Звіт лабораторного практикуму з інформаційних технологій "Фрагментарна реалізація систем управління табличними базами даних" Варіант - "16"

Демедюк Віталій

6 грудня 2022 р.

Зміст

1 Постановка задачі

Вимоги щодо структури бази:

- кількість таблиць принципово не обмежена (реляції між таблицями не враховувати);
- кількість полів та кількість записів у кожній таблиці також принципово не обмежені.

У роботі треба забезпечити підтримку (для полів у таблицях) наступних типів:

- integer;
- real;
- char;
- string;
- текстові файли;
- інтервальний integer.

Також у роботі треба реалізувати функціональну підтримку для:

- створення бази;
- створення (із валідацією даних) та знищення таблиці з бази; перегляду та редагування рядків таблиці;
- збереження табличної бази на диску та, навпаки, зчитування її з диску;
- прямий добуток двох таблиць.

2 Етап №0(попередній етап)

Функціональна специфікація системи управління табличними базами даних (СУТБД) у вигляді діаграм прецедентів UML.

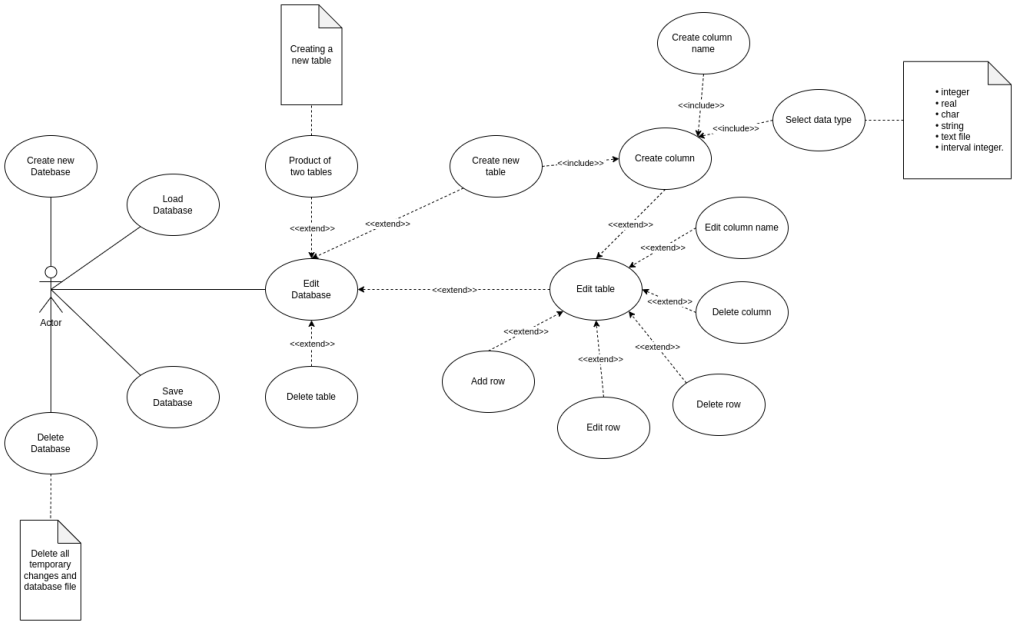

Рис. 1: Інтерфейс на основі форм:

3 Етап №1

Розробка власних класів для понять "Таблиця "База"та, можливо, деяких інших класів, спряжених із поняттям "Таблиця"(наприклад, "Схема таблиці "Атрибут "Рядок таблиці"тощо). Створення UML-діаграми класів (з наявними між класами відношеннями).

