МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Бази Даних

Лабораторний практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра
за освітньою програмою «Інженерія програмного забезпечення інформаційних систем»
спеціальності F2 «Інженерія програмного забезпечення»

Автор: К.І. Ліщук

Електронне мережеве навчальне видання

Київ
КПІ ім. Ігоря Сікорського
2026

УДК 004.65(075.8)

Автори: *Ліщук Катерина Ігорівна*, канд.техн.наук

Рецензент *Волокита А.М.,* канд. техн. наук, доцент, доцент кафедри обчислювальної техніки КПІ ім. Ігоря Сікорського

Відповідальний редактор *Олійник Ю.О.*, канд. техн. наук.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського
(протокол № 8 від 04.06.2026)
за поданням вченої ради Факультету інформатики та обчислювальної техніки
(протокол № 10 від 25.05.2026)

Бази даних [Електронний ресурс]: лаб. практикум: навч. посіб. для здобувачів ступеня бакалавра за освіт. програмою «Інженерія програмного забезпечення інформаційних систем» спец. F2 «Інженерія програмного забезпечення» / КПІ ім. Ігоря Сікорського; автори: Ліщук К.І. – Електрон. текст. дані (1 файл). – Київ : КПІ ім. Ігоря Сікорського, 2026. – 137 с.

Навчальний посібник містить теоретичний матеріал та практичні завдання, які необхідні для виконання лабораторних робіт з дисципліни «Бази даних» для студентів першого (бакалаврського) рівня вищої освіти спеціальності F2 «Інженерія програмного забезпечення» усіх форм навчання.
Навчальний посібник може бути корисним студентам відповідних спеціальностей при вивченні дисциплін, пов'язаних із проєктуванням баз даних та інженерією програмного забезпечення, а також при виконанні бакалаврських кваліфікаційних робіт, курсових проєктів.

УДК 004.65(075.8)

Реєстр. № НП 23/24-167. Обсяг 5,1 авт. арк.

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
проспект Берестейський, 37, м. Київ, 03056
https://kpi.ua
Свідоцтво про внесення до Державного реєстру видавців, виготовлювачів
і розповсюджувачів видавничої продукції ДК № 5354 від 25.05.2017 р.

© К. І. Ліщук, 2026
© КПІ ім. Ігоря Сікорського, 2026

ЗМІСТ

ЗАГАЛЬНІ ПОЛОЖЕННЯ 4

ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ 8

ЛАБОРАТОРНА РОБОТА № 1. ПОБУДОВА ER-МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ 13

ЛАБОРАТОРНА РОБОТА № 2. СТВОРЕННЯ БАЗИ ДАНИХ. КОРИСТУВАЧІ, РОЛІ, ПРАВА.. 27

ЛАБОРАТОРНА РОБОТА № 3. БАЗОВІ ОПЕРАТОРИ ВИБІРКИ, ПІДЗАПИТИ ТА З'ЄДНАННЯ В SQL 66

ЛАБОРАТОРНА РОБОТА № 4. СКЛАДНІ ЗАПИТИ З АГРЕГАТНИМИ ТА АНАЛІТИЧНИМИ ФУНКЦІЯМИ. РОБОТА З ПРЕДСТАВЛЕННЯМИ. 88

ЛАБОРАТОРНА РОБОТА № 5. ОСНОВИ ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ МОВИ SQL. ЗБЕРЕЖЕНІ ПРОЦЕДУРИ. КУРСОРИ. ТРИГЕРИ. 105

ЛІТЕРАТУРА.. 132

ДОДАТОК А 133

ДОДАТОК Б 133

ЗАГАЛЬНІ ПОЛОЖЕННЯ

Навчальний посібник «Бази даних» призначено для здобувачів вищої освіти ступеня бакалавра спеціальності F2 «Інженерія програмного забезпечення» за освітньою програмою «Інженерія програмного забезпечення інформаційних систем». Вивчення цього освітнього компонента посідає важливе місце у професійній підготовці майбутніх фахівців, оскільки сучасне програмне забезпечення інформаційних систем неможливо створювати, супроводжувати та розвивати без ґрунтовного розуміння принципів організації даних, способів їх структуризації, зберігання, опрацювання та захисту. Саме робота з даними є однією з базових складових інженерії програмного забезпечення, а тому опанування підходів до побудови баз даних, моделей подання даних і засобів керування ними є необхідною умовою підготовки фахівця, здатного розв'язувати практичні завдання у сфері розроблення інформаційних систем.

Метою вивчення є отримання здобувачами знань з основних понять з баз даних, формування практичних навичок проєктування реляційних моделей даних та використання мови SQL. Предметом вивчення є основні поняття ER-моделювання та реляційної теорії, введення в реляційні бази даних, нормальні форми, а також реляційні системи управління базами даних SQL. Лабораторна складова курсу зорієнтована на поетапне опанування етапів проєктування реляційних баз даних і практичне використання SQL, включаючи створення запитів та програмування в середовищі реляційної СУБД. Така побудова курсу забезпечує поєднання теоретичної підготовки з прикладною спрямованістю, що є особливо важливим для бакалаврської підготовки в галузі інженерії програмного забезпечення.

Основними завданнями освітнього компонента є формування у здобувачів знань про основні концепції побудови баз даних, методи і технології їх проєктування, адміністрування, створення та використання, а також основні підходи до маніпулювання даними та побудови SQL-запитів. У

процесі навчання здобувачі мають оволодіти вмінням проєктувати логічні та фізичні моделі баз даних, використовувати різноманітні СУБД, застосовувати безпечні методи роботи з даними, а також користуватися мовами опису реляційних моделей, запитів до баз даних, специфікацій і програмування. Очікується, що після завершення вивчення курсу студенти зможуть виконувати аналіз предметної області, будувати концептуальну модель, трансформувати її у фізичну модель бази даних, застосовувати нормалізацію таблиць і формалізацію зв'язків, створювати SQL-скрипти засобами DDL, DML, DCL і TCL та формувати SQL-запити різного рівня складності.

При вивченні освітнього компонента «Бази даних» у здобувачів вищої освіти забезпечується формування загальних і фахових компетентностей, котрі визначені силабусом. Насамперед робота з моделями даних, зв'язками між сутностями, абстрагуванням предметної області та переходом від реального об'єкта до його формалізованого опису забезпечує розвиток здатності до абстрактного мислення, аналізу та синтезу, тобто ЗК 01. Водночас аналіз інформаційних потреб предметної області, виділення сутностей, атрибутів і зв'язків, визначення обмежень цілісності та правил функціонування майбутньої системи формують здатність ідентифікувати, класифікувати та формулювати вимоги до програмного забезпечення, що відповідає ФК 01. Через побудову концептуальних, логічних і фізичних моделей бази даних, а також через розгляд структури сховища даних як невід'ємної частини інформаційної системи забезпечується формування ФК 02 і ФК 03, пов'язаних з участю у проєктуванні програмного забезпечення та розробленні його архітектурних компонентів.

Важливою складовою змісту є також орієнтація на якість програмного забезпечення та надійність даних. Саме тому вивчення підходів до нормалізації, підтримки цілісності, коректного моделювання зв'язків і організації доступу до даних безпосередньо сприяє формуванню ФК 04, тобто здатності формулювати та забезпечувати вимоги щодо якості програмного забезпечення у відповідность з вимогами замовника, технічнимо завданням та

стандартами. Центральне місце в курсі належить формуванню у здобувачів ФК 07, котра передбачає володіння знаннями про інформаційні моделі даних і здатність створювати програмне забезпечення для зберігання, видобування та опрацювання даних, адже саме ця компетентність реалізується під час побудови схем баз даних, написання запитів, створення представлень, процедур, тригерів і засобів керування доступом. Разом із цим під час виконання лабораторних робіт здобувачі застосовують фундаментальні та міждисциплінарні знання для розв’язання інженерних завдань, накопичують і систематизують професійні знання, обґрунтовано обирають інструментарій розробки та супроводження програмного забезпечення, а також розвивають алгоритмічне й логічне мислення, що відповідає ФК 08, ФК 10, ФК 12, ФК 13 і ФК 14.

Зміст лабораторного практикуму безпосередньо орієнтований і на досягнення програмних результатів навчання, визначених силабусом. Передусім це ПРН 13, який передбачає знання і вміння застосовувати методи конструювання програмного забезпечення та структур даних і знань. Досягнення цього результату забезпечується через побудову моделей даних, проєктування структури реляційної бази даних, застосування нормалізації, опрацювання реляційних операцій і розроблення SQL-запитів різної складності. Не менш важливим є ПРН 18, який полягає у здатності застосовувати інформаційні технології обробки, зберігання та передачі даних. Його формування відбувається під час практичної роботи зі створенням баз даних, виконанням операцій над даними, організацією доступу, використанням представлень, процедур, курсорів, тригерів та інших засобів реляційної СУБД. Таким чином, програмні результати навчання реалізуються не формально, а через систему конкретних навчальних дій і практичних завдань.

Структура практикуму побудована за логікою послідовного переходу від загальних теоретичних положень до виконання прикладних завдань. Зміст курсу охоплює основні поняття баз та моделей даних, архітектури СУБД,

реляційну модель даних, операції реляційної алгебри, нормалізацію, засоби SQL та програмування в SQL-середовищі. У межах лабораторних робіт студенти виконують аналіз предметної області, будують ER-моделі, створюють базу даних, таблиці та зв'язки між ними, працюють із користувачами, ролями та правами доступу, розробляють запити на вибірку, модифікацію та узагальнення даних, а також опановують засоби серверного програмування. Така послідовність забезпечує цілісне уявлення про процес створення бази даних — від концептуального проєктування до практичного використання в складі інформаційної системи.

Набуті під час його вивчення знання та вміння використовуються надалі в освітніх компонентах, пов'язаних з аналізом даних в інформаційних системах, компонентами програмної інженерії, переддипломною практикою та дипломним проєктуванням. Завдяки цьому лабораторний практикум виконує не лише допоміжну, а й системоутворювальну функцію в структурно-логічній схемі підготовки бакалавра, забезпечуючи основу для подальшого професійного розвитку у сфері розроблення інформаційних систем і програмного забезпечення, що працює з даними.

Посібник призначений для закріплення теоретичного матеріалу, формування практичних умінь проєктування та реалізації баз даних, а також набуття досвіду використання реляційних СУБД під час розв'язання професійно орієнтованих завдань. Його матеріали мають сприяти формуванню у здобувачів системного бачення процесу створення бази даних, уміння обґрунтовано приймати проєктні рішення, коректно організовувати структуру даних, забезпечувати цілісність інформації та ефективно використовувати засоби SQL у практиці інженерії програмного забезпечення. Саме тому лабораторний практикум слід розглядати як важливий інструмент поєднання теоретичної підготовки з професійною практикою, необхідною для майбутньої діяльності в галузі інформаційних технологій.

ЗАГАЛЬНІ МЕТОДИЧНІ ВКАЗІВКИ

Навчальний посібник призначений для методичного супроводу виконання лабораторних робіт з освітнього компоненту «Бази даних» і спрямований на систематизацію теоретичних знань та формування практичних умінь, необхідних для проєктування, створення та використання реляційних баз даних. Матеріал посібника побудовано з урахуванням необхідності послідовного переходу від теоретичного опрацювання основних понять і моделей даних до набуття практичного досвіду створення об'єктів бази даних, розроблення SQL-запитів і застосування засобів програмування в SQL-середовищі. Зміст лабораторного практикуму орієнтований на поетапне формування у здобувачів вищої освіти цілісного уявлення про процес створення бази даних як складової інформаційної системи — від аналізу предметної області та побудови її концептуальної моделі до реалізації фізичної схеми, організації доступу до даних, створення представлень, процедур, функцій, курсорів і тригерів.

Лабораторні роботи необхідно виконувати у послідовності, наведеній у посібнику, оскільки кожна наступна робота змістово пов'язана з попередньою та спирається на одержані раніше результати. На початковому етапі здобувач виконує аналіз предметної області, визначає її основні інформаційні об'єкти та їх атрибути, встановлює зв'язки між ними та будує ER-модель. Надалі на основі розробленої моделі створюється реляційна база даних, визначаються таблиці, атрибути, ключі, обмеження цілісності, а також реалізуються механізми керування доступом до даних. Після цього опановуються засоби побудови запитів на вибірку, модифікацію та узагальнення даних, створення представлень, а на завершальному етапі — програмування в середовищі SQL із використанням процедур, функцій, курсорів і тригерів. Така організація практикуму забезпечує логічну завершеність навчального процесу і дає змогу простежити взаємозв'язок між окремими етапами проєктування та реалізації бази даних.

Усі лабораторні роботи повинні виконуватися в межах однієї предметної області, визначеної індивідуальним варіантом завдання (приклади індивідуальних завдань наведені у Додатку Б). Результати, отримані під час виконання першої лабораторної роботи, мають використовуватися в наступних роботах без зміни загальної концепції предметної області, структури основних сутностей і логіки зв'язків між ними. Дотримання цієї вимоги є принципово важливим, оскільки забезпечує цілісність розроблюваної бази даних і формує у здобувача системне бачення процесу її створення. Не допускається довільна зміна предметної області або повне перепроєктування структури бази даних на наступних етапах без обґрунтованої потреби, оскільки це порушує логіку побудови лабораторного практикуму.

Перед початком виконання кожної лабораторної роботи здобувач повинен ознайомитися з відповідними теоретичними відомостями, уважно опрацювати постановку завдання, з'ясувати зміст індивідуального варіанта та підготувати необхідні вихідні дані. Під час виконання роботи слід керуватися не лише формальною вимогою отримати певний результат, а й усвідомлювати зміст виконуваних дій, їх логічне обґрунтування та місце в загальному процесі проєктування бази даних. Практична робота з базами даних передбачає не механічне відтворення прикладів, а самостійне застосування знань до конкретної предметної області, аналіз властивостей інформаційних об'єктів, обґрунтований вибір типів даних, ключів, обмежень і SQL-конструкцій.

Під час побудови бази даних необхідно дотримуватися загальноприйнятих принципів реляційного проєктування. Таблиці мають коректно відображати сутності предметної області, їх атрибути та зв'язки, а структура бази даних повинна бути логічно узгодженою, несуперечливою та придатною до подальшого використання в запитах і програмних об'єктах. Особливу увагу слід приділяти обґрунтованому вибору типів даних, визначенню первинних і зовнішніх ключів, встановленню обмежень NOT NULL, UNIQUE, CHECK, DEFAULT, а також забезпеченню цілісності та узгодженості даних. Під час створення таблиць, представлень, процедур,

функцій та інших об’єктів доцільно дотримуватися єдиного стилю іменування, використовувати зрозумілі та однозначні назви, уникати надмірних скорочень і забезпечувати відповідність між назвами об’єктів бази даних та термінологією, використаною в описі предметної області.

Під час виконання лабораторних робіт, пов’язаних зі створенням SQL-запитів, необхідно забезпечувати не лише їх синтаксичну правильність, а й змістову доцільність. Кожен запит має бути пов’язаний із конкретною інформаційною потребою, що випливає з особливостей предметної області, а його результат повинен мати чітке практичне призначення. Необхідно розуміти, які таблиці та зв’язки використовуються в запиті, які умови відбору застосовуються, які саме дані будуть отримані в результаті виконання та яким є зміст цього результату. Це особливо важливо для запитів із підзапитами, з’єднаннями, агрегатними та аналітичними функціями, операціями групування, сортування та для роботи з представленнями. У процесі виконання лабораторних робіт здобувач повинен демонструвати вміння не лише створити SQL-код, а й пояснити його логіку, призначення та особливості реалізації.

Під час створення програмних об’єктів бази даних, зокрема збережених процедур, функцій, курсорів і тригерів, необхідно враховувати доцільність їх застосування та змістовний зв’язок із розробленою базою даних. Такі об’єкти повинні реалізовувати практично значущі дії, характерні для відповідної предметної області, та бути органічною складовою структури бази даних. Особливу увагу слід приділяти перевірці правильності їх роботи, відповідності вхідних і вихідних параметрів поставленому завданню, а також впливу результатів виконання на стан даних у базі. Застосування курсорів має бути обґрунтованим і використовуватися лише в тих випадках, коли поставлену задачу недоцільно реалізувати засобами множинної обробки даних. Під час створення тригерів необхідно враховувати події, на які вони реагують, можливі наслідки їх виконання та їх вплив на цілісність і коректність даних.

Звіт з кожної лабораторної роботи оформлюється у вигляді окремого документа. Його зміст має бути логічно структурованим, послідовним і відображати основні етапи виконання завдання. Звіт повинен містити титульний аркуш (шаблон титульного листу наведений у Додатку А), постановку індивідуального завдання, результати виконання, необхідні ілюстративні матеріали, а також висновки. При цьому зміст звіту має враховувати специфіку конкретної лабораторної роботи. Для роботи, пов'язаної з побудовою ER-моделі, у звіті доцільно наводити опис предметної області, перелік сутностей, котрі виділені, їх атрибути та опис зв'язків між ними, пояснення до побудованої моделі та саму ER-модель. Для роботи зі створення бази даних необхідно подавати SQL-скрипти створення таблиць, ключів, обмежень, користувачів, ролей і прав доступу. У звітах до лабораторних робіт, присвячених SQL-запитам, слід наводити тексти запитів, їх словесний опис, результати виконання та пояснення отриманих результатів. У звіті до лабораторної роботи з програмування в SQL необхідно подавати тексти процедур, функцій, курсорів і тригерів, опис їх призначення та результати перевірки працездатності.

Усі SQL-скрипти, що включаються до звіту, повинні бути придатними до повторного виконання та містити коректні, узгоджені між собою назви об'єктів. За потреби до фрагментів коду можуть додаватися короткі коментарі, які пояснюють їх призначення або логіку роботи. Графічні матеріали, зокрема ER-діаграми та схеми бази даних, мають бути читабельними, акуратно оформленими, виконаними з дотриманням єдиної термінології та відповідати структурі створеної бази даних. Для побудови діаграм доцільно використовувати спеціалізовані програмні засоби або вебзастосунки, що забезпечують коректне графічне подання сутностей, атрибутів і зв'язків між ними.

Під час підготовки до захисту лабораторної роботи здобувач повинен бути готовим пояснити теоретичні положення, на яких ґрунтується виконане завдання, обґрунтувати прийняті проєктні рішення, охарактеризувати

структуру створеної бази даних, призначення таблиць, зв’язків, обмежень, запитів і програмних об’єктів, а також продемонструвати працездатність реалізованих рішень. Під час захисту особливе значення має не лише наявність виконаного коду або оформленого звіту, а й здатність здобувача усвідомлено пояснити логіку побудови бази даних, зміст отриманих результатів та їх відповідність поставленому завданню. Захист лабораторної роботи має підтверджувати, що здобувач володіє необхідними знаннями та практичними навичками, а також здатний самостійно застосовувати їх під час розв’язання професійно орієнтованих завдань.

Усі лабораторні роботи повинні виконуватися самостійно з дотриманням принципів академічної доброчесності. Самостійність виконання передбачає самостійне опрацювання теоретичного матеріалу, побудову моделей, розроблення SQL-коду, аналіз отриманих результатів і підготовку звіту. Використання зовнішніх джерел, довідкових матеріалів і документації допускається лише як допоміжний засіб для опанування навчального матеріалу та не повинно підміняти власну роботу здобувача. Дотримання зазначених методичних рекомендацій є необхідною умовою якісного виконання лабораторного практикуму та досягнення результатів навчання з дисципліни «Бази даних».

ЛАБОРАТОРНА РОБОТА № 1. ПОБУДОВА ER-МОДЕЛІ ПРЕДМЕТНОЇ ОБЛАСТІ

Мета:

– отримання навичок моделювання предметної області та побудови ER-моделі предметної області (діаграм «Сутність-Зв’язок»).

Короткі теоретичні відомості

Проєктування бази даних є багатоетапним процесом, де визначаються які дані треба зберігати, які правила їх організації, засоби ідентифікації об’єктів предметної області та логіка взаємодії між цими об’єктами. У загальному випадку виділяють концептуальний, логічний і фізичний рівні проєктування. Концептуальний рівень орієнтований на зміст предметної області й відповідає на запитання, які саме дані повинні зберігатися та які зв’язки між ними є суттєвими. Логічний рівень конкретизує структуру майбутньої бази даних у термінах сутностей, атрибутів, ключів і зв’язків. На фізичному рівні визначаються структура таблиць, типи даних, індекси та інші об’єкти у конкретній системі управління базами даних (СУБД) [1; 2; 6].

На концептуальному рівні одним із найпоширеніших інструментів є модель «Entity–Relationship Model» - модель «сутність–зв’язок» (зазвичай її скорочено називають ER-модель). Її призначення полягає у формалізованому поданні предметної області шляхом виділення інформаційних об’єктів, їх основних характеристик і залежностей між ними. Саме ER-модель дає змогу перейти від неструктурованого словесного опису предметнгого середовища до чіткої схеми даних, яка надалі може бути трансформована в реляційну модель. Історично підхід був запропонований Пітером Ченом у 1976 році й став підґрунтям більшості сучасних методів семантичного моделювання даних [1; 6].

Цінність розробки ER-моделі полягає в тому, що вона дозволяє ще до реалізації бази даних перевірити повноту опису предметної області, узгодити

термінологію між розробниками і замовником, виявити зайві або пропущені об'єкти, а також сформувати вимоги до майбутньої схеми бази даних. Побудова ER-моделі є необхідним першим кроком перед створенням схеми бази даних, написанням SQL-коду та реалізацією зв'язків між таблицями [2; 3].

Під предметною областю зазвичай розуміють частину реального або уявного світу, інформацію про який необхідно накопичувати, зберігати й обробляти в інформаційній системі. Під час аналізу предметної області виділяють об'єкти, події, процеси та характеристики, що мають інформаційну цінність. При розробці ER-моделі такі об'єкти формалізуються у вигляді сутностей, а їх властивості — у вигляді атрибутів [3; 4; 6].

Сутність — це інформаційно значущий об'єкт предметної області, дані про який доцільно зберігати в базі даних. Сутностями можуть бути документи, підрозділи, дисципліни, замовлення, товари, екзамени, аудиторії, операції тощо. Важливо розрізняти тип сутності й екземпляр сутності. Під поняттям «тип сутності» зазвичай маю на увазі клас однорідних об'єктів, наприклад «Студент», «Курс» або «Замовлення», тоді як екземпляр сутності — це конкретний представник такого класу, наприклад конкретний студент або конкретне замовлення [1; 3; 4].

Атрибутом називають властивість сутності, яка використовується для її опису. Саме атрибути визначають, які дані повинні бути збережені про об'єкт. Наприклад, для сутності «Студент» такими атрибутами можуть бути прізвище, ім'я, по-батькові, дата народження, група, електронна адреса та інші характеристики. Для сутності «Екзамен» атрибутами можуть бути дата, час, аудиторія та вид контролю. Доцільність включення атрибута до моделі визначається тим, наскільки ця характеристика потрібна для виконання функцій системи та розв'язання інформаційних задач.

Поряд із самим атрибутом важливим поняттям є домен атрибута, тобто множина допустимих значень, які може приймати атрибут. Наприклад, доменом для атрибута «Більксть балів» може бути множина значень від 0 до

100, а доменом для атрибута «Дата екзамену» — календарні дати. Урахування доменів є важливим як на етапі концептуального, так і на етапі логічного проєктування, оскільки надалі саме на основі доменів визначаються типи даних і перевірки коректності значень у СУБД.

Таблиця 1 - Основні види атрибутів у ER-моделі

Вид атрибута	**Характеристика**	**Приклад**
Простий	Не поділяється на змістовні складові частини.	Прізвище
Складений	Може бути логічно розкладений на кілька підатрибутів.	Адреса: країна, місто, вулиця
Багатозначний	Для одного екземпляра сутності може мати кілька значень.	Телефон, електронна адреса
Похідний	Обчислюється на основі інших атрибутів.	Вік за датою народження
Необов’язковий	Може бути відсутнім для окремих екземплярів сутності.	По батькові, примітка

У концептуальному моделюванні особливо важливо правильно відокремлювати атрибути від сутностей. Не кожне слово з опису предметної області повинно перетворюватися на окрему сутність. Якщо характеристика не має самостійного життєвого циклу, не бере участі у власних зв’язках і не потребує окремого зберігання як об’єкт, то, як правило, вона моделюється саме як атрибут. Наприклад, «Аудиторія» може бути атрибутом сутності «Екзамен», якщо не потрібно зберігати розширені відомості про аудиторії, але може бути окремою сутністю, якщо система повинна містити інформацію про корпус, місткість, обладнання, бронювання приміщень тощо.

Однією з базових вимог до коректної моделі даних є можливість однозначно ідентифікувати кожен екземпляр сутності. Для цього використовують ключі. У широкому сенсі ключем є атрибут або сукупність атрибутів, значення яких дають змогу відрізнити один екземпляр сутності від будь-якого іншого екземпляра тієї ж сутності. У теорії баз даних розрізняють

кандидатні ключі, первинні ключі, альтернативні ключі та зовнішні ключі [4; 6].

Первинний ключ повинен бути унікальним, стабільним і по можливості мінімальним за складом. Інколи в якості первинного ключа використовують штучний ідентифікатор, наприклад ID_студента або ID_замовлення, однак на концептуальному рівні доцільно спочатку з'ясувати, які природні властивості об'єкта можуть забезпечити однозначну ідентифікацію. У разі потреби природний ключ може бути доповнений або замінений сурогатним ключем на логічному чи фізичному рівні моделювання.

Правильне визначення ключів є критично важливим для подальшого переходу до реляційної схеми, оскільки саме ключі стають основою для створення первинних і зовнішніх ключів у таблицях, забезпечують цілісність даних і дозволяють реалізовувати зв'язки між сутностями. Відсутність чітко визначених ключів майже завжди призводить до дублювання даних, аномалій оновлення та труднощів при реалізації SQL-запитів [1; 4].

Зв'язок у ER-моделі відображає логічну залежність між двома або більше сутностями. Саме зв'язки формують цілісну структуру даних, перетворюючи множину окремих сутностей на узгоджену модель предметної області. Наприклад, студент навчається у групі, викладач читає дисципліну, клієнт оформлює замовлення, а замовлення містить товари. Без відображення таких зв'язків модель втратила б прикладний сенс, оскільки не давала б можливості описати взаємодію об'єктів у системі [1; 3; 4].

За кількістю сутностей, що беруть участь у взаємодії, розрізняють бінарні, тернарні та n-арні зв'язки. У більшості практичних задач використовуються саме бінарні зв'язки, котрі відображають зв'язки між двома сутностями. Однак у складніших предметних областях можуть виникати й тернарні зв'язки, коли повноцінний зміст взаємодії розкривається лише за участю трьох сутностей. Прикладом може бути взаємозв'язок між викладачем, дисципліною та академічною групою в контексті призначення занять [3; 6].

Найважливішою кількісною характеристикою зв’язку є його кардинальність. Розрізняють наступні типи зв’язків: «один до одного» (позначають як «1:1»), «один до багатьох» (позначають як «1:N») та «багато до багатьох» (позначають як «M:N»). Зв’язок 1:1 означає, що одному екземпляру першої сутності відповідає не більше одного екземпляра другої сутності. Зв’язок 1:N означає, що одному екземпляру першої сутності відповідає багато екземплярів другої. Зв’язок M:N означає, кожний екземпляр однієї сутності може бути пов’язаний з кількома екземплярами іншої сутності, і навпаки. Правильне визначення кардинальності надалі впливає на спосіб реалізації зв’язку в реляційній моделі [2; 4; 5; 6; 8].

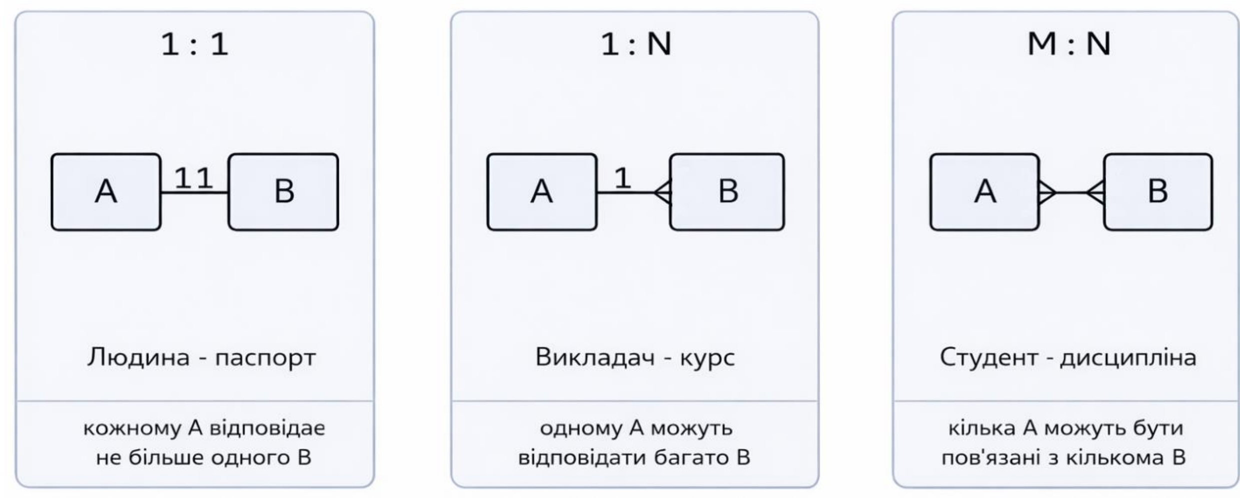


Рисунок 1 - Приклади кардинальностей зв’язків між сутностями

Окрім потужності зв’язку, враховується також обов’язковість або необов’язковість участі сутності у зв’язку. Тобто якщо кожен екземпляр сутності обов’язково повинен бути пов’язаний із екземпляром іншої сутності, то така участь є обов’язковою. Якщо ж наявність зв’язку для певного екземпляра можлива, але не гарантується, участь вважають необов’язковою. У подальшому це безпосередньо впливає на реалізацію обмежень цілісності, зокрема на допустимість значень NULL у зовнішніх ключах [4; 5; 6].

При проєктування важливо враховувати реальні бізнес-правила предметної області. Наприклад, твердження «кожен екзамен проводиться з однієї дисципліни» задає обмеження 1:N між сутностями «Дисципліна» і «Екзамен», тоді як твердження «один і той самий екзамен можуть складати

кілька груп» може вимагати введення окремої асоціативної сутності або проміжної таблиці при переході до реляційної моделі.

Окрім того, сутності умовно поділяють на сильні та слабкі. Сильна сутність має повний власний ключ і може бути ідентифікована незалежно від інших об’єктів. Слабка сутність не має достатнього набору атрибутів для самостійної ідентифікації й існує лише у зв’язку з іншою, сильною сутністю. У зв’язку з цим слабка сутність визначається через так званий ідентифікуючий зв’язок, який включає ключ батьківської сутності до складу її власної ідентифікації [3; 4; 6]. В якості прикладу слабкої сутності можна навести «Рядок замовлення». Номер рядка сам по собі не дає змоги однозначно ідентифікувати запис у масштабі всієї системи; він набуває змісту лише разом із ключем конкретного замовлення. Отже, первинний ключ такої слабкої сутності може бути складеним і містити як ключ сильної сутності, так і власний частковий ключ слабкої сутності.

Окремої уваги потребує розгляд асоціативних сутностей, які використовуються для декомпозиції зв’язків типу M:N. Оскільки в фізичній реалізації зв’язок «багато до багатьох» не реалізується безпосередньо, його, як правило, перетворюють на окрему проміжну таблицю. Така сутність містить зовнішні ключі на дві пов’язані сутності та, за потреби, власні атрибути, що описують сам факт взаємодії. Наприклад, зв’язок між студентом і дисципліною може бути поданий через сутність «Запис на дисципліну», яка додатково містить дату, статус, форму контролю або оцінку.

Якісна ER-модель повинна відображати не лише перелік сутностей, атрибутів і зв’язків, а й основні бізнес-правила предметної області. До таких правил належать умови унікальності, допустимі значення атрибутів, обов’язковість певних зв’язків, обмеження на кількість пов’язаних об’єктів, залежності між атрибутами, часові або організаційні умови функціонування системи. Формально частина цих правил фіксується вже на концептуальному рівні, а в подальшому реалізується через ключі, зовнішні ключі, перевірки

доменів, обмеження CHECK, UNIQUE, NOT NULL і правила посилальної цілісності в СУБД [2; 3; 6].

