

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій
Кафедра інформаційних технологій та систем

Лу Фен

АНАЛІЗ ТА ПРОЄКТУВАННЯ БАЗИ ДАНИХ ПЕРЕЛІКУ ВИБІРКОВИХ ДИСЦИПЛІН

кваліфікаційна робота

здобувача вищої освіти другого (магістерського) рівня

освітньої програми «Комп'ютерні мережі»

за спеціальністю 123 Комп'ютерна інженерія

Особистий підпис _____ Лу Фен

Науковий керівник _____ Геннадій МОГИЛЬНИЙ,
кандидат технічних наук, доцент
кафедри інформаційних технологій
та систем

Завідувач кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Полтава – 2025

АНОТАЦІЯ

Лу Фен

**Тема: АНАЛІЗ ТА ПРОЄКТУВАННЯ БАЗИ ДАНИХ ПЕРЕЛІКУ
ВИБІРКОВИХ ДИСЦИПЛІН**

Спеціальність: 123 «Комп'ютерна інженерія»

Установа: ЛНУ імені Тараса Шевченка, 2025р.

Магістерська робота містить: загальна кількість сторінок 75, з них 59
Пояснювальна записка, 26 рис., додатки., 21 джерел.

Об'єкт дослідження – система управління переліком вибіркового
дисциплін у закладах вищої освіти

Предмет дослідження –автоматизація визначення переліку вибіркового
дисциплін.

Мета роботи – розробка БД, яка забезпечить створення системи
визначення переліку вибіркового дисциплін.

Результати роботи. Проведено аналіз сучасних типів баз даних. Обрано
реляційну базу даних PostgreSQL. Наведено підходи до створення та
налаштування бази даних. Обрано бази даних. Підготовлено першочергові
інструменти для WEB системи визначення переліку вибіркового дисциплін.

Висновок. В результаті розробки було отримано база даних орієнтована
на роботу у системі визначення переліку вибіркового дисциплін.

Ключові слова. АВТОМАТИЗАЦІЯ, БАЗА ДАНИХ,
ПРОЄКТУВАННЯ БАЗИ ДАНИХ , МОВА JAVA, WEB СИСТЕМА

ABSTRACT

Lu Fen

Theme: ANALYSIS AND DESIGN OF DATABASE LIST OF ELECTIVE DISCIPLINES.

Speciality: 123 "Computer Engineering"

Institution: Luhansk Taras Shevchenko National University (LTSNU), 2025.

Diploma work contains: total number of pages 75, of which 59 Explanatory note, 26 fig., 2 appendices, 21 sources.

A research object is management system of the list of elective disciplines in higher education.

The article of research is automation of determining the list of elective disciplines.

An aim of work is development of a database that will ensure the creation of a system for determining the list of elective disciplines..

Job performances. The analysis of modern types of databases is carried out. The relational database PostgreSQL. Selected approaches to creating and configuring the database. Databases selected. The first tools for the WEB system for determining the list of elective disciplines have been prepared.

Conclusion. As a result of development the database focused on work in system of definition of the list of selective disciplines was received.

Keywords. AUTOMATION, DATABASE, DATABASE DESIGN, JAVA LANGUAGE, WEB SYSTEM

Міністерство освіти і науки України
Державний заклад „Луганський національний університет
імені Тараса Шевченка”
Факультет (інститут) Інститут математики та інформаційних технологій
(повна назва)
Кафедра Інформаційних технологій та систем
(повна назва)
Галузь знань 12, Інформаційні технології
(код, назва)
Напрямок підготовки (спеціальність) 123 «Комп’ютерна інженерія»
(код, назва)

ЗАВДАННЯ
на кваліфікаційну роботу освітньо-кваліфікаційного рівня
« магістр »
(назва рівня)

Студенту Лу Фен
(прізвище, ім'я, по батькові)

Керівник кваліфікаційної роботи Могильний Геннадій Анатолійович
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

1. Тема роботи Аналіз та проєктування бази даних переліку
вибіркових дисциплін

затверджена наказом по університету _____

2. Термін подання студентом закінченої роботи на кафедру _____

3. Вихідні дані до роботи У результаті виконання роботи необхідно
(проекту) повинна бути розроблена база даних, яка забезпечує
роботу ВЕБ системи визначення переліку вибіркових дисциплін
розробка повинна працювати в режимі ВЕБ системи
(визначаються кількісні або (та) якісні показники, яким повинен відповідати об'єкт розробки)

4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)
**ТИПИ СУЧАСНИХ БАЗ ДАНИХ, РОЗГОРТАННЯ ТА НАЛАШТУВАННЯ
БАЗИ ДАНИХ, ОПИС ОСНОВНИХ РІШЕНЬ ПО СТРУКТУРІ БАЗИ**

(визначаються назви розділів або (та) перелік питань, які повинні увійти до тексту ПЗ)

5. Індивідуальний план виконання кваліфікаційної роботи

№	Заходи	Термін виконання
1.	Вибір теми роботи, вивчення наукової літератури, затвердження теми та керівника.	До 30 жовтня 2023
2.	Аналіз літературних джерел за темою роботи. Розробка ТЗ. Розробка та апробація методики дослідно-експериментальної роботи. Подання структури теоретичної частини роботи (пояснювальної записки) та плану експериментальних досліджень.	Другий тиждень жовтня 2024
3.	Робота над теоретичною частиною. Подання теоретичної частини роботи для першого читання керівником. Розробка методики тестування	До 1 грудня 2024
4.	Усунення зауважень, урахування рекомендацій керівника. Аналіз структури програмного забезпечення.	Перший тиждень грудня 2024
5.	Поетапний аналіз та обговорення результатів. Перевірка стану виконання роботи.	Перший тиждень грудня 2024
6.	Урахування рекомендацій керівника, усунення недоліків, підготовка варіанта роботи до передзахисту. Оформлення документації до проекту.	До 15 грудня 2024
7.	Попередній захист роботи на кафедрі.	За місяць до державної атестації
8.	Доопрацювання роботи з урахуванням рекомендацій після передзахисту. Розробка презентації. Підготовка графічних матеріалів. Перевірка на плагіат. Подання роботи науковому керівникові та рецензентові на підготовку відгуку та рецензії	За 10 днів до державної атестації
9.	Подання на кафедру остаточного варіанта роботи, з відгуком керівника і рецензена.	За 3 дні до державної атестації

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП.....	8
РОЗДІЛ І. ЗАГАЛЬНИЙ ОГЛЯД СУЧАСНИХ БАЗ ДАНИХ	10
1.1. Історичний огляд розвитку БД	10
1.2 Реляційні БД	12
1.3 Об'єктні БД.....	15
1.4 NoSQL БД та bigdata.	17
1.5 Висновки до розділу 1	21
РОЗДІЛ ІІ. РОЗРОБКА БД НА ЗАСАДАХ POSTGRESQL	23
2.1 Розгортання PostgreSQL	23
2.2 Налаштування PostgreSQL	25
2.3 Робота з таблицями	29
2.4 Використання pgAdmin	30
2.4. Висновки до розділу 2	35
РОЗДІЛ 3. РОЗРОБКА БД СИСТЕМИ ВИЗНАЧЕННЯ ПЕРЕЛІКУ ВИБІРКОВИХ ДИСЦИПЛІН.....	36
3.1. Обґрунтування вибору мови та середовища розробки	36
3.2. Аналіз вимог до БД	37
3.3. Загальна структура БД.....	38
3.4. Загальна структура пакетів проекту та класів.....	51
ВИСНОВКИ	56
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	58
ДОДАТОК.....	61

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПК	Программный комплекс
ОС	Операционная система
БД	База данных
ОП	Освітня програма
ВНЗ	Вищий навчальний заклад
ОК	Освітній компонент
IDE	Integrated Development Environment (Интегрированная среда разработки)
HTTP	HyperText Transfer Protocol (Протокол передачи гипертекста)
JDBC	Java DataBase Connectivity (Соединение с базами данных)
RIA	Rich Internet Application (Насыщенное интернет приложение)
CSS	Cascading Style Sheets (Каскадные таблицы стилей)
MSN	Microsoft Network
IGMP	Internet Group Management Protocol (Протокол управления группами интернета)
IP	Internet Protocol

ВСТУП

Згідно законодавства Україна та розпоряджень МОН України сучасна система освіти у вищій школі складається та базується на засадах використання освітніх, освітньо-професійних та освітньо-наукових програм (ОП).

Не зважаючи на велику кількість різноманітних шляхів запровадження цих ОП, можливо виділити декілька спільних рис:

- Всі ОП складаються з окремих освітніх компонентів.
- Всі освітні компоненти розподіляються на нормативні та вибіркові.
- Не менш двадцять п'ять відсотків освітніх компонентів від загального обсягу ОП повинні складати вибіркові освітні компоненти.
- ВНЗ повинні розробити організаційні заходи для здійснення вільного вибору студентами (з певного переліку) вибіркових ОК.

Попередній аналіз акредитаційних справ та висновків експертних груп показує, що в багатьох ВНЗ не виконані умови, які забезпечують вільний вибір цих освітніх компонентів. Тому задача створення системи, яка буде забезпечувати вибір ОК є своєчасною та актуальною.

Об'єкт дослідження – система управління переліком вибіркових дисциплін у ВНЗ

Предмет дослідження – автоматизація визначення переліку вибіркових дисциплін.

Мета роботи – розробка БД, яка забезпечить створення системи визначення переліку вибіркових дисциплін.

Досягнення зазначеної мети передбачає вирішення таких основних завдань:

- Провести аналіз типів сучасних БД;
- Запропонувати методи розгортання та налаштування БД.
- Розробити БД та реалізувати програмний додаток.

У першому розділі надано аналіз та наведено загальних опис сучасних типів БД. Значна увага приділяється реляційним БД.

Другий розділ призначено особливостям розгортання та налаштування БД PostgreSQL

Третій розділ присвячений розробці БД, яка забезпечить створення системи визначення переліку вибіркового дисциплін.

РОЗДІЛ І. ЗАГАЛЬНИЙ ОГЛЯД СУЧАСНИХ БАЗ ДАНИХ

1.1. Історичний огляд розвитку БД

«Основний розвиток СУБД почався в 60-роки XX сторіччя, коли корпорація IBM разом з фірмою NAA (North American Aviation) розробили першу СУБД – ієрархічну систему IMS (Information Management System)» (див., наприклад, [2]). В середині 60-років поява системи IDS (Integrated Data Store) надала розвитку СУБД новий імпульс. Розвиток цієї СУБД призвів до появи нового типу СУБД – мережевого. В 1970 р. Е.Ф. Кодд в статті [3], визначив реляційну модель даних, яка усувала недоліки попередніх моделей. «В запропонованій моделі Кодда можна було виділити три аспекти: цілісний, структурний і маніпуляційний. В основі реляційної моделі лежать структури даних на “плоских” нормалізованих відношеннях, обмеження цілісності визначаються засобами логіки першого порядку, а маніпуляції над даними відбуваються засобами реляційної алгебри (реляційного числення на доменах чи кортежах). Саме завдяки розвиненому в математиці апарату теорії множин, логіки першого порядку та теорії відношень реляційна модель набула широкого розповсюдження. В цій моделі Кодд запропонував 8 операцій реляційної алгебри: традиційні операції над таблицями, які розуміються як множини рядків (об’єднання, перетин, різниця) та спеціальні операції над таблицями (проекція, декартове з’єднання, з’єднання (theta-, equi-), ділення, селекція), що використовують специфіку таблиць і рядків» [3]. Набір операцій реляційної алгебри, запропонований Коддом, з часом був розширений відповідно до потреб мов запитів. Крім вказаних вище операцій до сигнатури реляційної алгебри зараз також відносять операції перейменування та активного доповнення [4, 5].

Наступні типи СУБД, які з’явилися після реляційних, можна віднести до так званих “постреляційних”. Так, в [6] застосовується поняття

“постреляційні” БД для розширених реляційних, багатомірних та об’єктно-орієнтованих СУБД.

Основи об’єктно-орієнтованого програмування (ООП), зокрема, об’єктна модель даних (ОМД), були закладені наприкінці 60-х років минулого сторіччя [7] і з цього моменту набувала вдосконалення. Поява ОМД пов’язана з поняттям абстрактних типів даних (АТД) [8, 9]. Саме ООП дало поштовх до створення в середині 80-років минулого сторіччя нового типу СУБД – об’єктних. У 1996 р. з виходом СУБД Informix Universal Server корпорації Informix почалася епоха об’єктно-реляційних БД. Наприкінці 90-х років з’явився новий напрямок в розвитку СУБД – так звані NoSQL (Not only SQL) СУБД, націлені на обробку великих об’ємів слабоструктурованої інформації.

Зауважимо, що існують багато визначень терміну БД, які взаємно доповнюють одне одного. Так в [10] БД – це по суті, не що інше, як комп’ютеризована система збереження записів. В [11] БД визначається, в широкому розумінні, як набір логічно зв’язаних даних (або опис цих даних), призначений для задоволення інформаційних потреб організації. На більш конкретному рівні під БД розуміється єдине велике сховище даних, яке однократно визначається, а потім використовується одночасно багатьма користувачами. В [12] БД – це самодокументоване зібрання інтегрованих записів. БД є самодокументованою (selfdescribing), якщо вона містить з даними користувача опис власної структури. Цей опис називається словником даних (data dictionary), каталогом даних (data directory) або метаданими (metadata). Слід зауважити, що згідно [12] БД є моделлю моделі. БД не моделює реальність або якусь її частину, а є моделлю моделі користувача (user model). В [13] під БД розуміється сукупність призначених для машинної обробки даних, яка слугує для задоволення потреб багатьох користувачів в рамках однієї або декількох організацій. В [14] під БД розуміється набір даних, що знаходиться під контролем СУБД.

Історично першим типом БД по суті була файлова система, яка, як було сказано вище, мала ряд принципових недоліків, усунення яких спричинило подальший бурхливий розвиток БД.

1.2 Реляційні БД

В основі функціонування реляційних БД лежить реляційна модель даних. Наведемо одне із багатьох означень реляційної БД.

Реляційна БД – це скінченний набір відношень. Відношення використовуються для представлення сутностей та зв'язків між сутностями [15]. В основі реляційної моделі лежать такі складові [11]:

- «відношення, фізичним представленням якого є “плоска” таблиця, що складається із стовпців та рядків; рядки відповідають окремим записам, а стовпці – атрибутам;
- атрибут – іменований стовпець відношення; атрибути можуть розташовуватися в таблиці в довільному порядку;
- домен – множина допустимих значень одного або декількох атрибутів; кожен атрибут визначається на деякому домені; домен може бути системним або створеним користувачем на основі системних доменів;
- кортеж (синоніми – рядок, запис) – рядок відношення; кортежі в таблиці можуть бути розташовані в довільному порядку, при цьому відношення буде залишатися таким же самим, а отже мати той же зміст».

Реляційна модель тісно пов'язана з мовою SQL (Structured Query Language – мова структурованих запитів). Фактично коли говориться про мову програмування для реляційних БД, то мається на увазі саме SQL [18]. Дамо одне із означень мови SQL. SQL – формальна, непроцедурна мова

програмування, що застосовується для створення, модифікації та керування даними в реляційній БД, яка керується відповідною СУБД.

Мову SQL можна поділити на дві частини: мову DDL (Data Definition Language) та мову DML (Data Manipulation Language). Засобами мови DDL можна створювати (оператор Create), модифікувати (оператор Alter або Modify), знищувати (оператор Delete) елементи БД, а також можна керувати повноваженнями користувачів БД (оператори Grant, Revoke). Засобами мови DML можна здійснювати додавання (оператор Insert), знищення (оператор Delete), редагування (оператор Update) даних БД, а також їх вибірку (оператор запиту Select).

