МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

**ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА**

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання практичних та самостійних робіт
із навчальної дисципліни

«ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ»

(для здобувачів 2 курсу першого (бакалаврського) рівня вищої освіти денної форми навчання зі спеціальностей 126, F6 – Інформаційні системи та технології)

**Харків
ХНУМГ ім. О. М. Бекетова
2025**

Методичні рекомендації до виконання практичних та самостійних робіт із навчальної дисципліни «Організація баз даних та знань» (для здобувачів 2 курсу першого (бакалаврського) рівня вищої освіти денної форми навчання зі спеціальностей 126, F6 – Інформаційні системи та технології) / Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова ; уклад. : О. Б. Костенко, І. О. Гавриленко. – Харків : ХНУМГ ім. О. М. Бекетова, 2025. – 19 с.

Укладачі: канд. фіз.-мат. наук. О. Б. Костенко,
асист. І. О. Гавриленко

Рецензент
М. Ю. Карпенко, кандидат технічних наук, доцент Харківського національного університету міського господарства імені О. М. Бекетова

Рекомендовано кафедрою комп'ютерних наук та інформаційних технологій, протокол № 17 від 02.05.2025

1 МЕТА ПРАКТИЧНИХ ТА САМОСТІЙНИХ РОБІТ

Сформувати у студентів навички практичного застосування існуючих систем управління базами даних; вживання ефективних моделей забезпечення даних на основі вивчення предметної області, методів аналізу, пошуку та використання існуючих систем управління базами даних; ознайомити з існуючими системами управління базами даних реляційного типу; забезпечити теоретичну та інженерну підготовку фахівців у галузі проєктування та використання систем управління базами даних.

Згідно з вимогами освітньо-професійної програми під час вивчення навчальної дисципліни студенти повинні оволодіти:

– основними поняттями баз даних;

– поняттями архітектури та концепції систем управління базами даних;

– методами побудови структур і схем зберігання даних;

– способами проєктування баз даних на концептуальному і логічному рівнях, а також під час їх фізичної реалізації.

У процесі виконання практичних та самостійних завдань студенти повинні навчитися визначати предметну область бази даних, визначати об’єкти, що підлягають представленню в базі даних, будувати формалізований опис об’єктів, визначати первинні та зовнішні ключі, а також розробляти SQL-запити для отримання інформації з бази, а також змінювати структуру таблиць бази даних (включення нових полів, вилучення полів таблиць, змінювання опису полів та обмежень). Щоб змінити структуру таблиць, використовуються директиви мови SQL, які студенти повинні засвоїти в процесі виконання завдань.

2 ВИМОГИ ДО ЗВІТУ З ПРАКТИЧНОЇ ТА САМОСТІЙНОЇ РОБОТИ

Звіт щодо роботи повинен бути представлений в електронній та друкованій версіях.

Електронна версія (як варіант на зовнішньому носії) містить:

1. Файл звіту у форматі документа Word.
2. Базу даних (файл БД або основні таблиці) за вашим варіантом.

Друкована версія містить:

1. Друковану копію файлу звіту у форматі документа Word.
2. Файл звіту «Звіт_прізвище», наприклад: «Звіт_Костенко».

Зміст файлу звіту:

1. Титульний аркуш (зразок оформлення титульного аркуша наведено в дод. А).
2. Завдання на роботу (див. нижче).
3. Конспект ресурсу (п. 2 завдання).
4. Побудова моделі (п. 3 завдання).
5. Роздруківка таблиць бази даних із внесеними даними.
6. Тексти SQL-запитів до бази даних і результати цих запитів у табличній формі.
7. Висновок про можливості інформатизації обраної предметної області.

Загальний обсяг файлу звіту – 7–10 сторінок.

Порядок здачі роботи:

1. Пред’явлення файлів, що містять виконані завдання.
2. Співбесіда на підставі друкованої копії звіту і пред’явлених файлів.

Примітка 1. Можлива поетапна здача звіту.

Примітка 2. Звіт з практичної та самостійної роботи є допуском до заліку.

3 ЗАВДАННЯ ДО ПРАКТИЧНИХ ТА САМОСТІЙНИХ РОБІТ

1. Узгодити з викладачем предметну область для створення проєкту інформаційної системи.

2. Опрацювати лекційний матеріал із проєктування баз даних.

2.1. Законспектувати та усвідомити визначення понять «сутність», «асоціація», «ключ» та ін.

