

Лабораторна робота №10
«SQL Coding Conventions & GitHub Actions»
Автор: Олександр Блажко, blazhko@ieee.org

1 Основні методичні рекомендації до виконання завдань роботи

1.1 Особливості норм кодування *SQL Coding Conventions*

Відомо, що основним недоліком процесу *Code Review* є висока трудомісткість аналізу «чужого» програмного коду рецензентами (рев'юєрами).

Тому для розробника важливо представляти свій програмний код для подальшого *Code Review* у формі, звичній для рецензента.

Така форма визначається відомим поняттям «норми кодування» (*Coding conventions*) як сукупність вказівок або порад стосовно структурної якості програмного забезпечення для конкретної мови програмування, які охоплюють: організацію файлів та їх зміст стосовно оформлення відступів, пропусків між елементами програмного коду, коментарів, норм найменування змінних, практик та принципів програмування.

В посібнику <https://www.sqlstyle.guide/ua/> надано рекомендації, які розроблено з урахуванням змісту книги *Joe Celko's SQL Programming Style*.

Умови найменування:

- 1) назва повинна бути унікальною та не бути зарезервованим ключовим словом;
- 2) назва починається з літери і не може закінчуватися символом підкреслення;
- 3) довжина назви не перевищує 30 знаків;
- 4) назва містить лише латинські літери, цифри та підкреслення для розділення слів;
- 5) назва не може містити скорочення, які не є широко розповсюдженими;
- 6) назва не повинна мати описові префікси, наприклад, *tbl_*, *col_*, або Угорську нотацію;
- 7) назва таблиці має форму множини;
- 8) назва таблиці не повинна співпадати з назвою її стовпця;
- 9) назва таблиці не повинна містити об'єднання двох імен таблиць разом, щоб створити назву таблиці зв'язків, наприклад, замість *cars_mechanics* можна обрати *services*;
- 10) назва стовпця має форму однини;
- 11) для назв стовців не використовуйте самостійно назву *id* як основний ідентифікатор для таблиці, а лише як суфікс;
- 12) для обчислюваних даних, наприклад, *SUM()* або *col+100*, використовується псевдонім з ключовим словом *AS*;
- 13) назва збережених процедур має містити дієслово;

14) при створенні таблиці закриваюча кругла дужка розміщується на одному рівні зі словом *CREATE*;

15) при створенні таблиці опис стовпців починається з відступом у 4-ри прогалини;

16) для відступу не використовуються символи табуляції.

Додатково назва стовпця може містити суфікси, які мають універсальне значення, що забезпечує легке читання та розуміння стовпців із коду *SQL*, наприклад:

- *_id* – унікальний ідентифікатор, наприклад стовпець, який є первинним ключем;
- *_status* – значення прапорця або інший статус будь-якого типу;
- *_total* – загальна сума або сума набору значень;
- *_num* – позначає, що стовпець містить будь-який тип чисел;
- *_name* – позначає ім'я якоїсь сутності;
- *_seq* – містить безперервну послідовність значень;
- *_date* – позначає стовпець, який містить дату;
- *_tally* – підрахунок якихось значень;
- *_size* – розмір чогось, наприклад, розміру файлу;
- *_addr* – адреса для запису може бути фізичною або нематеріальною;

Додатковою умовою для синтаксису створення таблиці є використання типів даних, загальних для *ANSI SQL*, а не специфічних для СКБД, наприклад:

- *NUMBER* замість *DECIMAL*, *NUMERIC*, *INTEGER*, *BIGINT*;
- *FLOAT* замість *DOUBLE*, *REAL*;
- *VARCHAR* замість *STRING*, *TEXT*;
- *TIMESTAMP* замість *DATETIME*.

Умови синтаксису запитів:

- 1) зарезервовані ключові слова використовуються у верхньому регістрі;
- 2) не використовуйте ключові слова, специфічні для СКБД, якщо існує ключове слово *ANSI SQL*, яке виконує таку ж функцію;
- 3) ключові слова (*SELECT, FROM, WHERE, INSERT, UPDATE, DELETE*) основного запиту повинні закінчувати на одній межі за рахунок використання прогалин;
- 4) прогалина додається до і після знака “дорівнює” (=);
- 5) прогалина додається після коми (,) та перед наступним виразом;
- 6) новий рядок або прогалина додається перед *AND* або *OR*;
- 7) новий рядок або прогалина додається після кожного визначення ключового слова;
- 8) підзапит вирівнюється по праву сторону з додатковим відступом із закриваючою круглою дужкою на новому рядку в тій самій позиції символу, що й відкриваюча дужка.

1.2 SQL Linters

Для спрощення процесу перевірки відповідності програмного коду нормам кодування, а також для автоматичної форматування тексту створено багато так званих Літерів (*Linters* – інструмент для видалення ворсинок з поверхні тканини).

Серед програм *SQL Linters* відомим є *SQLFluff* - <https://github.com/sqlfluff/sqlfluff>

Розробник може перевірити відповідність створеного *SQL*-запиту, використовуючи:

- 1) *Online*-програму за адресою <https://online.sqlfluff.com/>
- 2) утиліту командного рядку *sqlfluff*

На рисунку 1 наведено приклад перевірки *SQL*-запиту *Online*-програмою, для чого розробнику необхідно виконати наступні дії:

- 1) включити рядок із *SQL*-запитом у розділ «*Your Terrible SQL*», наприклад, запит

```
SELECT a + b FROM tbl ;
```

, в якому прогалини відображено червоним кольором;

- 2) вибрати *SQL*-діалект, який враховує особливості мовних конструкцій, наприклад, СКБД *Oracle*;

- 3) натиснути кнопку «*Help me.*».

