Міністерство освіти та науки України
Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського”
Факультет прикладної математики
Кафедра системного програмування і спеціалізованих комп’ютерних систем

РОЗРАХУНКОВО-ГРАФІЧНА РОБОТА

з дисципліни

“Бази даних”

“Створення додатку бази даних, орієнтованого на взаємодію з СУБД PostgreSQL”

Виконав: Молчембаєв Я. А.
Студент групи КВ-23
Telegram: @m3r7v4i444c0m
Github: kpi_databases/rgr at main · r1pth3sl1t/kpi_databases

Київ 2024

Загальне завдання роботи

1. Реалізувати функції перегляду, внесення, редагування та вилучення даних у таблицях бази даних, створених у лабораторній роботі №1, засобами консольного інтерфейсу.
2. Передбачити автоматичне пакетне генерування «рандомізованих» даних у базі.
3. Забезпечити реалізацію пошуку за декількома атрибутами з двох та більше сутностей одночасно: для числових атрибутів – у рамках діапазону, для рядкових – як шаблон функції LIKE оператора SELECT SQL, для логічного типу – значення True/False, для дат – у рамках діапазону дат.
4. Програмний код виконати згідно шаблону MVC (модель-подання-контролер).

Опис структури бази даних

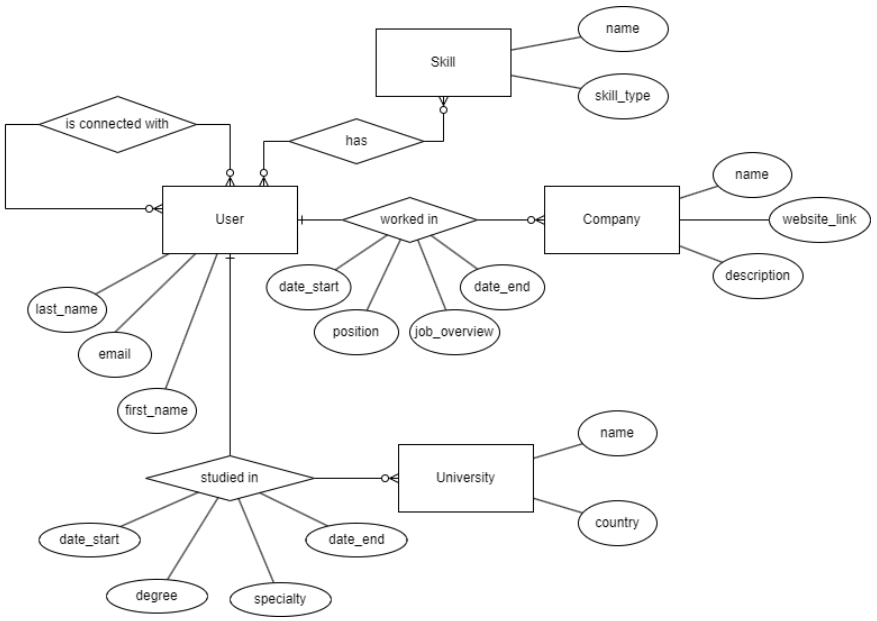
Тема бази даних: Соціальна мережа для професійних зв'язків.

Сутності бази даних:

- User - сутність користувача соціальної мережі.
- Skill - сутність навички користувача. Кожен користувач може налаштувати у своєму профілі набір навичок, якими він володіє, для підбору доступних вакансій або для роботодавця.
- Company - сутність компанії. Користувач може мати попередній досвід роботи в певних компаніях і опублікувати інформацію про цей досвід у своєму профілі.
- University - сутність навчального закладу. Користувач може мати освіту і опублікувати інформацію про неї у своєму профілі.

