ІНФОРМАТИКА ТА ПРОГРАМУВАННЯ

Тема 29. Використання баз даних

Бази даних

- У розробці промислового програмного забезпечення організація даних має не менше значення, ніж написання самих програм.
- Взагалі, ми весь час звертали увагу на дані, якими оперують програми, починаючи з простих типів даних.
- Складені типи даних: рядки, списки, словники, кортежі, а також класи допомагають у організації даних у пам’яті комп’ютера.
- Але у багатьох програмах, окрім організації даних у пам’яті, важливим є збереження даних.
- Раніше для збереження даних ми розглядали файли а також обмінні формати XML та JSON.
- В той же час, для складних систем, що працюють зі структурованими даними, неможливо обійтись без баз даних.
- Тому зараз практично будь-яка комерційна програма використовує бази даних.

Бази даних та СУБД

- **База даних** (БД) – це сукупність даних, організованих відповідно до концепції, яка описує характеристику цих даних і взаємозв'язки між їх елементами.
- Більш просте означення: база даних – це будь-яка організована сукупність інформації.
- **Система управління базами даних** (СУБД) — це комплекс програмного забезпечення, що надає можливості створення, збереження, оновлення та пошуку інформації в базах даних а також здійснює контроль доступу до даних.
- СУБД призначені для підтримки усіх видів робіт з даними, організованими у БД.

Типи СУБД

- За час свого існування СУБД пройшли еволюційний шлях, впродовж якого домінували та використовувались різни типи СУБД:

Тип	Представники
Ієрархічні	IMS
Мережні	IDMS
Реляційні	ORACLE, MS SQL Server, MySQL
Об’єктно-орієнтовані	ORION
Нереляційні (noSQL)	MongoDB, Riak, Memcached, Redis
Нові реляційні (newSQL)	ScaleBase, MemSQL, NuoDB

Типи СУБД.2

- Ієрархічні та мережні СУБД відійшли у минуле. Об’єктно-орієнтовані так і не набрали популярності.
- На сьогодні поширеними є реляційні та нереляційні СУБД.
- Реляційні СУБД використовують для традиційних задач, а нереляційні, - для задач, пов’язаних з обробкою надвеликих масивів розподілених даних або для підтримки великих масивів даних складної структури.
- Нові реляційні СУБД – це підхід, який намагається конкурувати з попередніми двома.

Реляційні СУБД

- У реляційних СУБД дані організовані у відношення (relations), звідки і пішла назва цих СУБД.
- Відношення представлені двовимірними таблицями.
- Отже, можна сказати, що у реляційних базах даних дані зберігаються у таблицях.
- Стовпчики цих таблиць називають **полями** (або стовпчиками), я рядки, - **записами**.
- Кожна таблиця має фіксований набір полів та змінну кількість записів.
- Кожне поле має власне ім’я та тип: число, рядок тощо.
- Рядки таблиці не нумеруються, а їх порядок вважається невизначеним.
- Прикладами таблиць можуть слугувати Робітник (містить інформацію про робітника) та Посада (містить перелік посад разом з їх характеристиками).

Реляційні СУБД.2

- Таблиці зв’язують між собою за допомогою ключів.
- **Ключ** – це одне або декілька полів таблиці.
- Якщо у таблицях T1 та T2 визначено для зв’язування ключі K1 та K2, то зв’язаними вважаються ті записи T1 та T2, які мають однакові значення ключових полів K1 та K2.
- Первинним ключем називається такий ключ у таблиці, значення якого є унікальними для усіх записів цієї таблиці.
- Значення первинного ключа однозначно ідентифікує запис таблиці.
- Зовнішнім ключем (Foreign key) називається такий ключ даної таблиці, який зв’язаний з первинним ключем іншої таблиці.

