Державний університет інформаційно-комунікаційних технологій
Навчально-науковий інститут Інформаційних технологій
Кафедра Штучного інтелекту

«Затверджую»
Директор Навчально-наукового інституту
Інформаційних технологій
_____Нестеренко К. С.
"25" серпня 2025 р.

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ ТА ЗАВДАННЯ

на курсову роботу з дисципліни

«ПРОЄКТУВАННЯ І СУПРОВІД БАЗ ДАНИХ ТА ЗНАНЬ»

для здобувачів другого (магістерського) рівня вищої освіти
спеціальність Комп'ютерні науки
освітньо-наукова програма Штучний інтелект

Київ-2025

Методичні рекомендації та завдання, щодо виконання курсової роботи з навчальної дисципліни «Проєктування і супровід баз даних та знань» для здобувачів другого (магістерського) рівня вищої освіти, спеціальності Комп'ютерні науки.

Розробник: Шантир А.С., к.т.н., доцент кафедри Штучного інтелекту.

Методичні рекомендації схвалено на засіданні кафедри Штучного інтелекту.
Протокол №__1__ від 26 серпня 2025 року

Завідувач кафедри Штучного інтелекту
_____(Зінченко О.В.)

СТРУКТУРА КУРСОВІ РОБОТИ І ПРАВИЛА ЇЇ ОФОРМЛЕННЯ

1. СТРУКТУРА КР

КР складається з пояснювальної записки і графічного матеріалу. Пояснювальна записка виконується на аркушах з однієї сторони скріплених швидкозшивачем, або будь-яким іншим образом. Вона містить у собі:

1) титульний лист;
2) завдання на курсове проектування;
3) зміст;
4) вступ;
5) основна частина;
6) висновок;
7) список використаних джерел;
8) додатки (при наявності).

Титульний лист містить назву кафедри, проекту, прізвище і групу студента.

Завдання на курсову роботу являє собою лист, що містить вихідні дані для даного студентові варіанта роботи. Вихідні дані для роботи визначаються з таблиці 1 «Індивідуальне завдання» методичних вказівок.

Зміст відображає структуру КР і містить у собі назви розділів (і підрозділів), а також номери сторінок, з яких вони починаються (вертикальний стовпчик у правій частині листа).

Вступ визначає постановку задачі на роботу і повинен коротко відображати, щонайменше, три моменти: задачі, що розраховуються; шляхи рішення цих задач; мета даного курсової. Обсяг, що рекомендується, 1-1,5 сторінки.

Основна частина - це сукупність усіх розділів і підрозділів КР у яких вирішуються задачі роботи. Питання, необхідні для розгляду в даній роботі відображені в пункті 4 «Завдання на КР». Кількість розділів і їх назва не обов'язкова повинно відповідати в пункті 4 «Завдання на КР».

Висновок повинен коротко відображати основні результати роботи - прийняті проєктні рішення і найбільш важливі числові величини. Обсяг, що рекомендується - 1 сторінка.

Список використаних джерел повинний містити перелік літературних джерел (книг, статей, ДСТ, ДСТУ і т.д.) які використовувалися при розробці КР.

Додаток створюються в разі потреби застосування в КР допоміжного матеріалу, який не відповідає безпосередньо на питання роботи (комп'ютерні програми, довідкові дані і т.п.).

2. ПРАВИЛА ОФОРМЛЕННЯ КР

Нижче перераховані деякі принципи оформлення пояснювальної записки і графічного матеріалу, що необхідні для виконання студентами КР.

Нумерація сторінок відображається в правій верхній частині листа, вище тексту, арабськими цифрами. Титульний лист і завдання в нумерацію включають, але номера на них не проставляють.

Написання тексту висота 12 пт, міжстроковий інтервал - 1,5 з урахуванням відступів: ліворуч - 20 мм, праворуч - 10 мм, зверху - 20 мм, знизу - 20 мм.

Текст основної частини при необхідності розділяють на розділи, підрозділи, пункти.

Заголовки розділів пишуться симетрично текстові (центруються) прописними буквами. Вони складаються з номера розділу (арабськими цифрами) відділеного крапкою від назви. Крапку наприкінці заголовків не ставлять, заголовки не підкреслюють.

Заголовки підрозділів пишуть з абзаців, малими літерами, крім першої прописної. Номер підрозділу складається з номера розділу і підрозділу розділених крапкою і з крапкою наприкінці, наприклад: «2.3.» (третій підрозділ другого розділу*).*

Номер пункту пишеться з абзацу і складається з номеру розділу, підрозділу, пункту розділених крапками. Наприкінці ставиться крапка, наприклад: «1.3.2.» (другий пункт третього підрозділу другого розділу). Пункт заголовка не має.

Оформлення таблиць виконується в такий спосіб. Над правим верхнім кутом її пишеться слово таблиця і проставляється її номер, наприклад: «Таблиця 1.2» (друга таблиця першого розділу). Кожна таблиця повинна мати заголовок, що поміщають над таблицею посередині (нижче слова «Таблиця»). Якщо величини зазначені в графах таблиці мають одиниці виміру, то вони також повинні бути зазначені.

Рисунки повинні мати назву, розташовувану знизу і центровану щодо рисунка, і номер у межах розділу, розташовуваний по центрі під рисунком.