Після виконання вказаних дій система надасть:

- 1) змінену форму *SQL*-запиту у розділі «*SQLFluff's Fixed SQL*»;
- 2) перелік порушень норм кодування з номером рядку, позиції у рядку, кодом порушення та коротким описом порушення.

The screenshot displays the SQLFluff online linter interface. On the left, under the heading "Your Terrible SQL", a text area contains the query "SELECT a + b FROM tbl ;" with line numbers 1 and 2. Below the text area is a "Dialect" dropdown menu set to "oracle" and a blue "Help me." button. On the right, under the heading "SQLFluff's Fixed SQL", a text area shows the formatted query "SELECT a + b FROM tbl;". Below this is a pink notification box stating "Hi! I found some errors for you to look at". A table lists the errors found:

| Code | Line / Position | Description |
|------|-----------------|--|
| L039 | 1 / 9 | Unnecessary whitespace found. |
| L039 | 1 / 12 | Unnecessary whitespace found. |
| L052 | 1 / 24 | Statements must end with a semi-colon. |

Рисунок 1 – Фрагмент сторінки зі результатом аналізу *SQL*-запиту

Online-програмою за адресою <https://online.sqlfluff.com/>

Подробиці про визначені порушення можна отримати за посиланням <https://docs.sqlfluff.com/en/stable/rules.html>

На рисунку 2 наведено фрагменти опису визначених двох порушень *L039* – «*Unnecessary whitespace found*» та *L052* «*Statements must end with a semi-colon*».

| | |
|--|---|
| <pre>class Rule_L039(code, description, **kwargs) Unnecessary whitespace found. This rule is sqlfluff fix compatible. Groups: all, core Anti-pattern SELECT a, b FROM foo Best practice Unless an indent or preceding a comment, whitespace should be a single space. SELECT a, b FROM foo</pre> | <pre>class Rule_L052(code, description, **kwargs) Statements must end with a semi-colon. This rule is sqlfluff fix compatible. Groups: all Anti-pattern A statement is not immediately terminated with a semi-colon. The . represents space. SELECT a FROM foo ; SELECT b FROM bar .; Best practice Immediately terminate the statement with a semi-colon. SELECT a FROM foo;</pre> |
|--|---|

Рисунок 2 – Фрагменти опису визначених двох порушень L039 та L052.

Кожний опис порушення визначає приклад порушення (*Anti-pattern*) та виправлений приклад (*Best practice*).

В подальшому розробник може використати змінений *SQL*-запит.

Для роботи з утилітою командного рядку *sqlfluff*, яка є *python*-утилітою, необхідно її встановити командою:

```
pip install sqlfluff
```

На рисунку 3 наведено приклад використання утиліти:

- в 1-му рядку створюється файл *test.sql* із *SQL*-запитом;
- у 2-му виконується утиліта в режимі *lint* (надання рекомендацій) для *SQL*-запиту із файлу *test.sql* з урахуванням особливостей діалекту мови *SQL* СКБД *Oracle*.

```
$ echo "SELECT a + b FROM tbl ;" > test.sql
$ sqlfluff lint test.sql --dialect oracle
== [test.sql] FAIL
L:   1 | P:   9 | L039 | Unnecessary whitespace found.
L:   1 | P:  12 | L039 | Unnecessary whitespace found.
L:   1 | P:  24 | L052 | Statements must end with a semi-colon.
```

Рисунок 3 – Фрагмент сторінки зі результатом аналізу *SQL*-запиту утилітою командного рядку *sqlfluff*

В результаті виконання утиліта представить перелік порушень норм кодування з номером рядку (*L:*), позиції у рядку (*P:*), кодом порушення та коротким описом порушення.

Для автоматичної заміни *SQL*-запиту на рекомендований, необхідно для утиліти *sqlfluff* замість опції *linter* вказати опцію *fix*.

Перелік правил, які використовує *sqlfluff*, є лише рекомендаціями, тому в деяких випадках, від деяких з них можна відповісти, якщо у вашій команді використовуються інші рекомендації. Для цього до командного рядку виклику утиліти можна додати опцію *--exclude-rules* зі списком номерів правил, наприклад:

```
--exclude-rules L029,L016
```

1.3 Автоматизація *SQL Coding Conventions*

1.3.1 *Continuous Integration*

Безперервна інтеграція (*Continuous Integration, CI*) – первинний, базовий процес оновлення ПЗ, в рамках якого всі зміни на рівні коду вносяться до єдиного центрального репозиторію через злиття робочих тимчасових гілок та основної гілки. Після кожного злиття (яке проходить по кілька разів на день) в системі, що змінюється, відбувається автоматичне складання, наприклад через упакування ПЗ у *Docker*-файл контейнерної віртуалізації) і тестування.

Приклади використання неперервної інтеграції в єдиному репозиторію:

- автоматична перевірка сирцевого коду файлів програмних модулів на відповідність нормам кодування мов програмування;
- автоматична компіляція та складання програмних модулів;
- автоматичне модульне тестування.

1.3.2 Налаштування сценаріїв *GitHub Actions*

GitHub Actions - це платформа для *Continuous Integration*.