Зв’язки моделі:

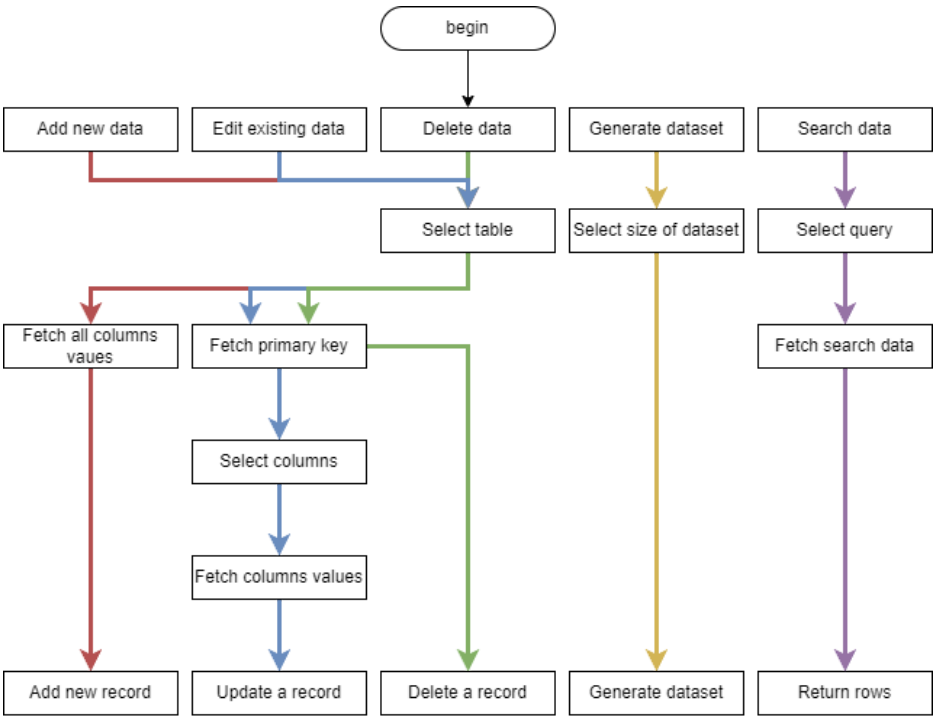
- M:N користувач - користувач
- 0:N користувач - навчальний заклад
- 0:N користувач - компанія
- M:N користувач - навичка



Опис меню користувача

1. Add new data - опція для додавання даних до таблиць. При виборі опції користувачу пропонується обрати таблицю та ввести усі атрибути нового запису.
2. Edit existing data - опція для редагування існуючих записів у таблицях. При виборі опції користувачу пропонується обрати таблицю, ввести первинний ключ запису, який необхідно відредагувати, вибрати атрибути, значення яких необхідно відредагувати, та ввести нові значення.
3. Delete data - опція для видалення записів у таблицях. При виборі опції користувачу пропонується обрати таблицю та ввести первинний ключ запису для видалення.
4. Generate dataset - опція для генерації даних у базі даних. При виборі опції користувачу пропонується ввести кількість записів, що будуть згенеровані.
5. Search data - опція для пошуку даних у таблиці. При виборі опції користувачу пропонується обрати тип пошукового запиту.

0. Вихід з програми



Використані технології

Для написання додатку було використано мову Go версії 1.22.4. Для взаємодії з базою даних було використано бібліотеку pq.

Хід виконання роботи

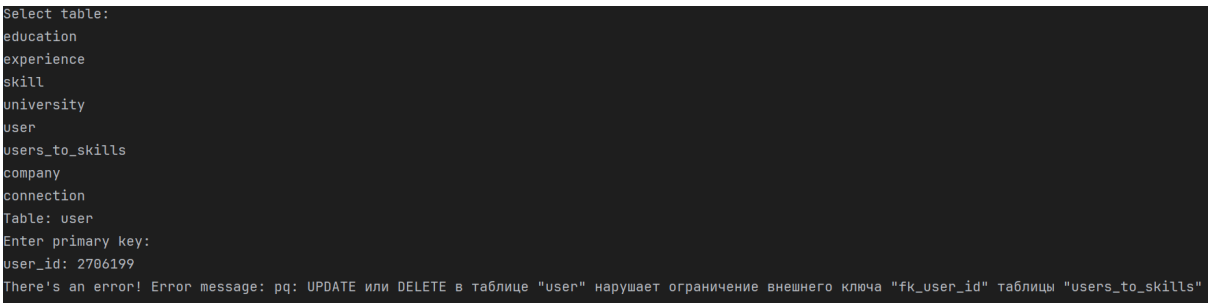
Завдання 1

Оскільки загальноприйнятою практикою є покладання якомога більшої роботи на СУБД, то для обробки помилкових ситуацій, наприклад, невідповідності даних або порушення обмежень первинних/зовнішніх ключів, було обрано перехоплення помилок від СУБД PostgreSQL. В мові Go відсутня конструкція try..catch, проте для обробки помилок як промисловий стандарт використовується повернення об’єкту типу error з функцій.