Реляційні СУБД.3

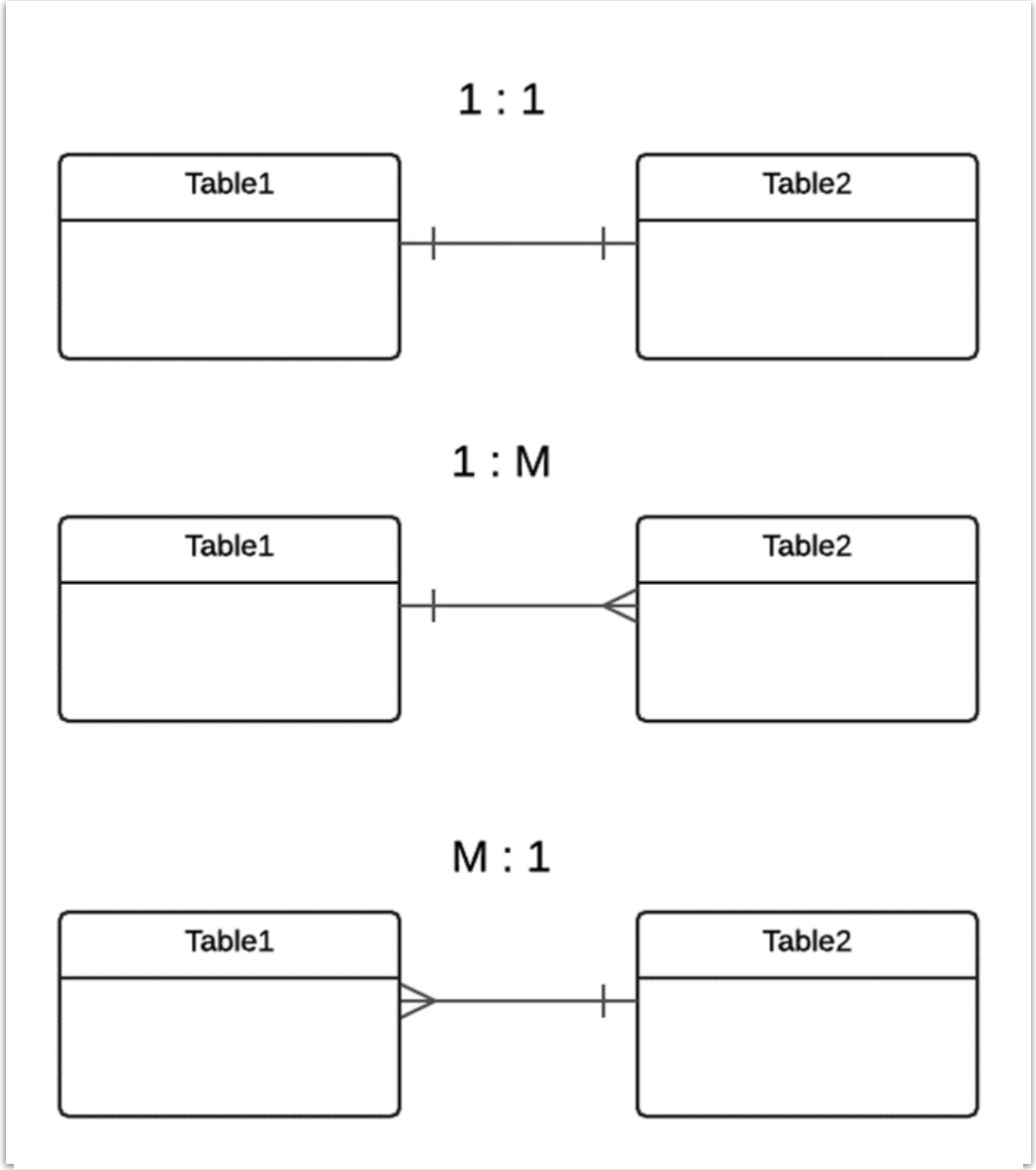
- Виділяють 4 типи зв’язків між таблицями T1 та T2:
  - Один до одного (1:1) – одному запису таблиці T1 відповідає не більше одного запису таблиці T2 (приклад: особа - студент)
  - Один до багатьох (1:M) – одному запису таблиці T1 відповідає багато записів таблиці T2 (приклад академічна група - студент)
  - Багато до одного (M:1) – багатьом записам таблиці T1 відповідає один запис таблиці T2 (приклад: студент – академічна група)
  - Багато до багатьох (M:M) – багатьом записам таблиці T1 відповідає багато записів таблиці T2 (приклад: студент – навчальна дисципліна)
- Зв’язок M:M реалізується у реляційних СУБД не напряму, а за допомогою проміжної таблиці, яка зв’язана з T1 та T2 зв’язками M:1.

ER-діаграми

- Для зображення структури реляційних баз даних використовують ER-діаграми.
- ER – це скорочення від Entity-Relation, тобто, сутність-зв’язок.
- У цих діаграмах відображають таблиці, поля а також зв’язки між таблицями.
- Таблиця має такий вигляд:

ER-діаграми.2

Зв’язки зображують стрілками наступним чином:



Короткі відомості про SQL

- Для обробки реляційних баз даних застосовують мову SQL (Structured Query Language –мова структурованих запитів).
- SQL дозволяє створювати бази даних, наповнювати та змінювати їх а також робити вибірки даних, що зберігаються у БД.
- SQL – це декларативна мова, одиницею якої є запит до БД.
- Ми обмежимось коротким описом основних можливостей SQL.
- Основні запити до БД:
  - SELECT – повертає вибірку даних з БД
  - INSERT – вставляє дані до таблиці БД
  - UPDATE - змінює дані у таблиці БД
  - DELETE - видаляє дані з таблиці БД
  - CREATE TABLE – створює нову таблицю у БД.