Подробиці - <https://docs.github.com/en/actions>

Усі файли з командами опису процесів *GitHub Actions* зберігаються в *yml*-файлах (*YAML*), розміщених у каталозі *.github/workflows* репозиторію.

Докладніше про синтаксис *YAML* - <https://learnxinyminutes.com/docs/yaml/>

Для створення файлу команд треба перейти до пункту меню *Actions* або відразу за посиланням, наприклад, <https://github.com/oleksandrblazhko/ai202-test/actions/new>

Після переходу за посиланням система пропонує вибрати готовий шаблон команд під завдання, що часто виконуються, як показано на рисунку 4.

За бажання можна створити опис команд без шаблону, для чого використовується перехід за посиланням "*set up a workflow yourself*", як показано на рисунку 4.

Роботу з розділом *Actions* може виконувати *GitHub*-користувач, який є власником репозиторія або має відповідні права.

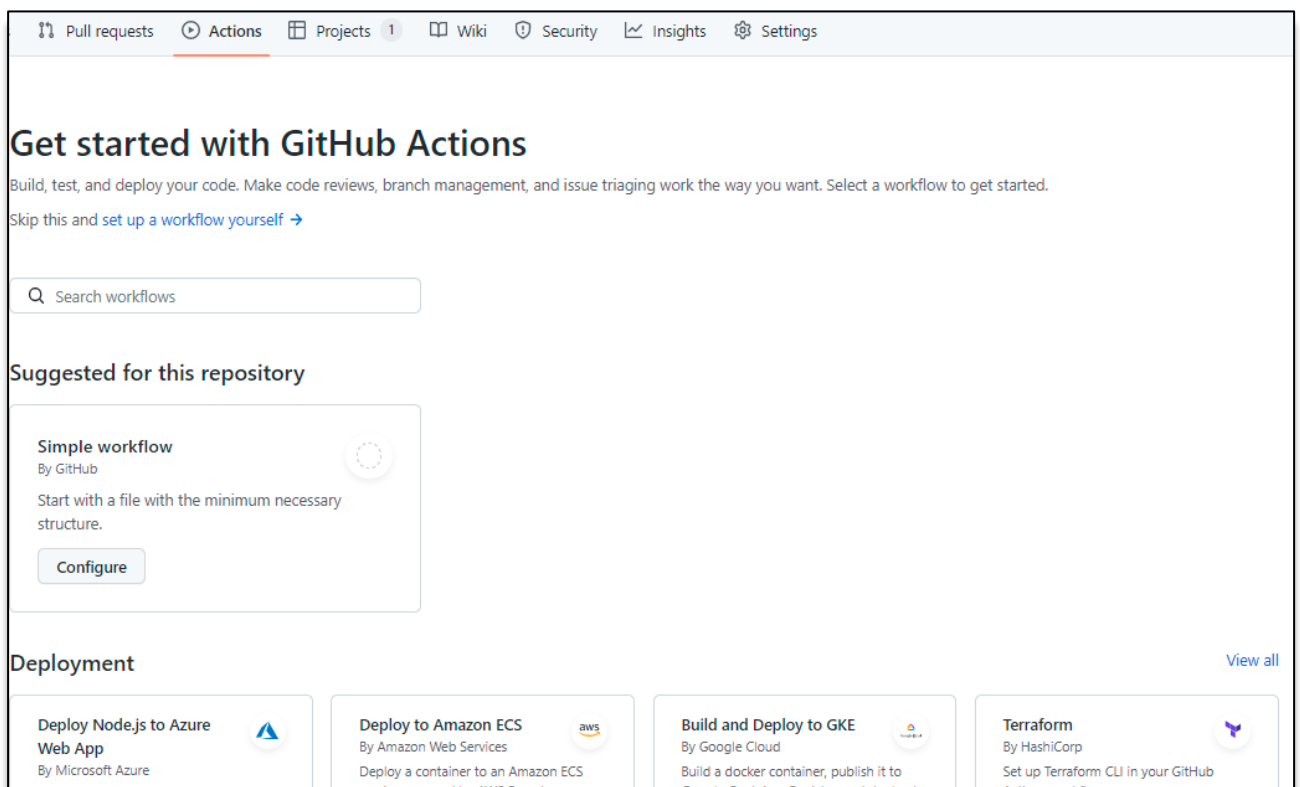


Рисунок 4 – Фрагмент сторінки вибору шаблону команд

Після переходу система покаже сторінку зі створення нового *yml*-файлу з назвою за замовчуванням *main.yml*, як показано на рисунку 5

Структура файлу сценарію визначається основними елементами:

- *Name* – назва workflow, наприклад, *Pull Request Control*;
- *On* – визначення типу подій, на які буде реагувати *workflow*, наприклад, *push*, *pull_request*, подробиці –

<https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows>

Jobs – опис команд, які необхідно виконати як реакцію на подію з використанням

окремого *Docker*-контейнеру з різними ОС, наприклад, *ubuntu-latest*, *windows-latest*, *macos-latest*, подробиці - <https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners>

При визначенні типу подій можна використовувати виклик підказок через клавіші *Ctrl+Space*, якщо курсор знаходиться у зоні *On*. На рисунку 5 показано приклад меню, яке пропонує вибрати необхідну подію. Першою подією є подія *push*.

Слід зазначити, що формат *YAML* вимагає коректно описувати ієрархію структури елементів з використанням прогалін.

