**Комп’ютерний практикум**

**з дисципліни “Основи аналізу алгоритмів”**

Версія 1  
18 вересня, 2022

**Мета:** Ознайомлення із різноманітними структурами даних, що використовуються при проектуванні та розробці баз даних. Отримання навичок реалізації запропонованих структур даних та алгоритмів для роботи з ними. Отримання навичок роботи із синтаксичним аналізом.

## Порядок виконання роботи

Необхідно розробити консольну програму, яка надає користувачу інтерфейс для взаємодії із колекціями даних певного типу, відповідно до отриманого варіанту. Команди повинні зчитуватись із стандартного потоку вводу, а результат виконання виводитись у стандартний потік виводу. Перелік команд, які необхідно підтримати а також детальний опис очікуваної поведінки програми залежить від варіанту і наводиться нижче. Рекомендований до ознайомлення теоретичний матеріал наводиться у вигляді набору посилань та розміщено в кінці документу.

Завдання кожного варіанту складається із обовʼязкової та додаткової частин, остання з яких позначається в тексті варіанту символом **\***. Обовʼязкова частина має бути виконана повністю, в той час як додаткову можна не робити взагалі або виконати частково. В такому випадку бали за неї будуть нараховані пропорційно до кількості виконаного завдання.

## Загальні зауваження стосовно визначення команд

* **[ ... ]** - опціональна частина команди. Тобто фрагмент, що міститься в дужках може бути як присутній так і відсутній в команді. Зокрема, **[, option\_n]** означає довільну кількість розділених комою повторень фрагменту команди, вказаного безпосередньо перед квадратними дужками. Запис \***[ ... ]** означає, що фрагмент в дужках є необовʼязковим і, при цьому, його підтримка очікується тільки у разі виконання додаткової частини завдання.
* **(option\_1 | option\_2 | ... | option\_n)** - вибір однієї з альтернатив, розділених символом **|**.

## Зауваження та рекомендації стосовно виконання роботи

* Що стосується роботи з командами, кінцева реалізація програми повинна задовольняти наступним вимогам:
  + Команди можуть включати в себе довільну кількість пробільних символів, які не повинні впливати на результат синтаксичного аналізу та інтерпретації введеної команди (тобто повинні просто ігноруватись). Виключення - якщо пробіли входять до складу рядка, тобто знаходяться між двома сусідніми символами **“** та **”**. Пробільними необхідно вважати щонайменше наступні символи: **‘ ‘**, **‘\t‘**, **‘\r‘**, **‘\n‘**.
  + Команди можуть займати декілька рядків. Ознакою кінця команди є символ **‘;‘**. Всі символи, що знаходяться у зчитаному рядку після символу **‘;‘** можуть бути проігноровані. Не дозволяється розбивати на декілька рядків ключові слова, ідентифікатори, рядки, числа тощо. Приклад коректно введної команди **CREATE cats (id INDEXED, name INDEXED, favourite\_food);**:  
    **> CREATE cats (  
    ... id INDEXED,  
    ... name   
    ... INDEXED  
    ... , favourite\_food  
    ... ); // ignored characters**
  + Програма повинна бути чутливою до регістру символів, використовуваних у рядках та ідентифікаторах (назвах таблиць, множин тощо) і нечутливою до регістру символів, використовуваних у ключових словах (виділених **синім** кольором у визначеннях).
  + Ідентифікатори (назви таблиць, множин тощо) повинні задовольняти наступному регулярному виразу - **[a-zA-Z][a-zA-Z0-9\_]\***. Тобто:
    - Першим символом може бути будь-який алфавітний символ - **[a-zA-Z]**.
    - Всі подальші символи (якщо є) можуть бути будь-якими алфавітно-цифровими символами або символом **\_** - **[a-zA-Z0-9\_]\***.
* Програма повинна виявляти та коректно обробляти помилкові ситуації і виводити зрозуміле повідомлення про помилку для користувача (формат не регламентується, але повинен бути адекватним) в якомога більшій кількості ситуацій. Приклади ситуацій, які варто відслідковувати та обробляти:
  + Некоректний синтаксис команди.
  + Спроба створити структуру даних із іменем, яке вже використано.
  + Спроба виконати команду пошуку або видалення даних відносно структури, якої не існує.
  + Спроба передати недостатню кількість значень команді **INSERT** у варіантах із реляційною моделлю даних.
  + Спроба отримати неіснуючі у таблиці стовпці командою **SELECT** у варіантах із реляційною моделлю даних.
  + Тощо.
* Синтаксичний розбір тексту (парсинг) зазвичай виконується у два етапи - **лексичний аналіз** і, власне, **синтаксичний аналіз**. На першому етапі вхідна послідовність символів (текст) розбивається на так звані лексеми (токени), що представляють собою примітивні елементи, з яких складаються більш складні вирази мови, з якою ми працюємо. Лексемами можуть бути ключові слова, ідентифікатори, літерали різноманітних типів даних (рядки, числа) тощо. На другому етапі послідовність лексем перевіряється на змістовність (коректність) та перетворюється у певну структуру даних, зручну для подальшої обробки (зазвичай якийсь із видів абстрактного синтаксичного дерева). Більше інформації про даний підхід, разом із практичними прикладами, можна знайти в джерелі [4]. Крім того, нижче наведено деякі додаткові зауваження стосовно використання готових бібліотек для парсингу.
* В усіх варіантах з реляційною моделлю даних (1-7) в якості додаткового завдання (**\***) необхідно підтримати індексацію довільних стовпців таблиці. Структура даних, використовувана в якості індексу, повинна задовольняти наступні вимоги: вона повинна бути відсортована за значенням проіндексованого стовпця і дозволяти ефективний пошук як за точним значенням стовпця (умови виду **column = “value”**) так і за діапазоном значень (умови виду **column ( > | < ) “value”**). З огляду на ряд спрощень, прийнятих в рамках данного практикуму, найкращою структурою даних, яка задовольняє наведеним вище умовам, є просте бінарне дерево, яке і рекомендується використовувати для реалізації індексів. Більш того, рекомендується використати готову реалізацію бінарного дерева або іншої відсортованої структури даних (див. нижче зауваження стосовно використання бібліотек).  
  Всі команди, які містять ключове слово **WHERE**, тобто передбачають певний пошук по таблиці, повинні перевіряти наявність індексів по стовпцям із умови і використовувати їх для оптимізації пошуку. Крім того, за наявності, підходящі індекси повинні використовуватись для пришвидшення виконання команд, які містять клюві слова **JOIN**, **LEFT\_JOIN**, **FULL\_JOIN**, **ORDER\_BY**, **GROUP\_BY**. У випадку, коли таблиця має одночасно декілька індексів, які можуть бути використані для виконання запиту (наприклад, один для виконання **WHERE**, а інший для виконання **JOIN**), ви можете вибрати будь-який із них на ваш розсуд (наприклад, завжди віддавати перевагу індексу, який пришвидшує виконання **JOIN**, а фільтрацію рядків виконувати повним перебором). За відсутності підходящих індексів пошук може бути виконано будь-яким іншим доступним способом. Для отримання додаткової інформації стосовно того як саме індекси можуть використовуватись для пришвидшення виконання SQL команд рекомендується джерело [3].
* У кожному варіанті вам пропонується працювати тільки з одним із двох примітивних типів даних - числами або рядками. При розробці програми можете вважати, що виконуються наступні припущення:
  + Рядки є послідовностями символів, що знаходяться між двома сусідніми символами **“** та **”**, тобто можна вважати, що всередині рядка символ **“** використовувати заборонено.
  + Реалізація рядків зобов’язана підтримувати тільки ASCII символи. Підтримка символів, що виходять за межі ASCII (наприклад, символів Юнікоду) вітається, але не вимагається.
  + Реалізація чисел зобов’язана підтримувати тільки цілі числа (в тому числі й від’ємні), але довільної розрядності.
* Зверніть увагу, що в літературі R-дерева та RD-дерева описуються так, що степінь розгалуження у них є більшою за 2 (подібно до B-дерев). Ця оптимізація має сенс в рамках БД, яка працює з диском, але не є обов’язковою в межах нашого практикуму. Тому рекомендується реалізовувати такі дерева зі степінню розгалуження 2, тобто кожен вузол має не більше двох нащадків.
* Використовувати стандартні та сторонні бібліотеки дозволяється, але із деякими обмеженнями. Не можна використовувати готові реалізації:
  + Баз даних та бібліотек, що дозволяють зберігати довільні масиви даних та виконувати пошук по ним вбудованими засобами (наприклад Pandas).
  + Бібліотек повнотекстового пошуку (наприклад Lucene).
  + Таких структур даних як R-дерева, KD-дерева, RD-дерева, префіксні дерева, інвертовані індекси.

Можна використовувати готові реалізації:

* + Генераторів парсерів, які працюють на основі граматик, наприклад ANTLR (для всіх варіантів,крім 7-го). Зверніть увагу, що у запропонованих вам варіантах це не завжди простіше, ніж написати парсер самостійно. Парсинг запитів в усіх варіантах, крім 7-го, можна виконати просто за допомогою регулярних виразів - це найшвидший та найпростіший спосіб.
  + Бінарних дерев або інших відсортованих структур даних (актуально для варіантів 1-7). Основною вимогою тут є використання правильних програмних інтерфейсів, які дозволяють виконувати пошук ефективно, на противагу повному обходу всієї структури даних.
  + Генераторів ASCII таблиць (актуально для варіантів 1-7). Такі бібліотеки є для кожної мови програмування, їх можна знайти в Google запитом виду {language} ascii table generator.
* Ви повинні бути спроможні проаналізувати часову і просторову складність побудованих вами алгоритмів (як внесення даних до колекцій так і пошуку по ним).

## Варіанти завдань

### **Варіант №1**: Реляційна модель із підтримкою сортування

Даний варіант передбачає, що, для спрощення реалізації, всі стовпці у створюваних таблицях мають рядковий тип даних.