З погляду теорії баз даних особливе значення мають три групи обмежень: цілісність сутності, цілісність домену та посилальна цілісність. Цілісність сутності передбачає, що кожен екземпляр сутності має бути ідентифікований ключем. Цілісність домену вимагає, щоб значення атрибутів належали визначеним множинам допустимих значень. Посилальна цілісність забезпечує узгодженість зв’язків між сутностями, тобто неможливість існування посилання на неіснуючий батьківський об’єкт. Навіть якщо на етапі ER-моделювання ці обмеження ще не виражаються SQL-конструкціями, вони повинні бути логічно зафіксовані в моделі [4; 6]. Важливо навчитися читати словесний опис предметної області як сукупність бізнес-правил. Наприклад, фраза «кожен студент належить рівно одній академічній групі» означає обов’язкову участь сутності «Студент» у зв’язку з «Групою» та кардинальність 1:N. Фраза «група може складати кілька екзаменів, а один екзамен можуть складати кілька груп» вказує на M:N-зв’язок.

Для графічного подання ER-моделей використовують кілька нотацій. Класичною є нотація Чена, у якій сутності зображуються прямокутниками, атрибути — овалами, а зв’язки — ромбами. Її перевагою є наочність і простота пояснення базових понять моделювання [1; 6].

У практиці інженерії баз даних часто застосовують нотацію Crow’s Foot. Вона є компактнішою та зручнішою для розуміння інженерів, оскільки сутності зазвичай подаються у вигляді таблиць зі списком атрибутів, а кардинальність відображається спеціальними позначеннями на кінцях ліній зв’язку. Вибір конкретної нотації не змінює сутності моделі, але впливає на зручність її читання, деталізацію та подальшу трансформацію в реляційну схему [2; 4; 5].

Побудова ER-моделі повинна здійснюватися не хаотично, а за певною послідовністю дій. Насамперед аналізують предметну область і виділяють інформаційні об’єкти, які мають самостійне значення для системи. Далі

формують перелік сутностей і визначають їх атрибути. Після цього встановлюють ключові атрибути, що забезпечують унікальну ідентифікацію екземплярів. На наступному етапі виявляють зв'язки між сутностями, визначають їх тип, кардинальність та обов'язковість участі. У разі потреби виділяють слабкі або асоціативні сутності. Далі обирають нотацію і будують саму ER-діаграму, яку в кінці перевіряють на повноту, несуперечливість і відповідність вихідному опису предметної області.

Після побудови первинного варіанта ER-діаграми її необхідно критично переглянути. Слід перевірити, чи всі важливі інформаційні об'єкти відображені у моделі, чи немає зайвих сутностей, які насправді є лише атрибутами, чи коректно визначені ключі, чи правильно задано кардинальність зв'язків, а також чи не приховано у формулюваннях опису предметної області додаткових обмежень, які ще не були зафіксовані. Така перевірка є обов'язковою, оскільки помилки, допущені на концептуальному етапі, надалі множаться під час створення таблиць і написання SQL-коду.

Хоча ER-модель належить до концептуального рівня, її головна практична функція полягає в підготовці до створення реляційної схеми. У найпростішому випадку сильна сутність перетворюється на таблицю, її атрибути — на стовпці таблиці, а ключовий атрибут — на первинний ключ. Зв'язки типу 1:N, як правило, реалізуються перенесенням первинного ключа батьківської сутності до дочірньої сутності у вигляді зовнішнього ключа. Для зв'язків 1:1 вибір конкретного способу реалізації залежить від обов'язковості участі й особливостей предметної області. Зв'язки M:N реалізуються через окремі проміжні таблиці або асоціативні сутності. На цьому етапі також виявляється важливість правильної декомпозиції складених атрибутів, обґрунтованого виділення багатозначних атрибутів і коректного тлумачення слабких сутностей. Наприклад, якщо атрибут є багатозначним, його зберігання в межах однієї таблиці може призвести до порушення нормалізації, тому доцільно винести його в окрему сутність. У свою чергу, складений атрибут може бути розкладений на окремі стовпці, якщо це покращує подальшу

обробку даних. Таким чином, якісно побудована ER-модель зменшує кількість помилок під час створення реляційної структури, полегшує реалізацію SQL-запитів і створює передумови для забезпечення цілісності, несуперечливості й масштабованості бази даних. Саме тому концептуальне моделювання не є формальною попередньою вправою, а виступає повноцінною стадією проєктування.

Серед найбільш поширених помилок під час побудови ER-моделей можна виділити наступні:

- відсутність чітко визначених ключів;
- змішування сутностей і атрибутів;
- надмірна деталізація або, навпаки, надмірне узагальнення сутностей;
- неправильне визначення кардинальності зв’язків;
- ігнорування обов’язковості участі;
- неврахування слабких або асоціативних сутностей;
- механічне копіювання формулювань із тексту предметної області без змістовного аналізу.

Щоб уникнути зазначених недоліків, рекомендується дотримуватися кількох практичних правил. По-перше, кожна сутність повинна відповідати реальному інформаційно значущому об’єкту предметної області. По-друге, кожен атрибут повинен належати саме тій сутності, яку він характеризує. По-третє, перед фіксацією кардинальності зв’язку слід формулювати її у вигляді зрозумілих словесних правил, наприклад: «один викладач може викладати кілька дисциплін, але кожна дисципліна закріплена за одним викладачем». По-четверте, всі спірні випадки потрібно перевіряти через подальший перехід до реляційної схеми: якщо реалізація в таблицях виглядає штучною або надмірно складною, це часто вказує на помилку в концептуальній моделі.

Особливої уваги потребує моделювання M:N-зв’язків. В практичні діяльності зазвичай оцінюють чи не має сам зв’язок власних атрибутів. Якщо такі атрибути існують, наприклад, дата запису на курс, оцінка, статус

реєстрації, примітка або порядок виконання, то зв'язок повинен бути поданий як асоціативна сутність. Такий підхід робить модель гнучкішою і ближчою до реальної логіки предметної області.

При перевірці на коректність кінцевого варіанту розробленої ER-діаграми варто поставити щонайменше такі контрольні запитання:

- чи має кожна сутність ключ;
- чи всі атрибути справді потрібні системі;
- чи не дублюється інформація в різних сутностях;
- чи зрозуміло сформульовано всі зв'язки;
- чи не приховано в тексті предметної області додаткових обмежень.

Якщо хоча б на одне з цих запитань відповідь є не стверджувальною, модель потребує доопрацювання.

Постановка задачі лабораторної роботи

При виконанні лабораторної роботи необхідно виконати наступні дії:

1) вивчити основні теоретичні засади проектування баз даних, семантичного моделювання, побудови ER-діаграм (моделей «сутність-зв'язок»);
2) виділити основні множини сутностей, їх атрибути, зв'язки між ними згідно наданого опису предметної області. Мінімальна кількість сутностей – 6;
3) побудувати ER-модель предметної області;
4) за бажанням декомпозувати зв'язки «багато-до-багатьох».

Вимоги до оформлення звіту з лабораторної роботи

Звіт повинен містити наступні складові частини:

- титульний лист;
- назву та мету роботи;
- варіант;
- перелік виділених сутностей та атрибутів;

- короткий опис сутностей та атрибутів;
- ER-модель;
- висновки.

Контрольні запитання

1) Що таке ER-модель? Яка її роль у процесі проєктування бази даних?
2) Що розуміють під поняттям «сутність» у контексті ER-моделі? Наведіть приклади.
3) Які типи атрибутів існують у ER-моделюванні? В чому різниця між простим та складеним атрибутом?
4) Що таке первинний ключ? Чим він відрізняється від зовнішнього ключа?
5) Поясніть, що таке зв'язок між сутностями. Які бувають типи зв'язків (1:1, 1:N, M:N)? Наведіть приклади.
6) Яку роль відіграє кардинальність зв'язків у ER-моделі? Як вона відображається?
7) Що таке обов'язковість участі сутності у зв'язку? Як це впливає на проєктування?
8) Поясніть відмінність між сильною та слабкою сутністю. Який зв'язок використовується для ідентифікації слабкої сутності?
9) Що таке ідентифікуючий зв'язок? У яких випадках його використовують?
10) Опишіть, що таке агрегація у ER-моделюванні. Коли вона застосовується?
11) Яка різниця між спеціалізацією та генералізацією? Наведіть приклади застосування.
12) Які нотації використовуються для побудови ER-моделей? У чому відмінність між нотаціями Чена та Crow's Foot?
13) Як визначити, що модель містить надлишкові або дубльовані зв'язки? Які наслідки це має?

14) Які типові помилки допускають при побудові ER-моделі? Як їх уникнути?
15) Як перетворити ER-модель у реляційну схему бази даних (таблиці)? Опишіть алгоритм трансформації.

Приклад виконання завдання лабораторної роботи

Розглянемо покроково процес виконання лабораторної роботи для наступного предметного середовища: ***Предметне середовище «Розклад екзаменів».*** Система, котра проєктується, повинна зберігати інформацію про студентів, їх групи, предмети, екзамени (по якому предмету, якою групою та коли здається). Для кожної групи повинна існувати можливість отримати розклад екзаменів кожної групи, списки груп та отримані оцінки. Окрім того, один й той самий екзамен можуть здавати одразу декілька груп.

1) Виділення множин сутностей, атрибутів та зв'язків

Сутності та їх атрибути:

№	Сутність	Атрибути
1	Студент	ПІБ, дата народження, email
2	Група	назва групи, рік вступу
3	Дисципліна	назва дисципліни, кількість кредитів
4	Екзамен	дата і час, аудиторія, дисципліна
5	Оцінка	студент, екзамен, оцінка
6	Група_Екзамен	екзамен, група, примітка

Зв'язки між сутностями:

Зв'язок	Тип зв'язку	Опис
Студент — належить до → Група	N:1	Кожен студент навчається в одній групі
Група — складає → Екзамен	M:N	Один і той самий екзамен можуть складати кілька груп
Екзамен — перевіряє → Дисципліна	N:1	Кожен екзамен пов'язаний з однією дисципліною
Студент — отримує → Оцінка	1:N	Кожен студент отримує одну оцінку за кожен екзамен
Екзамен — фіксується в → Оцінка	1:N	Один екзамен має багато оцінок (по кожному студенту)

2) Опис сутностей та їх призначення

Сутність «Студент». ***Призначення:*** Зберігає персональні дані студентів, які складають екзамени.

Основні атрибути:

- ПІБ — прізвище, ім’я, по батькові;
- дата народження;
- email.

Сутність «Група». ***Призначення:*** Відображає навчальні групи, в які об’єднані студенти.

Основні атрибути:

- назва групи — наприклад, ІП-31;
- рік вступу.

Сутність «Дисципліна». ***Призначення:*** Містить інформацію про академічні дисципліни.

Основні атрибути:

- назва дисципліни;
- кількість кредитів.

Сутність «Екзамен». ***Призначення:*** Представляє факт проведення екзамену з певної дисципліни у визначену дату та час.

Основні атрибути:

- дата і час;
- аудиторія;
- дисципліна — дисципліна, з якої проводиться екзамен.

Сутність «Оцінка». ***Призначення:*** Зберігає отримані студентами оцінки за екзамени.

Основні атрибути:

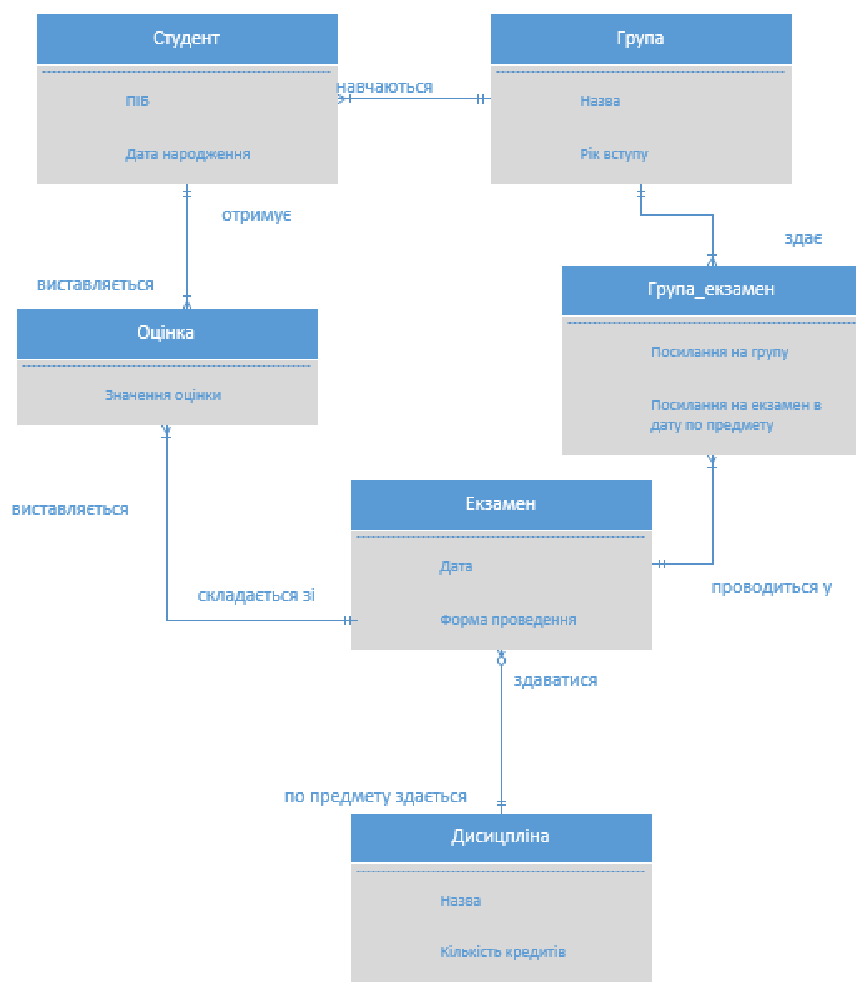
- студент;
- екзамен;
- оцінка.

Сутність «Група_Екзамен». ***Призначення:*** Реалізує зв'язок M:N між групами та екзаменами (кілька груп можуть здавати один і той самий екзамен).

Основні атрибути:

- екзамен;
- група;
- примітка — опціонально, наприклад, щодо дати оголошення

3) Побудова ER-моделі



ЛАБОРАТОРНА РОБОТА № 2. СТВОРЕННЯ БАЗИ ДАНИХ. КОРИСТУВАЧІ, РОЛІ, ПРАВА.

Мета:

– створення бази даних шляхом визначення схеми БД;

– вивчити DDL-команди SQL для роботи з таблицями (створення, модифікація та видалення таблиць);

– вивчити використовувані в SQL засоби для підтримки цілісності даних та їх практичне застосування;

– вивчити основні принципи керування обліковими записами та ролями.

Короткі теоретичні відомості

Бази даних (БД) є основою сучасних інформаційних систем. Вони забезпечують централізоване сховище даних, можливість багаторазового доступу, захист інформації та підтримку цілісності даних. Тобто база даних — це організована сукупність взаємопов'язаних даних, призначена для збереження, обробки та ефективного доступу до них. Реляційні бази даних ґрунтуються на математичній теорії відношень і представлені у вигляді таблиць.

Система управління базами даних (СУБД) — це програмне забезпечення, що забезпечує створення, наповнення, модифікацію та спільне використання баз даних.

Проектування баз даних — це процес створення структури БД, яка забезпечить ефективне збереження та доступ до даних. Процес передбачає аналіз вимог, побудову ER-діаграм, визначення сутностей, атрибутів, зв'язків, типів зв'язків, а також приведення до нормальних форм (від 1NF до 3NF і далі).

Проєктування бази даних починається з побудови концептуальної моделі предметної області (часто у вигляді ER-діаграми), яка фіксує ключові сутності, їх атрибути та зв'язки. Далі виконується логічне проєктування — перехід до реляційної моделі (таблиці, ключі, типи даних, обмеження).

Завершальний етап — фізична реалізація в обраній СУБД: створення схем, таблиць, індексів, налаштування користувачів, ролей і прав доступу. Така послідовність забезпечує узгодженість, масштабованість і безпеку системи.

Після побудови ER-моделі предметної області наступним етапом проєктування є перехід до логічної та фізичної реалізації бази даних. Якщо на концептуальному рівні визначаються сутності, їх атрибути та зв'язки, то на рівні реалізації ці елементи трансформуються у таблиці, стовпці, ключі, обмеження цілісності, індекси та інші об'єкти СУБД. Саме на цьому етапі формується реальна структура сховища даних, придатна до подальшого наповнення, обробки та використання прикладними програмами [7].

У реляційній моделі основним об'єктом зберігання є таблиця. Кожна таблиця представляє певну сутність або асоціацію предметної області, а кожний її рядок відповідає окремому екземпляру сутності. Стовпці таблиці відображають атрибути об'єкта та задають допустимий формат зберігання значень. Реляційна структура повинна бути логічно узгодженою, недвозначною та придатною до подальшої нормалізації, аналізу та побудови SQL-запитів.

З погляду методики проєктування розрізняють три рівні моделювання: концептуальний, логічний і фізичний. Концептуальна модель відображає бізнес-сутність даних без прив'язки до конкретної СУБД. Логічна модель деталізує майбутню структуру таблиць, ключів та зв'язків. Фізична модель фіксує конкретну реалізацію в певному середовищі, з урахуванням підтримуваних типів даних, синтаксису DDL-команд, індексації, схеми безпеки та організації зберігання. Щоб створити базу даних, спочатку необхідно визначити структуру (які таблиці потрібні, які в них стовпці, які зв'язки між таблицями), а потім реалізуємо цю структуру у визначеній СУБД.

Команди мови SQL можна поділити на чотири основні категорії:

– мова визначення даних (Data Definition Language - DDL) — містить команди, за допомогою яких створюють, модифікують і знищують об'єкти

(таблиці, індекси, представлення і так далі) у базі даних (CREATE, ALTER, DROP);

– мова маніпулювання даними (Data Manipulation Language - DML) — містить набір команд, за допомогою яких виконується додавання, модифікаця, видалення даних в об’єктах бази даних, створюють різноманітні запити до даних (INSERT, UPDATE, DELETE, SELECT);

– мова керування даними (Data Control Language - DCL) — складається із команд, за допомогою яких визначають дозволи на виконання певних дій користувачем ні (GRANT, DENY, REVOKE);

– транзакційна цілісність забезпечується засобами TCL.

Розуміння ролі кожної з цих груп команд є необхідним для цілісного сприйняття архітектури СУБД. База даних є окремим логічним контейнером для пов’язаних між собою об’єктів.

Мова визначення даних (DDL, Data Definition Language) призначена для опису структури бази даних або окремих її складових. За допомогою DDL-команд визначаються об’єкти бази даних, їхні властивості, типи даних, структура таблиць, зв’язки між об’єктами та обмеження цілісності. DDL належить до непроцедурних засобів SQL, оскільки її команди описують, якою має бути структура даних, а не задають покроковий алгоритм її обробки. Іншими словами, за допомогою DDL користувач формулює визначення об’єктів бази даних, конкретний спосіб їх створення, збереження та подальшої обробки забезпечується самою СУБД. Залежно від особливостей конкретної системи управління базами даних, обробка DDL-описів може здійснюватися по-різному. В одних випадках команда спочатку перевіряється на синтаксичну коректність і, за відсутності помилок, одразу виконується, створюючи або змінюючи відповідний об’єкт бази даних. В інших випадках можливе попереднє опрацювання описів з метою виявлення помилок, після чого коректні визначення зберігаються у внутрішніх системних структурах СУБД. Надалі ці описи використовуються системою під час виконання запитів, звернення до об’єктів бази даних або перевірки правил цілісності. Таким

чином, DDL-команди забезпечують формальне визначення схеми бази даних і створюють основу для подальшого збереження, обробки та контролю даних у реляційній СУБД.

До переліку основних функцій мови визначення даних відносять:

– ідентифікацію типів даних;

– специфікацію структури об'єктів бази даних;

– специфікацію ключів;

– визначення характеристик елементів даних: довжини символьних рядків, масштаб та точність числових даних, тощо;

– обмеження цілісності.

Дані, що зберігаються в стовпцях таблиць реляційної бази даних, є типізованими. Кожен стовпець під час створення таблиці пов’язується з певним типом даних або доменом, який визначає множину допустимих значень для цього атрибута. Такі типи можуть бути вбудованими в мову SQL або визначатися користувачем засобами конкретної СУБД. Після створення таблиці СУБД забезпечує контроль відповідності значень визначеним типам і не допускає збереження даних, що порушують встановлені правила допустимості.

Коректний вибір типів даних впливає на продуктивність, обсяг зберігання та цілісність і є однією з найважливіших умов якісного фізичного проєктування. Тип даних визначає, які значення дозволені в атрибуті, скільки пам’яті буде займати кожен запис, як відбуватиметься порівняння, сортування й обчислення, а також чи зможе оптимізатор ефективно використовувати індекси.

Основні типи даних:

– цілі - TINYINT/SMALLINT/INT/BIGINT, використовуються для цілих чисел;

– числові з фіксованою точністю - DECIMAL(p,s)/NUMERIC(p,s) — зазвичай використовують для фінансових чи кількісних значень із фіксованою точністю;

– дата/час - DATE, TIME, DATETIME2;

– символьні - CHAR/NCHAR, VARCHAR/NVARCHAR - фіксована довжина доцільна лише там, де значення мають стабільний розмір, а в інших випадках змінна довжина дозволяє економніше використовувати пам’ять. Використання префікса N є важливим для коректної підтримки Юнікоду, зокрема українських символів;

– логічні - BIT — для булевих ознак;

– унікальні ідентифікатори - UNIQUEIDENTIFIER (GUID).

Під час вибору типу даних необхідно виходити з природи атрибута, а не з тимчасових зручностей. Наприклад, номер залікової книжки або телефонний номер не слід зберігати як INT, якщо для нього істотними є нулі на початку чи формат запису. Аналогічно, дата не повинна зберігатися у вигляді тексту, оскільки це унеможливлює коректне сортування й арифметику дат. Отже, тип даних має відображати семантику атрибута і водночас бути технологічно доцільним.

Створення бази даних й схеми.

В одній базі даних можуть існувати схеми — логічні простори імен, що використовуються для впорядкування об’єктів і розмежування доступу. Схема не є простою папкою; вона має власника і бере участь у моделі безпеки. Практика використання окремих схем є корисною, оскільки дозволяє групувати об’єкти за функціональним призначенням: наприклад, навчальні таблиці можуть міститися у схемі edu, адміністративні — у schema admin, службові представлення — у reporting. Створити базу даних та схему можна за допомогою нижче наведеного скрипта.

```
CREATE DATABASE db_name
[ CONTAINMENT = { NONE | PARTIAL } ]
[ ON  PRIMARY ( NAME = logical_name, FILENAME = 'path\file.mdf', SIZE =
..., MAXSIZE = ..., FILEGROWTH = ... ) [ , ... ]
[ LOG ON ( NAME = logical_name_log, FILENAME = 'path\file.ldf', SIZE =
..., MAXSIZE = ..., FILEGROWTH = ... ) ] ];
```

де

– db_name — назва БД (рекомендується створювати без пробілів та англомовною транслітерацією);

– ON/LOG ON — в цих секціях задаються параметри розміщення файлів даних і журналу транзакцій (шлях, початковий розмір, зростання, максимум).

```
CREATE SCHEMA schema_name [ AUTHORIZATION owner_name ];
```

де

– schema_name — назва схеми (логічного контейнеру об’єктів у БД (напр., edu));

– AUTHORIZATION — власник схеми.

Створення таблиць.

Одним із базових етапів фізичного проєктування бази даних є створення таблиць. Саме таблиця в реляційній моделі виступає основною формою зберігання даних, а тому правильне визначення її структури безпосередньо впливає на цілісність, узгодженість, продуктивність і подальшу придатність бази даних до експлуатації. У загальному випадку створення таблиці передбачає визначення набору стовпців, вибір типів даних, формулювання правил допустимості значень, а також задання ключів і обмежень, що відображають логіку предметної області.

Слід зауважити, що команда створення таблиці задає не лише форму зберігання даних, а й механізми контролю їх коректності. Іншими словами, на цьому етапі визначається не тільки те, які саме дані будуть зберігатися в таблиці, а й те, які значення вважаються допустимими, які атрибути є обов’язковими, які з них повинні бути унікальними та яким чином таблиця пов’язується з іншими таблицями бази даних. Саме тому операція створення таблиці має розглядатися як одна з ключових процедур реалізації логічної моделі на фізичному рівні.

У більшості систем управління базами даних таблиця створюється за допомогою оператора CREATE TABLE. В загальному вигляді оператор створення таблиці виглядає наступним чином:

```
CREATE TABLE [database_name.][schema_name.]table_name (
    column_name data_type [ (length | precision [, scale]) ]
        [ COLLATE collation_name ]
        [ NULL | NOT NULL ]
        [ SPARSE ]
        [ ROWGUIDCOL ]
        [ IDENTITY (seed, increment) ]
        [ DEFAULT (constant | expression) ]
        [ <column_constraint> [ , ... ] ]
  [ , <table_constraint> [ , ... ] ]
)
[ ON { partition_scheme_name (column_name) | filegroup | "default" } ]
[ TEXTIMAGE_ON { filegroup | "default" } ];
```

У наведеній конструкції table_name визначає ім’я таблиці, а для кожного стовпця задаються його ім’я, тип даних та набір додаткових характеристик. Таким чином, структура таблиці формується як сукупність стовпців і табличних обмежень. Стовпці задають безпосередньо формат даних, а обмеження визначають правила, яких ці дані мають дотримуватися.

Розглянемо основні складові команди створення таблиці. Насамперед для кожного стовпця необхідно визначити тип даних. Тип даних задає формат збереження значень, допустимий діапазон, можливість виконання певних операцій, а також впливає на обсяг займаної пам’яті та продуктивність запитів. Коректний вибір типу даних є принципово важливим, оскільки він повинен відповідати змісту атрибута предметної області та забезпечувати достатню точність і надійність зберігання. Окрему увагу слід приділити властивості NULL або NOT NULL. Вона визначає, чи допускається відсутність значення у відповідному стовпці. Якщо участь атрибута в описі об’єкта є обов’язковою, слід використовувати NOT NULL. Якщо ж для певних записів значення може бути невідомим, допускається використання NULL. Таким чином, уже на етапі створення таблиці в її структурі формалізуються правила обов’язковості окремих атрибутів.

Для текстових стовпців може застосовуватися параметр COLLATE, який визначає правила порівняння та сортування символьних значень. За допомогою цього параметра задаються мовні особливості, правила регістрочутливості, тощо. Використання COLLATE є доцільним у тих

випадках, коли для окремого стовпця потрібно встановити спеціальні правила сортування, відмінні від загальних налаштувань бази даних.

У деяких випадках стовпці можуть мати додаткові спеціальні властивості. Зокрема, параметр SPARSE використовується для стовпців, у яких передбачається велика частка значень NULL. У такому разі SQL Server застосовує спеціальний механізм зберігання, що дозволяє зменшити обсяг зайнятого місця. Властивість ROWGUIDCOL використовується для позначення стовпця, який містить значення типу UNIQUEIDENTIFIER і виконує роль системного глобально унікального ідентифікатора.

Ще одним важливим елементом опису стовпця є DEFAULT. Цей параметр дозволяє визначити значення за замовчуванням, яке буде автоматично підставлятися під час вставлення нового рядка, якщо користувач не задав значення явно. Використання DEFAULT дає змогу зменшити кількість помилок під час введення даних і підтримати типові сценарії роботи з базою даних. Наприклад, для логічного стовпця стану можна встановити початкове значення 1, а для дати створення запису — поточну дату.

У деяких випадках під час створення таблиці необхідно визначити не лише її логічну структуру, а й фізичне розміщення даних. Для цього в SQL Server використовуються параметри ON і TEXTIMAGE_ON. Параметр ON дозволяє вказати файлову групу або схему партиціонування, у якій розміщуватимуться дані таблиці. Це особливо актуально у великих базах даних, де необхідно розподіляти дані між різними носіями або організовувати партиціонування. Параметр TEXTIMAGE_ON використовується для визначення місця зберігання великих об’єктів, таких як VARCHAR(MAX), NVARCHAR(MAX), VARBINARY(MAX), XML.

Слід підкреслити, що створення таблиці є не ізольованою технічною операцією, а завершальним етапом переходу від концептуальної та логічної моделі до фізичної реалізації бази даних. Саме на цьому етапі сутності предметної області перетворюються на таблиці, атрибути — на стовпці, а зв’язки між сутностями — на зовнішні ключі та інші обмеження цілісності.

Тому якість команди CREATE TABLE визначається не лише синтаксичною правильністю, а й тим, наскільки коректно вона відображає логіку моделі даних.

Забезпечення автонумерації стовпців.

Під час проєктування баз даних однією з типових задач є організація механізму автоматичного формування унікальних числових значень для нових записів. Такі значення найчастіше використовуються як первинні ключі, оскільки забезпечують однозначну ідентифікацію кожного рядка таблиці та спрощують встановлення зв’язків між таблицями. У різних СУБД ця можливість реалізується за допомогою різних засобів, зокрема через властивості автоінкременту стовпця, генератори, послідовності або спеціалізовані механізми нумерації. Незважаючи на відмінності в синтаксисі та способі реалізації, загальне призначення таких засобів полягає в автоматизації процесу присвоєння унікальних ідентифікаторів без необхідності ручного введення значень користувачем.

Використання системних механізмів нумерації дозволяє уникнути дублювання значень ключів, зменшити ймовірність помилок під час введення даних, а також централізувати керування процесом ідентифікації записів. Особливо важливим це є в багатокористувацьких інформаційних системах, де одночасне додавання нових записів кількома користувачами потребує гарантованого формування унікальних значень первинних ключів.

Одним із найпоширеніших підходів до автоматичної генерації ідентифікаторів є використання спеціальної властивості стовпця, яка забезпечує автоматичне збільшення числового значення під час вставлення нового запису. У MSSQL Server таким засобом є властивість IDENTITY. Вона задається безпосередньо під час опису стовпця таблиці та визначає початкове значення лічильника й крок його зміни.

Загальний синтаксис оголошення такого стовпця має вигляд:

```
Id INT IDENTITY(seed, increment)
```

де seed задає початкове значення, а increment — величину приросту. Таким чином, властивість IDENTITY дозволяє визначити, з якого числа починатиметься нумерація та з яким кроком формуватимуться наступні значення.

Розглянемо приклад створення таблиці з автоінкрементним ідентифікатором:

```
CREATE TABLE dbo.Student
(
    Id INT IDENTITY(1,1) PRIMARY KEY,
    FullName NVARCHAR(100) NOT NULL,
    GroupName NVARCHAR(20) NOT NULL
);
```

У наведеному прикладі стовпець Id автоматично набуватиме послідовних значень, починаючи з 1, із кроком 1. Отже, під час кожного додавання нового запису до таблиці система самостійно формуватиме нове значення первинного ключа. Такий підхід є зручним у тих випадках, коли генерація ідентифікаторів потрібна лише в межах однієї таблиці й не потребує складного керування.

Разом із тим на практиці можуть виникати ситуації, коли поточне значення лічильника IDENTITY необхідно змінити. Наприклад, така потреба може з'явитися після очищення таблиці, тестового заповнення даними, відновлення резервної копії або імпорту записів із зовнішнього джерела. У MSSQL Server для перевстановлення поточного значення генератора використовується команда:

```
DBCC CHECKIDENT ('dbo.Student', RESEED, 0);
```

Ця команда дозволяє змінити значення внутрішнього лічильника для таблиці. Якщо вказати значення 0, то наступний вставлений рядок, за умови стандартного інкременту 1, зазвичай отримає значення 1. Таким чином, команда DBCC CHECKIDENT використовується для синхронізації генератора з фактичним станом даних у таблиці.

Необхідно також враховувати, що за замовчуванням SQL Server не дозволяє явно вставляти значення у стовпець, визначений як IDENTITY, оскільки передбачається, що такі значення мають формуватися автоматично.

Однак у деяких практичних ситуаціях, наприклад під час перенесення даних або відновлення існуючих записів, може виникнути потреба тимчасово дозволити явне задання значень у такому стовпці. Для цього використовується команда:

```
SET IDENTITY_INSERT dbo.Student ON;
```

Після ввімкнення цього режиму допускається виконання вставлення з явним заданням значення для стовпця Id:

```
INSERT INTO dbo.Student (Id, FullName, GroupName) VALUES (100,
N'Іваненко Марія', N'ІП-31');
SET IDENTITY_INSERT dbo.Student OFF;
```

Слід підкреслити, що використання IDENTITY_INSERT має бути обґрунтованим і тимчасовим, оскільки некоректне ручне задання значень у стовпці автоінкременту може спричинити конфлікти унікальності або порушення логіки нумерації записів.

Більш гнучким засобом автоматичної генерації числових значень є SEQUENCE. На відміну від IDENTITY, послідовність не є властивістю конкретного стовпця, а створюється як окремий об’єкт бази даних. Це означає, що одна й та сама послідовність може використовуватися в різних таблицях, представленнях, процедурах, функціях або тригерах. Саме тому механізм SEQUENCE розглядається як універсальніший інструмент порівняно з IDENTITY.