Для здійснення фільтрування рядків таблиці в мові SQL після службового слова Where використовуються предикати на рядках. Предикати можуть повертати третє логічне значення unknown разом з стандартними значеннями істини та хибності (true та false, відповідно). Це означає, що в SQL використовується трьохзначна логіка замість класичної двозначної (булевої), а саме це так звана сильна тризначна логіка Кліні

БД може зберігати досить великий об'єм інформації, яка розподілена між таблицями цієї БД. Через це виникає ситуація, коли одні і ті ж дані можуть знаходитися (дублюватися) в декількох таблицях. Тоді можна говорити про надлишковість такої інформації в БД, наявність якої в БД вказує на можливі проблеми підтримки цілісності. При роботі з таблицями БД, які мають надлишкові дані, виникають проблеми, що називаються “аномаліями”. Можна умовно розділити аномалії на три види:

- аномалія включення – проблема, пов'язана з додаванням інформації в БД;
- аномалія модифікування – проблема, пов'язана зі змінами інформації в БД;
- аномалії знищення – проблема, пов'язана з видаленням інформації з БД.

Питання усунення аномалій БД постає на етапі проектування БД. Саме під час проектування БД за рахунок нормалізації повинні вирішуватися задачі мінімізації дублювання даних і спрощення методів їх оновлення та обробки. Нормалізацією називається формальна процедура, в ході якої атрибути даних групуються в таблиці, а таблиці групуються в БД. Задачами нормалізації є [21]:

- виключення в таблицях інформації, що повторюється;
- створення структури, в якій передбачена можливість її майбутніх змін;
- створення структури, в якій вплив структурних змін на додатки, що використовують дані цієї БД, зведено до мінімуму.

До реляційних БД відносяться

- MySQL
- MariaDB
- PostgreSQL
- SQLite

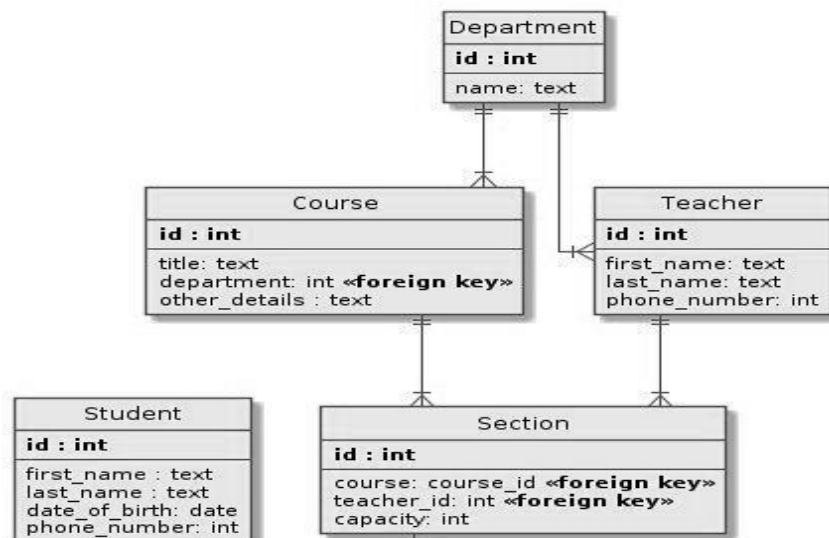


Рис.1.1 Приклад реляційної БД[<https://senior.ua/articles/11-tipv-suchasnih-baz-danih-korotkiy-opis-shemi--prikлади-bd>]

Вперше термін “нормалізація” був застосований у 1970 р. Е.Ф. Коддом для назви процедури усунення непротих доменів [3]. Історично поняття першої нормальної форми (1НФ) трансформувалося та уточнювалося. Так, спочатку це поняття уточнювалося наступним чином: “Відношення знаходиться в 1НФ, якщо кожне значення у відношенні є атомарним елементом даних” [10].

У 1971 р. Е.Ф. Кодд вказує на надлишковість даних та аномалії, які виникають при здійсненні операцій над відношеннями, вперше представляє концепцію функціональної залежності (ФЗ), демонструє можливість її використання для розв’язання проблем проектування БД, наводить означення другої нормальної форми (2НФ), транзитивної ФЗ та третьої нормальної форми (3НФ).

1.3 Об’єктні БД

Складна будова об’єктивної дійсності спонукає людину використовувати класифікаційні схеми, які дозволяють цілісно представляти цю реальність у вигляді об’єктів. Тому абсолютно природнім стало впровадження підтримки об’єктного підходу в мови програмування та в БД, зокрема. В основі об’єктного підходу лежить ОМД. В [11] ця модель визначається як модель, що враховує семантику об’єктів – головного поняття ООП. Основна ідея об’єктного підходу полягає в об’єднанні даних і операцій, які виконуються над цими даними, в одне концептуально замкнуте поняття – клас. Дані класу не повинні змінюватися ззовні, а доступ до даних слід здійснювати тільки через функції- члени (методи класу). Програма, яка написана на об’єктній мові, взаємодіє з сукупністю об’єктів, кожен з яких належить до певного АТД (класу) і має інтерфейс у вигляді сукупності методів для взаємодії з іншими об’єктами за допомогою повідомлень.

Розробка систем об’єктних БД (ОБД) розпочалася в 80-х роках минулого століття в зв’язку з впровадженням складних програмних додатків,

пов'язаних з автоматизованим проектуванням (Computer Aided Design – CAD), автоматизованим виробництвом (Computer Aided Manufacturing – CAM), системами, які основані на знаннях, тощо. Використання реляційного підходу в таких застосуваннях показало його неефективність. Іншими словами, поява ОБД була зумовлена більш адекватним представленням та моделюванням сутностей реального світу в ОБД в порівнянні з реляційними БД. Об'єктна теорія побудована з використанням базових понять об'єктного підходу Г. Буча та трикутника Г. Фреге, виходячи з наступних принципів:

- загальності об'єктного визначення: всі сутності – об'єкти;
- унікальності: кожен об'єкт – унікальний елемент;
- об'єктної впорядкованості: всі об'єкти впорядковані у відповідності з певними відношеннями;
- цілісності об'єктної моделі: об'єкти і відношення між ними однозначно визначаються в моделі на певному рівні абстракції;
- Інтер операбельності об'єкті: об'єкти пов'язуються операціями викликів на множинах вхідних і вихідних інтерфейсів.

Обов'язковою складовою ОБД є, по-перше, можливість маніпулювання об'єктами, що представляються у вигляді сутностей в адресному просторі обчислювальної системи, яка з'являється при створенні екземпляра класу, та, по-друге, використання концепцій ООП. Разом з цим, кожна одиниця (об'єкт) інформації в таких БД володіє двома характеристиками: станом та поведінкою. Стан визначається сукупністю поточних значень атрибутів та зав'язків, які має об'єкт. Значеннями атрибутів можуть виступати як елементи примітивних (системних) типів даних, так і інші об'єкти. Останній факт дає можливість визначати об'єкт рекурсивно через інші об'єкти. Стан визначає реакцію об'єкта на подію, яка в нього поступає. Об'єкт зберігає свій стан на протязі часу між двома послідовними подіями, які він приймає.

Поведінка визначається можливістю об'єкта реагувати на дії ззовні. Реагування на дії ззовні відбувається через методи об'єкта, які в нього

інтегровані. Кожен об'єкт в БД володіє унікальним ідентифікатором OID (Object Identifier). Об'єкти, які мають однакові схеми та поведінку, групуються в класи, створюючи при цьому інстанс (instance) класу. Об'єкт може належати як інстансу одного класу, так і інстансу декількох класів у випадку звичайного та множинного успадкування. Існує наступна відмінність у використанні типів даних атрибутів. Так, в реляційній БД, як правило, атрибути мають прості (примітивні) типи даних, а в ООБД значеннями атрибутів можуть бути складні типи даних. Причому складні типи даних можуть бути вбудовані в СУБД, а можуть бути створені користувачем (user-defined types).

1.4 NoSQL БД та bigdata.

З кожним роком кількість інформації, яка повинна зберігатися та оброблюватися в БД, стрімко зростає. Також сучасні СУБД стикаються з інформацією, яка не є структурованою, зростають вимоги до надзвичайно швидкого виконання запитів та оновлення фізично розподілених даних. Ця тенденція вимагає постійного підвищення функціональних можливостей як серверів (hardware: підвищення тактової частоти процесорів, пам'яті, зменшення часу запису-зчитування інформації з накопичувачів), так і реалізації механізмів підвищення швидкодії роботи з інформацією в самій БД на логічному рівні (software). Якщо розглянемо реляційні БД, то для підтримки цілісності та прискорення виконання запитів на великому об'ємі інформації були реалізовані механізми: нормалізація, денормалізація та шардінг (sharding, так зване вертикальне та горизонтальне масштабування, яке є архітектурним рішенням). Суть шардінгу полягає в розділенні БД на окремі частини з подальшим перенесенням цих частин на окремі сервери. Шардінг можна розділити на два типи: вертикальний та горизонтальний.

Це і дало поштовх для розвитку нового типу БД – NoSQL (Not Only SQL – не тільки SQL). Зауважимо, що термін NoSQL визначає скоріше напрямок розвитку БД, а ніж технологію. За моделлю даних БД типу NoSQL можна умовно поділити на 4 типи:

- документо-орієнтовані (OrientDB, MongoDB) (рис. 1.2);
- БД типу “ключ-значення” (Redis, Memcached)(рис. 1.3), для зберігання інформації ви надаєте ключ і об’єкт даних, який потрібно зберегти. Наприклад, JSON-об’єкт, зображення або текст. Щоб зробити запит, відправляєте ключ і отримуєте blob-об’єкт;
- стовпцеві БД (Cassandra, HBase) – рис. 1.4
- графові БД (OrientDB, HyperGraphDB, Infinite Graph) – рис. 1.5.

Документні бази даних (також документоорієнтовані БД або сховища документів), спільно використовують базову семантику доступу і пошуку сховищ ключів і значень. Такі БД також використовують ключ для унікальної ідентифікації даних. Різниця між сховищами «ключ-значення» і документними БД полягає в тому, що замість зберігання blob-об’єктів, документоорієнтовані бази зберігають дані в структурованих форматах - JSON, BSON або XML.

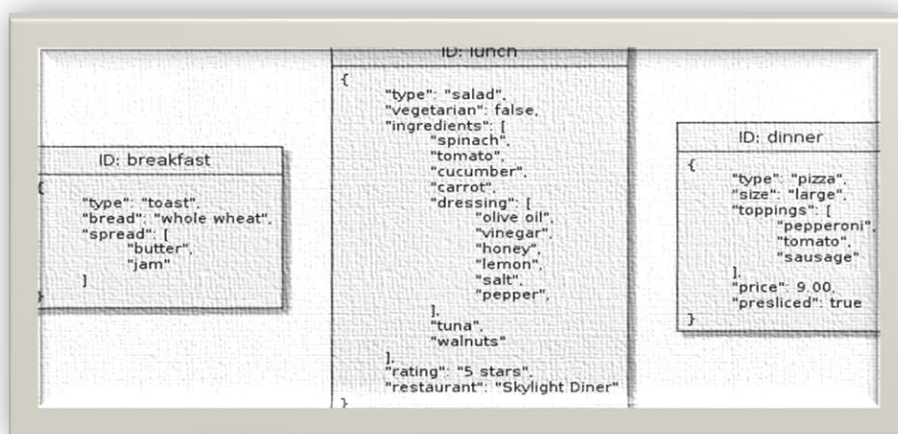


Рис. 1.2 Документо-орієнтовані БД <https://senior.ua/articles/11-tipv-suchasnih-baz-danih-korotkiy-opis-shemi--prikлади-bd>

Особливості:

- база даних не виділяє окремий формат або схему;
- кожен документ може мати свою внутрішню структуру;
- документні БД є хорошим вибором для швидкої розробки;
- в будь-який момент можна змінювати властивості даних, не змінюючи структуру або самі дані.

key:	value
user_id:	fsbadc33-5bd7-4b65-a737-b5304675f476
color:	blue
repetitions:	3
text:	hello world
data:	{ }

Рис. 1.3 БД - ключ-значення <https://senior.ua/articles/11-tipv-suchasnih-baz-danih-korotkiy-opis-shemi--prikлади-bd>

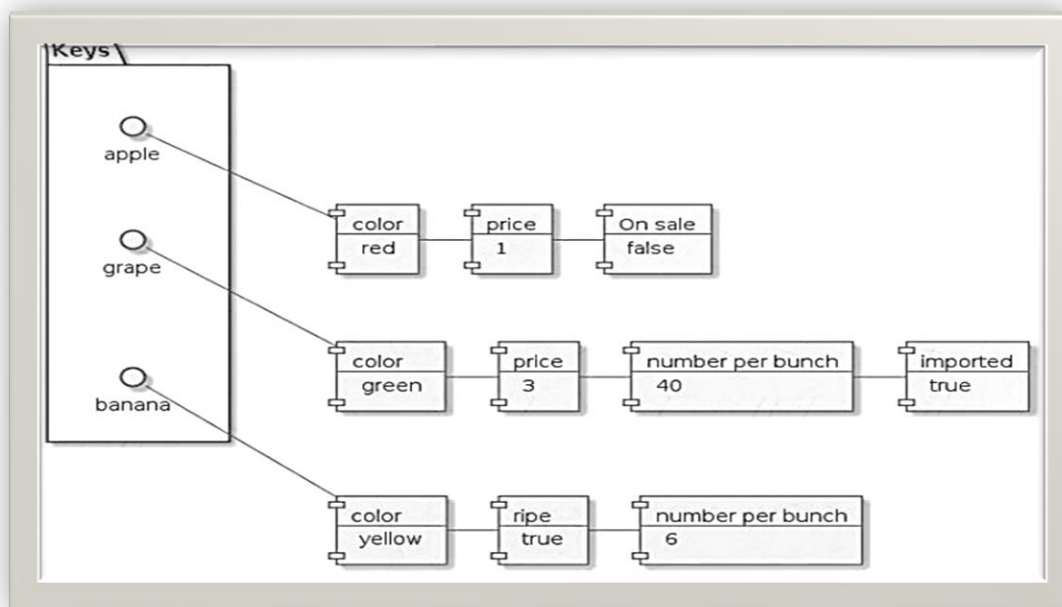


Рис. 1.4 Стовбцеві БД <https://senior.ua/articles/11-tipv-suchasnih-baz-danih-korotkiy-opis-shemi--prikлади-bd>