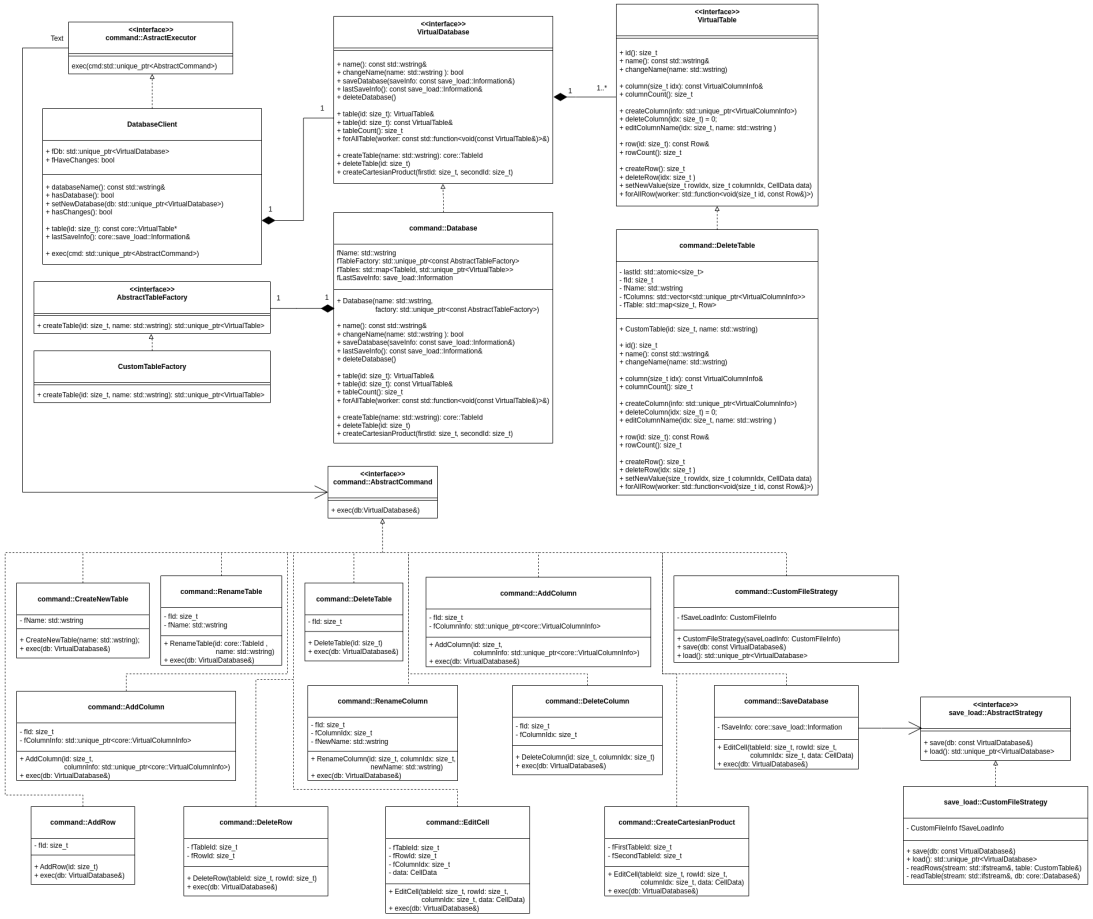

Рис. 2: Діаграма прицедентів

4 Етап №2

Проведення unit-тестування та забезпечення інтерфейсу користувача на основі форм

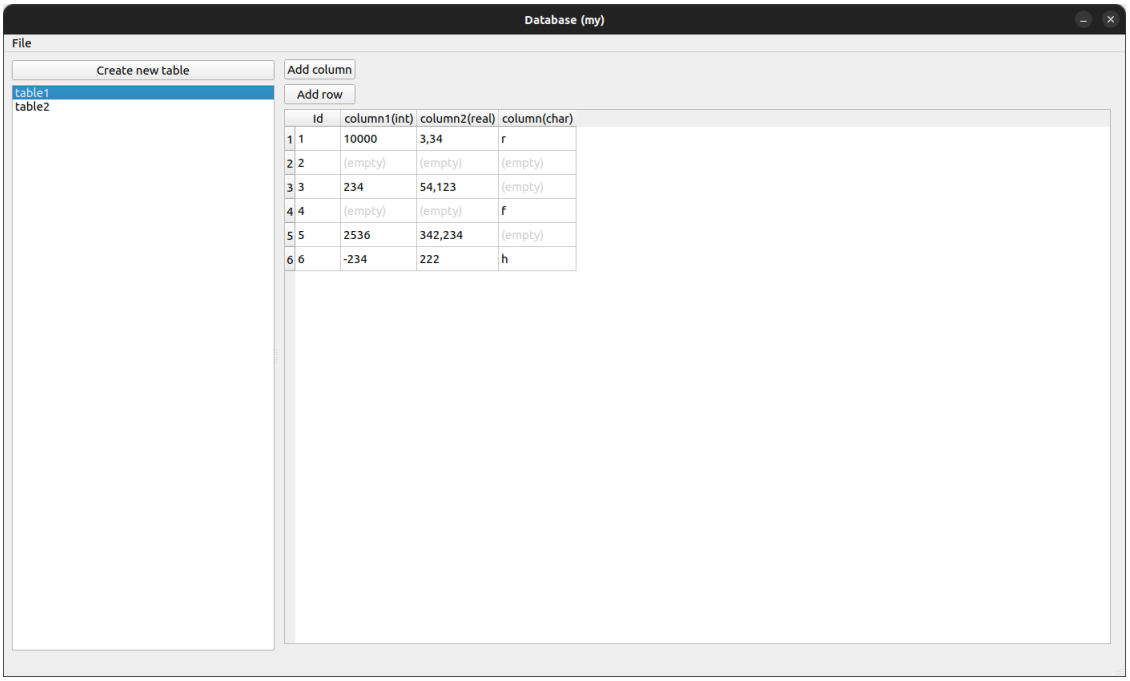

Рис. 3: UML діаграма класів

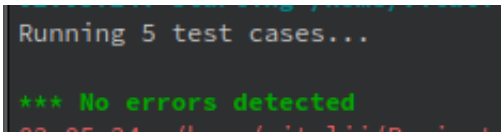

Рис. 4: Результат виконання unit-тестів

5 Етап №3

Використання реляційної СУБД(SQLite) для серіалізації об'єктів для збереження даних.