Створення послідовності в MSSQL Server виконується за допомогою команди:

```
CREATE SEQUENCE dbo.SeqStudent AS INT START WITH 1 INCREMENT BY 1
MINVALUE 1 MAXVALUE 2147483647 NO CYCLE CACHE 50;
```

У наведеному прикладі створюється послідовність dbo.SeqStudent, яка генерує цілі числа, починаючи з 1, із кроком 1. Параметри MINVALUE та MAXVALUE визначають допустимі межі значень, параметр NO CYCLE забороняє повторне використання значень після досягнення верхньої межі, а CACHE 50 дозволяє резервувати частину значень у пам’яті для підвищення продуктивності. У випадках, коли кешування не є бажаним, може використовуватися параметр NOCACHE.

Щоб отримати наступне значення послідовності, застосовується вираз:

```
SELECT NEXT VALUE FOR dbo.SeqStudent;
```

Зазначений вираз може бути використаний як у самостійному запиті, так і безпосередньо в операторі вставлення. Наприклад:

```
INSERT INTO dbo.Student (Id, FullName, GroupName) VALUES (NEXT VALUE
FOR dbo.SeqStudent, N'Петренко Олег', N'ІП-32');
```

Таким чином, механізм SEQUENCE дозволяє отримувати нові значення незалежно від конкретної таблиці й навіть до моменту фактичного додавання запису. Це суттєво розширює можливості його застосування, особливо в багатотабличних або багаторівневих системах.

Створення обмежень.

Без обмежень база даних швидко перетворюється на набір суперечливих записів, у якому можуть з’являтися дублікати, пропущені обов’язкові значення, посилання на неіснуючі об’єкти та інші логічні помилки. Саме тому під час створення таблиць необхідно не лише визначити перелік стовпців і їх типи даних, а й формалізувати правила предметної області за допомогою ключів та обмежень цілісності. Такі механізми дозволяють перенести контроль коректності даних на рівень СУБД і забезпечити єдині правила перевірки для всіх прикладних програм, що працюють з базою даних.

Первинний ключ забезпечує унікальну ідентифікацію кожного запису таблиці. Він не може містити значення NULL, повинен бути унікальним і, як правило, супроводжується створенням індексу. Вибір первинного ключа може ґрунтуватися на природному атрибуті предметної області або на штучному ідентифікаторі. Досить часто в прикладах і практичних рішеннях для первинного ключа додають окремий стовпець з автоматичною генерацією значень. Такий підхід спрощує зв’язування таблиць, однак не скасовує потреби додатково контролювати бізнес-унікальність інших атрибутів, наприклад номер залікової книжки, код дисципліни або електронна адреса.

Первинний ключ при створенні таблиці може визначатись як на рівні окремого стовпця, так і на рівні таблиці. Перший підхід доцільний для простих ключів, другий — для складених ключів, які складаються з кількох стовпців.

Приклад визначення простого первинного ключа на рівні стовпця:

```
StudentID INT CONSTRAINT PK_Students PRIMARY KEY CLUSTERED
```

Приклад визначення складеного первинного ключа на рівні таблиці:

```
CONSTRAINT PK_Enrollments PRIMARY KEY NONCLUSTERED (StudentID,
CourseID)
```

Розглянемо також повний приклад створення таблиці, у якій первинний ключ реалізовано за допомогою стовпця з автоматичною генерацією значень, а бізнес-унікальність номера студентського квитка додатково контролюється окремим обмеженням UNIQUE:

```
CREATE TABLE Students
(
    StudentID INT IDENTITY(1,1)
        CONSTRAINT PK_Students PRIMARY KEY,
    StudentCardNo NVARCHAR(20) NOT NULL
        CONSTRAINT UQ_Students_StudentCardNo UNIQUE,
    FullName NVARCHAR(100) NOT NULL,
    BirthDate DATE NULL
);
```

Зовнішній ключ є механізмом реалізації зв’язку між таблицями. Він задає залежність підлеглої таблиці від батьківської й забезпечує референтну цілісність. Завдяки зовнішньому ключу система не дозволяє додати запис, що посилається на неіснуючий об’єкт, а також контролює поведінку під час оновлення або видалення батьківського запису. В СУБД можуть використовуватися правила ON DELETE CASCADE, SET DEFAULT, SET NULL або NO ACTION. Вибір конкретної реакції повинен бути обґрунтований бізнес-логікою системи, оскільки автоматичне каскадне видалення не завжди є безпечним.

Загальний приклад оголошення зовнішнього ключа:

```
CONSTRAINT FK_Enrollments_Students
    FOREIGN KEY (StudentID)
    REFERENCES Students(StudentID)
    ON DELETE SET NULL ON UPDATE NO ACTION
```

Приклад зовнішнього ключа з каскадним видаленням:

```
CONSTRAINT FK_OrderItems_Orders
    FOREIGN KEY (OrderID)
    REFERENCES Orders(OrderID)
    ON DELETE NO ACTION ON UPDATE SET NULL
```

У наведеному випадку під час видалення замовлення автоматично видалятимуться й усі пов’язані з ним позиції. Такий підхід доцільний лише

тоді, коли дочірні записи не мають самостійного змісту без батьківського об’єкта.

Приклад зовнішнього ключа з встановленням значення NULL:

```
CONSTRAINT FK_Students_Groups
   FOREIGN KEY (GroupID)
   REFERENCES Groups(GroupID)
   ON DELETE SET NULL ON UPDATE NO ACTION
```

Такий варіант можливий лише за умови, що стовпець GroupID допускає значення NULL. Його доцільно застосовувати тоді, коли після видалення батьківського об’єкта дочірній запис може зберігатися без конкретного посилання.

Приклад зовнішнього ключа з установленням значення за замовчуванням:

```
GroupID INT NOT NULL CONSTRAINT DF_Students_GroupID DEFAULT 1,
CONSTRAINT FK_Students_Groups_Default
    FOREIGN KEY (GroupID) REFERENCES Groups(GroupID)
    ON DELETE SET DEFAULT ON UPDATE NO ACTION
```

Окрім ключів, важливу роль відіграють доменні обмеження. Саме вони переводять словесні правила предметної області у формалізований механізм контролю даних на рівні СУБД. До найвживаніших належать NOT NULL, UNIQUE, DEFAULT та CHECK.

Обмеження NOT NULL забороняє невизначені значення там, де участь атрибута є обов’язковою. Його слід використовувати для всіх стовпців, значення яких повинні бути наявними в кожному записі.

```
FullName NVARCHAR(100) NOT NULL,
CreatedAt DATETIME2 NOT NULL
```

Обмеження UNIQUE запобігає дублюванню значень у стовпці або в комбінації стовпців. Воно не замінює первинний ключ, а використовується для контролю додаткових вимог унікальності, зумовлених бізнес-логікою. У SQL Server стовпець з UNIQUE допускає лише одне значення NULL, тому за потреби повної заборони пропусків це обмеження доцільно поєднувати з NOT NULL.

Приклад обмеження UNIQUE для одного стовпця:

```
CONSTRAINT UQ_Students_Email UNIQUE NONCLUSTERED (Email)
```

Приклад складеного обмеження UNIQUE:

```
CONSTRAINT UQ_Timetable_Group_Day_Pair
UNIQUE (GroupID, LessonDate, PairNo)
```

Обмеження CHECK дозволяє явно описати допустимий діапазон значень або логічну умову для стовпця. Воно використовується для реалізації правил, які можна виразити булевим предикатом без звернення до інших таблиць.

```
CONSTRAINT CK_Courses_Credits CHECK (Credits BETWEEN 1 AND 10)
```

Ще один приклад CHECK-обмеження:

```
CONSTRAINT CK_Students_BirthDate
CHECK (BirthDate IS NULL OR BirthDate <= CAST(GETDATE() AS DATE))
```

Обмеження DEFAULT задає значення за замовчуванням і зменшує кількість помилок під час введення даних. Воно спрацьовує в тих випадках, коли під час вставлення нового запису для відповідного стовпця не передано явного значення.

Приклад використання DEFAULT для числового стовпця:

```
CONSTRAINT DF_Courses_Credits DEFAULT 3 FOR Credits
```

Приклад використання DEFAULT для дати створення запису:

```
CreatedAt DATETIME2 CONSTRAINT DF_Students_CreatedAt
DEFAULT SYSDATETIME()
```

Нижче наведено приклад фрагмента таблиці, у якій поєднано кілька різних обмежень:

```
CREATE TABLE Courses
(
    CourseID INT IDENTITY(1,1)
        CONSTRAINT PK_Courses PRIMARY KEY,
    CourseCode NVARCHAR(20) NOT NULL
        CONSTRAINT UQ_Courses_CourseCode UNIQUE,
    Title NVARCHAR(200) NOT NULL,
    Credits INT NOT NULL
        CONSTRAINT DF_Courses_Credits DEFAULT 3
        CONSTRAINT CK_Courses_Credits CHECK (Credits BETWEEN 1 AND 10),
    IsActive BIT NOT NULL
        CONSTRAINT DF_Courses_IsActive DEFAULT 1
);
```

Іменування обмежень за допомогою ключового слова CONSTRAINT є бажаною практикою. Воно спрощує супровід бази даних, оскільки дозволяє легко ідентифікувати потрібне обмеження під час виконання операцій ALTER TABLE, DROP CONSTRAINT або аналізу помилок. Зазвичай у назві відображають тип обмеження, назву таблиці та назву стовпця або логічного правила, наприклад PK_Students, FK_Enrollments_Students, UQ_Students_Email, CK_Courses_Credits, DF_Courses_Credits.

Отже, первинний ключ забезпечує ідентифікацію записів, зовнішній ключ — коректність зв'язків між таблицями, NOT NULL — обов'язковість значень, UNIQUE — унікальність, CHECK — відповідність заданим доменним правилам, а DEFAULT — автоматичне підставлення типових значень.

Зміни структури та видалення об'єктів бази даних (ALTER / DROP)

У процесі експлуатації бази даних структура таблиць не завжди залишається незмінною. У реальних проєктах вимоги до інформаційної системи можуть уточнюватися, доповнюватися або змінюватися, що зумовлює необхідність коригування вже створених таблиць. Основним інструментом для внесення змін до вже існуючих таблиць у більшості реляційних СУБД є оператор ALTER TABLE. Його призначення полягає в тому, щоб дозволити змінювати опис таблиці без необхідності її повного видалення та повторного створення. За допомогою цього оператора можна додавати нові стовпці, змінювати характеристики існуючих стовпців, вилучати непотрібні стовпці, а також додавати або видаляти обмеження цілісності. Отже, ALTER TABLE забезпечує адаптацію фізичної структури таблиці до нових вимог, що виникають під час розвитку інформаційної системи.

Слід зауважити, що модифікація структури таблиці є відповідальною операцією, оскільки вона виконується над об'єктом, який уже може містити реальні дані. Саме тому будь-яка зміна повинна бути узгоджена не лише з новими вимогами до системи, а й з фактичним станом наявної інформації. Наприклад, під час зміни типу даних необхідно переконатися в сумісності старих і нових значень, а під час додавання обмеження NOT NULL — у тому,

що серед уже існуючих записів відсутні NULL-значення у відповідному стовпці. Інакше операція модифікації або завершиться помилкою, або вимагатиме попередньої підготовки даних.

У загальному випадку за допомогою ALTER TABLE виконуються чотири основні групи операцій: додавання стовпців, зміна властивостей стовпців, видалення стовпців, а також додавання і видалення обмежень.

Додавання нового стовпця до таблиці виконується командою:

```
ALTER TABLE table_name
ADD column_name data_type [NULL | NOT NULL] [CONSTRAINT ...];
```

Такий запис дозволяє розширити структуру таблиці новим атрибутом. Наприклад, якщо до таблиці Students необхідно додати стовпець для номера телефону, це можна зробити так:

```
ALTER TABLE Students ADD PhoneNumber NVARCHAR(20) NULL;
```

У наведеному прикладі до таблиці додається новий стовпець PhoneNumber, який допускає невизначені значення. Такий підхід є безпечним для вже існуючих записів, оскільки наявні рядки автоматично отримають значення NULL у новому стовпці.

Якщо ж новий стовпець повинен бути обов'язковим, тобто визначатися як NOT NULL, ситуація ускладнюється. У таблиці, яка вже містить дані, додавання стовпця з вимогою NOT NULL можливе лише за умови, що для всіх наявних рядків буде задано певне значення. Найчастіше для цього одночасно задають значення за замовчуванням. Наприклад:

```
ALTER TABLE Students ADD IsContract BIT NOT NULL
   CONSTRAINT DF_Students_IsContract DEFAULT 0;
```

У цьому випадку новий стовпець IsContract додається як обов'язковий, а для вже існуючих і нових записів автоматично підставляється значення 0, якщо користувач не задав його явно. Таким чином забезпечується узгодженість структури таблиці з наявними даними.

Зміна вже існуючого стовпця виконується за допомогою конструкції ALTER COLUMN:

```
ALTER TABLE table_name
ALTER COLUMN column_name new_data_type [NULL | NOT NULL];
```

Цей варіант використовується в тих випадках, коли необхідно змінити тип даних, довжину символьного поля або дозволеність значень NULL. Наприклад, якщо потрібно збільшити довжину стовпця FullName, можна виконати таку команду:

```
ALTER TABLE Students ALTER COLUMN FullName NVARCHAR(150) NOT NULL;
```

Однак під час виконання таких операцій слід перевіряти, чи є зміна сумісною з уже наявними значеннями. Наприклад, якщо здійснюється перехід від NVARCHAR(150) до NVARCHAR(50), можливе обрізання даних або помилка виконання. Аналогічно, якщо стовпець переводиться з режиму NULL у режим NOT NULL, то серед наявних записів не повинно бути жодного рядка з відсутнім значенням. Саме питання переходу до NOT NULL потребує особливої уваги. Якщо в таблиці вже є записи, у яких змінюваний стовпець містить NULL, то команда ALTER COLUMN ... NOT NULL не буде виконана.

Видалення стовпця виконується за допомогою команди:

```
ALTER TABLE table_name
DROP COLUMN column_name;
```

Наприклад:

```
ALTER TABLE Students DROP COLUMN PhoneNumber;
```

Слід мати на увазі, що видалення стовпця призводить до втрати всіх даних, які в ньому зберігалися. Крім того, така операція може бути заборонена, якщо стовпець використовується в обмеженнях, індексах, представленнях, тригерах, процедурах або інших об’єктах бази даних. Отже, перед видаленням стовпця необхідно переконатися, що він не бере участі в інших структурних залежностях.

Окрім стовпців, оператор ALTER TABLE дозволяє модифікувати обмеження цілісності. Саме через обмеження у структурі таблиці формалізуються правила предметної області, тому можливість їх додавання або видалення є суттєвою складовою супроводу бази даних. Для додавання обмеження використовується конструкція:

```
ALTER TABLE table_name ADD CONSTRAINT constraint_name ...;
```

Наприклад, для заборони дублювання адрес електронної пошти можна додати обмеження унікальності:

```
ALTER TABLE Students ADD CONSTRAINT UQ_Students_Email UNIQUE (Email);
```

Це означає, що від моменту створення такого обмеження система не дозволятиме вставляти або оновлювати записи так, щоб у стовпці Email виникли повторювані значення.

Аналогічно можуть додаватися CHECK-обмеження, які задають логічні умови допустимості даних. Наприклад:

```
ALTER TABLE Courses
ADD CONSTRAINT CK_Courses_Credits CHECK (Credits BETWEEN 1 AND 10);
```

У цьому випадку гарантується, що значення в стовпці Credits будуть належати заданому діапазону. Таким чином, додавання обмежень через ALTER TABLE дає змогу поступово посилювати формалізований контроль даних у вже існуючій таблиці.

Видалення обмеження виконується через DROP CONSTRAINT:

```
ALTER TABLE table_name
DROP CONSTRAINT constraint_name;
```

Наприклад:

```
ALTER TABLE Students DROP CONSTRAINT UQ_Students_Email;
```

Ця команда скасовує дію відповідного обмеження. Слід підкреслити, що для видалення обмеження необхідно знати його ім'я, тому іменування обмежень під час створення таблиць є важливим не лише з погляду читабельності схеми, а й для подальшого супроводу бази даних.

У MSSQL Server під час додавання деяких обмежень може використовуватися конструкція WITH CHECK або WITH NOCHECK. Ці параметри визначають, чи потрібно перевіряти вже наявні дані таблиці на відповідність новому обмеженню. Якщо використовується WITH CHECK, то всі існуючі записи повинні задовольняти нову умову. Якщо ж застосовано WITH NOCHECK, перевірка старих даних не виконується, і обмеження починає діяти лише для нових або змінених записів. Наприклад:

```
ALTER TABLE Courses
WITH NOCHECK ADD CONSTRAINT CK_Courses_Hours CHECK (Hours >= 0);
```

Такий підхід інколи є корисним під час поетапного впровадження нових правил у вже наповнену базу даних, однак його слід застосовувати обережно. Якщо обмеження додається без перевірки старих даних, у таблиці можуть залишитися записи, які фактично не відповідають новим правилам. Тому WITH NOCHECK слід розглядати як технічний компроміс, а не як стандартний спосіб супроводу схеми.

Поряд з оператором ALTER TABLE у практиці модифікації структури бази даних важливу роль відіграє оператор DROP. Його призначення полягає у видаленні об’єктів бази даних, зокрема таблиць, обмежень, представлень, процедур, функцій тощо. Якщо ALTER TABLE змінює структуру наявного об’єкта, то DROP повністю вилучає його з бази даних. Наприклад:

```
DROP TABLE OldStudents;
```

Застосування оператора DROP потребує особливої обережності, оскільки його виконання, як правило, супроводжується безповоротною втратою структури об’єкта та пов’язаних із ним даних. Саме тому видалення таблиць або інших об’єктів повинно бути чітко обґрунтованим і, за можливості, передувати резервному копіюванню.

Слід підкреслити, що модифікація структури таблиць у реальних інформаційних системах зазвичай виконується не вручну, а в межах контрольованих сценаріїв оновлення схеми бази даних. Такі зміни часто оформлюються у вигляді окремих SQL-скриптів або міграцій, які можна виконувати послідовно в різних середовищах. Це дає змогу забезпечити відтворюваність змін, контроль версій схеми та узгодженість між розробницьким, тестовим і продуктивним середовищами.

Створення таблиць не може бути якісним без врахування принципів нормалізації. Якщо дані дублюються в кількох місцях, у таблицях з’являються аномалії вставки, оновлення та видалення. Наприклад, якщо назву викладача зберігати у кожному рядку журналу оцінок, будь-яка зміна прізвища вимагатиме масового оновлення багатьох записів. Нормалізація дозволяє розділити такі дані на окремі логічні сутності та зв’язати їх ключами. Отже,

лабораторна робота № 2 фактично перевіряє, наскільки коректно була виконана попередня робота з ER-моделювання, адже саме тут стають помітними помилки концептуального проєктування [1].

Хоча основною метою лабораторної роботи є створення бази даних та організація доступу, на етапі фізичного проєктування доцільно вже розуміти роль індексів. Індекс є спеціальною структурою, що прискорює пошук, сортування та з'єднання даних. Первинний ключ зазвичай автоматично супроводжується індексом, проте для часто використовуваних зовнішніх ключів і пошукових стовпців також нерідко створюються додаткові некластеризовані індекси. Водночас надмірна індексація збільшує вартість вставки та оновлення записів. Отже, індекси є не просто технічною деталлю, а важливим компромісом між швидкістю читання і вартістю модифікації даних [2].

Керування обліковими записами та ролями

У процесі проєктування та експлуатації баз даних важливого значення набуває не лише правильна організація структури даних, а й забезпечення контрольованого доступу до них. У реальних інформаційних системах різні категорії користувачів повинні мати різний рівень повноважень: одні можуть лише переглядати дані, інші — додавати та змінювати записи, а окремі адміністратори — створювати, змінювати або видаляти об'єкти бази даних. Саме тому в сучасних системах управління базами даних реалізовано механізми автентифікації, авторизації, розмежування прав доступу та централізованого керування дозволами.

Слід зауважити, що права доступу в СУБД задаються не абстрактно, а щодо конкретних захищених об'єктів — securables. До таких об'єктів можуть належати сервер, база даних, схема, таблиця, представлення, збережена процедура, функція, тип даних та інші елементи схеми. Отже, система керування правами доступу має ієрархічний характер: частина дозволів надається на серверному рівні, частина — на рівні бази даних, а частина — на

рівні окремих об'єктів. Відповідно, повноваження користувача визначаються не лише видом дії, а й тим, до якого саме об'єкта ця дія застосовується.

У SQL Server необхідно розрізняти логін, користувача бази даних і роль. Логін забезпечує автентифікацію на рівні сервера. Користувач створюється всередині конкретної бази даних і є представником відповідного логіна в її межах. Роль — це логічна група повноважень, до якої можуть бути включені користувачі. Така багаторівнева модель дозволяє відокремити питання входу в систему від питання доступу до конкретних об'єктів бази даних. Після створення користувача рекомендується додавати його до ролі та надавати права саме ролі, а не окремій особі, що спрощує адміністрування [3], [4].

Ролі можуть бути системними або користувацькими. Надання прав реалізується через GRANT, явна заборона — через DENY, а скасування — через REVOKE. Важливо розуміти, що модель безпеки будується за принципом мінімально необхідних повноважень: користувач повинен мати лише ті права, які необхідні йому для виконання конкретних дій, і не більше [3], [4], [5].

У MSSQL Server розрізняють рівень сервера (LOGIN) та рівень бази даних (USER). Користувач бази даних прив'язується до логіна. Права зазвичай надають не безпосередньо користувачам, а ролям. Об'єкти організовують у схеми (SCHEMA), що спрощує керування правами на рівні контейнера. Додатково діє механізм ownership chaining, який впливає на перевірку прав при виклику об'єктів.

Доступ в SQL Server налаштовують поетапно: спочатку створюють LOGIN (на сервері), потім USER (у конкретній базі), після цього додають користувача до ROLE і видають права ролі на рівні SCHEMA або окремих таблиць. Цей підхід спрощує адміністрування — достатньо керувати правами ролей.

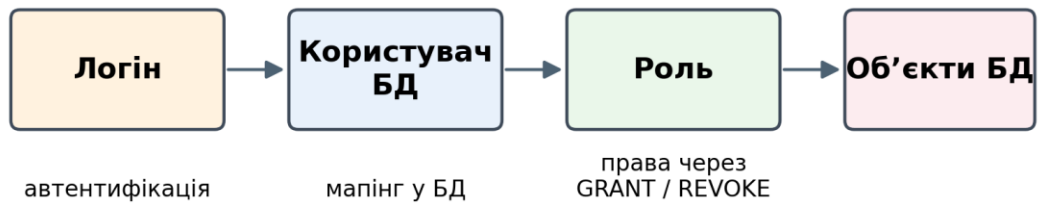


Рисунок 3 – Логічна схема організації доступу: логін, користувач бази даних, роль і об'єкти БД

Першим етапом організації доступу є створення логіну. Логін є обліковим записом на рівні сервера і використовується для автентифікації користувача. Іншими словами, логін визначає, чи має суб'єкт право підключитися до сервера баз даних. У MSSQL Server логіни можуть бути пов'язані як із Windows-автентифікацією, так і з власною SQL-автентифікацією. Для SQL-автентифікації під час створення логіну необхідно явно задати пароль.

Приклад створення логіну має такий вигляд:

```
CREATE LOGIN login_name
WITH PASSWORD = 'Pa$$w0rd',
     CHECK_POLICY = ON,
     CHECK_EXPIRATION = ON,
     DEFAULT_DATABASE = db_name,
     DEFAULT_LANGUAGE = us_english;
```

У наведеній конструкції PASSWORD задає пароль для SQL-автентифікації, CHECK_POLICY визначає, чи застосовуються до пароля правила складності Windows, CHECK_EXPIRATION вказує, чи контролюється термін дії пароля, DEFAULT_DATABASE визначає базу даних, яка відкриватиметься за замовчуванням після підключення, а DEFAULT_LANGUAGE задає мовні параметри сесії. Таким чином, логін є засобом входу на сервер, але сам по собі ще не надає права працювати з конкретною базою даних.

Наступним етапом є створення користувача бази даних. Користувач існує на рівні конкретної бази даних і пов'язується з уже створеним логіном. Саме користувач бази даних бере участь у схемі розмежування дозволів усередині конкретної БД. У MSSQL Server створення користувача виконується так:

```
CREATE USER user_name
FOR LOGIN login_name
WITH DEFAULT_SCHEMA = schema_name;
```

В наведеній конструкції user_name — ім'я користувача в межах бази даних, login_name — серверний логін, з яким він пов'язаний, а DEFAULT_SCHEMA — схема за замовчуванням, у межах якої система буде шукати об'єкти без явно заданого імені схеми. Отже, логін забезпечує доступ до сервера, а користувач — доступ до конкретної бази даних.

Однак у практичних системах надання прав безпосередньо кожному користувачеві не є найзручнішим підходом. Саме тому в СУБД застосовується механізм ролей. У MSSQL Server роль створюється наступним чином:

```
CREATE ROLE role_name AUTHORIZATION dbo;
```

Додавання користувача до ролі виконується командою:

```
ALTER ROLE role_name ADD MEMBER user_name;
```

Після створення логінів, користувачів і ролей виконується безпосереднє надання прав доступу. Для цього в SQL Server використовується оператор GRANT, загальна форма якого має вигляд:

```
GRANT <permission> [ (column_list) ]
ON <securable_class>::<securable_name>
TO <principal> [ , ... ]
[ WITH GRANT OPTION ]
[ AS <grantor> ];
```

У цій конструкції <permission> задає тип дозволеної операції, наприклад SELECT, INSERT, UPDATE, DELETE, EXECUTE, ALTER, CONTROL, VIEW DEFINITION; (column_list) використовується у випадках, коли право надається лише для окремих стовпців; <securable_class>::<securable_name> визначає тип захищеного об'єкта і його ім'я; <principal> — суб'єкт, якому надається право, тобто користувач, роль або логін. Параметр WITH GRANT

OPTION дозволяє не лише виконувати дію, а й передавати це право іншим суб'єктам.

Для скасування дозволів використовується оператор REVOKE. Його загальна форма має вигляд:

```
REVOKE [ GRANT OPTION FOR ] <permission> [ (column_list) ]
ON <securable>
FROM <principal> [ , ... ]
[ CASCADE ]
[ AS <grantor> ];
```

Оператор REVOKE не накладає явної заборони, а лише скасовує попередньо наданий дозвіл або право делегування.

Оператор DENY використовується для встановлення явної заборони. Його загальна форма є подібною до GRANT:

```
DENY <permission> [ (column_list) ]
ON <securable>
TO <principal> [ , ... ]
[ AS <grantor> ];
```

Окрему увагу слід звернути на класи захищених об'єктів. У SQL Server дозволи можуть надаватися, зокрема, на рівні:

- SCHEMA — для всієї схеми;
- OBJECT — для конкретного об'єкта, наприклад таблиці чи процедури;
- DATABASE — для бази даних у цілому;
- TYPE — для типу даних.

Це означає, що керування доступом може бути як дуже детальним, так і узагальненим. Такий підхід спрощує адміністрування тоді, коли роль повинна мати однаковий тип доступу до великої кількості однотипних об'єктів.

Слід також підкреслити, що ефективне керування доступом ґрунтується на принципі мінімальних повноважень. Це означає, що кожному користувачеві або ролі слід надавати лише ті права, які є необхідними для виконання їхніх функцій, і не більше. Надмірно широкі дозволи збільшують ризик випадкового пошкодження даних, порушення безпеки або несанкціонованого доступу. Саме тому в практиці проєктування систем

безпеки рекомендується надавати права ролям, використовувати GRANT як основний засіб призначення повноважень, а DENY — лише у виняткових випадках.

Постановка задачі лабораторної роботи № 2

При виконанні лабораторної роботи необхідно виконати наступні дії.

1) Розробити SQL-скрипти для:

a. створення БД згідно з розробленою в роботі №1 ER-моделлю;

b. створення таблиць в БД засобами мови SQL. Передбачити наявність обмежень для підтримки цілісності та коректності даних, котрі зберігаються та вводяться;

c. встановлення зв'язків між таблицями засобами мови SQL;

d. зміни в структурах таблиць, обмежень засобами мови SQL (до 10 різних за суттю запитів для декількох таблиць (використати DDL-команди SQL));

e. видалення окремих елементів таблиць/обмежень або самих таблиць засобами мови SQL (до 10 різних за суттю запитів (використати DDL-команди SQL));

f. визначити декілька (2-3) типів користувачів, котрі будуть працювати з розробленою базою даних. Для кожного користувача визначити набір привілеїв, котрі він буде мати;

g. для визначених типів користувачів створити відповідні ролі та наділити їх необхідними привілеями;

h. створити по одному користувачу в базі даних для кожного типу та присвоїти їм відповідні ролі.

2) Згенерувати схему бази даних засобами СУБД.

3) Імпортувати дані в створену БД з використанням засобів СУБД, а не DML SQL.

Вимоги до оформлення звіту з лабораторної роботи

Звіт повинен містити наступні складові частини:

- титульний лист;
- назву та мету роботи;
- варіант;
- SQL скрипти згідно завдання до лабораторної роботи;
- схема бази даних згенеровану засобами СУБД;
- висновки.

Контрольні запитання

1) Що таке база даних і чим відрізняються концептуальна, логічна та фізична моделі?

2) Що таке схема (SCHEMA) в SQL Server і навіщо її створювати, якщо вже є dbo?

3) Поясніть різницю між таблицею, рядком, стовпцем і типом даних.

4) Які критерії вибору типів даних (пам'ять, продуктивність, коректність)? Наведіть приклади.

5) Наведіть базовий синтаксис CREATE DATABASE і поясніть параметри COLLATE, ON/LOG ON.

6) Для чого використовується CREATE SCHEMA … AUTHORIZATION …?

7) Наведіть загальний шаблон CREATE TABLE та поясніть елементи: NULL/NOT NULL, DEFAULT, IDENTITY, COLLATE.

8) Різниця між обмеженнями рівня стовпця і рівня таблиці. Коли що застосовувати?

9) Для чого потрібні ALTER TABLE … ADD/ALTER/DROP COLUMN і на що звернути увагу при переході до NOT NULL?

10) Що таке обчислюваний стовпець (AS (...))?

11) Поясніть різницю між DROP TABLE, TRUNCATE TABLE і DELETE. (Який з них DDL? Які наслідки?)

12) Що таке PRIMARY KEY? Чи може він бути складеним? Чи допускає NULL?

13) Різниця між PRIMARY KEY та UNIQUE. Чому не варто опиратися лише на UNIQUE як ідентифікатор?

14) Що таке FOREIGN KEY? Поясніть опції ON DELETE/UPDATE { NO ACTION | CASCADE | SET NULL | SET DEFAULT }.

15) Що перевіряє CHECK-обмеження? Наведіть приклади валідних предикатів.

16) Для чого використовується DEFAULT і чому важливо іменувати обмеження?

17) Як перевірити наявні дані при додаванні нового обмеження? Різниця WITH CHECK і WITH NOCHECK.

18) Різниця між IDENTITY(seed, increment) та SEQUENCE. Переваги/недоліки кожного підходу.

19) Для чого використовують DBCC CHECKIDENT та SET IDENTITY_INSERT ON?

20) Поясніть різницю між LOGIN (рівень сервера) та USER (рівень бази). Як вони пов’язані?

21) Навіщо використовувати ролі? Чому не рекомендують призначати права напряму користувачам?

22) Які основні оператори керування доступом у SQL Server і чим вони відрізняються: GRANT, REVOKE, DENY?

23) На яких рівнях захищуваних об’єктів (securables) можуть надаватись права (SERVER/DATABASE/SCHEMA/OBJECT)?

24) Чим відрізняється видача прав на рівні SCHEMA:: від видачі на окремі таблиці?

25) Як перевірити фактичні права користувача без виходу з вашого сеансу?

26) Поясніть різницю між LOGIN та USER у SQL Server. Як вони пов’язані?

27) Чим відрізняються GRANT, REVOKE та DENY? У яких випадках застосовувати кожну команду?

28) Які дії виконуються при ON DELETE CASCADE? Коли їх доцільно застосовувати?