2.2. Законспектувати та усвідомити поняття «відношення», «операції з відношеннями», «атрибут», «кортеж», «домен», «ступінь відношення».

2.3. Законспектувати матеріал стосовно таких аспектів:

– «нормалізація» та її цілі;

– «універсальне відношення»;

– необхідність розбиття «універсального відношення»;

– «нормальні форми»;

– процедура «нормалізації»;

– мова інфологічного моделювання «Таблиці – зв'язки».

3. Побудувати інфологічну модель предметної області за вашим варіантом. Для цього:

– виділити «стержневі сутності» та їх «атрибути» (описово);

– побудувати зв'язки («асоціації», «характеристики») і їх «атрибути» (описово);

– визначити «первинні» та «зовнішні ключі» (описово);

– записати модель на мові «ER-діаграми».

Конспекти матеріалу (п. 2) і побудову інфологічної моделі (п. 3) виконати у файлі Word і розмістити у звіті.

4. Створити таблиці бази даних, які відповідають інфологічній моделі, і внести в них по сім записів. На схемі даних показати зв'язок між об'єктами моделі.

5. Створити запити до бази даних у формі SQL, які б розкривали суть інформаційної системи і відповідали на більшість питань користувача вашої інформаційної системи. Тексти запитів і результат їх роботи розмістити у звіті.

6. Підготувати і роздрукувати файл звіту.

Примітка 3. Конспект повинен містити приклади зі створеної інформаційної системи.

Примітка 4. Ім'я бази даних – це ім'я інформаційної системи «Ваше_прізвище», наприклад: «Ресторан_Костенко».

4 МЕТОДИКА ВИКОНАННЯ РОБІТ

4.1 Варіанти предметних областей для створення проєкту інформаційної системи

Предметною областю називається частина реального світу, що становить інтерес для дослідження. В автоматизованих інформаційних системах відображення предметної області представлено моделями даних декількох рівнів. Кількість рівнів моделей буде залежати від особливостей системи управління базами даних (СУБД).

Можливі варіанти предметних областей для створення проєкту інформаційної системи:

1. Продуктовий кіоск.
2. Хіти сезону та виконавці.
3. Виконавці популярної музики і альбоми.
4. Розклад літаків.
5. Розклад поїздів.
6. Розклад міжміських автобусів.
7. Відділ кадрів.
8. Готель.
9. Гуртожиток.
10. Комп’ютерний магазин.
11. Склад.
12. Бібліотека.
13. Кулінарний довідник.
14. Канцелярія.
15. Лікарня.
16. Майстерня з ремонту взуття.
17. Станція технічного обслуговування автомобілів.
18. Студенти і захоплення.
19. Студенти і спортивні секції.
20. Викладачі та предмети.
21. Кафедри і викладачі.
22. Студенти і улюблені страви.
23. Студенти і оператори мобільного зв’язку.
24. Замовлення таксі.

Здобувач може (і це рекомендовано) запропонувати свій варіант предметної області за аналогією із вказаними вище.

4.2 Рекомендації до інфологічного моделювання предметної області

Найменша одиниця даних реляційної моделі – це окреме атомарне (що не розкладається) для цієї моделі значення даних. Наприклад, в одній предметній області прізвище, ім'я та по батькові можуть розглядатися як єдине значення, а в іншій – як три різні значення.

Доменом називається безліч атомарних значень одного і того самого типу. Якщо значення двох атрибутів беруться з одного і того самого домена, то, ймовірно, мають сенс порівняння, які використовують ці два атрибути. Якщо ж значення двох атрибутів беруться з різних доменів, то їх порівняння, ймовірно, позбавлене сенсу.

Відношення на доменах D_1, D_2, ..., D_n складається із заголовка і тіла.

Заголовок складається з такої фіксованої множини атрибутів A_1, A_2, ..., A_n, що існує взаємно однозначна відповідність між цими атрибутами A_i і доменами D_i (i = 1, 2, ... , n), що їх визначають.

Тіло складається із змінюваної в часі множини *кортежів*, де кожен кортеж складається, своєю чергою, з безлічі пар атрибут-значень (A_i : V_i), (i = 1, 2, ..., n), по одній такій парі для кожного атрибута A_i в заголовку. Для будь-якої заданої пари атрибут-значень (A_i : V_i) V_i є значенням із єдиного домену D_i, який пов'язаний з атрибутом A_i.