Докладніше про структурні елементи - <https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions#on>

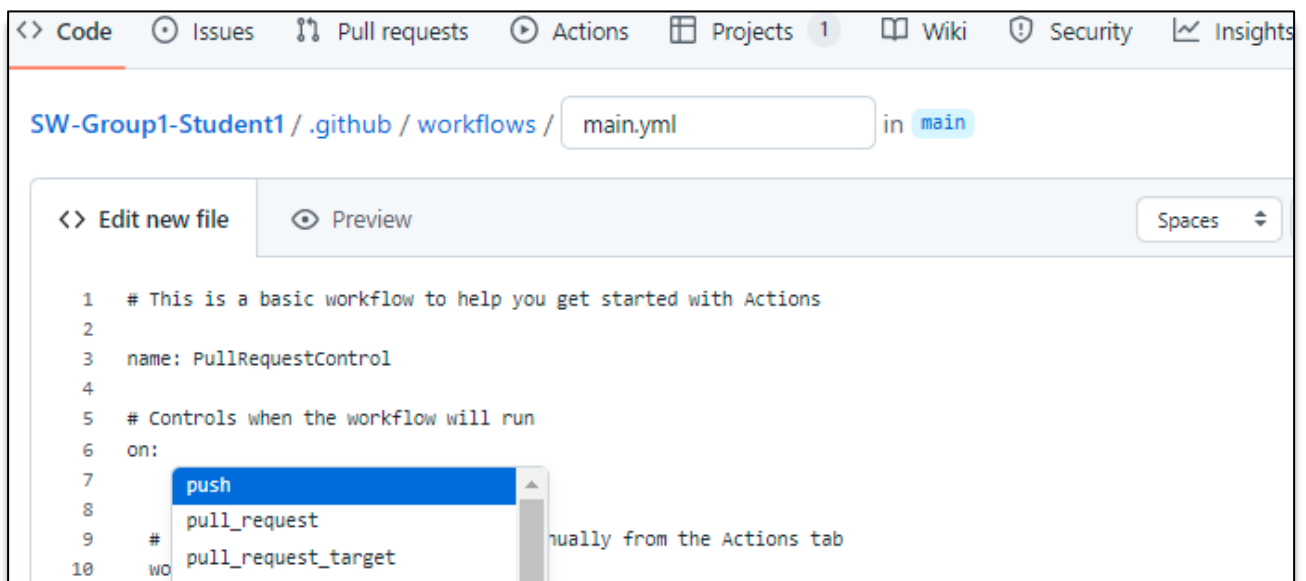


Рисунок 5 – Фрагмент сторінки зі створення нового *yml*-файлу
з назвою за замовчуванням *main.yml*

На рисунку 6 наведено приклад сценарію *HelloWorld.yml* з урахуванням наступних елементів:

- *name* - назва потоку робіт;
- *on*: обробка події *push*
- *branches*: обробка подій лише для гілки *LW11*
- *jobs*: назва роботи
- *runs-on*: виконання ОС *Ubuntu-latest* в окремому *Docker*-контейнері;
- *steps*: виконання кроків роботи
- *-name*: назва кроку Print Hello World;
- *run*: виконання Bash-команди `echo "Hello World!"`.

```
10 lines (10 sloc) | 174 Bytes

1  name: Hello World WorkFlow
2  on:
3    push:
4      branches: LW11
5  jobs:
6    HelloWorldWorkFlow:
7      runs-on: ubuntu-latest
8      steps:
9        - name: Print Hello World
10          run: echo 'Hello World!'
```

Рисунок 6 – Фрагмент сторінки з прикладом сценарію із файлу *main.yml*

Після завершення редагування файлу необхідно послідовно натиснути кнопки «*Start commit*» та «*Commit changes*», як показано на рисунку 7

Після виконання вказаних дій система автоматично запустить *workflow*-процес.