Короткі відомості про SQL.2

- Запит SELECT має вигляд:
- SELECT <поля> FROM <таблиця> WHERE <умова>,
  - де <поля> – це список полів таблиці, дані з яких будуть відібрані, <таблиця> – таблиця БД, <умова> – умова вибору. Замість списку полів може стояти ‘*’, що означає всі поля. Умова може містити імена полів, вирази, відношення та бульові операції. Відношення позначаються =, <>, >, <, >=, <=, IN та інші.
- Приклад запиту SELECT:

```
SELECT id FROM quiz WHERE theme="Ітератори та генератори"
```

-
- Запит INSERT має вигляд:
- INSERT INTO <таблиця> (<поля>) VALUES <значення>
  - де <таблиця> – таблиця БД, <поля> – це список полів таблиці, дані до яких будуть вставлені, <значення> - значення, які будуть вставлятись.
- Приклад запиту INSERT:

```
INSERT INTO res (points, maxpoints) VALUES (2, 6)
```

Короткі відомості про SQL.3

- Запит UPDATE має вигляд:
- UPDATE <таблиця> SET <поля=значення> WHERE <умова>
  - де <таблиця> – таблиця БД, у якій будуть змінюватись значення, <поля=значення> – це список пар: поле=значення через кому, <умова> – умова вибору записів.
- Приклад запиту UPDATE:
- `UPDATE friends SET phone="050 227-3946" WHERE name="Іван"`
-
- Запит DELETE має вигляд:
- DELETE FROM <таблиця> WHERE <умова>
  - де <таблиця> – таблиця БД, у якій будуть видалятись записи, <умова> – умова вибору записів.
- Приклад запиту DELETE:
- `DELETE FROM friends WHERE name="Іван"`

Короткі відомості про SQL.4

- Запит CREATE TABLE має вигляд:
- CREATE TABLE <таблиця> (<поля та типи>)
  - де <таблиця> – таблиця БД що буде створюватись, <поля та типи> - список полів та їх типів через кому.
- Приклад запиту CREATE TABLE:

```
CREATE TABLE `role` (
    `id`    INTEGER,
    `name`  TEXT
)
```

Запити SQL з параметрами

- Часто виникає необхідність використати параметри у запитах SQL, щоб один запит виконувався для різних значень цих параметрів.
- Щоб вказати параметр, у запиті у відповідне місце ставлять знак питання ‘?’ (деякі бази даних мають інший синтаксис параметрів).
- Приклад запиту з параметрами:

```
SELECT right_id, is_auth FROM function_right
WHERE role_id=? AND function_id=?
```

- Значення параметрів передаються СУБД окремо від самого запиту.

Транзакції

- Декілька запитів до БД можуть бути об’єднані у транзакцію.
- Кожна транзакція може бути здійсненою (commit) або скасованою (rollback).
- Поки транзакцію не здійснено, зміни до БД не вносяться.
- Якщо транзакцію скасовано, усі зміни анулюються.
- Механізм транзакцій дозволяє підтримувати цілісність даних у БД при виконанні низки взаємопов’язаних запитів.

Засоби обробки баз даних у Python

- Python надає можливість обробляти дані, що зберігаються у різних БД, та, навіть, має вбудовану СУБД SQLite.
- Обробка баз даних основана на єдиному програмному інтерфейсі доступу до даних.
- Для кожної СУБД створено (або створюється) окремий модуль, який реалізує цей інтерфейс.
- Такий підхід дозволяє писати програми обробки даних БД великою мірою незалежно від СУБД, яка буде використана.
- Більшість згаданих модулів не входять до стандартної поставки Python та потребують інсталяції.

Програмний інтерфейс баз даних Python

- Програмний інтерфейс баз даних Python називається DB API.
- Цей інтерфейс описує декілька класів, що взаємодіють з СУБД.
- Головними класами є Connection та Cursor.
- Клас Connection реалізує з'єднання з БД, включаючи аутентифікацію користувача, якщо потрібно.
- Також цей клас дозволяє отримати інформацію про налаштування СУБД та управляти транзакціями.
- Клас Cursor реалізує так званий курсор у базі даних.
- Курсор – це об'єкт, який містить інформацію про поточний стан БД або її певної таблиці.
- Курсор створюється у рамках існуючого з'єднання з БД (об'єкту Connection).
- Об'єкт курсор безпосередньо використовується для виконання запитів до БД та отримання даних.

База даних SQLite

- Ми розглянемо вбудовану у Python СУБД SQLite.
- Цю СУБД реалізовано у модулі sqlite3.
- Модуль реалізує DB API для SQLite.
- SQLite відрізняється від більшості інших СУБД тим, що для неї не потрібен окремий сервер.
- Ця СУБД запускається всередині процесу Python.
- Імпорт модуля виконується, як звичайно:

```
import sqlite3
```

- Для використання БД потрібно зв’язатись з нею (створити об’єкт класу Connection) за допомогою функції connect:

```
conn = sqlite3.connect(filename)
```

  - де filename – ім’я файлу БД, conn – об’єкт Connection.