Ступінь відношення – це число його атрибутів. Відношення ступеня один називають унарним, ступеня два – бінарним, ступеня три – тринарним, ..., ступеня n – n-арним. *Кардинальне число,* або *потужність відношення,* – це число його кортежів. Кардинальне число відношення змінюється в часі, на відміну від його ступеня.

Оскільки відношення – це множина, а множина за визначенням не містить співпадаючих елементів, то ніякі два кортежі відношення не можуть бути дублікатами один одного в будь-який довільно заданий момент часу.

Нехай R – відношення з атрибутами A_1, A_2, ..., A_n. Кажуть, що множина атрибутів K = (A_i, A_j, ..., A_k) відношення R є *можливим ключем* R тоді і тільки тоді, коли задовольняються дві незалежні від часу умови:

1. Унікальність: у довільний заданий момент часу ніякі два різні кортежі R не мають одного і того самого значення для A_i, A_j, ..., A_k.

2. Мінімальність: жоден з атрибутів A_i, A_j, ..., A_k не може бути виключений з K без порушення унікальності.

Кожне відношення володіє хоча б одним можливим ключем, оскільки комбінація всіх його атрибутів задовольняє умову унікальності. Один із можливих ключів, який обраний довільним способом, приймається за його *первинний ключ*. Інші можливі ключі, якщо вони є, мають назву «*альтернативні ключі»*.

Зазначені вище та деякі інші математичні поняття стали теоретичною базою для створення реляційних СУБД, розробки відповідних мовних засобів і програмних систем, що забезпечують їх високу продуктивність і створення засад теорії проєктування баз даних (БД). Однак для масового користувача реляційних СУБД можна з успіхом використовувати неформальні еквіваленти таких понять:

Відношення – Таблиця (іноді Файл),

Кортеж – Рядок (Запис),

Атрибут – Стовпець (Поле).

При цьому приймається, що «запис» означає «екземпляр запису», а «поле» означає «ім’я і тип поля».

Реляційна база даних – це сукупність відношень, що містять всю інформацію, яка повинна зберігатися в базі даних. Однак користувачі можуть сприймати таку базу даних, як сукупність таблиць. Існують такі правила:

1. Кожна таблиця складається з однотипних рядків (записів) і має унікальне ім’я.

2. Рядки (записи) мають фіксоване число полів (стовпців) і значень (множинні поля і повторювані групи неприпустимі). Інакше кажучи, в кожній позиції таблиці на перетині рядка і стовпця завжди є одне значення або нічого.

3. Рядки таблиці обов’язково відрізняються один від одного хоча б одним значенням, що дозволяє однозначно ідентифікувати будь-який рядок такої таблиці.

4. Стовпцям таблиці обов’язково присвоюються імена, і в кожному з них розміщуються однорідні значення даних (дати, прізвища, цілі числа або грошові суми).

5. Повний інформаційний зміст бази даних представляється у вигляді явних значень даних, і такий метод подання є єдиним. Зокрема, не існує будь-яких спеціальних «зв’язків» або покажчиків, що з’єднують одну таблицю з іншою.

6. Під час виконання операцій із таблицею її рядки і стовпці можна обробляти в будь-якому порядку без урахування їх інформаційного змісту. Цьому сприяє наявність імен таблиць та їх стовпців, а також можливість виділення будь-якого їх рядка або будь-якого набору рядків із зазначеними ознаками.

4.3 Рекомендації до проведення процедури «Нормалізація»

Нормалізація – це розбиття таблиці на дві або більше, що володіють кращими властивостями при включенні, змінюванні і видаленні даних. Остаточна мета нормалізації зводиться до отримання такого проєкту бази даних, в якому кожен факт з'являється лише в одному місці, тобто виключена надмірність інформації. Це робиться не стільки з метою економії пам'яті, скільки для унеможливлення можливої суперечливості збережених даних.

Кожна таблиця в реляційній БД задовольняє умову, відповідно до якої в позиції на перетині кожного рядка і стовпчика таблиці завжди міститься єдине значення і ніколи не може бути безлічі таких значень. Будь-яка таблиця, яка задовольняє цю умову, називається *нормалізованою*. Фактично, ненормалізовані таблиці, тобто таблиці, що містять повторювані групи, навіть не допускаються в реляційній БД.

