

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені Ігоря Сікорського»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

# Кафедра системного програмування та спеціалізованих комп’ютерних систем

**Лабораторна робота №3**

з дисципліни

**«Бази даних і засоби управління»**

Виконав: студент ІII курсу

ФПМ групи КВ-73

Тарновський А.М.

Перевірив(ла):

Київ – 2019

**Засоби оптимізації роботи СУБД PostgreSQL**

*Метою роботи* є здобуття практичних навичок використання засобів

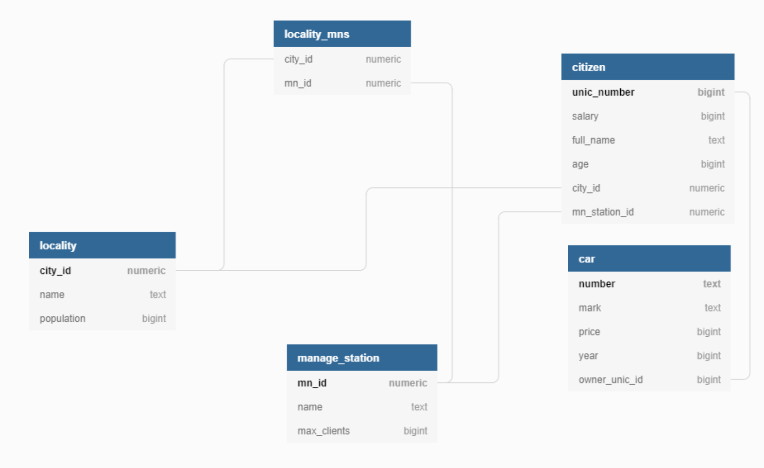
оптимізації СУБД PostgreSQL.

**Варіант 15**

Проаналізувати індекси GIN, BRIN.

**Завдання 1**

Схема бази даних “ДПС”



Для перетворення програми, створеної в 2 лабораторній роботі, у вигляд об’єктно-реляційної моделі використовую бібліотеку TypeORM для typescript, що створена для вирішення задач об’єктно-реляційного відображення.

**Опис програми**

Програма написана на мові TypeScript. В якості orm бібліотеки було обрано TypeORM.

Основні компоненти програми:

1. [CRUDRepository.ts](#СRUD) – містить опис класу, який реалізує crud операції.
2. [Entities.ts](#Entities) – містить опис сутностей та реалізацію відношень сутностей.
3. [Controller.ts](#Controllers) – містить функцію, яка створює колекцію об’єктів, які реалізують зв'язок між View та Repository компонентами.
4. [Repositoty.ts](#Repository) – містить функцію, яка створює колекцію об’єктів, які реалізують функціонал роботи з базою.
5. [View.ts](#View) – містить функції, які реалізують View компоненту.

**CRUDRepository.ts**

@EntityRepository()

export class ORMRepository<T> extends Repository<T> {

private readonly dbName;

private readonly entity

constructor({ entity, dbName }) {

super();

this.entity = entity;

this.dbName = dbName;

}

public async select(options) {

const connection = getConnection().getRepository(this.entity)

const builder = connection.createQueryBuilder(this.dbName).getMany();

return await builder;

}

public async delete(conditions) {

const connection = getConnection().getRepository(this.entity)

const builder = connection.createQueryBuilder(this.dbName)

.delete()

.from(this.entity)

.where(conditions[0])

.execute();

return await builder;

}

public async update(options) {

const { attributes, conditions } = options;

const connection = getConnection().getRepository(this.entity)

const builder = connection.createQueryBuilder(this.dbName)

.update(this.entity)

.set(attributes[0])

.where(conditions[0])

.execute();

return await builder;

}

public async insert(options) {

const data = options.reduce((acc, obj) => ({ ...acc, ...obj }), {});

const connection = getConnection().getRepository(this.entity)

const builder = connection.createQueryBuilder(this.dbName)

.insert()

.into(this.entity)

.values([data])

.execute();

return await builder;

}

public async retrieve(options) {

const connection = getConnection().getRepository(this.entity)

const builder = connection

.createQueryBuilder(this.dbName)

.leftJoinAndSelect('\*', 'city')

.getMany();

return await builder;

}

}

**Entities.ts**

@Entity({ name: 'locality' })

export class LocalityEntity {

@PrimaryGeneratedColumn({ name: 'city\_id' })

id: number;

@Column({ name: 'name', type: "varchar", length: 25 })

name: string;

@Column({ name: 'population', type: "integer", default: false })

population: number;

}

@Entity('car')

export class CarEntity {

@PrimaryGeneratedColumn({ name: 'number', type: "varchar", length: 10})

number: string;

@Column({ name: 'mark', type: "varchar", length: 25})

mark: string;

@Column({ name: 'price' })

mark: number;

@Column({ name: 'year' })

mark: number;

@OneToOne(type => CitizenEntity)

@JoinColumn()

Owner\_unic\_id: number;

}

@Entity('citizen')

export class CitizenEntity {

@PrimaryGeneratedColumn({ name: ‘unic\_number’})

