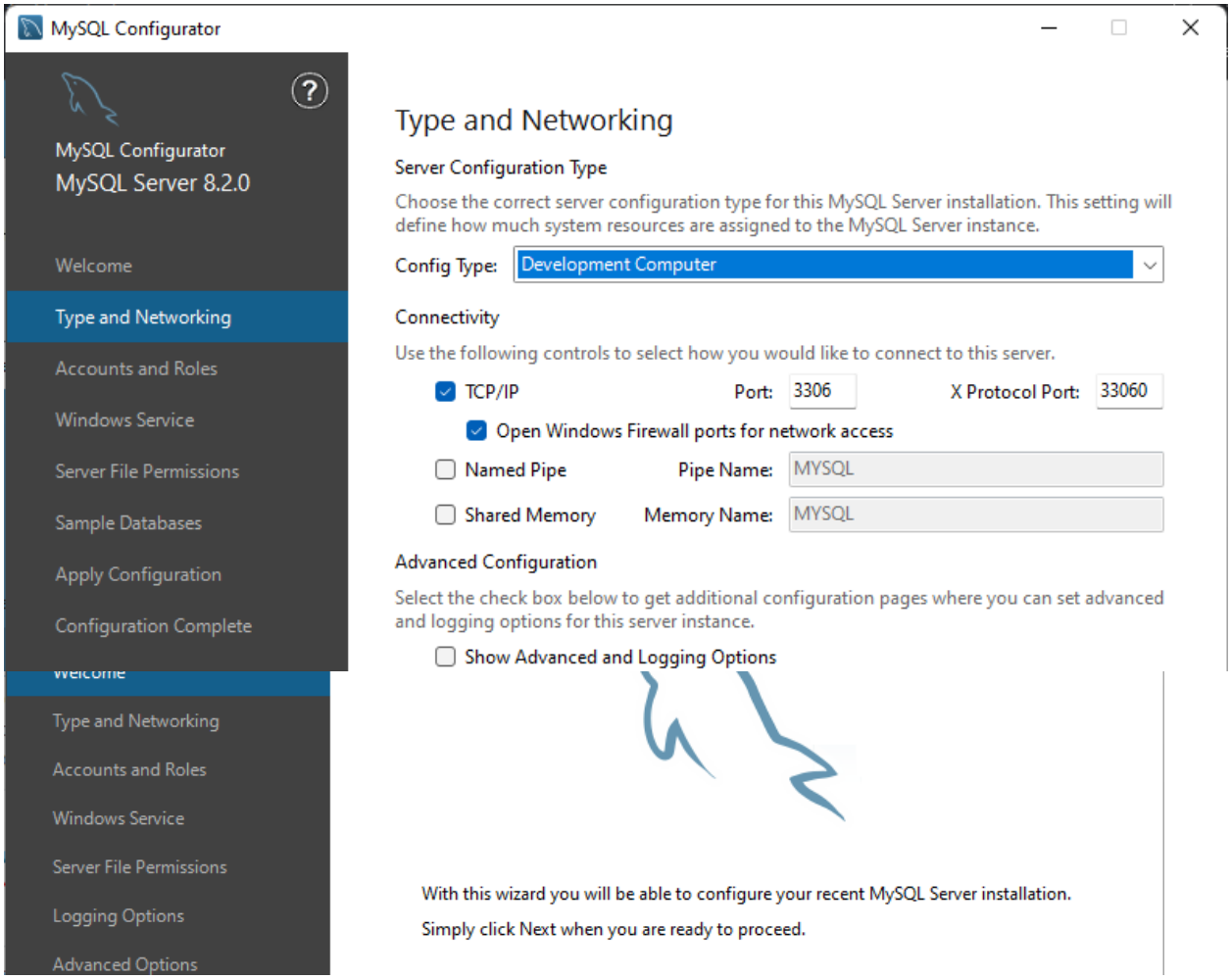
Спочатку буде запропоновано вибрати тип установки. Виберемо тип



Typical, якого вистачить для базових потреб, і натиснемо на кнопку Next:

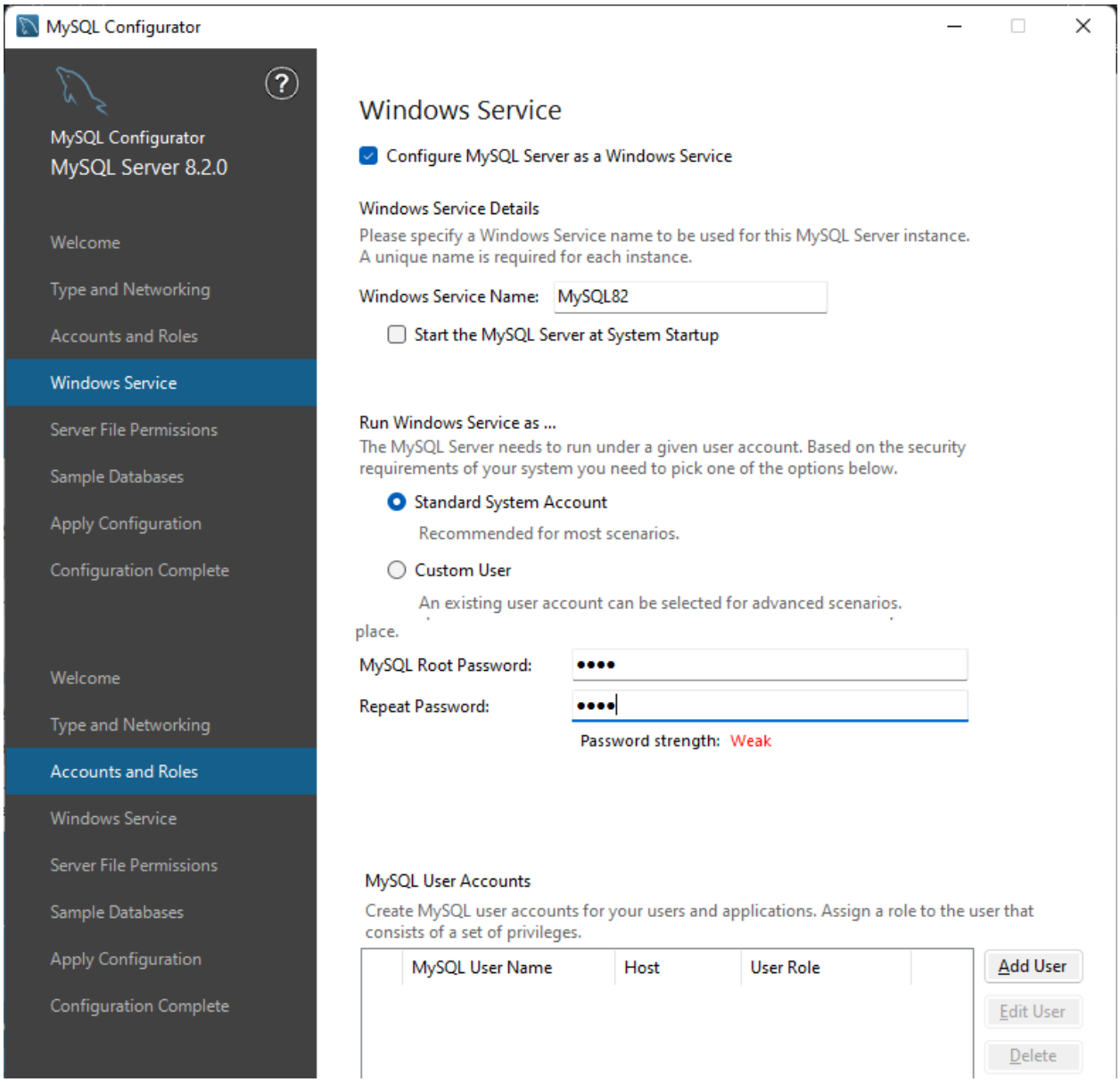
Після встановлення відкриємо конфігуратор та виконаємо налаштування.