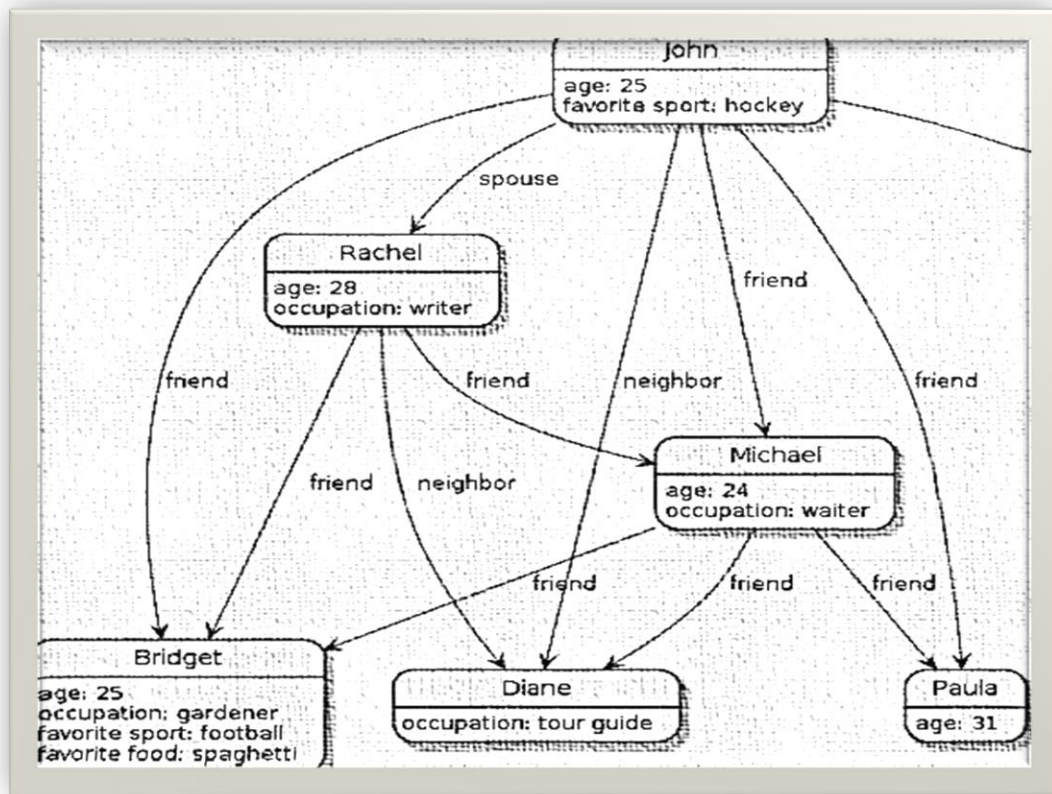


Рис. 1.5 Графові БД <https://senior.ua/articles/11-tipv-suchasnih-baz-danih-korotkiy-opis-shemi--prikлади-bd>

«Великі дані (англ. Big Data) в інформаційних технологіях — набори інформації (як структурованої, так і неструктурованої) настільки великих розмірів, що традиційні способи та підходи (здебільшого засновані на рішеннях класу бізнесової аналітики та системах управління базами даних) не можуть бути застосовані до них[1]. Альтернативне визначення називає великими даними феноменальне прискорення нагромадження даних та їх ускладнення. Важливо також відзначити те, що часто під цим поняттям у різних контекстах можуть мати на увазі як дані великого об'єму, так і набір інструментів та методів. В цілому, незважаючи на різноманітність типів БД, найбільш доцільним буде використання реляційну БД» [21].

1.5 Висновки до розділу 1

В цілому, незважаючи на різноманітність типів БД, найбільш доцільним буде використання реляційну БД.

У реляційній моделі даних об'єкти і взаємозв'язки між ними представляються за допомогою таблиць. Взаємозв'язки також подаються як об'єкти. Кожна таблиця представляє один об'єкт і складається з рядків і стовпців. Таблиця повинна мати первинний ключ (ключовий елемент) — поле чи комбінацію полів, що єдиним способом ідентифікують кожний рядок у таблиці.

Назва «реляційна» (relational) пов'язана з тим, що кожен запис у таблиці даних містить інформацію, яка стосується (related) якогось конкретного об'єкта. Крім того, зв'язані між собою (тобто такі, що знаходяться в певних відношеннях — relations) дані навіть різних типів в моделі можуть розглядатися як одне ціле.

- кожний елемент таблиці являє собою один елемент даних;
- повторювані групи відсутні;
- усі стовпці в таблиці однорідні; це означає, що елементи стовпця мають однакову природу;
- стовпцям присвоєні унікальні імена;
- у таблиці немає двох однакових рядків.

Порядок розміщення рядків і стовпців у таблиці довільний; таблиця такого типу називається відношенням. У сучасній практиці для рядка використовується термін «запис», а для стовпця термін «поле».

Основною відмінністю пошуку даних в ієрархічних, мережних і реляційних базах даних є те, що ієрархічні і мережні моделі даних здійснюють зв'язок і пошук між різними об'єктами за структурою, а реляційні — за значенням ключових атрибутів (наприклад, можна знайти всі записи,

значення яких у полі «номер будинку» дорівнює 3, але не можна знайти 3-й рядок).

Оскільки реляційна структура концептуально проста, вона дозволяє реалізовувати невеликі і прості (і тому легкі для створення) бази даних, навіть персональні, сама можливість реалізації яких ніколи навіть і не розглядалася в системах з ієрархічною чи мережною моделлю.

Недоліком реляційної моделі даних є надмірність по полях (для створення зв'язків між різними об'єктами бази даних).

Практично всі існуючі на сьогоднішній день комерційні бази даних і програмні продукти для їх створення використовують реляційну модель даних.

РОЗДІЛ II. РОЗРОБКА БД НА ЗАСАДАХ POSTGRESQL

PostgreSQL – це провідна СКБД з відкритим текстом програми, з глобальною спільнотою, що складається з тисяч користувачів і розробників, і яка об'єднує багато компаній і організацій. Проект PostgreSQL створюється на основі більш ніж 25-річного досвіду проектування і розробки, що почалась в Каліфорнійському університеті Берклі, і на даний час продовжує розроблятися безпрецедентними темпами. Продуманий набір можливостей PostgreSQL не тільки не поступається провідним комерційним СКБД, але й перевершує їх розвиненою функціональністю, розширюваністю, безпекою та стабільністю. Ви можете отримати додаткову інформацію про PostgreSQL і приєднатись до нашої спільноти за адресою [PostgreSQL.org](https://www.postgresql.org/about/press/presskit93/ua/). [https://www.postgresql.org/about/press/presskit93/ua/].

PostgreSQL використовує власну, BSD-подібну ліцензію, що потребує тільки зберігання інформації про авторські права і тексту самої ліцензії у ліцензованому тексті програми. Ця сертифікована організацією OSI ліцензія широко відома своєю гнучкістю і зручністю для бізнесу, оскільки вона не забороняє використовувати PostgreSQL у складі комерційних і закритих застосунків. Одночасно з підтримкою безліччю компаній і спільним володінням текстом програми, наша ліцензія робить PostgreSQL дуже популярною серед виробників, які хочуть впровадити СКБД у свій продукт без будь-яких відрахувань, без прив'язки до виробника чи ризику змін у ліцензуванні. [https://www.postgresql.org/about/press/presskit93/ua/]

2.1 Розгортання PostgreSQL

PostgreSQL використовує за замовчуванням кодування UTF-8. Оскільки в якості операційної системи обрано FreeBSD 8.1, яка не

повноцінно підтримує це кодування, є особливості локалізації. З UTF-8 не вміє працювати консольний драйвер syscons, тобто ви не зможете використовувати UTF-8 в текстовій консолі.

Найбільш зручним способом локалізації системи є метод Class Login, описаний в handbook – http://www.freebsd.org/doc/ru_RU.KOI8-R/books/handbook/using-localization.html.

Можливо використання іншого способу локалізації і тим більше, використовуєте KOI8-R, то вам доведеться подбати про те, щоб користувач pgsql (з'явиться після установки postgresql з порту) мав локаль UTF-8. Так само є можливість змусити працювати postgresql з кодуванням KOI8-R, але це вважається недоцільним. Використання KOI8-R далі в тексті не розглядається.

Установку робимо з портів, але не безпосередньо, а за допомогою portmaster, який можна знайти в

```
/usr/ports/ports-mgmt/portmaster
```

Якщо раніше була встановлена стара версія клієнта, замінюємо на нову:

```
portmaster -o databases / postgresql90-client postgresql-client - \ *
```

Якщо клієнт раніше не був встановлений, то встановимо його в такий спосіб:

```
portmaster databases / postgresql90-client
```

Встановлюємо сервер PostgreSQL :

```
portmaster databases / postgresql90-server
```

Після завантаження та збірки з вихідного коду ми отримуємо готовий сервер PostgreSQL. Тепер потрібно подбати про створення кластера бази

даних. Оскільки всю чорнову роботу зробив порт, ми можемо не дбати про створення необхідного користувача, а відразу перейти до ініціалізації бази.

По-перше, дозволяємо автоматичний запуск сервера в `/etc/rc.conf`:

```
postgresql_enable = "YES"
```

Всі необхідні команди і параметри ініціалізації, які описані в документації, будуть виконані при запуску:

```
/usr/local/etc/rc.d/postgresql initdb
```

На цьому кроці можуть чекати проблема з кодуваннями. Якщо це все-таки сталося, необхідно перечитати передмови і перейти до використання UTF-8.

По завершенні процедури ініціалізації база даних буде доступною за адресою:

```
/usr/local/pgsql/data/.
```

Запускаємо сервіс:

```
service postgresql start
```

Сервіс буде запускатися автоматично при кожному запуску сервера.

На даний час сервер налаштований так, що можливі будь-які підключення з локальної машини без пароля. Необхідно обмежити доступ до сервера без пароля.

2.2 Налаштування PostgreSQL

Існує два способи керувати базами даних і користувачами PostgreSQL:

- утиліти командного рядка (`createuser`, `createdb`, `dropuser`, `dropdb` і ін.)
- інтерактивний термінал.

Необхідно скористатись інтерактивним терміналом `psql` для налаштування прав доступу. Першу команду слід виконувати від `root`, тому що в цьому випадку не буде потрібно пароль для користувача `pgsql`:

```
# su pgsq1
$psql -U pgsq1 template1
```

Буде отримано відповідь:

```
psql (_____) БЕРСІЯ)
Type "help" for help.
template1=#
```

Тут можна ознайомитися з наданими можливостями за допомогою команд `\h` і `\?`. Для прикладу, створіть користувача `test` і базу даних `test` до якої він матиме доступ.

```
template1=# create user test;
CREATE ROLE
template1=# \du
```

List of roles		
Role name	Attributes	Member of
pgsql	Superuser, Create role, Create DB	{}
test		{}

Слід зауважити, що команда повинна завершуватися символом `;`. Якщо символ відсутній, то оболонка очікує продовження команди. Якщо ви раптом забули встановити `;` після команди, можливо зробити це в наступному рядку. Після виконання команди завжди видається відповідь, як в прикладі вище `"CREATE ROLE"`.

В результаті виведення команди `\du` у нас є 2 користувача PostgreSQL: `pgsql` - володіє правами на налаштування сервера, створення користувачів і баз даних, і наш новий користувач `test` з обмеженими правами. Ми можемо змінити права доступу користувача в такий спосіб:

```
template1=# alter user test superuser createrole createdb;
ALTER ROLE
```

```
template1=# \du
```

List of roles		
Role name	Attributes	Member of
-----+-----+-----		
pgsql	Superuser, Create role, Create DB	{}
test	Superuser, Create role, Create DB	{}

По можливості необхідно уникати роздачі таких високих прав доступу.

Як правило, немає сенсу давати такі права для звичайного користувача.

Повернемо нашому користувачеві test початкові права:

```
template1=# alter user test nosuperuser nocreaterole nocreatedb;
```

```
ALTER ROLE
```

```
template1=# \du
```

List of roles		
Role name	Attributes	Member of
-----+-----+-----		
pgsql	Superuser, Create role, Create DB	{}
test		{}

Обмежимо доступ паролями:

```
template1=# \password test
```

```
Enter new password:
```

```
Enter it again:
```

Процедура створення пароля стандартна – необхідно двічі ввести пароль для користувача. Аналогічним чином необхідно захистите суперкористувача (щоб мати можливість керувати сервером).

Необхідно включити перевірку пароля при підключенні до сервера. За це відповідає файл `/usr/local/pgsql/data/pg_hba.conf`:

```
# TYPE DATABASE USER CIDR-ADDRESS METHOD
# "local" is for Unix domain socket connections only
local all all trust
# IPv4 local connections:
host all all 127.0.0.1/32 trust
# IPv6 local connections:
host all all ::1/128 trust
```

Існує 3 записи для локальних підключень. Всі записи вказують на відсутність перевірки пароля - "trust". Необхідно змінити це значення на "md5":

#	TYPE	DATABASE	USER	CIDR-ADDRESS	METHOD
# "local" is for Unix domain socket connections only					
local	all		all		md5
# IPv4 local connections:					
host	all		all	127.0.0.1/32	md5
# IPv6 local connections:					
host	all		all	:::1/128	md5

Перезапуск сервер, для застосування налаштувань:

```
service postgresql restart
```

Спробуйте підключитися до інтерактивної оболонки. Як бачите, тепер нам пропонують ввести пароль для нашого користувача:

```
$ psql -U postgres template1
Password:
psql (9.0.0)
Type "help" for help.
template1=#
```

Створимо базу для нашого користувача і зробимо його власником бази:

```
template1=# create database test;
CREATE DATABASE
template1=# alter database test owner to test;
ALTER DATABASE
```

Для роботи з базою даних існують наступні графічні програми:

- **phpPgAdmin** – створена на PHP, тобто зручна в тих випадках, коли з базою працює веб-додаток. Для роботи потрібно будь-який веб-сервер з підтримкою PHP.
- **pgAdmin** – кроссплатформне додаток, написане на C ++. Підтримує безліч платформ: FreeBSD, Linux, Mac OS, Windows.

Користувачі FreeBSD можуть встановити додаток з портів: / usr / ports / databases / pgadmin3 /.

2.3 Робота з таблицями

Додавання таблиці

«CREATE TABLE [Назва таблиці]».

Після цієї команди в дужках напишіть параметри сітки «[Назва рядка] [Тип] ([Кількість позицій]) [Можливі обмеження]». Таким чином введіть кілька рядків, розділяючи їх комами. Після кожної натискайте Enter. У колонок можуть бути різні назви і типи. Коли закінчите записувати дані, закрийте дужку і поставте крапку з комою. Кількість позицій теж вказуйте в дужках.

Подивитися вміст таблиці —

«SELECT * FROM [Ім'я]»

Видалити таблиці

«DROP TABLE [Назва]».

Після цього натисніть Enter і напишіть «DROP TABLE» ще раз.

Ввести дані —

«INSERT INTO [Ім'я сітки] (Столбец1, Столбец2, Столбец3) VALUES ('Запись1', 'Запись2', 'Запись3');».

У стовпці будуть додані відповідні записи. Можете повторювати команду, щоб вписувати нову інформацію

Видалити значення —

«DELETE FROM [Назва таблиці] WHERE [Назва стовпця] \u003d '[Значення]';».

Новий стовпець —

«ALTER TABLE [Ім'я сітки] ADD [Ім'я колонки]».

Видалити стовпець —

«ALTER TABLE [Таблиця] DROP [Назва стовпця]».