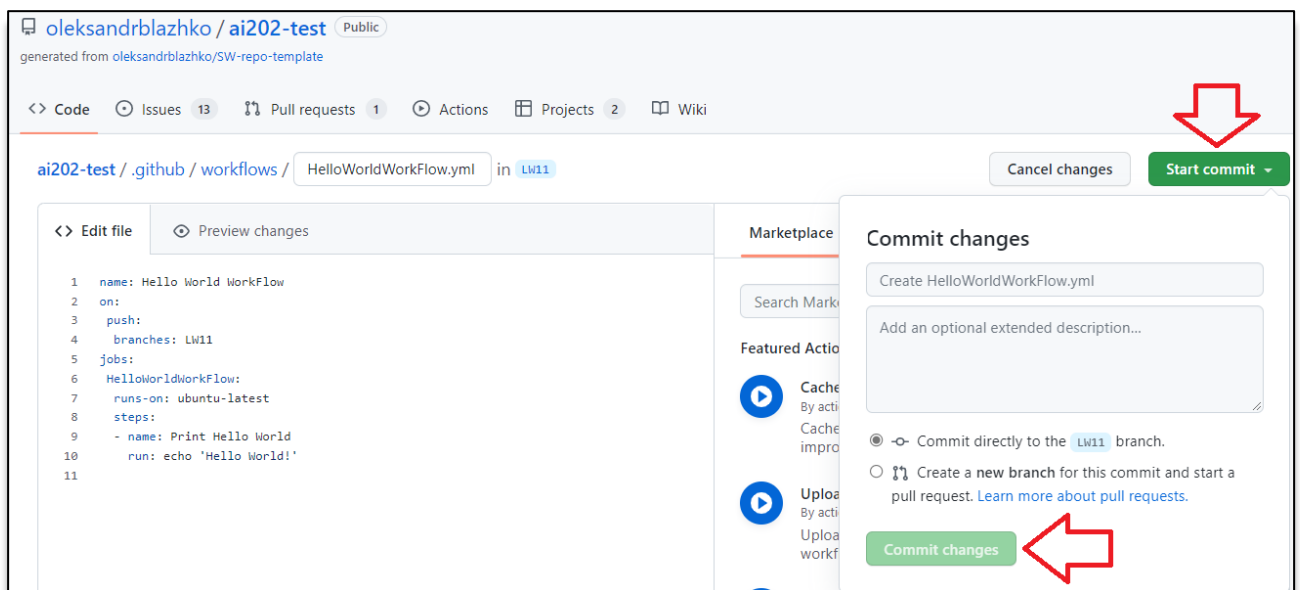


Рисунок 7 – Фрагмент сторінки із завершенням процесу редагування файлу сценарію *main.yml*

Для перегляду результатів запуску *workflow*-процесу необхідно перейти до пункту «*Actions*» як показано на рисунку 8.

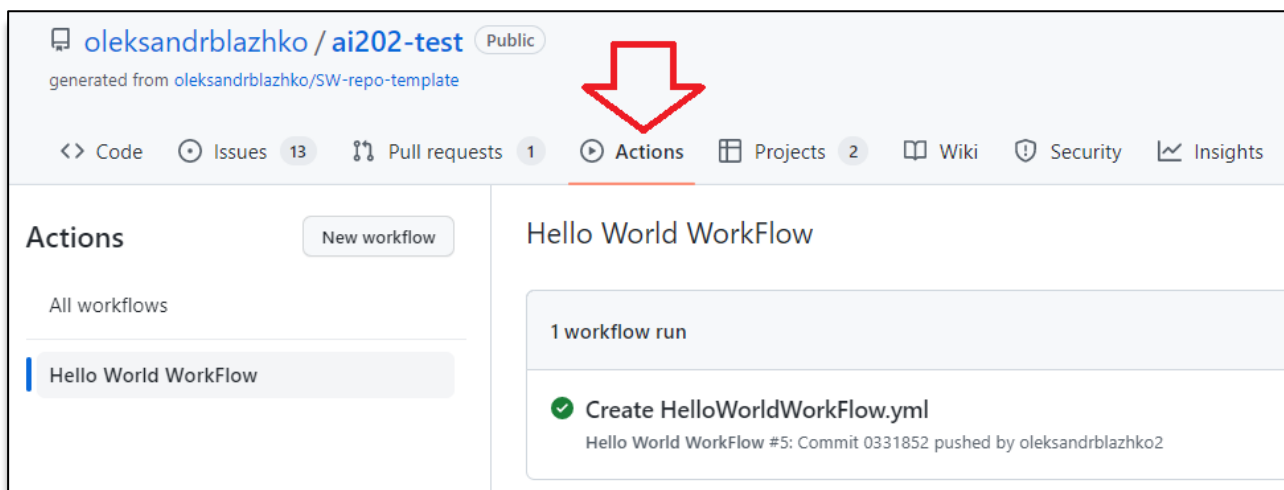


Рисунок 8 – Фрагмент сторінки зі списком виконання завдання *workflow*

Для обраного *workflow* система покаже його графічну візуалізацію виконання завдання, як представлено на рисунку 9.

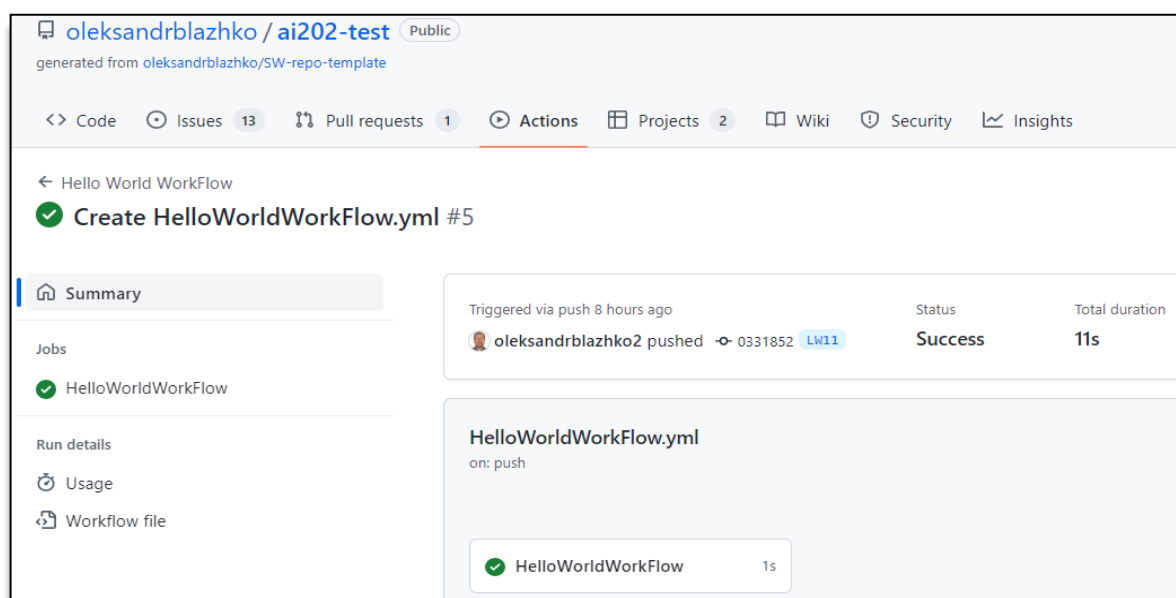


Рисунок 9 – Фрагмент сторінки з графічною візуалізацією виконання завдання

Натиснувши на блок з роботою, можна побачити процеси, які відбувалися під час виконання завдань, як показано на рисунку 10. В розділі «*Set up job*» розміщено опис результату виконання команди *runs-on* із запуском віртуальної ОС *Ubuntu*, де можна перейти за посиланнями з описом особливостей встановленої ОС та *Docker*-контейнером.