Перелік команд, які необхідно реалізувати:

1. **CREATE table\_name (column\_name \*[INDEXED] [, ...]);** - створення таблиці **table\_name**, що містить стовпці, визначення яких наведено в круглих дужках. Визначення стовпця складається щонайменше з його імені. Додаткова частина (**\***) передбачає, що у визначенні стовпця ще може бути присутнє опціональне ключове слово **INDEXED**, яке означає необхідність індексування даного стовпця. Якщо ключове слово **INDEXED** не наведено, індекс створювати не потрібно. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Table table\_name has been created”**.  
     
   **Приклади:**  
   **CREATE students (name, group);  
   CREATE cats (id INDEXED, name INDEXED, favourite\_food);**
2. **INSERT [INTO] table\_name (“value” [, ...]);** - додавання нового рядка з даними до таблиці **table\_name**. Кількість розділених комою значень в круглих дужках повинна дорівнювати кількості стовпців у таблиці. Порядок значень повинен відповідати тому порядку стовпців, який використовувався при створенні таблиці командою **CREATE**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“1 row has been inserted into table\_name.”**.  
     
   **Приклади:**

**INSERT INTO cats (“1”, “Murzik”, “Sausages”);  
INSERT cats (“2”, “Pushok”, “Fish”);**.

1. Формування вибірки із таблиці **table\_name**: **SELECT FROM table\_name  
    [WHERE condition]  
    [ORDER\_BY column\_name [(ASC|DESC)] [, ...] ];**

Вибірка повинна включати всі стовпці таблиці, причому в порядку, в якому вони були згадані в команді **CREATE** при створенні таблиці.

Якщо вказано ключове слово **WHERE** та слідуюча за ним умова, то команда повинна включити до вибірки тільки ті рядки, які задовольняють дану умову. В іншому випадку, необхідно повернути всі рядки із вказаної таблиці. Умова **condition** має наступну структуру:   
 **condition := column\_name\_1 > (column\_name\_2 | “value”)**.  
Тобто умова складається із однієї бінарної операції порівняння **>**, зліва від якої повинен бути вказаний стовпець таблиці (**column\_name\_1**), а справа може бути або інший стовпець таблиці (**column\_name\_2**) або константа, задана у вигляді літералу безпосередньо у команді (**“value”**).  
 Якщо вказане ключове слово **ORDER\_BY**, то рядки повинні повернутися впорядкованими за значенням вказаних у команді стовпців. Після імені стовпця може бути опціонально вказано порядок сортування за даним стовпцем - **ASC** означає прямий лексикографічний порядок, **DESC** - зворотний лексикографічний порядок. У випадку відсутності явно вказаного порядку, вважається, що порядок є прямим (**ASC**). Якщо два рядки є рівними за значенням крайнього лівого (серед перелічених в **ORDER\_BY**) стовпця, то вони порівнюються за значенням наступного і так далі. Якщо два рядки є рівними за значеннями всіх вказаних стовпців, то вони можуть повернутись у довільному, залежному від реалізації, порядку.  
 Якщо в команді використовуються і **WHERE**, і **ORDER\_BY**, то вони повинні йти саме в такому порядку (**WHERE** перед **ORDER\_BY**).  
 В якості результату команда повинна повертати сформовану вибірку, відформатовану у вигляді таблички, в якій порядок стовпців і рядків відповідають описаним вище умовам.

**Приклади команд:  
SELECT FROM cats;  
SELECT FROM cats WHERE name > “Murzik”;  
SELECT FROM cats ORDER\_BY name ASC, id DESC;  
  
Приклад форматування таблиці:**

+----+--------+----------------+

| id | name | favourite\_food |

+----+--------+----------------+

| 1 | Murzik | Sausages |

| 2 | Pushok | Fish |

+----+--------+----------------+

### **Варіант №2**: Реляційна модель із підтримкою агрегації даних (рядки)

Даний варіант передбачає, що, для спрощення реалізації, всі стовпці у створюваних таблицях мають рядковий тип даних.

Перелік команд, які необхідно реалізувати:

1. **CREATE table\_name (column\_name \*[INDEXED] [, ...]);** - створення таблиці **table\_name**, що містить стовпці, визначення яких наведено в круглих дужках. Визначення стовпця складається щонайменше з його імені. Додаткова частина (**\***) передбачає, що у визначенні стовпця ще може бути присутнє опціональне ключове слово **INDEXED**, яке означає необхідність індексування даного стовпця. Якщо ключове слово **INDEXED** не наведено, індекс створювати не потрібно. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Table table\_name has been created”**.  
     
   **Приклади:**  
   **CREATE students (name, group);**  
   **CREATE cats (id INDEXED, name INDEXED, favourite\_food);**
2. **INSERT [INTO] table\_name (“value” [, ...]);** - додавання нового рядка з даними до таблиці **table\_name**. Кількість розділених комою значень в круглих дужках повинна дорівнювати кількості стовпців у таблиці. Порядок значень повинен відповідати тому порядку стовпців, який використовувався при створенні таблиці командою **CREATE**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“1 row has been inserted into table\_name.”**.  
     
   **Приклади:**

**INSERT INTO cats (“1”, “Murzik”, “Sausages”);  
INSERT cats (“2”, “Pushok”, “Fish”);**.

1. Формування вибірки із таблиці **table\_name**: **SELECT [agg\_function(agg\_column) [, ... ]]   
    FROM table\_name  
    [WHERE condition]  
    [GROUP\_BY column\_name [, ...] ];**  
    Якщо в команді не використовується умова **GROUP\_BY**, то після ключового слова **SELECT** не повинно бути вказано нічого. В цьому випадку вибірка повинна включати всі стовпці таблиці, причому в порядку, в якому вони були згадані в команді **CREATE** при створенні таблиці.

Якщо в команді вказана умова **GROUP\_BY gr\_col\_1, ..., gr\_col\_n**, то це означає необхідність логічного групування рядків для подальшої агрегації даних в рамках сформованих груп і формування результуючої вибірки. В одну логічну групу потрапляють рядки, що мають однакові значення стовпців **gr\_col\_1**, **gr\_col\_2**, ..., **gr\_col\_n**, незалежно від значень інших стовпців. В такому випадку, після **SELECT** може бути вказано перелік агрегуючих функцій, які необхідно застосувати до стовпців таблиці в межах утворених логічних груп: **agg\_fun\_1(agg\_col\_1), ..., agg\_fun\_k(agg\_col\_k)**. Кожна агрегуюча функція може бути використана довільну кількість разів із довільними стовпцями таблиці. У фінальній вибірці кожна логічна група представляється рівно 1 рядком, який складається із:

* **gr\_col\_1**, **gr\_col\_2**, ..., **gr\_col\_n** - стовпців, вказаних після ключового слова **GROUP\_BY**, в точно такому ж порядку.
* Результату застосування агрегуючих функцій **agg\_fun\_1**, …, **agg\_fun\_k** до стовпців **agg\_col\_1**, …, **agg\_col\_1** відповідно, якщо вони вказані після ключового слова **SELECT**.

Перелік агрегуючих функцій, які потрібно підтримати:

* + **COUNT(column)** - кількість значень в групі, незалежно від того, до якого стовпця застосовується.
  + **MAX(column)** - максимальне в лексикографічному порядку значення стовпця **column** в межах групи.
  + **LONGEST(column)** - значення стовпця **column**, що має найбільшу довжину в межах групи.

Якщо вказано ключове слово **WHERE** та слідуюча за ним умова, то команда повинна включити до вибірки (або брати для групування, якщо вказана умова **GROUP\_BY**) тільки рядки, які задовольняють дану умову. В іншому випадку, необхідно включити всі рядки із таблиці. Умова **condition** має наступну структуру:   
 **condition := column\_name\_1 < (column\_name\_2 | “value”)**.  
Тобто умова складається із однієї бінарної операції порівняння **<**, зліва від якої повинен бути вказаний стовпець таблиці (**column\_name\_1**), а справа може бути або інший стовпець таблиці (**column\_name\_2**) або константа, задана у вигляді літералу безпосередньо у команді (**“value”**).  
 Якщо в команді використовуються і **WHERE**, і **GROUP\_BY**, то вони повинні йти саме в такому порядку (**WHERE** перед **GROUP\_BY**).  
 В якості результату команда повинна повертати сформовану вибірку, відформатовану у вигляді таблички, в якій порядок стовпців відповідає описаним вище умовам, а порядок рядків може бути довільним.

**Приклади команд:  
SELECT FROM cats;  
SELECT FROM cats WHERE name < “Murzik”;**

**SELECT COUNT(id), MAX(favourite\_food)  
 FROM cats**

**GROUP\_BY name, id;**

**SELECT LONGEST(id)  
 FROM cats  
 WHERE name < “Murzik”  
 GROUP\_BY favourite\_food;  
  
Приклад форматування таблиці:**

+--------+----+-----------+---------------------+

| name | id | COUNT(id) | MAX(favourite\_food) |

+--------+----+-----------+---------------------+

| Murzik | 1 | 1 | Sausages |

| Pushok | 2 | 1 | Fish |

+--------+----+-----------+---------------------+

### **Варіант №3**: Реляційна модель із підтримкою агрегації даних (числа)

Даний варіант передбачає, що, для спрощення реалізації, всі стовпці у створюваних таблицях мають числовий тип даних.

Перелік команд, які необхідно реалізувати:

1. **CREATE table\_name (column\_name \*[INDEXED] [, ...]);** - створення таблиці **table\_name**, що містить стовпці, визначення яких наведено в круглих дужках. Визначення стовпця складається щонайменше з його імені. Додаткова частина (**\***) передбачає, що у визначенні стовпця ще може бути присутнє опціональне ключове слово **INDEXED**, яке означає необхідність індексування даного стовпця. Якщо ключове слово **INDEXED** не наведено, індекс створювати не потрібно. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Table table\_name has been created”**.  
     
   **Приклади:**  
   **CREATE students (age, average\_grade);**  
   **CREATE measurements (id INDEXED, height INDEXED, weight);**
2. **INSERT [INTO] table\_name (N [, ...]);** - додавання нового рядка з даними до таблиці **table\_name**. Кількість розділених комою значень в круглих дужках повинна дорівнювати кількості стовпців у таблиці. Порядок значень повинен відповідати тому порядку стовпців, який використовувався при створенні таблиці командою **CREATE**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“1 row has been inserted into table\_name.”**.  
     