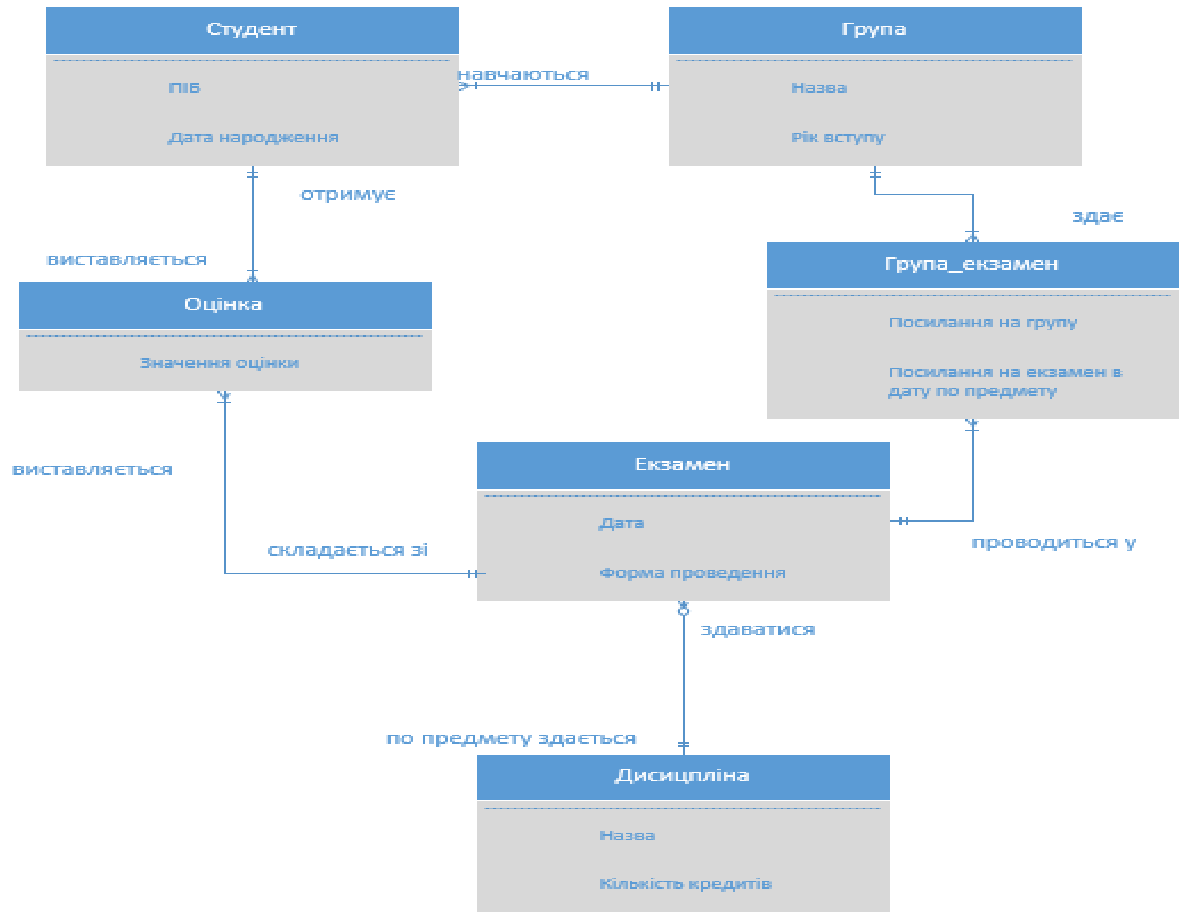
Приклад виконання завдання лабораторної роботи № 2

Розглянемо процес виконання лабораторної роботи для наведеного нижче предметного середовища.

Предметне середовище «Розклад екзаменів».

Система, котра проєктується, повинна зберігати інформацію про студентів, їх групи, предмети, екзамени (по якому предмету, якою групою та коли здається). Для кожної групи повинна існувати можливість отримати розклад екзаменів кожної групи, списки груп та отримані оцінки. Окрім того, один й той самий екзамен можуть здавати одразу декілька груп.

В процесі виконання першої лабораторної роботи була розроблена наступна ER-модель:



Для виконання поточної лабораторної роботи обрано СУБД MSSQL Server.

1) Розробити SQL-скрипти для:

a. створення БД згідно з розробленою в роботі №1 ER-моделлю;

```
IF EXISTS (
  SELECT *
      FROM sys.databases
      WHERE name = N'MyEdu'
)
   DROP DATABASE MyEdu
GO

CREATE DATABASE MyEdu
ON PRIMARY
      ( NAME = MyEdu_Data,
        FILENAME = N'C:\Kate\DB\25_26\lab\MyEdu_Data.mdf'
          )
LOG ON
      ( NAME = MyEdu_Log,
        FILENAME = N'C:\Kate\DB\25_26\lab\MyEdu_Log.ldf'
          )
GO
```

b. створення таблиць в БД засобами мови SQL. Передбачити наявність обмежень для підтримки цілісності та коректності даних, котрі зберігаються та вводяться;

```
-- Таблиця [Group]
CREATE TABLE [Group]
(
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[NAIM] [varchar](50) NOT NULL,
	[YEAR_ENTRANCE] [int]
	CONSTRAINT [PK_Group] PRIMARY KEY CLUSTERED ([ID])
)
-- Встановити обмеження
ALTER TABLE [dbo].[Group]  WITH CHECK ADD  CONSTRAINT [CK_GroupNaim] CHECK
(([NAIM]<>''))
GO
ALTER TABLE [dbo].[Group]  WITH CHECK ADD  CONSTRAINT [CK_GroupYEAR] CHECK
(([YEAR_ENTRANCE]>=1950 and [YEAR_ENTRANCE]<=2099))
GO

-- Таблиця [Student]
CREATE TABLE [dbo].[Student](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[Naim] [varchar](50) NOT NULL CONSTRAINT CK_StudentNaim CHECK ([Naim]
<> ''),
	[Surname] [varchar](50) NOT NULL CONSTRAINT CK_StudentSurnaim CHECK
([Surname] <> ''),
	[DATE_BIRTH] [datetime] NULL,
	[ID_Group] [int] NULL,
	CONSTRAINT [PK_Student] PRIMARY KEY CLUSTERED ([ID])
)

-- Таблиця [Subject]
CREATE TABLE [dbo].[Subject](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[Naim] [varchar](50) NOT NULL CONSTRAINT CK_SubjectNaim CHECK ([Naim]
<> ''),
	[NUM_CREDIT] [int] NOT NULL DEFAULT 4,
	CONSTRAINT [PK_Subject] PRIMARY KEY CLUSTERED ([ID])
)
-- Встановити обмеження
ALTER TABLE [dbo].[Subject] WITH CHECK ADD  CONSTRAINT
[CK_SubjectNumCreditPositive] CHECK (([NUM_CREDIT]>0))
GO

-- Таблиця [Exam]
CREATE TABLE [dbo].[Exam](
	[ID] [int] IDENTITY(1,1) NOT NULL,
	[DATE_EXAM] [datetime2](0) NULL,
	[ROOM_EXAM] [varchar](10) NOT NULL DEFAULT 'н/д',
	[FORM_EXAM] [varchar](50) NOT NULL CONSTRAINT CK_Exams_FormExam CHECK
(FORM_EXAM IN (N'очна', N'дистанційна')),
	[ID_SUBJECT] [int] NULL,
```

```
	CONSTRAINT [PK_Exam] PRIMARY KEY CLUSTERED ([ID])
)
-- Унікальність: один і той самий екзамен з дисципліни у той самий час в одній аудиторії
ALTER TABLE [dbo].[Exam] ADD CONSTRAINT UQ_Exams_Subject_Time_Room UNIQUE (ID_SUBJECT, DATE_EXAM, ROOM_EXAM)
GO

-- Таблиця [Grades]
CREATE TABLE [dbo].[Grades](
	[ID_Student] [int] NOT NULL,
	[ID_Exam] [int] NOT NULL,
	[EXAM_GRADE] [int] NOT NULL DEFAULT 0,
	CONSTRAINT [PK_GRADES] PRIMARY KEY CLUSTERED ([ID_Student],[ID_Exam])
)
-- Встановити обмеження
ALTER TABLE [dbo].[Grades] WITH CHECK ADD  CONSTRAINT [CK_Grades_GradePositive] CHECK (([EXAM_GRADE] between 0 and 100))
GO

-- Таблиця [GroupSchedule]
CREATE TABLE [dbo].[GroupSchedule](
	[ID_Group] [int] NOT NULL,
	[ID_Exam] [int] NOT NULL,
	[COMM] [varchar](120) NOT NULL DEFAULT '',
	CONSTRAINT [PK_GroupSchedule] PRIMARY KEY CLUSTERED ([ID_Group],[ID_Exam])
)
GO
```

c. встановлення зв'язків між таблицями засобами мови SQL;

```
-- Встановити зв'язок  FK між таблицями [Student] та [Group]
-- Зв'язок: [Student].[ID_Group] -> [Group].[ID]
ALTER TABLE [dbo].[Student]  WITH CHECK ADD CONSTRAINT [FK_Student_Group] FOREIGN KEY([ID_Group])
REFERENCES [dbo].[Group] ([ID])
GO

-- Встановити зв'язок  FK між таблицями [Exam] та [Subject]
-- Зв'язок: [Exam].[ID_Subject] -> [Subject].[ID]
ALTER TABLE [dbo].[EXAM]  WITH CHECK ADD CONSTRAINT [FK_Exam_Subject] FOREIGN KEY([ID_Subject])
REFERENCES [dbo].[Subject] ([ID])
GO

-- Встановити зв'язок  FK між таблицями [Student] та [Grade]
-- Зв'язок: [Grades].[ID_Student] -> [Student].[ID]
ALTER TABLE [dbo].[Grades]  WITH CHECK ADD CONSTRAINT [FK_Grades_Student] FOREIGN KEY([ID_Student])
REFERENCES [dbo].[Student] ([ID])
GO

-- Встановити зв'язок  FK між таблицями [Exam] та [Grade]
-- Зв'язок: [Grades].[ID_Exam] -> [Exam].[ID]
ALTER TABLE [dbo].[Grades]  WITH CHECK ADD CONSTRAINT [FK_Grades_Exam] FOREIGN KEY([ID_Exam])
```

```
REFERENCES [dbo].[Exam] ([ID])
GO

-- Встановити зв'язок  FK між таблицями [GroupSchedule] та [Group]
-- Зв'язок: [GroupSchedule].[ID_Group] -> [Group].[ID]
ALTER TABLE [dbo].[GroupSchedule]  WITH CHECK ADD CONSTRAINT
[FK_GroupSchedule_Group] FOREIGN KEY([ID_Group])
REFERENCES [dbo].[Group] ([ID])
GO

-- Встановити зв'язок  FK між таблицями [GroupSchedule] та [Exam]
-- Зв'язок: [GroupSchedule].[ID_Exam] -> [Exam].[ID]
ALTER TABLE [dbo].[GroupSchedule]  WITH CHECK ADD CONSTRAINT
[FK_GroupSchedule_Exam] FOREIGN KEY([ID_Exam])
REFERENCES [dbo].[Exam] ([ID])
GO
```

d. зміни в структурах таблиць, обмежень засобами мови SQL (до 10 різних за суттю запитів для декількох таблиць (використати DDL-команди SQL));

```
-- 1) Student: додати телефон з NOT NULL + значення за замовченням
IF COL_LENGTH('dbo.Student', 'Phone') IS NULL
BEGIN
    ALTER TABLE dbo.Student ADD Phone VARCHAR(30) NOT NULL CONSTRAINT
DF_Student_Phone DEFAULT('');
END
GO

--2) Student: змінити довжину та обов’язковість заповнення прізвища
ALTER TABLE dbo.Student
  ALTER COLUMN Surname VARCHAR(70) NOT NULL;
GO

-- 3) Student: додати обчислюваний стовпець AgeYears
IF COL_LENGTH('dbo.Student', 'AgeYears') IS NULL
BEGIN
    ALTER TABLE dbo.Student ADD AgeYears AS (DATEDIFF(YEAR, DATE_BIRTH,
GETDATE()));
END;
GO

-- 4) Student: додати перевірку, що дата народження менше поточної дати
IF OBJECT_ID('CK_Student_Birth_NotFuture', 'C') IS NULL
BEGIN
    ALTER TABLE dbo.Student WITH CHECK ADD CONSTRAINT
CK_Student_Birth_NotFuture CHECK (DATE_BIRTH IS NULL OR DATE_BIRTH <=
SYSUTCDATETIME());
END
GO

-- 5) Subject: додати унікальність назви дисципліни
IF OBJECT_ID('UQ_Subject_Naim', 'UQ') IS NULL
BEGIN
```

```
    ALTER TABLE dbo.[Subject] ADD CONSTRAINT UQ_Subject_Naim UNIQUE (Naim);
END
GO

-- 6) Subject: оновити перевірку кредитів, що кількість кредитів не може бути
більша за 30
IF OBJECT_ID('CK_Subject_NumCredit_Range', 'C') IS NULL
BEGIN
    ALTER TABLE dbo.[Subject] WITH CHECK ADD CONSTRAINT
CK_Subject_NumCredit_Range CHECK (NUM_CREDIT > 0 AND NUM_CREDIT <= 30);
END
GO

-- 7) Exam: додати тривалість екзамену
IF COL_LENGTH('dbo.Exam', 'DurationMinutes') IS NULL
BEGIN
    ALTER TABLE dbo.Exam
      ADD DurationMinutes INT NOT NULL CONSTRAINT DF_Exam_Duration
DEFAULT(90);
END
GO

-- 8) Exam: змінити довжину поля Room
ALTER TABLE dbo.Exam  ALTER COLUMN ROOM_EXAM VARCHAR(20) NOT NULL;
GO

-- 9) Exam: додати перевірку номеру аудиторії (допустимо
цифри/літери/дефіс/пропуски)
IF OBJECT_ID('CK_Exam_Room_Format', 'C') IS NULL
BEGIN
    ALTER TABLE dbo.Exam WITH CHECK ADD CONSTRAINT CK_Exam_Room_Format CHECK
(ROOM_EXAM LIKE '%[0-9A-Za-zА-Яа-яЁёЇїІіЄєҐґ\- ]%');
END
GO

-- 10) Student: додати Email студента
IF COL_LENGTH('dbo.Student', 'Email') IS NULL
BEGIN
    ALTER TABLE dbo.Student ADD Email NVARCHAR(120) NULL;
END
GO
```

e. видалення окремих елементів таблиць/обмежень або самих таблиць засобами мови SQL (до 10 різних за суттю запитів (використати DDL-команди SQL));

```
-- 1) Exam: прибрати унікальність по (ID_SUBJECT, DATE_EXAM, ROOM_EXAM)
IF OBJECT_ID('UQ_Exams_Subject_Time_Room', 'UQ') IS NOT NULL
BEGIN
```

```
    ALTER TABLE dbo.Exam DROP CONSTRAINT UQ_Exams_Subject_Time_Room;
END
GO

 -- 2) Student: видалити стовпець Phone
 IF OBJECTPROPERTY(OBJECT_ID(N'DF_Student_Phone'), 'IsDefaultCnst') = 1
BEGIN
    ALTER TABLE dbo.Student DROP CONSTRAINT DF_Student_Phone;
END
IF COL_LENGTH('dbo.Student', 'Phone') IS NOT NULL
BEGIN
    ALTER TABLE dbo.Student DROP COLUMN Phone;
END
GO

-- 3) Exam: видалити DEFAULT-constraint DurationMinutes
IF OBJECTPROPERTY(OBJECT_ID(N'DF_Exam_Duration'), 'IsDefaultCnst') = 1
BEGIN
    ALTER TABLE dbo.Exam DROP CONSTRAINT DF_Exam_Duration;
END
GO

-- 4) Exam: видалити стовпець DurationMinutes
IF COL_LENGTH('dbo.Exam', 'DurationMinutes') IS NOT NULL
BEGIN
    ALTER TABLE dbo.Exam DROP COLUMN DurationMinutes;
END
GO

-- 5) Exam: прибрати контроль формату введення аудиторыъ
IF OBJECT_ID('CK_Exam_Room_Format', 'C') IS NOT NULL
BEGIN
    ALTER TABLE dbo.Exam DROP CONSTRAINT CK_Exam_Room_Format;
END
GO

-- 6) GroupSchedule: вибалити таблицю
IF OBJECT_ID('dbo.GroupSchedule', 'U') IS NOT NULL
BEGIN
    IF OBJECT_ID('FK_GroupSchedule_Group', 'F') IS NOT NULL
        ALTER TABLE dbo.GroupSchedule DROP CONSTRAINT FK_GroupSchedule_Group;

    IF OBJECT_ID('FK_GroupSchedule_Exam', 'F') IS NOT NULL
        ALTER TABLE dbo.GroupSchedule DROP CONSTRAINT FK_GroupSchedule_Exam;

    DROP TABLE dbo.GroupSchedule;
END
GO


-- 7) Group: зняти перевірку діапазону року вступу
IF OBJECT_ID('CK_GroupYEAR', 'C') IS NOT NULL
BEGIN
    ALTER TABLE dbo.[Group] DROP CONSTRAINT CK_GroupYEAR;
END
GO
```

```
-- 8) Subject: прибрати унікальність назви дисципліни
IF OBJECT_ID('UQ_Subject_Naim', 'UQ') IS NOT NULL
BEGIN
    ALTER TABLE dbo.[Subject] DROP CONSTRAINT UQ_Subject_Naim;
END
GO

--  9) Student: видалити стовпець Email
IF COL_LENGTH('dbo.Student', 'Email') IS NOT NULL
BEGIN
    ALTER TABLE dbo.Student DROP COLUMN Email;
END
GO

-- 10) Exam: прибрати контроль форми екзамену
IF OBJECT_ID('CK_Exams_FormExam', 'C') IS NOT NULL
BEGIN
    ALTER TABLE dbo.Exam DROP CONSTRAINT CK_Exams_FormExam;
END
GO
```

f. визначити декілька (2-3) типів користувачів, котрі будуть працювати з розробленою базою даних. Для кожного користувача визначити набір привілеїв, котрі він буде мати;

Визначимо наступні типи користувачів:

- **Студент (role_reader)** — має права лише читання всіх таблиць (для перегляду довідників, розкладів, оцінок).
- Викладач - ексаменатор **(role_grader)** — може читати дані зі всіх таблиць, **вносити/змінювати оцінки** в dbo.Grades (тільки стовпець EXAM_GRADE), видаляти — заборонено.
- Працівник деканату **(role_exam_manager)** — читає дані зі всіх таблиць, **керує екзаменами та розкладами**: повний CRUD на dbo.Exam, dbo.GroupSchedule.

g. для визначених типів користувачів створити відповідні ролі та наділити їх необхідними привілеями;

```
-- !!!! role_reader: тільки читання всіх об'єктів у схемі dbo
```

```
GRANT SELECT ON SCHEMA::dbo TO role_reader;
GO

-- !!!! role_grader: читання всього + внесення/оновлення оцінок
-- тільки читання всіх об'єктів у схемі dbo
GRANT SELECT ON SCHEMA::dbo TO role_grader;
-- Додавання нових оцінок
GRANT INSERT ON OBJECT::dbo.Grades TO role_grader;
-- Оновлення лише колонки EXAM_GRADE
GRANT UPDATE (EXAM_GRADE) ON OBJECT::dbo.Grades TO role_grader;
-- На всяк випадок ЯВНО забороняємо видалення оцінок
DENY DELETE ON OBJECT::dbo.Grades TO role_grader;
GO

-- !!!! role_exam_manager: читає все + керує Exam і GroupSchedule
GRANT SELECT ON SCHEMA::dbo TO role_exam_manager;
-- Повний CRUD на Exam і GroupSchedule
GRANT SELECT, INSERT, UPDATE, DELETE ON OBJECT::dbo.Exam          TO
role_exam_manager;
GRANT SELECT, INSERT, UPDATE, DELETE ON OBJECT::dbo.GroupSchedule TO
role_exam_manager;
GO
```

h. створити по одному користувачу в базі даних для кожного типу та присвоїти їм відповідні ролі.

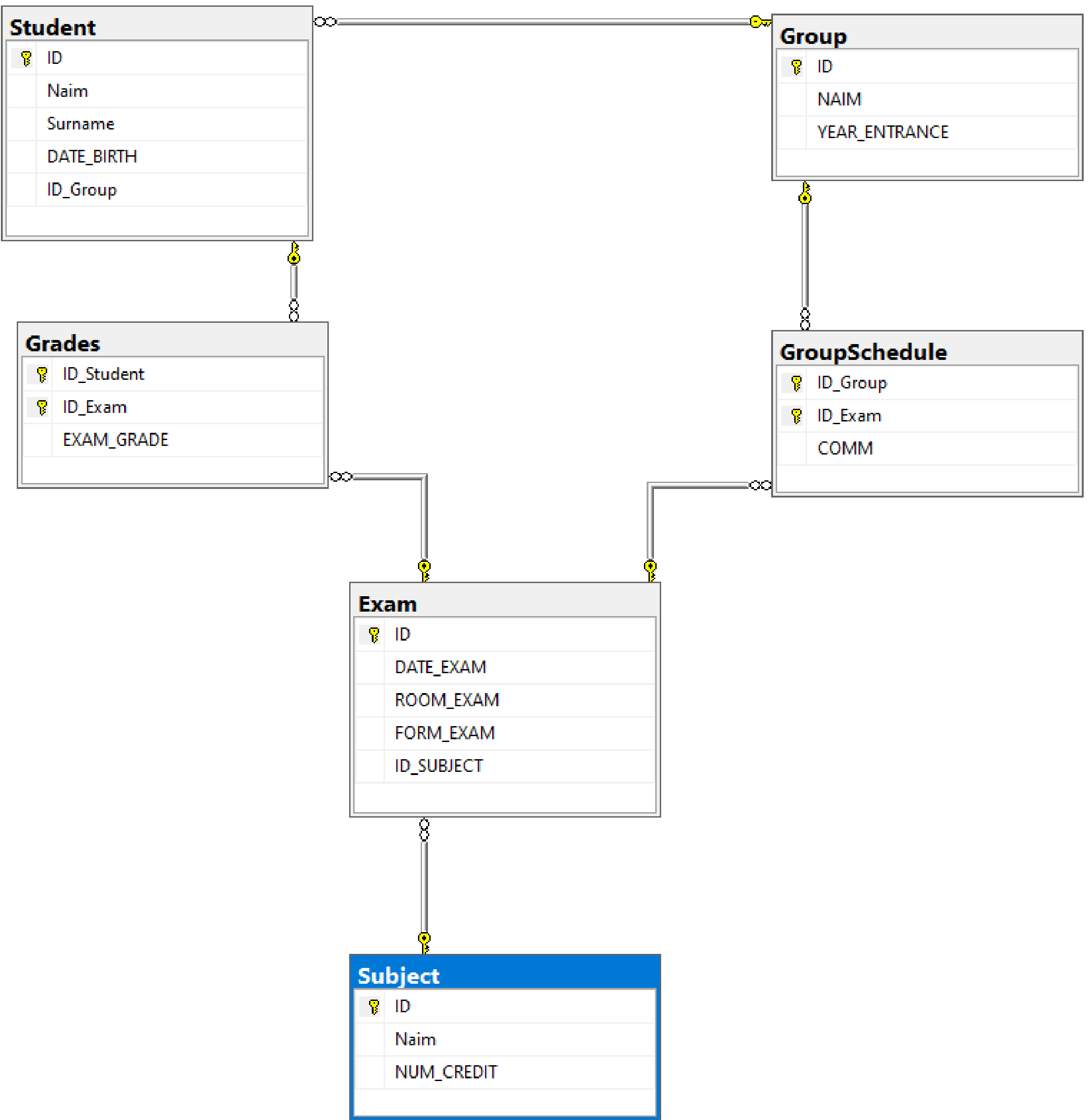
```
-- Створюємо логіни на сервері (master)
USE [master];
GO
IF NOT EXISTS (SELECT 1 FROM sys.sql_logins WHERE name = N'user_reader')
    CREATE LOGIN user_reader WITH PASSWORD = '111111', CHECK_POLICY = ON,
CHECK_EXPIRATION = ON;
IF NOT EXISTS (SELECT 1 FROM sys.sql_logins WHERE name = N'user_teacher')
    CREATE LOGIN user_teacher WITH PASSWORD = '222222', CHECK_POLICY = ON,
CHECK_EXPIRATION = ON;
IF NOT EXISTS (SELECT 1 FROM sys.sql_logins WHERE name = N'user_dekanat')
    CREATE LOGIN user_dekanat WITH PASSWORD = '333333', CHECK_POLICY = ON,
CHECK_EXPIRATION = ON;
GO

-- Прив'язуємо користувачів у базі MyEdu до логінів
USE [MyEdu];
GO
IF NOT EXISTS (SELECT 1 FROM sys.database_principals WHERE name =
N'user_reader')
    CREATE USER user_reader FOR LOGIN user_reader WITH DEFAULT_SCHEMA = dbo;
IF NOT EXISTS (SELECT 1 FROM sys.database_principals WHERE name =
N'user_teacher')
    CREATE USER user_teacher FOR LOGIN user_teacher WITH DEFAULT_SCHEMA =
dbo;
IF NOT EXISTS (SELECT 1 FROM sys.database_principals WHERE name =
N'user_dekanat')
```

```
    CREATE USER user_dekanat FOR LOGIN user_dekanat WITH DEFAULT_SCHEMA =
dbo;
GO

-- Додаємо користувачів у відповідні ролі --
ALTER ROLE role_reader       ADD MEMBER user_reader;
ALTER ROLE role_grader       ADD MEMBER user_teacher;
ALTER ROLE role_exam_manager ADD MEMBER user_dekanat;
GO
```

2) Згенерувати схему даних засобами СУБД



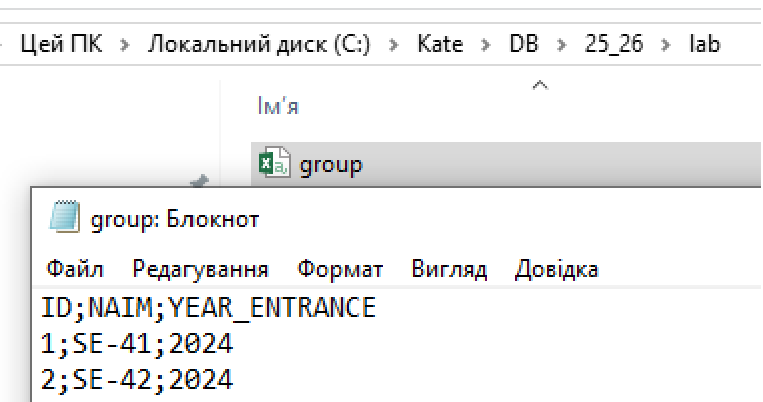
3) Імпортувати дані в створену БД з використанням засобів СУБД, а не DML SQL.

Один з можливих варіантів вставки даних – це використання BULK INSERT.

Наприклад, завантажити дані в таблицю Group можна наступним чином:

```
BULK INSERT dbo.[Group]
FROM 'C:\Kate\DB\25_26\lab\Group.csv'
WITH (
    FIRSTROW = 2,
    FIELDTERMINATOR = ';',
    ROWTERMINATOR   = '0x0a',
    CODEPAGE = '65001',
    TABLOCK
);
```

Приклад файлу, з якого завантажуються дані:



ЛАБОРАТОРНА РОБОТА № 3. БАЗОВІ ОПЕРАТОРИ ВИБІРКИ, ПІДЗАПИТИ ТА З’ЄДНАННЯ В SQL

Мета:

– вивчити команди DML, котрі використовуються в реляційних СУБД, для вибірки даних з таблиць;

– вивчити команди SQL для створення запитів з використанням підзапитів та з’єднань;

– навчитись створювати запити згідно їх словесного опису.

Короткі теоретичні відомості

Мова SQL (Structured Query Language) є стандартним засобом взаємодії з реляційними системами управління базами даних. Саме за допомогою SQL здійснюється створення структури бази даних, додавання, зміна та видалення записів, формування запитів на вибірку, організація доступу користувачів і керування транзакціями. У сучасних інформаційних системах SQL виконує роль універсальної мови опису, обробки та контролю даних, а тому її опанування є необхідною передумовою ефективної роботи з реляційними базами даних.

Слід зауважити, що SQL не є однорідним набором команд. Її засоби прийнято поділяти на кілька основних груп залежно від призначення. Умовно її команди поділяють на [7]:

- DDL (Data Definition Language) — створення, зміна та видалення об’єктів бази даних: таблиць, представлень, індексів, схем, послідовностей та інших об’єктів. До цієї групи належать, зокрема, команди CREATE, ALTER, DROP;

- DML (Data Manipulation Language) — засоби маніпулювання даними: вставлення, вибірка, оновлення та видалення записів. До цієї групи належать оператори SELECT, INSERT, UPDATE, DELETE. Саме цим командам присвячено більшість практичних задач із побудови запитів;

- DCL (Data Control Language) — засоби керування доступом до даних. Типовими командами є GRANT, REVOKE, а в Microsoft SQL Server також DENY;
- TCL (Transaction Control Language) — засоби керування транзакціями: BEGIN TRANSACTION, COMMIT, ROLLBACK, SAVEPOINT. Ці команди забезпечують узгоджене виконання операцій над даними.

Розглянемо короткий приклад використання різних груп команд SQL на практиці.

```
-- DDL: створення таблиці
CREATE TABLE Groups
(
    GroupID INT IDENTITY(1,1) PRIMARY KEY,
    GroupCode NVARCHAR(20) NOT NULL
);

-- DML: вставка та вибірка даних
INSERT INTO Groups (GroupCode) VALUES (N'ІП-31'), (N'ІП-32');

SELECT GroupID, GroupCode FROM Groups;

-- DCL: надання права на перегляд
GRANT SELECT ON OBJECT::Groups TO db_reader;

-- TCL: транзакція
BEGIN TRANSACTION;
UPDATE Groups SET GroupCode = N'ІП-31-оновлено' WHERE GroupID = 1;
ROLLBACK;
```

Основним оператором вибірки є оператор SELECT. Незважаючи на зовнішню простоту, він поєднує цілу систему засобів: визначення джерела даних, фільтрацію, з’єднання кількох таблиць, сортування, обмеження результату та використання підзапитів [1], [5], [6].

Слід підкреслити, що оператор SELECT є не просто засобом виведення даних з таблиці, а повноцінним механізмом побудови нового результуючого відношення на основі вже наявних даних. За допомогою цього оператора можна не лише вибирати окремі стовпці, а й обчислювати нові значення, поєднувати дані з кількох джерел, групувати дані, застосовувати умови відбору та впорядковувати кінцевий результат. Саме тому SELECT доцільно розглядати як центральний інструмент аналітичної та прикладної роботи з реляційною базою даних.

Важливо розуміти, що кожна секція SELECT має власне призначення, а їх поєднання утворює цілісну логіку формування вибірки. Неправильне розміщення умови, нерозуміння моменту агрегування або некоректне трактування джерела даних можуть призводити до логічно хибних результатів навіть тоді, коли запит є синтаксично правильним.

У загальному вигляді базовий скелет запиту на вибірку можна подати так:

```
SELECT <поля | вирази>
FROM <таблиця | представлення | підзапит> [AS псевдонім]
[JOIN інша_таблиця ON умова]
[WHERE умова]
[GROUP BY поля]
[HAVING умова_для_груп]
[ORDER BY поле [ASC | DESC]]
[OFFSET n ROWS FETCH NEXT m ROWS ONLY];
```

У наведеній конструкції секція SELECT визначає, які саме стовпці, вирази, агрегати та псевдоніми потраплять у фінальну вибірку. Секція FROM задає джерело даних, секція WHERE фільтрує рядки до агрегування, GROUP BY формує групи, HAVING фільтрує вже сформовані групи, а ORDER BY визначає остаточний порядок подання результатів.

Однією з найпоширеніших помилок під час формування SQL запиту є припущення, що запит виконується в тому самому порядку, у якому записані його секції. Насправді синтаксичний порядок запису не збігається з логічною послідовністю обробки. Логічна послідовність виконання запиту має фундаментальне значення для розуміння того, чому одні конструкції допустимі, а інші — ні. Наприклад, псевдонім, визначений у секції SELECT, не завжди можна використати в WHERE, оскільки секція WHERE логічно виконується раніше. Аналогічно агрегатні функції не застосовуються у WHERE саме тому, що на момент фільтрації окремих рядків групи ще не сформовано.

Отже, секції оператора SELECT слід розглядати не лише як послідовність ключових слів, а як етапи формування результату. Спочатку визначається джерело та спосіб з’єднання даних, потім відбувається відбір

рядків, далі — за потреби — їх групування, після чого формується підсумкова структура результату і лише наприкінці застосовується сортування та обмеження кількості рядків. Саме такий підхід дає змогу правильно інтерпретувати запити будь-якого рівня складності.

У логічному розумінні послідовність FROM → ON → JOIN → WHERE → GROUP BY → HAVING → SELECT → DISTINCT → ORDER BY → OFFSET/FETCH описує етапи поступового перетворення початкових даних у фінальну вибірку. Кожен наступний крок спирається на результат попереднього, тому зміна будь-якої секції впливає на підсумковий набір рядків. Саме ця послідовність лежить в основі семантики SQL-запиту на вибірку і є обов'язковою для усвідомленого конструювання складних запитів.

1) FROM — вибір джерел даних;
2) ON — застосування умов з'єднання;
3) JOIN — формування проміжного набору рядків у результаті з'єднань;
4) WHERE — фільтрація рядків до групування;
5) GROUP BY — формування груп;
6) HAVING — умова на відбір груп після агрегування;
7) SELECT — обчислення виразів, агрегатів і псевдонімів;
8) DISTINCT — усунення дублікатів, якщо ця конструкція задана;
9) ORDER BY — сортування результату;
10) OFFSET/FETCH або TOP — обмеження кількості рядків у фінальному результаті.