Основні формули, а також формули, на які повинні бути посилання нумерують арабськими цифрами в межах розділу. Номер формули повинний знаходитися з правої сторони листа на рівні формули в круглих дужках і містити номер розділу і порядковий номер формули розділені крапкою, наприклад: «(4.1)» (перша формула четвертого розділу). Пояснення значень символів і числових коефіцієнтів, що зустрічаються в КР уперше, варто приводити безпосередньо під формулою (кожен символ з нового рядка). ***Посилання на літературу*** проставляються в квадратних дужках відповідно до номеру джерела в списку використовуваних джерел, наприклад: «... згідно [4] одержуємо...».

3. ЗАГАЛЬНА ЧАСТИНА ЗАВДАННЯ

Розробити базу даних для системи **онлайн-бронювання робочих місць і кімнат у коворкінгу** з розміщенням на сервері **PostgreSQL**, використовуючи **Python** як основну мову для взаємодії з БД. База даних має включати **4–5 взаємопов'язаних таблиць** (наприклад: client, place, tariff, booking, за потреби допоміжну таблицю для журналу/статусів) та забезпечувати розв'язання **3–4 інформаційних задач** навчального характеру: (1) пошук і перегляд **вільних місць** на задану дату та часовий інтервал; (2) **створення, редагування та скасування** бронювань із перевіркою коректності даних; (3) перегляд **історії бронювань** конкретного клієнта; (4) формування **узагальненої статистики** завантаженості місць/зон за вибраний період.

Розробку **інформаційно-логічної моделі** виконати із застосуванням інструментів, дотичних до PostgreSQL (наприклад, **pgModeler**, **DBeaver**, **DataGrip** або ER-діаграми у **dbdiagram.io**), визначивши сутності, атрибути, первинні й зовнішні ключі, кардинальності зв'язків і правила предметної області (зокрема заборону конфліктних бронювань одного й того самого місця у перетинних часових інтервалах). **Фізичне моделювання** реалізувати у PostgreSQL шляхом створення DDL-скриптів (CREATE TABLE, ALTER TABLE, CHECK, UNIQUE, індекси, подання VIEW), виконуючи їх через **psql** або графічні інструменти **pgAdmin / DBeaver**.

У таблиці 1 пропонується 30 варіантів предметної області. ***Номер варіанта визначається відповідно запису у класному журналі.***

За бажанням студент *можете запропонувати свій варіант предметної області*, що ближче відповідає досвіду і перевагам. Однак по складності він повинний відповідати зазначеним вище загальним вимогам і дозволяти вирішувати всі задачі даної роботи.

Таблиця 1. Індивідуальні завдання

Вид індив. завдань	Тематика індивідуальних завдань
Курсова робота	1. Система онлайн-бронювання робочих місць у коворкінгу 2. Система бронювання аудиторій у навчальному закладі 3. Облік запису клієнтів до салону краси 4. Система бронювання номерів у мініготелі 5. Облік записів до лікаря в приватній клініці 6. Система прокату велосипедів 7. Облік оренди конференц-залів 8. Система управління замовленнями в кав'ярні

	9. Онлайн-запис до автошколи 10. Система обліку абонементів у спортзалі 11. Система бронювання спортивних майданчиків 12. Облік замовлень у невеликому інтернет-магазині 13. Система бронювання фотостудії 14. Облік оренди інструментів 15. Система реєстрації учасників тренінгів 16. Онлайн-запис на консультації 17. Система бронювання столиків у ресторані 18. Облік бібліотечних позик 19. Система управління завданнями команди 20. Облік замовлень кейтерингу 21. Система бронювання робочих змін 22. Облік заявок у сервісному центрі 23. Система бронювання комп'ютерів у комп'ютерному клубі 24. Облік онлайн-курсів 25. Система оренди парковочних місць 26. Облік доставок у службі кур'єрів 27. Система бронювання екскурсій 28. Облік заявок на технічну підтримку 29. Система оренди житла подобово 30. Система бронювання переговорних кімнат в офісі

Завдання повинно містити такі розділи, але не обмежується ними:

1. Аналіз предметної області
2. Формування вимог до бази даних
3. Розробка концептуальної моделі
4. Розробка логічної моделі бази даних
5. Фізичне проєктування бази даних
6. Забезпечення цілісності та обмежень даних
7. Оптимізація доступу до даних
8. Використання транзакцій
9. Створення подань (VIEW) та запитів
10. Висновки
11. Список використаної літератури.

4. МЕТОДИЧНІ РЕКОМЕНДАЦІЇ З НАПИСАННЯ РОЗДІЛІВ КП

Аналіз прикладної задачі та визначення потреб користувачів

У цьому підрозділі коротко описують предметну область і межі системи

(що саме автоматизуємо) та визначають ролі користувачів і їхні потреби. Для прикладу “онлайн-бронювання коворкінгу” потрібно назвати користувачів (клієнт, адміністратор, менеджер/власник), стисло перелічити, що кожен робить (перегляд вільних місць, створення/скасування бронювання, перегляд статистики), і на цій основі сформувати перелік даних для БД: клієнти (ім’я, email, телефон, компанія), місця/кімнати (назва, тип, місткість, зона), бронювання (дата, час початку/закінчення, client_id, place_id, статус), тарифи (тип тарифу, ціна, тип місця).