2.4 Використання pgAdmin

Для спрощення адміністрування на сервері postgresql в базовий комплект установки входить такий інструмент як pgAdmin. Він являє графічний клієнт для роботи з сервером, через який ми в зручному вигляді можемо створювати, видаляти, змінювати бази даних і управляти ними. Так, на Windows після установки ми можемо знайти значок pgAdmin в меню Пуск і запустити його:



Рис. 2.1 Завантаження PgAdmin

Після цього нам відкриється наступна програма і тепер підключимося до сервера PostgreSQL. Для цього в лівій частині вікна програми розкриємо пункт Servers, який містить набір серверів PostgreSQL. При установці останньої версії встановлюється сервер, який за замовчуванням має назву PostgreSQL 10. Натиснемо на цей пункт, і нам відкриється вікно для введення пароля:

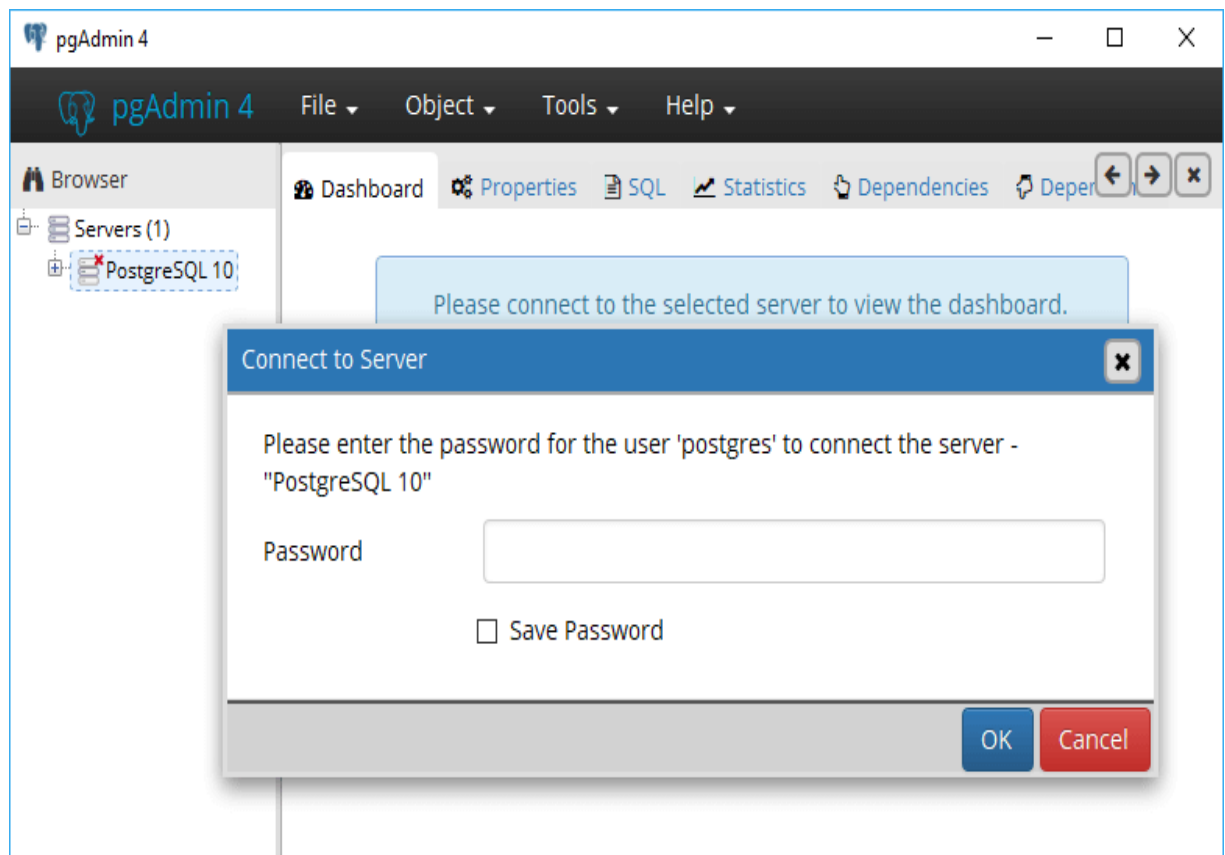


Рис. 2.2 Введення паролю для реєстрації

Тут необхідно ввести пароль для суперкористувача postgres, який був заданий при установці PostgreSQL. Після успішного логіна нам відкриється вміст сервера (рис. 2.4):

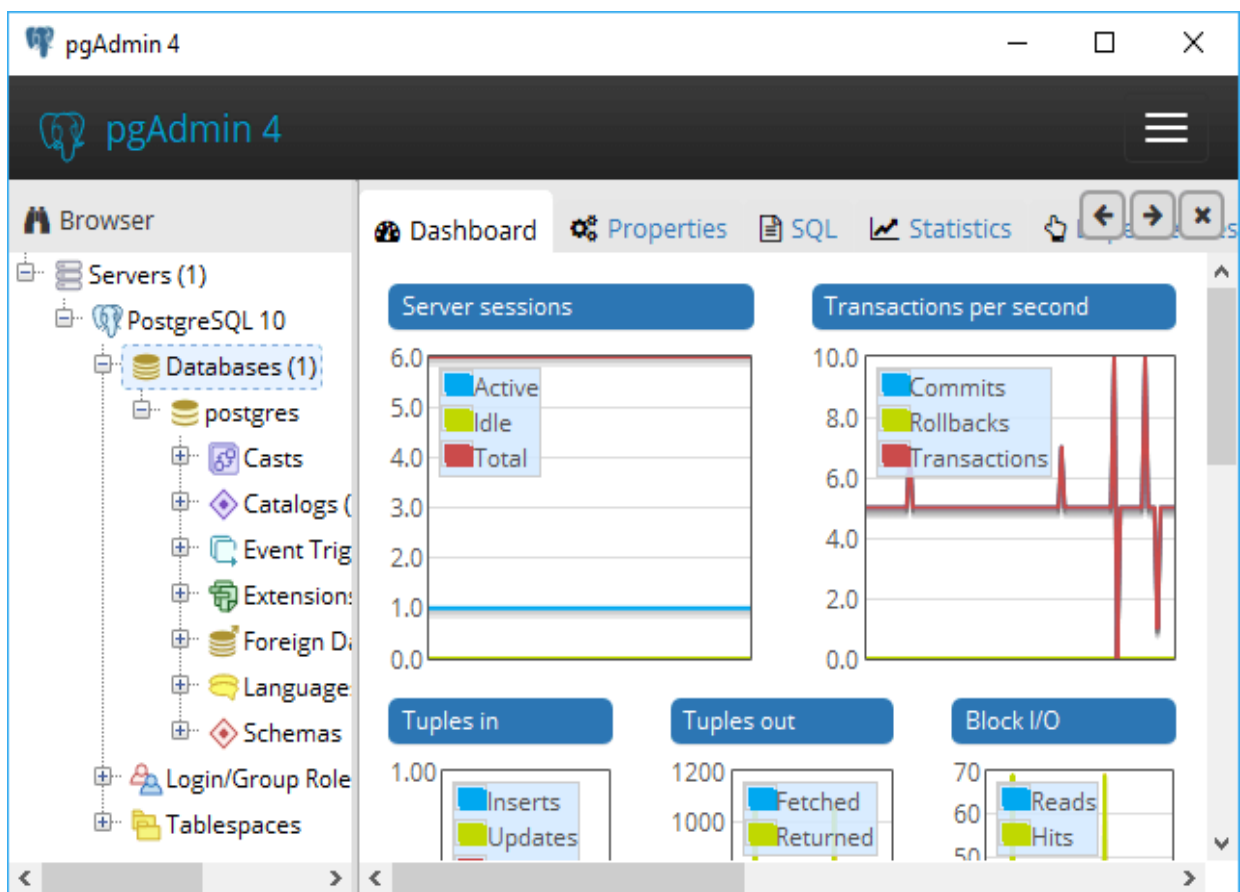


Рис. 2.4 Вміст сервера

Зокрема, в вузлі Databases ми можемо побачити всі наявні бази даних. За замовчуванням тут є тільки одна база даних - postgres.

Також в правій частині ми можемо побачити вузол Login / Group Roles, який призначений для управління користувачами і їх ролями.

Третій вузол - Tablespaces дозволяє управляти місцем зберігання файлів баз даних.

Тепер створимо свою базу даних.

Для цього натиснемо правою кнопкою миші на вузол Databases. І далі в контекстному меню виберемо Create-> Database ... (рис. 2.5)

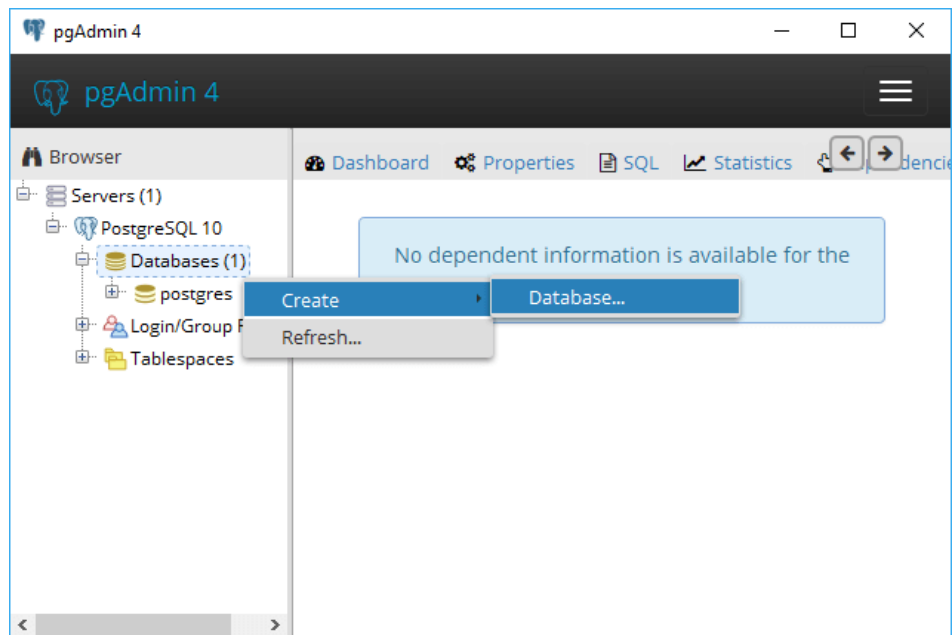


Рис. 2.5 Створення БД

Після цього нам відобразиться вікно для створення бази даних. Введемо назву для БД, наприклад, test1 і натиснемо на кнопку "Save" (рис. 2.6):

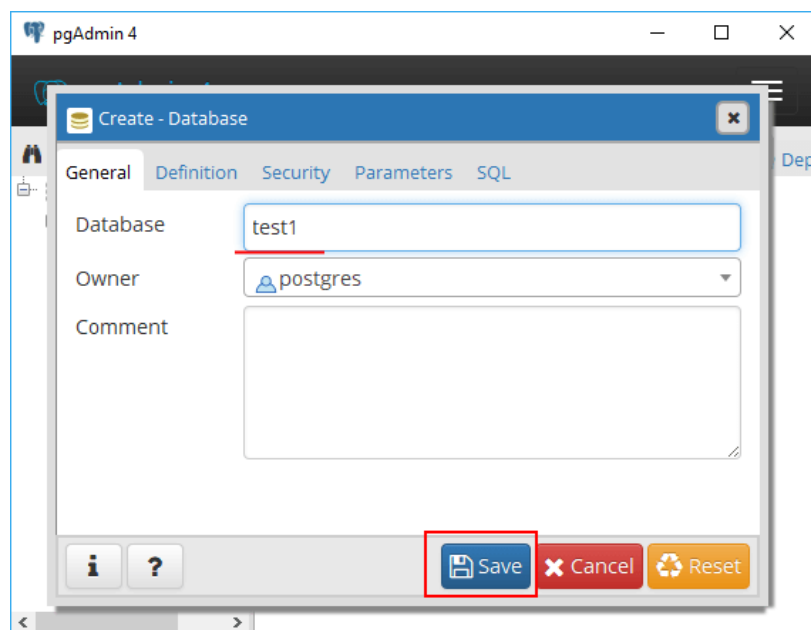


Рис. 2.6 Створення та збереження БД

Після цього в деревовидному меню зліва відобразиться вміст створеної бази даних test1:

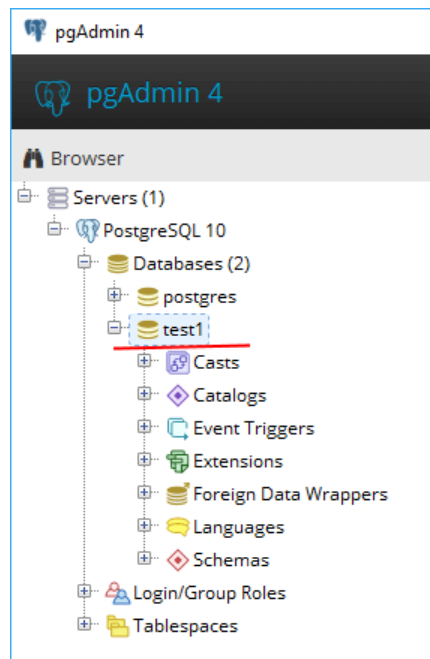


Рис. 2.7 Результат створення БД test

Аналогічно створюються інші БД. Загальний вигляд БД IFMIT показано на рис 2.8

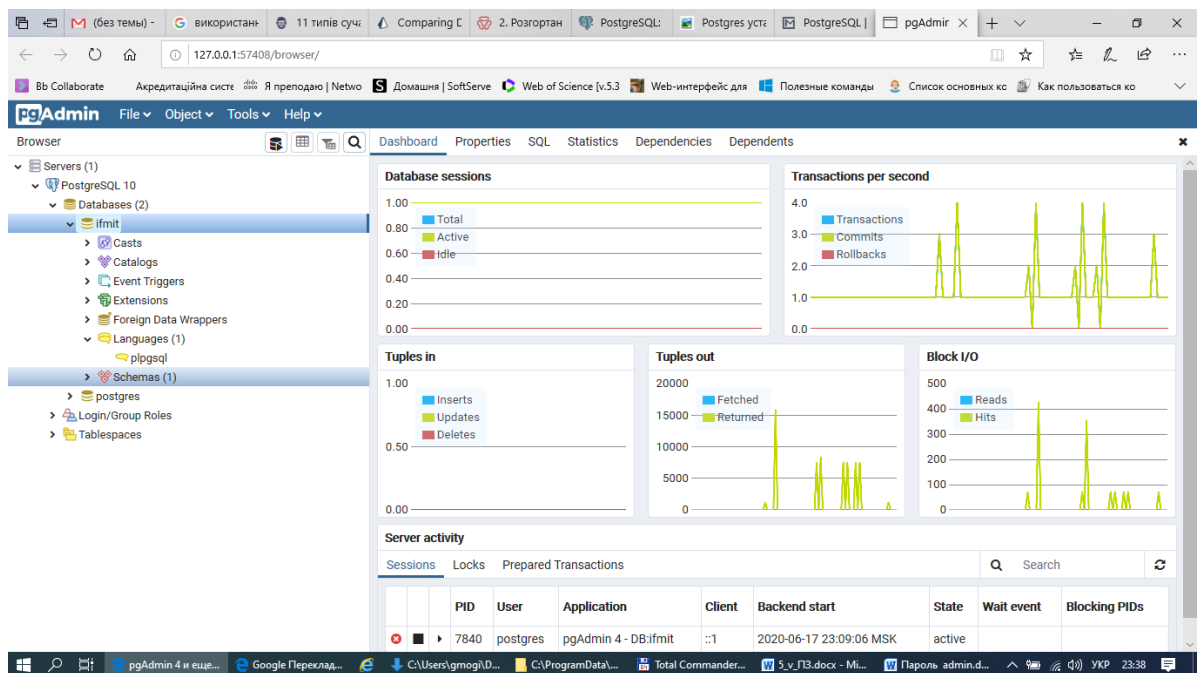


Рис 2.8 Створення БД IFMIT

2.4. Висновки до розділу 2

Таким чином, в результаті виконання даного розділу запропоновано наведено опис програмного забезпечення для створення БД ІМІТ в системі PostgreSQL та наведено опис процесу її налаштування.