   **Приклади:**

**INSERT INTO measurements (1, 180, 75);  
INSERT measurements (2, 175, 72);**.

1. Формування вибірки із таблиці **table\_name**: **SELECT [agg\_function(agg\_column) [, ... ]]   
    FROM table\_name  
    [WHERE condition]  
    [GROUP\_BY column\_name [, ...] ];**  
    Якщо в команді не використовується умова **GROUP\_BY**, то після ключового слова **SELECT** не повинно бути вказано нічого. В цьому випадку вибірка повинна включати всі стовпці таблиці, причому в порядку, в якому вони були згадані в команді **CREATE** при створенні таблиці.

Якщо в команді вказана умова **GROUP\_BY gr\_col\_1, ..., gr\_col\_n**, то це означає необхідність логічного групування рядків для подальшої агрегації даних в рамках сформованих груп і формування результуючої вибірки. В одну логічну групу потрапляють рядки, що мають однакові значення стовпців **gr\_col\_1**, **gr\_col\_2**, ..., **gr\_col\_n**, незалежно від значень інших стовпців. В такому випадку, після **SELECT** може бути вказано перелік агрегуючих функцій, які необхідно застосувати до стовпців таблиці в межах утворених логічних груп: **agg\_fun\_1(agg\_col\_1), ..., agg\_fun\_k(agg\_col\_k)**. Кожна агрегуюча функція може бути використана довільну кількість разів із довільними стовпцями таблиці. У фінальній вибірці кожна логічна група представляється рівно 1 рядком, який складається із:

* **gr\_col\_1**, **gr\_col\_2**, ..., **gr\_col\_n** - стовпців, вказаних після ключового слова **GROUP\_BY**, в точно такому ж порядку.
* Результату застосування агрегуючих функцій **agg\_fun\_1**, …, **agg\_fun\_k** до стовпців **agg\_col\_1**, …, **agg\_col\_1** відповідно, якщо вони вказані після ключового слова **SELECT**.

Перелік агрегуючих функцій, які потрібно підтримати:

* + **COUNT(column)** - кількість значень в групі, незалежно від того, до якого стовпця застосовується.
  + **MAX(column)** - максимальне значення стовпця **column** в межах групи.
  + **AVG(column)** - середнє значення стовпця **column** в межах групи.

Якщо вказано ключове слово **WHERE** та слідуюча за ним умова, то команда повинна включити до вибірки (або брати для групування, якщо вказана умова **GROUP\_BY**) тільки рядки, які задовольняють дану умову. В іншому випадку, необхідно включити всі рядки із таблиці. Умова **condition** має наступну структуру:   
 **condition := column\_name\_1 = (column\_name\_2 | N)**.  
Тобто умова складається із однієї бінарної операції порівняння **=**, зліва від якої повинен бути вказаний стовпець таблиці (**column\_name\_1**), а справа може бути або інший стовпець таблиці (**column\_name\_2**) або константа, задана у вигляді літералу безпосередньо у команді (**N**).  
 Якщо в команді використовуються і **WHERE**, і **GROUP\_BY**, то вони повинні йти саме в такому порядку (**WHERE** перед **GROUP\_BY**).  
 В якості результату команда повинна повертати сформовану вибірку, відформатовану у вигляді таблички, в якій порядок стовпців відповідає описаним вище умовам, а порядок рядків може бути довільним.

**Приклади команд:  
SELECT FROM measurements;  
SELECT FROM measurements WHERE id = 1;**

**SELECT COUNT(id), AVG(weight)  
 FROM measurements**

**GROUP\_BY height;**

**SELECT COUNT(id)  
 FROM measurements  
 WHERE height = 170  
 GROUP\_BY weight;  
  
Приклад форматування таблиці:**

+--------+-----------+-------------+

| height | COUNT(id) | AVG(weight) |

+--------+-----------+-------------+

| 180 | 1 | 75 |

| 175 | 1 | 72 |

+--------+-----------+-------------+

### **Варіант №4**: Реляційна модель із підтримкою операції JOIN

Даний варіант передбачає, що, для спрощення реалізації, всі стовпці у створюваних таблицях мають рядковий тип даних.

Перелік команд, які необхідно реалізувати:

1. **CREATE table\_name (column\_name \*[INDEXED] [, ...]);** - створення таблиці **table\_name**, що містить стовпці, визначення яких наведено в круглих дужках. Визначення стовпця складається щонайменше з його імені. Додаткова частина (**\***) передбачає, що у визначенні стовпця ще може бути присутнє опціональне ключове слово **INDEXED**, яке означає необхідність індексування даного стовпця. Якщо ключове слово **INDEXED** не наведено, індекс створювати не потрібно. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Table table\_name has been created”**.  
     
   **Приклади:**  
   **CREATE students (name, group);**  
   **CREATE owners (owner\_id INDEXED, owner\_name, owner\_age);  
   CREATE cats (cat\_id INDEXED, cat\_owner\_id INDEXED, cat\_name);**
2. **INSERT [INTO] table\_name (“value” [, ...]);** - додавання нового рядка з даними до таблиці **table\_name**. Кількість розділених комою значень в круглих дужках повинна дорівнювати кількості стовпців у таблиці. Порядок значень повинен відповідати тому порядку стовпців, який використовувався при створенні таблиці командою **CREATE**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“1 row has been inserted into table\_name.”**.  
     
   **Приклади:**

**INSERT INTO owners(“1”, “Vasya”, “30”);**  
**INSERT cats (“10”, “1”, “Murzik”);  
INSERT cats (“20”, “1”, “Pushok”);**

1. Формування вибірки із таблиці **table\_name**: **SELECT FROM table\_name\_1  
    [JOIN table\_name\_2 [ON t1\_column = t2\_column]]  
    [WHERE condition];**  
    Якщо в команді не використовується ключове слово **JOIN**, то вибірка повинна включати всі стовпці таблиці, причому в порядку, в якому вони були згадані в команді **CREATE** при створенні таблиці.

Якщо вказано ключове слово **JOIN**, то в якості результату необхідно повернути таблицю, рядки якої є комбінацією рядків із двох таблиць - **table\_name\_1** та **table\_name\_2**. В цьому випадку вибірка повинна включати всі стовпці обох таблиць - спочатку стовпці таблиці **table\_name\_1**, а потім **table\_name\_2**, причому в порядку, в якому вони були згадані в команді **CREATE** при створенні таблиць. Якщо вказано опціональну умову **ON t1\_column = t2\_column**, то комбінуються тільки пари рядків, в яких значення стовпця **t1\_column** (із таблиці **table\_name\_1**) дорівнює значенню стовпця **t2\_column** (із таблиці **table\_name\_2)**. В іншому випадку, необхідно побудувати декартів добуток двох таблиць, тобто скомбінувати кожен рядок таблиці **table\_name\_1** із кожним рядком таблиці **table\_name\_2**.

Якщо вказано ключове слово **WHERE** та слідуюча за ним умова, то команда повинна включити до вибірки тільки ті рядки, які задовольняють дану умову. В іншому випадку, необхідно повернути всі рядки із вказаної таблиці. Умова **condition** має наступну структуру:   
 **condition := column\_name\_1 > (column\_name\_2 | “value”)**.  
Тобто умова складається із однієї бінарної операції порівняння **>**, зліва від якої повинен бути вказаний стовпець таблиці (**column\_name\_1**), а справа може бути або інший стовпець таблиці (**column\_name\_2**) або константа, задана у вигляді літералу безпосередньо у команді (**“value”**). Варто зауважити, що, якщо в команді використовується ключове слово **JOIN**, то в якості **column\_name\_1** (та **column\_name\_2**) може виступати стовпець із будь-якої з двох таблиць.  
 Якщо в команді використовуються і **JOIN**, і **WHERE**, то вони повинні йти саме в такому порядку (**JOIN** перед **WHERE**).  
 В якості результату команда повинна повертати сформовану вибірку, відформатовану у вигляді таблички, в якій порядок стовпців і рядків відповідають описаним вище умовам.

**Приклади команд:  
SELECT FROM cats;  
SELECT FROM cats   
 WHERE name > “Murzik”;  
SELECT FROM cats  
 JOIN owners ON cat\_owner\_id = owner\_id;  
  
Приклад форматування таблиці:**

+---------+---------------+-----------+-----------+-------------+------------+

| cat\_id | cat\_owner\_id | cat\_name | owner\_id | owner\_name | owner\_age |

+---------+---------------+-----------+-----------+-------------+------------+

| 10 | 1 | Murzik | 1 | Vasya | 30 |

| 20 | 1 | Pushok | 1 | Vasya | 30 |

+---------+---------------+-----------+-----------+-------------+------------+

### **Варіант №5**: Реляційна модель із підтримкою операції LEFT\_JOIN

Даний варіант передбачає, що, для спрощення реалізації, всі стовпці у створюваних таблицях мають рядковий тип даних.

Перелік команд, які необхідно реалізувати:

1. **CREATE table\_name (column\_name \*[INDEXED] [, ...]);** - створення таблиці **table\_name**, що містить стовпці, визначення яких наведено в круглих дужках. Визначення стовпця складається щонайменше з його імені. Додаткова частина (**\***) передбачає, що у визначенні стовпця ще може бути присутнє опціональне ключове слово **INDEXED**, яке означає необхідність індексування даного стовпця. Якщо ключове слово **INDEXED** не наведено, індекс створювати не потрібно. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Table table\_name has been created”**.  
     
   **Приклади:**  
   **CREATE students (name, group);**  
   **CREATE owners (owner\_id INDEXED, owner\_name, owner\_age);  
   CREATE cats (cat\_id INDEXED, cat\_owner\_id INDEXED, cat\_name);**
2. **INSERT [INTO] table\_name (“value” [, ...]);** - додавання нового рядка з даними до таблиці **table\_name**. Кількість розділених комою значень в круглих дужках повинна дорівнювати кількості стовпців у таблиці. Порядок значень повинен відповідати тому порядку стовпців, який використовувався при створенні таблиці командою **CREATE**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“1 row has been inserted into table\_name.”**.  
     
