Praca domowa I—Baza Danych

7 listopada 2025

1 Instrukcja

Napisz prostą bazę danych.

Termin projektu: 28.11.2025. Każdy tydzień opóźnienia skutkuje ograniczeniem maksymalnej liczby punktów o 5.

W sekcji 2 znajdziesz wymaganą do zaimplementowania funkcjonalność. Za jej pełną implementację otrzymać możesz 10pkt. W sekcji 3 znajdziesz dodatkową funkcjonalność. Przy każdej funkcjonalności znajduje się liczba dodatkowych punktów za jej implementację i jej identyfikator. Aby otrzymać punkty za dodatkową funkcjonalność, musisz zaimplementować funkcjonalność z sekcji 2 za przynajmniej 8 punktów. W sekcji 4 znajdziesz rzeczy, których nie możesz robić w projekcie.

Jeśli w poleceniu jest napisane, że należy zaimplementować strukturę/cechę/funkcję o nazwie `X`, która nie jest jednak definiowana szczegółowo przez polecenie, to jej implementacja jest dowolna.

Całkowicie zabronione jest używanie bibliotek innych niż te, które dozwolone są w poleceniu. Projekty używające innych bibliotek nie będą sprawdzane.

Dozwolone jest używanie dowolnych elementów z biblioteki standardowej, o ile nie są one zakazane przez sekcję 4.

Ocena za projekt liczona jest ze wzoru

$$k = \min(r + a + p, 15),$$

gdzie k—ocena końcowa, r—punkty z sekcji 2, $a = 0$ jeśli $r < 8$, lub a jest równe punkty z sekcji 3 jeśli $r \geq 8$, p—punkty z sekcji 4.

2 Wymagana funkcjonalność

1. (3pkt) Implementacja bazy danych.
   (a) Baza danych składa się z *tabel*.
   (b) Struktura bazy danych jest generyczna po typie klucza `Database<K: DatabaseKey>`, gdzie `DatabaseKey` to cecha (trait) z metodami potrzebnymi dla *kluczy* (np. `is_equal_to(&self, other: &Self)`). Dopuszczalne typy kluczy to `String` i `i64`.
   (c) Tabela składa się z nazwanych *pól* z *typami*, spośród których jedno pole jest kluczem.
   (d) Możliwymi typami pól są: `bool`, `String`, `Int` (`i64`), `Float` (`f64`). W kodzie powinien znaleźć się `enum Value` opisujący te wartości.
   (e) Do tabeli można dodawać i usuwać z niej *rekordy*.
   (f) *Rekord* tabeli to kolekcja wartości odpowiadających polom tej tabeli. W danej tabeli nie może być wiele rekordów o tym samym kluczu. `struct Record` zawiera słownik `HashMap` przypisujący nazwom pól wartości typu `Value`.
   (g) Rekordy bazy danych w trakcie działania programu trzymane są w B-drzewie: `BTreeMap<K, Record>`, gdzie `K` jest typem klucza danej bazy danych.

   Podpowiedź: przydatne może być zdefiniowanie

```
enum AnyDatabase {
StringDatabase(Database<String>),
IntDatabase(Database<i64>)
}
```

2. Konsola, w której można wpisywać następujące polecenia.

   (a) (1pkt) Polecenie

   ```
   CREATE table KEY key_field
   FIELDS field_1: type_1, field_2: type_2, ..., field_n: type_n
   ```

   tworzące tabelę `table` z polami `field_i` o typach `type_i`, spośród których pole `key_field` jest kluczem tabeli. Kolumna klucza musi mieć typ, który jest typem klucza obecnej bazy danych.

   (b) (0.5pkt) Polecenie `INSERT field_1 = value_1, ..., field_n = value_n INTO table`, które dodaje rekord do tabeli `table`. Wymagane jest podanie wszystkich pól.

   (c) (0.5pkt) Polecenie `DELETE key FROM table`, które usuwa rekord o kluczu `key` z tabeli `table`.