Формування простих вимог до майбутньої БД

Тут описують вимоги у формі “БД/система повинна…”, перетворюючи потреби користувачів у конкретні правила зберігання та контролю даних. Для коворкінгу треба зафіксувати обов’язковість ключових полів (name/email/phone у клієнта; date/start_time/end_time у бронюванні), наявність зв’язків (бронювання обов’язково має client_id і place_id), допустимі значення статусу (active/cancelled/finished), а також ключову вимогу уникнення конфліктів: одне місце не може мати два активні бронювання, що перетинаються за часом, що далі реалізують обмеженнями або логікою застосунку/тригерами.

Створення концептуальної карти предметної області

У підрозділі подають концептуальну модель без SQL: сутності, їх атрибути, зв’язки та правила предметної області. Для прикладу виділяють сутності Клієнт, Місце, Бронювання, Тариф; задають атрибути (id, контактні дані, параметри місця, дата/час/статус бронювання, параметри тарифу); описують зв’язки (Клієнт 1:N Бронювання, Місце 1:N Бронювання, Тариф 1:N Місце або за потреби N:M через проміжну сутність); і формулюють бізнес-правила (заборона подвійних бронювань, обов’язковість прив’язки, коректність статусів, відповідність тарифу типу місця).

Виділення ключових об’єктів і взаємодій

Тут пояснюють, що центральним об’єктом системи є Бронювання, а інші сутності забезпечують його створення, контроль і аналіз. Далі наводять 3–5 типових сценаріїв кроками: створення бронювання клієнтом (вибір дати/часу → показ вільних місць → вибір місця → перевірка конфлікту → запис зі статусом active), скасування/редагування адміністратором (пошук → зміна статусу → місце стає доступним), перегляд історії клієнта (майбутні/минулі бронювання), перегляд статистики менеджером (агрегації по бронюваннях і місцях), обов’язково згадуючи обмеження взаємодій (немає перетинів, дані не видаляються зі звітності).

Побудова логічної структури для невеликої системи

У цьому підрозділі описують логічну модель БД: перелік таблиць, їх поля, первинні та зовнішні ключі й основні обмеження. Для коворкінгу доцільні таблиці client, place, tariff, booking із PK (client_id/place_id/tariff_id/booking_id), FK (booking.client_id → client, booking.place_id → place, place.tariff_id → tariff), і полями для дати/часу, місткості, ціни та статусу; також зазначають, які поля обов'язкові, які унікальні (email/phone) і які потребують перевірок (status з фіксованого набору, price ≥ 0, capacity > 0).

Моделювання історії змін (created_at, updated_at)

Тут показують мінімальний аудит змін без складних механізмів: у кожну основну таблицю додають created_at і updated_at та пояснюють їх роль. created_at дозволяє аналізувати появу нових клієнтів, місць, тарифів і момент створення бронювання (корисно для статистики), а updated_at фіксує останні зміни контактів, параметрів місць, цін/тарифів і статусів бронювань, що допомагає контролювати актуальність даних і готує модель до можливого розширення журналом змін у майбутньому.

Створення таблиць для навчальної задачі (SQL DDL)

У підрозділі наводять SQL-скрипти CREATE TABLE для всіх таблиць із типами даних, NOT NULL, PK та FK, а після кожного блоку дають коротке пояснення вибору полів і зв'язків. Важливо дотримати порядок створення таблиць через зовнішні ключі (спочатку tariff, потім place, далі client і booking або інший коректний порядок), окремо зазначити опційні поля (наприклад, tariff_id у booking для фіксації тарифу саме в момент бронювання) та забезпечити узгоджені назви полів і типів.

Додавання простих ключів і мінімальних обмежень

Тут демонструють підсилення цілісності даних через ALTER TABLE: додають UNIQUE для client.email і client.phone, CHECK для status (active/cancelled/finished), CHECK для tariff.price (≥0) і place.capacity (>0), а також показують базовий захист від дублювання бронювань, наприклад унікальністю комбінації (place_id, date, start_time, end_time). Обов'язково пояснюють, що повна перевірка "перетину інтервалів" для одного place_id складніша і зазвичай реалізується тригерами або логікою застосунку, але для навчальної роботи допускається спрощений варіант.

Створення одного простого індексу та перевірка (EXPLAIN)

У цьому підрозділі обґрунтовують індекс через типовий запит (наприклад, пошук клієнта за email або вибірка бронювань за місцем і датою), наводять CREATE INDEX і показують EXPLAIN/EXPLAIN ANALYZE, коротко

пояснюючи різницю між Seq Scan і Index Scan. Висновок має бути практичним: індекси прискорюють часті операції пошуку та фільтрації в реальній роботі системи бронювань, особливо коли обсяг даних зростає.

Початкові приклади використання транзакцій

Тут демонструють атомарність операцій: кілька пов'язаних змін виконуються разом або не виконуються взагалі. Наводять 2–3 приклади: успішне створення клієнта і бронювання в одній транзакції (BEGIN … COMMIT), приклад з помилкою (порушення NOT NULL/UNIQUE → транзакція не фіксується), та приклад зміни статусу бронювання разом із записом у таблицю логів, пояснюючи, що при збоях жодна частина операції не залишиться "напіввиконаною".

Створення простого VIEW та приклади запитів

У підрозділі створюють VIEW як збережений SELECT із JOIN-ами між booking, client, place, tariff, щоб спростити типові вибірки для адміністратора та менеджера. Після CREATE VIEW наводять 3–4 приклади використання: список активних бронювань на день, історія бронювань клієнта, завантаженість певної зони, звіт за періодом із тарифами; також коротко пояснюють використання LEFT JOIN, якщо деякі бронювання можуть не мати tariff_id.