Слід зазначити, що фізичний план виконання запиту, який створює оптимізатор СУБД, може відрізнятися від цієї логічної послідовності. Оптимізатор може змінювати порядок з'єднань, переносити предикати ближче до джерел даних або використовувати індекси. Проте результат завжди повинен відповідати саме логічній семантиці запиту.

Кожен стовпець таблиці має визначений тип даних, від якого залежать правила порівняння, сортування, арифметичних операцій та можливість

використання певних індексів. Як зазначалось раніше, вибір типів даних впливає не лише на можливість зберігання певних значень, а й на характер виконання обчислень, результат сортування, правила неявного приведення типів та ефективність використання індексів. Наприклад, порівняння числових значень із рядковими літералами або змішування дата-часових типів без явного перетворення може ускладнювати аналіз запиту та призводити до неочікуваних результатів. Саме тому доцільно дотримуватися правила: вирази у запиті повинні бути типово узгодженими, а там, де виникає сумнів щодо неявного приведення, слід застосовувати явне перетворення типів. Такий підхід підвищує передбачуваність поведінки запиту і спрощує його супровід.
Коли у виразах або порівняннях беруть участь значення різних типів, СУБД може виконувати неявне приведення типів. Проте в практиці написання запитів доцільніше за можливості використовувати явне перетворення через CAST або CONVERT, оскільки це підвищує зрозумілість запиту й дозволяє уникати неоднозначностей.

```
SELECT StudentID,
       FullName,
       CAST(BirthDate AS DATETIME2) AS BirthDateTime,
       CONVERT(NVARCHAR(10), BirthDate, 23) AS BirthDate
FROM Students;
```

Особливе значення в SQL має поняття NULL. Воно не означає ні нуль, ні порожній рядок, а відображає відсутність або невідомість значення. Саме тому будь-яке порівняння зі значенням NULL не повертає TRUE або FALSE, а породжує результат UNKNOWN. Його особливість полягає в тому, що NULL не є конкретним значенням у звичайному розумінні, а тому до нього не можна механічно застосовувати ті самі правила порівняння, що й до чисел або рядків. Саме через це в SQL виникає тризначна логіка, у межах якої результат логічного виразу може бути не лише істинним або хибним, а й невизначеним. У практичному сенсі це означає, що будь-яка умова, яка потенційно може містити NULL, потребує особливої уваги. Помилки, пов’язані з неврахуванням UNKNOWN, часто проявляються не як синтаксичні збої, а як логічно

неправильні результати вибірки, коли частина рядків несподівано не потрапляє до результату або, навпаки, опрацьовується не так, як очікувалося.

Умови в секції WHERE відбирають лише ті рядки, для яких предикат є TRUE. Рядки, для яких результатом є FALSE або UNKNOWN, відкидаються. Звідси випливають важливі наслідки: для перевірки відсутності значення слід використовувати IS NULL, а для перевірки наявності — IS NOT NULL.

```
-- Некоректно: умова ніколи не буде істинною
SELECT *
FROM Students
WHERE Email = NULL;

-- Коректно
SELECT *
FROM Students
WHERE Email IS NULL;

-- Різниця між COUNT(*) і COUNT(Email)
SELECT COUNT(*)      AS TotalRows,
       COUNT(Email)  AS RowsWithEmail
FROM Students;
```

Для побудови умов відбору в SQL використовуються оператори порівняння =, <>, !=, >, <, >=, <=, IS NULL, IS NOT NULL. Для поєднання складніших умов застосовуються логічні оператори AND, OR, NOT. При цьому важливо враховувати пріоритет їх виконання: найвищий пріоритет має NOT, потім AND, а далі OR. У складних виразах доцільно явно використовувати дужки. Логічні оператори дозволяють формувати складні предикати, однак при цьому особливого значення набуває правильне структурування умови. Навіть формально правильний вираз може бути інтерпретований не так, як очікує розробник, якщо не враховано пріоритет виконання NOT, AND і OR. Тому в складних умовах доцільно свідомо використовувати дужки, навіть якщо вони не є строго обов'язковими з точки зору синтаксису. Крім того, необхідно враховувати, що логічні оператори в SQL працюють у середовищі тризначної логіки. Отже, результат комбінованої умови залежить не лише від істинності або хибності її частин, а й від того, чи не породжують вони значення UNKNOWN через участь NULL.

```
SELECT StudentID, FullName, GroupID, IsActive
FROM Students
WHERE IsActive = 1
```

```
  AND (GroupID = 1 OR GroupID = 2)
  AND FullName IS NOT NULL;
```

Оператор IN є зручним для перевірки приналежності значення певному списку констант або множині, котра формується підзапитом. Оператор EXISTS використовується для перевірки факту існування принаймні одного рядка, що відповідає заданим умовам підзапиту. Оператор BETWEEN дозволяє перевірити належність значення інтервалу включно з обома межами, а LIKE використовується для пошуку за шаблоном. Вибір між цими конструкціями повинен визначатися семантикою задачі. Особливо це стосується відмінності між IN та EXISTS у підзапитах, а також роботи NOT IN за наявності NULL, оскільки саме тут найчастіше виникають приховані логічні помилки.

```
-- IN
SELECT StudentID, FullName
FROM Students
WHERE GroupID IN (1, 2, 3);

-- BETWEEN
SELECT CourseID, CourseName, Credits
FROM Courses
WHERE Credits BETWEEN 3 AND 5;

-- LIKE
SELECT StudentID, FullName
FROM Students
WHERE FullName LIKE N'Іван%';

-- EXISTS
SELECT g.GroupID, g.GroupCode
FROM Groups AS g
WHERE EXISTS
(
    SELECT 1
    FROM Students AS s
    WHERE s.GroupID = g.GroupID
);
```

Слід пам’ятати, що конструкція NOT IN може давати неочікувані результати, якщо підзапит повертає хоча б одне значення NULL. У таких випадках правильніше використовувати NOT EXISTS.

```
-- Потенційно невірно, якщо підзапит поверне NULL
SELECT GroupID, GroupCode
FROM Groups
WHERE GroupID NOT IN
(
    SELECT GroupID
```

```
    FROM Students
);

-- Коректніший варіант
SELECT g.GroupID, g.GroupCode
FROM Groups AS g
WHERE NOT EXISTS
(
    SELECT 1
    FROM Students AS s
    WHERE s.GroupID = g.GroupID
);
```

Підзапит — це оператор SELECT, вкладений в інший SQL-вираз. Тобто підзапит — це вкладений запит, результат якого використовується зовнішнім оператором. Залежно від того, який результат він повертає, розрізняють скалярні, рядкові та табличні підзапити. За характером залежності від зовнішнього запиту підзапити можуть бути некорельованими та корельованими. Незалежний (некорельований) підзапит обчислюється автономно й повертає множину або скалярне значення, яке потім використовується в основному запиті. Корельований підзапит, навпаки, залежить від поточного рядка зовнішнього запиту і виконується повторно для кожного рядка. Підзапит може виконувати функцію джерела даних, механізму перевірки умови, інструмента обчислення окремого значення або засобу логічного розбиття складної задачі на зрозуміліші етапи.

На практиці найчастіше використовуються підзапити з IN, EXISTS, ANY, ALL та скалярні підзапити у SELECT. Конструкція EXISTS перевіряє сам факт існування хоча б одного пов'язаного рядка й часто є зручнішою та безпечнішою, ніж NOT IN. Скалярний підзапит повертає одне значення і часто використовується у списку SELECT або в секції WHERE. Табличний підзапит повертає набір рядків і може застосовуватися в секції FROM як похідна таблиця. Корельований підзапит посилається на стовпці зовнішнього запиту і логічно виконується для кожного його рядка окремо.

```
-- Скалярний підзапит
SELECT s.StudentID,
       s.FullName,
       (
           SELECT COUNT(*)
           FROM Grades AS g
```

```sql
            WHERE g.StudentID = s.StudentID
        ) AS GradeCount
FROM Students AS s;

-- Табличний підзапит у FROM
SELECT x.GroupID, x.StudentCount
FROM
(
    SELECT GroupID, COUNT(*) AS StudentCount
    FROM Students
    GROUP BY GroupID
) AS x
WHERE x.StudentCount >= 10;
```

У реляційній моделі інформація про дані, котрі зберігаються, як правило, розподіляються між кількома таблицями. Саме тому для отримання цілісного результату часто необхідно поєднувати рядки різних таблиць за певною умовою. Для цього в SQL використовуються з'єднання — JOIN. Необхідність використання JOIN безпосередньо пов'язана з принципами нормалізації даних. Оскільки дані зберігаються в різних таблицях, отримання цілісної інформації про об'єкт або подію зазвичай потребує поєднання кількох джерел даних. Саме тому з'єднання таблиць є не додатковою, а базовою операцією в реляційних запитах.

Правильне застосування JOIN вимагає чіткого розуміння того, який саме результат очікується: лише ті рядки, для яких існує відповідність у всіх джерелах, чи також рядки без відповідників. Від цього залежить вибір між INNER JOIN і зовнішніми з'єднаннями. Найпоширенішими різновидами є INNER JOIN, LEFT JOIN, RIGHT JOIN, FULL OUTER JOIN і CROSS JOIN. INNER JOIN повертає лише ті рядки, для яких знайдено відповідність у двох таблицях. LEFT JOIN зберігає всі рядки лівої таблиці, навіть якщо для них не знайдено відповідності в правій. RIGHT JOIN є симетричним варіантом щодо правої таблиці. FULL OUTER JOIN повертає всі рядки з обох таблиць.

```sql
-- INNER JOIN
SELECT s.StudentID, s.FullName, g.GroupCode
FROM Students AS s
INNER JOIN Groups AS g
    ON s.GroupID = g.GroupID;

-- LEFT JOIN
SELECT g.GroupCode, s.FullName
FROM Groups AS g
```

```
LEFT JOIN Students AS s
    ON s.GroupID = g.GroupID
ORDER BY g.GroupCode, s.FullName;

-- FULL OUTER JOIN
SELECT s.StudentID, s.FullName, g.GroupCode
FROM Students AS s
FULL OUTER JOIN Groups AS g
    ON s.GroupID = g.GroupID;
```

Для зовнішніх з’єднань важливо правильно розміщувати умови фільтрації. Якщо фільтр повинен впливати лише на праву сторону LEFT JOIN, його слід розміщувати в ON, а не у WHERE. Інакше рядки з NULL у правій таблиці можуть бути відкинуті, і зовнішнє з’єднання фактично перетвориться на внутрішнє.

```
-- Правильний варіант: фільтр на праву таблицю в ON
SELECT g.GroupCode, s.FullName
FROM Groups AS g
LEFT JOIN tudents AS s
    ON s.GroupID = g.GroupID
   AND s.IsActive = 1;

-- Потенційно помилковий варіант: WHERE перетворює результат на INNER
JOIN
SELECT g.GroupCode, s.FullName
FROM Groups AS g
LEFT JOIN Students AS s
    ON s.GroupID = g.GroupID
WHERE s.IsActive = 1;
```

Оператор CROSS JOIN реалізує декартовий добуток двох джерел даних. Результатом є всі можливі комбінації рядків таблиці A та таблиці B. Хоча така операція використовується рідше, вона є корисною в задачах побудови календарів, генерації матриць і тестових даних. З теоретичного погляду декартовий добуток є базовою операцією реляційної алгебри, на основі якої в поєднанні з відбором формуються загальні моделі з’єднання. Крім того, саме неявне виникнення декартового добутку часто є джерелом помилок у запитах, коли між таблицями не задано коректної умови з’єднання. У такому разі результат різко зростає за кількістю рядків і втрачає змістовну коректність.

```
SELECT g.GroupCode, c.CourseName
FROM Groups AS g
CROSS JOIN Courses AS c
ORDER BY g.GroupCode, c.CourseName;
```

Окрім з’єднань, SQL підтримує операції над множинами результатів запитів. До них належать UNION, UNION ALL, INTERSECT і EXCEPT. Вони

дозволяють об’єднувати, перетинати та віднімати результати окремих вибірок. Операції над множинами дають змогу розглядати результати окремих запитів як самостійні відношення та виконувати над ними дії об’єднання, перетину й різниці. Ці конструкції корисні тоді, коли різні джерела даних мають бути порівняні або об’єднані без побудови складних з’єднань. Водночас їх коректне використання вимагає уважного контролю кількості стовпців, сумісності типів і місця застосування ORDER BY.

```
-- UNION: об’єднання без дублікатів
SELECT FullName
FROM Students_2024
UNION
SELECT FullName
FROM Students_2025;

-- UNION ALL: об’єднання зі збереженням дублікатів
SELECT FullName
FROM Students_2024
UNION ALL
SELECT FullName
FROM Students_2025;

-- INTERSECT: спільні рядки
SELECT FullName
FROM Students_2024
INTERSECT
SELECT FullName
FROM Students_2025;

-- EXCEPT: рядки, що є в першому наборі, але відсутні в другому
SELECT FullName
FROM Students_2024
EXCEPT
SELECT FullName
FROM Students_2025;
```

Під час використання операцій над множинами слід пам’ятати, що кількість стовпців у відповідних гілках має збігатися, а типи стовпців повинні бути сумісними. Також ORDER BY допускається лише в кінці всієї конструкції.

Важливо розуміти, що створення запиту починається не з написання коду, а з аналізу інформаційної потреби. Спочатку визначається, які сутності та зв’язки задіяні в задачі, які атрибути повинні бути в результаті, чи потрібна фільтрація, чи є необхідність у зовнішньому або внутрішньому з’єднанні, чи

потрібно перевіряти існування пов’язаних записів через підзапит. Лише після цього формується SQL-конструкція.

Постановка задачі лабораторної роботи № 3

При виконанні лабораторної роботи необхідно виконати наступні дії.

1) Створити запити для вибірки даних з використанням (разом 8 запитів):

   a. Найпростіших умов та операторів порівняння.

   b. Умов з використанням логічних операторів AND, OR та NOT та їх комбінацій.

   c. З використанням виразів над стовпцями, як в якості новостворених стовпців, так і умовах.

   d. Використання операторів:

      i. Приналежності множині;

      ii. Приналежності діапазону;

      iii. Відповідності шаблону.

2) Створити запити з використанням підзапитів та з’єднань (разом 11 запитів) (в запитах повинні використовуватись 2 та більше таблиць):

   a. Використання підзапитів в рядку вибірки полів (у секції select) та вибірки з таблиць (у секції from).

   b. Використання підзапитів в умовах з конструкціями EXISTS, IN.

   c. Декартовий добуток.

   d. З’єднання декількох таблиць за рівністю та умовою відбору.

   e. Внутрішнього з’єднання.

   f. Лівого зовнішнього з’єднання.

   g. Правого зовнішнього з’єднання.

   h. Об’єднання та перетин запитів.

3) До кожного запиту з п.1 та 2 навести їх словесний (сутнісний) опис.

4) Створити запити за словесним описом, наведеним в завданні згідно варіанту.

5) Оформити звіт з роботи.

Вимоги до оформлення звіту з лабораторної роботи

Звіт повинен містити наступні складові частини:

- титульний лист;
- назву та мету роботи;
- варіант;
- SQL скрипти згідно завдання до лабораторної роботи;
- словесний опис до кожного зі скриптів;
- результати їх виконання;
- висновки.

Контрольні запитання

1) Визначте основні категорії команд виділяють у SQL?
2) Які команди належать до DML? Що таке NULL? Чим він відрізняється від нуля і порожнього рядка?
3) Поясніть різницю між WHERE і HAVING.
4) Який пріоритет мають NOT, AND, OR? Чому важливо ставити дужки?
5) Поясніть, чому умова WHERE CONVERT(date, ShipDate) = '2025-10-01' може бути повільною. Як її переписати ефективніше?
6) Поясніть різницю між корельованим і некорельованим підзапитами.
7) Сформулюйте відмінність між INNER JOIN і LEFT OUTER JOIN своїми словами.
8) Що таке декартовий добуток? Коли він корисний, а коли шкідливий?
9) Поясніть різницю між UNION і UNION ALL.
10) Чому SELECT * небажаний?
11) Як відрізнити логічну проблему в JOIN від очікуваної кратності даних?

12) Наведіть альтернативу NOT IN через LEFT JOIN … WHERE … IS NULL.

13) Чому інколи корельований підзапит з EXISTS працює швидше, ніж IN? Від чого це залежить?

14) Як упевнитися, що типи стовпців у ON сумісні й не вимагають неявних перетворень?

Приклад виконання завдання лабораторної роботи № 3

Розглянемо процес виконання лабораторної роботи для наведеного нижче предметного середовища.

Предметне середовище «Розклад екзаменів».

Система, котра проєктується, повинна зберігати інформацію про студентів, їх групи, предмети, екзамени (по якому предмету, якою групою та коли здається). Для кожної групи повинна існувати можливість отримати розклад екзаменів кожної групи, списки груп та отримані оцінки. Окрім того, один й той самий екзамен можуть здавати одразу декілька груп.

Запити:

а) Визначить студентів певної групи, котрі здають екзамен з дисципліни бази даних в січні поточного року.

б) Визначити дисципліни, по яким минулого року був відсутній екзамен, однак в розкладі поточного року він присутній.

1) Створити запити для вибірки даних з використанням (разом 8 запитів):

a. Найпростіших умов та операторів порівняння

```
-- Виведіть групи, котрі були сформовані після 2023 року включно
SELECT g.NAIM, g.YEAR_ENTRANCE
FROM dbo.[Group] AS g
WHERE g.YEAR_ENTRANCE >= 2023;
```

	NAIM	YEAR_ENTRANCE
1	ІП-з51	2024
2	ІП-з11	2025
3	ІП-з12	2023

b. Умов з використанням логічних операторів AND, OR та NOT та їх комбінацій.

```
-- Виведіть студентів, котрі не додані до конкретної групи або не вказана
дата народження
SELECT s.Surname, s.Naim, s.ID_Group, s.DATE_BIRTH
FROM dbo.Student AS s
WHERE (s.[ID_Group] IS NULL OR s.[DATE_BIRTH] IS NULL)
  AND s.[Surname] <> '' AND s.[Naim] <> '';
```

	Surname	Naim	ID_Group	DATE_BIRTH

```
-- Виведіть предмети, котрі проводяться очно або вказана аудиторію
SELECT s.Naim, e.DATE_EXAM, e.ROOM_EXAM, e.FORM_EXAM
FROM dbo.Exam AS e join dbo.Subject s on e.ID_SUBJECT = s.ID
WHERE (e.[FORM_EXAM] = 'очна' OR e.ROOM_EXAM <> 'н/д')
           AND NOT (e.ID_SUBJECT IS NULL);
```

	Naim	DATE_EXAM	ROOM_EXAM	FORM_EXAM
1	Бази даних	2025-01-20 10:00:00	101	очна
2	Бази даних	2024-06-10 10:00:00	201	очна
3	Програмування	2025-02-05 09:00:00	102	очна
4	Алгоритми	2025-01-25 12:00:00	103	дистанційна
5	Операційні си...	2025-12-22 14:00:00	104	очна
6	Комп'ютерні м...	2025-03-03 11:00:00	105	очна
7	Філософія	2025-05-15 10:00:00	106	очна

c. З використанням виразів над стовпцями, як в якості новостворених стовпців, так і умовах

```
-- На поточну дату визначте студентів старше 18 років
SELECT s.Surname, s.Naim, s.DATE_BIRTH
FROM dbo.Student AS s
WHERE s.DATE_BIRTH IS NOT NULL
  AND DATEDIFF(year, s.DATE_BIRTH, SYSUTCDATETIME()) >= 18;
```

	Surname	Naim	DATE_BIRTH
1	Ліщук	Олександр	2006-04-11 00:00:00.000
2	Коваленко	Марія	2006-11-23 00:00:00.000
3	Іваненко	Дмитро	2005-07-05 00:00:00.000
4	Шевченко	Ірина	2006-01-30 00:00:00.000
5	Бондар	Тарас	2005-12-12 00:00:00.000
6	Мельник	Олег	2007-03-19 00:00:00.000
7	Романюк	Наталія	2006-09-08 00:00:00.000
8	Сидоренко	Андрій	2005-05-17 00:00:00.000

```
-- Визначте студентів, старших за 17 років та молодших за 30
SELECT
  s.Surname,
  s.Naim,
  s.DATE_BIRTH,
  DATEDIFF(year, s.DATE_BIRTH, SYSUTCDATETIME()) AS AgeYears
FROM dbo.Student AS s
WHERE s.DATE_BIRTH IS NOT NULL
  AND DATEDIFF(year, s.DATE_BIRTH, SYSUTCDATETIME()) BETWEEN 17 AND 30;
```

	Surname	Naim	DATE_BIRTH	AgeYears
4	Шевченко	Ірина	2006-01-30 00:00:00.000	19
5	Бондар	Тарас	2005-12-12 00:00:00.000	20
6	Мельник	Олег	2007-03-19 00:00:00.000	18
7	Романюк	Наталія	2006-09-08 00:00:00.000	19
8	Сидоренко	Андрій	2005-05-17 00:00:00.000	20
9	Кравченко	Олена	2006-02-25 00:00:00.000	19
10	Гриценко	Павло	2004-10-03 00:00:00.000	21
11	Сергієнко	Юлія	2005-06-28 00:00:00.000	20

d. Використання операторів:

i. Приналежності множині

```
-- Визначте групи, які були сформовані у 2021,2022,2023 роках
SELECT g.NAIM, g.YEAR_ENTRANCE
FROM dbo.[Group] AS g
WHERE g.YEAR_ENTRANCE IN (2021, 2022, 2023);
```

	NAIM	YEAR_ENTRANCE
1	ІП-з12	2023

ii. Приналежності діапазону

```
-- Визначте предмети, по яким призначені екзамени в першому кварталі 2025
року
SELECT s.Naim, e.DATE_EXAM, e.ROOM_EXAM
FROM dbo.Exam AS e JOIN dbo.Subject s on e.ID_SUBJECT = s.ID
WHERE e.DATE_EXAM >= DATETIMEFROMPARTS(2025,1,1,0,0,0,0)
  AND e.DATE_EXAM <  DATETIMEFROMPARTS(2025,4,1,0,0,0,0);
```

	Naim	DATE_EXAM	ROOM_EXAM
1	Бази даних	2025-01-20 10:00:00	101
2	Програмування	2025-02-05 09:00:00	102
3	Алгоритми	2025-01-25 12:00:00	103
4	Комп'ютерні м...	2025-03-03 11:00:00	105

iii. Відповідності шаблону

```
-- Визначете дисципліни, назва яких містить словосполучення дан
SELECT sub.Naim AS SubjectName, sub.NUM_CREDIT
FROM dbo.Subject AS sub
WHERE sub.Naim LIKE '%дан%';
```

	SubjectName	NUM_CREDIT
1	Бази даних	5

2) Створити запити з використанням підзапитів та з'єднань (разом 11 запитів) (в запитах повинні використовуватись 2 та більше таблиць):

a. Використання підзапитів в рядку вибірки полів (у секції select) та вибірки з таблиць (у секції from)

```
-- Для кожного студента обчисліть кількість призначених йому екзаменівм
SELECT
  s.ID,
  s.Surname,
  s.Naim,
  s.ID_Group,
```

```
  (SELECT COUNT(*)
   FROM dbo.GroupSchedule AS gs
   WHERE gs.ID_Group = s.ID_Group) AS ExamsAssignedToGroup
FROM dbo.Student AS s;
```

	ID	Surname	Naim	ID_Group	ExamsAssignedToGroup
1	25	Ліщук	Олександр	9	3
2	26	Коваленко	Марія	9	3
3	27	Іваненко	Дмитро	9	3
4	28	Шевченко	Ірина	9	3
5	29	Бондар	Тарас	9	3
6	30	Мельник	Олег	10	3
7	31	Романюк	Наталія	10	3
8	32	Сидоренко	Андрій	10	3
9	33	Кравченко	Олена	10	3
10	34	Гриценко	Павло	11	2
11	35	Сергієнко	Юлія	11	2
12	36	Данилюк	Микита	11	2

```
-- Для кожної групи визначте дату найближчого екзамену
SELECT g.NAIM, NextExam.NextExamAt
FROM dbo.[Group] AS g
JOIN (
    SELECT gs.ID_Group, MIN(e.DATE_EXAM) AS NextExamAt
    FROM dbo.GroupSchedule AS gs
    JOIN dbo.Exam AS e ON e.ID = gs.ID_Exam
    WHERE e.DATE_EXAM >= SYSUTCDATETIME()
    GROUP BY gs.ID_Group
) AS NextExam
  ON NextExam.ID_Group = g.ID;
```

	NAIM	NextExamAt
1	ІП-з11	2025-12-22 14:00:00

b. Використання підзапитів в умовах з конструкціями EXISTS, IN

```
-- Визначте групи, для яких створено розклад екзаменів
SELECT g.NAIM
FROM dbo.[Group] AS g
WHERE EXISTS (
    SELECT 1
    FROM dbo.GroupSchedule AS gs
    WHERE gs.ID_Group = g.ID
);
```

	NAIM
1	ІП-з51
2	ІП-з11
3	ІП-з12

```
-- Визначте студентів, котрі мають оцінку з зазначених дисциплін
SELECT DISTINCT s.Surname, s.Naim
FROM dbo.Student AS s
WHERE s.ID IN (
    SELECT gr.ID_Student
    FROM dbo.Grades AS gr
    JOIN dbo.Exam   AS e   ON e.ID = gr.ID_Exam
```

```
    JOIN dbo.Subject AS sub ON sub.ID = e.ID_SUBJECT
    WHERE sub.[Naim] IN ('Бази даних', 'Програмування', 'Алгоритми')
);
```

	Surname	Naim
1	Бондар	Тарас
2	Гриценко	Павло
3	Данилюк	Микита
4	Іваненко	Дмитро
5	Коваленко	Марія
6	Кравченко	Олена
7	Ліщук	Олександр
8	Мельник	Олег
9	Романюк	Наталія
10	Сергієнко	Юлія
11	Сидоренко	Андрій
12	Шевченко	Ірина

c. Декартовий добуток

```
-- Визначте всі предмети та групи, по яким потрібно сформувати розклад
SELECT g.NAIM AS GroupName,
       sub.Naim AS SubjectName
FROM dbo.[Group] AS g
CROSS JOIN dbo.Subject AS sub;
```

	GroupName	SubjectName
1	ІП-з51	Бази даних
2	ІП-з51	Програмування
3	ІП-з51	Алгоритми
4	ІП-з51	Операційні системи
5	ІП-з51	Комп'ютерні мережі
6	ІП-з51	Філософія
7	ІП-з11	Бази даних
8	ІП-з11	Програмування
9	ІП-з11	Алгоритми
10	ІП-з11	Операційні системи
11	ІП-з11	Комп'ютерні мережі
12	ІП-з11	Філософія
13	ІП-з12	Бази даних
14	ІП-з12	Програмування
15	ІП-з12	Алгоритми
16	ІП-з12	Операційні системи
17	ІП-з12	Комп'ютерні мережі
18	ІП-з12	Філософія

d. З’єднання декількох таблиць за рівністю та умовою відбору

```
-- Визначте студентів, котрі повинні здавати екзамени очно в найближчий
час
SELECT s.Surname, s.Naim,
       g.NAIM AS GroupName,
       sub.Naim AS SubjectName,
       e.DATE_EXAM, e.ROOM_EXAM, e.FORM_EXAM
FROM dbo.Student AS s, dbo.[Group]  AS g, dbo.GroupSchedule AS gs,
```

```
       dbo.Exam  AS e, dbo.Subject AS sub
WHERE g.ID = s.ID_Group
       AND gs.ID_Group = g.ID
       AND e.ID = gs.ID_Exam
       AND sub.ID = e.ID_SUBJECT
       AND e.FORM_EXAM = 'очна'
       AND e.DATE_EXAM >= SYSUTCDATETIME();
```

	Surname	Naim	GroupName	SubjectName	DATE_EXAM	ROOM_EXAM	FORM_EXAM
1	Мельник	Олег	ІП-з11	Операційні системи	2025-12-22 14:00:00	104	очна
2	Романюк	Наталія	ІП-з11	Операційні системи	2025-12-22 14:00:00	104	очна
3	Сидоренко	Андрій	ІП-з11	Операційні системи	2025-12-22 14:00:00	104	очна
4	Кравченко	Олена	ІП-з11	Операційні системи	2025-12-22 14:00:00	104	очна

e. Внутрішнього з’єднання

```
-- Створіть списки груп
SELECT s.Surname, s.Naim, g.NAIM AS GroupName
FROM dbo.Student AS s
INNER JOIN dbo.[Group] AS g
  ON g.ID = s.ID_Group
ORDER BY g.ID, s.Surname, s.Naim;
```

	Surname	Naim	GroupName
1	Бондар	Тарас	ІП-з51
2	Іваненко	Дмитро	ІП-з51
3	Коваленко	Марія	ІП-з51
4	Ліщук	Олександр	ІП-з51
5	Шевченко	Ірина	ІП-з51
6	Кравченко	Олена	ІП-з11
7	Мельник	Олег	ІП-з11
8	Романюк	Наталія	ІП-з11
9	Сидоренко	Андрій	ІП-з11
10	Гриценко	Павло	ІП-з12
11	Данилюк	Микита	ІП-з12
12	Сергієнко	Юлія	ІП-з12

f. Лівого зовнішнього з’єднання

```
-- Виведіть дисципліни та вкажіть чи здається по ній екзамен
SELECT distinct sub.Naim AS SubjectName,
        CASE WHEN e.ID is null THEN 'залік' ELSE 'екзамен' END AS isExams
FROM dbo.Subject AS sub
      LEFT JOIN dbo.Exam AS e ON e.ID_SUBJECT = sub.ID;
```

	SubjectName	isExams
1	Алгоритми	екзамен
2	Бази даних	екзамен
3	Комп'ютерні мережі	екзамен
4	Операційні системи	екзамен
5	Програмування	екзамен
6	Філософія	екзамен

g. Правого зовнішнього з’єднання

```
-- Сформуйте екзаменаційну відомість
```

```sql
SELECT e.DATE_EXAM, e.ROOM_EXAM, e.FORM_EXAM,
       s.Surname, s.Naim, gr.EXAM_GRADE
FROM dbo.Grades AS gr RIGHT OUTER JOIN dbo.Exam AS e
  ON e.ID = gr.ID_Exam
  LEFT JOIN dbo.Student s on s.ID = gr.ID_Student;
```

	DATE_EXAM	ROOM_EXAM	FORM_EXAM	Surname	Naim	EXAM_GRADE
1	2025-01-20 10:00:00	101	очна	Ліщук	Олександр	75
2	2025-01-20 10:00:00	101	очна	Коваленко	Марія	76
3	2025-01-20 10:00:00	101	очна	Іваненко	Дмитро	77
4	2025-01-20 10:00:00	101	очна	Шевченко	Ірина	78
5	2025-01-20 10:00:00	101	очна	Бондар	Тарас	79
6	2025-01-20 10:00:00	101	очна	Мельник	Олег	80
7	2025-01-20 10:00:00	101	очна	Романюк	Наталія	81
8	2025-01-20 10:00:00	101	очна	Сидоренко	Андрій	82
9	2025-01-20 10:00:00	101	очна	Кравченко	Олена	83
10	2024-06-10 10:00:00	201	очна	NULL	NULL	NULL
11	2025-02-05 09:00:00	102	очна	Ліщук	Олександр	85
12	2025-02-05 09:00:00	102	очна	Коваленко	Марія	86
13	2025-02-05 09:00:00	102	очна	Іваненко	Дмитро	87
14	2025-02-05 09:00:00	102	очна	Шевченко	Ірина	88
15	2025-02-05 09:00:00	102	очна	Бондар	Тарас	89
16	2025-02-05 09:00:00	102	очна	Гриценко	Павло	63
17	2025-02-05 09:00:00	102	очна	Сергієнко	Юлія	64
18	2025-02-05 09:00:00	102	очна	Данилюк	Микита	65
19	2025-01-25 12:00:00	103	дистанційна	Ліщук	Олександр	80
20	2025-01-25 12:00:00	103	дистанційна	Коваленко	Марія	81
21	2025-01-25 12:00:00	103	дистанційна	Іваненко	Дмитро	82

h. Об’єднання та перетин запитів

```sql
-- Виведіть всі предмети та назви груп
SELECT CAST(g.NAIM AS varchar(50)) AS NameLike
FROM dbo.[Group] AS g
UNION
SELECT CAST(sub.Naim AS varchar(50)) AS NameLike
FROM dbo.Subject AS sub;
```

	NameLike
1	Алгоритми
2	Бази даних
3	ІП-з11
4	ІП-з12
5	ІП-з51
6	Комп'ютерні мережі
7	Операційні системи
8	Програмування
9	Філософія

```sql
-- Виведіть студентів, котрі вже отримали оцінки по екзаменам
SELECT st.Surname,st.Naim
FROM [dbo].[Grades] AS gr JOIN dbo.Student st ON gr.ID_Student = st.ID
INTERSECT
SELECT s.Surname,s.Naim
FROM dbo.Student AS s
```

```
    WHERE s.ID_Group IN (SELECT DISTINCT gs.ID_Group FROM dbo.GroupSchedule AS gs);
```

	Surname	Naim
1	Бондар	Тарас
2	Гриценко	Павло
3	Данилюк	Микита
4	Іваненко	Дмитро
5	Коваленко	Марія
6	Кравченко	Олена
7	Ліщук	Олександр
8	Мельник	Олег
9	Романюк	Наталія
10	Сергієнко	Юлія
11	Сидоренко	Андрій
12	Шевченко	Ірина

Запити:

а) Визначить студентів певної групи, котрі здають екзамен з дисципліни бази даних в січні поточного року.

```
SELECT DISTINCT
    s.Surname,
    s.Naim      AS Name,
    g.NAIM      AS GroupName,
    sub.Naim    AS SubjectName,
    e.DATE_EXAM,
    gr.EXAM_GRADE
FROM dbo.Student AS s
JOIN dbo.[Group]  AS g   ON g.ID = s.ID_Group
JOIN dbo.GroupSchedule AS gs ON gs.ID_Group = g.ID
JOIN dbo.Exam    AS e   ON e.ID = gs.ID_Exam
JOIN dbo.Subject AS sub ON sub.ID = e.ID_SUBJECT
JOIN dbo.Grades  AS gr  ON gr.ID_Exam = e.ID AND gr.ID_Student = s.ID
WHERE g.NAIM = 'ІП-з51'
  AND sub.Naim = 'Бази даних'
  AND  e.DATE_EXAM  >=  DATETIMEFROMPARTS(YEAR(SYSUTCDATETIME()),  1,  1, 0,0,0,0)
  AND  e.DATE_EXAM  <   DATETIMEFROMPARTS(YEAR(SYSUTCDATETIME()),  2,  1, 0,0,0,0);
```

	Surname	Name	GroupName	SubjectName	DATE_EXAM	EXAM_GRADE
1	Бондар	Тарас	ІП-з51	Бази даних	2025-01-20 10:00:00	79
2	Іваненко	Дмитро	ІП-з51	Бази даних	2025-01-20 10:00:00	77
3	Коваленко	Марія	ІП-з51	Бази даних	2025-01-20 10:00:00	76
4	Ліщук	Олександр	ІП-з51	Бази даних	2025-01-20 10:00:00	75
5	Шевченко	Ірина	ІП-з51	Бази даних	2025-01-20 10:00:00	78

б) Визначити дисципліни, по яким минулого року був відсутній екзамен, однак в розкладі поточного року він присутній.

```
SELECT DISTINCT sub.Naim AS SubjectName
FROM dbo.Subject AS sub
WHERE NOT EXISTS (
    SELECT 1
```

```
            FROM dbo.Exam AS e_prev
            WHERE e_prev.ID_SUBJECT = sub.ID
              AND e_prev.DATE_EXAM >= DATETIMEFROMPARTS(YEAR(SYSUTCDATETIME())-
1, 1, 1, 0,0,0,0)
              AND e_prev.DATE_EXAM <  DATETIMEFROMPARTS(YEAR(SYSUTCDATETIME()),
1, 1, 0,0,0,0)
        )
        AND EXISTS (
            SELECT 1
            FROM dbo.GroupSchedule AS gs
            JOIN dbo.Exam AS e_now ON e_now.ID = gs.ID_Exam
            WHERE e_now.ID_SUBJECT = sub.ID
              AND  e_now.DATE_EXAM  >=  DATETIMEFROMPARTS(YEAR(SYSUTCDATETIME()),
1, 1, 0,0,0,0)
              AND e_now.DATE_EXAM <  DATETIMEFROMPARTS(YEAR(SYSUTCDATETIME()) +
1, 1, 1, 0,0,0,0)
        );
```

	SubjectName
1	Алгоритми
2	Комп'ютерні мережі
3	Операційні системи
4	Програмування
5	Філософія

ЛАБОРАТОРНА РОБОТА № 4. СКЛАДНІ ЗАПИТИ З АГРЕГАТНИМИ ТА АНАЛІТИЧНИМИ ФУНКЦІЯМИ. РОБОТА З ПРЕДСТАВЛЕННЯМИ.