   **Приклади:**

**INSERT INTO owners(“1”, “Vasya”, “30”);**

**INSERT INTO owners(“2”, “Kolya”, “28”);**.  
**INSERT cats (“10”, “1”, “Murzik”);  
INSERT cats (“20”, “1”, “Pushok”);**

1. Формування вибірки із таблиці **table\_name**: **SELECT FROM table\_name\_1  
    [LEFT\_JOIN table\_name\_2 ON t1\_column = t2\_column]  
    [WHERE condition];**  
    Якщо в команді не використовується ключове слово **LFET\_JOIN**, то вибірка повинна включати всі стовпці таблиці, причому в порядку, в якому вони були згадані в команді **CREATE** при створенні таблиці.

Якщо вказано ключове слово **LEFT\_JOIN**, то в якості результату необхідно повернути таблицю, рядки якої є комбінацією рядків із двох таблиць - **table\_name\_1** та **table\_name\_2**. В цьому випадку вибірка повинна включати всі стовпці обох таблиць - спочатку стовпці таблиці **table\_name\_1**, а потім **table\_name\_2**, причому в порядку, в якому вони були згадані в команді **CREATE** при створенні таблиць. Вибірка повинна включати:

* + Комбінації рядків із таблиць **table\_name\_1** та **table\_name\_2**, які задовольняють умові **t1\_column = t2\_column**, тобто в яких значення стовпця **t1\_column** (із таблиці **table\_name\_1**) дорівнює значенню стовпця **t2\_column** (із таблиці **table\_name\_2)**.
  + Рядки із таблиці **table\_name\_1**, для яких за умовою **t1\_column = t2\_column** не знайшлось жодного співпадіння в таблиці **table\_name\_2**. Значення стовпців таблиці **table\_name\_2** в даному випадку потрібно замінити порожнім рядком **“”**.

Якщо вказано ключове слово **WHERE** та слідуюча за ним умова, то команда повинна включити до вибірки тільки ті рядки, які задовольняють дану умову. В іншому випадку, необхідно повернути всі рядки із вказаної таблиці. Умова **condition** має наступну структуру:   
 **condition := column\_name\_1 < (column\_name\_2 | “value”)**.  
Тобто умова складається із однієї бінарної операції порівняння **<**, зліва від якої повинен бути вказаний стовпець таблиці (**column\_name\_1**), а справа може бути або інший стовпець таблиці (**column\_name\_2**) або константа, задана у вигляді літералу безпосередньо у команді (**“value”**). Варто зауважити, що, якщо в команді використовується ключове слово **LEFT\_JOIN**, то в якості **column\_name\_1** (та **column\_name\_2**) може виступати стовпець із будь-якої з двох таблиць.  
 Якщо в команді використовуються і **LEFT\_JOIN**, і **WHERE**, то вони повинні йти саме в такому порядку (**LEFT\_JOIN** перед **WHERE**). Крім того, умова, вказана після **WHERE**, повинна застосовуватись після комбінації таблиць, а не до неї. В протилежному випадку буде отримано некоректні результати.  
 В якості результату команда повинна повертати сформовану вибірку, відформатовану у вигляді таблички, в якій порядок стовпців і рядків відповідають описаним вище умовам.

**Приклади команд:  
SELECT FROM cats;  
SELECT FROM cats   
 WHERE name < “Pushok”;  
SELECT FROM owners  
 LEFT\_JOIN cats ON owner\_id = cat\_owner\_id;  
  
Приклад форматування таблиці:**

+-----------+-------------+------------+---------+---------------+-----------+

| owner\_id | owner\_name | owner\_age | cat\_id | cat\_owner\_id | cat\_name |

+-----------+-------------+------------+---------+---------------+-----------+

| 1 | Vasya | 30 | 10 | 1 | Murzik |

| 1 | Vasya | 30 | 20 | 1 | Pushok |

| 2 | Kolya | 28 | | | |

+-----------+-------------+------------+---------+---------------+-----------+

### **Варіант №6**: Реляційна модель із підтримкою операції FULL\_JOIN

Даний варіант передбачає, що, для спрощення реалізації, всі стовпці у створюваних таблицях мають рядковий тип даних.

Перелік команд, які необхідно реалізувати:

1. **CREATE table\_name (column\_name \*[INDEXED] [, ...]);** - створення таблиці **table\_name**, що містить стовпці, визначення яких наведено в круглих дужках. Визначення стовпця складається щонайменше з його імені. Додаткова частина (**\***) передбачає, що у визначенні стовпця ще може бути присутнє опціональне ключове слово **INDEXED**, яке означає необхідність індексування даного стовпця. Якщо ключове слово **INDEXED** не наведено, індекс створювати не потрібно. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Table table\_name has been created”**.  
     
   **Приклади:**  
   **CREATE students (name, group);**  
   **CREATE owners (owner\_id INDEXED, owner\_name, owner\_age);  
   CREATE cats (cat\_id INDEXED, cat\_owner\_id INDEXED, cat\_name);**
2. **INSERT [INTO] table\_name (“value” [, ...]);** - додавання нового рядка з даними до таблиці **table\_name**. Кількість розділених комою значень в круглих дужках повинна дорівнювати кількості стовпців у таблиці. Порядок значень повинен відповідати тому порядку стовпців, який використовувався при створенні таблиці командою **CREATE**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“1 row has been inserted into table\_name.”**.  
     
   **Приклади:**

**INSERT INTO owners(“1”, “Vasya”, “30”);**

**INSERT INTO owners(“2”, “Kolya”, “28”);**  
**INSERT cats (“10”, “1”, “Murzik”);  
INSERT cats (“20”, “1”, “Pushok”);**

**INSERT cats (“30”, “3”, “Tom”);**

1. Формування вибірки із таблиці **table\_name**: **SELECT FROM table\_name\_1  
    [FULL\_JOIN table\_name\_2 ON t1\_column = t2\_column]  
    [WHERE condition];**  
    Якщо в команді не використовується ключове слово **FULL\_JOIN**, то вибірка повинна включати всі стовпці таблиці, причому в порядку, в якому вони були згадані в команді **CREATE** при створенні таблиці.

Якщо вказано ключове слово **FULL\_JOIN**, то в якості результату необхідно повернути таблицю, рядки якої є комбінацією рядків із двох таблиць - **table\_name\_1** та **table\_name\_2**. В цьому випадку вибірка повинна включати всі стовпці обох таблиць - спочатку стовпці таблиці **table\_name\_1**, а потім **table\_name\_2**, причому в порядку, в якому вони були згадані в команді **CREATE** при створенні таблиць. Вибірка повинна включати:

* + Комбінації рядків із таблиць **table\_name\_1** та **table\_name\_2**, які задовольняють умові **t1\_column = t2\_column**, тобто в яких значення стовпця **t1\_column** (із таблиці **table\_name\_1**) дорівнює значенню стовпця **t2\_column** (із таблиці **table\_name\_2)**.
  + Рядки із таблиці **table\_name\_1**, для яких за умовою **t1\_column = t2\_column** не знайшлось жодного співпадіння в таблиці **table\_name\_2**. Значення стовпців таблиці **table\_name\_2** в даному випадку потрібно замінити порожнім рядком **“”**.
  + Рядки із таблиці **table\_name\_2**, для яких за умовою **t1\_column = t2\_column** не знайшлось жодного співпадіння в таблиці **table\_name\_1**. Значення стовпців таблиці **table\_name\_1** в даному випадку потрібно замінити порожнім рядком **“”**.

Якщо вказано ключове слово **WHERE** та слідуюча за ним умова, то команда повинна включити до вибірки тільки ті рядки, які задовольняють дану умову. В іншому випадку, необхідно повернути всі рядки із вказаної таблиці. Умова **condition** має наступну структуру:   
 **condition := column\_name\_1 = (column\_name\_2 | “value”)**.  
Тобто умова складається із однієї бінарної операції порівняння **=**, зліва від якої повинен бути вказаний стовпець таблиці (**column\_name\_1**), а справа може бути або інший стовпець таблиці (**column\_name\_2**) або константа, задана у вигляді літералу безпосередньо у команді (**“value”**). Варто зауважити, що, якщо в команді використовується ключове слово **FULL\_JOIN**, то в якості **column\_name\_1** (та **column\_name\_2**) може виступати стовпець із будь-якої з двох таблиць.  
 Якщо в команді використовуються і **FULL\_JOIN**, і **WHERE**, то вони повинні йти саме в такому порядку (**FULL\_JOIN** перед **WHERE**). Крім того, умова, вказана після **WHERE**, повинна застосовуватись після комбінації таблиць, а не до неї. В протилежному випадку буде отримано некоректні результати.  
 В якості результату команда повинна повертати сформовану вибірку, відформатовану у вигляді таблички, в якій порядок стовпців і рядків відповідають описаним вище умовам.

**Приклади команд:  
SELECT FROM cats;  
SELECT FROM cats   
 WHERE name = “Murzik”;  
SELECT FROM owners  
 FULL\_JOIN cats ON owner\_id = cat\_owner\_id;  
  
Приклад форматування таблиці:**

+-----------+-------------+------------+---------+---------------+-----------+

| owner\_id | owner\_name | owner\_age | cat\_id | cat\_owner\_id | cat\_name |

+-----------+-------------+------------+---------+---------------+-----------+

| 1 | Vasya | 30 | 10 | 1 | Murzik |

| 1 | Vasya | 30 | 20 | 1 | Pushok |

| 2 | Kolya | 28 | | | |

| | | | 30 | 3 | Tom |

+-----------+-------------+------------+---------+---------------+-----------+

### **Варіант №7**: Реляційна модель із фільтрацією довільної складності

Даний варіант передбачає, що, для спрощення реалізації, всі стовпці у створюваних таблицях мають рядковий тип даних.

Перелік команд, які необхідно реалізувати:

1. **CREATE table\_name (column\_name [INDEXED] [, ...]);** - створення таблиці **table\_name**, що містить стовпці, визначення яких наведено в круглих дужках. Визначення стовпця складається щонайменше з його імені. Додаткова частина (**\***) передбачає, що у визначенні стовпця ще може бути присутнє опціональне ключове слово **INDEXED**, яке означає необхідність індексування даного стовпця. Якщо ключове слово **INDEXED** не наведено, індекс створювати не потрібно. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Table table\_name has been created”**.  
     