   (d) (0.75pkt) Polecenie

   ```
   SELECT field_1, field_2, ..., field_n FROM table WHERE condition
   ```

   zwracające pola `field_1, field_2, ..., field_n` z tabeli `table` z rekordów, dla których spełnione jest `condition`. `condition` ma postać `column_name <op> value`, gdzie `<op>` $\in \{$`=`, `!=`, `<`, `<=`, `>`, `>=`$\}$. Klauzula `WHERE` jest opcjonalna. Przykłady polecenia:

   ```
   SELECT collection, name, type FROM glassware WHERE type = "martini glass"
   SELECT grape_variety, distillery FROM armagnacs WHERE year <= 1980
   SELECT name, is_cursive FROM fonts
   ```

   (e) (1pkt) Każdemu poleceniu odpowiada struktura zawierająca wszystkie dane, potrzebne do jego wykonania. Każda z tych struktur powinna implementować cechę `Command`. Dodatkowo struktura dla `CREATE` powinna zawierać referencję na obiekt bazy danych, do którego się odnosi, a `INSERT`, `DELETE` i `SELECT` na obiekt tabeli, do których się odnoszą.

   (f) (0.5pkt) Polecenie `SAVE_AS file_path`, które zapisuje dotychczas wykonane polecenia `CREATE`, `SELECT`, `INSERT` i `DELETE` do pliku `file_path`.

   (g) (0.5pkt) Polecenie `READ_FROM file_path`, które wykonuje polecenia w pliku `file_path` oddzielone znakami nowej linii. W wypadku błędu podczas wykonywania któregokolwiek polecenia przetwarzanie pliku jest przerywane.

3. (0.5pkt) Uruchamiając program podajemy opcję CLI wybierającą rodzaj klucza, z którym będziemy działać. Możesz użyć biblioteki *clap* (`https://docs.rs/clap/latest/clap/`).

4. (0.5pkt) Podział projektu na bibliotekę (definicja modułów w `lib.rs`) i część użytkową (definicja modułów w `main.rs`). Część użytkowa powinna zawierać kod *konsoli poleceń*, a biblioteka resztę kodu.

5. (0.25pkt) Typ `enum Error` dla funkcji w bazie danych. Powinien zawierać warianty dla wszystkich kategorii błędów, które można napotkać, np. mogą to być m.in. `InvalidField`, `InvalidKey`, `TableNotFound`, `IoError(std::io::Error)`, itd. Zalecane jest użycie biblioteki *thiserror* (`https://docs.rs/thiserror/latest/thiserror/`). Funkcje w modułach biblioteki, powinny przekazywać błędy dalej do kodu aplikacji, chyba że potrafią same sobie z nimi poradzić.

6. (1pkt) Testy w każdym napisanym module, w szczególności dwa testy dla każdego z sześciu poleceń—test dla polecenia konstruowanego w kodzie i test dla polecenia parsowanego ze stringa. Łącznie 12 testów. W szczególności testy powinny testować oba rodzaje kluczy, a `SELECT` powinien być testowany z i bez `WHERE`.

7. Plik `README.md`, który zawiera:
   (a) Opis struktury projektu.
   (b) Opis funkcjonalności nieopisanych w poleceniu.
   (c) Twój ulubiony moduł i uzasadnienie, dlaczego jest ulubiony.
8. Plik ADDITIONS zawierający listę zaimplementowanych funkcjonalności z sekcji 3 w postaci listy ich identyfikatorów, np. plik ten może wyglądać tak:

```
arithmetic
grammar
network
```

3 Dodatkowa funkcjonalność

1. (2pkt, `order`) Dodanie klauzul `LIMIT n` i `ORDER_BY field` do `SELECT`, które pozwalają odpowiednio na ograniczenie wyświetlanych wyników do pierwszych `n` rekordów i na sortowanie po polu `field`. Przykładowo:

```
SELECT better, amount_won FROM bets ORDER_BY amount_won
SELECT painter FROM paintings LIMIT 5
SELECT name, publication_count FROM scientists ORDER_BY publication_count LIMIT 10
```

2. (2pkt, `logical`) Wspieranie spójników logicznych `AND` i `OR` z nawiasami w klauzuli `WHERE`, np.

```
SELECT title FROM books WHERE
(printing_date = 1749 OR printing_date = 1750 AND page_count > 500)
AND colour = "red"
```

