**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №2

«Хранение статических данных PostgreSQL»

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-32Б |  | преподаватель каф. ИУ5 |
| Старкин Александр |  | Гапанюк Ю.Е. |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2024 г.

### Задание:

Разработать даталогическую модель на основе которой будет создана база данных в PostgreSQL, хранящая статическую информацию о группах, кафедрах, факультетах, курсах, филиалах Вуза МГТУ им. Баумана.

Статическое хранение необходимо для мгновенного и надежного доступа к данным.

1.Сущность faculties должна содержать колонки:

* id (ПК) – int64
* title - string

2.Сущность courses должна содержать колонки:

* id (ПК) – int64
* title – string

3.Сущность cathedras должна содержать колонки:

* id (ПК) – int64
* title – string

4.Сущность fillials должна содержать колонки:

* id (ПК) – int64
* title – string

5.Сущность groups должна содержать колонки:

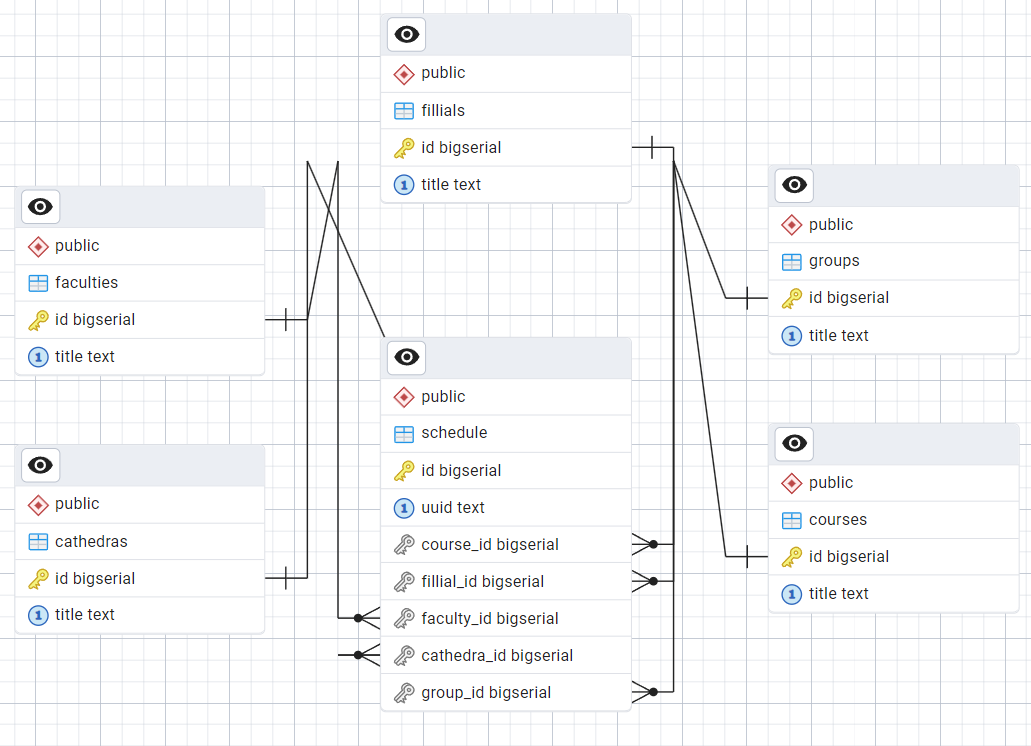
* id (ПК) – int64
* title – string

Данные таблицы призваны убрать повторяющие строки данных из главной таблицы schedule хранящую связи id всех вышеуказанных таблиц.

3.Сущность schedule должна содержать колонки:

* id (ПК) – int64
* uuid – string
* course\_id (ВК) – int64
* faculty\_id (ВК) – int64
* fillial\_id (ВК) – int64
* cathedra\_id (ВК) – int64
* group\_id (ВК) – int64

Даталогическая модель будет выглядеть так:



**Реализуем работу с БД в go.**

Данные сущности будут обновляться отдельным скриптом раз в пол года (синхронизация происходит с помощью http запроса к API МГТУ им. Баумана, что хранит информацию о наличии всех групп, кафедр, факультетов, филиалов вуза).

Текст программы

Код для обновления данных бд.

package database

import (

    "context"

    "encoding/json"

    "fmt"

    "github.com/Robocotik/IT-s\_Friends/internal/models/entities"

    "github.com/Robocotik/IT-s\_Friends/internal/services/output"

    "io/ioutil"

    "net/http"

    "os"

    "github.com/jackc/pgx/v5"

    "github.com/mymmrac/telego"

)

func ParseAllSchdule(ctx context.Context, conn \*pgx.Conn, bot \*telego.Bot, chatID int64) error {

    resp, err := http.NewRequestWithContext(ctx, http.MethodGet, "https://lks.bmstu.ru/lks-back/api/v1/structure", nil)

    var group\_id, cathedra\_id, faculty\_id, fillial\_id, course\_id int64

*if* err != nil {

        fmt.Println("Ошибка при get запросе :", err)

*return* err

    }

*defer* resp.Body.Close()

    data, err := ioutil.ReadAll(resp.Body)