Висновок

Висновок може містити коротку оцінку проробленої роботи (підбити підсумок). Ваші особисті оцінки (погляди) важливості, перспектив застосування СУБД. Можна вказати на питання, що залишилися неясними й ін.

Список використаних джерел

У список включати тільки ту літературу, що використовувалась при виконанні курсової роботи.

5. ПРИКЛАД ВИКОНАННЯ КУРСОВІ РОБОТИ

Аналіз прикладної задачі та визначення потреб користувачів

Обрати просту прикладну задачу й описати її з погляду того, яку інформацію потрібно зберігати в базі даних. Результатом виконання э короткий опис користувачів, їхніх потреб і списку даних.

Система онлайн-бронювання робочих місць у коворкінгу. Необхідно визначити користувачів, описати їхні потреби, описати дані, які повинні бути в БД.

Користувачі:

- клієнт (резидент коворкінгу) хоче бачити вільні місця на потрібну дату й час; хоче забронювати конкретне місце / переговорну кімнату; хоче переглядати свої майбутні та минулі бронювання;
- адміністратор коворкінгу має бачити всі бронювання за день; повинен мати можливість створювати/редагувати/скасовувати бронювання; хоче уникати ситуацій, коли одне й те саме місце заброньоване двічі;
- менеджер/власник (опційно) хоче бачити статистику завантаженості місць; цікавиться, які зони/кімнати найпопулярніші.

Дані треба зберігати в БД:

- клієнти (ім'я, email, телефон, назва компанії (необов'язково);
- робочі місця / кімнати (назва, тип (робоче місце / переговорна кімната / конференц-зал), місткість (кількість людей), опис / зона (open space, тихий зал тощо);
- бронювання (дата, час початку, час закінчення, посилання на клієнта (хто забронював), посилання на робоче місце / кімнату, статус бронювання (активне, скасоване, завершене);
- тарифи (тип тарифу (погодинний, денний, місячний), ціна, до якого типу місця застосовується).

Формування простих вимог до майбутньої БД

На основі попередньої практики сформувати початкові вимоги.

База даних має зберігати інформацію про клієнтів коворкінгу. Поля: name (обов'язкове), email (обов'язковий), phone (обов'язковий), company (необов'язкове поле).

БД повинна зберігати дані про робочі місця та кімнати. Поля: title, type, capacity, zone.

Система повинна дозволяти створювати бронювання на конкретний час. Поля: date, start_time, end_time, client_id, place_id, status.

Бронювання має бути прив'язане до конкретного клієнта та до конкретного місця. Вимога: наявність зв'язків (у майбутньому – зовнішні ключі client_id, place_id).

Адміністратор повинен мати можливість уникати подвійних бронювань одного й того ж місця. Це вимога, яка пізніше призведе до обмежень типу: перевірки перетину інтервалів часу для одного place_id, або обмеження/унікальності для комбінації (place_id, date, time_start, time_end) у поєднанні з логікою в застосунку.

Створення концептуальної карти предметної області

Потрібна система онлайн-бронювання робочих місць і кімнат у коворкінгу. Клієнти мають бачити вільні місця на потрібну дату й час, робити бронювання та переглядати свою історію. Адміністратор керує бронюваннями (створює, редагує, скасовує) й стежить, щоб не було подвійних бронювань. Менеджер/власник переглядає статистику завантаженості та популярності

зон/кімнат.

Сутності:

- Клієнт
- Місце (робоче місце / кімната / конференц-зал)
- Бронювання
- Тариф

Атрибути сутностей:

- Клієнт (id, ім’я, email, телефон, компанія (необов’язково))
- Місце (id, назва (title), тип (робоче місце / переговорна / конференц-зал), місткість (capacity), зона / опис (open space, тихий зал тощо))
- Бронювання (id, дата, час початку (start_time), час закінчення (end_time), client_id (посилання на Клієнта), place_id (посилання на Місце), статус (активне, скасоване, завершене))
- Тариф (id, тип тарифу (погодинний, денний, місячний), ціна, тип місця, до якого застосовується (наприклад: робоче місце, переговорна, конференц-зал))

Карта зв’язків (у вигляді тексту):

- Клієнт 1:N Бронювання (один клієнт може мати багато бронювань, кожне бронювання належить одному клієнту)
- Місце 1:N Бронювання (одне місце може мати багато бронювань у різний час, кожне бронювання прив’язане до одного місця)
- Тариф 1:N Місце (один тариф застосовується до багатьох місць одного типу, кожне місце має один основний тариф)

За потреби можна ускладнити й зробити Місце N:M Тариф через проміжну сутність PlaceTariff, якщо для одного місця може бути кілька тарифів.

Правила (обмеження предметної області):

- Подвійне бронювання заборонено: для одного і того ж place_id не можуть існувати два активні бронювання, що перетинаються за інтервалом часу (date + start_time–end_time).
- Обов’язковий зв’язок із клієнтом: бронювання не може існувати без вказаного client_id — клієнт повинен бути зареєстрований.
- Обов’язковий зв’язок із місцем: бронювання завжди прив’язане до конкретного місця (place_id обов’язковий).
- Статус бронювання має бути одним із допустимих значень: active, cancelled, finished.
- Тариф повинен відповідати типу місця: тариф із типом «погодинний для переговорної» не можна застосувати до робочого місця іншого типу.