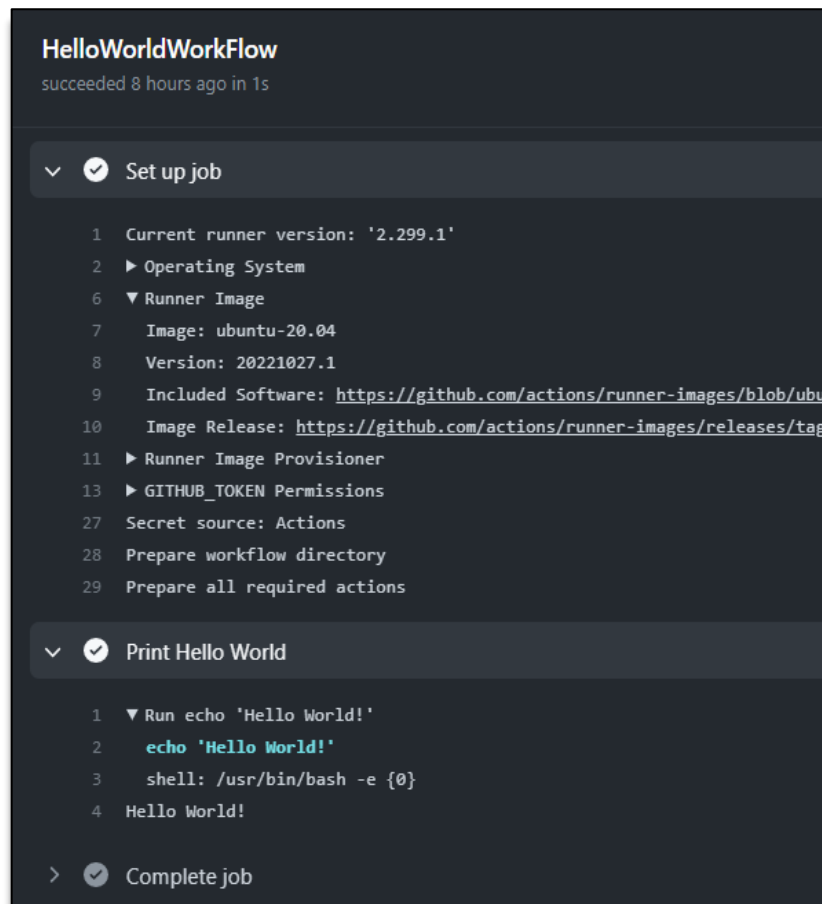


Рисунок 10 – Фрагмент сторінки з результатами виконання завдань

Для подальшого редагування *workflow* необхідно перейти до каталогу *.github/workflows/* репозиторія, як показано на рисунку 11.

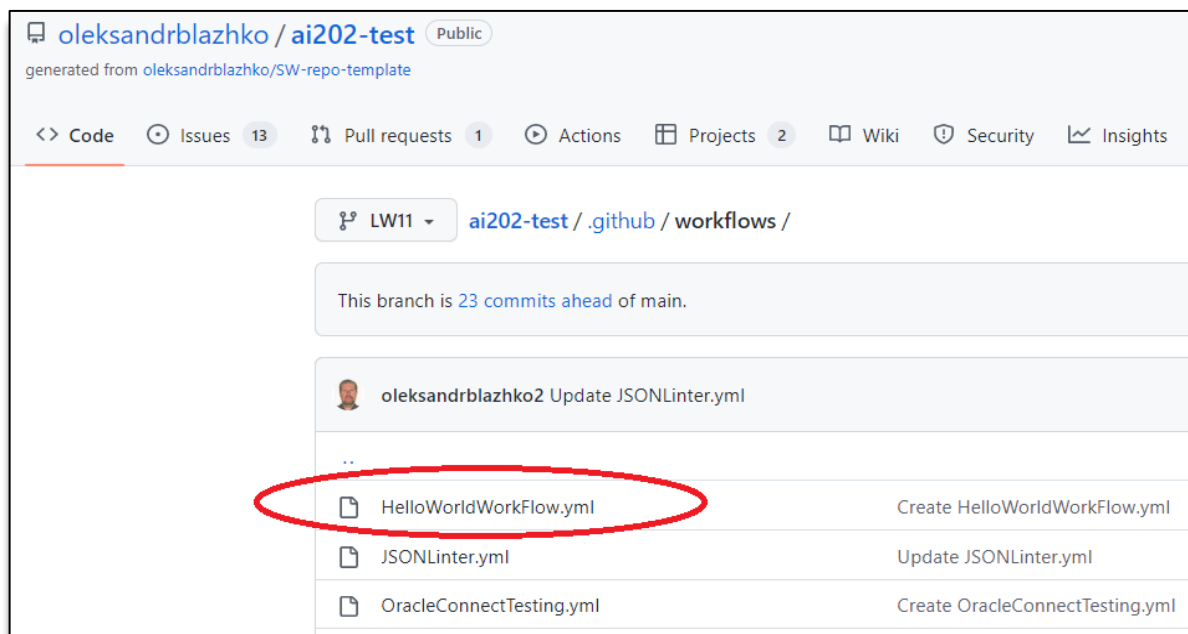


Рисунок 11 – Фрагмент сторінки зі списком *yml*-файлів в каталозі *.github/workflows/* репозиторія