Будь-яка нормалізована таблиця автоматично вважається таблицею в першій нормальній формі (1НФ). Однак на практиці термін «нормалізована» використовується в більш вузькому сенсі – «повністю нормалізована», який означає, що в проєкті не порушуються ніякі принципи нормалізації.

Таблиця міститься в *першій нормальній формі (1НФ)* тоді і тільки тоді, коли жоден з її рядків не містить у будь-якому своєму полі більше одного значення і жодне з її ключових полів не порожнє.

Таблиця міститься в *другій нормальній формі (2НФ)*, якщо вона задовольняє визначення 1НФ і всі її поля, що не входять у первинний ключ, пов'язані повною функціональною залежністю з первинним ключем.

Для спрощення нормалізації таблиць доцільно використовувати таку рекомендацію: при проведенні нормалізації таблиць, у які введені цифрові або інші замінники складових і (або) текстових первинних і зовнішніх ключів, потрібно хоча б подумки підміняти їх на вихідні ключі, а після закінчення нормалізації знову відновлювати.

Таблиця міститься в *третій нормальній формі (3НФ),* якщо вона задовольняє визначення 2НФ і ні одне з її неключових полів не залежить функціонально від будь-якого іншого неключового поля.

Таблиця міститься в *нормальній формі Бойса – Кодда (НФБК),* тоді і тільки тоді, коли будь-яка функціональна залежність між її полями зводиться до повної функціональної залежності від можливого ключа.

У наступних нормальних формах (4НФ і 5НФ) враховуються не тільки функціональні, але й багатозначні залежності між полями таблиці. Для їх опису познайомимося з поняттям повної декомпозиції таблиці.

Повною декомпозицією таблиці називають таку сукупність довільного числа її проєкцій, з'єднання яких повністю збігається із вмістом таблиці.

Таблиця міститься в п'ятій нормальній формі (5НФ) тоді і тільки тоді, коли в кожній її повній декомпозиції всі проєкції містять можливий ключ. Таблиця, яка не має жодної повної декомпозиції, також міститься в 5НФ.

Четверта нормальна форма (4НФ) є окремим випадком 5НФ, коли повна декомпозиція є з'єднанням двох проєкцій. Дуже не просто підібрати реальну таблицю, що може бути надана в 4НФ, але не була б у 5НФ.

Теорія нормалізації ґрунтується на наявності тієї чи іншої залежності між полями таблиці. Визначено два види таких залежностей – функціональні і багатозначні.

Функціональна залежність. Поле B таблиці функціонально залежить від поля A тієї самої таблиці в тому і тільки в тому випадку, коли в будь-який заданий момент часу для кожного з різних значень поля A обов'язково існує тільки одне з різних значень поля B. Відзначимо, що допускається, що поля A та B можуть бути складовими.

Повна функціональна залежність. Поле B міститься в повній функціональної залежності від складеного поля A, якщо воно функціонально залежить від A і не залежить функціонально від будь-якої підмножини поля A.

Багатозначна залежність. Поле A багатозначно визначає поле B тій самій таблиці, якщо для кожного значення поля A існує певна безліч відповідних значень B.

Інше визначення «нормалізації» – це процес послідовної заміни таблиці її повними декомпозиціями до тих пір, коли всі вони не будуть міститися в 5НФ. На практиці ж достатньо привести таблиці до НФБК і з великою гарантією вважати, що вони перебувають у 5НФ.

Процедура приведення таблиць до НФБК ґрунтується на тому, що єдиними функціональними залежностями в будь-якій таблиці повинні бути залежності виду K → F, де K – первинний ключ, а F – деяке інше поле. Це випливає з визначення первинного ключа таблиці, відповідно до якого K → F завжди існує для всіх полів цієї таблиці. Мета нормалізації полягає саме в тому, щоб позбутися всіх цих «інших» функціональних залежностей, тобто таких, які мають інший вигляд, ніж K → F.

Якщо скористатися рекомендацією, наведеною вище, і підмінити під час нормалізації коди *первинних (зовнішніх)* ключів на *вихідні ключі*, то, по суті, потрібно розглянути лише два випадки:

1. Таблиця має складовою первинний ключ виду ($К_1$, $К_2$) і включає також поле F, яке функціонально залежить від частини цього ключа, наприклад, від $К_2$, тільки не від повного ключа. У цьому випадку рекомендується сформувати

іншу таблицю, яка містить $К_2$ і F (первинний ключ – $К_2$), і видалити F з початкової таблиці: замінити T ($К_1$, $К_2$, F), первинний ключ ($К_1$, $К_2$), $Ф_3$ $К_2 \rightarrow F$ на $Т_1$ ($К_1$, $К_2$), первинний ключ ($К_1$, $К_2$), і $Т_2$ ($К_2$, F), первинний ключ $К_2$.