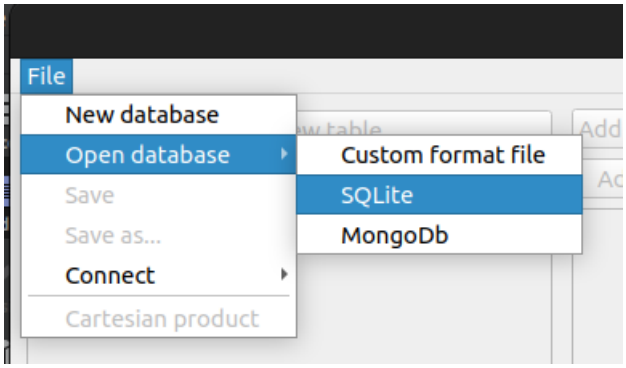

Рис. 5: Стільниковий клієнт. Завантаження даних з SQLite

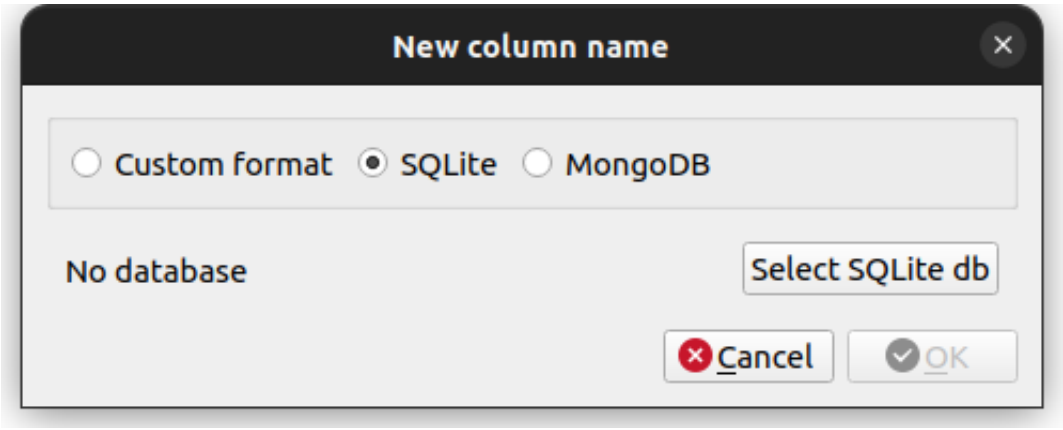

Рис. 6: Діалог збереження в SQLite

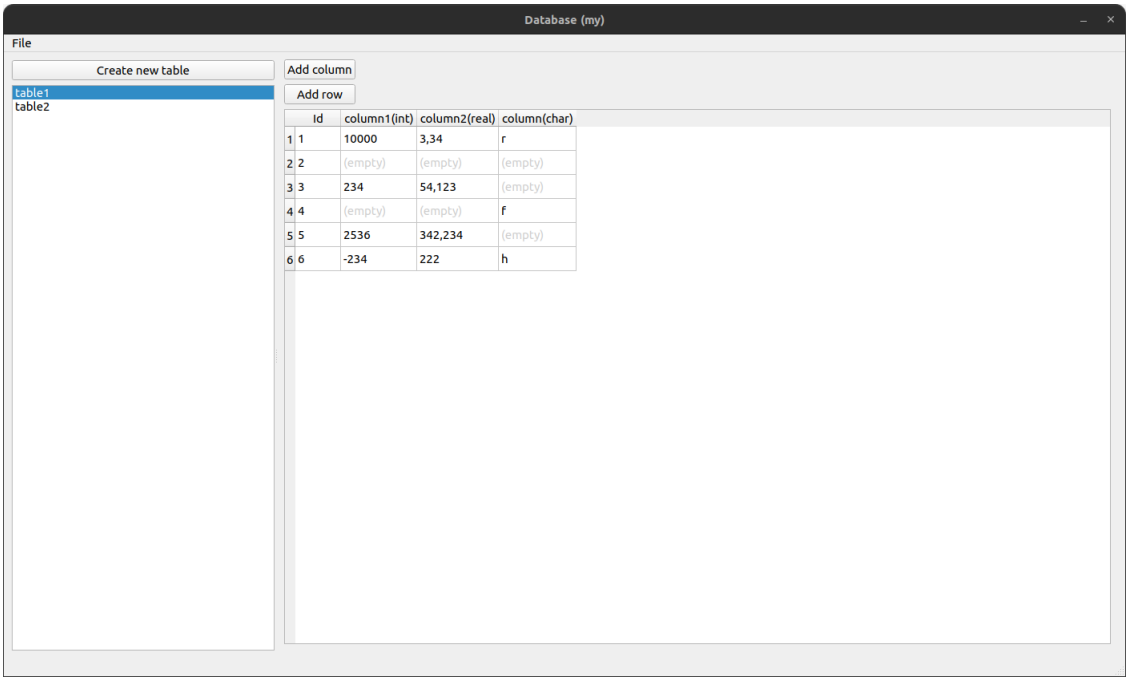

Рис. 7: SQLite база даних після збереження

6 Етап №4

Використання СУБД Mongo для серіалізації об'єктів для збереження даних.