3. (2pkt, `aggregate`) Funkcje agregujące `avg` (średnia), `min` i `max` w `SELECT` oraz klauzula `GROUP_BY` wybierająca dokładnie jedną kolumnę, np.

```
SELECT max(price), avg(rating), release_year FROM movies
GROUP_BY release_year
```

   Funkcje agregujące mogą wystąpić w `SELECT` wtedy i tylko wtedy, gdy występuje też `GROUP_BY`. Gdy występuje `GROUP_BY`, wybierać można tylko kolumny z nałożonymi funkcjami agregującymi lub kolumnę wybraną przez `GROUP_BY`.

4. (2pkt, `network`) Połączenie sieciowe z bazą danych: w projekcie dostępne są dwa pliki binarne, jeden otwiera bazę danych z konsolą, a drugi z serwerem `UDP`, który przyjmuje identyczne polecenia jak konsola i odsyła takie same odpowiedzi (w `Cargo.toml` jest możliwość zdefiniowania kilku plików wykonywalnych).
5. (3pkt, `arithmetic`) Wspieranie wyrażeń arytmetycznych z liczbami w klauzuli `WHERE`, np.

```
SELECT title, publisher FROM games
WHERE price / (user_rating + 0.2 * user_playtime) >= 10
```

6. (3pkt, `grammar`) Użycie biblioteki *pest* (`https://pest.rs/`) do parsowania zapytań (jedna gramatyka opisująca wszystkie wyrażenia).
7. (3pkt, `binary`) Binarny format zapisu bazy danych. Baza może być zapisana do pliku binarnego poleceniem `DUMP_TO file_path` i wczytana poleceniem `LOAD_FROM file_path`. Sam format jest dowolny, ale musi być binarny: nie może być to format tekstowy taki jak JSON, XML, YAML, itd.

8. (6pkt, `tui`) Interfejs TUI zaimplementowany przy pomocy biblioteki *ratatui* (`https://github.com/ratatui/ratatui`). Powinien pozwalać na wszystko, na co pozwala konsola poleceń w wymaganej funkcjonalności (tworzenie tabeli, dodawanie rekordów, usuwanie ich, wybieranie z warunkiem, itd.).

9. (1-3pkt, `custom`) Dowolna inna dodatkowa funkcjonalność, oceniana subiektywnie przez prowadzącego (najlepiej skonsultować przed terminem oddania).

4 Praktyki zakazane

Pojawienie się któregokolwiek z poniższych elementów w projekcie skutkuje odjęciem tylu punktów, ile sprecyzowano.

1. Użycie metod `unwrap`, `expect` lub makra `panic`. −1pkt za 1 lub 2 użycia, −2pkt za więcej użyć. Dozwolone jednak używanie w testach.

2. Używanie struktury, gdzie właściwy byłby `enum`, to jest trzymanie w strukturze pól, których wartość czasami ma, a czasami nie ma znaczenia, np.

   ```
   struct IP {
   v4: String,
   v6: String,
   is_v4: bool
   }
   ```

   -1pkt za jedno wystąpienie, -2pkt za więcej.

3. Ignorowanie błędów, kiedy powinny być obsłużone, np. wołanie funkcji, która zwraca `Result<T, E>` i sprawdzanie przez `if let` tylko przypadku `Ok`, kiedy błąd ma znaczenie (jeśli nie ma, to oczywiście dozwolone). -1pkt za jedno lub 2 wystąpienia, -2pkt za więcej.

4. Funkcje dłuższe niż 30 linijek. -1pkt za 1 lub 2, -2pkt za więcej.

5. Funkcje, które niepotrzebnie przejmują własność obiektów, kiedy mogłyby je tylko pożyczać. -1pkt za 1-3 takich funkcji. -2pkt za więcej.

6. Używanie `clone`, kiedy jest niepotrzebny (tzn. kod kompiluje się po jego usunięciu, a nie wpływa to na logikę). -1pkt za jedno lub więcej użyć.

7. Używanie `Rc`, `Arc`, `RefCell`, `Mutex`, `Cell`, -2pkt.

8. Używanie `unsafe`, -2pkt.