2. Таблиця має первинний (можливий) ключ К, поле F_1, яке не є можливим ключем, що зазвичай функціонально залежить від К, і інше неключове поле F_2, яке функціонально залежить від F_1. Рішення те саме, що й раніше: формується інша таблиця, яка містить F_1 і F_2, з первинним ключем F_1, і F_2 видаляється з початкової таблиці: замінити T (K, F_1, F_2), первинний ключ К, $Ф_3$ $F_1 \rightarrow F_2$ на $Т_1$ (K, F_1), первинний ключ К, і $Т_2$ (F_1, F2), первинний ключ F_1.

Для будь-якої заданої таблиці шляхом повторення застосування двох розглянутих правил майже у всіх практичних ситуаціях можна отримати в підсумку безліч таблиць, які містяться в «остаточній» нормальній формі і, таким чином, не містять будь-яких функціональних залежностей виду, відмінного від $K \rightarrow F$.

Для виконання цих операцій необхідно спочатку мати, як вхідні дані, будь-які «великі» таблиці (наприклад, універсальні відносини).

4.4 SQL-запити

Запит становить якусь команду, яка звертається до БД і повідомляє їй, щоб вона відобразила певну інформацію з таблиць у пам’ять. Ця інформація зазвичай виводиться безпосередньо на екран комп’ютера, термінал, надсилається на принтер, зберігається у файлі або слугує вихідними даними для іншої команди або запиту.

Усі SQL-запити складаються з одиничної команди SELECT із простою структурою, проте шляхом її використання можна виконати складну обробку даних. У найпростішій формі команда SELECT звертається до БД, щоб отримати інформацію з таблиці. Наприклад, щоб вивести таблицю студентів, потрібно надати такий запит:

```
SELECT SNUM, SFAM, SIMA, SOTCH, STIP FROM STUDENTS;
```

У цій команді SELECT – ключове слово, яке повідомляє БД, що ця команда є запитом, тобто всі запити починаються цим словом.

SNUM, SFAM, SIMA, SOTCH, STIP – список полів із таблиці, які вибираються запитом. Поля, не перелічені тут, не будуть включені у висновок команди. Запит не вплине на інформацію в таблицях; він тільки показує дані. FROM STUDENTS – ключове слово, подібне до SELECT, яке повинне бути представлене в кожному запиті. Воно супроводжується пропуском і потім ім’ям таблиці, використовуваної як джерело інформації. В цьому випадку – це таблиця студентів STUDENTS.

Крапка з комою («;») використовується у всіх інтерактивних командах SQL для повідомлення БД, що команда заповнена і готова виконуватися, а в деяких системах похила риска в рядку є індикатором кінця команди. Очевидно, запит такого характеру не обов'язково буде впорядковувати висновок будь-яким зазначеним способом. Та сама команда, виконана з тими самими даними, але в різний час, не зможе вивести результат в однаковому порядку. Зазвичай рядки виявляються в тому порядку, в якому вони знайдені в таблиці, а оскільки він довільний, то зовсім не обов'язково буде зберігатися той порядок, в якому дані вводилися або зберігалися.

Якщо необхідно отримати кожне поле таблиці, не обов'язковим є скорочення у вигляді символу «зірочка» (*), яке можна використовувати для виведення повного списку полів так: SELECT * FROM STUDENTS, що призведе до того самого результату, що й попередня команда. У загальному випадку запит починається з ключового слова SELECT, супроводжуваного пробілом. Далі буде розміщуватися список розділених комами імен полів, які необхідно вивести.

Наступне ключове слово FROM супроводжується пропуском і ім'ям таблиці, до якої робиться запит. Крапка з комою повинна використовуватися для того, щоб закінчити запит і вказати, що команда готова до виконання. Команда SELECT здатна витягти певну інформацію з таблиці. Наприклад, в разі необхідності виведення тільки певних полів таблиці зі списку виключаються непотрібні поля.

Цей спосіб дозволяє працювати з таблицями, які мають багато полів, що містять дані, які не потрібні в цей момент користувачеві.