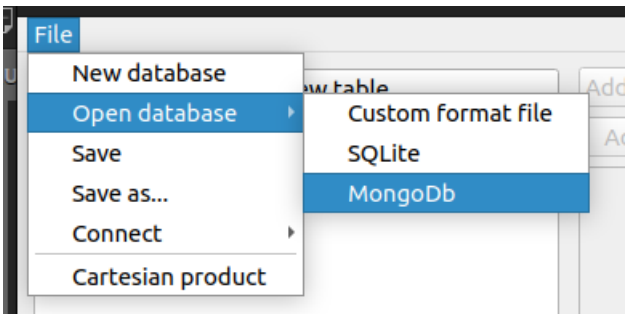

Рис. 8: Стільниковий клієнт. Завантаження даних з MongoDb

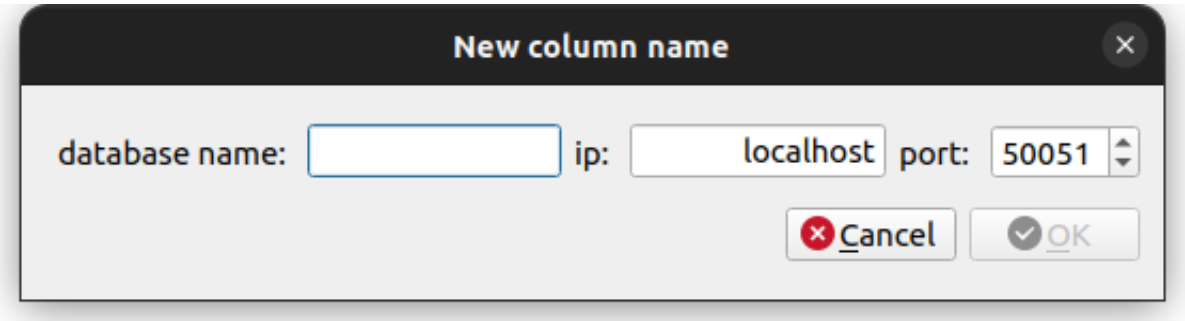

Рис. 9: Діалог завантаження з MongoDb

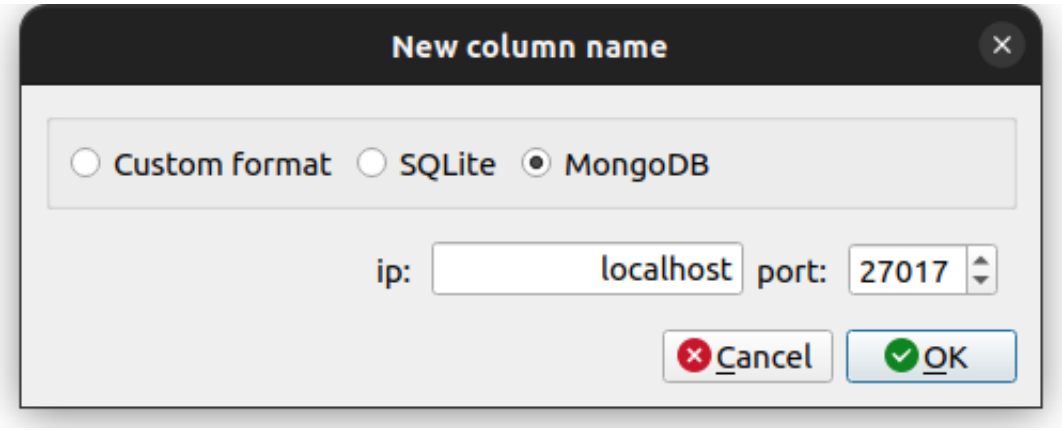

Рис. 10: Діалог збереження в MongoDb

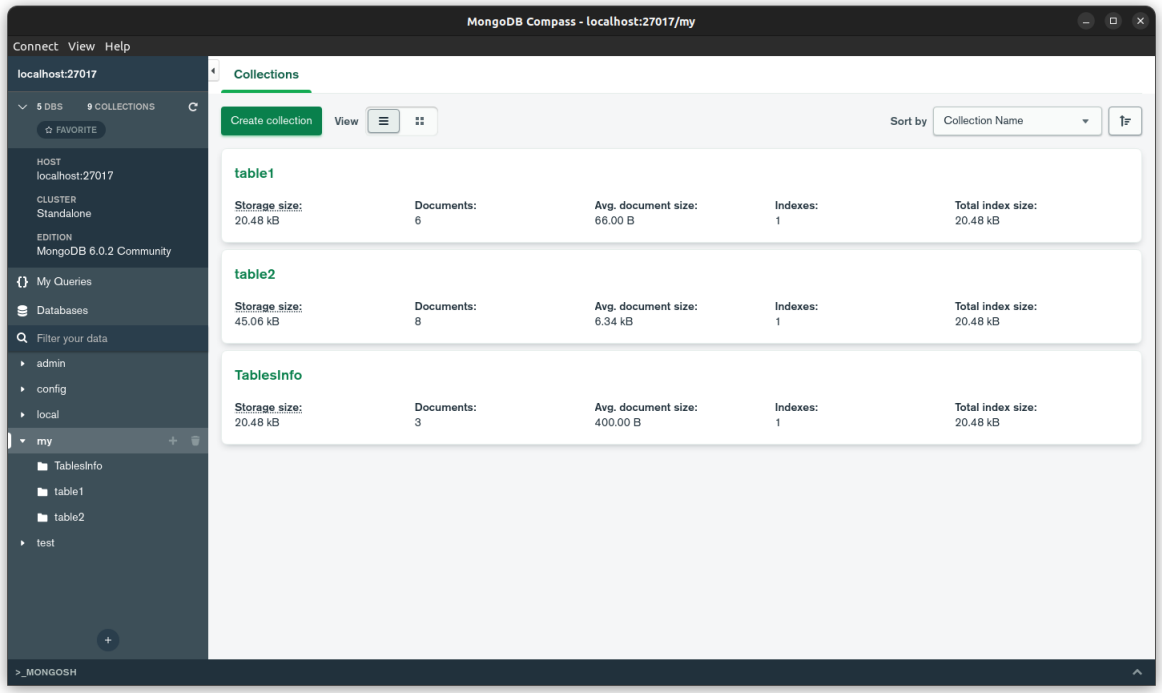

Рис. 11: MongoDb база даних після збереження

7 Етап №5

Розробка gRpc клієнта і сервера на одній мові програмування.

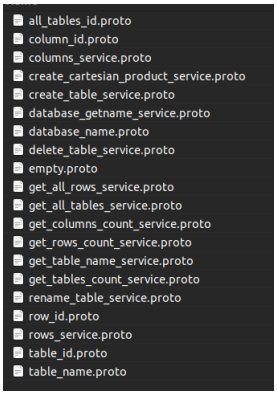

Рис. 12: Усі proto файли

```
service RowsService {
    rpc get(RowId) returns (Row) {}
```

```
    rpc create(TableId) returns (RowCreateResponse) {}
    rpc editCell(CellEditRequest) returns (Empty) {}
    rpc deleteRow(RowId) returns (Empty) {}
}
```

Лістинг 1: код gRpc сервіса для отриманя рядка таблиці

Рис. 13: Інтерфейс стільникового кліжента для підключення до сервера

```
Server listening on localhost:50051
Request. Get database name.
Request. Get all tables id.
Request. Get table name by id: 1
Request. Get table name by id: 2
Request. Get rows count for table 1
Request. Get column info for table1 column idx 0
Request. Get column info for table1 column idx 1
Request. Get column info for table1 column idx 2
Request. Get column info for table1 column idx 3
Request. Get columns count for table 1
Request. Get all rows id of table 1
```

Рис. 14: Фрагмент логів gRpc сервера

8 Етап №6-7

У цьому етапі був розроблений gRpc клієнт на мові Python для сервера, який був розроблений в етапі №5. Одночасно цей клієнт є GraphQL сервером.