Натисніть кнопку Next і далі буде запропоновано встановити ряд конфігураційних налаштувань сервера MySQL. Зокрема, тут ми бачимо, що для підключення буде застосовуватись протокол TCP/IP та порт 3306. Залишимо всі ці налаштування з'єднання та порту за замовчуванням:



Потім на наступному вікні програми установки вкажемо будь-який пароль, і запам'ятаємо його, тому що він потім буде потрібний при підключенні до сервера MySQL:

Наступний набір конфігурацій, який також залишимо за замовчуванням, але приберемо щоб сервер не запускався як служба Windows



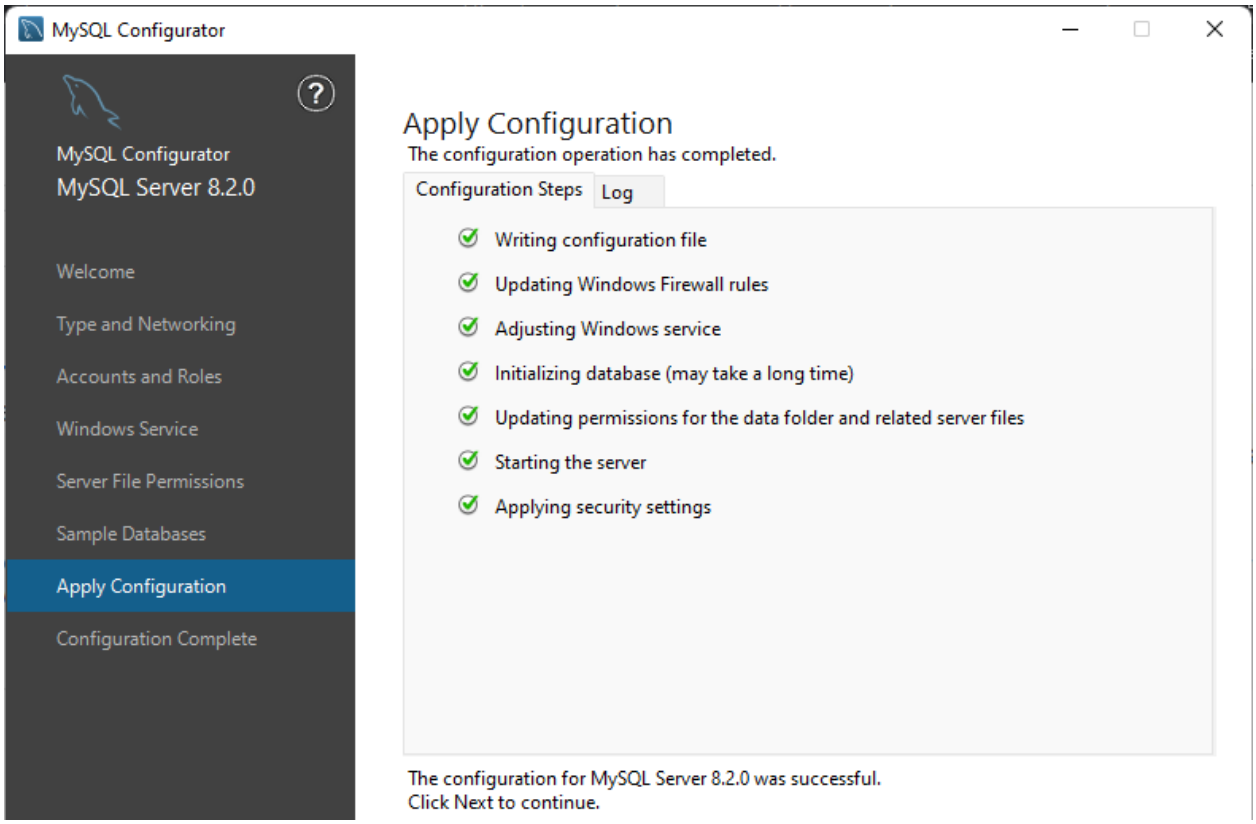
під час запуску операційної системи:

Натискаємо кнопку Next та дійдемо до наступного екрану, де необхідно застосувати всі раніше встановлені конфігураційні налаштування, натиснувши кнопку Execute:

Після застосування конфігураційних налаштувань сервер MySQL буде повністю встановлений та налаштований, натиснемо на кнопку "Finish". Якщо потрібно переналаштувати сервер, то MySQL 8.2 Configurator можна знайти у меню пуск – всі програми.

Вступ до Node JS. Що таке Node JS. Початок роботи

Node.js представляє середовище виконання коду JavaScript, яке побудоване на основі рушія JavaScript Chrome V8, який дозволяє транслювати виклики мовою JavaScript в машинний код. Node.js насамперед призначений для створення серверних програм на мові JavaScript. Ми говоримо про Node.js як про платформу для створення веб-додатків.



Особливістю Node.js є те, що Node.js використовує один (основний) потік, який отримує всі запити та керує ними через чергу запитів (таким чином, Node.js є однопотоковим сервером). Усередині цього потоку виконується так званий цикл подій (event loop), який безперервно перевіряє

запити з черги подій і обробляє події введення та виведення.

Якщо користувач надсилає запит на сервер Node.js, у циклі подій виконується перевірка, щоб визначити, чи вимагає наступний запит блокуючої операції введення або виведення (наприклад, звернення до бази даних або файлової системи). Якщо ні, запит обробляється безпосередньо, і користувачеві відправляється результат обробки.

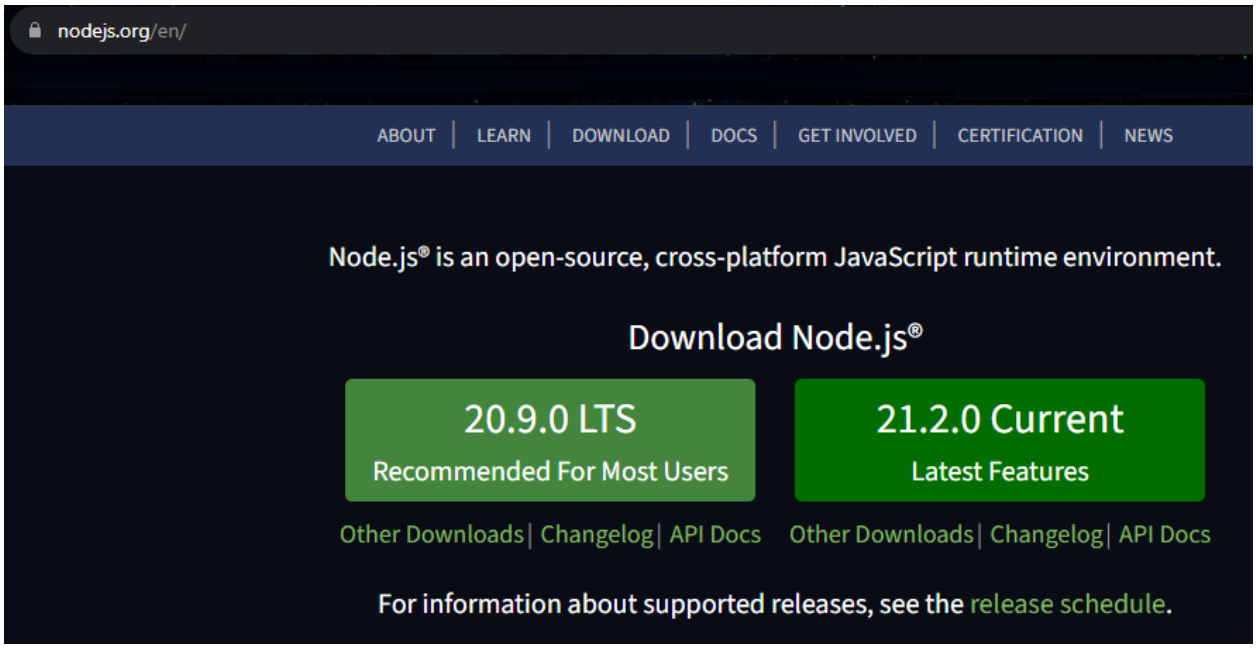
Якщо запит вимагає блокуючої операції введення/виводу, для виконання цієї операції запускається один з декількох внутрішніх обробників Node.js (потоків). В обробники передається функція зворотного виклику, яка, у свою чергу, викликається, як тільки буде виконана блокуюча операція вводу/виводу.

При цьому основний потік не зупиняється під час блокуючих операцій введення/виводу: цикл подій постійно здійснює свої обходи, перевіряє нові запити тощо. Завдяки цій архітектурі Node.js запобігає створенню все більшої кількості потоків та надмірного використання пам'яті.