РОЗДІЛ 3. РОЗРОБКА БД СИСТЕМИ ВИЗНАЧЕННЯ ПЕРЕЛІКУ ВИБІРКОВИХ ДИСЦИПЛІН

3.1. Обґрунтування вибору мови та середовища розробки

В цей час існує велика кількість засобів та мов програмування, які підтримують процесу створення та керування Веб орієнтованими системами.

Практично всі мови програмування підтримують можливість керування ДБ та мають у своєму арсеналі значну кількість методів та функцій, які можливо інтегрувати з певними ВЕБ серверами.

До таких мов відносяться:

- Python
- C#
- C++
- Java
- PHP
- Різноманітні бібліотеки на засадах javascript.

Однак значною мірою, при створенні корпоративних систем, виділяються різноманітні системи основані на java технологіях. Тому в якості мови програмування додатку обираємо JAVA.

Для java існує велика кількість різноманітних IDE

- Nebeans,
- Eclipse,
- Joracle IDE,
- IntelliJ IDEA.

Всі ці IDE мають певні переваги та недоліки та відповідають вимогам для моделювання та проектування отриманої задачі. Вибір IDE – це скоріш внутрішній стандарт компанії. Однак, останнім часом серед цих IDE

розпочав виділятися IntelliJ IDEA. Тому, в якості експерименти обираємо цю IDE (рис. 3.1)

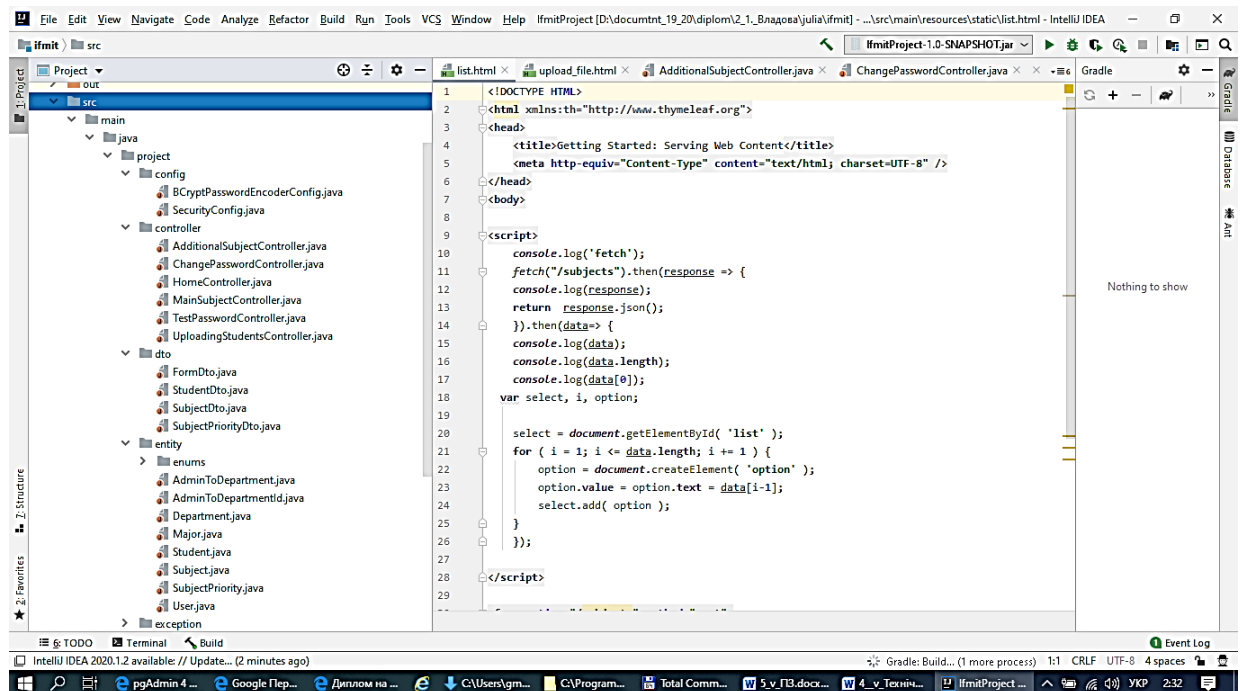


Рис. 3.1 IntelliJ IDEA

3.2. Аналіз вимог до БД

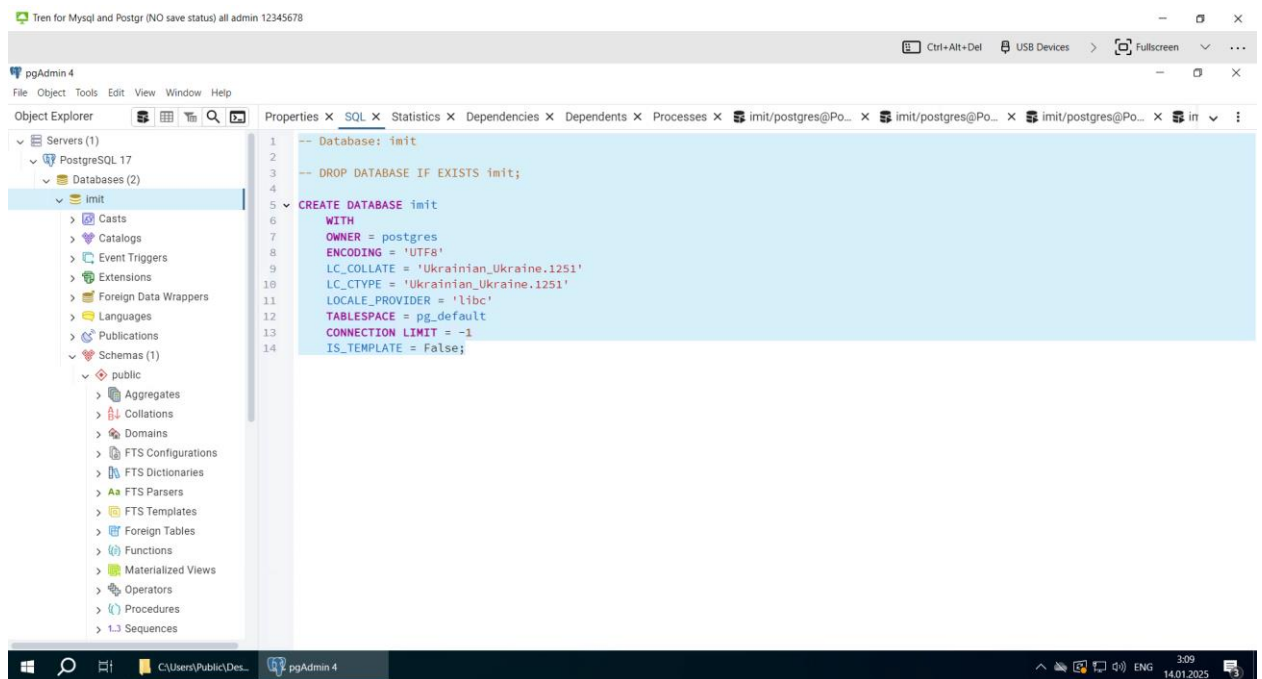
В результаті аналізу положень та процедур, які затверджені у ЛНУ імені Т. Шевченка та на засадах ґрунтовних спілкувань з представниками навчального відділу встановлена, що програмна розробка (на першому етапі впровадження) повинна відповідати наступним вимогам.

- Давати можливість переглянути, додати та перевірити контингент студентів;
- Давати можливість переглянути, додати перелік ОП та перевірити вибіркові дисципліни певної ОП;
- Враховувати наявність двох блоків вибірових дисциплін.
- Враховувати наявність різноманітних кредитів (обсягу) вибірових дисциплін.

- Враховувати пріоритети (бажаність) вибіркового дисциплін.
- Враховувати можливість різноманітних статусів процесу вибору вибіркового дисциплін.
- Враховувати можливість запровадити алгоритм формування груп студентів з урахування пріоритетів (бажань) студентів.
- Враховувати наявність рівнів освіти: бакалавр, магістр, аспірант.
- Надавати контрольований доступ до певних таблиць з урахуванням різноманітних ролей: студент - тільки використання, оператор - перегляд, менеджер-додавання та редагування.

3.3. Загальна структура БД

На засадах проведеного аналізу умов використання БД пропонується створити 10 таблиць (рис. 3.2)



a)

Type	Name	Restriction
Sequence	public.degree_id_seq	normal
Sequence	public.department_id_seq	normal
Sequence	public.major_id_seq	normal
Sequence	public.osvpr_id_seq	normal
Sequence	public.priority_id_seq	normal
Sequence	public.student_id_seq	normal
Sequence	public.subject_id_seq	normal
Sequence	public.subject_priority_id_seq	normal
Table	public.admin_to_department	normal
Table	public.degree	normal
Table	public.department	normal
Table	public.major	normal
Table	public.osvpr	normal
Table	public.priority	normal
Table	public.student	normal
Table	public.subject	normal
Table	public.subject_priority	normal
Table	public.user	normal

б)

Рис.3.2 Таблиці БД

Таблиця admin_to department(рис. 3.3)

Object Type	Name
Table	admin_to_department
Column	user_id
Column	department_id
Constraint	department_id_fk
Constraint	department_id_fk_atd
Constraint	user_id_fk

Рис. 3.3 Таблиця admin_to department

```
CREATE TABLE public.admin_to_department
(
    user_id bigint NOT NULL,
    department_id bigint NOT NULL,
    CONSTRAINT department_id_fk FOREIGN KEY (department_id)
        REFERENCES public.department (id) MATCH SIMPLE
        ON UPDATE NO ACTION
```

```

        ON DELETE NO ACTION,
CONSTRAINT department_id_fk_atd FOREIGN KEY (department_id)
    REFERENCES public.department (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION,
CONSTRAINT user_id_fk FOREIGN KEY (user_id)
    REFERENCES public."user" (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.admin_to_department
    OWNER to postgres;
-- Index: fki_department_id_fk_atd

-- DROP INDEX public.fki_department_id_fk_atd;

CREATE INDEX fki_department_id_fk_atd
    ON public.admin_to_department USING btree
    (department_id ASC NULLS LAST)
    TABLESPACE pg_default;
-- Index: fki_user_id_fk

-- DROP INDEX public.fki_user_id_fk;

CREATE INDEX fki_user_id_fk
    ON public.admin_to_department USING btree
    (user_id ASC NULLS LAST)
    TABLESPACE pg_default;

```

Таблиця department(рис. 3.4)

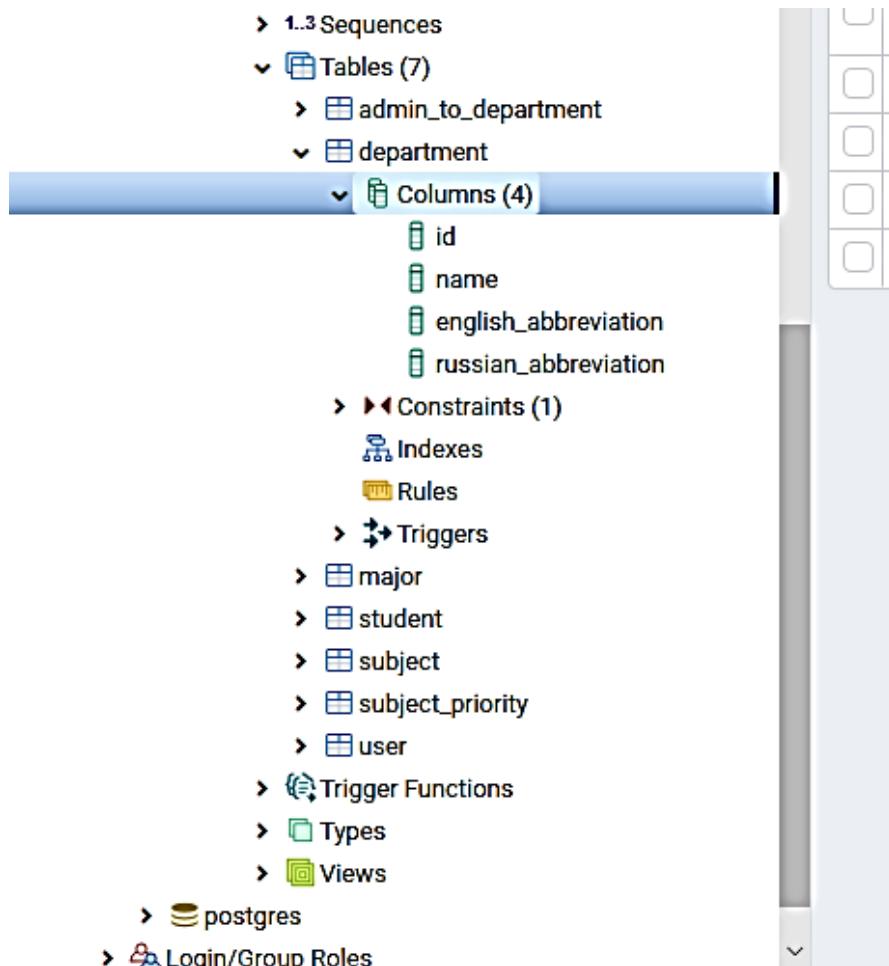


Рис. 3.4 Таблица department

```
CREATE TABLE public.department
(
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START
        1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),
    name text COLLATE pg_catalog."default" NOT NULL,
    english_abbreviation text COLLATE pg_catalog."default" NOT NULL,
    russian_abbreviation text COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT department_pkey PRIMARY KEY (id)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.department
    OWNER to postgres;
```

Таблица major(рис. 3.5)

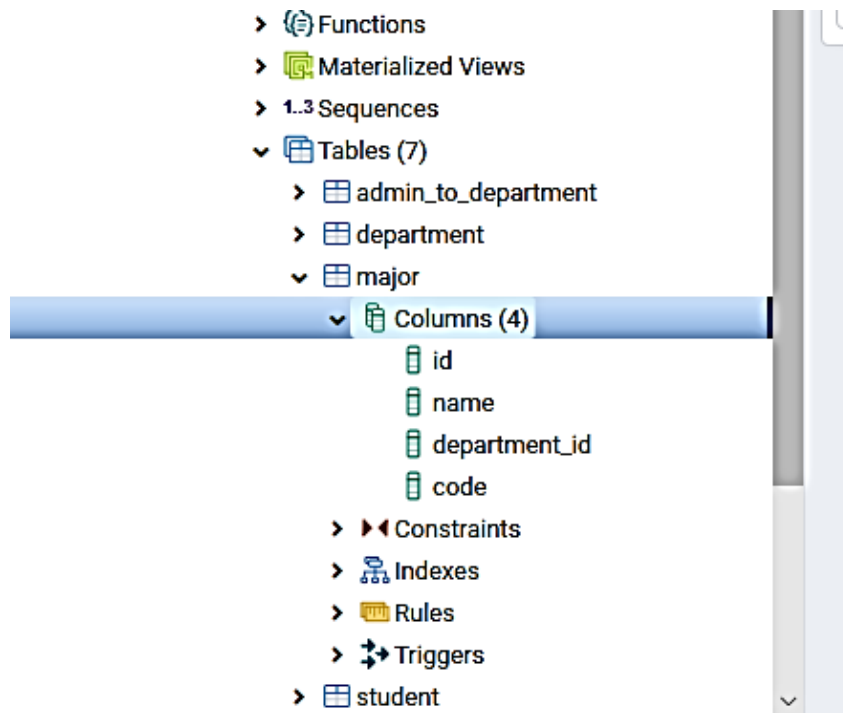


Рис. 3.5 Таблица major

```
CREATE TABLE public.major
(
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START
        1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),
    name text COLLATE pg_catalog."default" NOT NULL,
    department_id bigint NOT NULL,
    code text COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT major_pkey PRIMARY KEY (id),
    CONSTRAINT department_id_fk FOREIGN KEY (department_id)
        REFERENCES public.department (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.major
    OWNER to postgres;
-- Index: fki_department_id_fk

-- DROP INDEX public.fki_department_id_fk;

CREATE INDEX fki_department_id_fk
    ON public.major USING btree
    (department_id ASC NULLS LAST)
```