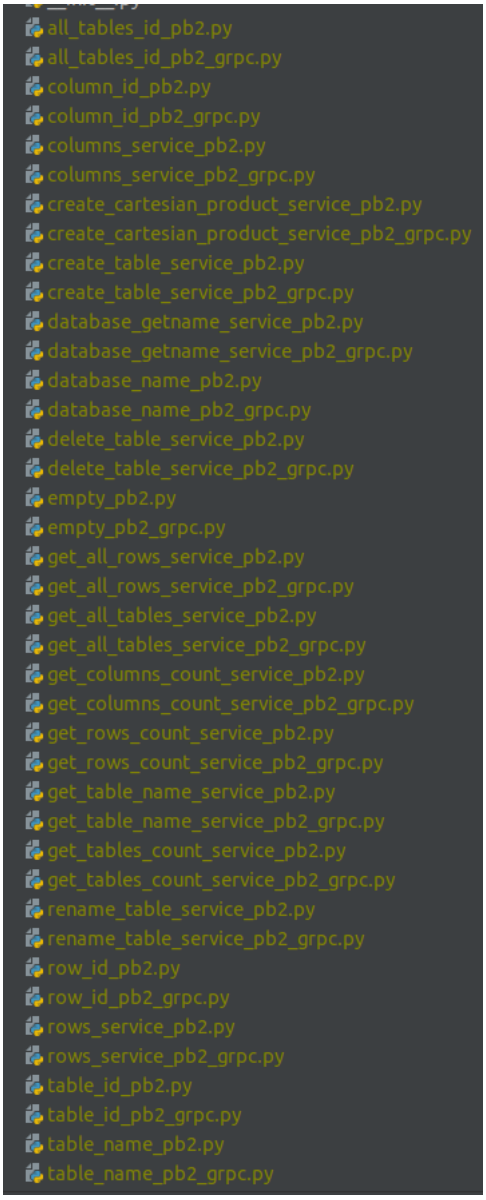

Рис. 15: Згенеровані proto/grpc файли для Python клієнта

```
schema {
    query: Query
    mutation: Mutation
}

type TableListItem {
    id: ID!
    name: String!
```

```
}

type GetTableListResult {
    success: Boolean!
    errors: [String!]
    tables_list: [TableListItem!]
}

type ColumnInfo {
    name: String!
    type: String!
    lower_limit: Int
    upper_limit: Int
}

type TableInfo {
    table_name: String!
    columns: [ColumnInfo!]
}

type GetTableInfoResult {
    success: Boolean!
    errors: [String!]
    table_info: TableInfo
}

type IntegerWrapper {
    int_value: Int!
}

type RealWrapper {
    real_value: Float!
}

type CharWrapper {
    char_value: String!
}

type StringWrapper {
    str_value: String
}

type File {
    filename: String!
}

type IntervalInteger {
    int_value: Int!
}
```

```
58
59 union Cell = IntegerWrapper | RealWrapper | CharWrapper |
       StringWrapper | File | IntervalInteger
60
61 type Row {
62     row_id: Int
63     cells: [Cell]
64 }
65
66 type Table {
67     rows: [Row!]
68 }
69
70 type GetTableResult {
71     success: Boolean!
72     errors: [String!]
73     table: Table
74 }
75
76 type Query {
77     getTableList: GetTableListResult!
78     getTableInfo(table_id: ID): GetTableInfoResult
79     getTable(table_id: ID): GetTableResult
80 }
81
82 type MutationResult {
83     success: Boolean!
84     errors: [String!]
85 }
86
87 type Mutation {
88     createNewTable(name: String!): MutationResult
89     renameTable(table_id: Int!, new_name: String!):
       MutationResult!
90     deleteTable(table_id: Int!): MutationResult!
91
92     createDefaultColumn(table_id: Int!, name: String!,
       column_type: String!): MutationResult!
93     createIntIntervalColumn(table_id: Int!, name: String!,
       lower_limit: Int!, upper_limit: Int!): MutationResult!
94     renameColumn(table_id: Int!, column_idx: Int!, new_name:
       String!): MutationResult!
95     deleteColumn(table_id: Int!, column_idx: Int!):
       MutationResult!
96
97     createNewRow(table_id: Int!): MutationResult!
98     deleteRow(table_id: Int!, row_id: Int!): MutationResult!
99     editIntCell(table_id: Int!, column_idx: Int!, row_id: Int
       !, data: Int!): MutationResult!
```

```
    editRealCell(table_id: Int!, column_idx: Int!, row_id:
   Int!, data: Float!): MutationResult!
    editCharCell(table_id: Int!, column_idx: Int!, row_id:
   Int!, data: String!): MutationResult!
    editStringCell(table_id: Int!, column_idx: Int!, row_id:
   Int!, data: String!): MutationResult!
    editIntervalIntCell(table_id: Int!, column_idx: Int!,
   row_id: Int!, data: Int!): MutationResult!
    clearCell(table_id: Int!, column_idx: Int!, row_id: Int!)
   : MutationResult!

    createCartesianProduct(first_table_id: Int!
   second_table_id: Int!): MutationResult!
}
```