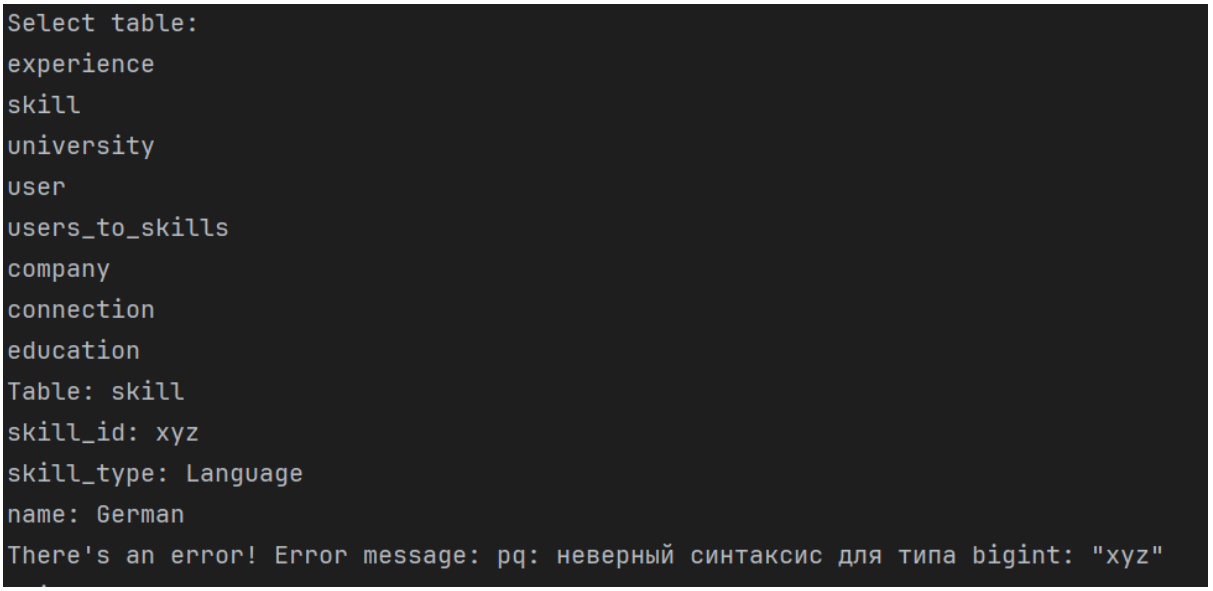
Приклади невдалих запитів до сервера:

```
Table: connection
Enter primary key:
u1: 1
u2: 2
Select columns or type '-' to end selecting:
u2
u1
Column:
u2
-
u2: 3
There's an error! Error message: pq: повторяющееся значение ключа нарушает ограничение уникальности "connection_pkey"
```

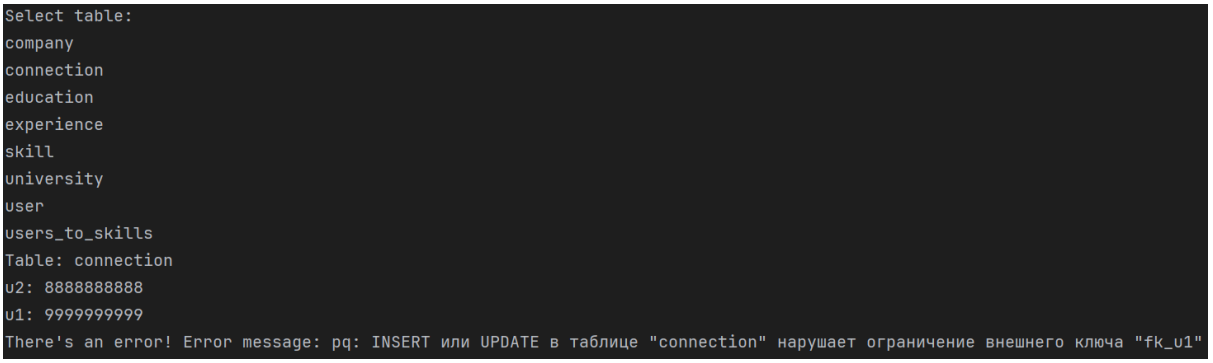
порушення унікальності первинного ключа при редагуванні: запис {u1=1,u2=3} вже присутній в таблиці connection