Виділення ключових об’єктів і взаємодій

Основна сутність системи – Бронювання. Усі інші сутності (Клієнт, Місце, Тариф) фактично «обертаються» навколо процесу створення та керування бронюваннями.

Залежні сутності

- Клієнт повинен існувати, щоб створити бронювання; без нього немає «власника» бронювання.
- Місце обов’язковий елемент бронювання; система має знати, що саме було заброньовано.
- Тариф впливає на розрахунок вартості бронювання залежно від типу місця і тривалості.

Сценарії взаємодії:

Сценарій 1: Створення бронювання клієнтом

1. Клієнт заходить у систему й обирає дату та часовий інтервал.
2. Система показує список вільних місць у цей проміжок.
3. Клієнт обирає конкретне місце.
4. Система визначає тариф для цього місця та розраховує вартість (за годину/день/місяць).
5. Створюється запис у сутності «Бронювання» зі статусом active, прив’язаний до client_id і place_id.
6. Адміністратор/система перевіряє, що для цього place_id немає пересічних бронювань.

Сценарій 2: Скасування бронювання адміністратором

1. Адміністратор відкриває список бронювань за обраний день.
2. Знаходить потрібне бронювання за клієнтом, місцем або часом.
3. Змінює статус бронювання на cancelled.
4. Місце в указаний час стає доступним для нових бронювань.

Сценарій 3: Перегляд історії клієнта

1. Клієнт відкриває розділ «Мої бронювання».
2. Система показує майбутні бронювання (status = active) та минулі (status = finished або cancelled).
3. Клієнт може дивитися деталі — дату, час, місце, тривалість і (опційно) вартість.

Сценарій 4: Перегляд статистики менеджером

1. Менеджер обирає період (наприклад, місяць).
2. Система агрегує дані по сутності «Бронювання» та «Місце».
3. Виводиться статистика завантаженості місць/кімнат, найпопулярніші зони, середня кількість бронювань за день тощо.

Обмеження взаємодій

- Одне місце не може мати два активні бронювання, що перетинаються за часом.
- Бронювання завжди належить рівно одному клієнту й одному місцю (зв'язки N:1).
- Клієнт повинен мати валідний email і телефон, щоб можна було зв'язатися у разі змін.
- Тариф, що використовується для розрахунку бронювання, має відповідати типу місця; некоректні комбінації типів не допускаються.
- Зміна статусу на cancelled не повинна видаляти дані про бронювання з історії (воно зберігається для звітів).

Підсумковий опис

Система онлайн-бронювання коворкінгу будується навколо сутності Бронювання, яка пов'язує в собі клієнта та конкретне робоче місце чи кімнату. Кожне бронювання має дату, інтервал часу, статус і посилання на клієнта та місце, що дозволяє відслідковувати історію відвідувань і завантаженість коворкінгу. Сутність Клієнт описує людей, які здійснюють бронювання, із збереженням їх контактних даних, щоб забезпечити комунікацію. Сутність Місце описує робочі зони коворкінгу — тип, місткість, зону розташування — і використовується для пошуку вільних місць та побудови статистики.

Сутність Тариф задає правила розрахунку вартості бронювання залежно від типу місця та формату оплати (година, день, місяць). При створенні бронювання система перевіряє, чи немає конфліктних бронювань на той самий час і те саме місце, забезпечуючи унікальність інтервалу. Адміністратор через систему може змінювати статуси бронювань і тим самим впливати на доступність місць. Менеджер, спираючись на дані про бронювання, аналізує завантаженість і популярність зон, що допомагає приймати управлінські рішення щодо тарифів та розвитку коворкінгу.

Побудова логічної структури для невеликої системи

Створити логічну модель для системи бронювання коворкінгу.

Визначаємо таблиці

Пропонуємо такі таблиці:

- client клієнти коворкінгу,
- place робочі місця / кімнати,
- tariff тарифи для місць,
- booking бронювання.

Атрибути таблиць

Таблиця client; client_id (PK) унікальний ідентифікатор клієнта, name ім'я клієнта, email email клієнта, phone телефон клієнта, company назва компанії (необов'язково).

Таблиця place: place_id (PK) унікальний ідентифікатор місця, title назва місця

/ кімнати, type тип (робоче місце / переговорна / конференц-зал), capacity місткість (кількість людей), zone зона / опис (open space, тихий зал тощо), tariff_id (FK → tariff) базовий тариф, що застосовується до цього місця.
Таблиця tariff: tariff_id (PK) унікальний ідентифікатор тарифу, tariff_type тип тарифу (погодинний, денний, місячний), price ціна, place_type до якого типу місця застосовується (робоче місце / переговорна / конференц-зал).

Таблиця booking: booking_id (PK) унікальний ідентифікатор бронювання, date дата бронювання, start_time час початку, end_time час завершення, status статус (active, cancelled, finished), client_id (FK → client) хто забронював, place_id (FK → place) яке місце заброньовано. Опційно можна додати tariff_id (FK → tariff) у booking, якщо хочемо фіксувати, за яким тарифом було зроблене бронювання.