Node.js є відкритим проектом, який можна побачити на github.com.

Встановлення

Для завантаження перейде на офіційний сайт https://nodejs.org/en/. На головній сторінці ми відразу побачимо дві можливі опції для завантаження:



остання версія NodeJS та LTS-версія.

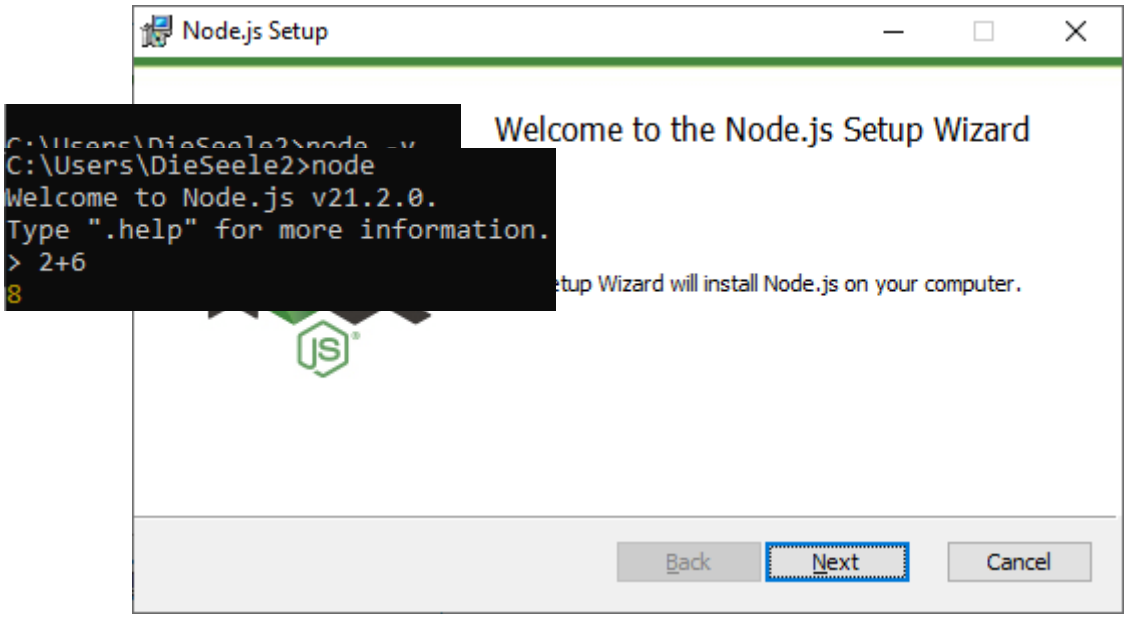
Завантажимо LTS-версію. Для Windows інсталятор представляє файл з розширенням msi. Після запуску відкриється програма інсталятора:

Після успішної установки можна ввести в командному рядку/терміналі команду *node -v* і відобразиться поточна версія node.js:

Інструменти розробки. Для розробки під Node JS можна використати, наприклад Visual Studio Code.

REPL

Після встановлення NodeJS нам стає доступним такий інструмент як REPL. REPL (Read Eval Print Loop) представляє можливість запуску виразів мовою JavaScript у командному рядку або терміналі.



Так, запустимо командний рядок (на Windows) або термінал (на OS X або Linux) та введемо команду node. Після введення цієї команди ми можемо виконувати різні вирази на JavaScript:

Або використовуємо якусь функцію JS:

Можна визначати свої функції і потім їх викликати, наприклад, піднесення числа до квадрату:

```
> console.log("Hello MNAU!!");
Hello MNAU!!
undefined
> function square(x){return x*x;}
undefined
> square(6)
36
```

Виконання файлу

Замість того, щоб вводити весь код безпосередньо в консоль, зручніше винести його у зовнішній файл. Наприклад, створимо на жорсткому диску новий каталог, скажімо, C:\node\, в який помістимо новий файл app.js з

```
C:\Users\DieSeele2>cd c:\node

c:\node>node app.js
Hello world
```

наступним кодом: `console.log("Hello world");`

Напишемо перший найпростіший додаток для NodeJS. Для створення програм можна використовувати практично всі стандартні конструкції мови JavaScript. Винятком є робота з DOM, оскільки програма буде запускатися на сервері, а не в браузері, тому DOM і такі об'єкти як window або document в даному випадку нам будуть недоступні.

Для цього спочатку створимо для програми каталог на жорсткому диску. Наприклад, я створив каталог C:\node\dbApp. У цьому каталозі

```
JS app.js
C: > node > dbApp > JS app.js > ...
1   const http = require("http");
2   http.createServer(function(request,response){
3
4       response.end("Hello NodeJS!");
5
6   }).listen(3000, "127.0.0.1",function(){
```

створимо файл app.js.

Підключення до MySQL

Для роботи з сервером MySQL у Node.js можна використовувати низку драйверів. Найпопулярніші з них mysql і mysql2. Здебільшого вони сумісні. В даному випадку ми будемо використовувати mysql2, оскільки, судячи з низки тестів, він надає більшу продуктивність.

```
C:\Users\DieSeele2>cd c:\node\dbApp

c:\node\dbApp>npm install --save mysql2

up to date, audited 13 packages in 2s

found 0 vulnerabilities
```

Отже, встановимо пакет mysql2:

Створення підключення

Для створення підключення застосовується метод createConnection(), який як параметр приймає налаштування підключення та повертає об'єкт, що

```
const mysql = require("mysql2");

const connection = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "1234"
});
```

представляє підключення.

Налаштування конфігурації, що передаються в метод, можуть містити ряд параметрів. Найбільш використовувані з них:

- host: хост, у якому запущено сервер mysql. За замовчуванням має значення "localhost";
- port: номер порту, на якому запущено сервер mysql. За замовчуванням має значення "3306";
- user: користувач MySQL, який використовується для підключення;
- password: пароль для користувача MySQL;
- database: ім'я бази даних, до якої йде підключення. Необов'язковий параметр, то якщо він не вказаний, то підключення йде до серверу;
- charset: кодування для підключення, наприклад, за замовчуванням використовується "UTF8_GENERAL_CI";
- timezone: часовий пояс сервера MySQL. За замовчуванням має значення "local".

Для встановлення підключення ми можемо використовувати метод

```
const mysql = require("mysql2");

const connection = mysql.createConnection({
  host: "localhost",
  user: "root",
  password: "1234"
});
 connection.connect(function(err){
    if (err) {
      return console.error("Помилка: " + err.message);
    }
    else{
      console.log("Підключення до серверу MySQL успішно встановлено");
    }
 });
```

connect() об'єкта connection:

Метод *connect()* приймає функцію, параметр якої містить помилку, що виникла при підключенні.

Можливі помилки під час підключення. Якщо під час підключення до сервера MySQL генерується помилка: *Client does not support authentication protocol requested by server; consider upgrading MySQL client.* У цьому випадку необхідно в MySQL Workbench виконати таку команду:

```
ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY 'password'
```

Замість 'password' має бути вказаний пароль від MySQL для користувача root.

```
Підключення до серверу MySQL успішно встановлено
```

Закриття підключення

Для закриття підключення застосовується метод end():

```
connection.end(function(err) {
 if (err) {
   return console.log("Помилка: " + err.message);
 }
 console.log("Підключення закрито");
});
```

При запуску програми та вдалому підключенні-закритті підключення

```
Підключення до серверу MySQL успішно встановлено
Підключення закрито
```

ми побачимо на консолі:

Метод *end()* гарантує, що перед закриттям підключення до бази даних будуть виконані всі запити, що залишилися, які не завершилися до моменту виклику методу.

Якщо ми не викличемо цей метод, то підключення залишатиметься активним, і програма Node.js продовжить свою роботу, поки сервер MySQL не закриє підключення.

Якщо ж нам треба негайно закрити підключення, не чекаючи виконання запитів, що залишилися, то в цьому випадку можна застосувати метод destroy(): `connection.destroy()`

Практична робота № 13

Введення в MySQL. Основні операції з даними

Створення бази даних

```
connection.query("CREATE DATABASE studentsDB",
 function(err, results) {
   if(err) console.log(err);
   else console.log("База даних створена");
});
```

Створимо базу даних на сервері MySQL через Node.js:

У цьому випадку за допомогою команди CREATE DATABASE створюється база даних studentsDB.