порушення обмеження зовнішнього ключа при спробі видалення



порушення відповідності типів при спробі додавання нового запису в таблицю



порушення зовнішнього ключа при спробі додавання нового запису в таблицю: записів користувачів з такими ідентифікаторами не існує

Приклади вдалих запитів до сервера:

```
Select table:
experience
skill
university
user
users_to_skills
company
connection
education
Table: skill
skill_id: 123
skill_type: Language
name: German
Main page
```

	skill_id [PK] bigint	name character varying (40)	skill_type character varying (16)
1	1	C++	Technical
2	2	Communication	Soft
3	3	English	Language
4	4	Responsibility	Soft
5	5	Docker	Technical
6	6	Proactive	Soft
7	7	Helpful	Soft
8	8	SQL	Technical
9	9	Data analysis	Technical
10	123	German	Language

вдале додавання запису до таблиці skill

```
Table: skill
Enter primary key:
skill_id: 123
Select columns or type '-' to end selecting:
skill_id
skill_type
name
Column:
name
-
name: French
```

	skill_id [PK] bigint	name character varying (40)	skill_type character varying (16)
1	1	C++	Technical
2	2	Communication	Soft
3	3	English	Language
4	4	Responsibility	Soft
5	5	Docker	Technical
6	6	Proactive	Soft
7	7	Helpful	Soft
8	8	SQL	Technical
9	9	Data analysis	Technical
10	123	French	Language

вдале редагування запису в таблиці skill

```
Select table:
education
experience
skill
university
user
users_to_skills
company
connection
Table: skill
Enter primary key:
skill_id: 123
```

	skill_id [PK] bigint	name character varying (40)	skill_type character varying (16)
1	1	C++	Technical
2	2	Communication	Soft
3	3	English	Language
4	4	Responsibility	Soft
5	5	Docker	Technical
6	6	Proactive	Soft
7	7	Helpful	Soft
8	8	SQL	Technical
9	9	Data analysis	Technical

вдале видалення запису з таблиці skill

Завдання 2

Запит для генерації випадкових даних до таблиці user:

```
insert into public.user(user_id, first_name, last_name, email)
select * from (select row_number() over() as user_id,
                first_name,
                last_name,
                lower(first_name || '_' || last_name) || '@mail.com' as
email
from unnest(array['Pavlo', 'Ruslan', 'Anna',
                  'Ivan', 'Yulia', 'Anastasia',
                  'Volodymyr', 'Sofia', 'Anton',
                  'Olexiy', 'Olexandr', 'Ivanna',
                  'Bohdan', 'Oleh', 'Fedir',
                  'Hryhorii', 'Serhii', 'Dmytro',
                  'Polina', 'Kateryna', 'Svitlana',
                  'Stepan', 'Ostap', 'Halyna',
                  'Viktor', 'Kyrylo', 'Roman']) as first_name
cross join (select generate_series(1, 10000)) as user_id
cross join unnest(array['Ivanenko', 'Petrenko', 'Shevchenko',
                        'Kuznets', 'Sobol', 'Mazur',
                        'Kvitka', 'Sydorenko', 'Koval',
                       'Zhuk', 'Tkach', 'Tkachuk',
                        'Vlasenko', 'Tymoshenko', 'Ostapenko',
                       'Rudenko','Moroz','Petrenko',
                        'Pavlenko','Vasilenko', 'Kryvenko',
                       'Shpak', 'Los', 'Shvets',
                        'Bondar', 'Savchenko', 'Korol']) as last_name) as
users
where user_id not in (select user_id from public.user)
order by random()
limit 100000;
```