Опис зв'язків

- client 1:N booking (один клієнт може мати багато бронювань, кожне бронювання належить одному клієнту)
- place 1:N booking (одне місце може мати багато бронювань у різний час, кожне бронювання прив'язане до одного місця)
- tariff 1:N place (один тариф може застосовуватись до багатьох місць, кожне місце має один базовий тариф через tariff_id)

Обмеження

- кожне поле *_id (тобто client_id, place_id, tariff_id, booking_id) обов'язкове у своїй таблиці;
- поля date, start_time, end_time у таблиці booking обов'язкові;
- поля client_id та place_id у таблиці booking обов'язкові, з зовнішніми ключами на client і place;
- телефон (phone) і email (email) у таблиці client мають бути обов'язковими;
- логічне обмеження: одне місце не може бути заброньоване двічі на той самий час. Це означає, що для одного й того ж place_id не повинно існувати двох активних записів booking, у яких інтервали (date, start_time–end_time) перетинаються;
- статус status у booking має бути одним із наперед визначених значень: active, cancelled, finished;
- price у tariff має бути невід'ємною (≥ 0), capacity у place додатною (> 0).

Моделювання історії змін (створено/оновлено) без складних механізмів

Відстеження історії – важлива складова проєктування. На фізичному рівні це можуть бути тригери, але на логічному ми просто додаємо відповідні атрибути. Додати до кожної таблиці атрибути для фіксації часу створення та оновлення.

Додаємо атрибути

У кожну таблицю (client, place, tariff, booking) додаємо:

- created_at дата та час створення запису;
- updated_at дата та час останнього оновлення запису.

Логічна роль полів у контексті коворкінгу

Поле created_at:

- дозволяє аналізувати, коли було додано: нового клієнта (коли він уперше зареєструвався/з'явився в системі); нове місце / кімнату (коли розширили коворкінг); новий тариф (коли ввели нову цінову політику); конкретне бронювання (коли клієнт зробив бронювання);
- на основі created_at можна будувати статистику: коли найчастіше створюються бронювання (години/дні пікового навантаження); як часто з'являються нові клієнти тощо.

Поле updated_at:

- показує, коли востаннє змінювалась інформація: про клієнта (оновлення контактів, компанії тощо); про місце (зміна місткості, типу, зони); про тариф (нова ціна, новий тип тарифу); про бронювання (зміна статусу з active на cancelled чи finished, зміна часу за домовленістю з клієнтом);
- дозволяє відстежувати актуальність записів, наприклад, коли тариф вже давно не змінювався;
- логічно готує модель до аудиту та журнала змін — у майбутньому можна додати тригери та історичні таблиці, але вже зараз є мінімально необхідна інформація про зміни.

Створення таблиць для навчальної задачі

Створимо фізичну структуру таблиць для системи онлайн-бронювання робочих місць і кімнат у коворкінгу: client, place, tariff, booking, додавши в кожну created_at та updated_at.

```
CREATE TABLE client (
    client_id serial PRIMARY KEY,
    name varchar(100) NOT NULL,
    email varchar(150) NOT NULL,
    phone varchar(20) NOT NULL,
    company varchar(150),
    created_at timestamp NOT NULL DEFAULT now(),
    updated_at timestamp NOT NULL DEFAULT now()
);
CREATE TABLE place (
    place_id serial PRIMARY KEY,
    title varchar(150) NOT NULL,
```

```
    type varchar(50) NOT NULL, -- робоче
місце/переговорна/конференц-зал
    capacity integer NOT NULL,
    zone varchar(100),
    tariff_id integer NOT NULL REFERENCES
tariff(tariff_id),
    created_at timestamp NOT NULL DEFAULT now(),
    updated_at timestamp NOT NULL DEFAULT now()
);
```

Примітка: tariff_id задає базовий тариф для цього місця.

```
CREATE TABLE tariff (
    tariff_id serial PRIMARY KEY,
    tariff_type varchar(50) NOT NULL, -- погодинний,
денний, місячний
    price numeric(10,2) NOT NULL,
    place_type varchar(50) NOT NULL, -- до якого типу
місця застосовується
    created_at timestamp NOT NULL DEFAULT now(),
    updated_at timestamp NOT NULL DEFAULT now()
);
CREATE TABLE booking (
    booking_id serial PRIMARY KEY,
    date date NOT NULL,
    start_time time NOT NULL,
    end_time time NOT NULL,
    status varchar(30) NOT NULL DEFAULT 'active',
    client_id integer NOT NULL REFERENCES
client(client_id),
    place_id integer NOT NULL REFERENCES place(place_id),
    tariff_id integer REFERENCES tariff(tariff_id),
    created_at timestamp NOT NULL DEFAULT now(),
```

```
    updated_at timestamp NOT NULL DEFAULT now()
);
```

tariff_id в booking можна використовувати для фіксації тарифу, за яким було зроблено конкретне бронювання (опційно, але логічно випливає з попередніх практичних).

Додавання простих ключів і мінімальних обмежень

На цьому етапі ми:

- уточнюємо унікальність важливих полів;
- додаємо мінімальні CHECK-обмеження;
- реалізуємо базовий захист від подвійних бронювань.