Клас Connection

- Методи класу Connection зібрано у таблиці:

Метод	Опис
curs = conn.cursor()	Створити та повернути об’єкт курсор для подальшої обробки даних
conn.commit()	Здійснити транзакцію
conn.rollback()	Скасувати транзакцію
conn.close()	Завершити з’єднання з БД

Клас Cursor

- Методи та властивості класу Cursor зібрано у таблиці:

Метод/властивість	Опис
curs.execute(sql[, params])	Виконати запит sql з параметрами params (якщо передано). params – це кортеж значень параметрів запиту.
result = curs.fetchone()	Повернути дані останнього виконаного запиту (SELECT). Повертає кортеж значень полів першого запису, який видає запит.
result = curs.fetchmany(size)	Повернути дані останнього виконаного запиту (SELECT). Повертає список кортежів значень полів перших size записів, які видає запит.
result = curs.fetchall()	Повернути дані останнього виконаного запиту (SELECT). Повертає список кортежів значень полів усіх записів, які видає запит.
curs.rowcount	Властивість, що повертає кількість записів, які видав останній запит (SELECT).
curs.description	Властивість, що повертає список кортежів з назвами полів, які видав останній запит (SELECT). Назви містяться у нульових елементах кортежів списку.

Приклад: Телефонний довідник (БД)

- Створити телефонний довідник, який містить телефони знайомих.
- Реалізувати функції створення нового довідника, додавання одного запису, пошуку телефону за прізвищем а також зміни існуючого номеру телефону.
- Довідник повинен зберігатись у БД.
- У темі «XML та JSON» ми розглядали телефонний довідник, який зберігається у файлі відповідного формату.
- Для розв’язання задачі з використанням БД опишемо клас DBRefBook.
- Дані зберігаються у одній таблиці з ім’ям friends та текстовими полями name та phone

Клас DBRefBook

- Клас DBRefBook має одне поле:
  - self.filename - ім'я файлу БД довідника
- Клас також має методи: конструктор __init__, createrb – створити довідник, apprb – додати запис, searchrb – знайти телефон, replacerb – замінити телефон.
- Основна частина програми забезпечує вибір та виконання однієї з доступних функцій довідника.

Природні та штучні первинні ключі. Автонумерація

- Як вже зазначалось, первинний ключ однозначно ідентифікує запис у таблиці БД.
- Тому його значення повинно бути унікальним у таблиці.
- У деяких даних є природні первинні ключі, значення яких не повторюються та обов’язково присутні в усіх записах.
- Наприклад, табельний номер у інформації про робітника підприємства.
- Однак, частіше дані таблиці не мають такого поля або сукупності полів.
- Тоді використовують штучні первинні ключі, які прийнято називати id.
- Ці ключі є цілими числами, а їх значення задає сама СУБД, збільшуючи це значення на одиницю при додаванні кожного нового запису.
- Якщо ж запис видаляють з таблиці, значення його id більше ніколи не використовується.
- Такий тип даних називають цілим типом з автонумерацією.

Вкладені запити SQL

- Окрім звичайних, запити SQL можуть також бути вкладеними.
- Здебільшого це стосується запитів SELECT.
- Зокрема, результати запиту SELECT можуть використовуватись для завдання множини значень в умові, що вказана після WHERE.
- Наприклад:

```
SELECT * FROM course WHERE id IN (SELECT
course_id FROM student_course WHERE
student_id=?)
```

Типи даних SQLite

- SQLite має декілька вбудованих типів даних, які відповідають типам даних Python:

Тип або значення SQLite	Опис	Тип або значення Python
NULL	Порожнє значення поля даних	None
INTEGER	Цілий тип	int
REAL	Дійсний тип	float
TEXT	Тип рядок	str
BLOB	Поле, що містить двійкові значення (зображення, звук тощо)	bytes

- Ці типи даних задаються при створенні таблиць за допомогою запиту CREATE TABLE.
- Насправді, SQLite не перевіряє тип даних, які записуються у поле. Тобто, у базі даних можуть зберігатись дані й інших типів.

Засоби для створення та перегляду БД SQLite

- Незважаючи на те, що усі дії над БД SQLite можна виконати за допомогою SQL-запитів, для створення та перегляду БД зручно мати спеціальну програму, яка надає графічний інтерфейс доступу до БД.
- Такі програми називають клієнтами БД.
- Є декілька клієнтів SQLite, які вільно розповсюджуються та надають можливості створення таблиць, заповнення таблиці даним а також читання даних.
- Як приклад можна навести:
  - DB Browser for SQLite - http://sqlitebrowser.org/
  - або
  - phpLiteAdmin - https://www.phpliteadmin.org/
- Перше застосування встановлюється локально, а для другого потрібен Web-сервер.