   **Приклад:**  
   **CREATE students (name, group);**  
   **CREATE cats (id INDEXED, name INDEXED, favourite\_food);**
2. **INSERT [INTO] table\_name (“value” [, ...]);** - додавання нового рядка з даними до таблиці **table\_name**. Кількість розділених комою значень в круглих дужках повинна дорівнювати кількості стовпців у таблиці. Порядок значень повинен відповідати тому порядку стовпців, який використовувався при створенні таблиці командою **CREATE**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“1 row has been inserted into table\_name.”**.  
     
   **Приклади:**

**INSERT INTO cats (“1”, “Murzik”, “Sausages”);  
INSERT cats (“2”, “Pushok”, “Fish”);**.

1. Формування вибірки із таблиці **table\_name**: **SELECT FROM table\_name  
    [WHERE condition];**

Вибірка повинна включати всі стовпці таблиці, причому в порядку, в якому вони були згадані в команді **CREATE** при створенні таблиці.

Якщо вказано ключове слово **WHERE** та слідуюча за ним умова, то команда повинна включити до вибірки тільки ті рядки, які задовольняють дану умову. В іншому випадку, необхідно повернути всі рядки із вказаної таблиці. Рекурсивне визначення умови **condition** має наступний вигляд:   
 **condition := column\_name operator “value”**

**| (condition) AND (condition)**

**| (condition) OR (condition)**  
 **operator := ( = | < )**.  
Тобто, в найпростішому випадку умова складається із однієї бінарної операції порівняння (**=** або **<**), зліва від якої повинен бути вказаний стовпець таблиці (**column\_name**), а справа константа, задана у вигляді літералу безпосередньо в команді (**“value”**). Крім того, умова може бути комбінацією двох інших умов, взятих в круглі дужки і розділених одним із двох операторів - **AND** або **OR**. Дужки необхідні для спрощення синтаксичного аналізу команд та уникнення необхідності введення пріоритетів між операторами **AND** або **OR**. Приклади коректно сформованих умов (для уявної таблички, яка містить стовпець **name**): **name = “Vasya”**, **((name < “C”) OR (name = “X”)) OR (name = “Petya”)**, **(name < “Dima”) OR ((name = “Kolya”) AND (name = “Vasya”))** тощо.  
При виконанні додаткової частини (**\***), для простоти, можна перевіряти наявність підходящого індексу для кожної умови виду **column\_name operator “value”** незалежно від інших, навіть якщо одна і та ж колонка **column\_name** зустрічається декілька разів.

В якості результату команда повинна повертати сформовану вибірку, відформатовану у вигляді таблички, в якій порядок стовпців відповідає описаним вище умовам. Порядок рядків може бути довільним.

**Приклади команд:  
SELECT FROM cats;  
SELECT FROM cats   
 WHERE (name < “Murzik”) OR (name = “Pushok”);  
  
Приклад форматування таблиці:**

+----+--------+----------------+

| id | name | favourite\_food |

+----+--------+----------------+

| 1 | Murzik | Sausages |

| 2 | Pushok | Fish |

+----+--------+----------------+

### **Варіант №8**: Колекція текстових документів із можливістю повнотекстового пошуку (1)

В даному варіанті передбачається, що по створеним колекціям будуються інвертовані індекси (inverted index), які і використовуються для прискорення виконання пошукових команд (**\***). Для побудови індексу програма повинна аналізувати документи, які додаються до колекцій, виділяти з них окремі слова та зберігати список позицій, де дане слово зустрічається в межах даного документу. Для простоти, в межах даного практикуму вважаємо, що “словом” в документі вважається довільна послідовність, яка складається із алфавітно-цифрових символів та символу **\_** (в термінах регулярних виразів - **[a-zA-Z0-9\_]+**). Решта символів, які зустрічаються в документах визначають межі слів та можуть ігноруватись.

Перелік команд, які необхідно реалізувати:

1. **CREATE collection\_name;** - створення нової колекції з назвою **collection\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Collection collection\_name has been created”**.  
     
   **Приклад:**  
   **CREATE wiki\_articles;**
2. **INSERT collection\_name “value”;** - додавання нового документу до колекції **collection\_name**. В якості результату команда повертати певне повідомлення для користувача, наприклад **“Document has been added to collection\_name”**.  
     
   **Приклад:**  
   **INSERT wiki\_articles “The word 'algorithm' has its roots in Latinizing the ... ”;**
3. **PRINT\_INDEX collection\_name;** - виведення на екран внутрішньої структури інвертованого індексу, побудованого для колекції **collection\_name**. Вивід повинен включати виділені ключові слова, документи, яким вони належать та позиції, в яких вони зустрічаються в даних документах.  
     
   **Приклад:**  
   Для індексу, що містить документи **“to be or not to be”** та **“to go or not to go”**, яким було присвоєно ідентифікатори **d1** та **d2** відповідно, вивід повинен бути схожим до наступного:  
     
   **"be":**

**d1 -> [2, 6]**

**"go":**

**d2 -> [2, 6]**

**"not":**

**d1 -> [4]**

**d2 -> [4]**

**"or":**

**d1 -> [3]**

**d2 -> [3]**

**"to":**

**d1 -> [1, 5]**

**d2 -> [1, 5]**

1. **SEARCH collection\_name [WHERE query];** - пошук документів по колекції **collection\_name**.

Якщо вказано ключове слово **WHERE** та слідуючий за ним пошуковий запит, то команда повинна включити до вибірки тільки ті документи, які задовольняють даний запит. В іншому випадку, необхідно повернути всі документи зі вказаної колекції. Визначення запиту **query** має наступний вигляд:   
 **query := “keyword”   
 \* | “prefix”\*  
 \* | “keyword\_1” <N> “keyword\_2”**   
Пояснення:

* + **“keyword”** - до вибірки повинні бути включені документи, які містять слово **“keyword”**.
  + **“prefix”\*** - префіксний пошук, тобто до вибірки повинні бути включені документи, які містять слово, що починається із **“prefix”**. Наприклад, якщо **query = “універ”\***, то у вибірку повинні бути включені всі документи, що містять слова **“університет”**, **“універсам”**, **“універсал”** тощо. Відноситься до додаткової частини завдання (**\***).
  + **“keyword\_1” <N> “keyword\_2”** - до вибірки повинні бути включені документи, які містять слова **“keyword\_1”** та **“keyword\_1”** на відстані **N** слів одне від одного. Позиція в документі та взаємний порядок слів не важливі. Наприклад, документ **“The word 'algorithm' has its roots in Latinizing the nisba ...”** задовольняє таким запитам: **“word” <1> “algorithm”**, **“algorithm” <1> “word”**, **“word” <4> “roots”**, **“roots” <4> “word”** тощо. Відноситься до додаткової частини завдання (**\***).

Можна вважати, що **“keyword”**, **“prefix”**, **“keyword\_1”** та **“keyword\_2”** вище є коректним словами, відповідно до визначення даного на початку даного варіанту, тобто задовольняють регулярному виразу **[a-zA-Z0-9\_]+**.  
 Пошук документів, що задовольняють певному запиту повинен проводитись без урахування регістру документів та ключових слів у цих запитах. Тобто, наприклад, документ **“The word 'algorithm' has its roots in Latinizing the nisba ...”** задовольняє усім наступним пошуковим запитам: **“algorithm”**, **“ALGORithm”**, **“ALGo”\*** , **“WORD” <2> “has”** тощо.  
 В якості результату команда повинна повертати список знайдених документів. Порядок рядків може бути довільним.  
  
**Приклади команд:  
SEARCH wiki\_articles;  
SEARCH wiki\_articles WHERE “algorithm”;**

**SEARCH wiki\_articles WHERE “latin”\*;  
SEARCH wiki\_articles WHERE “has” <2> “roots”;**

### **Варіант №9**: Колекція текстових документів із можливістю повнотекстового пошуку (2)

В даному варіанті передбачається, що по створеним колекціям будуються інвертовані індекси (inverted index), які і використовуються для прискорення виконання пошукових команд. Для побудови індексу програма повинна аналізувати документи, які додаються до колекцій, виділяти з них окремі слова та зберігати список позицій, де дане слово зустрічається в межах даного документу. Для простоти, в межах даного практикуму вважаємо, що “словом” в документі вважається довільна послідовність, яка складається із алфавітно-цифрових символів та символу **\_** (в термінах регулярних виразів - **[a-zA-Z0-9\_]+**). Решта символів, які зустрічаються в документах визначають межі слів та можуть ігноруватись.

Перелік команд, які необхідно реалізувати:

1. **CREATE collection\_name;** - створення нової колекції з назвою **collection\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Collection collection\_name has been created”**.  
     
   **Приклад:**  
   **CREATE wiki\_articles;**
2. **INSERT collection\_name “value”;** - додавання нового документу до колекції **collection\_name**. В якості результату команда повертати певне повідомлення для користувача, наприклад **“Document has been added to collection\_name”**.  
     
   **Приклад:**  
   **INSERT wiki\_articles “The word 'algorithm' has its roots in Latinizing the ... ”;**
3. **PRINT\_INDEX collection\_name;** - виведення на екран внутрішньої структури інвертованого індексу, побудованого для колекції **collection\_name**. Вивід повинен включати виділені ключові слова, документи, яким вони належать та позиції, в яких вони зустрічаються в даних документах.  
     
   **Приклад:**  
   Для індексу, що містить документи **“to be or not to be”** та **“to go or not to go”**, яким було присвоєно ідентифікатори **d1** та **d2** відповідно, вивід повинен бути схожим до наступного:  
     
   **"be":**

**d1 -> [2, 6]**

**"go":**

**d2 -> [2, 6]**

**"not":**

**d1 -> [4]**

**d2 -> [4]**

**"or":**

**d1 -> [3]**

**d2 -> [3]**

**"to":**

**d1 -> [1, 5]**

**d2 -> [1, 5]**

1. **SEARCH collection\_name [WHERE query];** - пошук документів по колекції **collection\_name**.