Первинні ключі (serial PRIMARY KEY) та зовнішні ключі (REFERENCES) ми вже задали в CREATE TABLE.
Телефон і email у клієнта обов'язкові, і доцільно робити їх унікальними.

```
ALTER TABLE client

ADD CONSTRAINT client_phone_unique UNIQUE (phone);

ALTER TABLE client

ADD CONSTRAINT client_email_unique UNIQUE (email);
```

Заборона подвійного бронювання одного місця на той самий інтервал. Мінімальний варіант без перевірки перетину інтервалів – унікальність комбінації (place_id, date, start_time, end_time) для активних бронювань.

```
ALTER TABLE booking

ADD CONSTRAINT unique_place_time UNIQUE (place_id, date,
start_time, end_time);
```

Зауваження: справжня перевірка «перетину інтервалів» потребує складнішої логіки (тригерів або перевірок у застосунку). Тут ми реалізуємо спрощене обмеження, достатнє для навчальної задачі.
Обмеження для статусу бронювання. Статус має бути одним із: active, cancelled, finished.

```
ALTER TABLE booking

ADD CONSTRAINT booking_status_check

CHECK (status IN ('active', 'cancelled', 'finished'));
```

Невід'ємна ціна тарифу.

```
ALTER TABLE tariff

ADD CONSTRAINT tariff_price_check

CHECK (price >= 0);
```

Додатна місткість місця.

```
ALTER TABLE place

ADD CONSTRAINT place_capacity_check

CHECK (capacity > 0);
```

Створення одного простого індексу

У нашій навчальній базі коворкінгу ми часто шукаємо клієнта за email, наприклад при вході в особистий кабінет або при створенні бронювання:

```
SELECT * FROM client WHERE email = 'user@example.com';
```

Щоб прискорити такий пошук, створимо індекс на полі email таблиці client:

```
CREATE INDEX idx_client_email ON client(email);
```

Пояснення:

- idx_client_email – ім'я індексу (добра практика: idx_<таблиця>_<поле>).
- client(email) – індекс створюється для стовпця email в таблиці client.
- Тип за замовчуванням – B-tree, що оптимально для умов =, <, >, BETWEEN, IN.

Примітка: ми вже робили UNIQUE (email) для клієнтів – у PostgreSQL це теж створює індекс. Але в навчальній роботі окремо показуємо явне створення індексу, щоб закріпити синтаксис.

Виконаємо:

```
EXPLAIN SELECT * FROM client WHERE email =
'user@example.com';
```

У плані виконання має з'явитися щось на кшталт:

```
Index Scan using idx_client_email on client  ...
```

Це означає, що оптимізатор використав індекс замість повного проходу по таблиці.

Спостереження різниці у виконанні SELECT

Щоб реально побачити різницю, виконайте наступні кроки.

Створіть таблицю для експерименту:

```
CREATE TABLE test_speed (

    id serial PRIMARY KEY,

    value varchar(50)

);
```

Додайте багато тестових даних:

```
INSERT INTO test_speed(value)

SELECT md5(random()::text)
```

```
FROM generate_series(1, 500000);
```

Виконайте пошук БЕЗ індексу:
EXPLAIN ANALYZE
```
SELECT * FROM test_speed WHERE value = 'abc';
```

У плані буде щось подібне до:
```
Seq Scan on test_speed (cost=0.00..... rows=1 width=40)

Execution time: ~XX ms
```

- Seq Scan – означає послідовне сканування всієї таблиці.
- Час (Execution time) залежить від машини, але зазвичай помітно більший.

Створіть індекс:
```
CREATE INDEX idx_test_speed_value ON test_speed(value);
```

Виконайте запит ще раз:
```
EXPLAIN ANALYZE

SELECT * FROM test_speed WHERE value = 'abc';
```

Тепер у плані має бути:
```
Index Scan using idx_test_speed_value on test_speed ...

Execution time: ~0.1 ms
```

Різниця в часі виконання може бути в десятки або сотні разів. Це переноситься на модель коворкінгу: аналогічно індекс на booking(place_id, date) або на client(email)/client(phone) дозволить значно прискорити реальні запити пошуку бронювань і клієнтів.

Початкові приклади використання транзакцій

Кілька операцій мають виконатися разом або не виконатися взагалі. Загальний синтаксис:
```
BEGIN;

-- одна або кілька операцій

UPDATE ... ;

INSERT ... ;

DELETE ... ;

COMMIT; -- підтвердити зміни

-- або

ROLLBACK; -- скасувати
```

Приклад 1. Успішна транзакція: створення клієнта і його бронювання
Припустимо, ми одночасно реєструємо нового клієнта і відразу створюємо

йому бронювання в коворкінгу.

```
BEGIN;

INSERT INTO client(name, email, phone, company)

VALUES ('Ірина', 'iryna@example.com', '0931112233',
'Freelancer');

INSERT INTO booking(date, start_time, end_time, status,
client_id, place_id)

VALUES ('2025-01-20', '12:00', '14:00', 'active', 1, 2);

COMMIT;
```

Тут:

- перший INSERT додає клієнта;
- другий INSERT створює бронювання (для прикладу використовуємо client_id = 1 і place_id = 2, які мають існувати в БД);
- обидві операції фіксуються разом у момент COMMIT.

Приклад 2. Помилка всередині транзакції

Спробуємо порушити обмеження NOT NULL (у нас телефон і email клієнта обов'язкові):

```
BEGIN;

INSERT INTO client(name, email, phone, company)

VALUES ('Олег', NULL, NULL, 'Designer'); -- email NOT
NULL, phone NOT NULL → помилка

INSERT INTO booking(date, start_time, end_time, status,
client_id, place_id)

VALUES ('2025-01-21', '10:00', '12:00', 'active', 2, 1);

COMMIT;
```

Результат:

- перший INSERT викликає помилку (порушення NOT NULL);
- PostgreSQL автоматично скасує всю транзакцію;
- у таблицях не буде жодного нового рядка – це і є атомарність.