Під час роботи з даними дуже часто виникає потреба видалення надлишкових даних. У такому випадку використовують DISTINCT – аргумент, який забезпечує можливість усувати повторювані значення з пропозиції SELECT. Припустимо, що необхідно дізнатися, які студенти в певний час здавали навчальні предмети, до того ж не потрібно уточнювати отриману оцінку і предмет. Запит SELECT SNUM FROM USP; надає такий висновок, проте в ньому є записи-дублікати:

SNUM
3412
3413
3414
3412
3416

Для отримання списку результатів без дублікатів в цьому випадку доцільно скористатися таким: SELECT DISTINCT SNUM FROM USP; в результаті чого буде отримано:

SNUM
3412
3413
3414
3416

Потрібно пам’ятати, що DISTINCT може вказуватися тільки один раз в цій пропозиції SELECT. Якщо пропозиція вибирає численні поля, DISTINCT опускає записи, де всі вибрані поля ідентичні. Якщо замість DISTINCT вказати ALL, то це буде мати протилежний ефект і дублювання рядків виводу збережеться.

Для використання умов пошуку для відбору рядків потрібно використовувати такі команди: WHERE – пропозиція команди SELECT, яка дозволяє встановлювати предикати, умова яких може бути або правильною або неправильною для будь-якого запису таблиці. Команда витягує тільки ті записи з таблиці, для якої таке твердження істинне. Припустимо, що необхідно вибрати прізвища і розміри стипендії студентів, при цьому цікавлять тільки такі, які отримують стипендію в розмірі 25.50. Такий запит буде виглядати так:

SELECT SFAM, STIP
FROM STUDENTS WHERE
STIP=25.50;

Зазвичай у предикатах потрібно не тільки оцінювати рівність оператора як істинного або хибного, а й здійснювати інші види зв’язків. Це реалізується за допомогою булевих операторів і знаків відносини, причому предикат може містити необмежену кількість умов. В цілому, реляційний оператор – це математичний символ, який вказує на певний тип порівняння між двома значеннями, при цьому SQL має такий їх набір:

= рівний чогось;
> більш ніж;
< менш ніж;
>= більш ніж або дорівнює;
<= менш ніж або дорівнює;
<> не дорівнює.

Ці оператори мають стандартні значення для числових даних, а для символьних їх визначення залежить від кодів ASCII символів – вони розміщуються в алфавітному порядку, до того ж великі літери мають менший код, ніж малі, тому, наприклад, «Z» < «a». Припустимо, що необхідно вивести

список студентів, які отримують стипендію, тобто для яких STIP > 0. Для цього скористаємося таким запитом:

```
SELECT *
FROM STUDENTS
WHERE STIP > 0;
```

Стандартними булевими операторами, які використовуються в SQL, є «AND», «OR» і «NOT». Вони працюють так:

– «AND» використовує два операнди у формі «A AND B» і оцінює їх по відношенню до істини: чи вони обидва правильні;

– «OR» використовує два операнди у формі «A OR B» і оцінює істинність: чи правильний один із них;

– «NOT» використовує один операнд у формі «NOT A» і замінює його значення з «ІСТИНА» на «НЕІСТИНА», або навпаки.

Умова «NOT» може використовуватися для інвертування логічних значень. Наприклад, для виведення інформації про студентів, у яких оцінки не 3, можна скористатися таким запитом:

```
SELECT * FROM USP WHERE NOT (OCENKA = 3);
```

Потрібно пам’ятати, що булевий оператор розміщується перед реляційним оператором, на який він діє, а за необхідності розширення дії використовуються дужки. Наприклад, для виведення інформації про студентів, у яких оцінки не 3 водночас і з навчального предмета з кодом, що не дорівнює 2005, можна скористатися таким запитом:

```
SELECT * FROM USP WHERE NOT {OCENKA = 3 AND PNUM = 2005);
```

У цьому випадку SQL розуміє круглі дужки як підтвердження, що все всередині них буде оцінюватися першим і оброблятися як єдине вираження за допомогою того оператора, який розміщується зовні. У реченні SELECT на додаток до традиційних реляційних і булевих операторів, розглянутих вище, можуть бути використані спеціальні оператори «IN», «BETWEEN», «LIKE» і «IS NULL». Оператор «IN» визначає набір значень, в який це значення має бути включене. Наприклад, якщо, відповідно до навчальної БД, виникає необхідність подання інформації про всіх студентів, ім’я яких є «Анатолій» або «Володимир», потрібно виконати такий запит:

```
SELECT *
FROM STUDENTS
WHERE SIMA = ‘Анатолій’ OR
SIMA = ‘Володимир’;
```