Мета:

– вивчити оператори, котрі використовуються в реляційних СУБД, для вибірки даних з таблиць, групування та сортування даних;

– навчитись використовувати вбудовані функції в запитах;

– вивчити призначення представлень (view) у базах даних, синтаксис та семантику команд SQL для їх створення, зміни та видалення.

Короткі теоретичні відомості

SQL є декларативною мовою для створення запитів до реляційних баз даних. Теоретичною основою SQL є реляційна модель даних, у якій таблиці розглядаються як відношення, рядки — як кортежі, а стовпці — як атрибути. У цьому контексті запит на вибірку можна тлумачити як композицію операцій відбору, проєкції, з’єднання, групування, агрегування та операцій над множинами. Складні запити з агрегатними та аналітичними функціями дозволяють не лише отримувати окремі рядки, а й формувати узагальнені, підсумкові та порівняльні характеристики даних [10].

Особливістю SQL є те, що результати запиту за замовчуванням можуть містити дублікати рядків. Якщо внаслідок з’єднання таблиць, операцій над множинами або вибору певних полів однакові кортежі повторюються, вони залишаються у відповіді, якщо не використано конструкцію DISTINCT. Крім того, SQL працює у тризначній логіці TRUE/FALSE/UNKNOWN, що пов’язано з наявністю спеціального значення NULL.

Під час побудови запиту важливо розуміти логічну послідовність обробки оператора SELECT. Хоча текст запиту починається із секції SELECT, фактично логіка його обробки є іншою. У загальному випадку логічна послідовність має вигляд: FROM → WHERE → GROUP BY → HAVING → SELECT → DISTINCT → ORDER BY. Якщо в запиті використовуються

з’єднання, то умови ON логічно застосовуються на етапі формування джерела даних у FROM. Саме така послідовність пояснює, чому, наприклад, у WHERE не можна безпосередньо використовувати агрегатні функції, а для фільтрації груп призначено секцію HAVING.

Секція FROM визначає джерело даних, тобто той набір рядків, над яким працюватимуть усі інші частини запиту. Джерелом можуть бути базові таблиці, представлення, підзапити у FROM. Якщо в запиті використовується кілька таблиць, їх поєднання виконується за допомогою JOIN. Найуживанішими є INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN та CROSS JOIN. Секція WHERE виконує фільтрацію окремих рядків до моменту групування. Тут використовуються оператори порівняння, логічні зв’язки, умови приналежності до множини, перевірки на NULL і підзапити. На цьому етапі ще не існує груп, а тому агрегатні функції в WHERE не застосовуються. Якщо ж потрібно відібрати не рядки, а вже сформовані групи, використовується HAVING.

Секція GROUP BY об’єднує рядки у групи за вказаними ключами. Всі рядки, що мають однакові значення полів групування, розглядаються як одна логічна група. Після цього до кожної групи можуть бути застосовані агрегатні функції, які повертають одне підсумкове значення для всієї групи. Секція HAVING, на відміну від WHERE, накладає умову відбору на рядки вже після групування і дозволяє задавати умови саме щодо агрегованих результатів.

Секція ORDER BY визначає остаточний порядок подання даних. Якщо сортування не задано явно, СУБД не гарантує жодного стабільного порядку виведення рядків.

```
SELECT <поля | вирази>
FROM <джерело_даних>
[WHERE умова]
[GROUP BY поля]
[HAVING умова_для_груп]
[ORDER BY поле [ASC | DESC]]
[OFFSET n ROWS FETCH NEXT m ROWS ONLY];
```

Агрегатні функції використовуються для обчислення підсумкових характеристик над множиною рядків. До базових агрегатних функцій

належать COUNT, SUM, AVG, MIN і MAX. Вони дозволяють порахувати кількість рядків, знайти суму числових значень, обчислити середнє арифметичне, визначити мінімум і максимум. У запитах без GROUP BY агрегатна функція застосовується до всього результуючого набору, а за наявності GROUP BY — окремо до кожної групи.

Необхідно враховувати, що COUNT(*) рахує всі рядки, тоді як COUNT(стовпець) враховує лише ті з них, де значення відповідного стовпця не є NULL. Функції SUM, AVG, MIN і MAX також ігнорують NULL-значення, тому за наявності пропусків результати агрегування можуть відрізнятися від інтуїтивного очікування, якщо не врахувати цю особливість заздалегідь.

Однією з найважливіших можливостей мови SQL є не лише вибірка окремих рядків, а й узагальнення даних. Слід зауважити, що агрегування є не просто технічним прийомом скорочення кількості рядків. По суті, воно переводить запит із рівня аналізу окремих записів на рівень аналізу сукупностей записів, об’єднаних за певною ознакою. У найпростішому випадку агрегатні функції можуть застосовуватися без групування, тобто до всієї таблиці як до єдиної множини рядків. Наприклад:

```
SELECT
    COUNT(*) AS StudentCount,
    AVG(AverageMark) AS AvgMark,
    MIN(BirthDate) AS OldestBirthDate,
    MAX(BirthDate) AS YoungestBirthDate
FROM Students;
```

Однак значно частіше виникає потреба виконувати агрегування не над усією таблицею, а окремо для різних категорій записів. Для цього використовується оператор GROUP BY. Його призначення полягає в тому, щоб об’єднати рядки з однаковими значеннями в заданих стовпцях у логічні групи, після чого до кожної з них застосувати агрегатні функції.

Загальна форма запиту з групуванням має вигляд:

```
SELECT <атрибути_групування>, <агрегатні_вирази>
FROM <джерело_даних>
[WHERE <умова_для_рядків>]
GROUP BY <атрибути_групування>
[HAVING <умова_для_груп>]
[ORDER BY <вирази>];
```

Важливо розуміти, що після застосування GROUP BY секція SELECT підпорядковується особливому правилу: у ній можна використовувати або стовпці, зазначені в GROUP BY, або агрегатні функції. Це пояснюється тим, що після групування кожна група представляється як єдиний узагальнений об'єкт, тому довільний неагрегований стовпець, який не входить до ключа групування, вже не має однозначного значення. Наприклад, коректним є такий запит:

```
SELECT
    GroupCode,
    COUNT(*) AS StudentCount
FROM Students
GROUP BY GroupCode;
```

Групування може виконуватися не лише за одним, а й за кількома стовпцями. У такому разі кожна група формується на основі комбінації значень усіх полів групування. Наприклад:

```
SELECT
    GroupCode,
    IsActive,
    COUNT(*) AS StudentCount
FROM Students
GROUP BY GroupCode, IsActive;
```

Особливу увагу слід приділити поведінці NULL у групуванні. На відміну від логічних порівнянь, де NULL породжує UNKNOWN, у GROUP BY усі рядки з NULL у відповідному стовпці вважаються такими, що належать до однієї групи. Тобто NULL у цьому контексті не ігнорується, а виступає окремим значенням групування. Це може мати важливе значення під час побудови звітів, якщо в даних присутні пропуски.

Для узагальнення інформації в SQL використовуються агрегатні функції. До основних належать MIN, MAX, SUM, AVG, COUNT. Функція COUNT(*) рахує всі рядки в групі, тоді як COUNT(ColumnName) рахує лише ті рядки, у яких відповідний стовпець не містить NULL. Функції SUM і AVG застосовуються до числових значень і також ігнорують NULL. Функції MIN і MAX повертають відповідно мінімальне та максимальне значення в межах групи. Однак у практичних задачах часто недостатньо просто сформувати групи; необхідно ще й відібрати лише ті з них, які задовольняють певні умови.

Саме для цього використовується секція HAVING. На відміну від WHERE, яка фільтрує окремі рядки до групування, HAVING застосовується після групування і дозволяє накладати умови на вже сформовані групи. Наприклад:

```
SELECT
    GroupCode,
    COUNT(*) AS StudentCount
FROM Students
GROUP BY GroupCode
HAVING COUNT(*) >= 20;
```

Необхідно чітко розмежовувати призначення WHERE і HAVING. Секція WHERE працює на рівні рядків і відкидає записи ще до етапу групування. Секція HAVING працює на рівні груп і застосовується після того, як групи вже сформовано, тому в умовах HAVING можна використовувати як агрегатні функції, так і стовпці групування. Наприклад:

```
SELECT
    GroupCode,
    COUNT(*) AS StudentCount
FROM Students
GROUP BY GroupCode
HAVING GroupCode LIKE N'ІП-%'
   AND COUNT(*) >= 15;
```

Окрім агрегатних функцій, SQL надає великий набір вбудованих функцій, які використовуються для перетворення й аналізу даних безпосередньо в запитах. Їх доцільно розглядати за кількома основними групами: числові, рядкові, дата-часові, функції перетворення типів, функції роботи з NULL, статистичні та віконні функції. До числових функцій належать, зокрема, ABS, CEILING, FLOOR, POWER, ROUND. Вони дають змогу виконувати математичні обчислення й нормалізувати числові значення у результаті запиту. До рядкових функцій належать LEN, SUBSTRING, REPLACE, CONCAT, TRIM, UPPER, LOWER, CHARINDEX. Їх застосовують для очищення, форматування та аналізу текстових даних. Для роботи з датами та часом у T-SQL використовуються функції GETDATE, SYSDATETIME, DATEADD, DATEDIFF, DATENAME, DATEPART, а також конструктори на кшталт DATETIMEFROMPARTS. Під час побудови складних запитів ці функції дозволяють визначати проміжки часу, обчислювати вік записів, виділяти рік, місяць чи день і формувати часові зрізи даних. До функцій

перетворення типів належать CAST, CONVERT, TRY_CAST, TRY_CONVERT. Їх застосування є особливо важливим у складних запитах, де дані різних типів беруть участь в одному виразі. Функції COALESCE, ISNULL і NULLIF використовуються для керування пропущеними значеннями та побудови більш стійких до NULL виразів.

Особливе місце при побудові запитів займають аналітичні, або віконні, функції. На відміну від звичайних агрегатних функцій, вони не згортають набір рядків до одного результату для групи, а обчислюють значення для кожного рядка над певним «вікном» даних. Завдяки цьому можна одночасно зберегти деталізацію рядків і одержати аналітичні характеристики, пов'язані з групою, підгрупою або порядком рядків. Віконна функція використовує конструкцію OVER(...), у межах якої можуть задаватися PARTITION BY, ORDER BY та параметри кадру. PARTITION BY розбиває дані на незалежні логічні групи, ORDER BY визначає порядок рядків у межах кожної групи, а кадр вікна уточнює, які саме рядки беруть участь у конкретному обчисленні.

До основних класів віконних функцій належать функції ранжування ROW_NUMBER, RANK, DENSE_RANK, NTILE; функції навігації LAG, LEAD, FIRST_VALUE, LAST_VALUE; а також агрегатні функції, наприклад SUM(...) OVER(...), AVG(...) OVER(...), MIN(...) OVER(...), MAX(...) OVER(...). Вони широко застосовуються для побудови рейтингів, пошуку попереднього або наступного значення, обчислення накопичувальних сум, порівняння поточного рядка з середнім по групі тощо. Аналітичні функції особливо важливі в тих випадках, коли потрібно сформувати результат, у якому кожен рядок зберігає свою індивідуальність, але водночас отримує контекст — місце в групі, порядковий номер, відхилення від середнього, попереднє чи наступне значення. Наприклад,

```
-- Ранжуйте студентів у групі за балом
SELECT StudentID,
       GroupCode,
       Score,
       ROW_NUMBER() OVER (PARTITION BY GroupCode ORDER BY Score DESC)
AS RowNumInGroup
FROM Results;
```

```sql
-- Ранжуйте студентів в межах групи за результатом іспиту
SELECT StudentID,
       GroupCode,
       Score,
       RANK() OVER (PARTITION BY GroupCode ORDER BY Score DESC) AS
ScoreRank
FROM Results;
-- Виведіть результати попереднього та наступного балів студента у
хронології іспитів
SELECT StudentID,
       ExamDate,
       Score,
       LAG(Score) OVER (PARTITION BY StudentID ORDER BY ExamDate) AS
PrevScore,
       LEAD(Score) OVER (PARTITION BY StudentID ORDER BY ExamDate) AS
NextScore
FROM Results;
-- Наведіть загальну суму балів студента після кожного іспиту
SELECT StudentID,
       ExamDate,
       Score,
       SUM(Score) OVER (
           PARTITION BY StudentID
           ORDER BY ExamDate
           ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
       ) AS RunningTotal
FROM Results;
```

Представлення, або VIEW, є збереженим запитом, який виглядає для користувача як логічна таблиця. Представлення не обов'язково зберігає власні дані, у більшості випадків воно лише зберігає визначення запиту, що буде повторно виконуватися під час звернення до цього об'єкта. Такий механізм створює додатковий рівень абстракції між користувачем і фізичною схемою бази даних. Основне призначення представлень полягає в повторному використанні складних запитів, приховуванні зайвих деталей структури, обмеженні доступу до окремих полів або рядків, а також у створенні стабільного інтерфейсу до даних. Якщо фізична структура таблиць змінюється, користувачі можуть і надалі працювати з тим самим представленням, не змінюючи прикладний код, за умови що його зовнішня логіка залишається сталою.

У MSSQL Server створення представлення виконується оператором CREATE VIEW. Базовий синтаксис має вигляд CREATE VIEW schema.ViewName AS SELECT ... FROM ... WHERE ... . Для модифікації

використовується ALTER VIEW, а для видалення — DROP VIEW. Таким чином, представлення є повноцінним об'єктом бази даних, який має власне ім'я й власний опис у схемі. У деяких випадках під час створення представлення можуть використовуватися додаткові параметри. Зокрема, WITH SCHEMABINDING жорстко прив'язує представлення до схеми базових таблиць і забороняє змінювати їх структуру способом, що зробив би представлення некоректним. WITH CHECK OPTION забезпечує, щоб операції INSERT або UPDATE, виконувані через оновлюване представлення, не порушували його фільтр. В окремих СУБД або режимах можуть існувати також матеріалізовані або індексовані представлення, результат яких частково або повністю зберігається для підвищення продуктивності.

Оновлюваність представлення залежить від його структури. Прості представлення, побудовані над однією таблицею без агрегатів, DISTINCT, складних JOIN і операцій над множинами, у багатьох випадках можуть бути оновлюваними. Натомість складні представлення з групуванням, агрегуванням, декількома таблицями чи усуненням дублікатів найчастіше використовуються лише для читання. Наприклад,

```
-- Створення представлення
CREATE VIEW vw_ActiveStudents
AS
SELECT StudentID, FullName, GroupCode
FROM Students
WHERE IsActive = 1;
-- Оновлення представлення
ALTER VIEW vw_ActiveStudents
AS
SELECT StudentID, FullName, GroupCode, Email
FROM Students
WHERE IsActive = 1;
-- Видалення представлення
DROP VIEW vw_ActiveStudents;
```

Постановка задачі лабораторної роботи № 4

При виконанні лабораторної роботи необхідно виконати наступні дії:

1) Створити наступні запити (в запитах повинні використовуватись 2 та більше таблиць):

   a. запит з використанням функцій COUNT або SUM;

b. запит з використанням групування по декільком стовпцям;

c. запит з використанням умови відбору груп HAVING;

d. запит з використанням HAVING без GROUP BY;

e. запит з використанням функцій row_number() over ….;

f. запит, в котрому значення одного зі стовпців таблиці будуть виведені в рядок через кому;

g. запит з використанням сортування по декільком стовпцям в різному порядку;

h. запити згідно варіанту завдання.

2) Робота з представленнями (view):

a. створити представлення з конкретним переліком атрибутів, котрі обираються, та котре містить дані з декількох таблиць;

b. створити представлення, котре містить дані з декількох таблиць та використовує представлення, котре створене в п.а;

c. модифікувати представлення з використанням команди ALTER VIEW;

3) Для кожного з запитів та завдань п.1 та п.2 навести їх словесний (сутнісний) опис та призначення.

4) Оформити звіт з роботи.

Вимоги до оформлення звіту з лабораторної роботи

Звіт повинен містити наступні складові частини:

- титульний лист;
- назву та мету роботи;
- варіант;
- SQL скрипти згідно завдання до лабораторної роботи;
- словесний опис до кожного зі скриптів;
- результати їх виконання;
- висновки.

Контрольні запитання

1) У чому полягає відмінність між логічним і фізичним порядком виконання SQL-запиту?
2) Яка логічна послідовність виконання оператора SELECT?
3) Для чого використовується секція GROUP BY?
4) Які стовпці дозволено вказувати в секції SELECT, якщо в запиті використовується GROUP BY?
5) У чому полягає відмінність між секціями WHERE і HAVING?
6) Для чого призначені агрегатні функції в SQL?
7) Яка різниця між COUNT(*) і COUNT(column_name)?
8) Як поводяться агрегатні функції при наявності значень NULL?
9) Для чого використовується оператор DISTINCT?
10) Яка різниця між UNION і UNION ALL?
11) Для чого застосовується оператор ORDER BY?
12) Чому без ORDER BY порядок рядків у результаті не гарантується?
13) Яке призначення вбудованих числових функцій у SQL?
14) Яке призначення вбудованих рядкових функцій у SQL?
15) Яке призначення функцій дати і часу в SQL?
16) Для чого використовуються функції перетворення типів даних?
17) Яке призначення функцій COALESCE, ISNULL і NULLIF?
18) Що таке аналітичні (віконні) функції в SQL?
19) У чому полягає відмінність між агрегатними та аналітичними функціями?
20) Для чого використовуються конструкції PARTITION BY та ORDER BY у віконних функціях?
21) Яке призначення функцій ROW_NUMBER(), RANK() і DENSE_RANK()?
22) Для чого застосовуються функції LAG() і LEAD()?
23) Що таке представлення (VIEW) у базі даних і яке його призначення?

24) Яка різниця між командами CREATE VIEW, ALTER VIEW і DROP VIEW?

Приклад виконання завдання лабораторної роботи № 4

Розглянемо процес виконання лабораторної роботи для наведеного нижче предметного середовища.

Предметне середовище «Розклад екзаменів».

Система, котра проєктується, повинна зберігати інформацію про студентів, їх групи, предмети, екзамени (по якому предмету, якою групою та коли здається). Для кожної групи повинна існувати можливість отримати розклад екзаменів кожної групи, списки груп та отримані оцінки. Окрім того, один й той самий екзамен можуть здавати одразу декілька груп.

Запити:

а) Визначить студентів певної групи, котрі здають екзамен з дисципліни бази даних в січні поточного року.

б) Визначити дисципліни, по яким минулого року був відсутній екзамен, однак в розкладі поточного року він присутній.

1) Створити наступні запити (в запитах повинні використовуватись 2 та більше таблиць):

   a. запит з використанням функцій COUNT або SUM;

```
-- Визначте по кожній дисципліні кількість студентів, котрі здали дисципліну
та середній бал
SELECT
    MAX(sub.Naim )       AS SubjectName,
    COUNT(gr.EXAM_GRADE) AS GradeCount,
    AVG(gr.EXAM_GRADE)   AS GradeAvg
FROM dbo.Subject sub JOIN dbo.Exam    e   ON e.ID_SUBJECT = sub.ID
                     JOIN dbo.Grades  gr  ON gr.ID_Exam   = e.ID
GROUP BY sub.ID
```

	SubjectName	GradeCount	GradeAvg
1	Бази даних	9	79
2	Програмування	8	78
3	Алгоритми	5	82
4	Операційні системи	4	81
5	Комп'ютерні мережі	4	75
6	Філософія	3	72

b. запит з використанням групування по декільком стовпцям;

```
-- Визначить в розрізі років кількість екзаменів, котрі здає кожна з груп
SELECT year(e.DATE_EXAM), MAX(g.NAIM) AS GroupName, COUNT(e.ID_SUBJECT) AS
ExamsCount
FROM dbo.[Group] g JOIN dbo.GroupSchedule gs ON gs.ID_Group = g.ID
                  JOIN dbo.Exam e            ON e.[ID] = gs.ID_Exam
                  JOIN dbo.Subject sub       ON sub.[ID] = e.ID_SUBJECT
GROUP BY g.ID, year(e.DATE_EXAM)
```

	(No column name)	GroupName	ExamsCount
1	2025	ІП-з51	3
2	2025	ІП-з11	3
3	2025	ІП-з12	2

c. запит з використанням умови відбору груп HAVING;

```
-- Визначить групи, котрі в поточному році здають більше 2 екзаменів
SELECT MAX(g.NAIM) AS GroupName
FROM dbo.[Group] g JOIN dbo.GroupSchedule gs ON gs.ID_Group = g.ID
                            JOIN dbo.Exam e            ON e.[ID] = gs.ID_Exam
                            JOIN dbo.Subject sub      ON sub.[ID] = e.ID_SUBJECT
WHERE YEAR(e.DATE_EXAM) = year(GETDATE())
GROUP BY g.ID, year(e.DATE_EXAM)
HAVING COUNT(e.ID_SUBJECT)>2
```

	GroupName
1	ІП-з51
2	ІП-з11

d. запит з використанням HAVING без GROUP BY;

```
-- Чи існують групи, котрі в поточному році здають 4 та більше екзаменів
SELECT 'Так' AS Відповідь
FROM (
    SELECT COUNT(*) AS Cnt
    FROM (
        SELECT g.ID
        FROM dbo.[Group] g JOIN dbo.GroupSchedule gs ON gs.ID_Group = g.ID
                               JOIN dbo.Exam e            ON e.ID        = gs.ID_Exam
        WHERE YEAR(e.DATE_EXAM) >= YEAR(GETDATE())
        GROUP BY g.ID
        HAVING COUNT(*) >= 4
    ) q
) agg
HAVING SUM(agg.Cnt) >= 1
UNION ALL
SELECT 'Hi'
FROM (
```

```
    SELECT COUNT(*) AS Cnt
    FROM (
        SELECT g.ID
        FROM dbo.[Group] g JOIN dbo.GroupSchedule gs ON gs.ID_Group = g.ID
                           JOIN dbo.Exam e           ON e.ID        = gs.ID_Exam
        WHERE YEAR(e.DATE_EXAM) >= YEAR(GETDATE())
        GROUP BY g.ID
        HAVING COUNT(*) >= 4
    ) q
) agg
HAVING SUM(agg.Cnt) = 0;
```

Results | Messages

	Відповідь
1	Ні

e. запит з використанням функцій row_number() over ….;

```
-- В поточному році для кожної групи визначте послідовність екзаменів
SELECT g.NAIM AS GroupName, sub.Naim AS SubjectName, e.DATE_EXAM AS ExamDate,
    e.ROOM_EXAM AS RoomExam, e.FORM_EXAM  AS FormExam,
    ROW_NUMBER() OVER (PARTITION BY g.ID
        ORDER BY e.DATE_EXAM ) AS SeqInYear
FROM dbo.[Group] AS g JOIN dbo.GroupSchedule  AS gs ON gs.ID_Group = g.ID
                      JOIN dbo.Exam           AS e  ON e.ID        = gs.ID_Exam
                      JOIN dbo.Subject        AS sub ON sub.ID     = e.ID_SUBJECT
WHERE YEAR(e.DATE_EXAM) = YEAR(GETDATE())
ORDER BY g.NAIM, SeqInYear
```

Results | Messages

	GroupName	SubjectName	ExamDate	RoomExam	FormExam	SeqInYear
1	ІП-з11	Бази даних	2025-01-20 10:00:00	101	очна	1
2	ІП-з11	Комп'ютерні мережі	2025-03-03 11:00:00	105	очна	2
3	ІП-з11	Операційні системи	2025-12-22 14:00:00	104	очна	3
4	ІП-з12	Програмування	2025-02-05 09:00:00	102	очна	1
5	ІП-з12	Філософія	2025-05-15 10:00:00	106	очна	2
6	ІП-з51	Бази даних	2025-01-20 10:00:00	101	очна	1
7	ІП-з51	Алгоритми	2025-01-25 12:00:00	103	дистанційна	2
8	ІП-з51	Програмування	2025-02-05 09:00:00	102	очна	3

f. запит, в котрому значення одного зі стовпців таблиці будуть виведені в рядок через кому;

```
-- В поточному році для кожної групи виведіть послідовність екзаменів
-- Варіант 1
SELECT
    g.NAIM AS GroupName,
    STRING_AGG( CASE WHEN e.ID IS NULL THEN NULL
             ELSE CONCAT(CONVERT(varchar(10), e.DATE_EXAM, 23), ' ', sub.Naim)
END, ', '
    ) WITHIN GROUP (ORDER BY e.DATE_EXAM, sub.Naim, e.ID) AS ExamsSequence
FROM dbo.[Group] g LEFT JOIN dbo.GroupSchedule gs ON gs.ID_Group = g.ID
                   LEFT JOIN dbo.Exam e          ON e.ID = gs.ID_Exam
                          AND YEAR(e.DATE_EXAM) = YEAR(GETDATE())
                   LEFT JOIN dbo.Subject sub     ON sub.ID = e.ID_SUBJECT
GROUP BY g.NAIM
ORDER BY g.NAIM;
GO
-- Варіант 2
```

```
SELECT
    g.NAIM AS GroupName,
    STUFF((SELECT ', ' +
               CONVERT(varchar(10), e2.DATE_EXAM, 23) + ' ' + sub2.Naim
        FROM dbo.GroupSchedule gs2 JOIN dbo.Exam    e2   ON e2.ID = gs2.ID_Exam
                              JOIN dbo.Subject sub2 ON sub2.ID = e2.ID_SUBJECT
        WHERE gs2.ID_Group = g.ID AND YEAR(e2.DATE_EXAM) = YEAR(GETDATE())
        ORDER BY e2.DATE_EXAM, sub2.Naim, e2.ID
        FOR  XML  PATH(''),  TYPE).value('.','nvarchar(max)'),  1,  2,  '')  AS
ExamsSequence
FROM dbo.[Group] g
ORDER BY g.NAIM
```

Results | Messages

	GroupName	ExamsSequence
1	ІП-з11	2025-01-20 Бази даних, 2025-03-03 Комп'ютерні мережі, 2025-12-22 Операційні системи
2	ІП-з12	2025-02-05 Програмування, 2025-05-15 Філософія
3	ІП-з51	2025-01-20 Бази даних, 2025-01-25 Алгоритми, 2025-02-05 Програмування

	GroupName	ExamsSequence
1	ІП-з11	2025-01-20 Бази даних, 2025-03-03 Комп'ютерні мережі, 2025-12-22 Операційні системи
2	ІП-з12	2025-02-05 Програмування, 2025-05-15 Філософія
3	ІП-з51	2025-01-20 Бази даних, 2025-01-25 Алгоритми, 2025-02-05 Програмування

g. запит з використанням сортування по декільком стовпцям в різному порядку;

```
-- Відсортуйте екзамени по дисциплінам в порядку спадання за датою
SELECT
    sub.Naim  AS SubjectName,
    e.DATE_EXAM,
    e.ROOM_EXAM,
    e.FORM_EXAM
FROM dbo.Subject sub JOIN dbo.Exam e ON e.ID_SUBJECT = sub.ID
ORDER BY sub.Naim ASC, e.DATE_EXAM DESC
```

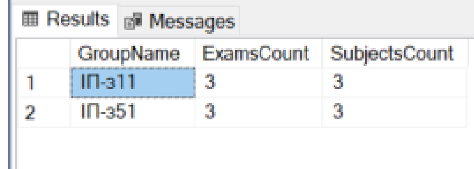
Results | Messages

	SubjectName	DATE_EXAM	ROOM_EXAM	FORM_EXAM
1	Алгоритми	2025-01-25 12:00:00	103	дистанційна
2	Бази даних	2025-01-20 10:00:00	101	очна
3	Бази даних	2024-06-10 10:00:00	201	очна
4	Комп'ютерні мережі	2025-03-03 11:00:00	105	очна
5	Операційні системи	2025-12-22 14:00:00	104	очна
6	Програмування	2025-02-05 09:00:00	102	очна
7	Філософія	2025-05-15 10:00:00	106	очна

h. запити згідно варіанту завдання.

```
-- Визначити групи, у яких в поточному році не менше 3 іспитів та 2 різних
дисциплін
WITH YearExams AS (
    SELECT
        gs.ID_Group,
        e.ID          AS ExamID,
        e.ID_SUBJECT  AS SubjectID
    FROM dbo.GroupSchedule gs JOIN dbo.Exam e ON e.ID = gs.ID_Exam
    WHERE YEAR(e.DATE_EXAM) = YEAR(GETDATE())
```

```
)
SELECT MAX(g.NAIM) AS GroupName,
    COUNT(ye.ExamID) AS ExamsCount,
    COUNT(DISTINCT ye.SubjectID) AS SubjectsCount
FROM dbo.[Group] g JOIN YearExams ye  ON ye.ID_Group = g.ID
GROUP BY g.ID
HAVING COUNT(ye.ExamID) >= 3
   AND COUNT(DISTINCT ye.SubjectID) >= 2
ORDER BY ExamsCount DESC, GroupName;
```