Unic\_number: number;

@Column({ name: 'salary' })

salary: number;

@Column({ name: 'full\_name', type: "varchar", length: 25 })

Full\_name: string;

@OneToOne(type => LocalityEntity)

@JoinColumn()

city\_id: number;

@OneToOne(type => MnStationEntity)

@JoinColumn()

mn\_station\_id: number;

}

@Entity('manage\_station')

export class MnStationEntity {

@PrimaryGeneratedColumn({ name: 'mn\_id' })

mn\_id: number;

@Column({ name: 'name', type: "varchar", length: 50 })

name: string;

@Column({ name: 'max\_clients' })

max\_clients: number;

}

**Сontrollers.ts**

export const CAR\_CONTROLLER\_TYPE = 'carController';

export const CITIZEN\_CONTROLLER\_TYPE = 'citizenController';

export const LOCALITY\_CONTROLLER\_TYPE = 'localityController';

export const MANAGESTATION\_CONTROLLER\_TYPE = 'managestationController';

export const controllerModule = {

[CAR\_CONTROLLER\_TYPE]: new Controller({ repository: repositoryModule[CAR\_REPOSITORY\_TYPE] }),

[CITIZEN\_CONTROLLER\_TYPE]: new Controller({ repository: repositoryModule[CITIZEN\_REPOSITORY\_TYPE] }),

[LOCALITY\_CONTROLLER\_TYPE]: new Controller({ repository: repositoryModule[LOCALITY\_REPOSITORY\_TYPE] }),

[MANAGESTATION\_CONTROLLER\_TYPE]: new Controller({ repository: repositoryModule[MANAGESTATION\_REPOSITORY\_TYPE] })

};

**Repository.ts**

export const CAR\_REPOSITORY\_TYPE = 'carRepository';

export const CITIZEN\_REPOSITORY\_TYPE = 'citizenRepository';

export const LOCALITY\_REPOSITORY\_TYPE = 'localityRepository';

export const MANAGESTATION\_REPOSITORY\_TYPE = 'managestationRepository';

function createConection() {

const connection = new Pool(poolConnection);

return () => connection;

}

const dbConnection = createConection();

export const repositoryModule = {

[CAR\_REPOSITORY\_TYPE]: new CRUDRepository({ database: dbConnection(), dbName: CAR}),

[CITIZEN\_REPOSITORY\_TYPE]: new CRUDRepository({ database: dbConnection(), dbName: CITIZEN }),

[LOCALITY\_REPOSITORY\_TYPE]: new CRUDRepository({ database: dbConnection(), dbName: LOCALITY }),

[MANAGESTATION\_REPOSITORY\_TYPE]: new CRUDRepository({ database: dbConnection(), dbName: MANAGESTATION })

};

**View.ts**

const updateTableRouting = (options) => ({

[CAR]: async () => await controllerModule[CAR\_CONTROLLER\_TYPE].updateData(options),

[CITIZEN]: async () => await controllerModule[CITIZEN\_CONTROLLER\_TYPE].updateData(options),

[LOCALITY]: async () => await controllerModule[LOCALITY\_CONTROLLER\_TYPE].updateData(options),

[MANAGESTATION]: async () => await controllerModule[MANAGESTATION\_CONTROLLER\_TYPE].updateData(options),

});

const tableAttributesToSearch = {

[CAR]: [NUMBER],

[CITIZEN]: [UNIC\_NUMBER, FULL\_NAME, CITY\_ID, MN\_STATION\_ID],

[LOCALITY]: [CITY\_ID, NAME],

[NAMAGESTATION]: [MN\_ID, NAME]

};

const tableAttributesToUpdate = {

[CAR]: [NUMBER, OWNER\_UNIC\_ID],

[CITIZEN]: [SALARY, FULL\_NAME, CITY\_ID, MN\_STATION\_ID],

[LOCALITY]: [CITY\_ID, NAME, POPULATION],

[MANAGESTATION]: [NAME, MAX\_CLIENTS]

};

export async function updateView(): Promise<void> {

const table = await inquirer

.prompt([{

type: 'rawlist',

name: ITEM,

message: 'Choose table',

choices: [ CAR, CITIZEN, LOCALITY, MANAGESTATION],

}]);

const attributeToUpdate = await inquirer

.prompt([{

type: 'rawlist',

name: ITEM,

message: 'Choose attribute to update',

choices: tableAttributesToUpdate[table[ITEM]],

}]);

const valueToUpdate = await inquirer

.prompt([{

type: 'input',

name: ITEM,

message: `Attribute ${attributeToUpdate[ITEM]} =`,

}]);

const attributes = [{ [attributeToUpdate[ITEM]]: valueToUpdate[ITEM] }];

const attribute = await inquirer

.prompt([{

type: 'rawlist',

name: ITEM,

message: 'Choose condition attribute',

choices: tableAttributesToSearch[table[ITEM]],

}]);

const value = await inquirer

.prompt([{

type: 'input',

name: ITEM,

message: `Attribute ${attribute[ITEM]} =`,

}]);

const conditions = [{ [attribute[ITEM]]: value[ITEM] }];

await route(deleteTableRouting({conditions, attributes})[table[ITEM]])