*if* err != nil {

        fmt.Println("Ошибка при чтении DATA:", err)

*return* err

    }

    var result entities.Final

    err = json.Unmarshal(data, &result)

*for* \_, filial := *range* result.Data.Children {

        err := conn.QueryRow(context.Background(),

            `

    WITH ins AS (

        INSERT INTO fillials (title) VALUES($1)

        ON CONFLICT (title) DO NOTHING

        RETURNING id

    )

    SELECT COALESCE((SELECT id FROM ins), (SELECT id FROM fillials WHERE title = $1));

    `,

            filial.Abbr).Scan(&fillial\_id)

*if* err != nil {

            output.RiseError(bot, chatID, err)

        }

*for* \_, faculty := *range* filial.Children {

            err := conn.QueryRow(context.Background(),

                `

    WITH ins AS (

        INSERT INTO faculties (title) VALUES($1)

        ON CONFLICT (title) DO NOTHING

        RETURNING id

    )

    SELECT COALESCE((SELECT id FROM ins), (SELECT id FROM faculties WHERE title = $1));

    `,

                faculty.Abbr).Scan(&faculty\_id)

*if* err != nil {

                output.RiseError(bot, chatID, err)

            }

*for* \_, cathedra := *range* faculty.Children {

                err := conn.QueryRow(context.Background(),

                    `

    WITH ins AS (

        INSERT INTO cathedras (title) VALUES($1)

        ON CONFLICT (title) DO NOTHING

        RETURNING id

    )

    SELECT COALESCE((SELECT id FROM ins), (SELECT id FROM cathedras WHERE title = $1));

    `,

                    cathedra.Abbr).Scan(&cathedra\_id)

*if* err != nil {

                    output.RiseError(bot, chatID, err)

                }

*for* \_, course := *range* cathedra.Children {

                    err := conn.QueryRow(context.Background(),

                        `

    WITH ins AS (

        INSERT INTO courses (title) VALUES($1)

        ON CONFLICT (title) DO NOTHING

        RETURNING id

    )

    SELECT COALESCE((SELECT id FROM ins), (SELECT id FROM courses WHERE title = $1));

    `,

                        course.Abbr).Scan(&course\_id)

*if* err != nil {

                        output.RiseError(bot, chatID, err)

                    }

*for* \_, group := *range* course.Children {

                        err := conn.QueryRow(context.Background(),

                            `

    WITH ins AS (

        INSERT INTO groups (title) VALUES($1)

        ON CONFLICT (title) DO NOTHING

        RETURNING id

    )

    SELECT COALESCE((SELECT id FROM ins), (SELECT id FROM groups WHERE title = $1));

    `,

                            group.Abbr).Scan(&group\_id)

*if* err != nil {

                            output.RiseError(bot, chatID, err)

                        }

                        \_, err = conn.Exec(context.Background(),

                            "INSERT INTO schedule (uuid, course\_id, fillial\_id, faculty\_id, cathedra\_id, group\_id) VALUES($1, $2, $3, $4, $5, $6) on conflict (uuid) do nothing",

                            group.Uuid, course\_id, fillial\_id, faculty\_id, cathedra\_id, group\_id)

*if* err != nil {

                            fmt.Fprintf(os.Stderr, "Query failed: %v\n", err)

                            output.RiseError(bot, chatID, err)

*return* err

                        }

                    }

                }

            }

        }

    }

*if* err != nil {

        fmt.Println("Ошибка при декодировании JSON:", err)

*return* err

    }

*return* nil

}

Код для получения статических данных с бд.

package postgres

import (

    "github.com/Robocotik/IT-s\_Friends/internal/models/structures"

    "github.com/Robocotik/IT-s\_Friends/internal/services/output"

    "context"

    "fmt"

    "os"

    "github.com/mymmrac/telego"

)

func (psql Postgres) GetFaculties(bot \*telego.Bot, chatID int64, identity \*structures.Identity) []string {

    var res []string

    var facultyTitle string

    rows, err := psql.Conn.Query(context.Background(), `

    SELECT DISTINCT f.title

    FROM faculties f

    JOIN schedule s ON f.id = s.faculty\_id

    JOIN fillials fi ON s.fillial\_id = fi.id

    WHERE fi.title = $1;

`, identity.Filial)

*if* err != nil {

        fmt.Fprintf(os.Stderr, "Query failed in getting faculties: %v\n", err)

        output.RiseError(bot, chatID, err)

*return* []string{""}

    }

*defer* rows.Close()

*for* rows.Next() {

        err := rows.Scan(&facultyTitle)

*if* err != nil {

            fmt.Fprintf(os.Stderr, "Failed to scan faculty title: %v\n", err)

            output.RiseError(bot, chatID, err)

*return* []string{""}

        }

        res = append(res, facultyTitle)

    }

*if* err := rows.Err(); err != nil {

        fmt.Fprintf(os.Stderr, "Error occurred during row iteration: %v\n", err)

        output.RiseError(bot, chatID, err)

*return* []string{""}

    }

    fmt.Println("I FOUND FACULTIES: ", res)

*return* res

}