Створення таблиці

Тепер додамо до вище створеної бази даних з studentsDB таблицю. Для

```
c:\node\dbApp>node app.js
Підключення до серверу MySQL успішно встановлено
const mysql = require("mysql2");
const connection = mysql.createConnection({
  host: "localhost",
  user: "root",
  database: "studentsDB",
  password: "1234"
});
 connection.connect(function(err){
    if (err) {
      return console.error("Помилка: " + err.message);
    }
    else{
      console.log("Підключення до серверу MySQL успішно встановлено");
    }
 });
const sql = `create table if not exists students(
  id int NOT NULL UNIQUE auto_increment,
  name varchar(255) NOT NULL,
  sGroup varchar(255) NOT NULL,
  PRIMARY KEY (id)
  )`;
connection.query(sql, function(err, results) {
    if(err) console.log(err);
    else console.log("Таблиця створена");
});
```

створення таблиці застосовується команда CREATE TABLE, яка створюється

таблицю students із трьома стовпцями - id, name і sGroup.

```
Підключення до серверу MySQL успішно встановлено
Підключення закрито
Таблица создана
```

Додавання даних

Для додавання використовується SQL-команда INSERT. Додамо дані

```
const sql = `INSERT INTO students(name, sGroup) VALUES('Artem', 'KN2')`;

connection.query(sql, function(err, results) {
    if(err) console.log(err);
    console.log(results);
});
```

до раніше створеної таблиці students:

У цьому випадку таблицю додається один рядок, де стовпець name має значення "Artem", стовпець sGroup - значення “KN2”. За допомогою параметра results у функції зворотного виклику ми можемо отримати результати операції. Наприклад, у консольний виклик буде таким:

Повернено об'єкт, де можна виділити ряд властивостей. Перш за все, це affectedRows - кількість змінених операцією рядків (у даному випадку кількість доданих рядків) і insertId - ідентифікатор (значення поля id)

```
Підключення до серверу MySQL успішно встановлено
Підключення закрито
ResultSetHeader {
  fieldCount: 0,
  affectedRows: 1,
  insertId: 1,
  info: '',
  serverStatus: 2,
  warningStatus: 0,
  changedRows: 0
}
```

доданого запису. Відповідно, якби нам знадобилося отримати id доданого

```
connection.query(sql, function(err, results) {
  if(err) console.log(err);
  console.log(results.insertId);
});
```

рядка, то ми могли б написати так:

Додавання масиву значень

```
const students = [
  ["Roman", 'KN2'],
  ["Kostya", 'KN3'],
  ["Edmen", 'KN4']
];
const sql = `INSERT INTO students(name, sGroup) VALUES ?`;

connection.query(sql, [students], function(err, results) {
    if(err) console.log(err);
    console.log(results);
});
```

Додамо відразу кілька значень:

При додаванні безлічі об'єктів в SQL-запиті після VALUES вказується один знак питання. І при успішному додаванні властивість results.affectedRows виведе, що додано три рядки:

Однак у цьому випадку слід враховувати, що ми не зможемо отримати id всіх доданих рядків.

Отримання даних

Для отримання даних застосовується команда SELECT. Наприклад,

```
const sql = `SELECT * FROM students`;

connection.query(sql, function(err, results) {
    if(err) console.log(err);
    console.log(results);
});
```

```
  affectedRows: 3,
  insertId: 2,
  info: 'Records: 3  Duplicates: 0  Warnings: 0',
  serverStatus: 2,
  warningStatus: 0,
  changedRows: 0
}
```

отримаємо всі дані з таблиці students:

Об'єкт results функції зворотного виклику буде представляти масив отриманих з БД даних:

```
Підключення до серверу MySQL успішно встановлено
Підключення закрито
[
  { id: 1, name: 'Artem', sGroup: 'KN2' },
  { id: 2, name: 'Roman', sGroup: 'KN2' },
  { id: 3, name: 'Kostya', sGroup: 'KN3' },
  { id: 4, name: 'Edmen', sGroup: 'KN4' }
]
```

Відповідно після отримання ми зможемо працювати з цими даними як зі звичайним масивом об'єктів. Наприклад, виведемо лише ім'я для кожного користувача з бази даних:

```
const sql = "SELECT * FROM students";
connection.query(sql,  function(err, results) {
    if(err) console.log(err);
    const students = results;
    for(let i=0; i < students.length; i++){
      console.log(students[i].name);
    }
});
```

```
Підключення до серверу MySQL успішно встановлено
Підключення закрито
Artem
Roman
Kostya
Edmen
```

Консольний вивід:

Фільтрування даних

```
const sql = `SELECT * FROM students WHERE name=? AND sGroup=?`;
const filter = ["Edmen", "KN4"];
connection.query(sql, filter, function(err, results) {
    if(err) console.log(err);
    console.log(results);
  });
```

Виконаємо фільтрацію даних із застосуванням виразу WHERE:

Тут запит буде виглядати як SELECT * FROM students WHERE name="Edmen" AND sGroup=”KN4”, і в принципі ми могли б безпосередньо ввести дані в запит. Однак, щоб уникнути sql-ін'єкцій при передачі в запит даних ззовні рекомендується використовувати параметризацію.


```
Підключення до серверу MySQL успішно встановлено
Підключення закрито
[ { id: 4, name: 'Edmen', sGroup: 'KN4' } ]
```

Оновлення

```
const sql = `UPDATE students SET name=? WHERE sGroup=?`;
const data = ["Konstantyn", "KN3"];
connection.query(sql, data, function(err, results) {
    if(err) console.log(err);
    console.log(results);
});
```

Для оновлення даних застосовується sql-команда UPDATE:

Консольний вивід:

За допомогою властивостей affectedRows об’єкту results, ми можемо перевірити, скільки рядків було оновлено.

Видалення

```
Підключення до серверу MySQL успішно встановлено
const sql = "DELETE FROM students WHERE name=?";
const data = ["Roman"];
connection.query(sql, data, function(err, results) {
    if(err) console.log(err);
    console.log(results);
});
  changedRows: 1
}
```

Для видалення застосовується sql-команда DELETE:

Консольний вивід:

Для виконання запитів об'єкта підключення застосовується метод query(). Найбільш проста його форма:

Де sqlString - виконувана SQL-команда, а callback - функція зворотного

```
Підключення до серверу MySQL успішно встановлено
Підключення закрито
ResultSetHeader {
  fieldCount: 0,
  affectedRows: 1,
  insertId: 0,
  info: '',
  serverStatus: 34,
  warningStatus: 0,
  changedRows: 0
}
```

виклику, через параметри якої ми можемо отримати результати виконання sql-команди або помилку.

```
connection.query("SELECT * FROM students",
function(err, results, fields) {
  console.log(err);
  console.log(results);
  console.log(fields);
});
```

```
Підключення до серверу MySQL успішно встановлено
Підключення закрито
null
[
  { id: 1, name: 'Artem', sGroup: 'KN2' },
  { id: 3, name: 'Konstantyn', sGroup: 'KN3' },
  { id: 4, name: 'Edmen', sGroup: 'KN4' }
]
[
  `id` INT NOT NULL PRIMARY KEY UNIQUE_KEY AUTO_INCREMENT,
  `name` VARCHAR(255) NOT NULL,
  `sGroup` VARCHAR(255) NOT NULL
]
```

Наприклад, отримаємо всі дані з таблиці:

У цьому випадку виконується команда SELECT, яка витягує всі дані з таблиці "students". Функція зворотного виклику приймає три параметри. Перший параметр передає помилку, якщо вона виникла під час виконання запиту. Другий параметр - results власне представляє як масиву ті дані, які отримала команда SELECT. І третій параметр fields зберігає метадані поля таблиці та додаткову службову інформацію.

Варто зазначити, що при виконанні запитів неявно встановлюється підключення, тому перед виконанням запиту нам у принципі необов'язково викликати в об'єкт підключення методу connect().