Data Output | Messages | Notifications

	user_id bigint	first_name text	last_name text	email text
1	1010025	Oleh	Koval	oleh_koval@mail.com
2	2405461	Polina	Kuznets	polina_kuznets@mail.com
3	4319504	Volodymyr	Moroz	volodymyr_moroz@mail.com
4	1066791	Olexiy	Kryvenko	olexiy_kryvenko@mail.com
5	1580011	Olexiy	Bondar	olexiy_bondar@mail.com
6	5692363	Bohdan	Kvitka	bohdan_kvitka@mail.com
7	2840611	Hryhorii	Shpak	hryhorii_shpak@mail.com
8	6026416	Polina	Rudenko	polina_rudenko@mail.com
9	1082318	Dmytro	Los	dmytro_los@mail.com
10	534760	Fedir	Bondar	fedir_bondar@mail.com
11	7104568	Dmytro	Kuznets	dmytro_kuznets@mail.com
12	4252206	Kyrylo	Shevchenko	kyrylo_shevchenko@mail.com
13	2627154	Svitlana	Korol	svitlana_korol@mail.com
14	6351016	Kyrylo	Shpak	kyrylo_shpak@mail.com
15	1418410	Polina	Pavlenko	polina_pavlenko@mail.com
16	6509001	Polina	Shevchenko	polina_shevchenko@mail.com

✓ Successfully run. Total query runtime: 19 secs 927 msec. 100000 rows affected.

✓ Query returned successfully in 69 msec.

Запит для генерації зв’язків між користувачами:

```
insert into "connection"(u1, u2)
select u1t.user_id as u1, u2t.user_id as u2 from "user" as u1t
cross join "user" as u2t
where u1t.user_id != u2t.user_id
and not(
      u1t.user_id in (select u1 from connection) and
      u2t.user_id in (select u2 from connection)
      )
order by random()
limit 100000;
```

	u1 bigint	u2 bigint
1	2415336	7123271
2	3959485	4146369
3	3517144	767172
4	6122243	4146369
5	7123271	5130518
6	3115260	6775520
7	4543599	3066771
8	1086786	2954938
9	5130518	767172
10	3959485	3517144
11	2810303	4153107
12	1105158	1459

Successfully run. Total query runtime: 87 msec. 14635 rows affected.

Запит для генерації навичок, якими володіють користувачі:

```
insert into users_to_skills(user_id, skill_id)
    select user_id, skill_id from "user"
    cross join skill
    where not(user_id in (select user_id from "users_to_skills")
                  and
              skill_id in (select skill_id from "users_to_skills"))
    order by random()
    limit 100000;
```

Data Output Messages Notifications

	user_id bigint	skill_id bigint
1	1955510	6
2	6986440	8
3	5485858	7
4	5653668	2
5	1459	3
6	1086786	1
7	4704380	7
8	45346	1
9	5130518	7

Successfully run. Total query runtime: 62 msec. 909 rows affected.

Завдання 3

Для пошуку даних у таблиці було реалізовано 3 запити, кожен з яких об’єднує таблиці, фільтрує записи та групує їх.

Перший запит виконує пошук користувачів за іменем та прізвищем та повертає ідентифікатор, ім’я, прізвище та кількість користувачів, з якими він з'єднаний:

```
select user_id, first_name, last_name, count(u1) as connections_num
        from (select * from "user"
            left join "connection"
            on "user".user_id = "connection".u1
            union
            select * from "user"
            left join "connection"
            on "user".user_id = "connection".u2) as users
        where first_name like ‘Olexandr’ and last_name like ‘Vlasenko’
        group by user_id, first_name, last_name
```

```
Select searching mode:
1. Search users by first name and last name
2. Search users by connections number range
3. Search users and skills amount by specific skill type
1
First name: Olexandr
Last name: Vlasenko
Time: 3
Columns:
 user_id |      first_name |       last_name | connections_num
 1318315 |        Olexandr |        Vlasenko |               3
```