На рисунку 12 наведено приклад сценарію *PrintSoftwareVersions.yml*, який містить вже дві роботи. Кожна робота виконує декілька кроків, які виконують команди виведення на екран версій програмного забезпечення.

За замовчуванням всі роботи *workflow* виконуються паралельно.

Яле, якщо необхідно, щоб друга робота починалася лише після успішного завершення першої роботи, використовується елемент *needs*, як показано в рядку 17 на рисунку 12.

```
1  name: Print Software Versions
2  on:
3    push:
4      branches: LW11
5  jobs:
6    PrintLanguageVersions:
7      runs-on: ubuntu-latest
8      steps:
9        - name: Print Java Version
10          run: java --version
11        - name: Print Node Version
12          run: node --version
13        - name: Print Python Version
14          run: python --version
15    PrintDBMSVersions:
16      runs-on: ubuntu-latest
17      needs: [PrintLanguageVersions]
18      steps:
19        - name: Print PostgreSQL Version
20          run: psql --version
21        - name: Print MySQL Version
22          run: mysql -V
23        - name: Print MongoDB Version
24          run: mongod --version
```

Рисунок 12 – Фрагмент сторінки з прикладом сценарію із файлу *PrintSoftwareVersions.yml*

На рисунку 13 наведено приклад змісту сценарію *linter_models.yml* з коментарями представлених команд *Git Hub Actions*. Як і раніше, файл може бути створений *GitHub*-користувачем, який є власником репозиторія або має відповідні права.

Розглядаючи формат розміщення команд слід ще раз зазначити, що *YAML*-розмітка вимагає зсуву право хоча б на одне знакомісце елементу-нащадка від батьківського елемента, інакше система буде підкреслювати червоним кольором невірно розташовані елементи.

```

1  name: SQL Linter
2  on:
3    # обробка лише події типу push
4    # для декількох значень використовується дужки, наприклад, [push, pull_request]
5    push:
6      # обробка подій лише для окремої гілки
7      branches: LW11
8  jobs:
9    SQLLinter:
10     # використання Docker-image із вказаною версією ОС
11     runs-on: ubuntu-latest
12     steps:
13     - name: Checkout Code in Repository
14       # клонування репозиторія для подальшої роботи з файлами на сервері
15       # в каталозі $GITHUB_WORKSPACE
16       uses: "actions/checkout@v3"
17     - name: Start SQL Linter
18       # виконання декількох послідовних команд shell-інтерпретатором
19       run: |
20         pip install sqlfluff
21         sqlfluff lint $GITHUB_WORKSPACE/query.sql --dialect oracle

```

Рисунок 13 – Фрагмент сторінки із прикладом змісту сценарію *SQLLinter.yml* з коментарями представлених команд *Git Hub Actions*

У рядку 17 рисунку 13 показано приклад команди *uses:*, яка дозволяє виконувати системні програмні *actions*-модулі, а також програмні модулі, створені зовні, які розміщено у каталозі - <https://github.com/marketplace?type=actions>

У рядку 17 рисунку 13 показано команду *checkout* для клонувати репозиторія у файлову систему віртуального серверу. Використовується версія v3 програмного модуля команди *checkout*.

Для того, щоб посилатися на поточний каталог репозиторія системна змінна *\$GITHUB_WORKSPACE*

В рядках 19-21 показано приклади команд, які виконуються в конвеєрному ланцюжку (символ *|*):

—для роботи з утилітою командного рядку *sqlfluff*, яка є *python*-утилітою, необхідно її встановити командою:

- pip install sqlfluff

—для використання утиліти можна виконати наступ команду:

- sqlfluff lint \$GITHUB_WORKSPACE/query.sql --dialect oracle

Остання команда у ланцюжку команд повинна визначати один з двох варіантів коду завершення:

- 0 – код успішного завершення останньої операції;
- будь-яке ціле ненульове значення – код невдалого завершення останньої операції.

На рисунку 14 наведено приклад результату виконання *workflow*-процесу, описаного файлом *SQLLinter.yml*:

- утиліта обробила файл *query.sql* із запитом *SELECT * FROM t1*;
- утиліта повідомила про дві помилки як перелік порушень норм кодування з номером рядку (*L:*), позиції у рядку (*P:*), кодом порушення та коротким описом порушення;
- подробиці про визначені порушення можна отримати за посиланням <https://docs.sqlfluff.com/en/stable/rules.html>
- виправити помилки можна за адресою <https://online.sqlfluff.com/>
- для автоматичної заміни *SQL*-запиту на рекомендований, необхідно для утиліти *sqlfluff* замість опції *linter* вказати опцію *fix*.

```
tblib-1.7.0 toml-0.10.2 tomli-2.0.1 tqdm-4.64.1 typing-extensions-4.4.0
45 == [/home/runner/work/ai202-test/ai202-test/query.sql] FAIL
46 L:  1 | P:  1 | L044 | Query produces an unknown number of result columns.
47 L:  2 | P:  1 | L009 | Files must end with a single trailing newline.
48 All Finished!
49 Error: Process completed with exit code 1.
```