Якщо вказано ключове слово **WHERE** та слідуючий за ним пошуковий запит, то команда повинна включити до вибірки тільки ті документи, які задовольняють даний запит. В іншому випадку, необхідно повернути всі документи зі вказаної колекції. Визначення запиту **query** має наступний вигляд:   
 **query := “keyword”   
 \* | “keyword\_1” - “keyword\_1”  
 \* | “keyword\_1” <N> “keyword\_2”**   
Пояснення:

* + **“keyword”** - до вибірки повинні бути включені документи, які містять слово **“keyword”**.
  + **“keyword\_1” - “keyword\_2”** - до вибірки повинні бути включені документи, які містять будь-яке слово **“keyword”**, що задовольняє умові **“keyword\_1” <= “keyword” <= “keyword\_2”**. Відноситься до додаткової частини завдання (**\***).
  + **“keyword\_1” <N> “keyword\_2”** - до вибірки повинні бути включені документи, які містять слова **“keyword\_1”** та **“keyword\_1”** на відстані **N** слів одне від одного. Позиція в документі та взаємний порядок слів не важливі. Наприклад, документ **“The word 'algorithm' has its roots in Latinizing the nisba ...”** задовольняє таким запитам: **“word” <1> “algorithm”**, **“algorithm” <1> “word”**, **“word” <4> “roots”**, **“roots” <4> “word”** тощо. Відноситься до додаткової частини завдання (**\***).

Можна вважати, що **“keyword”**, **“keyword\_1”** та **“keyword\_2”** вище є коректним словами, відповідно до визначення даного на початку даного варіанту, тобто задовольняють регулярному виразу **[a-zA-Z0-9\_]+**.  
 Пошук документів, що задовольняють певному запиту повинен проводитись без урахування регістру документів та ключових слів у цих запитах. Тобто, наприклад, документ **“The word 'algorithm' has its roots in Latinizing the nisba ...”** задовольняє усім наступним пошуковим запитам: **“algorithm”**, **“ALGORithm”**, **“ALGo”\*** , **“WORD” <2> “has”** тощо.  
 В якості результату команда повинна повертати список знайдених документів. Порядок рядків може бути довільним.  
  
**Приклади команд:  
SEARCH wiki\_articles;  
SEARCH wiki\_articles WHERE “algorithm”;**

**SEARCH wiki\_articles WHERE “haa” - “haz”;  
SEARCH wiki\_articles WHERE “has” <2> “roots”;**

### Варіант №10: Множина точок у двовимірній декартовій площині (R-Tree)

В даному варіанті передбачається, що створені множини точок зберігаються у вигляді структури даних відомої як R-дерево.  
Перелік команд, які необхідно реалізувати:

1. **CREATE set\_name;** - створення нової множини з назвою **set\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Set set\_name has been created”**.  
     
   **Приклад:**  
   **CREATE cities;**
2. **INSERT set\_name (x, y);** - додавання нової точки з координатами **(x, y)** до множини **set\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Point (x, y) has been added to set\_name”**.  
     
   **Приклад:**  
   **INSERT cities (1, 2);**
3. **PRINT\_TREE set\_name;** - виведення на екран внутрішньої структури R-дерева, побудованого для множини **set\_name**. Вивід повинен включати обмежуючі прямокутники для внутрішніх вузлів та самі точки для листових вузлів. Для внутрішніх вузлів спочатку повинно виводитись ліве піддерево, а потім праве. Кожен прямокутник повинно бути виведено у форматі **[(x\_left, y\_bottom), (x\_right, y\_top)]**, де **(x\_left, y\_bottom)** - координати нижнього лівого кута, **(x\_right, y\_top)** - координати верхнього правого кута. Див. [9].  
     
   **Приклад:**  
   Для множини, що містить точки **(3, 4)**, **(2, 2)**, **(5, 3)**, **(1, 3)** та **(4, 4)** вивід повинен бути схожим до наступного (структура дерева може відрізнятись):  
     
   **[(1, 2), (5, 4)]**

**├── [(1, 2), (2, 3)]**

**│ ├── (1, 3)**

**│ └── (2, 2)**

**└── [(3, 4), (5, 4)]**

**├── (3, 4)**

**└── [(4, 4), (5, 3)]**

**├── (4, 4)**

**└── (5, 3)**

1. **CONTAINS set\_name (x, y);** - перевірка входження точки **(x, y)** до множини **set\_name**. В якості результату команда повинна повертати **TRUE**, якщо точка входить до множини та **FALSE**, якщо ні.  
     
   **Приклад:**  
   **CONTAINS cities (1, 2);**
2. **SEARCH set\_name \*[WHERE query];** - пошук точок у множині **set\_name**.

В основній частині завдання команда має форму **SEARCH set\_name;** і повинна просто повернути всі точки зі вказаної множини. Додаткова частина (**\***) передбачає підтримку ключового слова **WHERE** та слідуючого за ним пошукового запиту. В цьому випадку команда повинна включити до вибірки тільки ті точки, які задовольняють даний запит. Визначення запиту **query** має наступний вигляд:   
 **query := INSIDE (x\_left, y\_bottom), (x\_right, y\_top)  
 | LEFT\_OF x  
 | NN (x, y)**   
Пояснення:

* + **INSIDE (x\_left, y\_bottom), (x\_right, y\_top)** - до вибірки повинні бути включені точки, які містяться всередині або лежать на сторонах прямокутника, який задано координатами його вершин: **(x\_left, y\_bottom)** - координати нижнього лівого кута, **(x\_right, y\_top)** - координати верхнього правого кута.
  + **LEFT\_OF x** - до вибірки повинні бути включені точки, x-координата яких менша за **x**.
  + **NN (x, y)** - до вибірки повинна бути включена точка, що є найближчим сусідом (nearest neighbour - звідси NN) точки з координатами **(x, y)**. Якщо у множині є декілька точок з однаковою мінімальною відстанню, то потрібно повернути їх усі.

В якості результату команда повинна повертати список знайдених точок, в будь-якому порядку.  
  
**Приклади команд:  
SEARCH cities;  
SEARCH cities WHERE INSIDE (0, 0), (3, 5);**

**SEARCH cities WHERE LEFT\_OF 10;**

**SEARCH cities WHERE NN (3, 3);**

### Варіант №11: Множина точок у двовимірній декартовій площині (KD-Tree)

В даному варіанті передбачається, що створені множини точок зберігаються у вигляді структури даних відомої як KD-дерево.

Перелік команд, які необхідно реалізувати:

1. **CREATE set\_name;** - створення нової множини з назвою **set\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Set set\_name has been created”**.  
     
   **Приклад:**  
   **CREATE cities;**
2. **INSERT set\_name (x, y);** - додавання нової точки з координатами **(x, y)** до множини **set\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Point (x, y) has been added to set\_name”**.  
     
   **Приклад:**  
   **INSERT cities (1, 2);**
3. **PRINT\_TREE set\_name;** - виведення на екран внутрішньої структури KD-дерева, побудованого для множини **set\_name**. Для внутрішніх вузлів спочатку повинно виводитись ліве піддерево, а потім праве. Див. [9].  
     
   **Приклад:**  
   Для множини, що містить точки **(3, 4)**, **(2, 2)**, **(5, 3)**, **(1, 3)** та **(4, 4)** вивід повинен бути схожим до наступного (структура дерева може відрізнятись):  
     
   **(3, 4)**

**├── (2, 2)**

**│ └── (1, 3)**

**└── (5, 3)**

**└── (4, 4)**

1. **CONTAINS set\_name (x, y);** - перевірка входження точки **(x, y)** до множини **set\_name**. В якості результату команда повинна повертати **TRUE**, якщо точка входить до множини та **FALSE**, якщо ні.  
     
   **Приклад:**  
   **CONTAINS cities (1, 2);**
2. **SEARCH set\_name \*[WHERE query];** - пошук точок у множині **set\_name**.

В основній частині завдання команда має форму **SEARCH set\_name;** і повинна просто повернути всі точки зі вказаної множини. Додаткова частина (**\***) передбачає підтримку ключового слова **WHERE** та слідуючого за ним пошукового запиту. В цьому випадку команда повинна включити до вибірки тільки ті точки, які задовольняють даний запит. Визначення запиту **query** має наступний вигляд:   
 **query := INSIDE (x\_left, y\_bottom), (x\_right, y\_top)  
 | ABOVE\_TO x  
 | NN (x, y)**   
Пояснення:

* + **INSIDE (x\_left, y\_bottom), (x\_right, y\_top)** - до вибірки повинні бути включені точки, які містяться всередині або лежать на сторонах прямокутника, який задано координатами його вершин: **(x\_left, y\_bottom)** - координати нижнього лівого кута, **(x\_right, y\_top)** - координати верхнього правого кута.
  + **ABOVE\_TO y** - до вибірки повинні бути включені точки, y-координата яких більша за **y**.
  + **NN (x, y)** - до вибірки повинна бути включена точка, що є найближчим сусідом (nearest neighbour - звідси NN) точки з координатами **(x, y)**. Якщо у множині є декілька точок з однаковою мінімальною відстанню, то потрібно повернути їх усі.

В якості результату команда повинна повертати список знайдених точок, в будь-якому порядку.  
  
**Приклади команд:  
SEARCH cities;  
SEARCH cities WHERE INSIDE (0, 0), (3, 5);**

**SEARCH cities WHERE ABOVE\_TO 10;**

**SEARCH cities WHERE NN (3, 3);**

### Варіант №12: Множина відрізків на прямій (R-Tree)

В даному варіанті передбачається, що створені множини відрізків зберігаються у вигляді структури даних відомої як R-дерево.  
Перелік команд, які необхідно реалізувати:

1. **CREATE set\_name;** - створення нової множини з назвою **set\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Set set\_name has been created”**.  
     
   **Приклад:**  
   **CREATE reservations;**
2. **INSERT set\_name [l, h];** - додавання нового відрізку з нижньою границею **l** та верхньою границею **h** до множини **set\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Range [l, h] has been added to set\_name”**.  
     
   **Приклад:**  
   **INSERT reservations [10, 20];**
3. **PRINT\_TREE set\_name;** - виведення на екран внутрішньої структури R-дерева, побудованого для множини **set\_name**. Для внутрішніх вузлів спочатку повинно виводитись ліве піддерево, а потім праве. Див. [9].  
     
