**Практична робота №4. Концептуальне проектування баз даних. Створення таблиць в SQLite.**

**Мета:** Ознайомлення із методологією концептуального проектування

**Завдання до практичної роботи**

1. Повторити лекцію №5. Ознайомитися з теоретичною частиною.
2. В попередній ПР та ЛР були створені сутності Викладач та Студент, Група, зв’язаної з сутністю Студент, Предмети, зв’язаної з сутністю Викладач. В рамках проведення концептуального проектування визначити / уточнити питання:

Визначити типи користувачів та представлення про предметну область кожного з типів користувачів.

Визначити можливі додаткові типи сутностей.

Визначити типів зв'язків.

Визначити атрибути і зв'язування їх з типами сутностей і зв'язків.

Визначити домени атрибутів.

Визначити атрибути що є потенційними і первинними ключами.

Створити / модифікувати діаграми "сутність-зв'язок" для опису схеми БД.

1. Оформити питання у вигляді короткого опису. Скріншот схеми включити до звіту.
2. Розглянути SQLiteта при виборі в якості інструменту відпрацювати техніку створення таблиць за розробленою схемою БД. Сформулювати висновки щодо використання СКБД як інструменту в подальшому вивченні предмету. Якщо це інша СКБД – надати коротке обґрунтування.
3. Результати надсилати на електронну адресу викладача [t.i.lumpova@gmail.com](mailto:t.i.lumpova@gmail.com)

Файл повинен мати назву в такому форматі:

**DB<Номер групи><Номер лекції / практичної / лабораторної [літера позначення типу роботи L – лекція, P – практична, R – лабораторна]<Прізвище англійською>**. Наприклад, **DB3101R**buts.doc.

Не копіюйте фрагментів з різних інформаційних джерел, подумайте і викладіть свою точку зору. При наявності робіт -"близнюків" відповідь буде зараховуватися першому за часом надсилання.