Приклад 3. Оновлення пов'язаних даних: зміна статусу бронювання + лог

Припустимо, ми хочемо змінити статус бронювання і записати це в журнал дій. Спочатку (разово) можемо створити просту таблицю логів:

```
CREATE TABLE log (

    log_id serial PRIMARY KEY,

    table_name varchar(50) NOT NULL,
```

```
    operation text NOT NULL,
    created_at timestamp NOT NULL DEFAULT now()
);
```

Тепер транзакція:

```
BEGIN;
UPDATE booking
SET status = 'cancelled'
WHERE booking_id = 10;
INSERT INTO log(table_name, operation, created_at)
VALUES ('booking', 'Booking 10 cancelled by manager',
now());
COMMIT;
```

Якщо:

- UPDATE не виконається (немає такого booking_id або інша помилка),
- або INSERT INTO log впаде,

вся транзакція буде скасована, і статус бронювання теж не зміниться.

Створення простого VIEW

VIEW – це збережений SELECT, який:

- приховує складні JOIN-и;
- стандартизує повторювані запити;
- спрощує доступ до даних у застосунку.

Працюємо з таблицями: booking, client, place, tariff. Створимо VIEW з деталями бронювання. Потрібно створити уявлення, яке «склеює» дані про бронювання з інформацією про клієнта, місце та тариф. Так адміністратору й менеджеру буде простіше робити вибірки.

Створення VIEW:

```
CREATE VIEW v_booking_details AS
SELECT
    b.booking_id,
    b.date,
    b.start_time,
    b.end_time,
    b.status,
```

```
    c.name   AS client_name,
    c.email  AS client_email,
    c.phone  AS client_phone,
    p.title  AS place_title,
    p.type   AS place_type,
    p.zone   AS place_zone,
    t.tariff_type,
    t.price
FROM booking b
JOIN client c ON b.client_id = c.client_id
JOIN place  p ON b.place_id  = p.place_id
LEFT JOIN tariff t ON b.tariff_id = t.tariff_id;
```

LEFT JOIN tariff – на випадок, якщо в деяких бронюваннях tariff_id може бути NULL (опційне поле).

Список активних бронювань на певний день:

```
SELECT * FROM v_booking_details
WHERE date = '2025-03-10' AND status = 'active';
```

Історія бронювань конкретного клієнта:

```
SELECT date, start_time, end_time, place_title, status
FROM v_booking_details
WHERE client_email = 'user@example.com'
ORDER BY date DESC, start_time DESC;
```

Перегляд завантаженості певної зони (наприклад, тихий зал):

```
SELECT date, start_time, end_time, client_name,
place_title
FROM v_booking_details
WHERE place_zone = 'тихий зал' AND status = 'active';
```

Вибірка бронювань із зазначенням тарифів (для звітності):

```
SELECT date, place_title, tariff_type, price, client_name
FROM v_booking_details
WHERE date BETWEEN '2025-03-01' AND '2025-03-31';
```

Висновок

У курсовій роботі було розглянуто задачу проєктування бази даних для системи онлайн-бронювання робочих місць і кімнат у коворкінгу. Визначено основних користувачів системи та їхні потреби, що дало змогу сформувати перелік необхідних даних і ключових бізнес-процесів, зосереджених навколо бронювання.

Було розроблено концептуальну та логічну моделі бази даних із основними сутностями: клієнт, місце, бронювання та тариф. Визначено атрибути, зв'язки, первинні й зовнішні ключі, а також основні обмеження, зокрема заборону подвійних бронювань і контроль коректності даних. Додано поля для фіксації часу створення та оновлення записів.

У результаті створено фізичну структуру БД з індексами, транзакціями та уявленням для зручної роботи з даними. Запропонована база даних є логічно узгодженою, забезпечує цілісність інформації та може бути використана як основа для розробки повноцінної системи бронювання коворкінгу.

Література

1. Beaulieu, A. (2009). SQL: The complete reference (3rd ed.). McGraw-Hill.
2. Valdés, A. (2018). PostgreSQL: Basics. Packt Publishing.
3. Harrington, J. L. (2022). Relational database design and implementation (4th ed.). Morgan Kaufmann.
4. PostgreSQL Global Development Group. (2025). PostgreSQL documentation. https://www.postgresql.org/docs/
5. Elmasri, R., & Navathe, S. B. (2020). Fundamentals of database systems (8th ed.). Pearson.

6. ІНФОРМАЦІЙНІ РЕСУРСИ

1. Harrington, J. L. (2022). Relational database design and implementation (4th ed.). Morgan Kaufmann.
2. PostgreSQL Global Development Group. (2025). PostgreSQL documentation. https://www.postgresql.org/docs/
3. Elmasri, R., & Navathe, S. B. (2020). Fundamentals of database systems (8th ed.). Pearson.
4. McCreary, T. (2021). The practical guide to data modeling: Data modeling techniques for database design (2nd ed.). Apress
5. Schönig, H.-J. (2023). Mastering PostgreSQL 16 (3rd ed.). Packt Publishing.
6. Ferrari, L., & Pirozzi, E. (2023). Learn PostgreSQL (2nd ed.). Packt Publishing. ISBN: 9781837635641.
7. Packt Publishing. (n.d.). PostgreSQL 11 quick start guide. GitHub. Retrieved July 25, 2024, from https://github.com/PacktPublishing/PostgreSQL-11-Quick-Start-Guide/blob/master/README.md