	GroupName	ExamsCount	SubjectsCount
1	ІП-з11	3	3
2	ІП-з51	3	3

```
-- Визначити групи, у яких в поточному році середній бал по дисциплінах більше 75
WITH YearGrades AS (
    SELECT
        gs.ID_Group         AS GroupID,
        e.ID_SUBJECT        AS SubjectID,
        gr.EXAM_GRADE       AS GradeValue
    FROM dbo.Grades gr JOIN dbo.Exam e ON e.ID = gr.ID_Exam
                     JOIN dbo.GroupSchedule AS gs ON gs.ID_Exam = e.ID
    WHERE YEAR(e.DATE_EXAM) = YEAR(GETDATE())
)
SELECT
    g.NAIM AS GroupName,
    sub.Naim AS SubjectName,
    COUNT(*) AS Attempts,
    CAST(AVG(CAST(yg.GradeValue AS decimal(9,2)))  AS decimal(5,2)) AS AvgGrade
FROM YearGrades yg JOIN dbo.[Group] g  ON g.ID = yg.GroupID
              JOIN dbo.Subject sub ON sub.ID = yg.SubjectID
GROUP BY g.NAIM, sub.Naim
HAVING AVG(CAST(yg.GradeValue AS decimal(9,2))) >= 75
ORDER BY GroupName, SubjectName;
```

Results | Messages

	GroupName	SubjectName	Attempts	AvgGrade
1	ІП-з11	Бази даних	9	79.00
2	ІП-з11	Комп'ютерні мережі	4	75.50
3	ІП-з11	Операційні системи	4	81.50
4	ІП-з12	Програмування	8	78.38
5	ІП-з51	Алгоритми	5	82.00
6	ІП-з51	Бази даних	9	79.00
7	ІП-з51	Програмування	8	78.38

2) Робота з представленнями (view):
   a. створити представлення з конкретним переліком атрибутів, котрі обираються, та котре містить дані з декількох таблиць;

```
-- Створити представлення для відображення студентів та екзаменів, котрі вони здають
CREATE OR ALTER VIEW dbo.viewStudentExamSchedule
AS
SELECT
```

```
    s.ID                  AS StudentID,
    s.Surname,
    s.Naim                AS Name,
    g.ID                  AS GroupID,
    g.NAIM                AS GroupName,
    sub.ID                AS SubjectID,
    sub.Naim              AS SubjectName,
    e.ID                  AS ExamID,
    e.DATE_EXAM           AS ExamDate,
    e.ROOM_EXAM           AS ExamRoom,
    e.FORM_EXAM           AS ExamForm
FROM dbo.Student s JOIN dbo.[Group] g          ON g.ID = s.ID_Group
                        JOIN dbo.GroupSchedule gs ON gs.ID_Group = g.ID
                        JOIN dbo.Exam e           ON e.ID = gs.ID_Exam
                        JOIN dbo.Subject sub      ON sub.ID = e.ID_SUBJECT

select * from dbo.viewStudentExamSchedule
```

Results | Messages

	StudentID	Surname	Name	GroupID	GroupName	SubjectID	SubjectName	ExamID	ExamDate	ExamRoom	ExamForm
7	27	Іваненко	Дмитро	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	101	очна
8	27	Іваненко	Дмитро	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна
9	27	Іваненко	Дмитро	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна
10	28	Шевченко	Ірина	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	101	очна
11	28	Шевченко	Ірина	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна
12	28	Шевченко	Ірина	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна
13	29	Бондар	Тарас	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	101	очна
14	29	Бондар	Тарас	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна
15	29	Бондар	Тарас	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна
16	30	Мельник	Олег	10	ІП-з11	13	Бази даних	15	2025-01-20 10:00:00	101	очна
17	30	Мельник	Олег	10	ІП-з11	16	Операційні системи	19	2025-12-22 14:00:00	104	очна
18	30	Мельник	Олег	10	ІП-з11	17	Комп'ютерні мережі	20	2025-03-03 11:00:00	105	очна
19	31	Романюк	Наталія	10	ІП-з11	13	Бази даних	15	2025-01-20 10:00:00	101	очна
20	31	Романюк	Наталія	10	ІП-з11	16	Операційні системи	19	2025-12-22 14:00:00	104	очна
21	31	Романюк	Наталія	10	ІП-з11	17	Комп'ютерні мережі	20	2025-03-03 11:00:00	105	очна
22	32	Сидоренко	Андрій	10	ІП-з11	13	Бази даних	15	2025-01-20 10:00:00	101	очна
23	32	Сидоренко	Андрій	10	ІП-з11	16	Операційні системи	19	2025-12-22 14:00:00	104	очна
24	32	Сидоренко	Андрій	10	ІП-з11	17	Комп'ютерні мережі	20	2025-03-03 11:00:00	105	очна

b. створити представлення, котре містить дані з декількох таблиць та використовує представлення, котре створене в п.а;

```
-- Створити представлення для відображення студентів та екзаменів, котрі вони здають та отримані ними оцінки
CREATE OR ALTER VIEW dbo.viewStudentExamGrades
AS
SELECT
    v.StudentID,
    v.Surname,
    v.Name,
    v.GroupID,
    v.GroupName,
    v.SubjectID,
    v.SubjectName,
    v.ExamID,
    v.ExamDate,
    v.ExamRoom,
    v.ExamForm,
    gr.EXAM_GRADE
FROM dbo.viewStudentExamSchedule v LEFT JOIN dbo.Grades gr ON gr.ID_Student = v.StudentID
                                                            AND gr.ID_Exam = v.ExamID
```

```
select * from dbo.viewStudentExamGrades
```

90 %

Results | Messages

	StudentID	Surname	Name	GroupID	GroupName	SubjectID	SubjectName	ExamID	ExamDate	ExamRoom	ExamForm	EXAM_GRADE
4	26	Коваленко	Марія	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	101	очна	76
5	26	Коваленко	Марія	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна	86
6	26	Коваленко	Марія	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна	81
7	27	Іваненко	Дмитро	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	101	очна	77
8	27	Іваненко	Дмитро	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна	87
9	27	Іваненко	Дмитро	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна	82
10	28	Шевченко	Ірина	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	101	очна	78
11	28	Шевченко	Ірина	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна	88
12	28	Шевченко	Ірина	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна	83
13	29	Бондар	Тарас	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	101	очна	79
14	29	Бондар	Тарас	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна	89
15	29	Бондар	Тарас	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна	84
16	30	Мельник	Олег	10	ІП-з11	13	Бази даних	15	2025-01-20 10:00:00	101	очна	80
17	30	Мельник	Олег	10	ІП-з11	16	Операційні системи	19	2025-12-22 14:00:00	104	очна	80
18	30	Мельник	Олег	10	ІП-з11	17	Комп'ютерні мережі	20	2025-03-03 11:00:00	105	очна	74
19	31	Романюк	Наталія	10	ІП-з11	13	Бази даних	15	2025-01-20 10:00:00	101	очна	81
20	31	Романюк	Наталія	10	ІП-з11	16	Операційні системи	19	2025-12-22 14:00:00	104	очна	81
21	31	Романюк	Наталія	10	ІП-з11	17	Комп'ютерні мережі	20	2025-03-03 11:00:00	105	очна	75
22	32	Сидоренко	Андрій	10	ІП-з11	13	Бази даних	15	2025-01-20 10:00:00	101	очна	82

c. модифікувати представлення з використанням команди ALTER VIEW;

```
-- В представлення, котре відображає студентів та екзамени, котрі вони здають
додати рік здачі екзамену
ALTER VIEW dbo.viewStudentExamSchedule
AS
SELECT
    s.ID                 AS StudentID,
    s.Surname,
    s.Naim               AS Name,
    g.ID                 AS GroupID,
    g.NAIM               AS GroupName,
    sub.ID               AS SubjectID,
    sub.Naim             AS SubjectName,
    e.ID                 AS ExamID,
    e.DATE_EXAM          AS ExamDate,
    e.ROOM_EXAM          AS ExamRoom,
    e.FORM_EXAM          AS ExamForm,
    year(e.DATE_EXAM)    AS ExamYear
FROM dbo.Student s JOIN dbo.[Group] g          ON g.ID = s.ID_Group
                   JOIN dbo.GroupSchedule gs ON gs.ID_Group = g.ID
                   JOIN dbo.Exam e           ON e.ID = gs.ID_Exam
                   JOIN dbo.Subject sub      ON sub.ID = e.ID_SUBJECT
```

```
select * from dbo.viewStudentExamSchedule
```

90 %

Results | Messages

	StudentID	Surname	Name	GroupID	GroupName	SubjectID	SubjectName	ExamID	ExamDate	ExamRoom	ExamForm	ExamYear
4	26	Коваленко	Марія	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	101	очна	2025
5	26	Коваленко	Марія	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна	2025
6	26	Коваленко	Марія	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна	2025
7	27	Іваненко	Дмитро	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	101	очна	2025
8	27	Іваненко	Дмитро	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна	2025
9	27	Іваненко	Дмитро	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна	2025
10	28	Шевченко	Ірина	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	101	очна	2025
11	28	Шевченко	Ірина	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна	2025
12	28	Шевченко	Ірина	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна	2025
13	29	Бондар	Тарас	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	101	очна	2025
14	29	Бондар	Тарас	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна	2025
15	29	Бондар	Тарас	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна	2025
16	30	Мельник	Олег	10	ІП-з11	13	Бази даних	15	2025-01-20 10:00:00	101	очна	2025
17	30	Мельник	Олег	10	ІП-з11	16	Операційні системи	19	2025-12-22 14:00:00	104	очна	2025
18	30	Мельник	Олег	10	ІП-з11	17	Комп'ютерні мережі	20	2025-03-03 11:00:00	105	очна	2025

ЛАБОРАТОРНА РОБОТА № 5. ОСНОВИ ПРОГРАМУВАННЯ З ВИКОРИСТАННЯМ МОВИ SQL. ЗБЕРЕЖЕНІ ПРОЦЕДУРИ. КУРСОРИ. ТРИГЕРИ.

Мета:

– вивчити правила побудови ідентифікаторів, правила визначення змінних та типів. Визначити правила роботи з циклами та умовними конструкціями, роботу зі змінними типу Table;

– вивчити синтаксис та семантику функцій та збережених процедур, способів їх ідентифікації, методів визначення та специфікації параметрів та значень, котрі повертаються, виклик функцій та збережених процедур;

– застосування команд для створення, зміни та видалення як скалярних, так і табличних функцій, збережених процедур;

– вивчити призначення та типи курсорів, синтаксис та семантику команд мови SQL для створення курсорів, вибірки даних з курсорів, зміни даних із застосуванням курсорів;

– вивчити призначення та типи тригерів, умов їх активації, синтаксису та семантики для їх створення, модифікації, перейменування, програмування та видалення.

Короткі теоретичні відомості

У сучасних реляційних системах управління базами даних мова SQL використовується не лише як засіб створення таблиць і формування запитів на вибірку даних, а й як основа для реалізації програмної логіки безпосередньо в середовищі СУБД. Такий підхід дає змогу переносити частину бізнес-правил на рівень бази даних, централізувати обробку інформації, забезпечити єдині правила доступу до даних, зменшити дублювання коду в прикладних програмах і підвищити контрольованість роботи з даними. У MSSQL Server для цього використовується розширення мови SQL — Transact-SQL (T-SQL), котре підтримує змінні, умовні конструкції, цикли, табличні змінні, тимчасові

таблиці, користувацькі функції, збережені процедури, курсори, тригери, транзакції та обробку помилок [11].

У програмному коді T-SQL активно використовуються локальні змінні. Змінна — це іменований контейнер для тимчасового зберігання значення всередині пакета команд, процедури або функції. Ім’я локальної змінної в SQL Server починається із символу @. Загальний шаблон оголошення змінної має вигляд:

```
DECLARE @variable_name data_type [= initial_value];
```

де @variable_name — ім’я змінної, data_type — тип даних, а initial_value — необов’язкове початкове значення. Наприклад:

```
DECLARE @StudentCount INT;
DECLARE @AverageScore DECIMAL(5,2);
DECLARE @GroupCode NVARCHAR(20) = N'ІП-31';
```

Значення змінній можна надати за допомогою SET або SELECT:

```
DECLARE @StudentCount INT;

SELECT @StudentCount = COUNT(*)
FROM Students
WHERE GroupCode = N'ІП-31';

SELECT @StudentCount AS StudentCount;
```

У цьому прикладі змінна @StudentCount використовується для збереження кількості студентів певної групи. Такий підхід є зручним у процедурах, коли проміжний результат потрібно використати в подальшій логіці. Змінні мають область видимості. Вони існують лише в межах пакета команд, процедури або функції, де були оголошені, і не є постійними об’єктами бази даних. Після завершення виконання відповідного блоку змінні зникають. Особливим різновидом змінних є табличні змінні. Вони дозволяють тимчасово зберігати набір рядків у структурі, подібній до таблиці. Загальний шаблон оголошення табличної змінної має вигляд:

```
DECLARE @table_variable TABLE
(
    column_name data_type [column_constraint],
    ...
);
```

Наприклад:

```
DECLARE @TopStudents TABLE
(
    StudentID INT,
```

```
    FullName NVARCHAR(100),
    Score INT
);

INSERT INTO @TopStudents (StudentID, FullName, Score)
SELECT s.StudentID, s.FullName, r.Score
FROM Students AS s JOIN Results AS r
    ON r.StudentID = s.StudentID
WHERE r.Score >= 90;

SELECT *
FROM @TopStudents;
```

Табличні змінні зручні для невеликих локальних наборів даних усередині процедур або пакетів команд. Водночас для великих проміжних результатів доцільніше використовувати тимчасові таблиці, оскільки вони можуть мати індекси, статистики та часто краще оптимізуються. Тимчасові таблиці створюються в системній базі tempdb і позначаються символом # для локальних тимчасових таблиць. Загальний шаблон створення локальної тимчасової таблиці має вигляд:

```
CREATE TABLE #temp_table_name
(
    column_name data_type [constraint],
    ...
);
```

Наприклад:

```
CREATE TABLE #GroupStatistics
(
    GroupCode NVARCHAR(20),
    StudentCount INT,
    AvgScore DECIMAL(5,2)
);

INSERT INTO #GroupStatistics (GroupCode, StudentCount, AvgScore)
SELECT s.GroupCode,
       COUNT(*) AS StudentCount,
       AVG(CAST(r.Score AS DECIMAL(5,2))) AS AvgScore
FROM Students AS s JOIN Results AS r
    ON r.StudentID = s.StudentID
GROUP BY s.GroupCode;

SELECT *
FROM #GroupStatistics;

DROP TABLE #GroupStatistics;
```

Тимчасова таблиця є доцільною тоді, коли проміжний результат має значний обсяг, використовується кілька разів або потребує індексації.

У SQL Server також можна передавати в процедуру цілий набір рядків за допомогою табличного параметра. Для цього спочатку створюється користувацький табличний тип:

```
CREATE TYPE schema_name.type_name AS TABLE
(
    column_name data_type [constraint],
    ...
);
```

Наприклад:

```
CREATE TYPE StudentIdList AS TABLE
(
    StudentID INT PRIMARY KEY
);
```

Після цього такий тип може використовуватися як параметр процедури:

```
CREATE PROCEDURE usp_GetStudentsByList
    @Students StudentIdList READONLY
AS
BEGIN
    SELECT s.StudentID, s.FullName, s.GroupCode
    FROM Students AS s JOIN @Students AS p
        ON p.StudentID = s.StudentID;
END;
```

Ключове слово READONLY є обов'язковим для табличних параметрів у SQL Server. Це означає, що процедура може читати переданий набір, але не може змінювати його вміст. Табличні параметри зручні для пакетної передачі даних із прикладної програми до СУБД.

Попри те, що SQL загалом є декларативною мовою, T-SQL підтримує імперативні конструкції, зокрема IF ... ELSE, CASE, WHILE, TRY ... CATCH. Умовна конструкція IF ... ELSE використовується тоді, коли потрібно виконати різні дії залежно від результату логічної умови. В загальному вигляді використання умовних конструкцій можна представити у наступному вигляді:

```
IF <condition>
BEGIN
    -- команди, які виконуються, якщо умова істинна
END
ELSE
BEGIN
    -- команди, які виконуються, якщо умова хибна
END;
```

Наприклад:

```
DECLARE @AvgScore DECIMAL(5,2);

SELECT @AvgScore = AVG(CAST(Score AS DECIMAL(5,2)))
```

```
FROM Results
WHERE StudentID = 1;

IF @AvgScore >= 60
BEGIN
    PRINT N'Студент має позитивний результат';
END
ELSE
BEGIN
    PRINT N'Студент не набрав мінімальний бал';
END;
```

Якщо потрібно не змінювати хід виконання програми, а сформувати умовне значення всередині запиту, використовується вираз CASE. В загальному можна представити наступним чином:

```
CASE
    WHEN <condition_1> THEN <result_1>
    WHEN <condition_2> THEN <result_2>
    ELSE <default_result>
END
```

Наприклад:

```
SELECT StudentID,
       Score,
       CASE
           WHEN Score >= 90 THEN N'Відмінно'
           WHEN Score >= 75 THEN N'Добре'
           WHEN Score >= 60 THEN N'Задовільно'
           ELSE N'Незадовільно'
       END AS GradeText
FROM Results;
```

Конструкція CASE часто використовується для класифікації значень, формування статусів і побудови обчислюваних стовпців.

Для повторення команд у T-SQL використовується цикл WHILE, котрий має наступний вигляд:

```
WHILE <condition>
BEGIN
    -- команди циклу
END;
```

Наприклад:

```
DECLARE @i INT = 1;

WHILE @i <= 5
BEGIN
    PRINT CONCAT(N'Поточне значення: ', @i);
    SET @i = @i + 1;
END;
```

Разом із тим у реляційних базах даних цикли слід використовувати обережно. SQL природно орієнтований на обробку наборів рядків, тому

подібну задачу краще розв’язувати одним оператором UPDATE, MERGE, запитом із групуванням або віконною функцією, такий підхід зазвичай є ефективнішим. Цикли доцільні тоді, коли сама постановка задачі вимагає послідовності дій.

Для обробки помилок у T-SQL використовується конструкція TRY ... CATCH, котрий має наступний вигляд:

```
BEGIN TRY
    -- команди, які можуть спричинити помилку
END TRY
BEGIN CATCH
    -- команди обробки помилки
END CATCH;
```

Найчастіше TRY ... CATCH застосовується разом із транзакціями:

```
BEGIN TRY
    BEGIN TRANSACTION;

    UPDATE Students
    SET GroupCode = N'ІП-32'
    WHERE StudentID = 1;

    COMMIT TRANSACTION;
END TRY
BEGIN CATCH
    IF @@TRANCOUNT > 0
        ROLLBACK TRANSACTION;

    THROW;
END CATCH;
```

Значне місце у програмуванні в SQL посідають функції. Функція — це іменований програмний об’єкт, який приймає параметри й повертає результат. У SQL розрізняють скалярні функції, вбудовані табличні функції та багатокрокові табличні функції. Скалярна функція повертає одне значення. Вбудована таблична функція повертає таблицю і має тіло у вигляді одного запиту SELECT. Багатокрокова таблична функція дозволяє виконувати кілька команд усередині тіла функції, але може бути менш ефективною для великих наборів даних.

В загальному вигляді шаблон скалярної функції має наступний вигляд:

```
CREATE FUNCTION schema_name.function_name
(
    @parameter_name data_type
)
RETURNS return_data_type
```

```
AS
BEGIN
    -- оголошення змінних та обчислення
    RETURN <scalar_value>;
END;
```

Наприклад:

```
CREATE FUNCTION fn_GetGradeText
(
    @Score INT
)
RETURNS NVARCHAR(20)
AS
BEGIN
    DECLARE @Result NVARCHAR(20);

    SET @Result =
        CASE
            WHEN @Score >= 90 THEN N'Відмінно'
            WHEN @Score >= 75 THEN N'Добре'
            WHEN @Score >= 60 THEN N'Задовільно'
            ELSE N'Незадовільно'
        END;

    RETURN @Result;
END;
```

Виклик скалярної функції може виконуватися безпосередньо в запиті:

```
SELECT StudentID,
       Score,
       fn_GetGradeText(Score) AS GradeText
FROM Results;
```

Скалярна функція є доцільною тоді, коли потрібно багаторазово використовувати однакове обчислення в різних запитах. Водночас складні функції можуть впливати на продуктивність, тому їх доцільність слід оцінювати з урахуванням обсягів даних і плану виконання.

Вбудована таблична функція повертає результат у вигляді таблиці й має тіло у формі одного запиту SELECT та має наступний вигляд:

```
CREATE FUNCTION schema_name.function_name
(
    @parameter_name data_type
)
RETURNS TABLE
AS
RETURN
(
    SELECT ...
);
```

Наприклад:

```
CREATE FUNCTION fn_GetGroupResults
(
    @GroupCode NVARCHAR(20)
)
RETURNS TABLE
AS
RETURN
(
    SELECT s.StudentID,
           s.FullName,
           r.CourseID,
           r.Score
    FROM Students AS s JOIN Results AS r
        ON r.StudentID = s.StudentID
    WHERE s.GroupCode = @GroupCode
);
```

Таку функцію можна використовувати як джерело даних:

```
SELECT *
FROM fn_GetGroupResults(N'ІП-31');
```

Вбудовані табличні функції зручні для параметризованих вибірок і добре поєднуються з іншими запитами, оскільки їхнє тіло фактично є табличним виразом.

Багатокрокова таблична функція дозволяє виконувати кілька команд усередині тіла функції та має наступний вигляд:

```
CREATE FUNCTION schema_name.function_name
(
    @parameter_name data_type
)
RETURNS @result TABLE
(
    column_name data_type,
    ...
)
AS
BEGIN
    INSERT INTO @result
    SELECT ...;

    RETURN;
END;
```

Такий тип функцій є гнучкішим, але може бути менш ефективним для великих наборів даних, оскільки оптимізатору складніше оцінити проміжний результат.

Для зміни функції використовується ALTER FUNCTION, а для видалення — DROP FUNCTION:

```
ALTER FUNCTION fn_GetGradeText
(
    @Score INT
)
RETURNS NVARCHAR(20)
AS
BEGIN
    RETURN
        CASE
            WHEN @Score >= 90 THEN N'Відмінно'
            WHEN @Score >= 75 THEN N'Добре'
            WHEN @Score >= 60 THEN N'Задовільно'
            ELSE N'Незадовільно'
        END;
END;
DROP FUNCTION fn_GetGradeText;
```

Процедура — це іменований сценарій дій, який може читати, змінювати, створювати та видаляти дані, керувати транзакціями, викликати інші процедури та реалізовувати складну бізнес-логіку. Якщо функція є формулою, то процедура є послідовністю дій.

В загальному вигляді створення процедури можна представити наступним чином:

```
CREATE PROCEDURE schema_name.procedure_name
    @parameter_name data_type [= default_value] [OUTPUT],
    ...
AS
BEGIN
    SET NOCOUNT ON;

    -- тіло процедури
END;
```

Параметри процедури можуть бути вхідними або вихідними. Вихідні параметри позначаються ключовим словом OUTPUT. Наприклад:

```
CREATE PROCEDURE usp_AddStudent
    @FullName NVARCHAR(100),
    @GroupCode NVARCHAR(20),
    @NewStudentID INT OUTPUT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO Students (FullName, GroupCode)
    VALUES (@FullName, @GroupCode);

    SET @NewStudentID = SCOPE_IDENTITY();
END;
```

Виклик процедури з вихідним параметром має вигляд:

```
DECLARE @Id INT;

EXEC usp_AddStudent
    @FullName = N'Петренко Олег',
    @GroupCode = N'ІП-31',
    @NewStudentID = @Id OUTPUT;

SELECT @Id AS NewStudentID;
```

Ключове слово SET NOCOUNT ON використовується для вимкнення службових повідомлень про кількість оброблених рядків. Процедура може містити транзакцію й обробку помилок:

```
CREATE PROCEDURE usp_UpdateStudentGroup
    @StudentID INT,
    @NewGroupCode NVARCHAR(20)
AS
BEGIN
    SET NOCOUNT ON;

    BEGIN TRY
        BEGIN TRANSACTION;

        UPDATE Students
        SET GroupCode = @NewGroupCode
        WHERE StudentID = @StudentID;

        IF @@ROWCOUNT = 0
            THROW 50001, N'Студента з таким ідентифікатором не
знайдено.', 1;

        COMMIT TRANSACTION;
    END TRY
    BEGIN CATCH
        IF @@TRANCOUNT > 0
            ROLLBACK TRANSACTION;

        THROW;
    END CATCH
END;
```

Для модифікації процедури використовується ALTER PROCEDURE, а для видалення — DROP PROCEDURE:

```
ALTER PROCEDURE usp_UpdateStudentGroup
    @StudentID INT,
    @NewGroupCode NVARCHAR(20)
AS
BEGIN
    SET NOCOUNT ON;

    UPDATE Students
    SET GroupCode = @NewGroupCode
    WHERE StudentID = @StudentID;
END;
DROP PROCEDURE usp_UpdateStudentGroup;
```

Курсор — це механізм послідовного обходу результату запиту. У реляційних СУБД курсори слід застосовувати обережно. Курсори можуть бути корисними там, де обробка повинна виконуватися послідовно.

Загальний шаблон роботи з курсором має наступний вигляд:

```
DECLARE cursor_name CURSOR [LOCAL | GLOBAL]
    [FORWARD_ONLY | SCROLL]
    [STATIC | KEYSET | DYNAMIC | FAST_FORWARD]
    [READ_ONLY | SCROLL_LOCKS | OPTIMISTIC]
FOR
    SELECT ...;

OPEN cursor_name;

FETCH NEXT FROM cursor_name INTO @variable_list;

WHILE @@FETCH_STATUS = 0
BEGIN
    -- обробка поточного рядка

    FETCH NEXT FROM cursor_name INTO @variable_list;
END;

CLOSE cursor_name;
DEALLOCATE cursor_name;
```

Наприклад:

```
DECLARE @StudentID INT;
DECLARE @FullName NVARCHAR(100);

DECLARE student_cursor CURSOR LOCAL FAST_FORWARD FOR
SELECT StudentID, FullName
FROM Students
WHERE IsActive = 1;

OPEN student_cursor;

FETCH NEXT FROM student_cursor INTO @StudentID, @FullName;

WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT CONCAT(N'Обробляється студент: ', @StudentID, N' ',
@FullName);

    FETCH NEXT FROM student_cursor INTO @StudentID, @FullName;
END;

CLOSE student_cursor;
DEALLOCATE student_cursor;
```

У цьому прикладі курсор послідовно проходить активних студентів. Тип FAST_FORWARD означає, що курсор призначений для проходження лише

вперед і лише для читання. Після завершення роботи курсор обов’язково потрібно закрити командою CLOSE і звільнити командою DEALLOCATE.

Тригер — це спеціальний програмний об’єкт, який автоматично виконується у відповідь на певну подію. У SQL Server найчастіше використовуються DML-тригери, які реагують на операції INSERT, UPDATE, DELETE. Також існують DDL-тригери, пов’язані зі зміною структури бази даних, і логон-тригери, які спрацьовують під час входу на сервер.

В загальному DML-тригер має вигляд:

```
CREATE TRIGGER schema_name.trigger_name
ON schema_name.table_name
AFTER INSERT | UPDATE | DELETE
AS
BEGIN
    SET NOCOUNT ON;

    -- тіло тригера
END;
```

AFTER-тригер виконується після успішного виконання відповідної операції. Наприклад, тригер для журналювання додавання нових студентів може мати наступний вигляд:

```
CREATE TABLE StudentAudit
(
    AuditID INT IDENTITY(1,1) PRIMARY KEY,
    StudentID INT NOT NULL,
    ActionType NVARCHAR(20) NOT NULL,
    ActionDate DATETIME2 NOT NULL DEFAULT SYSDATETIME()
);
CREATE TRIGGER trg_Students_Insert_Audit
ON Students
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO StudentAudit (StudentID, ActionType)
    SELECT StudentID, N'INSERT'
    FROM inserted;
END;
```

Ключовою особливістю DML-тригерів є використання псевдотаблиць inserted і deleted. Таблиця inserted містить нові рядки, які були додані або отримані після оновлення. Таблиця deleted містить старі рядки, які були видалені або існували до оновлення. Важливо розуміти, що ці псевдотаблиці можуть містити не один, а багато рядків.

Тригер INSTEAD OF виконується замість стандартної дії і в загальному має наступний вигляд:

```
CREATE TRIGGER schema_name.trigger_name
ON schema_name.table_or_view_name
INSTEAD OF INSERT | UPDATE | DELETE
AS
BEGIN
    SET NOCOUNT ON;

    -- альтернативна логіка виконання операції
END;
```

Наприклад:

```
CREATE TRIGGER trg_Students_Delete_Prevent
ON Students
INSTEAD OF DELETE
AS
BEGIN
    SET NOCOUNT ON;

    IF EXISTS
    (
        SELECT 1
        FROM deleted AS d
        JOIN Results AS r
            ON r.StudentID = d.StudentID
    )
    BEGIN
        THROW 50002, N'Неможливо видалити студента, для якого існують
результати навчання.', 1;
    END;

    DELETE s
    FROM Students AS s
    JOIN deleted AS d
        ON d.StudentID = s.StudentID;
END;
```

Для аудиту оновлень можна використати AFTER UPDATE-тригер:

```
CREATE TRIGGER trg_Results_Update_Audit
ON Results
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO ResultAudit
    (
        StudentID,
        OldScore,
        NewScore,
        ChangedAt
    )
    SELECT d.StudentID,
           d.Score AS OldScore,
```

```
            i.Score AS NewScore,
            SYSDATETIME()
    FROM deleted AS d
    JOIN inserted AS i
        ON i.ResultID = d.ResultID
    WHERE ISNULL(d.Score, -1) <> ISNULL(i.Score, -1);
END;
```

Для зміни тригера використовується ALTER TRIGGER, а для видалення — DROP TRIGGER:

```
ALTER TRIGGER trg_Students_Insert_Audit
ON Students
AFTER INSERT
AS
BEGIN
    SET NOCOUNT ON;

    INSERT INTO StudentAudit (StudentID, ActionType)
    SELECT StudentID, N'INSERT'
    FROM inserted;
END;
DROP TRIGGER trg_Students_Insert_Audit;
```

Необхідно зазначити, що тригери виконуються в межах транзакції, яка змінила дані. Тому в них не слід виконувати тривалі зовнішні виклики або складні операції, що можуть затримати завершення транзакції та утримувати блокування. Якщо правило цілісності можна виразити через FOREIGN KEY, CHECK, UNIQUE або інше декларативне обмеження, перевагу слід надавати саме обмеженням. Тригери доцільно використовувати для складних бізнес-правил, аудиту, журналювання або спеціальних сценаріїв, які неможливо реалізувати простішими засобами.

Постановка задачі лабораторної робота № 5

При виконанні лабораторної роботи необхідно виконати наступні дії:

1) Збережені процедури:

– створення процедури з використанням умовної конструкції IF та WHILE;

– створення процедури без параметрів;

– створення процедури з вхідним параметром та RETURN;

– створення процедури оновлення даних в деякій таблиці БД;

2) Функції:

– створити функцію, котра повертає деяке скалярне значення;

– створити функцію, котра повертає таблицю наперед заданої структури.

3) Робота з курсорами (створити процедуру, в котрій демонструються наведені нижче дії):

– створення курсору;

– відкриття курсору;

– вибірка даних;

– робота з курсорами.

4) Робота з тригерами:

– створити тригер, котрий буде спрацьовувати при видаленні даних;

– створити тригер, котрий буде спрацьовувати при модифікації даних;

– створити тригер, котрий буде спрацьовувати при додаванні даних.

5) Оформити звіт з роботи. В звіт включити тексти кодів збережених процедур, функцій, тригерів, їх словесний опис та результати виконання.

Вимоги до оформлення звіту з лабораторної роботи

Звіт повинен містити наступні складові частини:

- титульний лист;
- назву та мету роботи;
- варіант;
- SQL скрипти згідно завдання до лабораторної роботи;
- словесний опис до кожного зі скриптів;
- результати їх виконання;
- висновки.

Контрольні запитання

1) Що таке локальна змінна в T-SQL і як вона оголошується?

2) Яка різниця між присвоєнням значення змінній за допомогою SET і SELECT?

3) Що таке таблична змінна і в яких випадках її доцільно використовувати?

4) Чим таблична змінна відрізняється від тимчасової таблиці?

5) Для чого використовуються табличні параметри процедур?

6) Для чого використовується конструкція IF ... ELSE?

7) У яких випадках доцільно використовувати вираз CASE?