>  Post Checkout Code in Repository

Рисунок 14 – Фрагмент сторінки з результату виконання *workflow*-процесу, описанного файлом *SQLLinter.yml*

Приклади файлів GitHub Actions можна знайти за посиланням – https://drive.google.com/drive/folders/1JcqS7j_3TiI4Zb1zY7PD6LgFXtF5MD2b

2 Завдання

Етап 1. *SQL Coding Conventions*

1.1 Візьміть окремі *SQL*-команди по одному рішенню декількох лабораторних робіт та назвіть їх відповідними номерами:

лабораторна робота №2 – 2.sql, лабораторна робота №3 – 3.sql, лабораторна робота №4 – 4.sql, лабораторна робота №6 – 6.sql, лабораторна робота №7 – 7.sql.

1.2 Протестуйте вибрані *SQL*-запити, послідовно скопіювавши їх до *Online*-утиліти <https://online.sqlfluff.com/>.

1.3 Перегляньте перелік виявлених помилок, скопіюйте їх та додайте у вигляді коментарів до кожного файлу перед рядком із запитом.

1.4 Збережіть запропоновані зміни *SQL*-запитів у відповідних файлах:

2-fixed.sql, 3-fixed.sql, 4-fixed.sql, 6-fixed.sql, 7-fixed.sql

Етап 2. Документування результатів роботи на Веб-сервісі *GitHub*

2.1 Розпочинаючи роботу над документуванням рішень лабораторної роботи, необхідно у вашому *GitHub*-репозиторії:

- створити *Issue* з назвою «*tasks-of-laboratory-work-10*».
- підключити до *Issue* ваш *GitHub-project* (правий розділ «*Projects*» сторінки з *Issue*);
- змінити статус *Issue* з «*Todo*» на «*In progress*», автоматично перевівши *Scrum*-картку з цим *Issue* на *Scrum*-дошку «*In progress*»;
- створити нову *Git*-гілку з назвою, яка відповідає назві *Issue*, наприклад, «*tasks-of-laboratory-work-10*» (використовується посилання «*Create a branch*» у правому розділі «*Development*» сторінки з *Issue*).

2.2 Після створення *Git*-гілки перейти до цієї гілки для створення оновлень файлів *Git*-репозиторію.

2.3 У новій гілці *Git*-репозиторію створити каталог з назвою «*10-SQLCoding ConventionsGitHubActions*» (кнопка «*Add file*» - «*Create new file*»), при створенні якого одночасно створити файл *README.md* з першим рядком «*SQL Coding Conventions & GitHub Actions*» зі стилем «Заголовок 3-го рівня» мови розмітки *Markdown* (три символи решітка ###).

Етап 3. *SQL Coding Conventions & GitHub Actions*

3.1 Розмістити файли *2.sql*, *3.sql*, *4.sql*, *6.sql*, *7.sql* в *GitHub*-каталозі «*10-SQLCoding ConventionsGitHubActions*».

3.2 Створити сценарій *SQLCodeConvention.yml* з перевірки норм кодування *SQL*-запитів для *push*-події за прикладом з рисунку 13, але враховуючи вашу гілку репозиторію. В сценарії вказати шаблон назв файлів **.sql*, розміщених у каталозі «*10-SQLCoding ConventionsGitHubActions*»

3.3 Переглянути результат виконання сценарію, який автоматично запустив перевірку норм кодування *SQL*-команд. Результат перевірки зафіксувати у вигляді фрагменту знімку екрану за прикладом з рисунку 14. Фрагмент знімку екрану зберігти у файлі *SQLCodeConventionResult.jpg*

3.4 Розмістити файли *2-fixed.sql*, *3-fixed.sql*, *4-fixed.sql*, *6-fixed.sql*, *7-fixed.sql* в *GitHub*-каталозі «*10-SQLCoding ConventionsGitHubActions*».

3.5 Змінити у сценарію *SQLCodeConvention.yml* шаблон назв файлів з **.sql* на **-fixed.sql*

3.6 Переглянути результат виконання сценарію, який автоматично запустив перевірку норм кодування *SQL*-команд. Результат перевірки зафіксувати у вигляді фрагменту знімку екрану за прикладом з рисунку 14. Фрагмент знімку екрану зберігти у файлі *SQLFixedCodeConventionResult.jpg*

3.4 Розмістити в каталозі «*10-SQLCoding ConventionsGitHubActions*» *GitHub*-репозиторія файли *SQLCodeConventionResult.jpg*, *SQLFixedCodeConventionResult.jpg*

Етап 4. Завершення лабораторної роботи

Виконати запит *Pull Request*, розпочавши процес *Code Review*.

Під час створення *Pull Request* необхідно вказати:

- *Reviewers* = *Oleksandr Blazhko*, *Maria Glava*;
- *Labels* = *enhancement (New feature or request)*;
- *Projects* = посилання на *GitHub-project*.

Завершення процесу *Code Review* відбудеться до початку нового заняття, після чого викладач закриє *Issue*, завершуючи процес виконання завдань з лабораторної роботи.

Під час консультації в понеділок ви зможете отримати більше коментарів щодо ваших рішень.