Другий запит шукає користувачів за заданим діапазоном:

```
select user_id, first_name, last_name, count(u1) as connections_num
        from (select * from "user"
            left join "connection"
            on "user".user_id = "connection".u1
            union
            select * from "user"
            left join "connection"
            on "user".user_id = "connection".u2) as users
        group by user_id, first_name, last_name
        having count(u1) between 2 and 4
```

```
Select searching mode:
1. Search users by first name and last name
2. Search users by connections number range
3. Search users and skills amount by specific skill type
2
min: 2
max: 4
Time: 3
Columns:
 user_id |      first_name |       last_name | connections_num
 5495058 |          Stepan |        Petrenko |               4
 1318315 |        Olexandr |        Vlasenko |               3
       2 |            Mark |            Ford |               4
 2706199 |       Anastasia |         Rudenko |               4
 3287650 |          Stepan |           Shpak |               2
 1630600 |        Svitlana |         Rudenko |               2
 6959473 |          Serhii |          Kvitka |               2
       1 |            John |           Smith |               4
```

Третій запит шукає користувачів, які володіють навичками такого типу, який задає користувач:

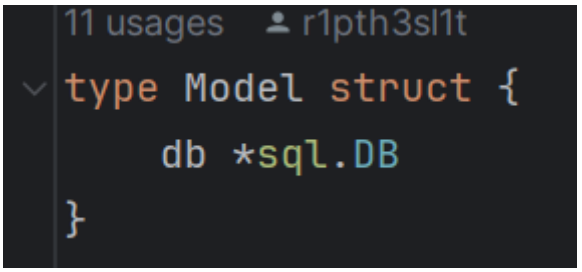
```
select "user".user_id, first_name, last_name, count(*) as skill_count
        from "user"
        join users_to_skills
        on "user".user_id = users_to_skills.user_id
        join skill
        on users_to_skills.skill_id = skill.skill_id
        where skill_type ilike ‘Soft’
        group by "user".user_id, first_name, last_name
```

```
Select searching mode:
1. Search users by first name and last name
2. Search users by connections number range
3. Search users and skills amount by specific skill type
3
Skill type: Soft
Time: 4
Columns:
 user_id |      first_name |       last_name |     skill_count
       1 |            John |           Smith |               1
       2 |            Mark |            Ford |               3
       3 |           Jenna |            Jane |               3
  841846 |          Stepan |        Vlasenko |               4
 1318315 |        Olexandr |        Vlasenko |               4
 1630600 |        Svitlana |         Rudenko |               4
 2340980 |       Anastasia |       Savchenko |               4
 2415336 |       Anastasia |          Shvets |               4
 2706199 |       Anastasia |         Rudenko |               4
```

Завдання 4

Ілюстрації програмного коду модуля “Model”

“Model” представляє з себе структуру з полем db, за допомогою якого додаток взаємодіє з базою даних. Згідно шаблону MVC, модель інкапсулює в собі бізнес-логіку додатку, тому тут обробляються безпосередньо запити до бази.



При створенні об’єкту моделі здійснюється з’єднання з СУБД, у випадку помилки повертається об’єкт типу error.

```
func New() (*Model, error) {
    config, err := os.Open( name: "config.json")
    if err != nil {
        return nil, err
    }
    var dbc DatabaseCredentials
    decoder := json.NewDecoder(config)
    err = decoder.Decode(&dbc)

    if err != nil {
        return nil, err
    }

    m := new(Model)
    connStr := fmt.Sprintf( format: "user=%s host=%s port=%s dbname=%s password=%s sslmode=disable",
        dbc.User, dbc.Host, dbc.Port, dbc.Dbname, dbc.Password)

    db, err := sql.Open( driverName: "postgres", connStr)

    if err != nil {
        return nil, err
    }
    m.db = db
```

За допомогою методу Close модель коректно завершує сесію мережевого з’єднання з СУБД. Цей метод викликається в контролері в методі Destroy, який в свою чергу викликається в функції main з ключовим словом defer.

```
func (m *Model) Close() {
    m.db.Close()
}
```