**Строк виконання цієї роботи ІПЗ-31 - 05.10.2022**

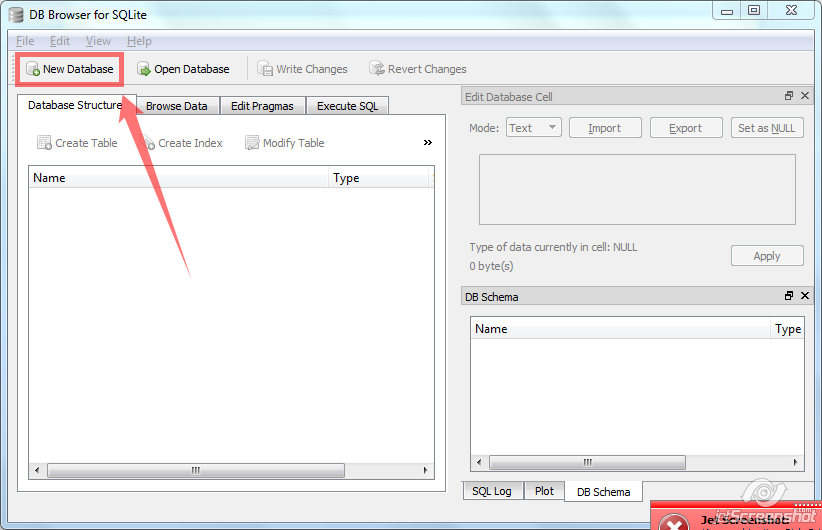
**КОНТРОЛЬНІ ПИТАННЯ**

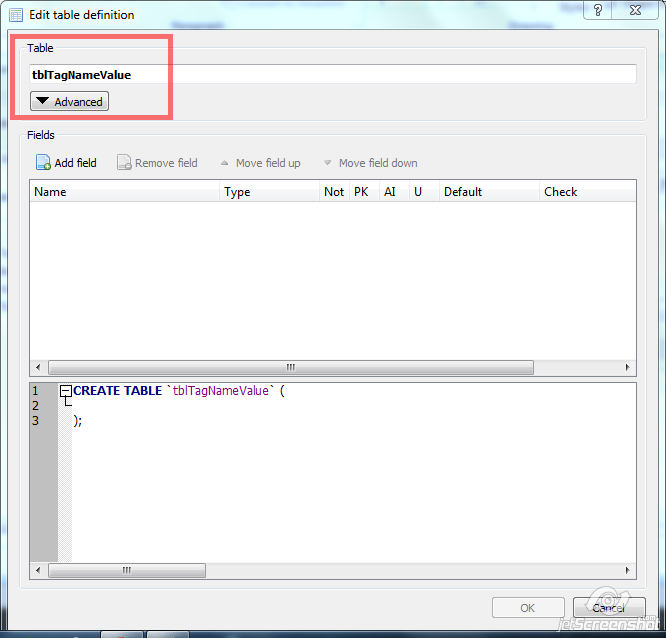
**Теоретична частина**

Особливістю SQLite є те, що вона не використовує парадигму клієнт-сервер, тобто рушій SQLite не є окремим процесом, з яким взаємодіє [застосунок](https://www.wiki.uk-ua.nina.az/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA.html), а надає бібліотеку, з якою програма компілюється і рушій стає складовою частиною програми. Таким чином, як протокол обміну використовуються виклики функцій (API) бібліотеки SQLite. Такий підхід зменшує накладні витрати, час відгуку і спрощує програму. SQLite зберігає всю базу даних (включаючи визначення, [таблиці](https://www.wiki.uk-ua.nina.az/%D0%A2%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D1%8F_(%D0%B1%D0%B0%D0%B7%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85).html), [індекси](https://www.wiki.uk-ua.nina.az/%D0%86%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_(%D0%B1%D0%B0%D0%B7%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85).html) і дані) в єдиному стандартному файлі на тому комп'ютері, на якому виконується застосунок. Простота реалізації досягається за рахунок того, що перед початком виконання транзакції весь файл, що зберігає базу даних, блокується; ACID-функції (про що нижче) досягаються зокрема за рахунок створення файлу-журналу.

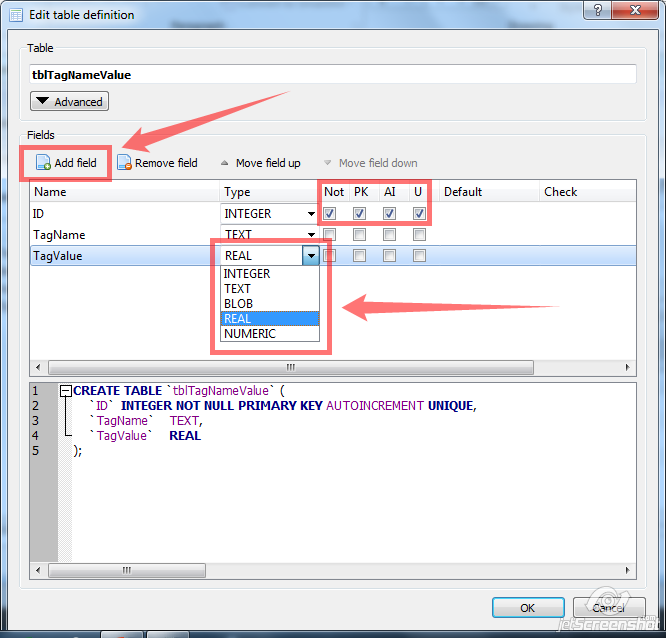
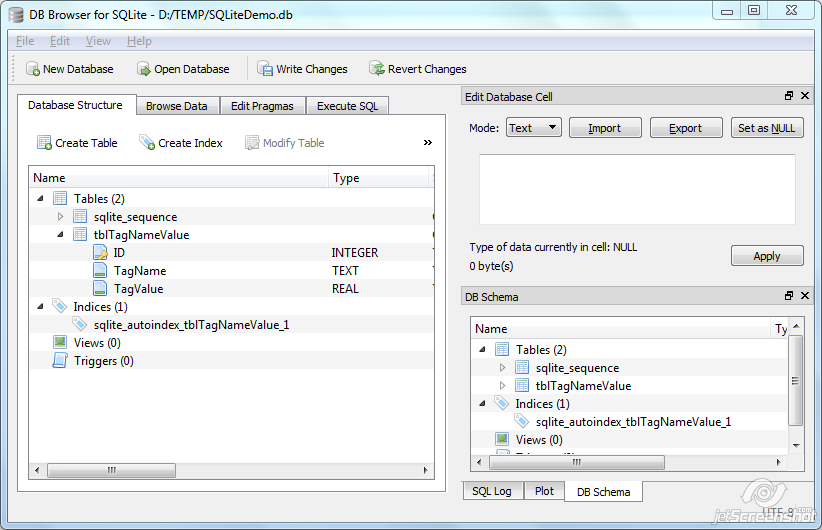
Кілька процесів або потоків можуть одночасно без жодних проблем читати дані з однієї бази. Запис в базу можна здійснити тільки в тому випадку, коли жодних інших запитів у цей час не обслуговується; інакше спроба запису закінчується невдачею, і в програму повертається код помилки. Іншим варіантом розвитку подій є автоматичне повторення спроб запису протягом заданого інтервалу часу.