TABLESPACE pg_default; OWNER to postgres;

Таблица student(рис. 3.6)

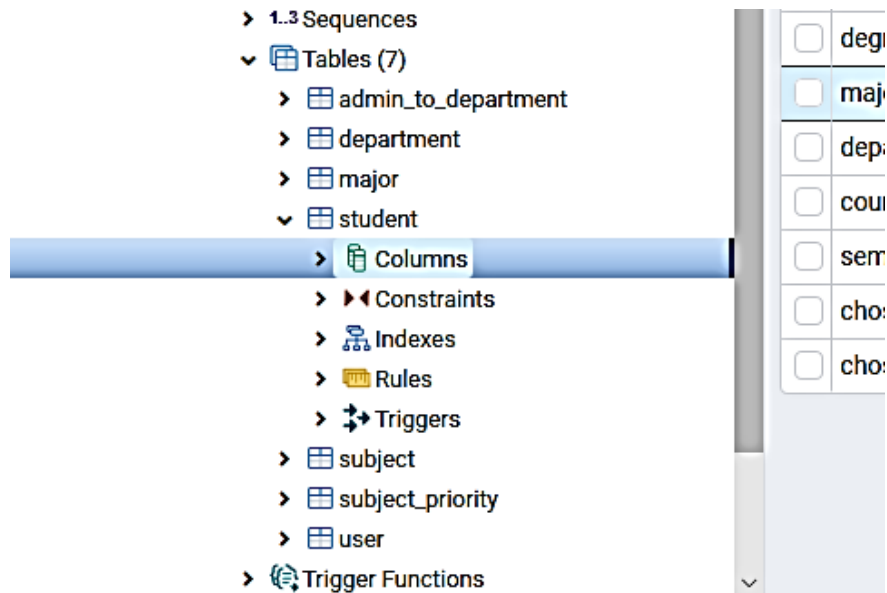


Рис. 3.6 Таблица student

```
CREATE TABLE public.student
(
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START
        1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),
    name text COLLATE pg_catalog."default" NOT NULL,
    last_name text COLLATE pg_catalog."default" NOT NULL,
    patronymic text COLLATE pg_catalog."default" NOT NULL,
    email text COLLATE pg_catalog."default" NOT NULL,
    degree text COLLATE pg_catalog."default" NOT NULL,
    major_id bigint NOT NULL,
    department_id bigint NOT NULL,
    course smallint NOT NULL,
    semester smallint NOT NULL,
    chose_additional_courses boolean,
    chose_main_courses boolean,
    CONSTRAINT student_pkey PRIMARY KEY (id),
    CONSTRAINT email UNIQUE (email),
    CONSTRAINT department_id_fk FOREIGN KEY (department_id)
        REFERENCES public.department (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT major_id_fk FOREIGN KEY (major_id)
        REFERENCES public.major (id) MATCH SIMPLE
```

```

        ON UPDATE NO ACTION
        ON DELETE NO ACTION
    )
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.student
    OWNER to postgres;
-- Index: fki_department_id

-- DROP INDEX public.fki_department_id;

CREATE INDEX fki_department_id
    ON public.student USING btree
    (department_id ASC NULLS LAST)
    TABLESPACE pg_default;
-- Index: fki_major_id_fk

-- DROP INDEX public.fki_major_id_fk;

CREATE INDEX fki_major_id_fk
    ON public.student USING btree
    (major_id ASC NULLS LAST)
    TABLESPACE pg_default;

```

Таблиця subject(рис. 3.7)

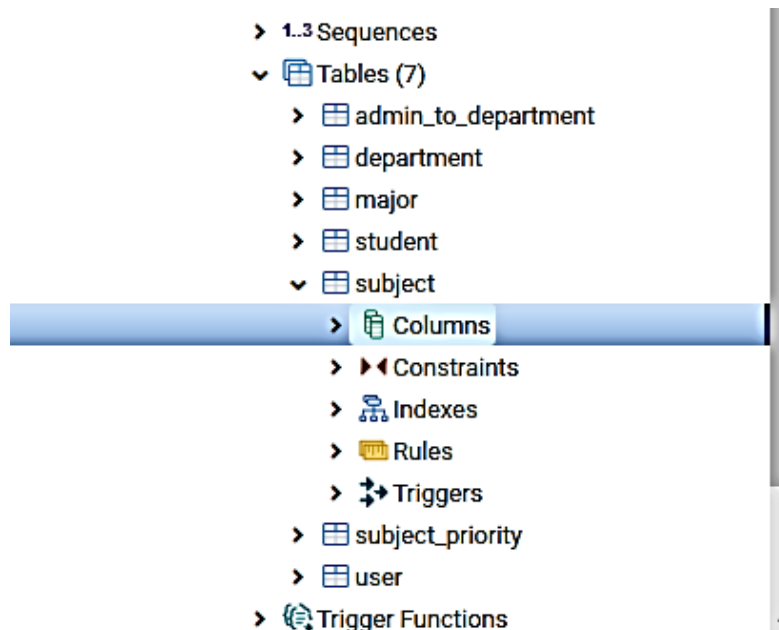


Рис. 3.7 Таблица subject

```
CREATE TABLE public.subject
(
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START
        1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),
    department_id bigint NOT NULL,
    major_id bigint NOT NULL,
    course smallint NOT NULL,
    semester smallint NOT NULL,
    name text COLLATE pg_catalog."default" NOT NULL,
    type text COLLATE pg_catalog."default" NOT NULL,
    degree text COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT subject_pkey PRIMARY KEY (id),
    CONSTRAINT subject_department_fk FOREIGN KEY (department_id)
        REFERENCES public.department (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT subject_major_fk FOREIGN KEY (major_id)
        REFERENCES public.major (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.subject
    OWNER to postgres;
-- Index: fki_subject_department_fk
```

```
-- DROP INDEX public.fki_subject_department_fk;

CREATE INDEX fki_subject_department_fk
    ON public.subject USING btree
    (department_id ASC NULLS LAST)
    TABLESPACE pg_default;
-- Index: fki_subject_major_fk

-- DROP INDEX public.fki_subject_major_fk;

CREATE INDEX fki_subject_major_fk
    ON public.subject USING btree
    (major_id ASC NULLS LAST)
    TABLESPACE pg_default;
```

Таблица subject_priority(рис. 3.8)

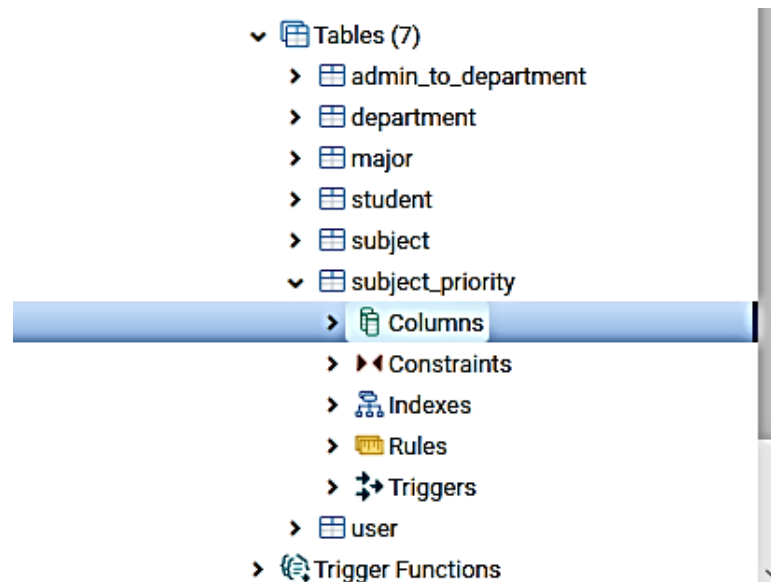


Рис. 3.8 Таблица subject_priority

```
CREATE TABLE public.subject_priority
(
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START
        1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),
    subject_id bigint NOT NULL,
    student_id bigint NOT NULL,
    subject_type text COLLATE pg_catalog."default" NOT NULL,
    priority smallint NOT NULL,
    CONSTRAINT subject_priority_pkey PRIMARY KEY (id)
)
WITH (
```

```

        OIDS = FALSE
    )
    TABLESPACE pg_default;

ALTER TABLE public.subject_priority
    OWNER to postgres;

```

Таблица user(рис. 3.9)

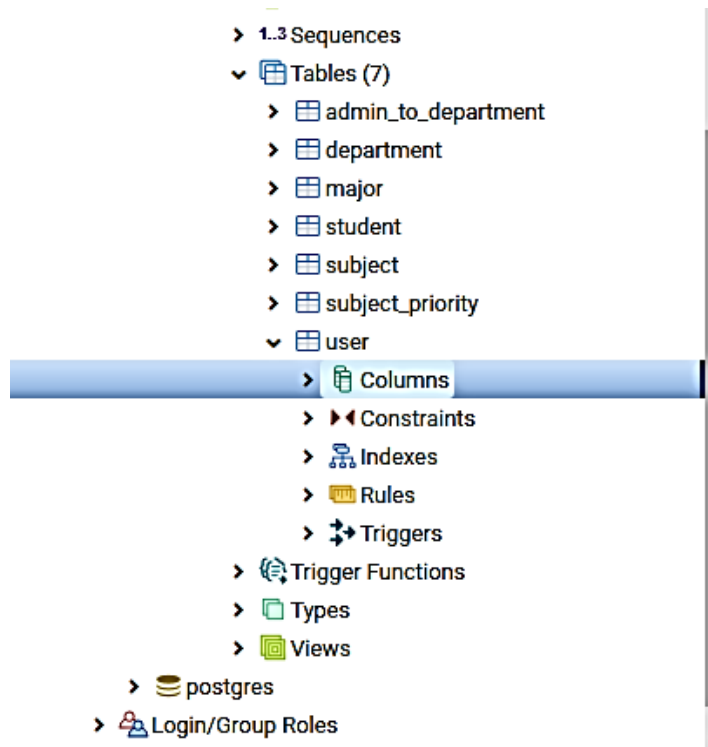


Рис. 3.9 Таблица user

```

CREATE TABLE public."user"
(
    id bigint NOT NULL,
    name text COLLATE pg_catalog."default" NOT NULL,
    password text COLLATE pg_catalog."default" NOT NULL,
    roles text COLLATE pg_catalog."default" NOT NULL,
    active boolean NOT NULL,
    CONSTRAINT user_pkey PRIMARY KEY (id)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public."user"
    OWNER to postgres;

```

Таблица osvpr(рис. 3.10)

Type	Name	Restriction
Index	public.osv_department_id_fk	auto
Foreign key	public.osvpr.degree_id_fk	auto
Foreign key	public.osvpr.department_id_fk	auto
Foreign key	public.osvpr.major_id_fk	auto
Foreign key	public.osvpr.user_id_fk	auto
Primary key	public.osvpr_pkey	auto
Foreign key	public.student.osvpr_id_fk	normal

Рис. 3.10 Таблица osvpr

```
CREATE TABLE public.osvpr
-- list all osvitniy programs
(
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1 START
        1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),
    name text COLLATE pg_catalog."default" NOT NULL,
    code text COLLATE pg_catalog."default" NOT NULL,
    department_id bigint NOT NULL,
    major_id bigint NOT NULL,
    degree_id bigint NOT NULL,
    user_garant_id bigint NOT NULL,
    CONSTRAINT osvpr_pkey PRIMARY KEY (id),
    CONSTRAINT degree_id_fk FOREIGN KEY (degree_id)
        REFERENCES public.degree (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT department_id_fk FOREIGN KEY (department_id)
        REFERENCES public.department (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT major_id_fk FOREIGN KEY (major_id)
        REFERENCES public.major (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION,
    CONSTRAINT user_id_fk FOREIGN KEY (user_garant_id)
        REFERENCES public."user" (id) MATCH SIMPLE
        ON UPDATE NO ACTION
```



```

        ON DELETE NO ACTION

    )
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.osvpr
    OWNER to postgres;
-- Index: fki_department_id_fk

-- DROP INDEX public.fki_department_id_fk;
-- non use name index
CREATE INDEX osv_department_id_fk
    ON public.osvpr USING btree
    (department_id ASC NULLS LAST)
    TABLESPACE pg_default ;

```

Таблица priory(рис. 3.11)

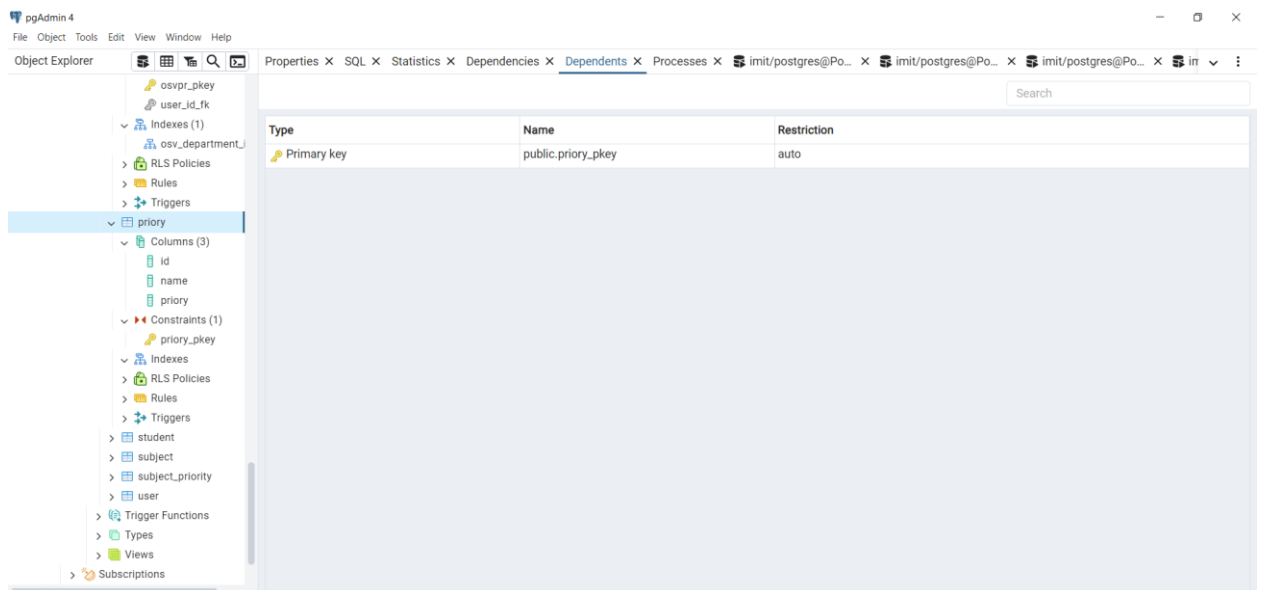


Рис. 3.11 Таблица priory

```

CREATE TABLE public.priory
-- list all prioritetov (from 1 to 3)
(
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1
START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),
    name text COLLATE pg_catalog."default" NOT NULL,
    priory bigint NOT NULL,
    CONSTRAINT priory_pkey PRIMARY KEY (id)
)