Метод FetchTableData отримує з СУБД дані про таблиці і їх атрибути та повертає хеш-карту з цими даними.

```
func (m *Model) FetchTableData() map[string][]string {
    rows, err := m.db.Query(queries.GetFetchingTablesDataQuery())
    if err != nil {
        fmt.Println(err)
    }
    tables := make(map[string][]string)
    for rows.Next() {
        var tableName, column string
        rows.Scan(&tableName, &column)
        tables[tableName] = append(tables[tableName], column)
    }
    return tables
}
```

Метод FetchTablePrimaryKeys отримує дані про первинні ключі таблиць та повертає хеш-карту з цими даними.

```
func (m *Model) FetchTablePrimaryKeys() map[string][]string {
    rows, err := m.db.Query(queries.GetFetchingPrimaryKeysQuery())
    if err != nil {
        fmt.Println(err)
    }
    tables := make(map[string][]string)

    for rows.Next() {

        var tableName, column string
        rows.Scan(&tableName, &column, nil)

        tables[tableName] = append(tables[tableName], column)
    }
    return tables
}
```

Метод Insert готує INSERT-запит та посилає його до СУБД, у випадку помилки повертає об’єкт error

```go
func (m *Model) Insert(table string, data map[string]string) error {

    query, values := queries.PrepareInsertQuery(table, data)

    _, err := m.db.Query(query, values...)

    return err
}
```

```go
func PrepareInsertQuery(table string, data map[string]string) (string, []any) {
    query := "INSERT INTO \"" + table + "\"("
    var values []any
    for column := range data {
        query += column + ","
        values = append(values, data[column])
    }
    query = query[:len(query)-1]
    query += ") VALUES ("
    for i := 1; i <= len(data); i++ {
        query += "$"
        query += strconv.Itoa(i)
        query += ","
    }
    query = query[:len(query)-1]
    query += ")"
    return query, values
}
```

Метод Update готує UPDATE-запит та посилає його до СУБД, у випадку помилки повертає об’єкт error

```go
func (m *Model) Update(tableName string, data map[string]string, pkey map[string]string) error {
    query, values := queries.PrepareUpdateQuery(tableName, data, pkey)

    _, err := m.db.Query(query, values...)

    return err
}
```

```go
func PrepareUpdateQuery(table string, data map[string]string, pkey map[string]string) (string, []any) {
    query := "UPDATE \"" + table
    query += "\" SET "
    i := 1
    var values []any
    for column := range data {
        query += column + " = $" + strconv.Itoa(i) + ","
        values = append(values, data[column])
        i++
    }
    query = query[:len(query)-1]

    query += " WHERE "
    for column := range pkey {
        query += column + " = $" + strconv.Itoa(i) + " AND "
        values = append(values, pkey[column])
        i++
    }
    query = query[:len(query)-4]
    return query, values
}
```

Метод Delete готує DELETE-запит та посилає його до СУБД, у випадку помилки повертає об’єкт error

```go
func (m *Model) Delete(table string, pkey map[string]string) error {
    query, values := queries.PrepareDeleteQuery(table, pkey)

    _, err := m.db.Query(query, values...)
    return err
}
```

```go
func PrepareDeleteQuery(table string, pkey map[string]string) (string, []any) {
    var values []any
    query := "DELETE FROM \"" + table + "\" WHERE "
    i := 1
    for key := range pkey {
        query += key + " = $" + strconv.Itoa(i) + " AND "
        values = append(values, pkey[key])
        i++
    }

    query = query[:len(query)-4]

    return query, values
}
```

Метод GenerateDataSet посилає запит на генерацію випадкових даних до таблиць user, users_to_skills, connection для заданого значення size

```go
func (m *Model) GenerateDataSet(size int) error {
    _, err := m.db.Query(queries.GetUserGeneratingQuery(), size)
    if err != nil {
        return err
    }
    _, err = m.db.Query(queries.GenerateSkillsQuery(), size)
    if err != nil {
        return err
    }
    _, err = m.db.Query(queries.GenerateConnectionQuery(), size)
    if err != nil {
        return err
    }
    return err
}
```