   **Приклад:**  
   Для множини, що містить відрізки **[3, 4]**, **[2, 7]**, **[5, 8]**, **[4, 6]** та **[5, 10]** вивід повинен бути схожим до наступного (структура дерева може відрізнятись):  
     
   **[2, 10]**

**├── [3, 4]**

**└── [2, 10]**

**├── [2, 7]**

**│ ├── [2, 7]**

**│ └── [4, 6]**

**└── [5, 10]**

**├── [5, 8]**

**└── [5, 10]**

1. **CONTAINS set\_name [l, h];** - перевірка входження відрізка **[l, h]** до множини **set\_name**. В якості результату команда повинна повертати **TRUE**, якщо відрізок входить до множини та **FALSE**, якщо ні.  
     
   **Приклад:**  
   **CONTAINS reservations [10, 20];**
2. **SEARCH set\_name \*[WHERE query];** - пошук відрізків у множині **set\_name**.

В основній частині завдання команда має форму **SEARCH set\_name;** і повинна просто повернути всі відрізки зі вказаної множини. Додаткова частина (**\***) передбачає підтримку ключового слова **WHERE** та слідуючого за ним пошукового запиту. В цьому випадку команда повинна включити до вибірки тільки ті відрізки, які задовольняють даний запит. Визначення запиту **query** має наступний вигляд:   
 **query := CONTAINS [L, H]  
 | INTERSECTS [L, H]  
 | LEFT\_OF x**   
Пояснення:

* + **CONTAINS [L, H]** - до вибірки повинні бути включені відрізки, які містять в собі заданий відрізок **[L, H]** (тобто всі відрізки **[l, h]** такі, що: **l <= L <= H <= h**).
  + **INTERSECTS [L, H]** - до вибірки повинні бути включені всі відрізки, які перетинаються із заданим відрізком **[L, H]**.
  + **LEFT\_OF x** - до вибірки повинні бути включені всі відрізки, які знаходяться повністю лівіше вказаного значення **x** (тобто всі відрізки **[l, h]** такі, що: **l <= h <= x**).

В якості результату команда повинна повертати список знайдених відрізків, в будь-якому порядку.  
  
**Приклади команд:  
SEARCH reservations;  
SEARCH reservations WHERE CONTAINS [10, 12];**

**SEARCH reservations WHERE INTERSECTS [5, 20];**

**SEARCH reservations WHERE LEFT\_OF 15;**

### Варіант №13: Множина відрізків на прямій (KD-Tree)

В даному варіанті передбачається, що створені множини відрізків зберігаються у вигляді структури даних відомої як KD-дерево.  
Перелік команд, які необхідно реалізувати:

1. **CREATE set\_name;** - створення нової множини з назвою **set\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Set set\_name has been created”**.  
     
   **Приклад:**  
   **CREATE reservations;**
2. **INSERT set\_name [l, h];** - додавання нового відрізку з нижньою границею **l** та верхньою границею **h** до множини **set\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Range [l, h] has been added to set\_name”**.  
     
   **Приклад:**  
   **INSERT reservations [10, 20];**
3. **PRINT\_TREE set\_name;** - виведення на екран внутрішньої структури KD-дерева, побудованого для множини **set\_name**. Для внутрішніх вузлів спочатку повинно виводитись ліве піддерево, а потім праве. Див. [9].  
     
   **Приклад:**  
   Для множини, що містить відрізки **[3, 4]**, **[2, 7]**, **[5, 8]**, **[4, 6]** та **[5, 10]** вивід повинен бути схожим до наступного (структура дерева може відрізнятись):  
     
   **[3, 4]**

**├── [2, 7]**

**└── [5, 8]**

**├── [4, 6]**

**└── [5, 10]**

1. **CONTAINS set\_name [l, h];** - перевірка входження відрізка **[l, h]** до множини **set\_name**. В якості результату команда повинна повертати **TRUE**, якщо відрізок входить до множини та **FALSE**, якщо ні.  
     
   **Приклад:**  
   **CONTAINS reservations [10, 20];**
2. **SEARCH set\_name \*[WHERE query];** - пошук відрізків у множині **set\_name**.

В основній частині завдання команда має форму **SEARCH set\_name;** і повинна просто повернути всі відрізки зі вказаної множини. Додаткова частина (**\***) передбачає підтримку ключового слова **WHERE** та слідуючого за ним пошукового запиту. В цьому випадку команда повинна включити до вибірки тільки ті відрізки, які задовольняють даний запит. Визначення запиту **query** має наступний вигляд:   
 **query := CONTAINED\_BY [L, H]  
 | INTERSECTS [L, H]  
 | RIGHT\_OF x**   
Пояснення:

* + **CONTAINED\_BY [L, H]** - до вибірки повинні бути включені відрізки, які містяться всередині заданого відрізка **[L, H]** (тобто всі відрізки **[l, h]** такі, що: **L <= l <= h <= H**).
  + **INTERSECTS [L, H]** - до вибірки повинні бути включені всі відрізки, які перетинаються із заданим відрізком **[L, H]**.
  + **RIGHT\_OF x** - до вибірки повинні бути включені всі відрізки, які знаходяться повністю правіше вказаного значення **x** (тобто всі відрізки **[l, h]** такі, що: **x <= l <= h**).

В якості результату команда повинна повертати список знайдених відрізків, в будь-якому порядку.  
  
**Приклади команд:  
SEARCH reservations;  
SEARCH reservations WHERE CONTAINED\_BY [0, 30];**

**SEARCH reservations WHERE INTERSECTS [5, 20];**

**SEARCH reservations WHERE RIGHT\_OF 15;**

### **Варіант № 14:** Множина рядків на основі префіксного дерева (trie)

В даному варіанті передбачається, що створені множини рядків зберігаються у вигляді структури даних відомої як префіксне дерево (trie).

Варто зауважити, що існує два можливих представлення префіксних дерев: звичайне та компактне (див. приклад нижче). Різниця полягає в тому, що у компактному дереві внутрішні вузли зберігають не по одному символу, а довші фрагменти рядків, якщо жоден символ в межах даного фрагменту не має інших нащадків. Дане представлення є більш ефективним з точки зору використовуваної памʼяті, але суттєво ускладнює алгоритм вставки нових рядків у дерево (з практичної точки зору, його часова складність залишається такою ж). Тому в рамках даного завдання рекомендується реалізувати звичайне представлення.

| **“a”**  **└── “b”**  **├── “c”**  **└── “d”** | **“ab”**  **├── “c”**  **└── “d”** |
| --- | --- |

Можливі префіксні дерева, побудовані з рядків **“abc”** та **“abd”**: звичайне (зліва) та компактне (справа).

Перелік команд, які необхідно реалізувати:

1. **CREATE set\_name;** - створення нової множини з назвою **set\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Set set\_name has been created”**.  
     
   **Приклад:**  
   **CREATE descriptions;**
2. **INSERT set\_name “value”;** - додавання нового рядка до множини **set\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“String has been added to set\_name”**.  
     
   **Приклад:**  
   **INSERT descriptions “value”;**
3. **PRINT\_TREE set\_name;** - виведення на екран внутрішньої структури префіксного дерева, побудованого для множини **set\_name**. Див. [9].  
     
   **Приклад:**  
   Для множини, що містить рядки **“ab”**, **“abc”**, **“abd”** та **”xy”** вивід повинен бути схожим до наступного:  
     
   **[root]**

**├── “a”**

**│ └── “b”**

**│ ├── “c”**

**│ └── “d”**

**└── “x”**

**└── “y”**

1. **CONTAINS set\_name “value”;** - перевірка входження рядка **“value”** до множини **set\_name**. В якості результату команда повинна повертати **TRUE**, якщо рядок входить до множини та **FALSE**, якщо ні.  
     
   **Приклад:**  
   **CONTAINS descriptions “value”;**
2. **SEARCH set\_name \*[WHERE query] \*[ASC | DESC];** - пошук рядків у множині **set\_name**.

В основній частині завдання команда має форму **SEARCH set\_name;** і повинна просто повернути всі рядки зі вказаної множини. Додаткова частина (**\***) передбачає підтримку ключового слова **WHERE** та слідуючого за ним пошукового запиту. В цьому випадку команда повинна включити до вибірки тільки ті множини, які задовольняють даний запит. Визначення запиту **query** має наступний вигляд:   
 **query := BETWEEN “from”, “to”  
 | MATCH “pattern”**   
Пояснення:

* + **BETWEEN “from”, “to”** - до вибірки повинні бути включені рядки, які в лексикографічному порядку знаходяться між рядками **“from”** та **“to”** (тобто всі рядки **“value”** такі, що: **“from” <= “value” <= “to”**).
  + **MATCH “pattern”** - до вибірки повинні бути включені всі рядки, які відповідають заданому шаблону **“pattern”**. **“pattern”** може містити два спеціальних символи - **?** та **\***.
    - **?** - відповідає одному довільному символу. Може зустрічатись у **“pattern”** довільну кількість разів на будь-яких позиціях.
    - **\*** - відповідає довільній кількості (включаючи нуль) довільних символів. Може зустрічатись у **“pattern”** тільки один раз, на останній позиції. Таким чином, наявність у шаблону символу **\*** означає необхідність виконання префіксного пошуку.

**Приклади:**Шаблону **“hello”** відповідає тільки рядок **“hello”**.  
Шаблону **“?ad”** відповідають рядки **“bad”**, **“mad”**, **“dad”** тощо.  
Шаблону **“happ\*”** відповідають рядки **“happy”**, **“happiness”**, **“happen”** тощо.  
Шаблону **“?ace\*”** відповідають рядки **“face”**, **“faced”**, **“race”**, **“racer”** тощо.

В якості результату команда повинна повертати список знайдених рядків. (**\***) Якщо команда завершується опціональним ключовим словом **ASC**, то рядки потрібно вивести в прямому лексикографічному порядку, якщо **DESC**, то в зворотньому. Якщо ключових слів **ASC** та **DESC** немає взагалі, то також в прямому лексикографічному порядку.  
  