Також mysql2 визначено метод execute(), який працює аналогічним

```
connection.execute("SELECT * FROM students",
function(err, results, fields) {
  console.log(err);
  console.log(results);
  console.log(fields);
});
```

чином:

```
Підключення до серверу MySQL успішно встановлено
Підключення закрито
null
[
  { id: 1, name: 'Artem', sGroup: 'KN2' },
  { id: 3, name: 'Konstantyn', sGroup: 'KN3' },
  { id: 4, name: 'Edmen', sGroup: 'KN4' }
]
[
  `id` INT NOT NULL PRIMARY KEY UNIQUE_KEY AUTO_INCREMENT,
  `name` VARCHAR(255) NOT NULL,
  `sGroup` VARCHAR(255) NOT NULL
]
```

Параметризація запитів

Якщо в запит треба вводити дані, які приходять ззовні, то для уникнення ін'єкцій sql рекомендується використовувати параметризацію.

При параметризації замість конкретних даних у тексті запиту ставляться плейсхолдери - знаки питання, замість яких під час виконання

```
const user = ["Alex","KN5"];
const sql = "INSERT INTO students(name, sGroup) VALUES(?, ?)";

connection.query(sql, user, function(err, results) {
    if(err) console.log(err);
```

запиту вставлятимуться власне дані. Наприклад, додавання даних:

В даному випадку дані визначені у вигляді масиву students, яка як параметр передається в метод connection.query(). При виконанні запиту ці дані по порядку ставляться у запит місце знаки запитання. Тобто фактично запит буде мати такий вигляд: INSERT INTO students(name, sGroup) VALUES("Alex", “KN5”).

Індивідуальне завдання

Розробити власну БД, таблицю, додати, оновити та видалити дані.

```
Підключення до серверу MySQL успішно встановлено
Підключення закрито
null
[
  { id: 1, name: 'Artem', sGroup: 'KN2' },
  { id: 3, name: 'Konstantyn', sGroup: 'KN3' },
  { id: 4, name: 'Edmen', sGroup: 'KN4' },
  { id: 5, name: 'Alex', sGroup: 'KN5' }
]
[
  `id` INT NOT NULL PRIMARY KEY UNIQUE_KEY AUTO_INCREMENT,
  `name` VARCHAR(255) NOT NULL,
  `sGroup` VARCHAR(255) NOT NULL
]
```

Практична робота № 14

Введення в MySQL. Створення форм для роботи з базою даних

Створимо найпростіший веб-застосунок, який взаємодіє з MySQL. Для роботи візьмемо створену раніше базу даних, яка зберігає студентів і має три стовпці: id, name та sGroup. Під час відправлення якихось складних даних зазвичай використовуються форми.

```
c:\node\dbApp>npm install --save hbs express
```

Встановимо в існуючий проект пакети *express* та *handlebars*.

Для графічного відображення ми будемо використовувати уявлення Handlebars. Тому визначимо у проекті нову папку **views**. До неї додамо новий файл **index.hbs**:

C: > node > dbApp > views > index.hbs > html > html

```
<!DOCTYPE html>
<html>
<head>
    <title>Список студентів</title>
    <meta charset="utf-8" />
</head>
<body>
    <h1>Список студентів</h1>
    <p><a href="/insert">Додати студента</a></p>
    <table>
        <tr><th>ПІБ студента</th><th>Номер групи</th><th></th></tr>
        {{#each students}}
            <tr>
                <td>{{this.name}}</td>
                <td>{{this.sGroup}}</td>
                <td>
                    <a href="/update/{{this.id}}">Edit</a>|
                    <form action="delete/{{this.id}}" method="POST" style="display:inline;">
                        <input type="submit" value="Delete" />
                    </form>
                </td>
            </tr>
        {{/each}}
    </table>
</body>
<html>
```

Дане уявлення виводить список об'єктів у вигляді таблиці. Поряд із кожним студентом визначено посилання на редагування його даних та кнопка для видалення. Над таблицею визначено посилання на форму для додавання нових студентів.

Також додамо до папки views новий файл **insert.hbs** з формою для

створення нового студента.

```
C: > node > dbApp > views > insert.hbs > html > html
<!DOCTYPE html>
<html>
<head>
    <title>Додати студента</title>
    <meta charset="utf-8" />
</head>
<body>
    <h1>Додати студента</h1>
    <form method="POST">
        <label>ПІБ студента</label><br>
        <input name="name" class="field"/><br><br>
        <label>Номер групи</label><br>
        <input name="sGroup" class="field" /><br><br>
        <input type="submit" value="Додати" />
    </form>
    <a href="/">До списку студентів</a>
</body>
</html>
```

Також додамо в папку views новий файл **update.hbs** з формою для

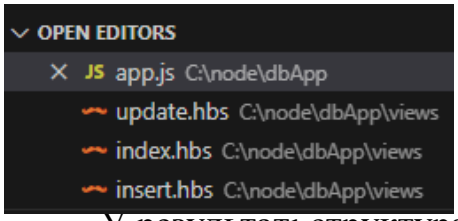
```
C: > node > dbApp > views > update.hbs > html > html
<!DOCTYPE html>
<html>
<head>
    <title>Редагування інформації про студентів</title>
    <meta charset="utf-8" />
</head>
<body>
    <h1>Редагування інформації про студентів</h1>
    <form action="/update" method="POST">
        <input type="hidden" name="id" value="{{students.id}}" />
        <label>ПІБ студента</label><br>
        <input name="name" value="{{students.name}}"  class="field" /><br><br>
        <label>Номер групи</label><br>
        <input name="sGroup"  value="{{students.sGroup}}"  class="field" /><br><br>
        <input type="submit" value="Оновити" />
    </form>
    <a href="/">До списку студентів</a>
</body>
<html>
```

редагування студента:

Перевизначимо в кореневій папці проекту файл app.js, який представлятиме головний файл програми:

```
const mysql = require("mysql2");
const express = require("express");
const app = express();
const urlencodedParser = express.urlencoded({extended: false});
```

```
const pool = mysql.createPool({
  connectionLimit: 5,
  host: "localhost",
  user: "root",
  database: "studentsDB",
  password: "1234"
});
app.set("view engine", "hbs");
app.get("/", function(req, res){
    pool.query("SELECT * FROM students", function(err, data) {
      if(err) return console.log(err);
      res.render("index.hbs", {
          students: data
      });
    });
});
app.get("/insert", function(req, res){
    res.render("insert.hbs");
});
app.post("/insert", urlencodedParser, function (req, res) {
    if(!req.body) return res.sendStatus(400);
    const name = req.body.name;
    const sGroup = req.body.sGroup;
    pool.query("INSERT INTO students (name, sGroup) VALUES (?,?)", [name, sGroup], function(err, data) {
      if(err) return console.log(err);
      res.redirect("/");
    });
});
app.get("/update/:id", function(req, res){
  const id = req.params.id;
  pool.query("SELECT * FROM students WHERE id=?", [id], function(err, data) {
    if(err) return console.log(err);
     res.render("update.hbs", {
        students: data[0]
    });
  });
});
app.post("/update", urlencodedParser, function (req, res) {
  if(!req.body) return res.sendStatus(400);
  const name = req.body.name;
  const sGroup = req.body.sGroup;
  const id = req.body.id;

  pool.query("UPDATE students SET name=?, sGroup=? WHERE id=?", [name, sGroup, id], function(err, data) {
    if(err) return console.log(err);
    res.redirect("/");
  });
});
app.post("/delete/:id", function(req, res){
  const id = req.params.id;
  pool.query("DELETE FROM students WHERE id=?", [id], function(err, data) {
    if(err) return console.log(err);
    res.redirect("/");
  });
});
app.listen(3000, function(){
  console.log("Сервер очікує підключення...");
});
```

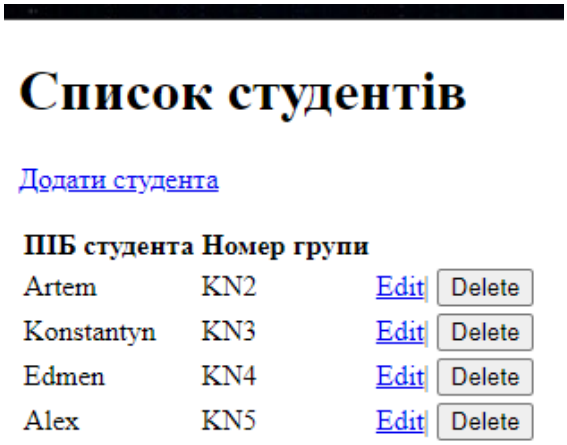


У результаті структура готового проекту виглядатиме таким чином:

При зверненні до кореня програми спрацьовує метод `app.get("/", function(req, res)`

```
app.get("/", function(req, res){
    pool.query("SELECT * FROM students", function(err, data) {
      if(err) return console.log(err);
      res.render("index.hbs", {
          students: data
      });
    });
});
```

Цей метод отримує дані з бази даних і передає їх уявленню **index.hbs**. І якщо у базі даних є вже якісь дані, то при запуску програми ми побачимо їх



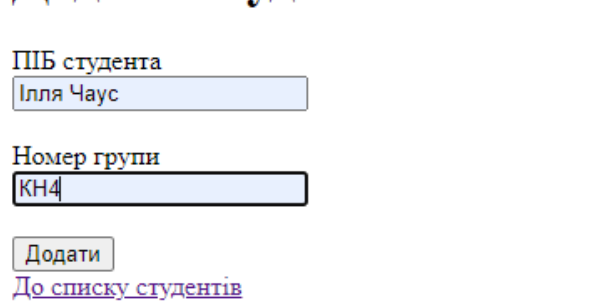
на сторінці:

При натисканні на посилання додавання студента - серверу відправлятиметься get-запит, який оброблятиметься наступним методом:

Метод повертає користувачеві форму для додавання студента:

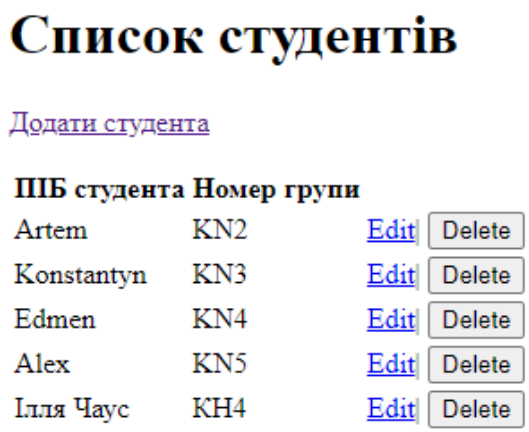
Після заповнення форми та натискання на кнопку дані у запиті POST відправляються методом:

```
app.get("/insert", function(req, res){
```



```
app.post("/insert", urlencodedParser, function (req, res) {
    if(!req.body) return res.sendStatus(400);
    const name = req.body.name;
    const sGroup = req.body.sGroup;
    pool.query("INSERT INTO students (name, sGroup) VALUES (?,?)", [name, sGroup], function(err, data) {
      if(err) return console.log(err);
      res.redirect("/");
    });
});
```

Метод отримує відправлені дані та за допомогою SQL-команди

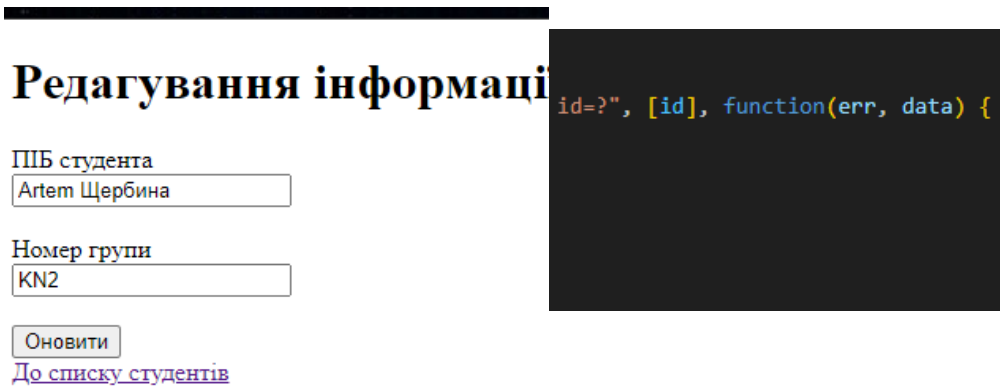


INSERT відправляє їх до бази даних.

При натисканні на посилання «редагування студента» у списку об'єктів наступному методу у GET-запиті передається id об'єкта:

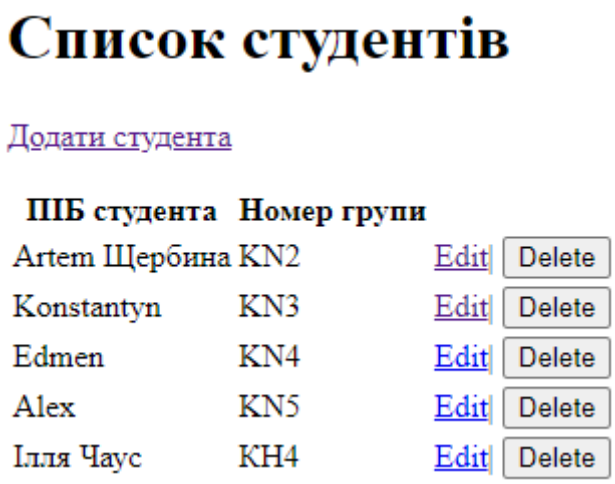
Метод отримує id і по ньому витягує з бази даних потрібний об'єкт та передає його на форму уявлення update.hbs:

Після редагування та натискання на кнопку дані відправляються у POST-запиті наступним методом:



```
app.post("/update", urlencodedParser, function (req, res) {
  if(!req.body) return res.sendStatus(400);
  const name = req.body.name;
  const sGroup = req.body.sGroup;
  const id = req.body.id;

  pool.query("UPDATE students SET name=?, sGroup=? WHERE id=?", [name, sGroup, id], function(err, data) {
    if(err) return console.log(err);
    res.redirect("/");
  });
});
```

Метод отримує дані та за допомогою команди UPDATE відправляє їх у

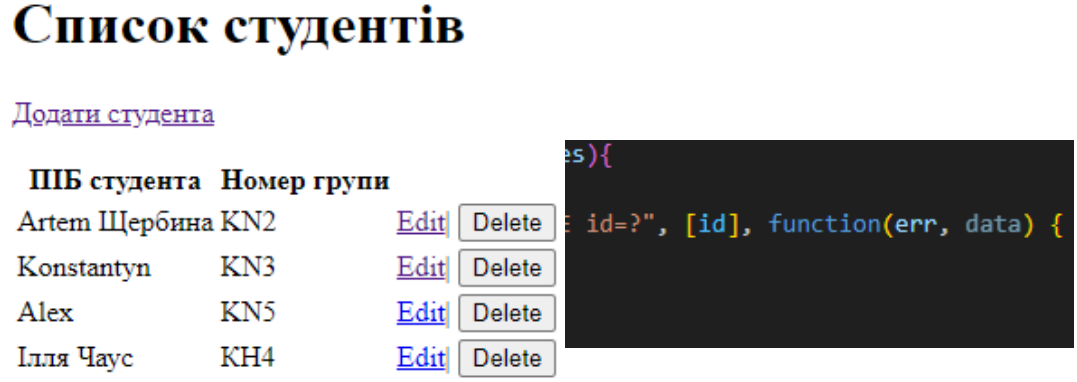


базу даних.

При натисканні на кнопку видалення у списку об'єктів спрацьовує метод:

Метод отримує id об'єкта, що видаляється, і видаляє його з БД за допомогою команди DELETE.

Таким чином, ми можемо зв'язати в Node.js Express і взаємодію з базою даних MySQL.



Написання клієнтського додатку на мові PHP

Для роботи із БД будемо використовувати вбудований клас SQLite3. При створенні екземпляра класу відувається автоматичне підключення БД, отже в конструктор необхідно передати параметр – ім’я файлу БД.

```
$myDB = new SQLite3('myDatabase.db');
```

Крім цього, будемо використовувати ще 2 методи класу SQLite3.

Перший - метод exec(), наприклад

```
$myDB->exec('create table t1(f1 int);');
```

якому у якості параметру передається SQL запит для виконання. Метод повертає істину у разі успішного виконання, і хибність у випадку виникнення помилки.