Приклад: Зберігання тестів у БД. Проходження тестів

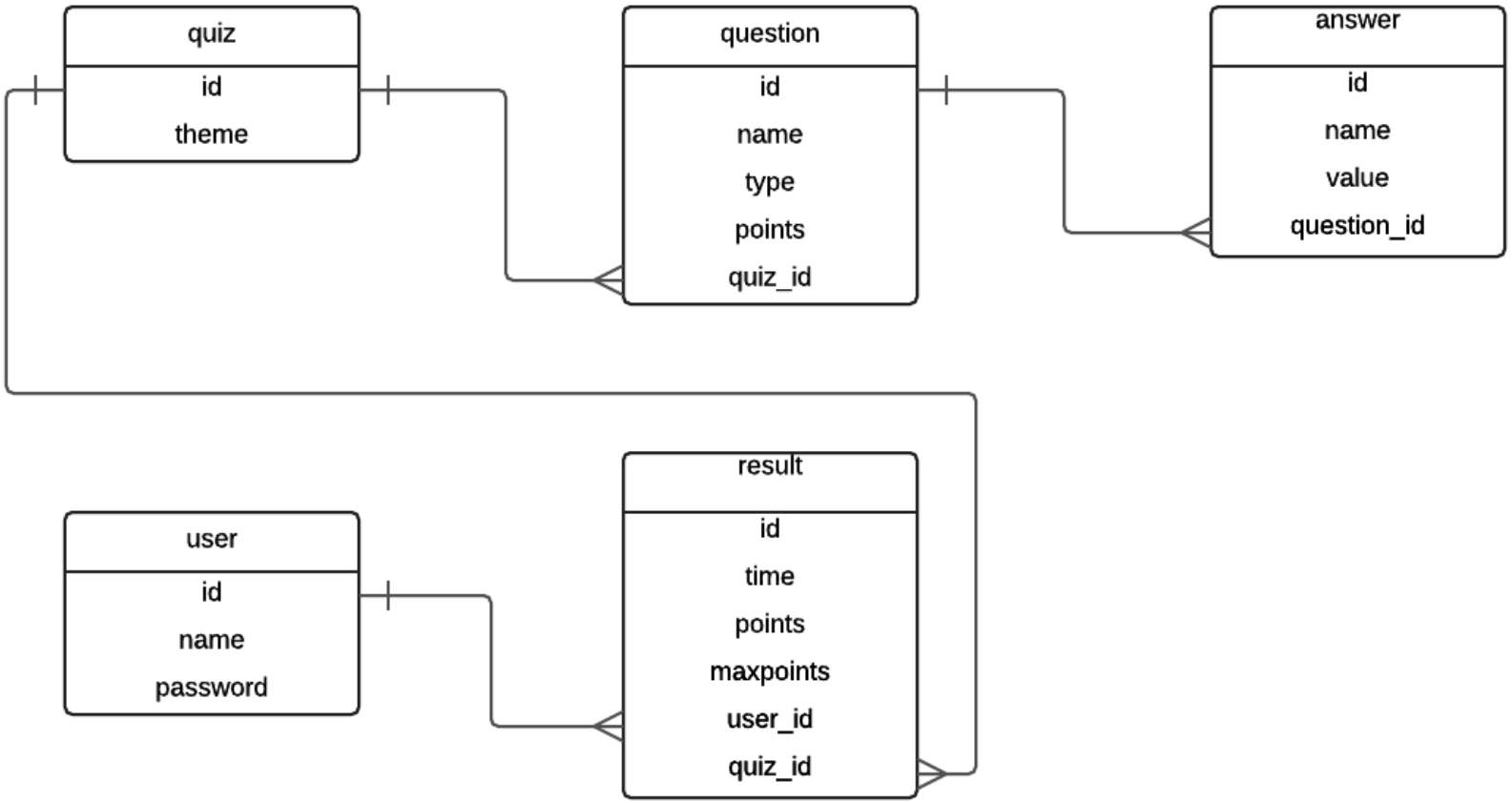
- Скласти програму, яка підтримує проходження тестів у веб-браузері.
- Програма повинна забезпечити введення імені та паролю користувача, вибір теми тесту, передачу питань, отримання та аналіз відповідей, а також показ результату тесту (кількості балів).
- Тест може містити питання трьох типів:
  - Так або ні
  - З вибором одного варіанту відповіді
  - З вибором декількох варіантів відповіді
- За кожне питання, на яке дано правильну відповідь, нараховується визначена кількість балів.
- Цю задачу ми розглядали у темах «Побудова веб-серверів» а також «XML та JSON».
- Тоді тести зберігались у файлі MS Excel або JSON.
- Зараз для збереження тестів, даних користувачів, результатів використаємо базу даних.