Однак є і більш простий спосіб отримати ту саму інформацію – за допомогою такого запиту:

```
SELECT *
```

FROM STUDENTS

WHERE SIMA IN (‘Анатолій’, ‘Володимир’);

Звідси зрозуміло, що «IN» визначає набір значень за допомогою списку, укладеного в круглі дужки з роздільниками у вигляді ком. Він перевіряє різні значення вказаного поля, намагаючись знайти збіг із значеннями з набору. Якщо це стається, то предикат правильний. Якщо набір містить числові, а не символьні значення, то потрібно використовувати одиничні лапки, що опускаються. Прикладом такого запиту може слугувати пошук всіх студентів, які мають стипендію 17.00 і 25.50:

SELECT *

FROM STUDENTS

WHERE STIP IN (17.00, 25.50);

Оператор «BETWEEN» дещо схожий на «IN», але, на відміну від визначення з набору, «BETWEEN» визначає діапазон значень, у який повинні вміщатися шукані значення, що й робить предикат правильним. Структура оператора «BETWEEN» така: вводиться початкове значення, ключове слово «AND» і кінцеве значення. На відміну від «IN», «BETWEEN» розрізняє порядок значень, отже, перше з них в пропозиції повинне бути першим за алфавітним або числовим порядком. Наступний приклад – відібрати з таблиці успішності номери і оцінки всіх студентів, оцінки яких укладені між 3 і 5:

SELECT SNUM, OCENKA FROM USP WHERE

OCENKA BETWEEN 3 AND 5;

Оператор «LIKE» застосовується тільки до полів типу «CHAR» або «VARCHAR», у яких він шукає підрядки, тобто він шукає символи і перевіряє, чи збігаються вони з умовою. Як умову, оператор використовує групові символи – спеціальні символи, які відповідають будь-чому. Існує два типи групових символів, використовуваних із «LIKE»:

– символ підкреслення заміщує будь-який одиничний символ, наприклад: «М_Л» буде відповідати словами «МОЛ» або «МЕЛ», але не буде відповідати «МЕТАЛ»;

– символ відсотка заміщує послідовність будь-якого числа символів, в тому числі нульової довжини. Наприклад, «% М% Л» буде відповідати словами «МЕЛ» або «ПОМОЛ», але не відповідає «МОЛОКО».

Як приклад, знайдемо всіх викладачів, прізвища яких починаються з літери «К»:

SELECT TFAM, TIMA, TOTCH

FROM TEACHERS WHERE

TFAM LIKE ‘K%’;

Для отримання підсумкових даних використовуються агрегатні функції. За їхньою допомогою запити можуть виконувати узагальнену групову обробку значень полів. У SQL допускаються такі агрегатні функції.

– «COUNT» – проводить підрахунок кількості рядків або «НЕ-NULL значень полів, які вибрав запит;

– «SUM» – розраховує арифметичну суму всіх обраних значень заданого поля;

– «AVG» – здійснює усереднення всіх обраних значень заданого поля;

– «MAX» – знаходить і повертає найбільше з усіх обраних значень заданого поля;

– «MIN» – знаходить і повертає найменше з усіх обраних значень заданого поля.

Варто пам'ятати, що з «SUM» і «AVG» використовуються тільки числові поля, а з «COUNT», «MAX» і «MIN» можуть використовуватися числові або символьні поля. Коли функції записуються з символьними полями, «MAX» і «MIN» використовують їх як еквівалент ASCII, відповідно до якого вибирається максимальне або мінімальне значення. Щоб знайти суму всієї виплаченої стипендії в таблиці з даними про студентів, потрібно виконати такий запит:

```
SELECT SUM (STIP)
FROM STUDENTS;
```

результат якого складається з одного значення 68.00.