```

```

WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.priory
    OWNER to postgres;

```

Таблиця degree(рис. 3.12)

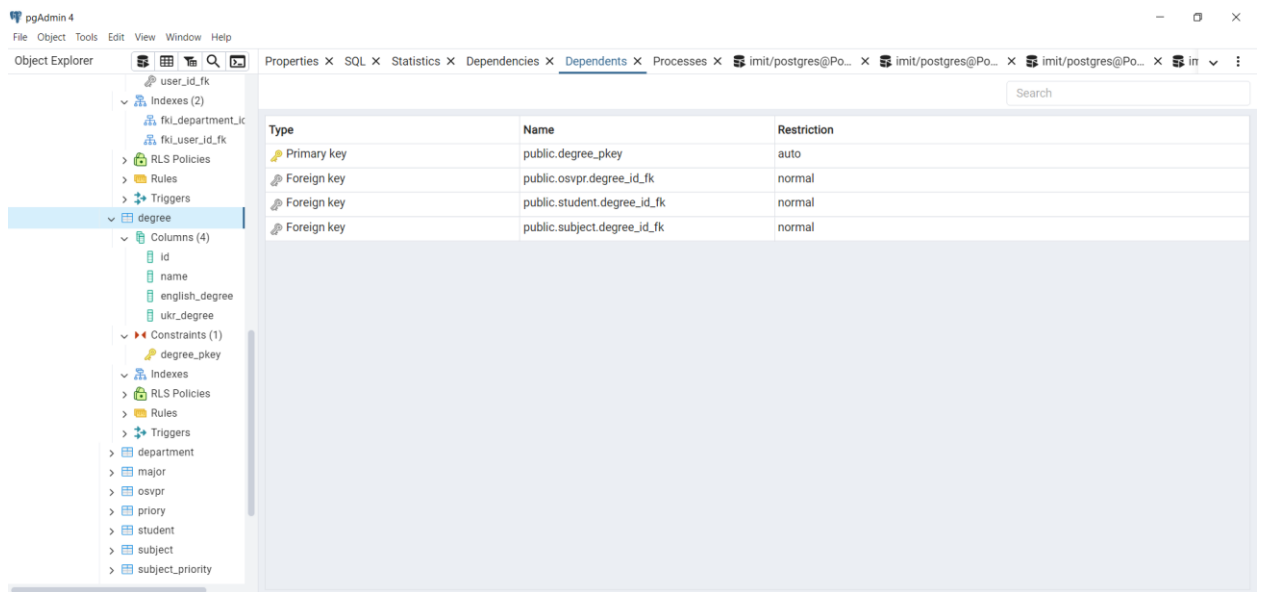


Рис. 3.12 Таблиця degree

```

CREATE TABLE public.degree
-- list all urovniy osvity (bak, mag, phd)
(
    id bigint NOT NULL GENERATED BY DEFAULT AS IDENTITY ( INCREMENT 1
START 1 MINVALUE 1 MAXVALUE 9223372036854775807 CACHE 1 ),
    name text COLLATE pg_catalog."default" NOT NULL,
    english_degree text COLLATE pg_catalog."default" NOT NULL,
    ukr_degree text COLLATE pg_catalog."default" NOT NULL,
    CONSTRAINT degree_pkey PRIMARY KEY (id)
)
WITH (
    OIDS = FALSE
)
TABLESPACE pg_default;

ALTER TABLE public.degree
    OWNER to postgres;

```

Загальна схема БД(рис. 3.13)

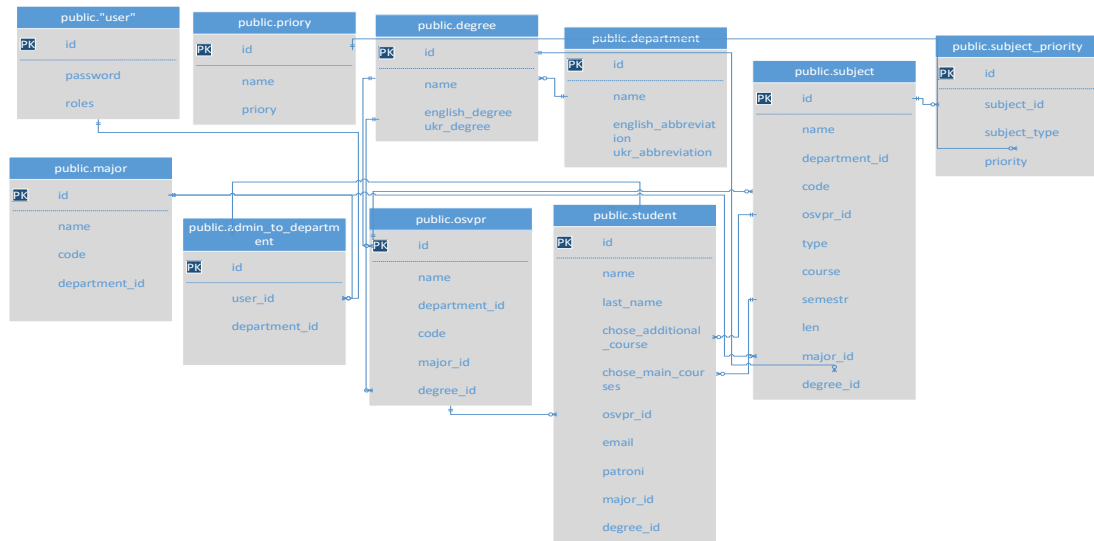


Рис.3.13 Загальна схема БД

3.4. Загальна структура пакетів проекту та класів

До загальної структури пакетів проекту програмного додатку відносяться наступні:

1. project – головний пакет;
2. project.config - пакет для конфігурації кодування пароля і настроювання захисту програми;
3. project.controller - пакет з контролерами;
4. project.dto - пакет з POJO, які передаються на фронт енд або з нього;
5. project.entity - пакет класів, які зв'язані з базою даних;

6. `project.entity.enums` – енуми, які використовуються в ентиті;
7. `project.exception` – для відловлювання експешинів;
8. `project.mapper` – для мапінга одного об'єкта до іншого;
9. `project.projection` - класи, які використовуються з базою даних, якщо потрібно отримати не всі або специфічні колонки;
10. `project.repository` – класи репозиторії;
11. `project.service` – усі сервіси ;
12. `project.utils` – додаткові класи (клас з константами).

До загального переліку класів відносяться

1. `project.App` – стартовий клас. При старті програми додаткову конфігураційну інформацію бере з ресурсів `application.yaml`
2. `project.config . BCryptPasswordEncoderConfig` – при вході в програму паролі перевіряються завдяки енкодингу. Паролі не зберігаються в чистому виді, всі закодовані
3. `project.config .SecurityConfig` - налаштування захисту програми.
4. `project.controller.AdditionalSubjectController` – відображає лист з додатковими предметами, та зберігає предмети з обраними пріоритетами
5. `project.controller.ChangePasswordController` – для зміни пароля
6. `project.controller.HomeController` – відображає на екрані головну сторінку, та якщо залогінений користувач - студент, відображає його ім'я, факультет, спеціальність, курс
7. `project.controller.MainSubjectController` - відображає лист з головними предметами

8. `project.controller.TestPasswordController` – відображає закодований пароль
9. `project.controller.UploadingStudentsController` – завантаження файлу зі студентами
10. `project.dto.StudentDto` – для мапінга полів з файлу зі студентами
11. `project.dto.SubjectDto` – для відображення полів дисциплін на фронт енді
12. `project.dto.SubjectPriorityDto` - для передачі пріоритетів в контролер
13. `project.entity.AdminToDepartment`- для зв'язування з таблицею `admin_to_department`
14. `project.entity.AdminToDepartmentId` - ключ для `AdminToDepartment`
15. `project.entity.Department` - для зв'язування з таблицею `department`
16. `project.entity.Major`- для зв'язування з таблицею `major`
17. `project.entity.Student`- для зв'язування з таблицею `student`
18. `project.entity.Subject`- для зв'язування з таблицею `subject`
19. `project.entity.SubjectPriority`- для зв'язування з таблицею `subject_priority`
20. `project.entity.User`- для зв'язування з таблицею `user`
21. `project.mapper.StudentDtoToStudentEntityMapper` – мапінг з `StudentDto` на `Student` об'єкти
22. `project.mapper.StudentToUserMapper` – мапінг з `Student` на `User` об'єкти

- 23.project.projection .SubjectProjection – для отримання тільки двох полів
- 24.project.repository.AdminToDepartmentRepository – модифікація та отримання даних в таблиці admin_to_department
- 25.project.repository.DepartmentRepository - модифікація та отримання даних в таблиці department
- 26.project.repository.MajorRepository - модифікація та отримання даних в таблиці major
- 27.project.repository.StudentRepository - модифікація та отримання даних в таблиці admin
- 28.project.repository.SubjectPriorityRepository - модифікація та отримання даних в таблиці student
- 29.project.repository.SubjectRepository - модифікація та отримання даних в таблиці subject_priority
- 30.project.repository.UserRepository - модифікація та отримання даних в таблиці user
- 31.project.service.FileParserService – з файлу отримує POJO
- 32.project.service.FileProcessingService - робота з файлом
- 33.project.service.NotificationService – для відправлення повідомлень з паролем на пошту
- 34.project.service.StudentService – робота з об'єктами Student
- 35.project.service.SubjectPriorityService – для роботи з пріоритетами предметів
- 36.project.service.SubjectService - для роботи з предметами
- 37.project.service.UserService – для роботи з користувачами

Таким чином до тестової розробка, яка не включає до себе елементи обробки інтерфейсу та елементи дизайну сайту включено 37 класів, що груповані у 12 пакетів.

ВИСНОВКИ

Таким чином, в результаті виконання роботи встановлено та запропоновано наступні рішення:

Для вибору певної БД було проаналізовано різноманітні типи баз даних, наведено їх історичний опис та надано перелік особливостей їх використання та застосування.

Серед типів БД окреслені наступні

- Реляційні БД.
- Об'єктні БД.
- NoSQL БД.

Оскільки реляційна БД концептуально проста, вона дозволяє реалізовувати невеликі і прості (і тому легкі для створення) бази даних, навіть персональні, сама можливість реалізації яких ніколи навіть і не розглядалася в системах з ієрархічною чи мережною моделлю.

Недоліком реляційної моделі даних є надмірність по полях (для створення зав'язків між різними об'єктами бази даних).

Практично всі існуючі на сьогоднішній день комерційні бази даних і програмні продукти для їх створення використовують реляційну модель даних

В роботі наведено опис процесу налаштування та розгортання реляційної БД PostgreSQL, використання pgAdmin, запропоновано на наведено опис програмного забезпечення для створення БД ІФМІТ в системі PostgreSQL.

В результаті аналізу положень та процедур, які затверджені у ЛНУ імені Т. Шевченка та на засадах ґрунтовних спілкувань з представниками навчального відділу знайдені вимоги до цієї БД.

Для виконання цих вимог було спроектовано БД, яка складається з 7 таблиць та додатковий програмний модуль на мові Java з 32 класів, які згруповано по 12 пакетам.

Таким чином, головна мета роботи – розробка БД, яка забезпечить створення системи визначення переліку вибіркових дисциплін, була виконана.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Типи баз даних: особливості, відмінності та приклади [Online] – <https://dou.ua/lenta/articles/types-of-databases/>
2. Основні відомості про ба [Online] – <https://support.microsoft.com/uk-ua/topic/%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%96-%D0%B2%D1%96%D0%B4%D0%BE%D0%BC%D0%BE%D1%81%D1%82%D1%96-%D0%BF%D1%80%D0%BE-%D0%B1%D0%B0%D0%B7%D0%B8-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85-a849ac16-07c7-4a31-9948-3c8c94a7c204>
3. Klein, B. An Introduction to IMS. Second Edition / B. Klein, R. Long, K. Ray // IBM Press, 2012, 622 p.
4. Codd, E.F. A Relational Model of Data for Large Shared Data Banks / E.F. Codd // Communications of the ACM. – 1970. – Vol. 13, num. 6. – P. 377-387.
5. Нормальні форми бази даних [Online] – <https://javarush.com/ua/quests/lectures/ua.questhibernate.level17.lecture02>
6. Редько В.Н. Реляційні бази даних: табличні алгебри та SQL-подібні мови / В.Н. Редько, Ю.Й. Брона, Д.Б. Буй, С.А. Поляков. – К.: Видавничий дім “Академперіодика”, 2001. – 196 с.
7. Поняття бази даних і систем керування базами даних, їх призначення [Online] – <https://www.miyklas.com.ua/p/informatica/10-klas/sistemi-keruvannia-bazami-danikh-326161/skbd-reلياتciini-bazi-danikh-326453/re-ecb32162-0c19-4c15-bc5d-61d53b7add6b>
8. Моделі даних. Ієрархічна модель даних. Мережна модель даних. Реляційна модель даних [Online] – https://geoknigi.com/book_view.php?id=589

9. Cook, W. Object-Oriented Programming Versus Abstract Data Types [Online] – Hewlett-Packard Laboratories. Available from: www.cs.utexas.edu/users/wcook/papers/OOPvsADT/CookOOPvsADT90.pdf
10. Грей П. Логика, алгебра и базы данных. – М.: Машиностроение, 1989. – 360 с.
11. Bubel R. A Formalisation of Java Strings for Program Specification and Verification [Online] . – Available from: <http://202.154.59.182/mfile/files/Information%20System/Software%20Engineering%20and%20Formal%20Methods%3B%209th%20SEFM%202011/Chapter%208%20A%20Formalisation%20of%20Java%20Strings%20for%20Program%20Specification%20and%20Verification.pdf>
12. Загальна теорія баз даних [Online] – https://stud.com.ua/35677/informatika/zagalna_teoriya_danih#google_vignette
13. Буй Д.Б. Огляд сучасної теорії нормалізації у реляційних базах даних / Д.Б. Буй, А.В. Пузікова // Матеріали XVII Міжнародної конференції “Проблеми теоретичної кібернетики” – 20 червня 2014 р.). С. 39-43.
14. Основи баз даних [Online] – <https://step.org.ua/konspekt/base/tema1>
15. Конспект з теорії по БД [Online] – <https://www.scribd.com/document/620744777/%D0%A5%D0%B0%D1%80%D1%87%D0%B5%D0%BD%D0%BA%D0%BE-%D0%9A%D0%BE%D0%BD%D1%81%D0%BF%D0%B5%D0%BA%D1%82-%D0%B7-%D1%82%D0%B5%D0%BE%D1%80%D1%96%D1%97-%D0%BF%D0%BE-%D0%91%D0%94>
16. База даних [Online] – https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B7%D0%B0_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85
17. Нормалізація баз даних [Online] – <https://uk.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%BD%D0%B8%D1%85>

D0%BB%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F_%D0%B1%D
0%B0%D0%B7_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85

18. Теорія нормалізації реляційної моделі даних [Online] –
https://elearning.sumdu.edu.ua/free_content/lectured:89b3d175c06a6b137e410cb14821d0e94549ad5a/20151013153156/44494/index.html

19. Основи розробки баз даних [Online] –
<https://support.microsoft.com/uk-ua/topic/%D0%BE%D1%81%D0%BD%D0%BE%D0%B2%D0%B8-%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B8-%D0%B1%D0%B0%D0%B7-%D0%B4%D0%B0%D0%BD%D0%B8%D1%85-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5>

20. Великі дані [Online] –
https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%BB%D0%B8%D0%BA%D1%96_%D0%B4%D0%B0%D0%BD%D1%96

21. Codd E.F. A Relational Model of Database Management : version 2. /
E. Codd. – Addison- Wesley, 1990. – 567 p.

ДОДАТОК

Міністерство освіти і науки України
Державний заклад «Луганський національний університет
імені Тараса Шевченка»

Факультет (інститут) Навчально-науковий інститут фізики, математики та
інформаційних технологій
(повна назва)

Кафедра Інформаційних технологій та систем
(повна назва)

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання програмної розробки (ПР):

" ПРОЄКТУВАННЯ БАЗИ ДАНИХ ПЕРЕЛІКУ ВИБІРКОВИХ ДИСЦИПЛІН "

Полтава 2024

ЗМІСТ

<u>ВСТУП</u>	63
<u>1. ХАРАКТЕРИСТИКА ОБ'ЄКТА</u>	63
<u>2. ПРИЗНАЧЕННЯ ТОВАРІВ</u>	64
<u>3. ОСНОВНІ ВИМОГИ ДО ПРОГРАМНОГО КОМПЛЕКСУ</u>	64
<u>4. ТЕХНІКО - ЕКОНОМІЧНІ ВИМОГ ДО КІНЦЕВОГО ПРОДУКТУ</u>	66
<u>5. ВИМОГИ ДО МАТЕРІАЛІВ І КОМПЛЕКТУЮЧИХ</u>	66
<u>6. ЕТАПИ ВИКОНАННЯ ПР</u>	66
<u>7. ПРИЙОМ</u>	67
<u>8. ПОРЯДОК ВНЕСЕННЯ ЗМІН ДО ТЕХНІЧНЕ ЗАВДАННЯ, ЩО ЗАТВЕРДЖЕНО</u>	67

ВСТУП

1.1 Найменування: Проектування клієнт-серверної бази даних для веб-системи визначення переліку вибірових дисциплін.