Проходження тестів. Структура БД

- База даних для проходження тестів складається з таблиць:
  - quiz - тест
  - question – питання тесту
  - answer – відповідь на питання тесту
  - user - користувач
  - result – результат проходження тесту
- Усі таблиці мають первинні ключі, що називаються id, які й використовуються для зв’язування з іншими таблицями.

Проходження тестів. Структура БД.2

- ER-діаграма для БД проходження тестів зображена нижче:



Проходження тестів. Реалізація.

- Реалізація здійснюється з використанням WSGI.
- Щоб реалізувати тести зі збереженням даних у БД, достатньо описати клас читання/запису даних тестів, який буде нащадком абстрактного класу TestIO, а також описати головний модуль.
- Назвемо клас, який реалізує інтерфейс читання/запису набору тестів з БД, TestDBIO.
- Окрім реалізації абстрактних методів та властивостей класу TestIO, цей клас також має внутрішні методи _readusers (прочитати список користувачів) та _readquiz (прочитати один тест).
- Головний модуль створює об’єкт класу QuizApplication та передає йому потрібні параметри.
- Також у цьому модулі ініціюється робота веб-сервера WSGI.

Приклад: Запис студентів на курси

- Скласти програму, яка здійснює запис студентів на курси за вибором.
- Кожен студент має вибрати задану кількість курсів за вибором.
- Курси додають або редагують викладачі – автори курсів.
- Також усі можуть подивитись перелік та опис курсів.
- Викладачі та студенти можуть подивитись список студентів, що вже записались на кожний курс.
- Це завдання передбачає, що до нашої програми будуть мати доступ користувачі, які здійснюють різні дії.

Приклад: Запис студентів на курси.2

- Отже, варто виділити ролі та функції (дії) користувачів:
  - Студент
    - Переглядає перелік курсів
    - Переглядає опис курсу
    - Записується на курс
    - Скасовує свій запис на курс («відписується» від курсу)
    - Переглядає список студентів, що вже записані на курс
  - Викладач
    - Переглядає перелік курсів
    - Переглядає опис курсу
    - Створює новий курс
    - Змінює курс (де він є автором)
    - Видаляє курс (де він є автором)
    - Переглядає список студентів, що вже записані на курс
  - Гість
    - Переглядає перелік курсів
    - Переглядає опис курсу
- У подальшому можливо додавання інших ролей, наприклад, «методист».