**Приклади команд:  
SEARCH descriptions;  
SEARCH descriptions DESC;  
SEARCH descriptions WHERE BETWEEN “abc”, “xyz”;**

**SEARCH descriptions WHERE MATCH “?ell\*” ASC;**

### Варіант №15: Колекція числових множин на основі інвертованого індексу

В даному варіанті передбачається, що по створеним колекціям будуються інвертовані індекси (inverted index), які використовуються для прискорення виконання пошукових команд. В якості елементів такого індексу виступають елементи множин, з яких складається колекція.  
Перелік команд, які необхідно реалізувати:

1. **CREATE collection\_name;** - створення нової колекції з назвою **collection\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Collection collection\_name has been created”**.  
     
   **Приклад:**  
   **CREATE grades;**
2. **INSERT collection\_name {val\_1, val\_2, ..., val\_N};** - додавання нової множини з елементами **val\_1**, **val\_2**, …, **val\_N** до колекції **collection\_name**. Можна вважати, що всі елементи множини є унікальними, тобто зустрічаються у фігурних дужках рівно один раз. В якості результату команда повертати певне повідомлення для користувача, наприклад **“Set has been added to collection\_name”**.  
     
   **Приклад:**  
   **INSERT grades {90, 85, 95};**
3. **PRINT\_INDEX collection\_name;** - виведення на екран внутрішньої структури інвертованого індексу, побудованого для колекції **collection\_name**. Вивід повинен включати елементи та множини, яким вони належать.  
     
   **Приклад:**  
   Для індексу, що містить множини **{1, 2, 3}**, **{4, 5}** та **{1, 4}**, яким було присвоєно ідентифікатори **s1,** **s2** та **s3** відповідно, вивід повинен бути схожим до наступного:  
     
   **1: s1, s3  
   2: s1  
   3: s1  
   4: s2, s3  
   5: s2**

1. **CONTAINS collection\_name {val\_1, val\_2, ..., val\_N};** - перевірка входження множини до колекції **collection\_name**. В якості результату команда повинна повертати **TRUE**, якщо множина входить до колекції та **FALSE**, якщо ні. Зверніть увагу, що оскільки ми працюємо із множинами, то порядок елементів у фігурних дужках не повинен впливати на результат.  
     
   **Приклад:**  
   **CONTAINS grades {90, 85, 95};**
2. **SEARCH collection\_name \*[WHERE query];** - пошук множин у колекції **collection\_name**.

В основній частині завдання команда має форму **SEARCH collection\_name;** і повинна просто повернути всі множини зі вказаної колекції. Додаткова частина (**\***) передбачає підтримку ключового слова **WHERE** та слідуючого за ним пошукового запиту. В цьому випадку команда повинна включити до вибірки тільки ті множини, які задовольняють даний запит. Визначення запиту **query** має наступний вигляд:   
 **query := INTERSECTS {val\_1, ..., val\_N}  
 | CONTAINS {val\_1, ..., val\_N}  
 | CONTAINED\_BY {val\_1, ..., val\_N}**   
Пояснення:

* + **INTERSECTS {val\_1, ..., val\_N}** - до вибірки повинні бути включені множини, які мають спільні елементи із множиною **{val\_1, ..., val\_N}**.
  + **CONTAINS {val\_1, ..., val\_N}** - до вибірки повинні бути включені множини, які включають в себе множину **{val\_1, ..., val\_N}**, тобто містять всі її елементи.
  + **CONTAINED\_BY {val\_1, ..., val\_N}** - до вибірки повинні бути включені множини, які є підмножинами множини **{val\_1, ..., val\_N}**.

В якості результату команда повинна повертати список знайдених множин. Порядок рядків може бути довільним.  
  
**Приклади команд:  
SEARCH grades;  
SEARCH grades WHERE INTERSECTS {82, 90, 92};**

**SEARCH grades WHERE CONTAINS {85, 90};  
SEARCH grades WHERE CONTAINED\_BY {75, 80, 85, 90, 95, 100};**

### Варіант №16: Колекція числових множин на основі RD-дерева

В даному варіанті передбачається, що створені колекції зберігаються у вигляді структури даних відомої як RD-дерево.  
Перелік команд, які необхідно реалізувати:

1. **CREATE collection\_name;** - створення нової колекції з назвою **collection\_name**. В якості результату команда повинна повертати певне повідомлення для користувача, наприклад **“Collection collection\_name has been created”**.  
     
   **Приклад:**  
   **CREATE grades;**
2. **INSERT collection\_name {val\_1, val\_2, ..., val\_N};** - додавання нової множини з елементами **val\_1**, **val\_2**, …, **val\_N** до колекції **collection\_name**. Можна вважати, що всі елементи множини є унікальними, тобто зустрічаються у фігурних дужках рівно один раз. В якості результату команда повертати певне повідомлення для користувача, наприклад **“Set has been added to collection\_name”**.  
     
   **Приклад:**  
   **INSERT grades {90, 85, 95};**
3. **PRINT\_TREE collection\_name;** - виведення на екран внутрішньої структури RD-дерева, побудованого для колекції **collection\_name**. Для внутрішніх вузлів спочатку повинно виводитись ліве піддерево, а потім праве. Див. [9].  
     
   **Приклад:**  
   Для колекції, що містить множини **{1, 2, 3}**, **{2, 5}**, **{1, 4, 8}** та **{1, 4, 9}** вивід повинен бути схожим до наступного (структура дерева може відрізнятись):  
     
   **{1, 2, 3, 4, 5, 6, 8, 9}**

**├── {1, 2, 3, 5}**

**│ ├── {1, 2, 3}**

**│ └── {2, 5}**

**└── {1, 4, 8, 9}**

**├── {1, 4, 8}**

**└── {1, 4, 9}**

1. **CONTAINS collection\_name {val\_1, val\_2, ..., val\_N};** - перевірка входження множини до колекції **collection\_name**. В якості результату команда повинна повертати **TRUE**, якщо множина входить до колекції та **FALSE**, якщо ні. Зверніть увагу, що оскільки ми працюємо із множинами, то порядок елементів у фігурних дужках не повинен впливати на результат.  
     
   **Приклад:**  
   **CONTAINS grades {90, 85, 95};**
2. **SEARCH collection\_name \*[WHERE query];** - пошук множин по колекції **collection\_name**.

В основній частині завдання команда має форму **SEARCH collection\_name;** і повинна просто повернути всі множини зі вказаної колекції. Додаткова частина (**\***) передбачає підтримку ключового слова **WHERE** та слідуючого за ним пошукового запиту. В цьому випадку команда повинна включити до вибірки тільки ті множини, які задовольняють даний запит. Визначення запиту **query** має наступний вигляд:   
 **query := INTERSECTS {val\_1, ..., val\_N}  
 | CONTAINS {val\_1, ..., val\_N}  
 | CONTAINED\_BY {val\_1, ..., val\_N}**   
Пояснення:

* + **INTERSECTS {val\_1, ..., val\_N}** - до вибірки повинні бути включені множини, які мають спільні елементи із множиною **{val\_1, ..., val\_N}**.
  + **CONTAINS {val\_1, ..., val\_N}** - до вибірки повинні бути включені множини, які включають в себе множину **{val\_1, ..., val\_N}**, тобто містять всі її елементи.
  + **CONTAINED\_BY {val\_1, ..., val\_N}** - до вибірки повинні бути включені множини, які є підмножинами множини **{val\_1, ..., val\_N}**.

В якості результату команда повинна повертати список знайдених множин. Порядок рядків може бути довільним.  
  
**Приклади команд:  
SEARCH grades;  
SEARCH grades WHERE INTERSECTS {82, 90, 92};**

**SEARCH grades WHERE CONTAINS {85, 90};  
SEARCH grades WHERE CONTAINED\_BY {75, 80, 85, 90, 95, 100};**

## Перелік посилань

Посилання на Wikipedia не наводяться, оскільки передбачається, що студенти в змозі самостійно знайти таку інформацію.

1. <https://habr.com/ru/company/postgrespro/blog/326096/> - прекрасна в усіх сенсах серія статей про індекси в PostgreSQL, яка багато в чому лежить в основі даного практикуму. Містить інформацію, корисну для всіх варіантів.
2. <https://slides.com/fxku/sotm19> - певна корисна інформація по GiST та SP-GiST індексам в PostgreSQL.
3. <https://use-the-index-luke.com> - чудовий ресурс про індекси в SQL, на прикладі різних БД. Корисний з практичної точки зору, оскільки містить досить вичерпну інформацію про те як індекси пришвидшують виконання різних видів SQL запитів.
4. <https://ruslanspivak.com/lsbasi-part1/> - серія статей на тему розробки власної мови програмування та інтерпретатора для неї. Перші 8 частин особливо цікаві, оскільки розповідають про граматики та синтаксичний аналіз на їх основі.
5. <https://www.coursera.org/lecture/algorithms-part1/kd-trees-Yionu> - відео по KD-деревам, що є складовою одного із курсів по алгоритмам, доступних на Coursera.

# T. Cormen, C. Leiserson, R. Rivest, C. Stein “Introduction to Algorithms, 3rd Edition” - фундаментальна праця, яка покриває більшість потреб всього курсу, не тільки комп’ютерного практикуму. Серед іншого, містить інформацію про бінарні дерева та B-дерева.

1. R. Sedgewick, K. Wayne “Algorithms (4th Edition)” - теж чудова книга про алгоритми з практичними прикладами на Java. Серед іншого, містить інформацію про префіксні дерева.
2. Статті, які проливають світло на те як влаштовані реляційні (а насправді - не тільки) бази даних зсередини, включаючи планування, оптимізацію та виконання запитів. Зважаючи на те, що в даному практикумі кожен окремий варіант пропонує реалізувати тільки деяку обмежену підмножину можливостей сучасних реляційних БД, дані матеріали несуть не велику практичну цінність, проте я не міг не поділитись посиланнями з тими, кому буде цікаво ознайомитись із роботою баз даних ще глибше:
   * <https://habr.com/ru/company/mailru/blog/266811/>
   * <https://habr.com/ru/company/badoo/blog/461699/>
   * <https://habr.com/ru/company/oleg-bunin/blog/569254/>
   * <https://www.cockroachlabs.com/blog/building-cost-based-sql-optimizer/>
3. <https://stackoverflow.com/a/8948691> - алгоритм виводу на екран дерева, який ми наполегливо і рекомендуємо адаптувати для своєї реалізації.