Команда «GROUP BY» може застосовуватися з агрегатними функціями незалежно від серій груп, які визначаються за допомогою значення поля в цілому. У цьому випадку кожна група складається з усіх рядків з тим самим значенням поля «SNUM», а «MIN» функція застосовується окремо для кожної такої групи. Значення поля, до якого застосовується «GROUP BY», має тільки одне значення на групу виводу, так само як це робить агрегатна функція, тому в результаті і з'являється сумісність, яка дозволяє агрегатним функціям і полям об'єднатися. Також допускається використання «GROUP BY» одночасно з декількома полями. Для прикладу створимо вибір найменшого значення оцінки за кожен день. При цьому запит буде таким:

```
SELECT SNUM, UDATE, MIN (OCENKA)
FROM USP
GROUP BY SNUM, UDATE;
```

При багатотабличному запиті таблиці, представлені у вигляді списку у «FROM», відокремлюються одна від одної комами. Предикат запиту може посилатися до будь-якому стовпця будь-якої пов'язаної таблиці і, отже, може використовуватися для зв'язку між ними. Зазвичай предикат порівнює значення

в стовпцях різних таблиць, щоб визначити, чи задовольняє «WHERE» встановлену умову.

Припустимо необхідно поставити у відповідність викладачу навчальні предмети, які він веде. Фактично, SQL доведеться вибирати з таблиці викладачів відповідний йому код і, переглядаючи таблицю предметів, здійснювати пошук відповідного коду. Це можна реалізувати таким запитом:

```
SELECT TEACHERS.TFAM, PREDMET.PNAME
FROM TEACHERS, PREDMET WHERE
TEACHERS.TNUM = PREDMET.TNUM;
```

Оскільки поле TNUM є і в таблиці викладачів, і в таблиці предметів, то імена таблиць повинні використовуватися як префікси, хоча це необхідно тільки в тому випадку, коли два або більше полів мають одне і те саме ім’я. Однак в будь-якому випадку включати ім’я таблиці в запити з об’єднанням зручно для кращого розуміння і несуперечності отриманих результатів. При виконанні багатотабличного запиту SQL досліджує кожну комбінацію рядків двох або більше можливих таблиць і перевіряє ці комбінації за їхніми предикатам. Якщо комбінація створює таке значення, яке робить предикат правильним, то значення буде вибране для виводу.

Об’єднання в багатотабличних запитах, які використовують предикати, базуються на рівності й називаються об’єднаннями за рівністю. Об’єднання за рівністю – це найбільш загальний вигляд об’єднання, однак існують і інші види об’єднань. Фактично, можна використовувати будь-який із реляційних операторів. Прикладом іншого виду об’єднання може слугувати такий запит:

```
SELECT TEACHERS. TFAM, PREDMET . PNAME
FROM TEACHERS, PREDMET WHERE
TEACHERS. TFAM < PREDMET. PNAME
AND TEACHERS.TFAM BETWEEN ‘K’ AND ‘C’ ;
```

За допомогою цього запиту виводиться інформація про викладачів, прізвище яких за алфавітом знаходиться раніше, ніж найменування навчального предмета з таблиці «PREDMET», при цьому вибираються викладачі з прізвищем, яке розміщується між буквами «К» та «С».

ДОДАТОК А

Зразок оформлення титульного аркуша

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА

Кафедра комп'ютерних наук та інформаційних технологій

ЗВІТ

з виконання завдань
із дисципліни
«Організація баз даних та знань»

Виконав:
студент (-ка) __ курсу

групи _____
ПІБ (повністю)

Перевірив:

Харків
ХНУМГ ім. О. М. Бекетова
20__

Електронне навчальне видання

Методичні рекомендації
до виконання практичних та організації самостійних робіт
з навчальної дисципліни

«ОРГАНІЗАЦІЯ БАЗ ДАНИХ ТА ЗНАНЬ»

(для здобувачів 2 курсу першого (бакалаврського) рівня вищої освіти денної форми навчання зі спеціальностей 126, F6 – Інформаційні системи та технології)

Укладачі: **КОСТЕНКО** Олександр Борисович,
ГАВРИЛЕНКО Ірина Олександрівна

Відповідальний за випуск *О. Б. Костенко*

Редактор *О. А. Норик*

Комп’ютерне верстання *І. О. Гавриленко*

План 2023, поз. 230М

Підп. до друку 19.05.2025. Формат 60 × 84/16.
Ум. друк. арк. 1,1.

Видавець і виготовлювач:
Харківський національний університет
міського господарства імені О. М. Бекетова,
вул. Чорноглазівська (Маршала Бажанова), 17, Харків, 61002.
Електронна адреса: office@kname.edu.ua
Свідоцтво суб’єкта видавничої справи:
ДК № 5328 від 11.04.2017.