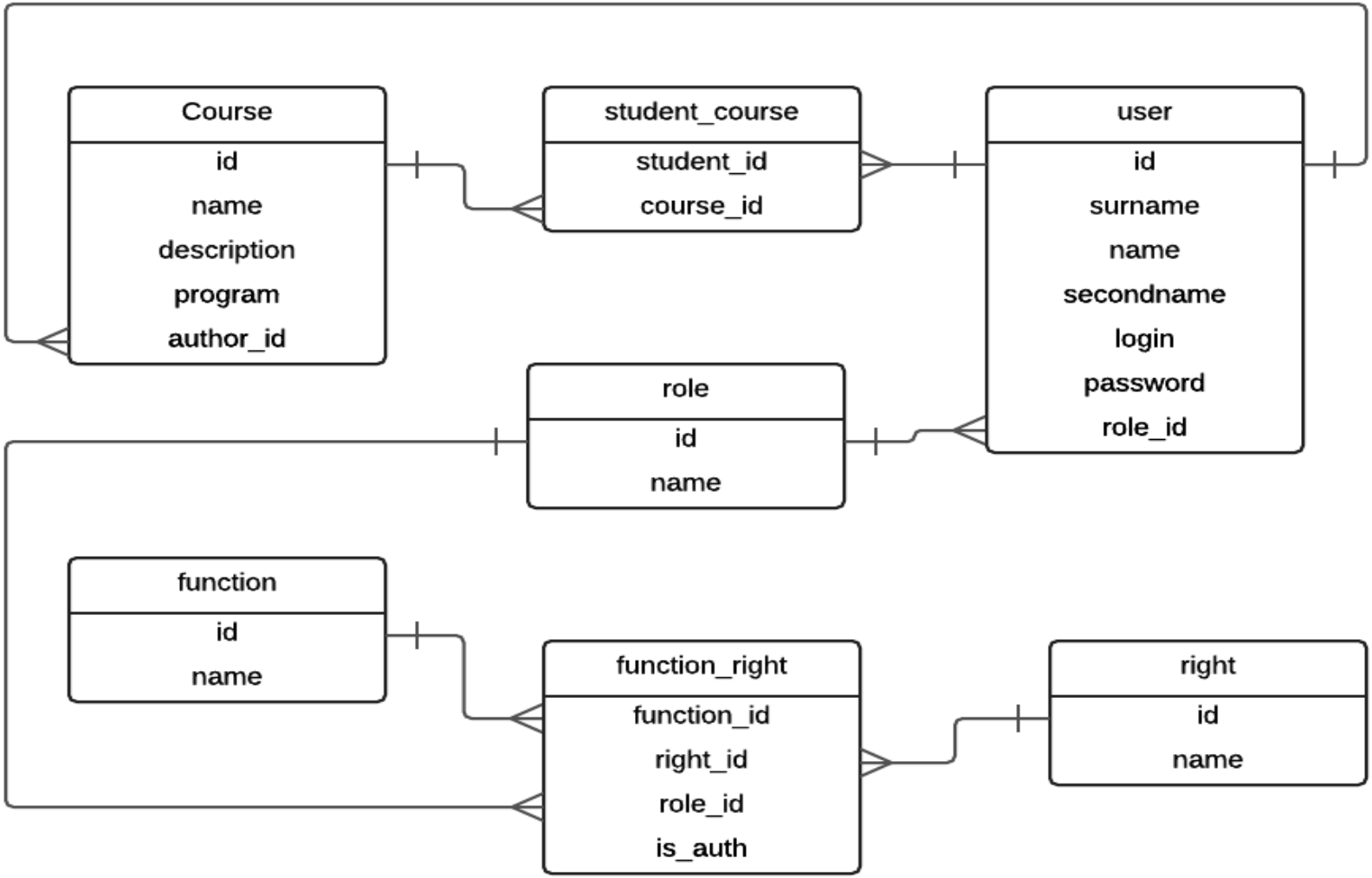
Запис студентів на курси. Реалізація

- Реалізація даної програми передбачає, що різні користувачі повинні мати різні права на виконання тих чи інших дій.
- При цьому, частина функцій є загальнодоступною, а частина - не перетинається.
- Будемо здійснювати керування правами доступу до функцій програми з бази даних.
- Для цього створимо декілька таблиць.
- У таблицю role (роль) будемо записувати ролі користувачів.
- У таблицю function (функція) – функції, які доступні користувачам програми.
- У таблицю right (право) – права користувачів:
  - "browse" – переглянути перелік
  - "create" - створити
  - "modify" - змінити
  - "delete" - видалити
  - "list" – показати список
  - "view" - переглянути
  - "apply" - записатись

Запис студентів на курси. Реалізація.2

- Таблиця function_right зв'язана з трьома останніми таблицями та визначає права, які має певна роль на певну функцію.
- Поле is_auth цієї таблиці вказує, чи обмежуються права тільки автором (наприклад, видалення курсу).
- Дані користувачів зібрано у таблиці user, а дані про курси, - у таблиці course. Зв'язок M:M між цими таблицями реалізовано за допомогою проміжної таблиці student_course.
- У самій програмі здійснимо розбиття функціональності, що пов'язана з базою даних, інтерфейсом користувача та управлінням на три окремих модулі:
  - t29_21_courses_db_io – модуль роботи з базою даних курсів, містить класи CourseDB та CourseCollection
  - t29_23_courses_application – модуль, що керує послідовністю виконання, містить класи CoursesApplication та CoursesSession
  - t29_22_courses_iface – модуль, що відповідає за інтерфейс користувача, містить клас CoursesInterface.
- Такий підхід (розбиття функціональності) є характерним для великих програмних систем.
- Головний модуль t29_24_courses_main починає роботу та запускає веб-сервер.

Запис студентів на курси. Реалізація.3



Клас CourseDB

- Клас CourseDB призначено для з'єднання з базою даних курсів та надання більш простих можливостей роботи з БД.
- Клас має поля:
  - self.urn - розташування БД курсів
  - self.conn - об'єкт зв'язку з базою даних
- Клас CourseDB також має методи __init__ - конструктор, get_cursor – з’єднатись з БД та повернути курсор, close – закрити з’єднання, get_data_dicts – отримати дані з запиту SQL у вигляді списку словників: [{<поле1>:<дані1>, …}, …].