**Створення нової БД.**





Додавання нової таблиці

  **Додаткові відомості про SQlite**

У комплекті постачання йде також функціональна клієнтська частина у вигляді виконуваного файлу sqlite3, за допомогою якого демонструється реалізація функцій основної бібліотеки. Клієнтська частина працює з командного рядка, і дозволяє звертатися до файлу БД на основі типових функцій ОС.

Завдяки архітектурі рушія можливо використовувати SQlite як на вбудовуваних (embedded) системах, так і на виділених машинах з гігабайтними масивами даних.

* [транзакції](https://www.wiki.uk-ua.nina.az/%D0%A2%D1%80%D0%B0%D0%BD%D0%B7%D0%B0%D0%BA%D1%86%D1%96%D1%8F_(%D0%B1%D0%B0%D0%B7%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85).html) [атомарні](https://www.wiki.uk-ua.nina.az/%D0%90%D1%82%D0%BE%D0%BC%D0%B0%D1%80%D0%BD%D0%B0_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%8F.html), послідовні, ізольовані, і міцні (ACID) навіть після збоїв системи і збоїв живлення
* Встановлення без конфігурації — не потребує ані установки, ані адміністрування
* Реалізує значну частину стандарту SQL92
* База даних зберігається в одному крос-платформовому файлі на диску
* Підтримка терабайтних розмірів баз даних і гігабайтного розміру рядків і [BLOBів](https://www.wiki.uk-ua.nina.az/BLOB.html)
* Малий розмір коду: менше ніж 350KB повністю налаштований, і менш 200KB з опущеними додатковими функціями
* Швидший за популярні рушії [клієнт-серверних](https://www.wiki.uk-ua.nina.az/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80.html) [баз даних](https://www.wiki.uk-ua.nina.az/%D0%A1%D0%9A%D0%91%D0%94.html) для найпоширеніших операцій
* Простий, легкий у використанні API
* Написана в ANSI C, включена прив'язка до TCL[[1]](#footnote-1); доступні також прив'язки для десятків інших мов
* Добре прокоментований сирцевий код зі 100 % тестовий покриттям гілок
* Доступний як єдиний файл сирцевого коду на ANSI C, який можна легко вставити в інший проект
* Автономність: немає зовнішніх залежностей
* Крос-платформовість: з коробки підтримується [Unix](https://www.wiki.uk-ua.nina.az/Unix.html) ([Linux](https://www.wiki.uk-ua.nina.az/Linux.html) і [Mac OS X](https://www.wiki.uk-ua.nina.az/Mac_OS_X.html)), [OS/2](https://www.wiki.uk-ua.nina.az/OS/2.html), [Windows](https://www.wiki.uk-ua.nina.az/Windows.html) ([Win32](https://www.wiki.uk-ua.nina.az/Win32.html) і [WinCE](https://www.wiki.uk-ua.nina.az/WinCE.html)). Легко переноситься на інші системи
* Сирці перебувають в суспільному надбанні
* Поставляється з автономним клієнтом інтерфейсу командного рядка, який може бути використаний для управління базами даних SQLite

Створення та обслуговування БД можуть здійснюватись через текстову консоль SQL-командами або через спеціальні інструменти, у тому числі — з графічним інтерфейсом користувача.

Сама бібліотека *SQLite* написана мовою C. Проте є реалізація бібліотеки на [JavaScript](https://www.wiki.uk-ua.nina.az/JavaScript.html) sql.js, яка дозволяє обробляти файли БД безпосередньо в браузері. Для інших мов програмування розроблено механізм підключення й роботи з БД через цю бібліотеку: [C++](https://www.wiki.uk-ua.nina.az/C%2B%2B.html), [Java](https://www.wiki.uk-ua.nina.az/Java.html), [Python](https://www.wiki.uk-ua.nina.az/Python.html), [Perl](https://www.wiki.uk-ua.nina.az/Perl.html), [PHP](https://www.wiki.uk-ua.nina.az/PHP.html), [Ruby](https://www.wiki.uk-ua.nina.az/Ruby.html), [Haskell](https://www.wiki.uk-ua.nina.az/Haskell.html), [Scheme](https://www.wiki.uk-ua.nina.az/Scheme.html), [Smalltalk](https://www.wiki.uk-ua.nina.az/Smalltalk.html), [Lua](https://www.wiki.uk-ua.nina.az/Lua.html) тощо. Засоби для роботи з [Tcl](https://www.wiki.uk-ua.nina.az/Tcl.html) включені в комплект постачання SQLite. Повний список наявних засобів можна знайти на сторінці проекту.