Лістинг 2: GraphQL schema

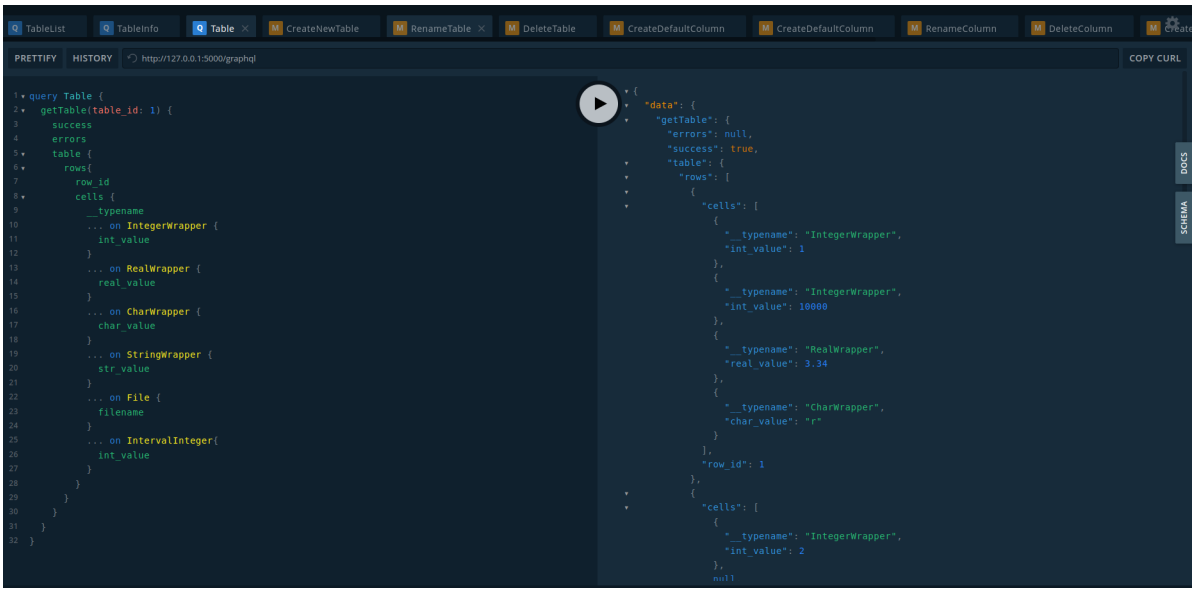

Рис. 16: GraphQL. Запит на отримання всієї таблиці

9 Етап №8

Розгортання gRpc сервера з 5-го етапу.

```
ENTRYPOINT export LD_LIBRARY_PATH=/usr/local/lib \
           && ./build/gRpcDatabaseServer --load-type="custom"
    --database-file="../custom_test_db_projects/my.cdb" --
   server-ip="0.0.0.0" --server-port="50051"
```

Лістинг 3: фрагмент Dockerfile для деплою сервера

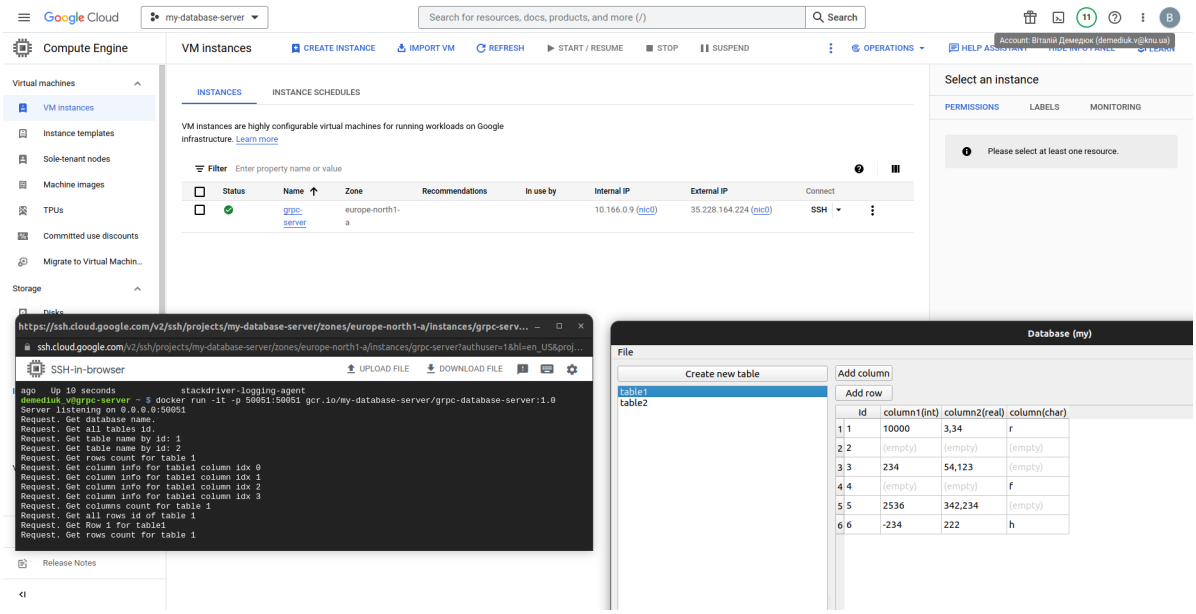

Рис. 17: Cloud Google Platform