Клас CourseCollection

- Клас CourseCollection призначено для отримання даних курсів.
- Клас має одне поле self.db - об'єкт БД – та низку методів, що повертають або змінюють дані БД та будуть використані у функціях управління (методах класу CoursesApplication).
- Це методи:
  - __init__ - конструктор,
  - get_courses отримати весь список курсів,
  - get_courses_authored - отримати список курсів авторства заданого викладача,
  - get_courses_applied – отримати список курсів, на які підписався студент,
  - get_user – отримати користувача,
  - get_course_by_id – отримати дані курсу за id,
  - create_course – створити курс,
  - modify_course – змінити курс,
  - delete_course – видалити курс,
  - apply_to_course – записатись на курс,
  - unapply_from_course – відписатись від курсу,
  - get_students_applied – отримати список студентів, що записались на курс

Клас CoursesApplication

- Клас CoursesApplication реалізує взаємодію з клієнтом під час запису на курси. Клас (метод __call__) викликається WSGI-сервером у відповідь на запит веб-клієнта.
- У запиті передається команда, яка має вигляд: <функція>/<підфункція>, наприклад “course/apply”.
- Ця команда трансформується заміною косої риски на підкреслення (“course_apply”), після чого здійснюється пошук та виклик метода з таким ім’ям.
- Функція записана у БД у таблиці function, а підфункція – це, як правило, одне з прав користувача.
- Завдяки такому підходу досягається просте розширення функцій системи: описати клас-нащадок CoursesApplication та додати методи обробки нових функцій/прав.
- Клас має поля:
  - self.last_id - номер останньої започаткованої сесії
  - self.sessions - словник сесій (об'єктів класу QuizSession)
  - self.cc - клас взаємодії з БД
  - self.iface - клас інтерфейсу користувача
  - self.path - шлях до поточного каталогу від wsgi-сервера

Клас CoursesApplication.2

- Клас також містить методи:
  - __init__ - конструктор,
  - __call__ - обробити запит клієнта,
  - start - обробити команду початку роботи (/),
  - course - спрямувати клієнту сторінку курсів,
  - course_create - спрямувати клієнту сторінку створення курсу,
  - course_created - додати створений курс до БД,
  - course_modify - спрямувати клієнту сторінку зміни курсу,
  - course_modified - змінити курс у БД,
  - course_delete - видалити курс у БД,
  - course_view - спрямувати клієнту сторінку перегляду курсу,
  - course_list - спрямувати клієнту сторінку списку студентів по курсу,
  - course_apply - записати студента на курс,
  - course_unapply - відписати студента від курсу

Клас CoursesSession

- Клас CoursesSession реалізує один сеанс запису на курси.
- Клас має тільки конструктор та зберігає у своїх полях:
  - self.app - клас, що містить даний (CoursesApplication)
  - self.sid - номер сесії
  - self.user - користувач

Клас CoursesInterface

- Клас CoursesInterface реалізує інтерфейс користувача запису на курси.
- У методах класу для побудови сторінок інтерфейсу використовуються HTML-файли та вставки:
  - course_login.html – сторінка входу до системи
  - courses.html – сторінка переліку курсів
  - course_row.html – вставка до сторінки переліку курсів, один рядок для курсу
  - course_create.html – сторінка створення нового (зміни існуючого) курсу
  - course_view.html – сторінка перегляду курсу
  - course_list.html – сторінка списку студентів, що записались на курс

Клас CoursesInterface.2

- Клас має одне поле self.path - шлях до поточного каталогу від WSGI-сервера.
- Клас також містить методи, що формують HTML-сторінки інтерфейсу користувача:
  - start - сформувати сторінку входу до системи,
  - login_error - сформувати сторінку у разі помилки входу,
  - courses_page - сформувати сторінку курсів,
  - course_create_modify_page - сформувати сторінку створення/зміни курсу,
  - course_view_page - сформувати сторінку перегляду курсу,
  - course_list_page - сформувати сторінку списку студентів курсу.
- Внутрішній метод _add_course додає рядок з назвою курсу та кнопками до таблиці курсів, а метод _add_applied – додає список курсів, на які записався студент.

Резюме

- Ми розглянули:
  1. Бази даних та СУБД. Типи СУБД
  2. Реляційні СУБД
  3. ER-діаграми
  4. Короткі відомості про SQL.
  5. Запити SQL з параметрами.
  6. Транзакції
  7. Засоби обробки баз даних у Python
  8. Програмний інтерфейс баз даних Python
  9. База даних SQLite
  10. Класи Connection та Cursor
  11. Природні та штучні первинні ключі. Автонумерація
  12. Вкладені запити SQL
  13. Типи даних SQLite

Де прочитати

1. Peter Norton, Alex Samuel, David Aitel та інші - Beginning Python
2. Magnus Lie Hetland - Beginning Python from Novice to Professional, 2nd ed – 2008
3. Mark Lutz - Programming Python. 4th Edition - 2011
4. Прохоренок Н.А. - Python 3 и PyQt. Разработка приложений – 2012
5. Jim Knowlton - Python Create Modify Reuse – 2008
6. Марк Саммерфилд, Программирование на Python 3. Подробное руководство. - Символ-Плюс, 2009.
7. Christopher A. Jones, Fred L. Drake, Jr. XML Processing with Python. - 2002
8. Alan Gauld, Python Tutorial - Learning to Program, - 2006