**ACID** (англ. *Atomicity, Consistency, Isolation, Durability*) — це набір властивостей, що гарантують надійну роботу транзакцій бази даних: атомарність, узгодженість, ізольованість, довговічність. В контексті баз даних, послідовність операцій з базою даних, яка задовольняє властивостям ACID, можна розглядати як одну логічну операцію над даними. Така послідовність операцій називається транзакцією. Наприклад, переказ коштів з одного банківського рахунку на інший містить численні операції, але є єдиною транзакцією.

### Atomicity — Атомарність

Транзакції часто містять в собі багато операцій. Атомарність (англ. *Atomicity*) гарантує, що жодна транзакція не буде виконана частково. Будуть або виконані всі операції, що беруть участь у транзакції, або не виконано жодної. Якщо протягом роботи однієї з операцій виникне помилка і операцію буде відхилено, то будуть відхилені також усі інші зміни, здійснені в межах транзакції. Система має бути атомарною у кожній ситуації, враховуючи відключення електроенергії, помилки та збої. Гарантія атомарності перешкоджає частковому оновленню бази даних, яке насправді може спричинити ще більші проблеми, аніж оновлення всієї бази в межах однієї транзакції.

Прикладом атомарної транзакції є переказ грошей з рахунку на рахунок, який проходить двома операціями: зняття грошей з першого рахунку та збереження їх на другому. Виконання цих операцій в атомарній транзакції забезпечує узгодженість даних бази, тобто гроші не віднімуться та не зарахуються, якщо одна з цих двох операцій зазнає невдачі.

### Consistency — Узгодженість

Відповідно до вимоги узгодженості (англ. *Consistency*), система має перебувати в узгодженому, несуперечливому стані до початку дії транзакції і по її завершенню. При цьому вона може перебувати в неузгодженому стані протягом виконання транзакції, проте ця неузгодженість не буде видимою за межами транзакції завдяки іншим властивостям — атомарності та ізольованості.

Таким чином, узгодженість гарантує [інваріантивність](https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D0%B2%D0%B0%D1%80%D1%96%D0%B0%D0%BD%D1%82_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) бази даних: будь-які дані, записані в базу, мають відповідати усім визначеним правилам, враховуючи обмеження, каскади, тригери та будь-яку їхню комбінацію. Це запобігає пошкодженню бази даних некоректною транзакцією, але не гарантує *правильність* транзакції. Посилальну цілісність гарантує відношення унікального ключа до зовнішнього.

Наприклад, при переведенні коштів з рахунку на рахунок, кошти можна спочатку зняти з першого рахунку, після чого нараховувати на другий. Відповідно, після зняття коштів, але до їх нарахування система перебуває в неузгодженому стані: коштів немає на жодному з рахунків. Але після завершення транзакції повна сума перебуватиме на другому (або першому у випадку скасування транзакції) рахунку.

### Isolation — Ізольованість

Ізольованість (англ. *Isolation*) означає, що жодні проміжні зміни не будуть видимі за межами транзакції аж до її завершення. Питання ізоляції стає актуальним при одночасній роботі багатьох транзакцій з тими самими даними. За цією вимогою, якщо дві транзакції намагатимуться змінити одні й ті самі дані, то одну з них буде відхилено або призупинено до завершення другої[6].

### Durability — Довговічність

Довговічність (англ. *Durability*) гарантує, що незалежно від інших проблем після відновлення працездатності системи результати завершених транзакцій будуть збережені. Іншими словами, якщо користувач отримав повідомлення про успішне завершення транзакції, то він може бути впевнений, що дані будуть збережені та відновлені у випадку збоїв.

1. *Tcl* орієнтована на автоматизацію рутинних процесів, швидке [прототипування](https://www.wiki.uk-ua.nina.az/%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D1%82%D0%B8%D0%BF%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F.html) та розробку платформо-незалежного програмного забезпечення, простоту використання.

   Незважаючи на те, що основне поширення *Tcl* отримала для створення інтерфейсів користувача і як вбудовувана мова, *Tcl* також підходить для інших завдань, таких як веб-розробка, створення [мережевих](https://www.wiki.uk-ua.nina.az/%D0%9A%D0%BE%D0%BC%D0%BF%27%D1%8E%D1%82%D0%B5%D1%80%D0%BD%D0%B0_%D0%BC%D0%B5%D1%80%D0%B5%D0%B6%D0%B0.html) [застосунків](https://www.wiki.uk-ua.nina.az/%D0%97%D0%B0%D1%81%D1%82%D0%BE%D1%81%D1%83%D0%BD%D0%BE%D0%BA.html), вирішення задач з адміністрування систем і тестування. [↑](#footnote-ref-1)