8) Для чого застосовується цикл WHILE у T-SQL?

9) Чому в SQL бажано надавати перевагу набірно-орієнтованій обробці замість циклів?

10) Для чого в процедурах використовуються транзакції?

11) Що таке користувацька функція в SQL Server і які основні типи функцій існують?

12) Чим скалярна функція відрізняється від табличної функції?

13) Що таке збережена процедура і чим вона відрізняється від функції?

14) Для чого використовуються вхідні та вихідні параметри збережених процедур?

15) Що таке курсор і в яких випадках його застосування може бути виправданим?

16) Що таке тригер, які типи тригерів існують у SQL Server і для чого використовуються псевдотаблиці inserted та deleted?

Приклад виконання завдання лабораторної роботи № 5

Розглянемо процес виконання лабораторної роботи для наведеного нижче предметного середовища.

Предметне середовище «Розклад екзаменів».

Система, котра проєктується, повинна зберігати інформацію про студентів, їх групи, предмети, екзамени (по якому предмету, якою групою та коли здається). Для кожної групи повинна існувати можливість отримати розклад екзаменів кожної групи, списки груп та отримані оцінки. Окрім того, один й той самий екзамен можуть здавати одразу декілька груп.

1) Збережені процедури:
   a. створення процедури з використанням умовної конструкції IF та WHILE;

```
-- Додавання екзамену для певного предмету в певному році
CREATE OR ALTER PROCEDURE dbo.IsExistsMinExams
    @SubjectID int,
    @DateExam datetime2,
    @MinExams int = 1,
      @Room varchar(20)
AS
BEGIN
      -- Скільки вже є іспитів у заданому році для предмета
    DECLARE @haveExams int =
    (
        SELECT COUNT(*)
        FROM dbo.Exam e
        WHERE e.ID_SUBJECT = @SubjectID
          AND YEAR(DATE_EXAM) =YEAR(@DateExam)
    );
    -- Якщо вже достатньо — нічого не робимо
    IF @haveExams >= @MinExams
        RETURN;
    -- Перевіряємо чи випадково немає інших екзаменів на вказану дату у
вказаній аудиторії
    IF EXISTS (
            SELECT 1
            FROM dbo.Exam
            WHERE ID_SUBJECT = @SubjectID
              AND DATE_EXAM  = @DateExam
              AND ROOM_EXAM  = @Room
        )
            RETURN;

    -- Інакше додаємо відсутні записи у циклі
    DECLARE @i int = 0;
    WHILE (@haveExams < @MinExams)
    BEGIN
        INSERT INTO dbo.Exam (DATE_EXAM, ROOM_EXAM, FORM_EXAM, ID_SUBJECT)
            VALUES (@DateExam, @Room, N'очна', @SubjectID);

        SET @haveExams = @haveExams + 1;
            SET @i = @i + 1;
```

```
    END
END
GO
```

Маємо:

	ID	DATE_EXAM	ROOM_EXAM	FORM_EXAM	ID_SUBJECT
1	15	2025-01-20 10:00:00	101	очна	13
2	16	2024-06-10 10:00:00	201	очна	13
3	17	2025-02-05 09:00:00	102	очна	14
4	18	2025-01-25 12:00:00	103	дистанційна	15
5	19	2025-12-22 14:00:00	104	очна	16
6	20	2025-03-03 11:00:00	105	очна	17
7	21	2025-05-15 10:00:00	106	очна	18

Виконуємо:

```
EXEC dbo.IsExistsMinExams @SubjectID = 13, @DateExam = '20250115', @MinExams
= 3, @Room = '433-18';
```

Отримаємо:

	ID	DATE_EXAM	ROOM_EXAM	FORM_EXAM	ID_SUBJECT
1	15	2025-01-20 10:00:00	101	очна	13
2	16	2024-06-10 10:00:00	201	очна	13
3	17	2025-02-05 09:00:00	102	очна	14
4	18	2025-01-25 12:00:00	103	дистанційна	15
5	19	2025-12-22 14:00:00	104	очна	16
6	20	2025-03-03 11:00:00	105	очна	17
7	21	2025-05-15 10:00:00	106	очна	18
8	26	2025-01-15 00:00:00	433-18	очна	13
9	27	2025-01-15 00:00:00	433-18	очна	13

b. створення процедури без параметрів;

```
-- Визначити кількість екзаменів по дисципліні загалом і окремо в поточному
році
CREATE OR ALTER PROCEDURE dbo.Report_SubjectExamSummary
AS
BEGIN
    DECLARE @ThisYear int = YEAR(GETDATE());

    SELECT
        s.ID          AS SubjectID,
        s.Naim        AS SubjectName,
        TotalExams    = COUNT(e.ID),
        ExamsThisYear = SUM(CASE WHEN YEAR(e.DATE_EXAM) = @ThisYear THEN 1
ELSE 0 END)
    FROM dbo.Subject s LEFT JOIN dbo.Exam e ON e.ID_SUBJECT = s.ID
    GROUP BY s.ID, s.Naim
    ORDER BY s.Naim;
END
GO
```

Виконуємо:

```
EXEC dbo.Report_SubjectExamSummary
```

Отримаємо:

	SubjectID	SubjectName	TotalExams	ExamsThisYear
1	15	Алгоритми	1	1
2	13	Бази даних	4	3
3	17	Комп'ютерні мережі	1	1
4	16	Операційні системи	1	1
5	14	Програмування	1	1
6	18	Філософія	1	1

c. створення процедури з вхідним параметром та RETURN;

```
-- Визначити кількість студентів в певній групі
CREATE OR ALTER PROCEDURE dbo.GetGroupStudentCount
    @GroupID int
AS
BEGIN
    IF @GroupID IS NULL
    BEGIN
        RAISERROR(N'Потрібен @GroupID', 16, 1);
        RETURN -1;
    END
    DECLARE @cnt int = (SELECT COUNT(*) FROM dbo.Student WHERE ID_Group =
@GroupID);
    RETURN @cnt;
END
GO
```

Виконуємо:

```
DECLARE @cnt int;
EXEC @cnt = dbo.GetGroupStudentCount @GroupID = 9;
SELECT @cnt AS ReturnCode;
```

Отримаємо:

	ReturnCode
1	5

d. створення процедури оновлення даних в деякій таблиці БД;

```
-- Оновлення аудиторії проведення екзамену
CREATE OR ALTER PROCEDURE dbo.UpdateExamRoom
    @ExamID  int,
    @NewRoom varchar(10)
AS
BEGIN
    BEGIN TRY
        IF NOT EXISTS (SELECT 1 FROM dbo.Exam WHERE ID = @ExamID)
        BEGIN
            RAISERROR(N'Екзамен із таким ID не знайдено', 16, 1);
            RETURN -1;
        END
```

```
        UPDATE dbo.Exam SET ROOM_EXAM = @NewRoom WHERE ID = @ExamID;
        RETURN 0;
    END TRY

    BEGIN CATCH
        DECLARE @msg nvarchar(4000) = ERROR_MESSAGE();
        RAISERROR(@msg, 16, 1);
        RETURN -2;
    END CATCH
END
GO
```

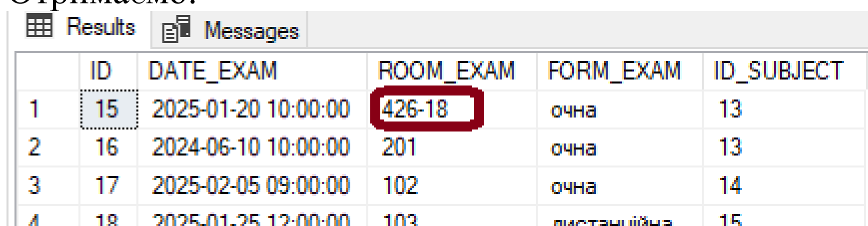
Маємо:

	ID	DATE_EXAM	ROOM_EXAM	FORM_EXAM	ID_SUBJECT
1	15	2025-01-20 10:00:00	101	очна	13
2	16	2024-06-10 10:00:00	201	очна	13
3	17	2025-02-05 09:00:00	102	очна	14
4	18	2025-01-25 12:00:00	103	дистанційна	15
5	19	2025-12-22 14:00:00	104	очна	16
6	20	2025-03-03 11:00:00	105	очна	17
7	21	2025-05-15 10:00:00	106	очна	18
8	26	2025-01-15 00:00:00	433-18	очна	13
9	27	2025-01-15 00:00:00	433-18	очна	13

Виконуємо:

```
exec dbo.UpdateExamRoom 15, '426-18'
```

Отримаємо:



	ID	DATE_EXAM	ROOM_EXAM	FORM_EXAM	ID_SUBJECT
1	15	2025-01-20 10:00:00	426-18	очна	13
2	16	2024-06-10 10:00:00	201	очна	13
3	17	2025-02-05 09:00:00	102	очна	14
4	18	2025-01-25 12:00:00	103	дистанційна	15

2) Функції:

a. створити функцію, котра повертає деяке скалярне значення;

```
-- Перетворення оцінки студента в шкалу ECTS
CREATE OR ALTER FUNCTION dbo.EctsFromGrade (@Grade int)
RETURNS nvarchar(2)
AS
BEGIN
    DECLARE @res nvarchar(2);

    IF @Grade IS NULL       SET @res = '—';
    ELSE IF @Grade >= 95    SET @res = 'A';
    ELSE IF @Grade >= 85    SET @res = 'B';
    ELSE IF @Grade >= 75    SET @res = 'C';
    ELSE IF @Grade >= 65    SET @res = 'D';
    ELSE IF @Grade >= 60    SET @res = 'E';
```

```
    ELSE IF @Grade >= 30     SET @res = 'FX';
    ELSE                     SET @res = 'F';

    RETURN @res;
END
GO
```

Виконуємо:

```
SELECT g.ID_Student, trim(s.Surname)+' '+trim(s.Naim) as Name_Student,
      g.ID_Exam, sbj.Naim as Subject_Naim,
      g.EXAM_GRADE, ECTS = dbo.EctsFromGrade(g.EXAM_GRADE)
FROM Grades g JOIN Student s on g.ID_Student = s.ID
            JOIN Exam e on g.ID_Exam = e.ID
            JOIN Subject sbj on e.ID_SUBJECT = sbj.ID;
```

Отримаємо:

Results | Messages

	ID_Student	Name_Student	ID_Exam	Subject_Naim	EXAM_GRADE	ECTS
15	29	Бондар Тарас	18	Алгоритми	84	C
16	30	Мельник Олег	15	Бази даних	80	C
17	30	Мельник Олег	19	Операційні си...	80	C
18	30	Мельник Олег	20	Комп'ютерні ...	74	D
19	31	Романюк Наталія	15	Бази даних	81	C
20	31	Романюк Наталія	19	Операційні си...	81	C
21	31	Романюк Наталія	20	Комп'ютерні ...	75	C
22	32	Сидоренко Анд...	15	Бази даних	82	C
23	32	Сидоренко Анд...	19	Операційні си...	82	C
24	32	Сидоренко Анд...	20	Комп'ютерні ...	76	C
25	33	Кравченко Олена	15	Бази даних	83	C
26	33	Кравченко Олена	19	Операційні си...	83	C
27	33	Кравченко Олена	20	Комп'ютерні ...	77	C
28	34	Гриценко Павло	17	Програмування	63	E

b. створити функцію, котра повертає таблицю наперед заданої структури.

```
-- Створити розклад певної групи на певний рік
CREATE OR ALTER FUNCTION dbo.GroupExamsForYear
(
    @GroupID int,
    @Year    int
)
RETURNS TABLE
AS
RETURN
(
    SELECT
        g.ID           AS GroupID,
        g.NAIM         AS GroupName,
        sub.ID         AS SubjectID,
        sub.Naim       AS SubjectName,
        e.ID           AS ExamID,
        e.DATE_EXAM    AS ExamDate,
```

```
        e.ROOM_EXAM    AS RoomExam,
        e.FORM_EXAM    AS FormExam
    FROM dbo.[Group] g
    JOIN dbo.GroupSchedule gs ON gs.ID_Group = g.ID
    JOIN dbo.Exam e           ON e.ID        = gs.ID_Exam
    JOIN dbo.[Subject] sub    ON sub.ID      = e.ID_SUBJECT
    WHERE g.ID = @GroupID
      AND YEAR(e.DATE_EXAM) = @Year
);
GO
```

Виконуємо:

```
SELECT * FROM dbo.GroupExamsForYear(9, 2025);
```

Отримаємо:

Results | Messages

	GroupID	GroupName	SubjectID	SubjectName	ExamID	ExamDate	RoomExam	FormExam
1	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	426-18	очна
2	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна
3	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна

3) Робота з курсорами (створити процедуру, в котрій демонструються наведені нижче дії):
   a. створення курсору;
   b. відкриття курсору;
   c. вибірка даних;
   d. робота з курсорами.

```
-- Сформувати звіт з відображенням кількості екзаменів у груп в певному році
CREATE OR ALTER PROCEDURE dbo.CursorGroupExamStats
    @Year int
AS
BEGIN
    DECLARE @GroupID int, @GroupName varchar(50);

    DECLARE @Stats TABLE
    (
        GroupID   int,
        GroupName varchar(50),
        ExamsThisYear int
    );

    DECLARE cur CURSOR FAST_FORWARD FOR
        SELECT g.ID, g.NAIM
        FROM dbo.[Group] g
        ORDER BY g.NAIM;

    OPEN cur;
    FETCH NEXT FROM cur INTO @GroupID, @GroupName;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        DECLARE @cnt int =
```

```
        (
            SELECT COUNT(*)
            FROM dbo.GroupSchedule gs
            JOIN dbo.Exam e ON e.ID = gs.ID_Exam
            WHERE gs.ID_Group = @GroupID
              AND YEAR(e.DATE_EXAM) = @Year
        );

        INSERT INTO @Stats (GroupID, GroupName, ExamsThisYear)
        VALUES (@GroupID, @GroupName, @cnt);

        FETCH NEXT FROM cur INTO @GroupID, @GroupName;
    END

    CLOSE cur;
    DEALLOCATE cur;

    SELECT * FROM @Stats ORDER BY GroupName;
END
GO
```

Виконуємо:

```
exec dbo.CursorGroupExamStats 2025
```

Отримаємо:

Results | Messages

	GroupID	GroupName	ExamsThisYear
1	10	ІП-з11	3
2	11	ІП-з12	2
3	9	ІП-з51	3

4) Робота з тригерами:

a. створити тригер, котрий буде спрацьовувати при видаленні даних;

```
-- Заборонити видаляти екзамен для групи, якщо в році, з якого видаляється
екзамен їх для неї не залишиться
CREATE OR ALTER TRIGGER dbo.GroupSchedule_BlockDeleteLastExamInYear
ON dbo.GroupSchedule AFTER DELETE
AS
BEGIN
 -- Якщо після видалення для якоїсь (Група, Рік) не лишилось жодного екзамену
→ відхиляємо операцію
    IF EXISTS (
        SELECT 1
        FROM (
            -- Пари (Група, Рік), яких торкнулося видалення
            SELECT DISTINCT d.ID_Group, YEAR(e.DATE_EXAM) AS Yr
            FROM deleted AS d
            JOIN dbo.Exam AS e ON e.ID = d.ID_Exam
        ) AS dp
        LEFT JOIN (
            -- Скільки екзаемнів залишилось для кожної (Група, Рік) після
видалення
            SELECT gs.ID_Group, YEAR(e.DATE_EXAM) AS Yr, COUNT(*) AS Cnt
```

```
            FROM dbo.GroupSchedule AS gs
            JOIN dbo.Exam AS e ON e.ID = gs.ID_Exam
            GROUP BY gs.ID_Group, YEAR(e.DATE_EXAM)
        ) AS r
          ON r.ID_Group = dp.ID_Group
         AND r.Yr       = dp.Yr
        WHERE ISNULL(r.Cnt, 0) = 0
    )
    BEGIN
            RAISERROR(N'Заборонено видаляти: для деяких груп це залишає 0 екзаемнів у відповідному році.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END
END
GO
```

Маємо:

	ID_Group	ID_Exam	COMM
1	9	15	Група ІП-з51 на БД (січень)
2	9	17	Група ІП-з51 на Програмування
3	9	18	Група ІП-з51 на Алгоритми
4	10	15	Група ІП-з11 на БД (січень)
5	10	19	Група ІП-з11 на ОС
6	10	20	Група ІП-з11 на Мережі
7	11	17	Група ІП-з12 на Програмування
8	11	21	Група ІП-з12 на Філософію

Виконуємо:

```
delete from dbo.GroupSchedule where ID_Group = 11
```

Отримаємо:

Messages

```
Msg 50000, Level 16, State 1, Procedure GroupSchedule_BlockDeleteLastExamInYear, Line 27 [Batch Start Line 35]
Заборонено видаляти: для деяких груп це залишає 0 екзаемнів у відповідному році.
Msg 3609, Level 16, State 1, Line 36
The transaction ended in the trigger. The batch has been aborted.
```

b. створити тригер, котрий буде спрацьовувати при модифікації даних;

```
-- Заборонити модифікацію дати екзамену для групи, якщо між екзаменами для цієї групи стане менше 3 днів
CREATE OR ALTER TRIGGER dbo.trg_Exam_BlockTooCloseReschedule
ON dbo.Exam AFTER UPDATE
AS
BEGIN
    -- Виконуємо тільки якщо змінювали DATE_EXAM
    IF NOT UPDATE(DATE_EXAM)
        RETURN;

    IF EXISTS (
```

```
        SELECT 1
        FROM inserted i
        JOIN deleted  d ON d.ID = i.ID
        WHERE ISNULL(i.DATE_EXAM, '19000101') <> ISNULL(d.DATE_EXAM,
'19000101')
          AND EXISTS (
                SELECT 1
                FROM dbo.GroupSchedule gs_i
                JOIN dbo.GroupSchedule gs_other
                     ON gs_other.ID_Group = gs_i.ID_Group
                JOIN dbo.Exam e_other
                     ON e_other.ID = gs_other.ID_Exam
                WHERE gs_i.ID_Exam = i.ID
                  AND e_other.ID <> i.ID
                  AND ABS(DATEDIFF(HOUR, e_other.DATE_EXAM, i.DATE_EXAM)) <
72
            )
    )
    BEGIN
        RAISERROR('Заборонено змінювати дату: між екзаменами певної групи
стане менше 3 діб.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END
END
GO
```

Маємо:

```
SELECT * FROM dbo.GroupExamsForYear(9, 2025);
```

Results | Messages

	GroupID	GroupName	SubjectID	SubjectName	ExamID	ExamDate	RoomExam	FormExam
1	9	ІП-з51	13	Бази даних	15	2025-01-20 10:00:00	426-18	очна
2	9	ІП-з51	14	Програмування	17	2025-02-05 09:00:00	102	очна
3	9	ІП-з51	15	Алгоритми	18	2025-01-25 12:00:00	103	дистанційна

Виконуємо:

```
update dbo.Exam set DATE_EXAM = '20250121' where id = 18
```

Отримаємо:

Messages

```
Msg 50000, Level 16, State 1, Procedure trg_Exam_BlockTooCloseReschedule, Line 28 [Batch Start Line 34]
Заборонено змінювати дату: між екзаменами певної групи стане менше 3 діб.
Msg 3609, Level 16, State 1, Line 35
The transaction ended in the trigger. The batch has been aborted.
```

c. створити тригер, котрий буде спрацьовувати при додаванні даних.

```
-- Заборонити додавати студентів з такими самим ім'ям, прізвищем та датою
народження
CREATE OR ALTER TRIGGER dbo.trg_Student_BlockDuplicates
ON dbo.Student
AFTER INSERT
```

```
AS
BEGIN
    -- 1) Перевірка дубля щодо вже наявних у БД
    IF EXISTS (
        SELECT 1
        FROM inserted i
        JOIN dbo.Student s
          ON s.Naim = i.Naim
         AND s.Surname = i.Surname
         AND s.DATE_BIRTH IS NOT NULL
         AND i.DATE_BIRTH IS NOT NULL
         AND s.DATE_BIRTH = i.DATE_BIRTH
    )
    BEGIN
        RAISERROR('Дубль студента: ім’я, прізвище та дата народження вже
існують.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END

    -- 2) Перевірка дубля всередині самого INSERT-пакета (масова вставка)
    IF EXISTS (
        SELECT i.Naim, i.Surname, i.DATE_BIRTH
        FROM inserted i
        WHERE i.DATE_BIRTH IS NOT NULL
        GROUP BY i.Naim, i.Surname, i.DATE_BIRTH
        HAVING COUNT(*) > 1
    )
    BEGIN
        RAISERROR(N'Дубль у пакеті вставки: є повтори ім’я+прізвище+дата
народження.', 16, 1);
        ROLLBACK TRANSACTION;
        RETURN;
    END
END
GO
```

Маємо:

Results | Messages

	ID	Naim	Surname	DATE_BIRTH	ID_Group	AgeYears
1	25	Олександр	Ліщук	2006-04-11 00:00:00.000	9	19
2	26	Марія	Коваленко	2006-11-23 00:00:00.000	9	19
3	27	Дмитро	Іваненко	2005-07-05 00:00:00.000	9	20
4	28	Ірина	Шевченко	2006-01-30 00:00:00.000	9	19
5	29	Тарас	Бондар	2005-12-12 00:00:00.000	9	20
6	30	Олег	Мельник	2007-03-19 00:00:00.000	10	18
7	31	Наталія	Романюк	2006-09-08 00:00:00.000	10	19
8	32	Андрій	Сидоренко	2005-05-17 00:00:00.000	10	20
9	33	Олена	Кравченко	2006-02-25 00:00:00.000	10	19
10	34	Павло	Гриценко	2004-10-03 00:00:00.000	11	21
11	35	Юлія	Сергієнко	2005-06-28 00:00:00.000	11	20

Виконуємо:

```
insert into Student (Naim, Surname, DATE_BIRTH) values
('Ірина','Шевченко','20060130')
```

Отримаємо:

```
Msg 50000, Level 16, State 1, Procedure trg_Student_BlockDuplicates, Line 19 [Batch Start Line 41]
Дубль студента: ім’я, прізвище та дата народження вже існують.
Msg 3609, Level 16, State 1, Line 42
The transaction ended in the trigger. The batch has been aborted.
```

ЛІТЕРАТУРА

1) Chen P. P.-S. The Entity-Relationship Model — Toward a Unified View of Data. ACM Transactions on Database Systems. 1976. Vol. 1, no. 1. P. 9–36.

2) Oracle. Data Modeler Concepts and Usage. URL: https://docs.oracle.com/en/database/oracle/sql-developer-data-modeler/19.1/dmdug/data-modeler-concepts-usage.html (дата звернення: 21.02.2026).

3) IBM Docs. Overview of the entity-relationship data model. URL: https://www.ibm.com/docs/en/SSGU8G_12.1.0/com.ibm.ddi.doc/ids_ddi_163.htm (дата звернення: 28.02.2026).

4) IBM Think. What is an Entity Relationship Diagram? URL: https://www.ibm.com/think/topics/entity-relationship-diagram (дата звернення: 01.03.2026).

5) Microsoft Support. Create a database model, also known as Entity Relationship Diagram, in Visio. URL: https://support.microsoft.com/en-us/office/create-a-database-model-also-known-as-entity-relationship-diagram-in-visio-7042e719-384a-4b41-b29c-d1b35719fc93 (дата звернення: 05.03.2026).

6) Elmasri R., Navathe S. B. Fundamentals of Database Systems. 7th ed. Boston : Pearson, 2016.

7) Пасічник В. В., Резниченко В. А. Організація баз даних та знань. Київ : Видавнича група BHV, 2006. 384 с.

8) Garcia-Molina H., Ullman J. D., Widom J. Database Systems: The Complete Book. 2nd ed. Upper Saddle River : Pearson Prentice Hall, 2009.

9) Ullman J. D. Principles of Database Systems. Rockville : Computer Science Press, 1982.

10) Forta B. SQL in 10 Minutes, Sams Teach Yourself. 5th ed. Indianapolis : Sams Publishing, 2019.

11) Берко А. Ю., Верес О. М. Застосування баз даних : навч. посіб. Львів : Ліга-Прес, 2007. 208 с.

ДОДАТОК А

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
(повна назва інституту/факультету)

КАФЕДРА інформатики та програмної інженерії
(повна назва кафедри)

Звіт

з лабораторної роботи №_ з дисципліни «Бази даних»

«Назва лабораторної»

Варіант___

Виконав студент ___________________________
(шифр групи, прізвище, ім'я, по батькові)

Перевірив викладач _________________________
(прізвище, ім'я, по батькові)

Київ 202_ рік

ДОДАТОК Б

Приклади індивідуальних варіантів завдань до лабораторної роботи.

1) Програмне забезпечення військової частини.

Необхідно спроєктувати базу даних для обліку організаційної структури військового округу, його особового складу, місць дислокації, техніки та озброєння. Військові частини округу розміщуються у різних місцях дислокації, при цьому в одному місці може перебувати декілька військових частин. Кожна військова частина має внутрішню структуру: вона складається з рот, роти — із взводів, а взводи — з відділень. Водночас військові частини можуть об'єднуватися у дивізії, корпуси або бригади, які, у свою чергу, входять до складу армій.

У базі даних необхідно передбачити збереження відомостей про військовослужбовців різних категорій. Особовий склад військового округу поділяється на офіцерський склад, до якого належать генерали, полковники, підполковники, майори, капітани, лейтенанти, а також на рядовий і сержантський склад, до якого належать старшини, сержанти, прапорщики та рядові. Для окремих категорій військовослужбовців можуть зберігатися додаткові характеристики, властиві лише цій категорії. Наприклад, для генералів можуть фіксуватися дата закінчення академії, дата присвоєння генеральського звання та інші специфічні відомості.

Кожний підрозділ повинен мати командира. При цьому військовослужбовці офіцерського складу можуть командувати будь-яким із зазначених типів підрозділів, тоді як військовослужбовці рядового і сержантського складу можуть командувати лише взводами або відділеннями. Також необхідно врахувати, що кожний військовослужбовець може мати одну або декілька військових спеціальностей.

Окремо потрібно забезпечити облік бойової та транспортної техніки, а також озброєння, що перебувають у військових частинах. До техніки можуть належати бойові машини піхоти, тягачі, автотранспорт тощо, а до озброєння — карабіни, автоматична зброя, артилерійські системи, ракетне озброєння та інші види. Кожна категорія техніки або озброєння може мати власний набір специфічних атрибутів, а в межах кожної категорії може існувати декілька конкретних видів.

Розроблена база даних повинна забезпечувати можливість отримання інформації про військові частини округу, їх структуру, місця дислокації, особовий склад за категоріями, командирів підрозділів, військові спеціальності, а також наявну техніку та озброєння.

2) Програмне забезпечення готелю.

Необхідно спроєктувати базу даних для програмного забезпечення готелю, основним призначенням якого є облік фінансової сторони його діяльності та контроль використання номерного фонду. Готель надає клієнтам номери на визначений термін. Кожний номер характеризується місткістю, рівнем комфортності, наприклад люкс, напівлюкс, стандартний номер тощо, а також ціною проживання.

У базі даних необхідно зберігати відомості про клієнтів готелю. Для кожного клієнта фіксуються прізвище, ім'я, по батькові, серія та номер паспорта або іншого документа, що посвідчує особу, а також додаткова інформація, необхідна для обслуговування. Поселення клієнта в номер виконується за умови наявності вільних місць відповідно до параметрів, зазначених клієнтом. Під час поселення фіксується дата заселення, а під час виїзду — дата звільнення номера.

База даних повинна підтримувати не лише облік фактичного проживання клієнтів у номерах, а й можливість попереднього бронювання номерів. Для бронювання необхідно враховувати бажані дати проживання, параметри номера, а також доступність номерів у зазначений період.

Крім того, у системі слід передбачити механізм нарахування знижок. Знижки можуть надаватися постійним клієнтам, а також окремим категоріям клієнтів. Необхідно врахувати, що декілька знижок можуть підсумовуватися відповідно до правил, визначених адміністрацією готелю.

Розроблена база даних повинна забезпечувати можливість отримання інформації про номери, їх зайнятість, бронювання, клієнтів, історію проживання, нараховані знижки та фінансові показники діяльності готелю.

3) Формування індивідуальних планів студентів.

Необхідно спроєктувати базу даних для формування та обліку індивідуальних навчальних планів студентів закладу вищої освіти. Кожний студент повинен мати індивідуальний навчальний план на кожний навчальний рік. Такий план формується на початку навчального року та визначає перелік дисциплін, які студент має опанувати протягом відповідного року навчання.

Індивідуальний навчальний план включає обов'язкові та вибіркові дисципліни. Обов'язкові дисципліни є однаковими для всіх студентів певної освітньої програми на відповідному році навчання. Вибіркові дисципліни студент обирає самостійно відповідно до правил освітньої програми та встановлених обмежень.

У базі даних необхідно зберігати відомості про студентів, зокрема прізвище, ім'я, по батькові, адресу, телефон та інші стандартні анкетні дані. Також потрібно врахувати

структуру закладу вищої освіти: певна освітня програма в межах спеціальності закріплюється за відповідною кафедрою факультету.

Для кожної дисципліни, що викладається в межах освітньої програми, необхідно зберігати кількість годин, кількість кредитів, види навчальних занять, зокрема лекції, практичні заняття, лабораторні роботи, кафедру, яка забезпечує викладання дисципліни, а також вид підсумкової атестації — залік або екзамен.

Потрібно також врахувати, що окремі дисципліни можуть тривати більше ніж один семестр. У такому випадку дисципліна поділяється на кредитні модулі. Для кожного кредитного модуля в межах конкретного семестру необхідно визначати кількість кредитів, обсяг лекційних, практичних і лабораторних занять у годинах.

Розроблена база даних повинна забезпечувати можливість формування індивідуальних навчальних планів студентів, обліку обов'язкових і вибіркових дисциплін, аналізу навчального навантаження за семестрами, освітніми програмами, дисциплінами та кредитними модулями.

4) Розподіл учбового навантаження.

Необхідно спроєктувати базу даних для обліку та розподілу навчального навантаження викладачів закладу вищої освіти. Щороку в закладі формується навчальне навантаження кафедр і викладачів. Кожний викладач входить до штатного розкладу певної кафедри, однак окремі викладачі можуть одночасно входити до штатних розкладів декількох кафедр.

У базі даних необхідно зберігати відомості про викладачів. Такі відомості мають включати стандартні анкетні дані, а також інформацію про вчений ступінь, займану посаду та стаж роботи. Необхідно також зберігати інформацію про кафедри, до штатного розкладу яких належать викладачі.

Викладачі кафедри повинні забезпечувати проведення занять з певних дисциплін. При цьому дисципліна не обов'язково має належати тій кафедрі, у штаті якої перебуває викладач. Для кожної дисципліни встановлюється певна кількість годин, а під час формування навантаження всі види навчальної діяльності за дисципліною поділяються на лекції, практичні заняття, лабораторні роботи, консультації, заліки та екзамени. Для кожного виду навчальної діяльності має бути визначена відповідна кількість годин.

У результаті розподілу навантаження система повинна забезпечувати можливість отримання інформації у вигляді: який викладач проводить заняття з якої дисципліни, для якої академічної групи та в якому обсязі. Також необхідно врахувати, що дані про навчальне навантаження мають зберігатися за декілька навчальних років, що дає змогу аналізувати історію навантаження викладачів і кафедр.

Розроблена база даних повинна забезпечувати облік викладачів, кафедр, дисциплін, академічних груп, видів навчальної роботи, навчальних років, а також формування звітів щодо розподілу навантаження за викладачами, кафедрами, дисциплінами та групами.

5) Облік донорів крові.

Необхідно спроєктувати базу даних для програмного забезпечення, призначеного для обліку донорів крові. У системі мають зберігатися відомості про донорів, зокрема прізвище, ім'я, по батькові, група крові, резус-фактор, адреса, телефон, дата народження та стать. Крім основних анкетних даних, необхідно вести облік медичних обстежень донорів, дат здачі крові, обсягу кожної здачі та загального обсягу крові, зданої донором за весь період спостереження. База даних повинна підтримувати історичність таких відомостей, оскільки один донор може проходити багато медичних обстежень і багаторазово здавати кров.

Система повинна забезпечувати можливість відбору донорів за групою крові та резус-фактором, а також формування різноманітних звітів. Зокрема, необхідно передбачити групування інформації за групами крові, регіонами, обсягами зданої крові, періодами здачі та іншими показниками.

Окремо необхідно врахувати медичні обмеження щодо донорства. За наявності певних захворювань особа не може бути донором, навіть якщо раніше вона мала відповідний статус. Тому база даних повинна зберігати інформацію про захворювання або протипоказання, а також дозволяти змінювати статус донора залежно від результатів медичного обстеження.

Розроблена база даних повинна забезпечувати облік донорів, їх медичних обстежень, випадків здачі крові, протипоказань, статусу донорства, а також формування звітності для пошуку потенційних донорів і аналізу донорської активності.