Другий – метод query(), він є дещо схожим на попередній, також приймає параметр - SQL запит для виконання але повертає об’єкт SQLite3Result, або false у випадку виникнення помилки.

Клас SQLite3Result представляє доступ до результуючого набору розширення SQLite3. В цьому класі використаємо метод fetchArray, що повертає рядок із результуючого набору. Метод викликається із параметром SQLITE3_ASSOC, що повертає асоціативний масив, у якому індекс відповідає імені стовпчика у результуючому наборі.

Переходимо до реалізації програмного коду, що відповідатиме за відображення, додавання, зміну та видалення даних із таблиці «Типи страв» бази даних «Їжа». В папці з сервером у директорії domains/localhost/ папку «sqlite». В цій папці створимо файл sqlite.php, в якому створимо клас Food, що буде відповідати за операції із БД. Підключатися до БД будемо під час створення екземпляра, тобто у конструкторі. Крім того знадобиться ще 5 методів для отримання списку типів їжі, читання одного з типів їжі, додавання, зміни та видалення типи їжі відповідно.

Нижче наведено клас Food

sqlite.php

```php
<?
class Food {
    private $db;

    function __construct() {
        $this->db = new SQLite3('c:\\temp\\sqlite\\foods.db');
    }

    function getFoodTypes() {
        $res = array();
        $sql = 'select id, name from food_types';
        $ret = $this->db->query($sql);
        while($row = $ret->fetchArray(SQLITE3_ASSOC) ) {
            $res[] = $row;
        }
        return $res;
    }

    function addFoodType($name) {
        $sql = "insert into food_types(id, name) select max(id)+1, '" . $name ."' from food_types";
        $this->db->exec($sql);
    }

    function removeFoodType($id) {
        $sql = "delete from food_types where id = " . $id;
        $this->db->exec($sql);
    }

    function editFoodType($id, $name) {
        $sql = "update food_types set name = '" . $name . "' where id = " . $id;
        $this->db->exec($sql);
    }

    function getFoodTypeName($id) {
        $sql = 'select name from food_types where id = ' . $id;
        $ret = $this->db->query($sql);
        if($row = $ret->fetchArray(SQLITE3_ASSOC) ) {
            return $row['name'];
        }
        return '';
    }
}
```

Далі створимо index.php, що буде відповідати за відображення списку типів їжі у таблиці. Крім того, тут реалізуємо форму додавання нового типу їжі. На початку підключаємо sqlite.php та далі у коді створюємо екземпляр Food та використовуємо метод getFoodTypes() для отримання вмісту таблички «Типи їжі».

index.php

```php
<?php
    require 'sqlite.php';
?>
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>food types</title>
</head>
<body>
    <form name='add-food-type' action='add.php' method='post'>
        Новий тип їжі <input type="text" name="food-type-name">
        <input type="submit" value="Додати">
    </form>
    <table class="food-types-list">
        <thead>
            <tr>
                <th class='id'>id</th><th class='name'>name</th><th></th>
            </tr>
        </thead>
        <tbody>
            <?php
            $food = new Food();
            $foodTypes = $food->getFoodTypes();
                ?>
                <tr>
                    <td class='id'><?php echo $foodType['id']; ?></td>
                    <td class='name'><?php echo $foodType['name']; ?></td>
                    <td>
                        <a href="edit.php?id=<?php echo $foodType['id']; ?>">Змінити</a> | <a href="
                        delete.php?id=<?php echo $foodType['id']; ?>">Видалити</a>
                    </td>
                </tr>
                <?php
            }
            ?>
        </tbody>
    </table>

</body>
</html>
```

В головному файлі ми посилаємось на add.php, що має додавати новий рядок до таблиці, delete.php для видалення рядку, та edit.php, в якому реалізована форма редагування та безпосередньо сама зміна даних у БД. Наведемо програмний код їх реалізації.

add.php

```php
<?php
require 'sqlite.php';
if ($_POST['food-type-name']) {
    $food = new Food();
    $food->addFoodType($_POST['food-type-name']);
}
header("Location: index.php");
```

Add.php

Delete.php

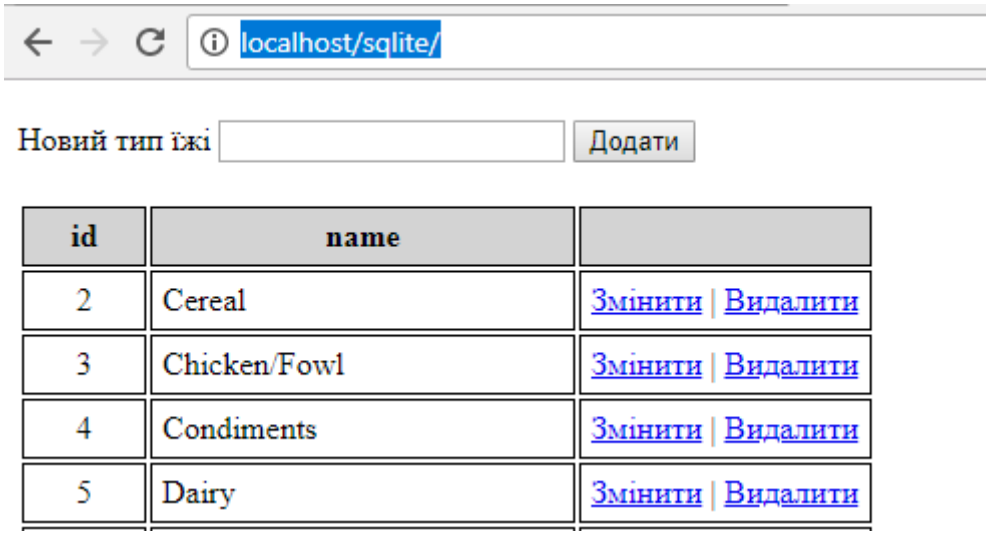
Edit.php

```php
<?php
require 'sqlite.php';
$food = new Food();
if ($_GET['id'] && $_POST['food-type-name']) {
    $food->editFoodType($_GET['id'], $_POST['food-type-name']);
    header("Location: index.php");
}
?>
<!DOCTYPE html>
<html>
<head>
    <title>food types</title>
</head>
<body>
    <form name='food-types-edit' method='post' action='edit.php?id=<?php echo $_GET['id'];?>'>
        Тип їжі <input type="text" name="food-type-name" value="<?php echo $food->getFoodTypeName($_GET['id']); ?>">
        <input type="submit" value="Змінити">
    </form>
</body>
</html>
```

Для форматування виводу створимо style.css та підключимо його в index.php

```css
table.food-types-list td, table.food-types-list th {
    border: solid 1px black;
    padding: 5px;
}
table.food-types-list th {
    background-color: lightgray;
}
table.food-types-list th.id, table.food-types-list td.id {
    width: 50px;
    text-align: center;
}
table.food-types-list th.name, table.food-types-list td.name {
    width: 200px;
}
form[name='add-food-type'] {
    margin-top: 20px;
    margin-bottom: 20px;
}
```

Перевіримо роботу додатку, набравши в адресному рядку браузеру «http://localhost/sqlite/». Спробуємо додати, змінити та видалити дані.



Індивідуальне завдання

На основі розробленої бази даних, створити форму та змінити таблицю: додати, оновити та видалити дані. Продемонструвати результати.

Перелік питань для підсумкового контролю знань

1. Поняття інформації та інформаційної системи. Класифікація інформаційних систем.

2. Історія розвитку баз даних.

3. Логічна та фізична структура баз даних.

4. Архітектура інформаційної системи.

5. Банки даних. Бази даних та системи управління базами даних.

6. Архітектура СУБД.

7. Функції СУБД.

8. Мовні засоби СУБД: мова структурованих запитів та її підмови.

9. Ієрархічна та мережна моделі даних. Переваги та недоліки.

10. Проблеми маніпулювання даними та обмеження цілісності даних.

11. Реляційна модель та її характеристики.

12. Структура реляційних даних. Домени. Схема баз даних. Таблиці баз даних. Первинні та зовнішні ключі. Індекси.

13. Методи та способи доступу до даних.

14. Цілісність реляційних даних.

15. Зв’язки між таблицями.

16. Поняття транзакції. Механізм транзакцій.

17. Нормалізація реляційних баз даних. Перша нормальна форма, друга нормальна форма, третя нормальна форма.