1.2 Шифр ПР: ІТС.ІП04.0220

1.3 Підстава для виконання ПР: Підставою для виконання даної розробки є необхідність розробки бази даних для використання в процесі організації вільного вибору дисциплін.

1.4 Терміни розробки:

1.4.1 Початок 30 жовтня 2023 р.

1.4.2 Закінчення січня 2025р.

1.5 Фінансується за рахунок коштів замовника. Умови фінансування - за договором – безкоштовна / а і протоколу узгодження ціни – не надано.

1. ХАРАКТЕРИСТИКА ОБ'ЄКТА

1.1. Розроблюваний програмний комплекс повинен вдути орієнтований на використання у складі майбутньої ВЕБ системи. До складу об'єкта, який створюється має входити:

1.1.1. Розроблюваний програмний комплекс,

1.2. До вхідної інформації відносяться вимоги та розпорядження системи організації навчального процесу у ЛНУ імені Тараса Шевченка.

1.3. До вихідної інформації належить веб сторінки з використанням бази даних та база даних:

Таблиці з контингентом студентів.

Таблиці переліку ОП

Таблиці різноманітних ролей БД для підтримки метдів редагування БД

Таблиці зі списками вибірових дисциплін

Таблиці з пріоритетами для вибірових дисциплін;

2. ПРИЗНАЧЕННЯ ТОВАРІВ

2.1. Призначення: веб додаток повинен, на засадах певної бази даних забезпечувати збереження інформації щодо визначення переліку вибірових дисциплін.

2.2. Основні критерії ефективності.

2.2.1. Зручний інтерфейс – не пред’являється .

2.2.2. Програмний комплекс повинен:

2.2.2.1. Давати можливість переглянути, додати та перевірити контингент студентів;

2.2.2.2. Давати можливість переглянути, додати перелік ОП та перевірити вибірові дисципліни певної ОП;

2.2.2.3. Враховувати наявність двох блоків вибірових дисциплін.

2.2.2.4. Враховувати наявність різноманітних кредитів (обсягу) вибірових дисциплін.

2.2.2.5. Враховувати пріоритети (бажаність) вибірових дисциплін.

2.2.2.6. Враховувати можливість різноманітних статусів процесу вибору вибірових дисциплін.

2.2.2.7. Враховувати можливість запровадити алгоритм формування груп студентів з урахування пріоритетів (бажань) студентів.

2.2.2.8. Враховувати наявність рівнів освіти: бакалавр, магістр, аспірант.

2.2.2.9. Надавати контрольований доступ до певних таблиць з урахуванням різноманітних ролей: студент - тільки використання, оператор - перегляд, менеджер-додавання та редагування.

2.3. Основною функцією програмного комплексу є створення бази даних.

3. ОСНОВНІ ВИМОГИ ДО ПРОГРАМНОГО КОМПЛЕКСУ

3.1. Загальні вимоги

3.1.1. Програмний комплекс працює під керування будь-якого ВЕБ серверу ;

- 3.1.2. Вимоги до апаратного забезпечення персонального комп'ютера - не передбачені і можуть встановлюватися розробником програмного комплексу;
- 3.1.3. Програмний комплекс повинен мати зручний інтерфейс;
- 3.1.4. До складу програмного комплексу входить набір початкових (тестових) HTML сторінок;
- 3.1.6. До складу програмного комплексу необхідно додати метод швидкого розгортання БД та методи імпорту списків студентів та списків дисциплін
- 3.2. Додаткові вимоги
 - 3.2.1. Мова програмування – на вибір розробника.
 - 3.2.2. Вимоги до ліцензійного ПЗ не передбачаються і вирішуються замовником.
- 3.3. Вимоги до складу і архітектури
 - 3.3.1. Розробник самостійно вибирає склад і виконує розробку архітектури ПР
 - 3.3.2. Особливих умов до складу і архітектури ПР не передбачено.
- 3.4. Вимоги до якості і надійності
 - 3.4.1. Програмний комплекс повинен надійно працювати/
 - 3.4.2. Розробник вибирає технічні характеристики персонального комп'ютера, налаштовує системне програмне забезпечення.
 - 3.4.3. Розробник гарантує роботу програмного комплексу без збоїв і перенастроювань.
 - 3.4.4. Виконавець гарантує придбання додаткового обладнання (кондиціонер, UPS) за власні кошти.
- 3.5. Вимоги до експлуатації
 - 3.5.1. Розробник використовує персональний комп'ютер, на якому програмний комплекс повинен надійно працювати.
 - 3.5.2. Персональний комп'ютер буде задіяно в розрахунках і буде встановлений в приміщенні обчислювального центру.

4. ТЕХНІКО - ЕКОНОМІЧНІ ВИМОГ ДО КІНЦЕВОГО ПРОДУКТУ

Вартість робіт по розробці даної ПР визначається згідно з договором на розробку. Вартість пропонованих аналогів повинна забезпечити економічну доцільність їх застосування.

5. ВИМОГИ ДО МАТЕРІАЛІВ І КОМПЛЕКТУЮЧИХ

В процесі розробки програмного комплексу можливе використання програмних java - бібліотек.

5.1. Вимоги до екологічної безпеки при експлуатації.

Не пред'являються.

5.2. Спеціальні вимоги до кінцевого продукту.

Не пред'являються.

5.3. Вимоги до безпеки для населення при експлуатації продукції.

Не пред'являються.

6. ЕТАПИ ВИКОНАННЯ ПР

Етапи виконання ПР можуть уточнюватися згідно календарного плану робіт за погодженням між замовником і виконавцем

№	Етапи виконання роботи	Термін виконання і обсяг робіт	Звітні матеріали
1	Аналіз розробки програмного комплексу та розробка першої версії. Аналіз вимог. Розробка структури. Попереднє тестування.		Фрагмент програмного комплексу на ЕОМ замовника, який виконує всі основні функції і звітна документація п.8.2
2	Коригування структури. Розробка допоміжних функцій. Розробка остаточної версії програмного комплексу і його обробки. Тестування.		Готовий програмний комплекс на ЕОМ замовника і звітна документація п.8.2
3	Доопрацювання окремих модулів і навчання користувачів. Розробка звітних матеріалів по п.8 цього ТЗ		Звітні матеріали згідно з пунктом 8.

7. ПРИЙОМ

7.1. Необхідні вимоги для впровадження ПР і завершення робіт.

Оцінка результатів розробки і доцільність її продовження здійснюється замовником за поданням наступних матеріалів:

- встановлено програмний комплекс на ЕОМ замовника;
- перелік файлів на резервному носії;
- короткий опис роботи ПР і опис всіх файлів, які необхідні для роботи ПР.
- перелік документів
 - Технічне завдання
 - Пояснювальна записка

7.2. Перелік звітних документів, необхідних для прийняття етапів роботи:

- короткий опис результатів етапу у вигляді анотованого звіту (для 1 та 2 етапів);
- частковий програмний комплекс на ЕОМ замовника згідно календарного плану робіт;
- акт приймання продукції.

Звітні матеріали подаються у вигляді звітів на папері по ДСТУ

7.3. Загальний перелік до прийому звітних документів, макетів, експериментальних зразків.

До прийому пред'являються: акт здачі-приймання продукції, акт впровадження ПР.

8. ПОРЯДОК ВНЕСЕННЯ ЗМІН ДО ТЕХНІЧНЕ ЗАВДАННЯ, ЩО ЗАТВЕРДЖЕНО.

Дане технічне завдання може уточнюватися в процесі розробки ПР при узгодженні сторін з оформленням доповнень до ТЗ.

Лістинг коду

```
package project.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

@Configuration
public class BCryptPasswordEncoderConfig {

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}

package project.config;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
    org.springframework.security.config.annotation.authentication.b
    uilders.AuthenticationManagerBuilder;

import
    org.springframework.security.config.annotation.web.builders.Http
    pSecurity;

import
    org.springframework.security.config.annotation.web.configuratio
    n.EnableWebSecurity;

import
    org.springframework.security.config.annotation.web.configuratio
    n.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import
    org.springframework.security.core.userdetails.UsernameNotFoundE
    xception;
import project.repository.UserRepository;

@EnableWebSecurity
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    UserRepository userRepository;

    @Override
```

```

protected void configure(AuthenticationManagerBuilder auth) throws
    Exception {
    auth.userDetailsService(new UserDetailsService() {
        @Override
        public UserDetails loadUserByUsername(String username) throws
            UsernameNotFoundException {
            return userRepository.findByName(username);
        }
    });
}

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests()
        .antMatchers("/file-uploading/**").hasRole("ADMIN")
        .antMatchers("/testPassword/**", "/change-
password").hasRole("ADMIN")
        .anyRequest().authenticated()
        .and()
        .formLogin().defaultSuccessUrl("/", true).permitAll()
        .and()
        .logout();
    http.csrf().disable();
}
}

```

```

package project.controller;

```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import project.dto.SubjectPriorityDto;
import project.service.SubjectPriorityService;
import project.service.SubjectService;

```

```

import static project.entity.enums.SubjectType.ADDITIONAL;

```

```

@Controller
@RequestMapping("/additional-subject")
public class AdditionalSubjectController {

```

```

    @Autowired
    private SubjectService subjectService;

```

```

    @Autowired
    private SubjectPriorityService subjectPriorityService;

```

```

    @GetMapping
    public String showList(Authentication authentication, Model model) {
        model.addAttribute("subjects",

            subjectService.getSubjectsForStudent(authentication.getName(),
            ADDITIONAL));

        return "additionalSubjectList";
    }

    @PostMapping
    public String create(SubjectPriorityDto subjectPriorityDto,
        Authentication authentication) {
        subjectPriorityService.processPriorities(subjectPriorityDto,
            authentication.getName(), ADDITIONAL);

        return "successfulPage";
    }
}

```

```

package project.controller;

```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import project.service.UserService;

```

```

@RestController
@RequestMapping("change-password")
public class ChangePasswordController {

    @Autowired
    private UserService userService;

    @GetMapping("/{userId}/{newPassword}")
    public String changePassword(@PathVariable Long userId, @PathVariable
        String newPassword) {
        userService.changePassword(userId, newPassword);
        return "Password was changed";
    }
}

```

```

package project.controller;

```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import project.entity.Student;
import project.repository.StudentRepository;
import project.repository.UserRepository;
import project.service.StudentService;

@Controller
@RequestMapping("/")
public class HomeController {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private StudentRepository studentRepository;

    @Autowired
    private StudentService studentService;

    @GetMapping
    public String hello(Model model) {
        Authentication authentication =
            SecurityContextHolder.getContext().getAuthentication();
        boolean hasStudentRole = authentication.getAuthorities().stream()
            .anyMatch(role ->
                role.getAuthority().equalsIgnoreCase("ROLE_STUDENT"));

        if (hasStudentRole) {
            Student student =
                studentService.getStudent(authentication.getName());
            String message = String.format("Вы %s студент факультета %s
                специальности %s курса %s " +
                    "степень %s",
                student.getName() + " " + student.getLastName(),
                student.getDepartment().getName(),
                student.getMajor().getName(),
                student.getCourse(), student.getDegree().toString());
            model.addAttribute("student", message);
        }

        return "home";
    }
}

package project.controller;

import org.springframework.beans.factory.annotation.Autowired;

```

```

import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import project.service.SubjectService;

import static project.entity.enums.SubjectType.ADDITIONAL;
import static project.entity.enums.SubjectType.MAIN;

@Controller
@RequestMapping("/main-subject")
public class MainSubjectController {

    @Autowired
    private SubjectService subjectService;

    @GetMapping
    public String showList(Authentication authentication, Model model) {
        model.addAttribute("subjects",

            subjectService.getSubjectsForStudent(authentication.getName(),
            MAIN));

        return "mainSubjectList";
    }
}

```

```

package project.controller;

```

```

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

```

```

@RestController
@RequestMapping("/testPassword")
public class TestPasswordController {

    @GetMapping("/{password}")
    public String testPassword(@PathVariable String password) {
        BCryptPasswordEncoder bCryptPasswordEncoder = new
            BCryptPasswordEncoder();
        return bCryptPasswordEncoder.encode(password);
    }
}

```

```

package project.controller;

```



```

import lombok.NonNull;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.multipart.MultipartFile;
import project.service.FileProcessingService;

import java.security.Principal;

@Controller
@RequestMapping("file-uploading")
public class UploadingStudentsController {

    @Autowired
    private FileProcessingService fileProcessingService;

    @GetMapping
    public String getName() {
        return "upload";
    }

    @PostMapping
    public String uploadFile(@NonNull @RequestParam("file") MultipartFile
        file, Principal principal) {
        fileProcessingService.uploadFile(file, principal.getName());
        return "successfulPage";
    }
}

package project.dto;

public class FormDto {

    private String name;

    private String list;

    public FormDto() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}

```

```

    public String getList() {
        return list;
    }

    public void setList(String list) {
        this.list = list;
    }
}

package project.dto;

import com.opencsv.bean.CsvBindByName;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

@AllArgsConstructor
@NoArgsConstructor
@ToString
@Data
public class StudentDto {

    @CsvBindByName(required = true, column = "Name")
    private String name;

    @CsvBindByName(required = true, column = "Last name")
    private String lastName;

    @CsvBindByName(required = true)
    private String patronymic;

    @CsvBindByName(required = true)
    private String degree;

    @CsvBindByName(required = true)
    private String email;

    @CsvBindByName(required = true)
    private short course;

    @CsvBindByName(required = true)
    private short semester;
}

package project.dto;

```

```

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class SubjectDto {

    private Long id;

    private String name;
}

package project.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import java.util.List;

@Data
@AllArgsConstructor
@NoArgsConstructor
public class SubjectPriorityDto {

    private List<Long> firstPriority;

    private List<Long> secondPriority;

    private List<Long> thirdPriority;

    private List<Long> fourthPriority;
}

```