}

const tableOrder = [

CAR,

CITIZEN,

LOCALITY,

MANAGESTATION

];

const showRouting = {

[SHOW\_ALL\_TABLES]: () => Promise.all(Object.values(showTableRouting).map(fn => fn())),

[SHOW\_ONE\_TABLE]: () => showOneTableView()

};

const showTableRouting = {

[CAR]: async () => await controllerModule[CAR\_CONTROLLER\_TYPE].showTable(),

[CITIZEN]: async () => await controllerModule[CITIZEN\_CONTROLLER\_TYPE].showTable(),

[LOCALITY]: async () => await controllerModule[LOCALITY\_CONTROLLER\_TYPE].showTable(),

[MANAGESTATION]: async () => await controllerModule[MANAGESTATION\_CONTROLLER\_TYPE].showTable()

};

export async function showOneTableView() {

const answer = await inquirer

.prompt([

{

type: 'rawlist',

name: ITEM,

message: 'DPS',

choices: [ CAR, CITIZEN, LOCALITY, MANAGESTATION],

},

]);

const result = await route(showTableRouting[answer[ITEM]]);

if (result.length) console.table(result);

else console.log(`Table "${answer[ITEM]}" is empty;`);

return;

}

export async function showView(): Promise<void> {

const answer = await inquirer

.prompt([

{

type: 'rawlist',

name: ITEM,

message: 'DPS',

choices: [

SHOW\_ONE\_TABLE,

SHOW\_ALL\_TABLES

],

},

]);

const result = await route(showRouting[answer[ITEM]]);

if (answer[ITEM] !== SHOW\_ONE\_TABLE) {

result.forEach((table, i) => table.length ? console.table(table) : console.log(`Table "${tableOrder[i]}" is empty;`));

}

return;

}

const searchTableRouting = (options) => ({

[CAR]: async () => await controllerModule[CAR\_CONTROLLER\_TYPE].searchData(options),

[CITIZEN]: async () => await controllerModule[CITIZEN\_CONTROLLER\_TYPE].searchData(options),

[LOCALITY]: async () => await controllerModule[LOCALITY\_CONTROLLER\_TYPE].searchData(options),

[MANAGESTATION]: async () => await controllerModule[MANAGESTATION\_CONTROLLER\_TYPE].searchData(options),

});

const tableAttributesToDoSearch = {

[MANAGESTATION]: [NAME],

};

export async function searchView(): Promise<void> {

const table = await inquirer

.prompt([{

type: 'rawlist',

name: ITEM,

message: 'Choose table',

choices: [ MANAGESTATION ],

}]);

const attribute = await inquirer

.prompt([{

type: 'rawlist',

name: ITEM,

message: 'Choose attribute to search',

choices: tableAttributesToDoSearch[table[ITEM]],

}]);

const value = await inquirer

.prompt([{

type: 'input',

name: ITEM,

message: `Search text =`,

}]);

const conditions = {

query: value[ITEM],

attribute: attribute[ITEM]

}

const result = await route(deleteTableRouting(conditions)[table[ITEM]])

if (result.length) {

console.table(table);

} else {

console.log('Nothing found');

}

return;

}

const deleteTableRouting = (options) => ({

[CAR]: async () => await controllerModule[CAR\_CONTROLLER\_TYPE].deleteData(options),

[CITIZEN]: async () => await controllerModule[CITIZEN\_CONTROLLER\_TYPE].deleteData(options)

});

const tableAttributesToDelete = {

[CAR]: [OWNER\_UNIC\_ID, PRICE],

[CITIZEN]: [MN\_STATION\_ID]

};

export async function deleteView(): Promise<void> {

const table = await inquirer

.prompt([{

type: 'rawlist',

name: ITEM,

message: 'Choose table',

choices: [ CAR, CITIZEN],

}]);

const attribute = await inquirer

.prompt([{

type: 'rawlist',

name: ITEM,

message: 'Choose table',

choices: tableAttributesToDelete[table[ITEM]],

}]);

const value = await inquirer

.prompt([{

type: 'input',

name: ITEM,

message: `Attribute ${attribute[ITEM]} = `,

}]);

const conditions = [{ [attribute[ITEM]]: value[ITEM] }];

return await route(deleteTableRouting(conditions)[table[ITEM]])

}

const createTableRouting = (options) => ({

[CAR]: async () => await controllerModule[CAR\_CONTROLLER\_TYPE].insertData(options),

[LOCALITY]: async () => await controllerModule[LOCALITY\_CONTROLLER\_TYPE].insertData(options),

[CITIZEN]: async () => await controllerModule[CITIZEN\_CONTROLLER\_TYPE].insertData(options),

[MANAGESTATION]: async () => await controllerModule[MANAGESTATION\_CONTROLLER\_TYPE].insertData(options)

});