18. Основні поняття мови SQL.

19. Запити на читання даних. Склеювання таблиць. Умови відбору рядків таблиць.

20. Агрегатні функції.

21. Запити з групуванням.

22. Складні запити. Запити на оновлення даних. Запити на створення та оновлення схеми баз даних, таблиць та представлень.

23. Поняття індексації даних. Способи організації індексів.

24. Внутрішня мова програмування СУБД.

25. Збережені процедури сервера та тригери. Призначення та переваги.
26. Безпека баз даних. Управління користувачами. Привілеї.
27. Проєкт реляційної бази даних.
28. Етапи проєктування.
29. Системний аналіз предметної області.
30. Даталогічне проєктування.
31. Логічне проєктування БД.
32. Інфологічна модель БД.
33. Правила перетворення ER-моделі на реляційну.
34. Алгоритм приведення семантичної моделі до 5-ї нормальної форми.
35. Методи захисту баз даних.
36. Безпека даних та її функції.
37. Адміністрування баз даних.
38. «Локальна» архітектура та архітектура «файл-сервер» баз даних.
39. Експорт та імпорт таблиць баз даних.
40. Архітектура клієнт-серверних СУБД.
41. Концепція відкритих систем.
42. Відкритий зв’язок з базою даних.
43. Транзакції.
44. Проблеми паралелізму.
45. Розробка баз даних в середовищі СУБД MS SQL Server.
46. Створення таблиць.
47. Створення тригерів.
48. Управління доступом.
49. Шифрування даних.
50. Засоби підтримки безпеки в SQL.
51. Пост реляційні, об’єктно-орієнтовані та XML бази даних.
52. Технології інтелектуальної обробки даних.
53. Методи та засоби багатовимірного статистичного аналізу даних.
54. Система управління базами даних MySQL.

55. Основні команди СУБД MySQL.

56. Функції, типи даних, робота з таблицями.

57. Створення ключів та індексів. Зовнішні ключі. Зв’язування таблиць. Захист даних в MySQL.

58. Адміністрування. Привілеї.

59. Організація транзакцій.

60. Проектування баз даних за допомогою інструментальних засобів програми SQLite.

Список рекомендованих та використаних джерел

Основна

1. Берко А. Ю., Верес О. М., Пасічник В. В. Системи баз даних та знань. Кн. 1. Організація баз даних та знань : навч. посібник. Львів : Магнолія 2006, 2024. 456 с.
2. Берко А.Ю., Верес О.М., Пасічник В.В. Моделі баз даних та знань : підручник. Львів : ПП "Магнолія 2006", 2025. 463 с
3. Верес О. М., Берко А. Ю., Пасічник В. В. Технології баз даних та знань : підручник. Львів : Магнолія 2006, 2024. 636 с.
4. Гайна Г. А. Основи проектування баз даних : навчальний посібник. Київ : Кондор, 2024. 208 с.
5. Демиденко М. А. Введення в сучасні бази даних : навч. посіб. / НТУ «Дніпровська політехніка». Дніпро : Дніпровська політехніка, 2020. 38 с. URL: https://ir.nmu.org.ua/server/api/core/bitstreams/7d1d93bf-305f-4d4c-ba18-aa4eb2735c87/content
6. Доценко С. І. Організація та системи керування базами даних : навч. посібник. Харків : УкрДУЗТ, 2023. 117 с. URL: https://surli.cc/wgzyzk
7. Костенко О. Б. Бази даних : конспект лекцій. Харків : ХНУМГ ім. О. М. Бекетова, 2021, 92 с.
8. Мікула М. П., Коцюк Ю. А., Мікула О. М. Організація баз даних та знань : навчальний посібник. Острог : Острозька академія, 2021. 194 с.
9. Організація баз даних та знань : конспект лекцій / уклад. О. Б. Костенко, І. О. Гавриленко. Харків : ХНУМГ ім. О. М. Бекетова а, 2021. 92 с. URL: https://eprints.kname.edu.ua/60505/1/2020%20%D0%BF%D0%B5%D1%87.%20134_%D0%9B.pdf
10. Павловський В. І., Петрашенко А. В. Бази даних та засоби управління : підручник. Київ : КПІ ім. Ігоря Сікорського, 2024. 326 с. URL: https://ela.kpi.ua/server/api/core/bitstreams/9f26675d-c42f-485e-aab5-13e6e014b560/content
11. Проектування та використання баз даних. Комп'ютерний практикум : навчальний посібник / уклад. І. В. Сегеда. Київ : КПІ ім. Ігоря Сікорського, 2021. 49 с. URL: https://ela.kpi.ua/server/api/core/bitstreams/a20f3199-33a5-47d1-ba26-0a2ef8b73790/content
12. Рзаєва С. Л., Харченко О. А. Бази даних : посібник. Київ : Київський національний торговельно-економічний університет, 2021. 228 с.
13. Технології баз даних : навчально-практичний посібник / уклад. А. А. Гаврилова, С. С. Погасій, Р. В. Корольов, В. С. Хвостенко, Т. С. Мілевська ; за ред. С. П. Євсеєва. Харків : НТУ «ХПІ», 2025. 222 с.

Додаткова

1. Анісімов А. В., Кулябко П. П. Інформаційні системи та бази даних : навчальний посібник для студентів факультету комп'ютерних наук та кібернетики. Київ : Київський національний університет ім. Тараса Шевченка, 2017.

2. Балик Н.Р., Мандзюк В.І. Бази даних MySQL: Навчальний посібник. Тернопіль. 2010. 160 с.

3. Берко А.Ю., Верес О.М., Пасічник В.В. Системи баз даних та знань. Книга 1. Бази даних. «Комп'ютинг», 2012, 460 с.

4. Берко А.Ю., Верес О.М., Пасічник В.В. Системи баз даних та знань. Книга 2. Бази даних. «Комп'ютинг», 2014, 590с.

5. Власюк А. Г. Основи використання SQL у серверних системах : підручник. Київ: Компрінт, 2017, 125 с.

6. Гайдаржи В. І., Ізварін І. В. Бази даних в інформаційних системах : підруч. Київ, 2018, 418 с.

7. Костріков С. В. Інформаційні технології в БД. Навчально-методичний Посібник. Харків, 2015, 56 с.

8. Лосєв М. Ю., Федько В. В. Бази даних : навчально-практичний посібник для самостійної роботи студентів Харків : ХНЕУ ім. С. Кузнеця, 2018. 233 с.

9. Мікула М. П., Коцюк Ю. А., Мікула О. М. Організація баз даних та знань: навчальний посібник для студентів спеціальності «Комп'ютерні науки». Острог: Острозька академія, 2021. 194 с.

10. Павлиш В. А. Основи інформаційних технологій і систем : підручник. Львів : Львівська політехніка, 2018, 619 с.

11. Павловський В. І., Петрашенко А. В. Бази даних та засоби управління : підручник / ; КПІ ім. Ігоря Сікорського. Київ : КПІ ім. Ігоря Сікорського, 2024. 326 с.

12. Пасічник В.В., Резніченко В. А. Бази даних. Київ. Видавнича група «BHV», 2011, 384 с.

13. Сидоренко В. В., Константинова Л. В., Смірнов С. А. Організація баз даних : навчальний посібник. Кропивницький : ЦНТУ, 2018. 274 с.

14. Цеслів О. В., Коломієць А. С. Технологія проектування та адміністрування баз даних і сховищ даних. Навч. посібник. Київ, 2017, 281 с.

15. Шаховська Н. Б., Камінський Р. М., Вовк О. Б. Системи штучного інтелекту. Навч. посібник. Львів, 2018, 391 с.

Навчальне видання

БАЗИ ДАНИХ

Методичні рекомендації

Укладачі: **Тищенко** Світлана Іванівна
Пархоменко Олександр Юрійович
Жебко Олександр Олегович
Петренко Вадим
Коломієць Андрій Миколайович

Формат 60х84 1/16. Ум. друк. арк. 2.94.
Наклад 50 прим. Зам. № _____

Надруковано у видавничому відділі
Миколаївського національного аграрного університету
54020, м. Миколаїв, вул. Георгія Гонгадзе, 9

Свідоцтво суб'єкта видавничої справи ДК № 4490 від 20.02.2013